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:

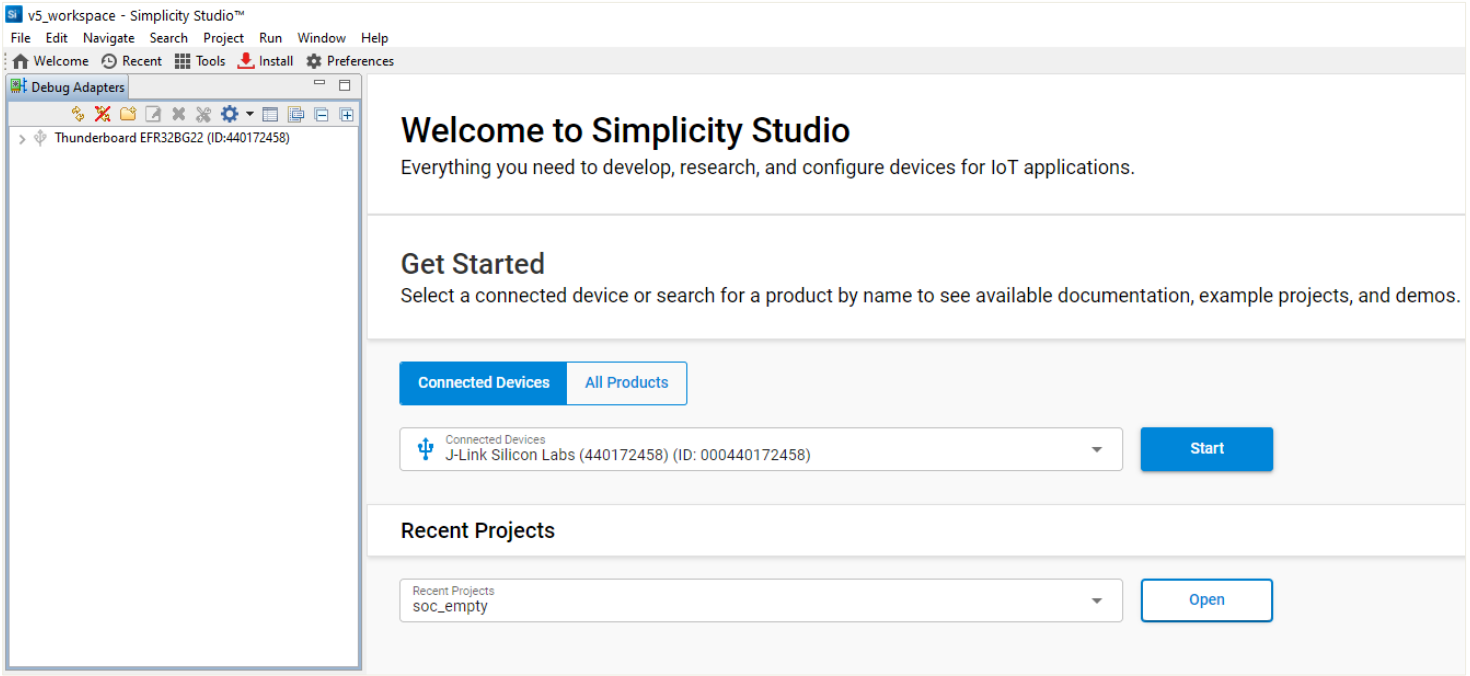
- Complete complementary labs:
  - Lab 1 – Out of the Box Beaconing
- EFR32BG22 Thunderboard (SLTB010A)
- Micro-USB to USB Type-A cable
- Simplicity Studio 5
  - https://www.silabs.com/products/development-tools/software/simplicity-studio
  - Gecko SDK Suite 3.0.2 or later
  - Bluetooth SDK 3.0.2 or later
- EFR Connect Mobile App

2 Creating a Low-Power Beacon

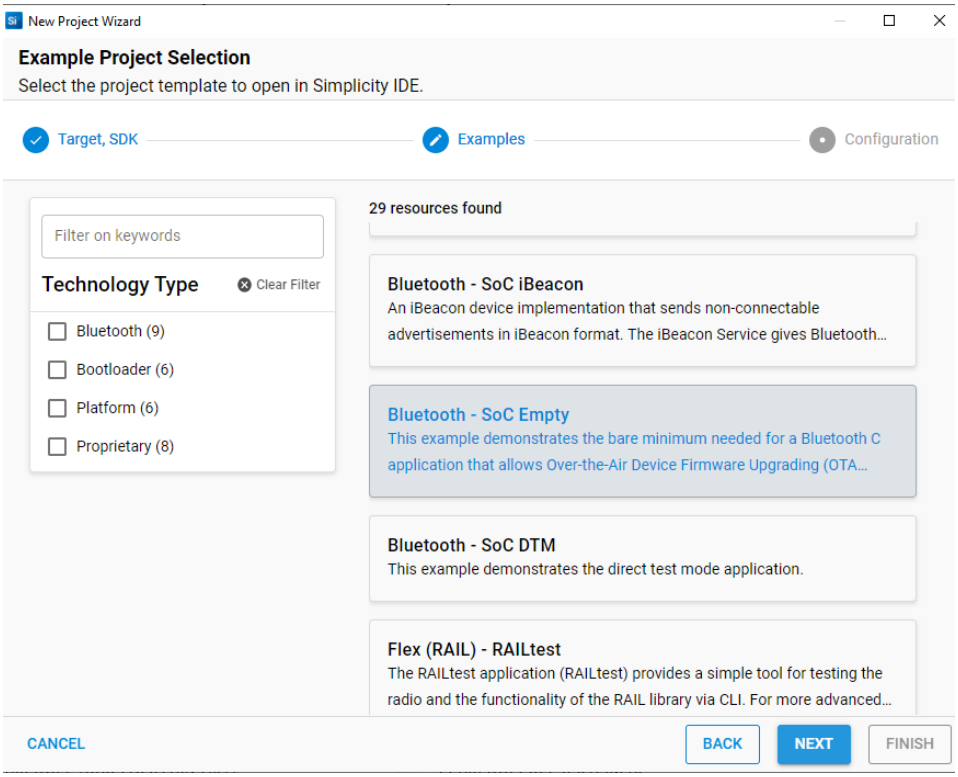
2.1 Creating the Project

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.

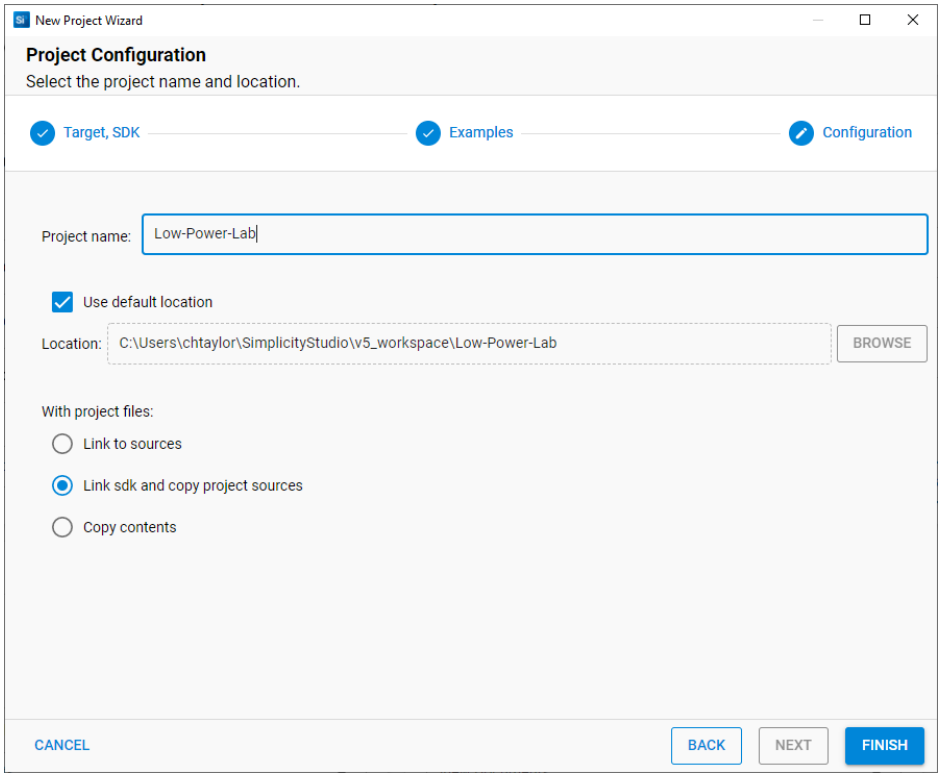
1. Launch Simplicity Studio
2. Connect the Thunberboard BG22 to your PC using a micro-USB cable
3. Once the device is connected to your PC, you should see it listed in the **Debug Adapters** window in Simplicity Studio
4. Select the device in the **Launcher** window and click **Start**



5. Verify that the Preferred SDK to Gecko SDK Suite v3.0.0 or later
6. Click the **Create New Project** button in the upper-right corner of the Launcher window
7. In the New Project Wizard window, select **Bluetooth – SoC Empty** and click **Next**



8. Set the project name to **Low-Power-Lab** and click **Finish**. Once the project is created, the Simplicity IDE is launched



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 default 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.

Here are the steps required to make the changes above:

1. Open **app.c** in from the project folder in the **Project Explorer**

2. Beginning in **app.c**, navigate to Bluetooth stack event handler and set the TX output power to 0 dBm using the following code beginning at **line 97**

```
// Set TX Power to 0 dBm
sl_bt_system_set_tx_power(0,0,0,0);
```

3. Inside the system boot event, configure the device to advertise on a single advertising channel by modifying the advertising channel map. Add the following on or near **line 100**

```
// Set advertisements to channel 37 only
sl_bt_advertiser_set_channel_map(0, 1);
```

4. Modify the advertising payload such that it advertises the device name "**BG22**" by adding the following code. 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'};
sl_bt_advertiser_set_data(0, 0, sizeof(name), name);
```

5. Increase the advertising interval to 1 second by *modifying* the advertising interval max and min values in **bold** below. The advertising interval setting can be found near line 110.

```
// Set advertising interval to 1000ms.
sc = sl_bt_advertiser_set_timing(
  advertising_set_handle,
  1600, // min. adv. interval (milliseconds * 1.6)
  1600, // max. adv. interval (milliseconds * 1.6)
  0,   // adv. duration
  0);  // max. num. adv. events
```

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

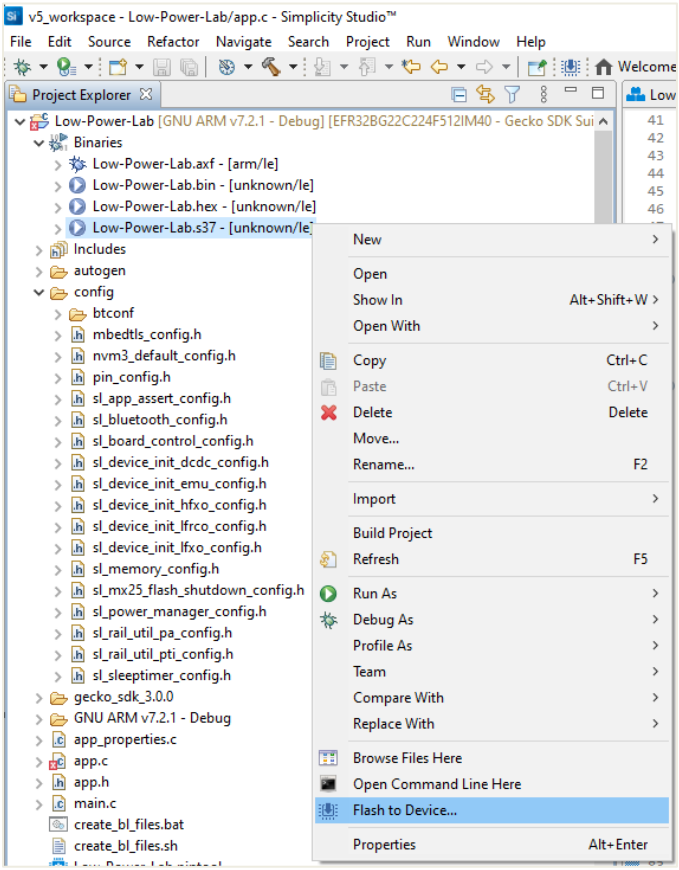
```
// Start advertising and disable connections.
sc = sl_bt_advertiser_start(
  advertising_set_handle,
  advertiser_user_data,
  advertiser_non_connectable);
```

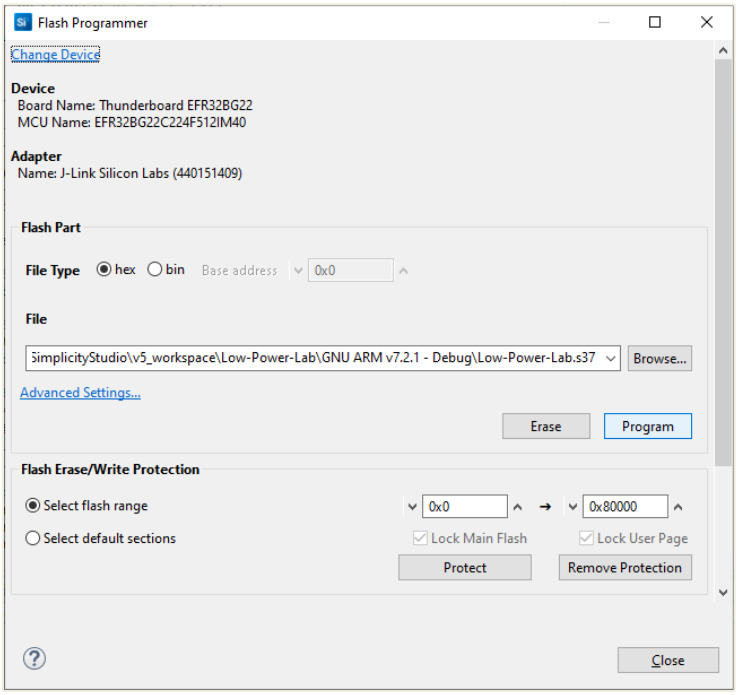
```
92        // Create an advertising set.
93        sc = sl_bt_advertiser_create_set(&advertising_set_handle);
94        sl_app_assert(sc == SL_STATUS_OK,
95                      "[E: 0x%04x] Failed to create advertising set\n",
96                      (int)sc);
97        // Set TX Power to 0 dBm
98        sl_bt_system_set_max_tx_power(0, 0);
99
100       // Set advertisements to channel 37 only
101       sl_bt_advertiser_set_channel_map(0, 1);
102
103       // Set the device name to 'BG22'
104       uint8_t name[] = {5,9,'B','G','2','2'};
105       sl_bt_advertiser_set_data(0, 0, sizeof(name), name);
106
107       // Set advertising interval to 1000ms.
108       sc = sl_bt_advertiser_set_timing(
109         advertising_set_handle,
110         1600, // min. adv. interval (milliseconds * 1.6)
111         1600, // max. adv. interval (milliseconds * 1.6)
112         0,   // adv. duration
113         0);  // max. num. adv. events
114       sl_app_assert(sc == SL_STATUS_OK,
115                     "[E: 0x%04x] Failed to set advertising timing\n",
116                     (int)sc);
117       // Start general advertising and enable connections.
118       sc = sl_bt_advertiser_start(
119         advertising_set_handle,
120         advertiser_user_data,
121         advertiser_non_connectable);
122       sl_app_assert(sc == SL_STATUS_OK,
123                     "[E: 0x%04x] Failed to start advertising\n",
124                     (int)sc);
125       break;
```

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.

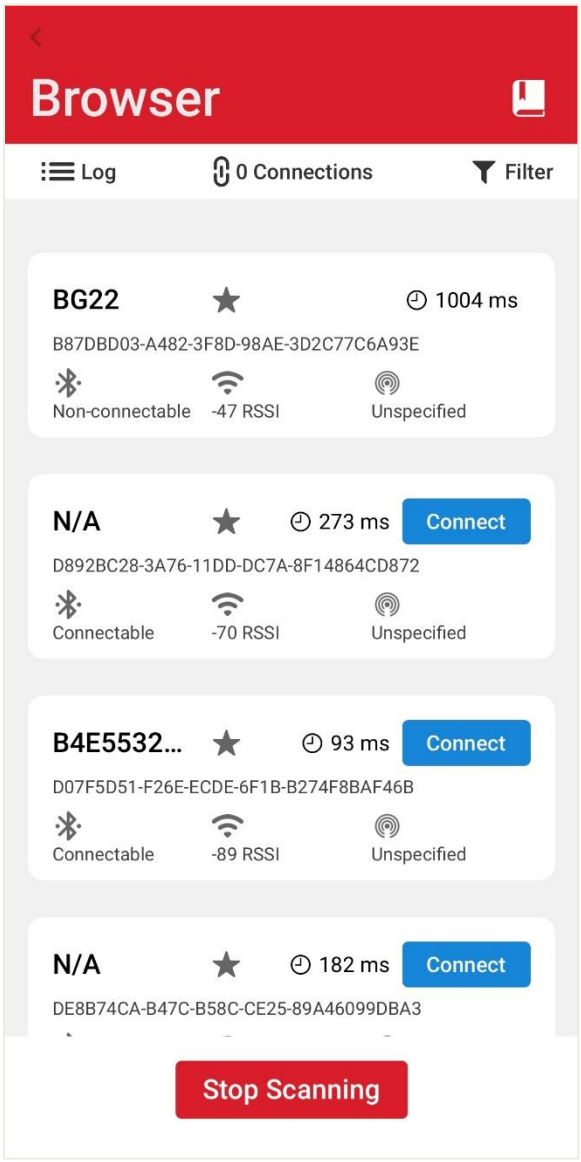
1. With the project selected in the **Project Explorer** window, click the **Build** icon ( ) in the toolbar or right-click on the project and select **Build Project**
2. Once the project has built successfully, open the **Binaries** folder
3. Right-click the **Low-Power-Lab.s37** file and select **Flash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





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 with the device name **BG22**, and the device should be marked as **Non-connectable**.



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 130 µA. When measuring the current consumption of the low-power beacon created above, the average current consumption is measured to be approximately 3.25 µA.

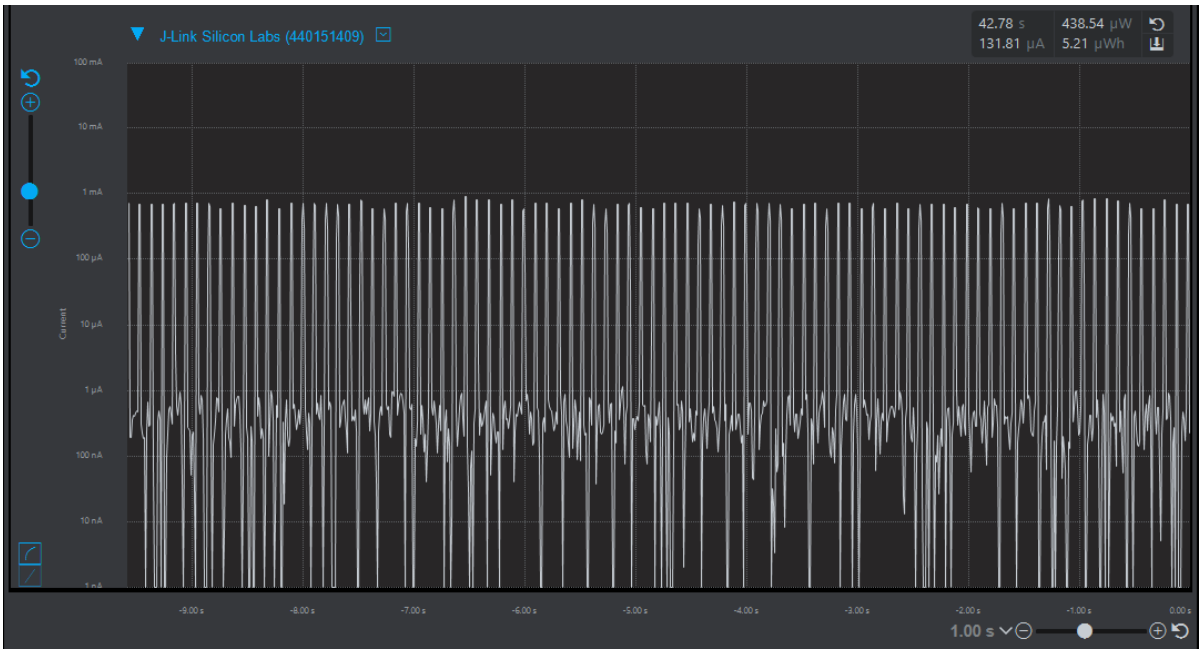


Figure 1: Default SOC-Empty Project

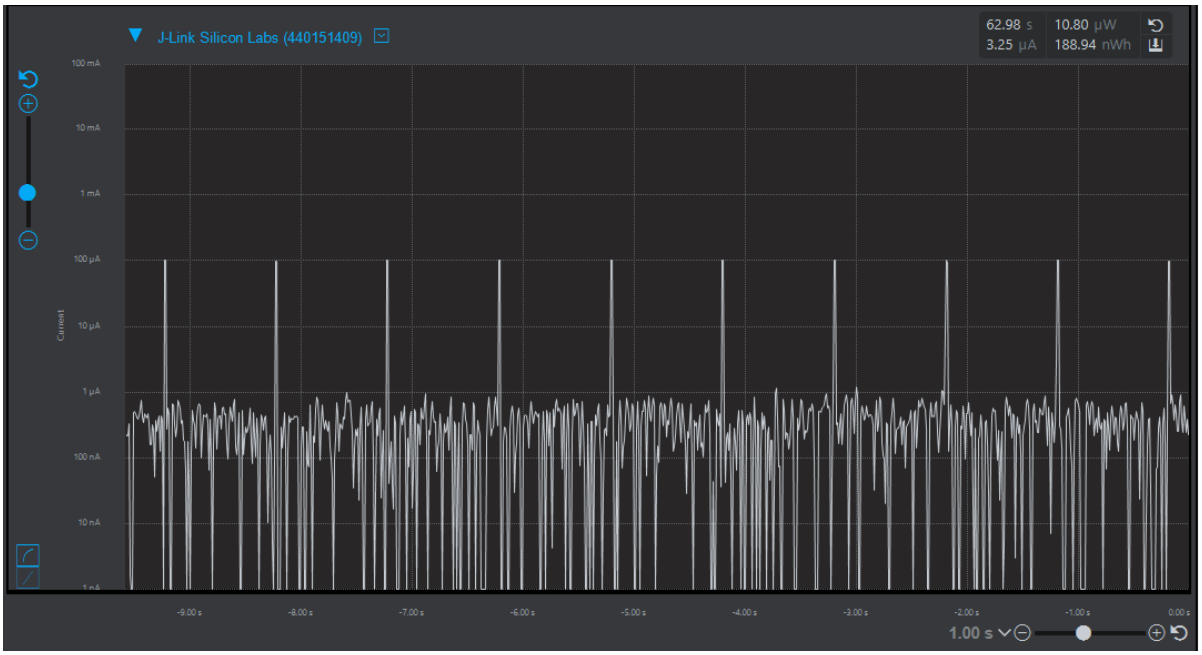


Figure 2: Modified Low-Power Beacon Project

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 advertisements to all 3 channels
sl_bt_advertiser_set_channel_map(0, 7);
```

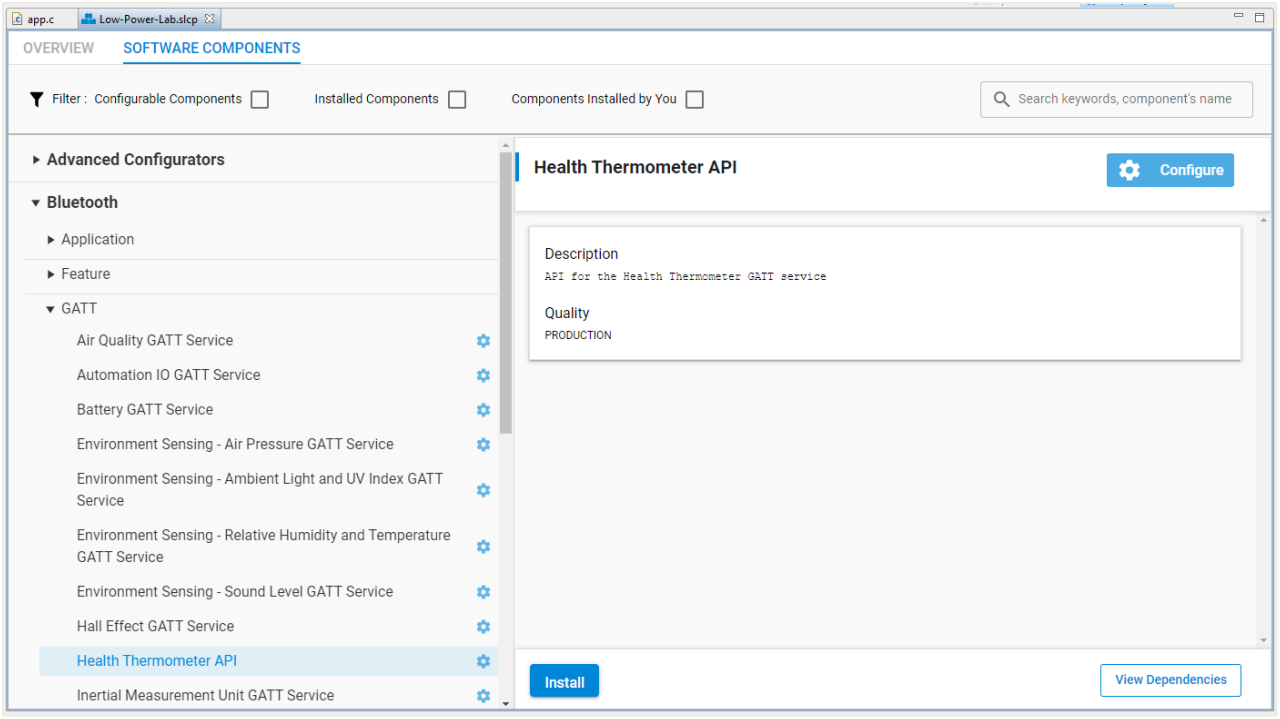
2. Configure the device to advertise as discoverable and connectable by modifying the start advertising command

```
// Start general advertising and enable connections.
sc = sl_bt_advertiser_start(
  advertising_set_handle,
  advertiser_general_discoverable,
  advertiser_connectable_scannable);
```

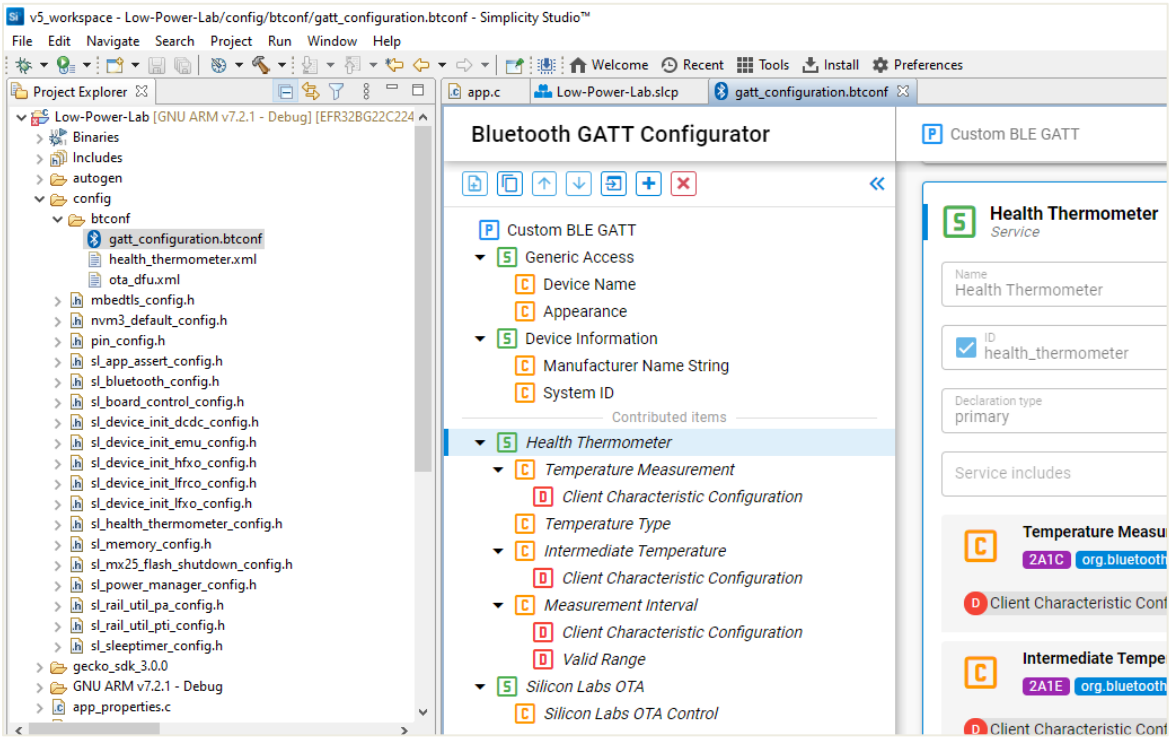
3.2 Adding the Health Thermometer Characterisitc

In this section, we will add a health thermometer 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.slcp** file
2. Open the **Software Components** tab and select the **Health Thermometer API** in Bluetooth > GATT
3. Click **Install** to add the component to the project



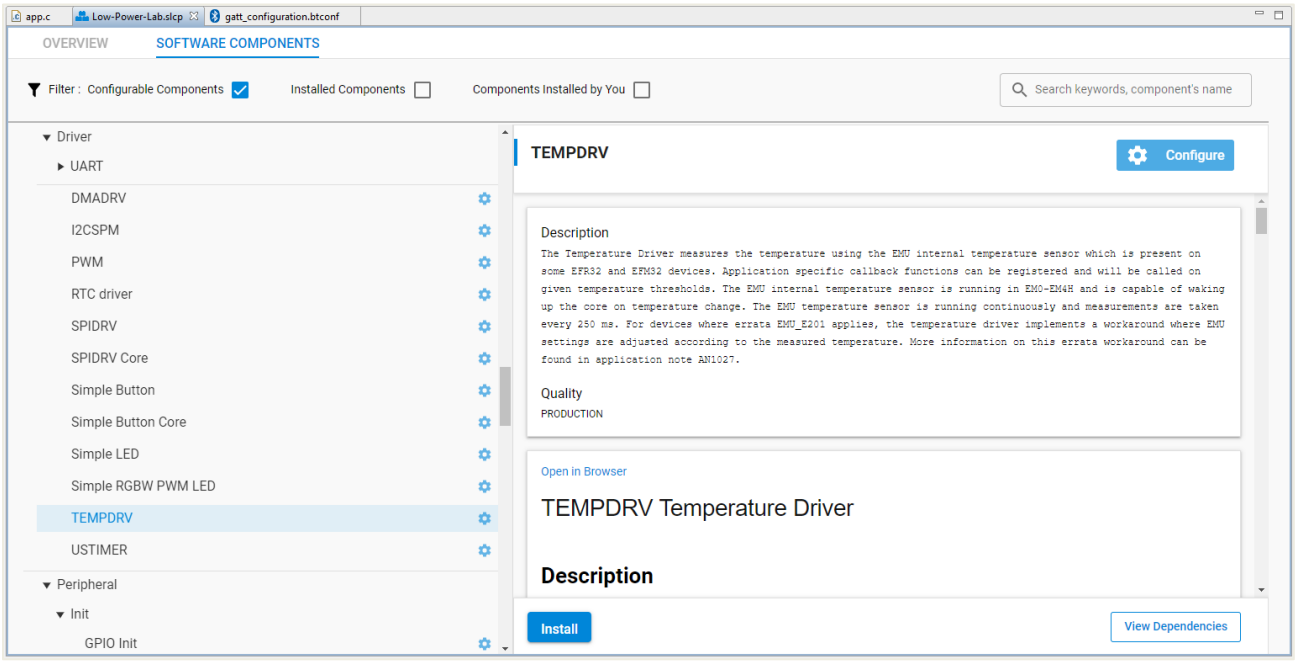
Installing the Health Thermometer API component automatically adds relevant source files to the project and adds the health thermometer service to the Bluetooth GATT. To verify, we can open the Bluetooth GATT Configurator file in the Project Explorer located at: **config > btconf > gatt_configuration.btconf**



3.3 Configuring the Internal Temperature Sensor

With the health thermometer service added to the project, we will now set up the device to read the internal temperature once per second during an active Bluetooth connection and write that temperature value to the temperature measurement characteristic.

1. Open the **Low-Power-Lab.slcp** file
2. Open the **Software Components** tab and install the **TEMPDRV** in Platform > Driver



3. Open **app.c** and locate the include statements near the top of the file. Place the following code beginning at **line 22**

```
#include "sl_health_thermometer.h"
#include "tempdrv.h"
static uint8_t conn_handle = 0xff;
void temperatureMeasure()
{
  int32_t temperature;
  temperature = TEMPDRV_GetTemp()*1000;

  // Send temperature measurement indication to connected client.
  sl_bt_ht_temperature_measurement_indicate(conn_handle,temperature, false);
}
```

4. Navigate to the **connection_opened** event handler (near line 140) and add the following to set the connection handle and initialize a software timer

```
    case sl_bt_evt_connection_opened_id:
      // Set the connection handle
      conn_handle = evt->data.evt_connection_opened.connection;
      // Create a software timer with an interval of 1 second
      sl_bt_system_set_soft_timer(32768, 0, 0);
      break;
```

5. Immediately following the connection_opened event handler, add a handler for the software timer and set it to trigger the temperatureMeasure() function created in step 3 above

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

3.4 Power Optimizing Connection Parameters and Flashing to the Device

The connection interval is set by the central device and in many cases is quite short (~25ms). Because the temperature sensor is configured to measure and update only once every second, we can configure the peripheral device to request a longer connection interval and add peripheral latency to lessen the wake-up frequency and thus the on-time of the radio.

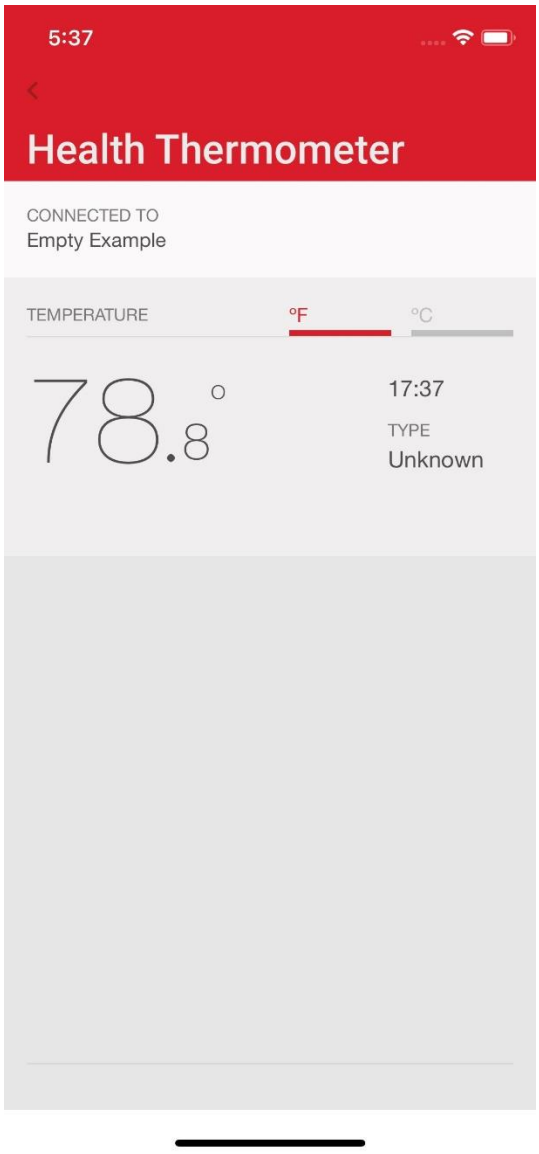
1. In the event **connection_opened** event, add the code below in **bold** to set the timing parameters to increase the connection interval to 200 ms and add a latency of 4 intervals:

```
    case sl_bt_evt_connection_opened_id:
      // Set the connection handle
      conn_handle = evt->data.evt_connection_opened.connection;

      // Create a software timer with an interval of 1 second
      sl_bt_system_set_soft_timer(32768, 0, 0);

      // Request a change in the connection parameters of a Bluetooth connection
      // 200ms connection interval, latency of 4 intervals
    sl_bt_connection_set_parameters(evt->data.evt_connection_opened.connection, 160, 160, 4, 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**

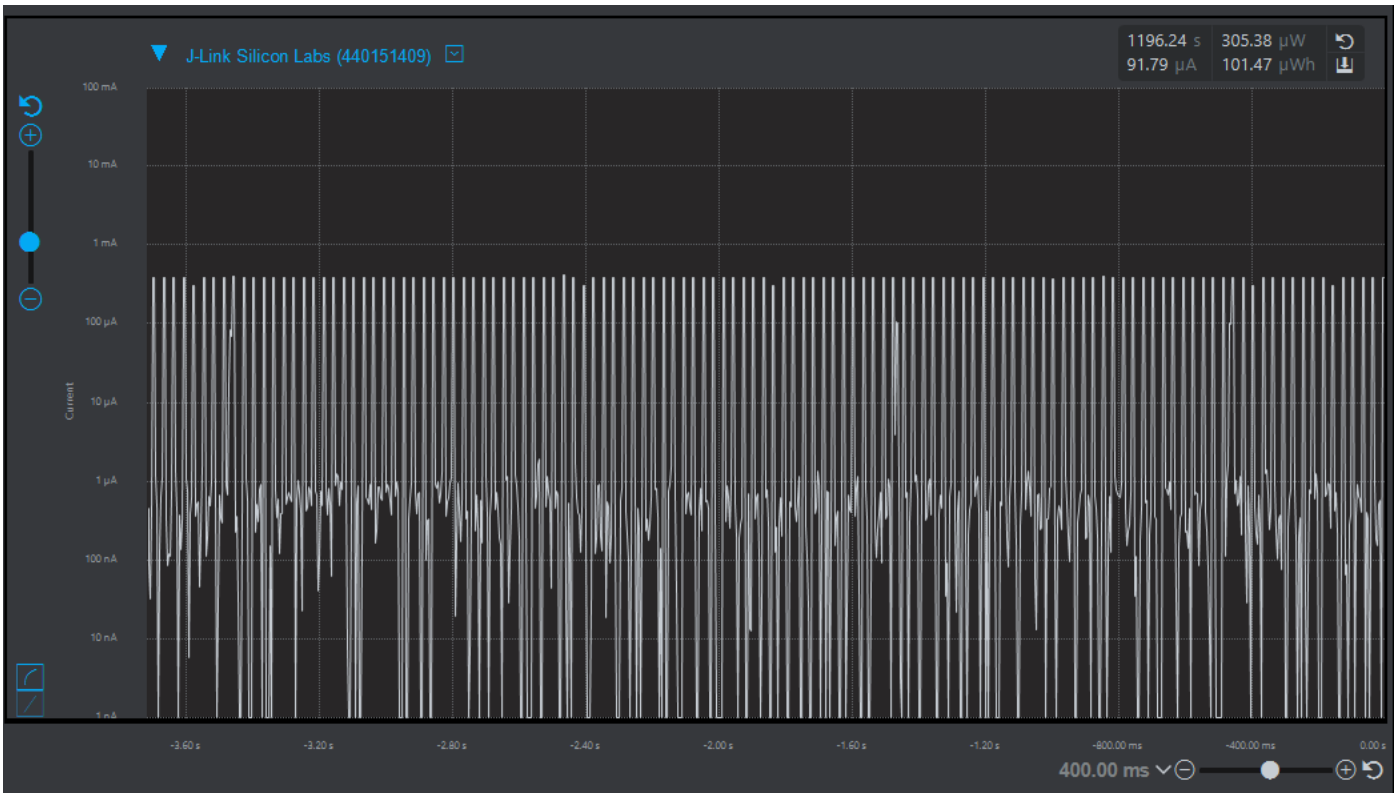
3.5 View the Temperature Readings in EFR Connect

1. Open the **EFR Connect** app on your mobile device
2. Select the **Demo** tab and open the **Health Thermometer** demo
3. Locate your device and **Connect** (the default device name is ‘Empty Example)

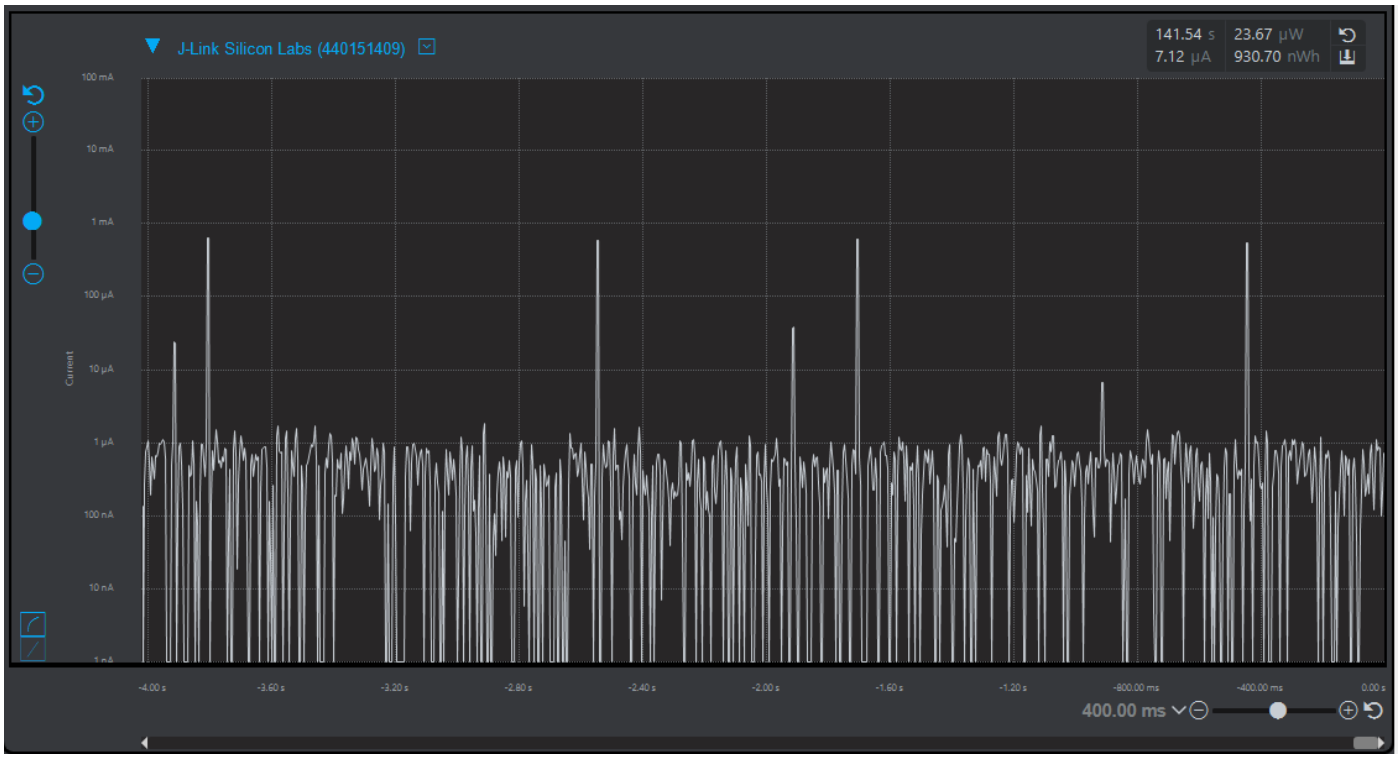


3.6 Energy Profiler (Hardware Not Included)

With the thermometer characteristic configured and using the default connection parameters, the averaged current consumption of the application is measured to be about 92 µA.



After changing the connection interval and adding slave latency, the averaged current consumption is measured to be < 7.2 µA.


J-Link Silicon Labs (440151409)
141.54 s
23.67 µW
7.12 µA
930.70 nWh
100 mA
10 mA
1 mA
100 µA
10 µA
1 µA
100 nA
10 nA
Current
-4.00 s
-3.60 s
-3.20 s
-2.80 s
-2.40 s
-2.00 s
-1.60 s
-1.20 s
-800.00 ms
-400.00 ms
0.00 s
400.00 ms

4 Appendix

4.1 Part 1 – Low Power Beacon App.c

```
/***************************************************************************//**
 * @file
 * @brief Core application logic.
 *******************************************************************************
 * # License
 * <b>Copyright 2020 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.
 *
 ******************************************************************************/
#include "em_common.h"
#include "sl_app_assert.h"
#include "sl_bluetooth.h"
#include "gatt_db.h"
#include "app.h"

// The advertising set handle allocated from Bluetooth stack.
static uint8_t advertising_set_handle = 0xff;

/**************************************************************************//**
 * Application Init.
 *****************************************************************************/
SL_WEAK void app_init(void)
{
  /////////////////////////////////////////////////////////////////////////////
  // Put your additional application init code here!                         //
  // This is called once during start-up.                                    //
  /////////////////////////////////////////////////////////////////////////////
}

/**************************************************************************//**
 * Application Process Action.
 *****************************************************************************/
SL_WEAK void app_process_action(void)
{
  /////////////////////////////////////////////////////////////////////////////
  // Put your additional application code here!                              //
  // This is called infinitely.                                              //
  // Do not call blocking functions from here!                               //
  /////////////////////////////////////////////////////////////////////////////
}

/**************************************************************************//**
 * Bluetooth stack event handler.
 * This overrides the dummy weak implementation.
 *
 * @param[in] evt Event coming from the Bluetooth stack.
 *****************************************************************************/
void sl_bt_on_event(sl_bt_msg_t *evt)
{
  sl_status_t sc;
  bd_addr address;
  uint8_t address_type;
  uint8_t system_id[8];

  switch (SL_BT_MSG_ID(evt->header)) {
    // -------------------------------
    // This event indicates the device has started and the radio is ready.
    // Do not call any stack command before receiving this boot event!
```

```
case sl_bt_evt_system_boot_id:

  // Extract unique ID from BT Address.
  sc = sl_bt_system_get_identity_address(&address, &address_type);
  sl_app_assert(sc == SL_STATUS_OK,
                "[E: 0x%04x] Failed to get Bluetooth address\n",
                (int)sc);

  // Pad and reverse unique ID to get System ID.
  system_id[0] = address.addr[5];
  system_id[1] = address.addr[4];
  system_id[2] = address.addr[3];
  system_id[3] = 0xFF;
  system_id[4] = 0xFE;
  system_id[5] = address.addr[2];
  system_id[6] = address.addr[1];
  system_id[7] = address.addr[0];

  sc = sl_bt_gatt_server_write_attribute_value(gattdb_system_id,
                                               0,
                                               sizeof(system_id),
                                               system_id);
  sl_app_assert(sc == SL_STATUS_OK,
                "[E: 0x%04x] Failed to write attribute\n",
                (int)sc);

  // Create an advertising set.
  sc = sl_bt_advertiser_create_set(&advertising_set_handle);
  sl_app_assert(sc == SL_STATUS_OK,
                "[E: 0x%04x] Failed to create advertising set\n",
                (int)sc);
  // Set TX Power to 0 dBm
  sl_bt_system_set_tx_power(0,0,0,0);

  // Set advertisements to all 3 channels
  sl_bt_advertiser_set_channel_map(0, 1);

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

  // Set advertising interval to 1000ms.
  sc = sl_bt_advertiser_set_timing(
    advertising_set_handle,
    1600, // min. adv. interval (milliseconds * 1.6)
    1600, // max. adv. interval (milliseconds * 1.6)
    0,   // adv. duration
    0);  // max. num. adv. events
  sl_app_assert(sc == SL_STATUS_OK,
                "[E: 0x%04x] Failed to set advertising timing\n",
                (int)sc);
  // Start general advertising and enable connections.
  sc = sl_bt_advertiser_start(
    advertising_set_handle,
    advertiser_user_data,
    advertiser_non_connectable);
  sl_app_assert(sc == SL_STATUS_OK,
                "[E: 0x%04x] Failed to start advertising\n",
                (int)sc);
  break;

// -------------------------------
// This event indicates that a new connection was opened.
case sl_bt_evt_connection_opened_id:
  break;

// -------------------------------
// This event indicates that a connection was closed.
case sl_bt_evt_connection_closed_id:
  // Restart advertising after client has disconnected.
  sc = sl_bt_advertiser_start(
```

```
        advertising_set_handle,
        advertiser_general_discoverable,
        advertiser_connectable_scannable);
      sl_app_assert(sc == SL_STATUS_OK,
                    "[E: 0x%04x] Failed to start advertising\n",
                    (int)sc);
      break;

    ///////////////////////////////////////////////////////////////////////////
    // Add additional event handlers here as your application requires!      //
    ///////////////////////////////////////////////////////////////////////////

    // -------------------------------
    // Default event handler.
    default:
      break;
  }
}
```

4.2 Part 2 – Low Power Thermometer App.c

```
/***************************************************************************//**
 * @file
 * @brief Core application logic.
 *******************************************************************************
 * # License
 * <b>Copyright 2020 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.
 *
 ******************************************************************************/
#include "em_common.h"
#include "sl_app_assert.h"
#include "sl_bluetooth.h"
#include "gatt_db.h"
#include "app.h"
#include "sl_health_thermometer.h"
#include "tempdrv.h"

static uint8_t conn_handle = 0xff;
void temperatureMeasure()
{
  int32_t temperature;
  temperature = TEMPDRV_GetTemp()*1000;

  // Send temperature measurement indication to connected client.
  sl_bt_ht_temperature_measurement_indicate(conn_handle,temperature, false);
}

// The advertising set handle allocated from Bluetooth stack.
static uint8_t advertising_set_handle = 0xff;

/***************************************************************************//**
 * Application Init.
 ******************************************************************************/
SL_WEAK void app_init(void)
{
  /////////////////////////////////////////////////////////////////////////////
  // Put your additional application init code here!                         //
  // This is called once during start-up.                                    //
  /////////////////////////////////////////////////////////////////////////////
}
```

```
/**************************************************************************//**
 * Application Process Action.
 *****************************************************************************/
SL_WEAK void app_process_action(void)
{
  /////////////////////////////////////////////////////////////////////////////
  // Put your additional application code here!                              //
  // This is called infinitely.                                              //
  // Do not call blocking functions from here!                               //
  /////////////////////////////////////////////////////////////////////////////
}

/**************************************************************************//**
 * Bluetooth stack event handler.
 * This overrides the dummy weak implementation.
 *
 * @param[in] evt Event coming from the Bluetooth stack.
 *****************************************************************************/
void sl_bt_on_event(sl_bt_msg_t *evt)
{
  sl_status_t sc;
  bd_addr address;
  uint8_t address_type;
  uint8_t system_id[8];

  switch (SL_BT_MSG_ID(evt->header)) {
    // -------------------------------
    // This event indicates the device has started and the radio is ready.
    // Do not call any stack command before receiving this boot event!
    case sl_bt_evt_system_boot_id:

      // Extract unique ID from BT Address.
      sc = sl_bt_system_get_identity_address(&address, &address_type);
      sl_app_assert(sc == SL_STATUS_OK,
                    "[E: 0x%04x] Failed to get Bluetooth address\n",
                    (int)sc);

      // Pad and reverse unique ID to get System ID.
      system_id[0] = address.addr[5];
      system_id[1] = address.addr[4];
      system_id[2] = address.addr[3];
      system_id[3] = 0xFF;
      system_id[4] = 0xFE;
      system_id[5] = address.addr[2];
      system_id[6] = address.addr[1];
      system_id[7] = address.addr[0];

      sc = sl_bt_gatt_server_write_attribute_value(gattdb_system_id,
                                                   0,
                                                   sizeof(system_id),
                                                   system_id);
      sl_app_assert(sc == SL_STATUS_OK,
                    "[E: 0x%04x] Failed to write attribute\n",
                    (int)sc);

      // Create an advertising set.
      sc = sl_bt_advertiser_create_set(&advertising_set_handle);
      sl_app_assert(sc == SL_STATUS_OK,
                    "[E: 0x%04x] Failed to create advertising set\n",
                    (int)sc);
      // Set TX Power to 0 dBm
      sl_bt_system_set_tx_power(0,0,0,0);

      // Set advertisements to all 3 channels
      sl_bt_advertiser_set_channel_map(0, 7);

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

```
      // Set advertising interval to 1000ms.
      sc = sl_bt_advertiser_set_timing(
        advertising_set_handle,
        1600, // min. adv. interval (milliseconds * 1.6)
        1600, // max. adv. interval (milliseconds * 1.6)
        0,   // adv. duration
        0);  // max. num. adv. events
      sl_app_assert(sc == SL_STATUS_OK,
                    "[E: 0x%04x] Failed to set advertising timing\n",
                    (int)sc);
      // Start general advertising and enable connections.
      sc = sl_bt_advertiser_start(
        advertising_set_handle,
        advertiser_general_discoverable,
        advertiser_connectable_scannable);
      sl_app_assert(sc == SL_STATUS_OK,
                    "[E: 0x%04x] Failed to start advertising\n",
                    (int)sc);
      break;

    // -------------------------------
    // This event indicates that a new connection was opened.
    case sl_bt_evt_connection_opened_id:
      // Set the connection handle
      conn_handle = evt->data.evt_connection_opened.connection;
      // Create a software timer with an interval of 1 second
      sl_bt_system_set_soft_timer(32768, 0, 0);

      // Request a change in the connection parameters of a Bluetooth connection
      // 200ms connection interval, latency of 4 intervals
      sl_bt_connection_set_parameters(evt->data.evt_connection_opened.connection, 160, 160, 5, 450, 0, 0xffff);
      break;

    case sl_bt_evt_system_soft_timer_id:
      temperatureMeasure();
      break;

    // -------------------------------
    // This event indicates that a connection was closed.
    case sl_bt_evt_connection_closed_id:
      // Restart advertising after client has disconnected.
      sc = sl_bt_advertiser_start(
        advertising_set_handle,
        advertiser_general_discoverable,
        advertiser_connectable_scannable);
      sl_app_assert(sc == SL_STATUS_OK,
                    "[E: 0x%04x] Failed to start advertising\n",
                    (int)sc);
      break;

    ///////////////////////////////////////////////////////////////////////////
    // Add additional event handlers here as your application requires!      //
    ///////////////////////////////////////////////////////////////////////////

    // -------------------------------
    // Default event handler.
    default:
      break;
  }
}
```