AN902: Building Low Power Networks with the Silicon Labs Connect Stack v2.x

This document illustrates techniques for reducing power consumption in a sensor network developed using Simplicity Studio and the Silicon Labs Connect Stack, distributed as part of the Silicon Labs Flex SDK (Software Development Kit). Techniques include creating sleepy end devices that, by their nature, consume less energy, reducing reporting frequency, and removing unnecessary peripheral energy use. The techniques are illustrated with the sensor and sink example applications provided with the Flex SDK. For more in-depth background on energy savings in Connect networks see *UG235.07: Energy Saving with Silicon Labs Connect*, part of the *Connect User's Guide* series.

Proprietary is supported on all EFR32FG devices. For others, check the device's data sheet under Ordering Information > Protocol Stack to see if Proprietary is supported. In Proprietary SDK version 2.7.n, Connect is not supported on EFR32xG22.

KEY POINTS

- Building example sensor and sink applications.
- Creating a sensor-sink network.
- Monitoring baseline average current consumption.
- Reducing energy use in the network.

1. Introducing Low Power Sensor Networks

A typical low power network includes a single "always-on" device that serves as the coordinator for the network and one or more end nodes operating as sleepy end devices. Such devices spend most of their time in deep sleep mode, waking only briefly to transmit data to the central coordinator. An example of a low power network is a simple sensor network where the sensors are all sleepy end devices and the central hub serves as the always-on coordinator, or sink, for the network.

The Silicon Labs Flex SDK includes example sensor and sink applications, based on the Connect stack, for exactly this scenario. Supported network sructures are star and extended star topologies:
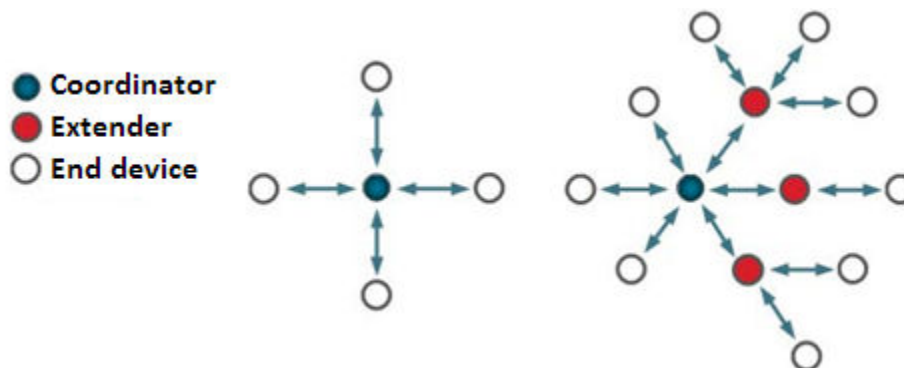
The example applications are:

- Connect (SoC): Sensor
- Connect (SoC): Sink

Together the applications demonstrate a star topology. Bi-directional communication is possible between the sensor(s) and the sink nodes. This document reviews using Simplicity Studio to build a sensor-sink network, and then to modify it for the lowest power consumption.

If you are not familiar with building the example applications in Simplicity Studio, please review *QSG138: Getting Started with the Silicon Labs Flex SDK for the Wireless Gecko (EFR32™) Portfolio*. If you are not familiar with the features and functions of the Silicon Labs Connect stack, see *UG103.12: Silicon Labs Connect Fundamentals*.

2. Create the Baseline Network

This chapter describes how to:
1. Build and run the the example coordinator/sink application.
2. Build and run a non-sleepy sensor application.
3. Create the network.
4. Evaluate baseline energy consumption.

You will need at least two devices connected to a computer running Simplicity Studio.

2.1 Build and Run the Coordinator/Sink Application

1. In the Simplicity Studio Launcher perspective, select the device to be used as the coordinator/sink.
2. Start a new project by clicking [**New Project**].
3. Create the project by selecting the Flex SDK, selecting the **Connect (SoC): Sink** example, and completing the steps in the New Project wizard.

The AppBuilder interface of the Simplicity IDE opens. Here you can configure the project's high level settings. For example, to configure radio parameters, click the **Radio Configuration** tab.

Once you have made any required changes (if any), click [**Generate**]. Studio generates the additional files needed for the project. A pop-up window appears about overwriting the `flex-callbacks.c` file, which is unselected by default. Leave it unselected.

Compile ( ) and debug ( ) the project. The coordinator application is now installed on the device.

To configure the coordinator, terminal software is necessary to set up the parameters. Use a separate application or the Simplicity Studio's built-in console tool. Issue `help` in the terminal to see some basic help from the firmware. When the firmware starts, issue `form 0` (form a network on channel 0) and `pjoin 255` (enable join of nodes for unlimited time) commands:

At this point, the coordinator is up and running, and ready to accept incoming joining requests, so sensor nodes can connect to the sink.

2.2 Build and and Run the Sensor Application

1. Return to the Launcher perspective and select the device to be used as the sensor.
2. Start a new project by clicking [**New Project**].
3. Create the project by selecting the Flex SDK, selecting the **Connect (SoC): Sensor** example, and completing the steps in the New Project wizard.

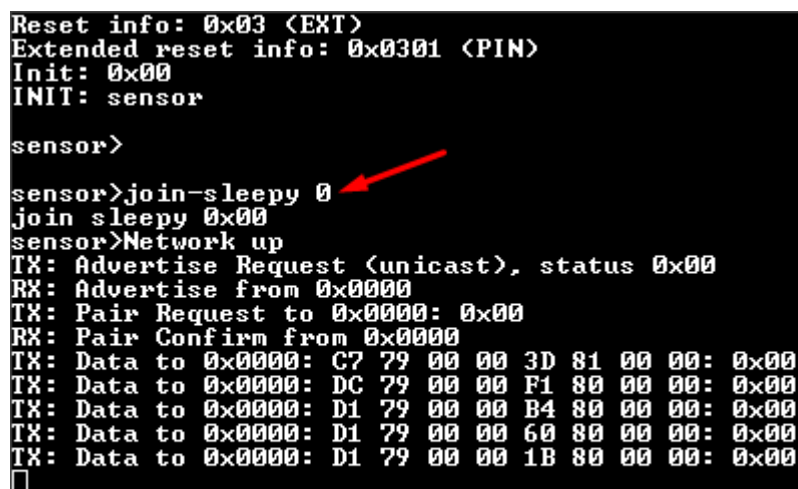All the settings can be kept as default, but the radio configuration for the sensor must match the sink settings.

Compile and debug/run the application.

2.3 Create the Network

The two most important terminal commands for the sensor are `join` and `join-sleepy`. On issuing `join`, the device attempts to connect to the coordinator and, if the join process is successful, the node becomes part of the network. The device will not try to enter to sleep or idle mode at all, keeping the device awake with relatively high current consumption.

The `join-sleepy` command enables the device to enter the sleep state. However, since by default the **Idle/Sleep** plugin is not enabled in AppBuilder, even connecting by `join-sleepy` will not cause the device to sleep whenever possible. However, using this command now does not cause any issues and it must be used later, when the **Idle/Sleep** plugin is enabled, in order to see energy savings from that feature.

After a successful join process, the device starts sending the sensor data to the sink periodically (with a 1s rate by default):

```
Reset info: 0x03 (EXT)
Extended reset info: 0x0301 (PIN)
Init: 0x00
INIT: sensor

sensor>

sensor>join-sleepy 0
join sleepy 0x00
sensor>Network up
TX: Advertise Request (unicast), status 0x00
RX: Advertise from 0x0000
TX: Pair Request to 0x0000: 0x00
RX: Pair Confirm from 0x0000
TX: Data to 0x0000: C7 79 00 00 3D 81 00 00: 0x00
TX: Data to 0x0000: DC 79 00 00 F1 80 00 00: 0x00
TX: Data to 0x0000: D1 79 00 00 B4 80 00 00: 0x00
TX: Data to 0x0000: D1 79 00 00 60 80 00 00: 0x00
TX: Data to 0x0000: D1 79 00 00 1B 80 00 00: 0x00
```

At this point, a network consisting of a coordinator/sink and one node/sensor should work, the sensor data should be sent by the sensor and received by the sink with 1 s periodicity. More nodes can be added to the network by downloading the sensor firmware into additional devices. The status code of a successful join is `0x90`: Network up, and the firmware prints `Network up` on the console.

The firmware prints numerical status codes on the console, instead of a textual description, if the sensor fails to join the network.

- `0x40`: No ACK received, the node to which a message was transmitted did not respond.
- `0x41`: Indirect MAC timeout, may occur if the sender did not get a poll request to send its pending message.
- `0x70`: Invalid call, usually when an unexpected operation is carried out (for example, joining when joined or sending when not joined, and so on).
- `0xAB`: No valid beacons, the other radio cannot be heard (for example, no antenna attached).

Before each step in the experiment with the sensor/sink examples it is a good practice to issue the `leave` command on both the coordinator/sink and sensors. For example, the coordinator maintains information about the previously connected nodes, based on their EUI64 ID. The nodes also store information and try to reconnect to the coordinator automatically at power-up based on this information. By issuing the `leave` command this information will be cleared.

2.4 Monitor Baseline Energy Use

The Energy Profiler is a useful tool to inspect current consumption real-time. For details on using the Energy Profiler see *UG343: Multi-Node Energy Profiler User's Guide*.

The following graph shows the consumption of a non-optimized sensor node.
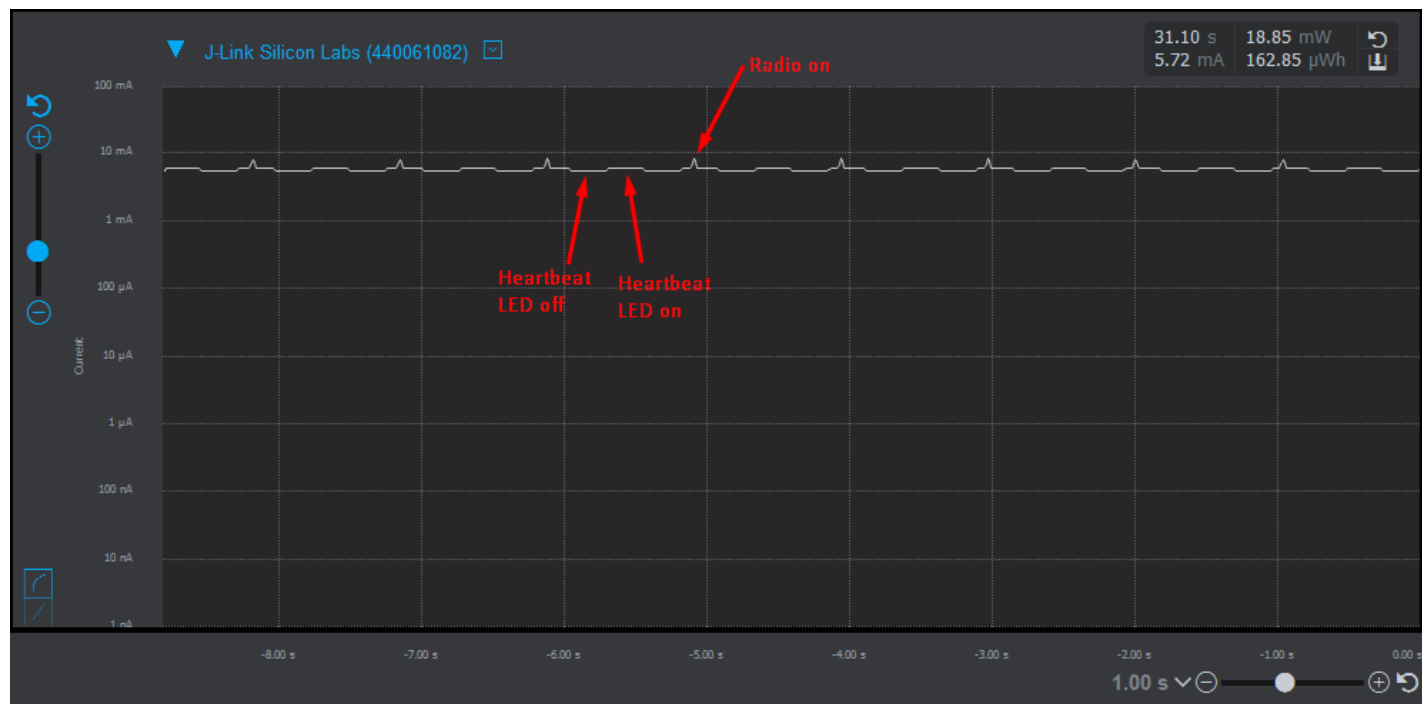
3. Reduce Power Consumption

This chapter describes techniques for reducing power consumption, including

- Enable the Idle/Sleep plugin
- Change Report Periodicity
- Change LED Status Behavior
- Disable the Heartbeat Feature
- Initialize the SPI Flash Chip

Finally, the reduced power consumption results are described.

3.1 Enable the Idle/Sleep Plugin

The first step, and the one that reduces the consumption the most, is enabling the **Idle/Sleep** plugin. The plugin enables entering EM2 mode when it is possible. The **Idle/Sleep** plugin can be enabled in AppBuilder under the **Plugins** tab by checking the **Idle/Sleep** plugin:
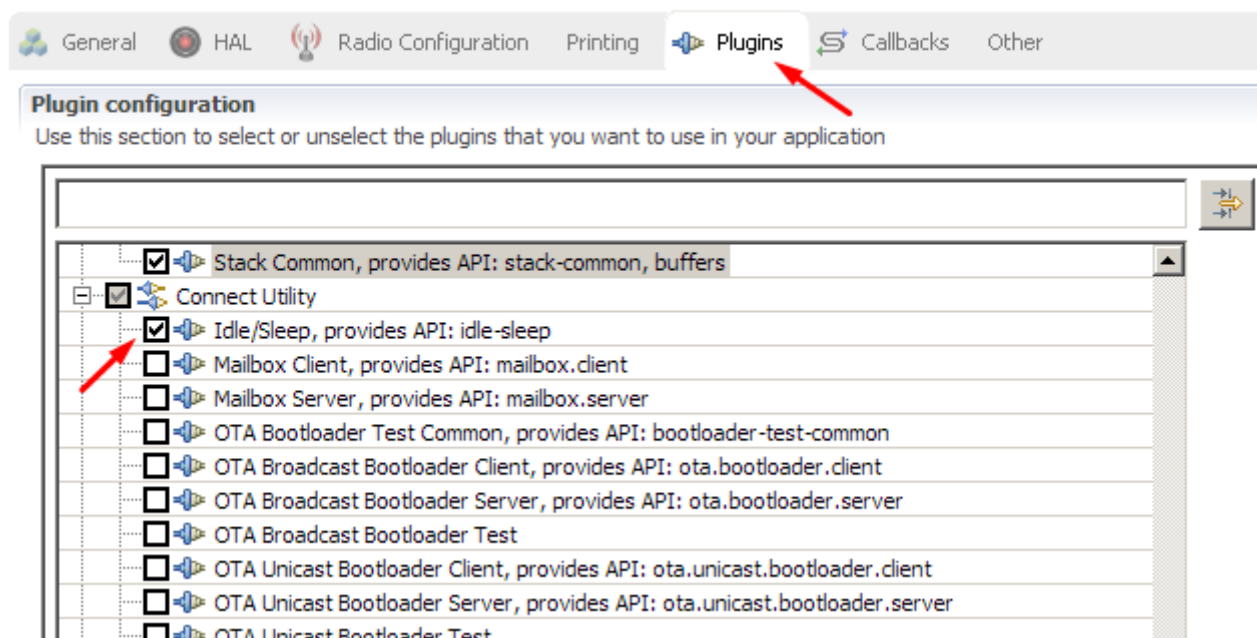
Note that if the **Idle/Sleep** plugin is enabled, issuing commands through CLI (UART) is not possible. Thus, it is practical to apply all other necessary settings before enabling the **Idle/Sleep** plugin.

3.2 Change Report Periodicity

Changing the report period can significantly reduce the average consumption if the sleep consumption is significantly lower than consumption in active periods. To change the report period, open **flex-callbacks.c** and modify the `sensorReportPeriodMs` value:

3.3 Change LED Status Behavior

By default, the network status indicator LED is on when the network stack is up. In other words, after the join process it is continuously on. Changing this behavior can reduce the consumption by a few hundred µA. The code responsible for setting the LED status is also in **flex-callbacks.c** in the `emberAfMainTickCallback` function:

```
void emberAfMainTickCallback(void)
{
  #ifndef UNIX_HOST
  if (emberStackIsUp()) {
    halSetLed(NETWORK_UP_LED);
  } else {
    halClearLed(NETWORK_UP_LED);
  }
  #endif
}
```

To change, either comment out the `halSetLed()` portion or invert the if statement.

3.4 Disable the Heartbeat Feature

In the example project, the **Heartbeat** plugin is enabled. To decrease the average consumption further, this feature can be disabled in AppBuilder under the **Plugins** tab by unchecking the **Heartbeat** plugin:
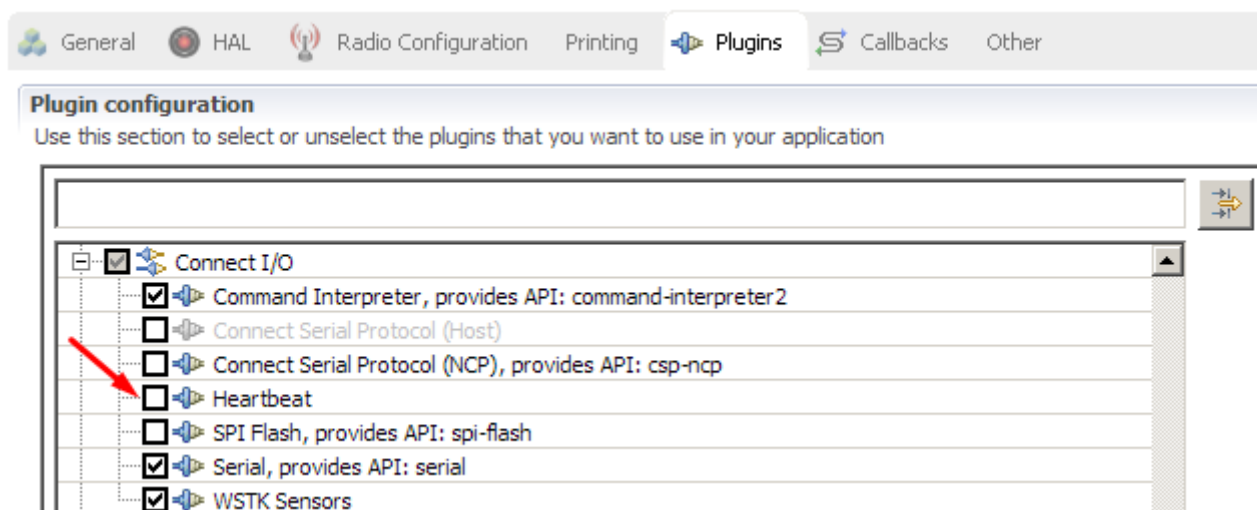
3.5 Initialize the SPI Flash Chip

By applying the modifications above the sleep current can be reduced to ~10-12 μA. However, all the radio boards designed by Silicon Labs contain an SPI flash chip that consumes ~8-10 μA if left uninitialized. The required **SPI Flash** plugin must be enabled:
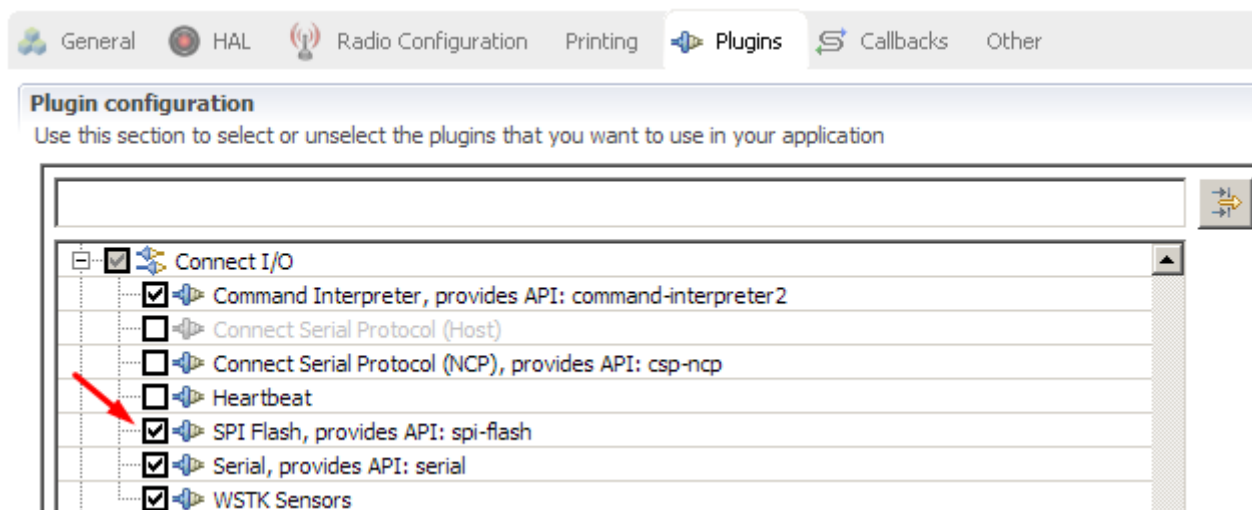
Additionally, the `halEepromInit()` and `halEepromShutdown()` functions that initialize the flash and set deep power down mode of the device have to be called. These should be added manually to **flex-callbacks.c** (preferably to `emberAfMainInitCallback`):

```
void emberAfMainInitCallback(void)
{
  emberAfCorePrintln("INIT: %p", EMBER_AF_DEVICE_NAME);
  emberAfCorePrintln("\n%p>", EMBER_AF_DEVICE_NAME);

  halEepromInit();
  halEepromShutdown();

  emberNetworkInit();
}
```

3.6 Reduced Power Consumption Results

Finally, if these modifications are applied the sleep mode current consumption should be reduced significantly:
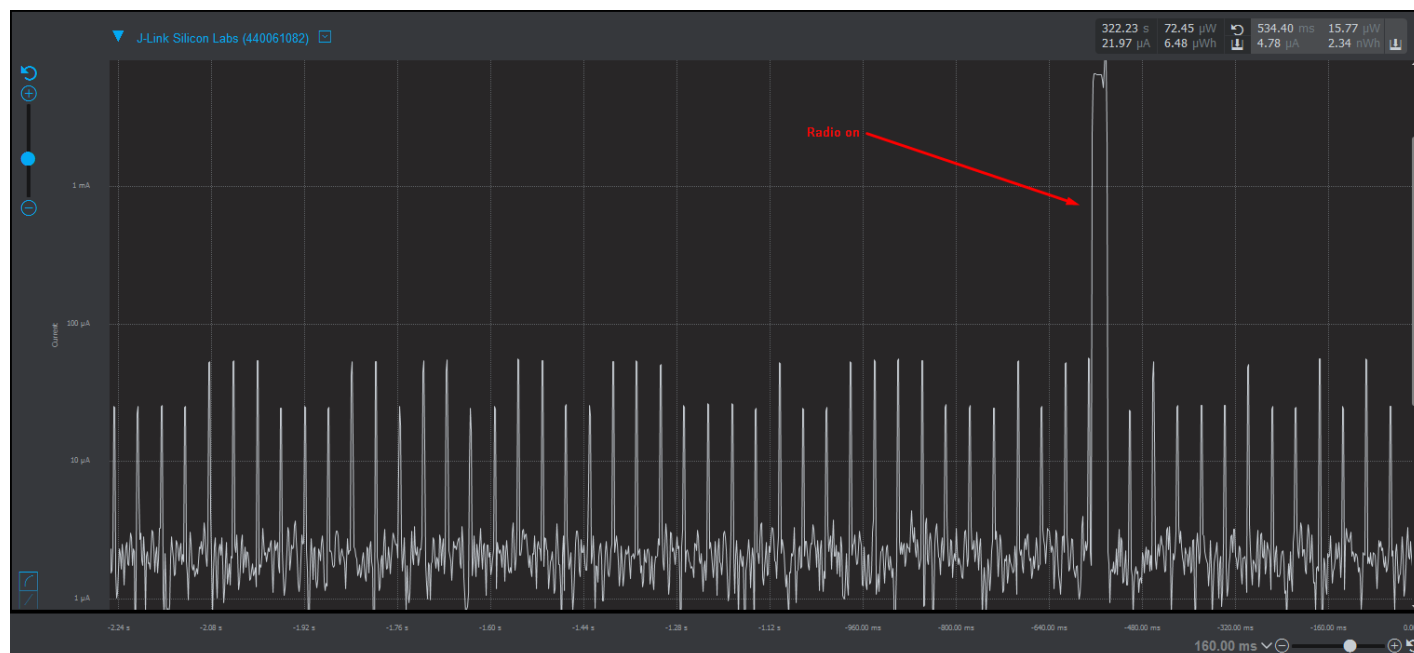
Note that the built-in current measurement of the WSTK board is not really accurate in the µA range and below. The sleep mode current consumption of the sensor node once all modifications have been applied should be in the 2-5 µA range.
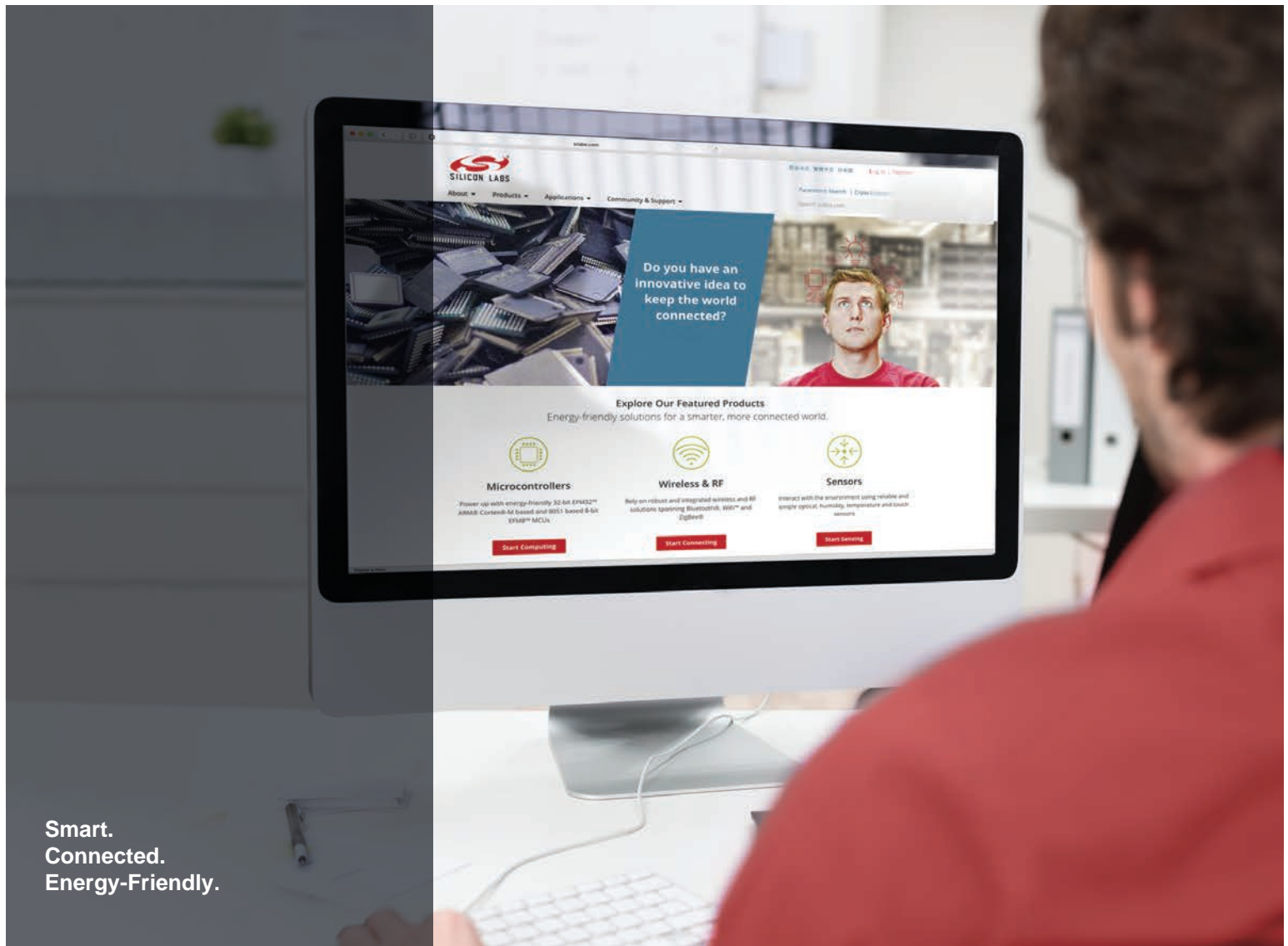
**Smart.
Connected.
Energy-Friendly.**

Products	Quality	Support and Community
www.silabs.com/products	*www.silabs.com/quality*	*community.silabs.com*
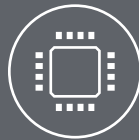
**Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA**

http://www.silabs.com