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

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.

KEY FEATURES

- Building example sensor and sink applications.
- Creating a network, joining the sensor as a sleepy end device.
- 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. 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: Application Development Fundamentals: Silicon Labs Connect*.

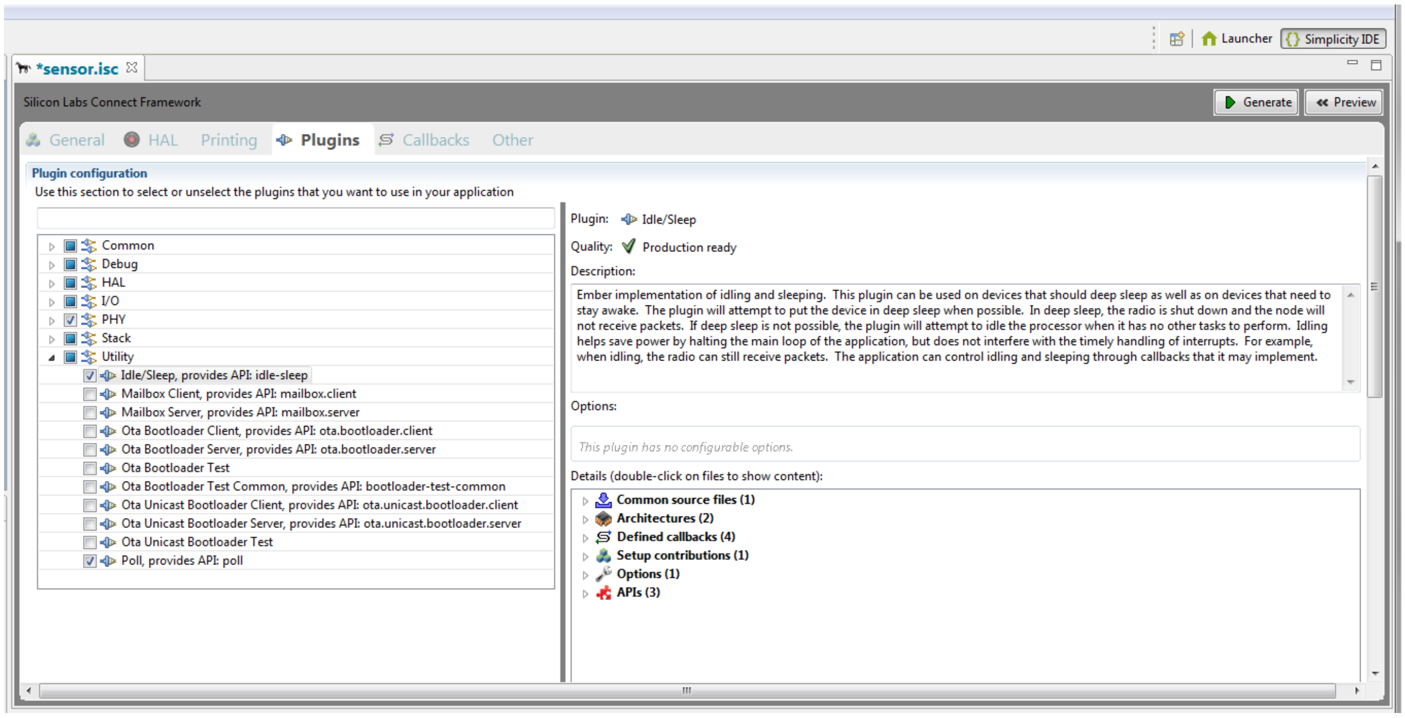
2 Theory of Operation

The process of reducing the energy use in a simple sensor-sink network includes the following steps:

1. Build the example sensor and sink applications, with the sensor application configured to run sleepy end devices.
2. Create the network, joining the sensor as a sleepy end device.
3. Monitor the sensor's baseline average current consumption.
4. Reduce the frequency with which the sensor wakes up and sends data to the coordinator.
5. Remove unnecessary sources of energy use.

2.1 Step 1: Build the Example Sensor and Sink Applications

When building the sensor application, after you have entered configuration information on the General tab, click the Plugins tab. Under **Utilities**, check the Idle/Sleep plugin, so that the sensor acts as a sleepy end device.



Plugin configuration
Use this section to select or unselect the plugins that you want to use in your application

- Common
- Debug
- HAL
- I/O
- ✓ PHY
- Stack
- Utility
  - ✓ Idle/Sleep, provides API: idle-sleep
  - Mailbox Client, provides API: mailbox.client
  - Mailbox Server, provides API: mailbox.server
  - Ota Bootloader Client, provides API: ota.bootloader.client
  - Ota Bootloader Server, provides API: ota.bootloader.server
  - Ota Bootloader Test
  - Ota Bootloader Test Common, provides API: bootloader-test-common
  - Ota Unicast Bootloader Client, provides API: ota.unicast.bootloader.client
  - Ota Unicast Bootloader Server, provides API: ota.unicast.bootloader.server
  - Ota Unicast Bootloader Test
  - ✓ Poll, provides API: poll

Plugin: Idle/Sleep

Quality: ✓ Production ready

Description:

Ember implementation of idling and sleeping. This plugin can be used on devices that should deep sleep as well as on devices that need to stay awake. The plugin will attempt to put the device in deep sleep when possible. In deep sleep, the radio is shut down and the node will not receive packets. If deep sleep is not possible, the plugin will attempt to idle the processor when it has no other tasks to perform. Idling helps save power by halting the main loop of the application, but does not interfere with the timely handling of interrupts. For example, when idling, the radio can still receive packets. The application can control idling and sleeping through callbacks that it may implement.

Options:

This plugin has no configurable options.

Details (double-click on files to show content):

- **Common source files (1)**
- **Architectures (2)**
- **Defined callbacks (4)**
- **Setup contributions (1)**
- **Options (1)**
- **APIs (3)**

Generate the application, and load it and the sink application onto the WSTK hardware.

2.2 Step 2: Create the Sensor Network

Next, use the console in the Simplicity IDE to connect the sensor and sink into a network. On the sink, start the network by issuing the following commands:

- `form 0` - Starts the network on channel 0.
- `pjoin 0xff` - Permanently opens the network for joining.

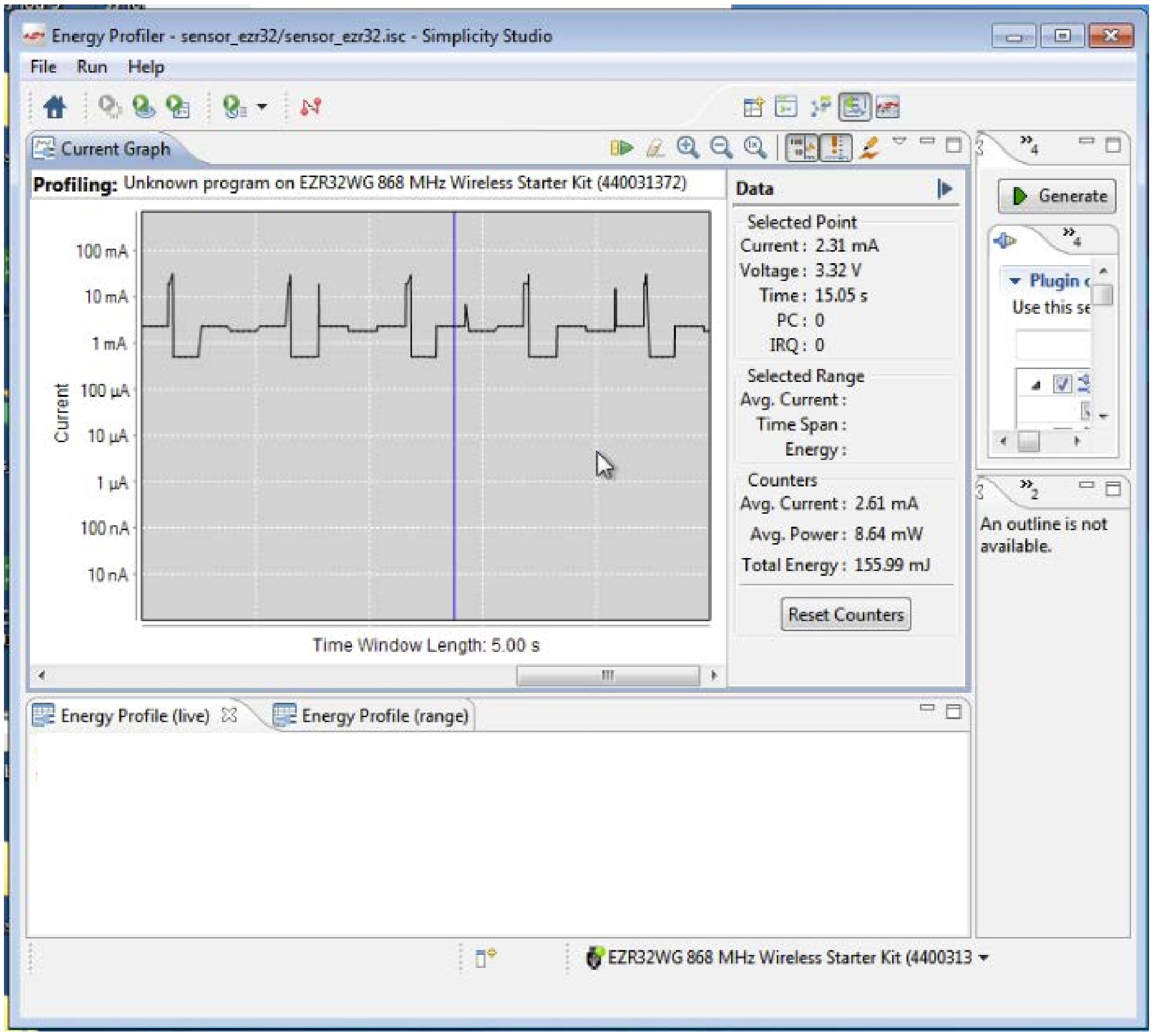
On the sensor, join the network as a sleepy end device by issuing the following commands:

- `join-sleepy 0` – Causes the sensor to join the sink's network on channel 0.

Close the console. Right-click on the sensor and click **Disconnect**. Return to the Launcher perspective, open the **Tools** menu, and select **Energy Profiler**.

2.3 Step 3: Monitor Energy Use

The Energy Profiler perspective is shown in the following figure.



Profiling: Unknown program on EZR32WG 868 MHz Wireless Starter Kit (440031372)

Selected Point
Current : 2.31 mA
Voltage : 3.32 V
Time : 15.05 s
PC : 0
IRQ : 0

Selected Range
Avg. Current :
Time Span :
Energy :

Counters
Avg. Current : 2.61 mA
Avg. Power : 8.64 mW
Total Energy : 155.99 mJ

An outline is not available.

The drop-down list on the bottom right shows the device being profiled. Make sure the WSTK development board with the sensor application loaded is shown. If not, drop down the list and select it.

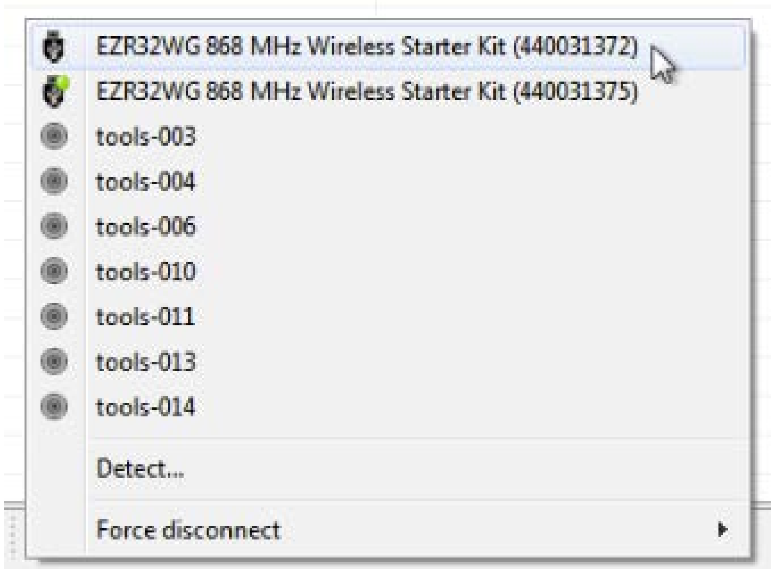
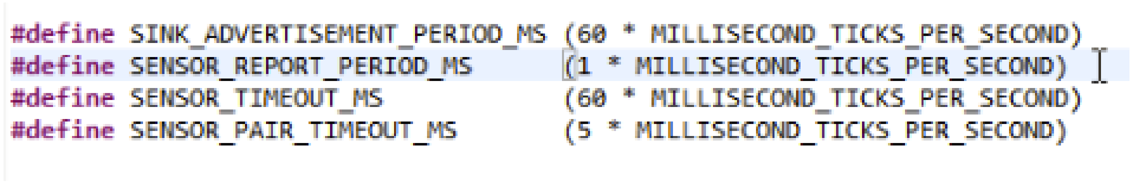

EZR32WG 868 MHz Wireless Starter Kit (440031372)
EZR32WG 868 MHz Wireless Starter Kit (440031375)
tools-003
tools-004
tools-006
tools-010
tools-011
tools-013
tools-014
Detect...
Force disconnect

Figure 1. The Energy Profiler Device Drop Down

The Current Graph above shows that the sensor's average current consumption out of the box, with the sleep plugin included, is still quite high at 2.61 mA. A target average sleep current consumption is approximately 2.4 µA. The first step to achieving this target is to reduce the reporting frequency.

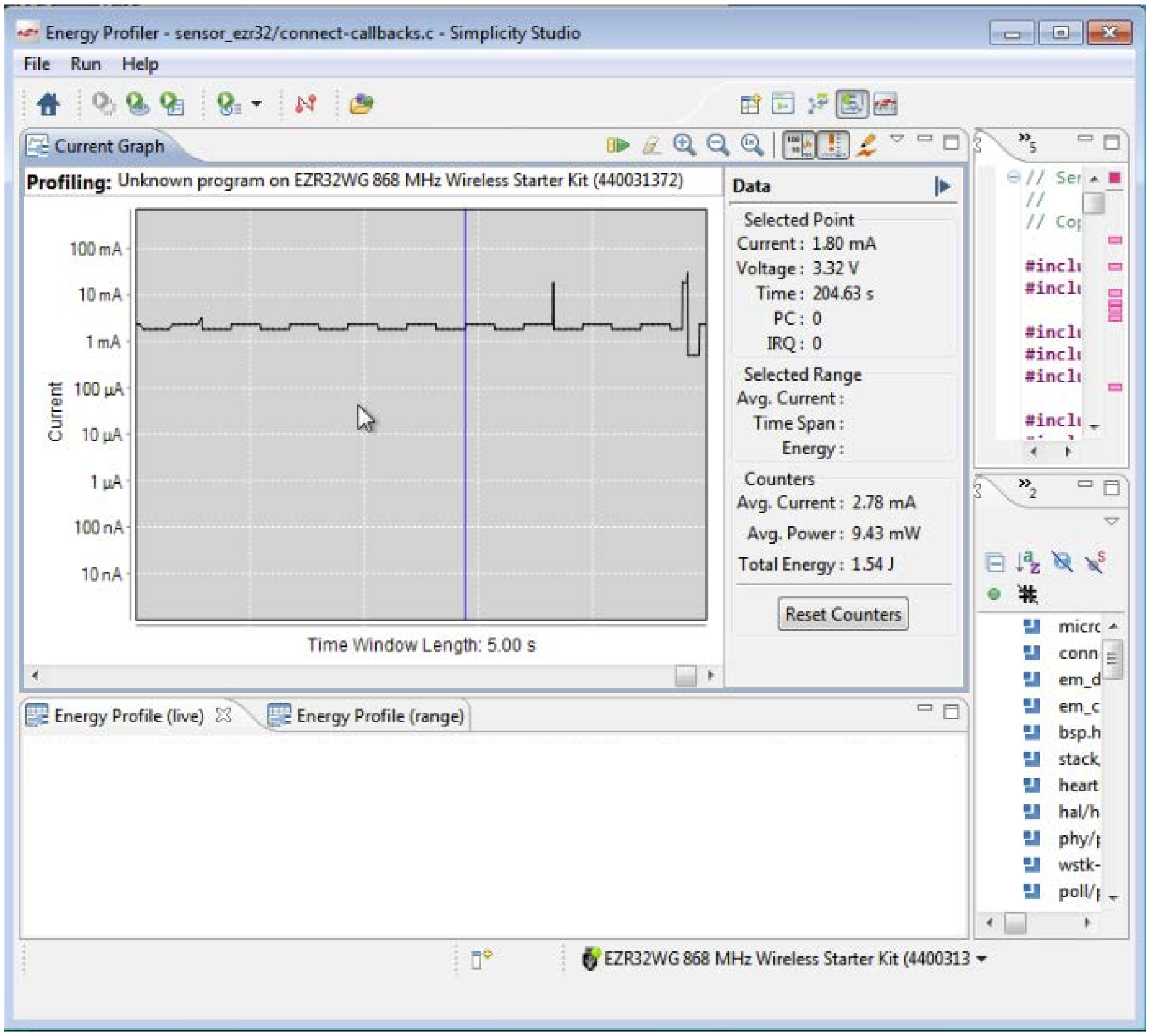
2.4 Step 4: Reduce Reporting Frequency

The reporting frequency is the frequency with which the sleepy end device wakes up and sends data to the sink. Reducing that frequency should reduce energy consumption. In the sensor application, the reporting frequency is controlled by a `SENSOR_REPORT_PERIOD_MS` define at the top of the connect-callbacks.c file. The default reports once every second. To reduce the reporting interval, change this to report about once every 5 seconds (5 * MILLISECOND_TICKS_PER_SECOND).



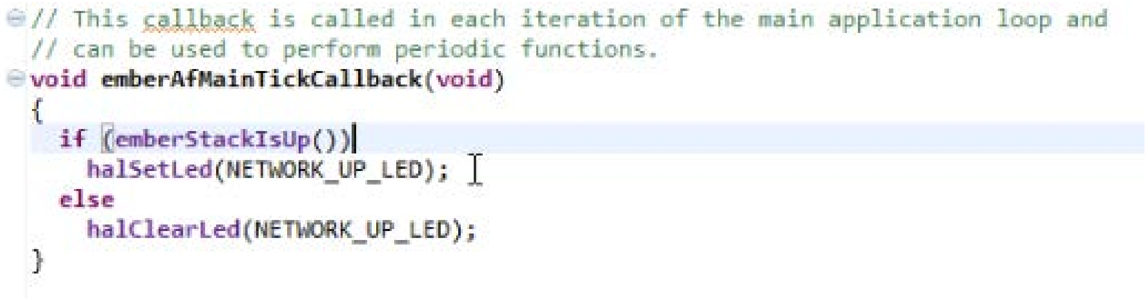
```
#define SINK_ADVERTISEMENT_PERIOD_MS (60 * MILLISECOND_TICKS_PER_SECOND)
#define SENSOR_REPORT_PERIOD_MS      (1 * MILLISECOND_TICKS_PER_SECOND)
#define SENSOR_TIMEOUT_MS            (60 * MILLISECOND_TICKS_PER_SECOND)
#define SENSOR_PAIR_TIMEOUT_MS       (5 * MILLISECOND_TICKS_PER_SECOND)
```

Recompile the application, load it onto the sensor module, and connect the Energy Profiler. As shown in ~~Figure 5~~ the following figure, simply changing the reporting interval reduces average current consumption to 2.78 mA. However, this is still much higher than the target level. To reduce it further, determine if the implementation contains any unnecessary sources of energy use.



2.5 Step 5. Remove Unnecessary Sources of Energy Use

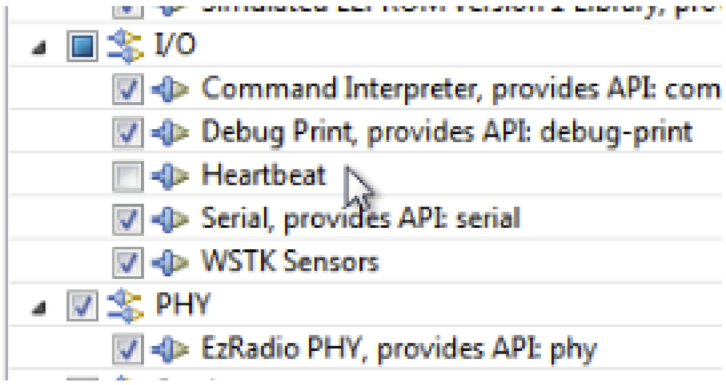
The sensor development board has one LED that is always on and another that is flashing as a heartbeat. Neither is necessary for this application. The always-on LED is controlled by a network join command in the `emberAfMainTickCallback()`, located in the connect-callbacks.c file, shown below..



```
// This callback is called in each iteration of the main application loop and
// can be used to perform periodic functions.
void emberAfMainTickCallback(void)
{
  if (emberStackIsUp())
    halSetLed(NETWORK_UP_LED);
  else
    halClearLed(NETWORK_UP_LED);
}
```

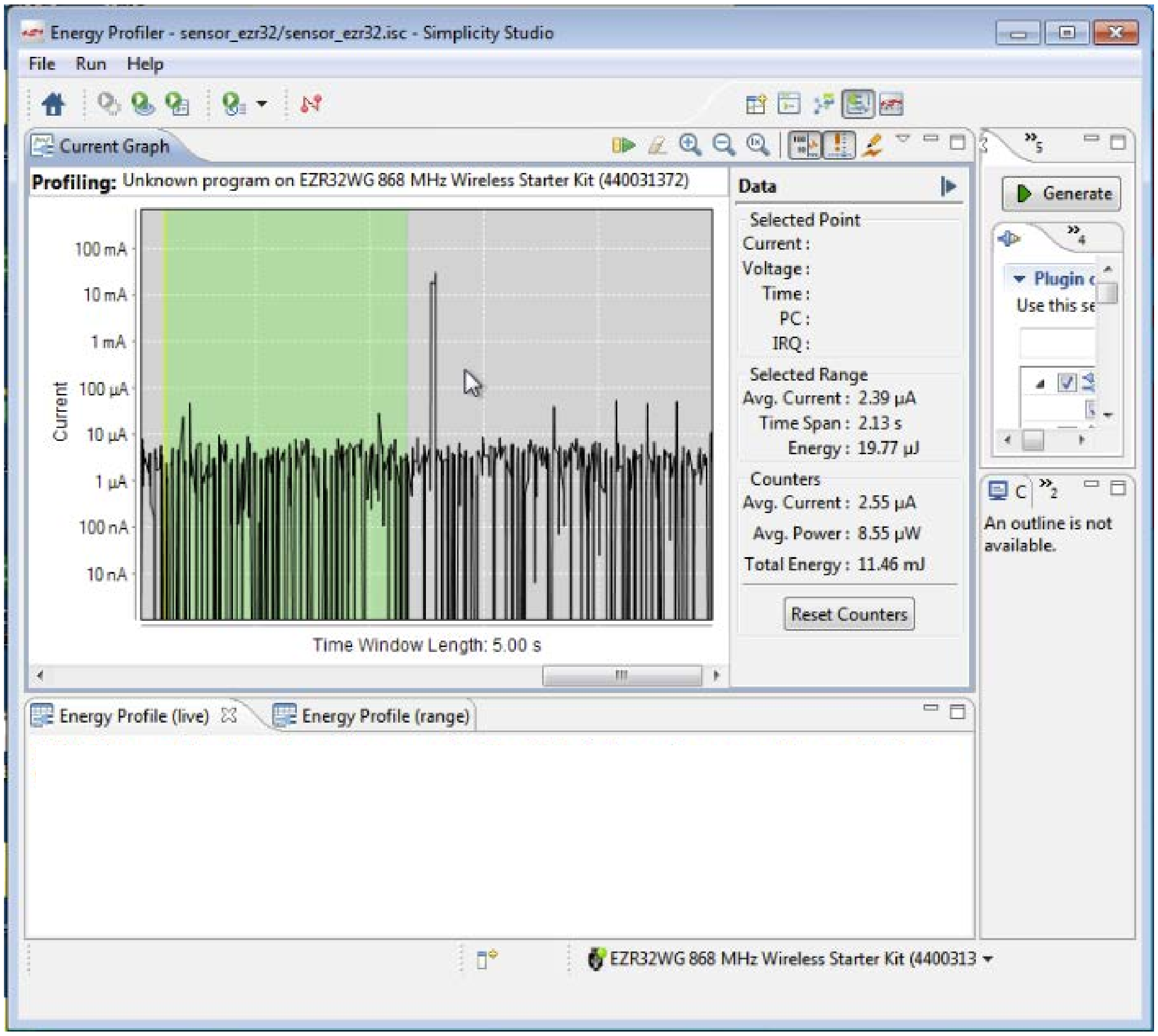
Comment this code out for some major savings in current consumption.

The heartbeat is provided in a plugin. On the Application Builder Plugins tab, uncheck that plugin, as shown.


I/O
Command Interpreter, provides API: com
Debug Print, provides API: debug-print
Heartbeat
Serial, provides API: serial
WSTK Sensors
PHY
EzRadio PHY, provides API: phy

Regenerate and rebuild the application, load it, and connect the Energy Profiler to see the effect of turning off the always-on and heart-beat LED functions.

As shown the following figure, average current consumption for the selected range, which represents the sleep current consumption, is now on target at 2.39 µA.


Energy Profiler - sensor_ezr32/sensor_ezr32.isc - Simplicity Studio
File
Run
Help
Current Graph
Profiling: Unknown program on EZR32WG 868 MHz Wireless Starter Kit (440031372)
100 mA
10 mA
1 mA
100 µA
10 µA
1 µA
100 nA
10 nA
Current
Time Window Length: 5.00 s
Data
Generate
Plugin c
Use this se
An outline is not available.
Energy Profile (live)
Energy Profile (range)
EZR32WG 868 MHz Wireless Starter Kit (4400313

Selected Point
Current :
Voltage :
Time :
PC :
IRQ :

Selected Range
Avg. Current : 2.39 µA
Time Span : 2.13 s
Energy : 19.77 µJ

Counters
Avg. Current : 2.55 µA
Avg. Power : 8.55 µW
Total Energy : 11.46 mJ

Reset Counters