Blue Gecko Xpress Lab Worksheet

This lab provides hands-on experience with Blue Gecko Xpress BGX13 using the BGX13P EXP board, an EFM8UB1 STK, and the BGX Commander mobile apps.

## Key points

* How to use BGX13P out-of-box
* Operating BGX in BGX-to-phone mode with BGX commander
* Operating BGX in BGX-to-BGX mode

# Getting Started

* Install the following PC software:
  + Simplicity Studio: <https://www.silabs.com/products/development-tools/software/simplicity-studio>
  + Silicon Labs VCP driver: <https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers>)
* Copy the contents of this Dropbox folder onto your laptop:
  + <https://www.dropbox.com/sh/qo8b47iii6x73la/AAAIiy_v-bAF49P_I2EUEEefa?dl=0>)
* Download one of the two available mobile apps to your iOS/Android phone:
  + iOS app (<https://itunes.apple.com/us/app/bgxcommander/id1350920514?mt=8&ign-mpt=uo%3D4>)
  + Android app (<https://play.google.com/apps/testing/com.silabs.bgxcommander?pli=1>)
* Make sure you have the following, which you’ll receive on training day:
  + BGX13P EXP board
  + EFM8UB1 STK

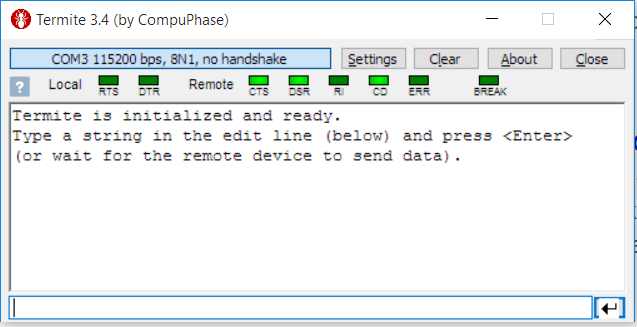
## Hardware setup

Setting up the BGX13P EXP board

1. Install the CP2102N VCP driver from the link above if it has not already been installed on your laptop.
2. Connect the micro USB cable between the BGX EXP board and your laptop.

### Setting up Termite serial terminal

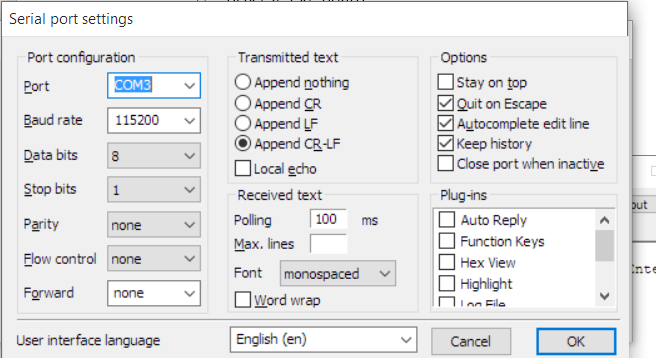
1. Run Termite once installed
2. Go to settings:



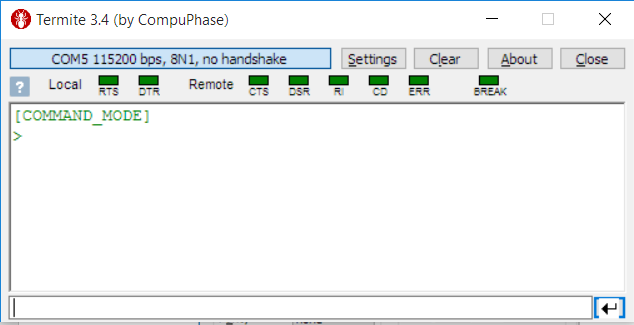
1. Select the com port of the BGX EXP board from the ‘Port’ pulldown.

Select 115200 as the baud rate.

Click ‘OK’.



1. Press ‘RESET’ on the BGX EXP board to see output from the UART interface to confirm a successful connection.



## Finding documentation

In addition to the datasheets for the BGX13P and BGX13S, available through Simplicity Studio and through the silabs.com website at launch, BGX control and functionality is defined through online documentation, which will be available at launch at docs.silabs.com.

For this training, I have included the documentation site as a local, static website available at the Dropbox link above.

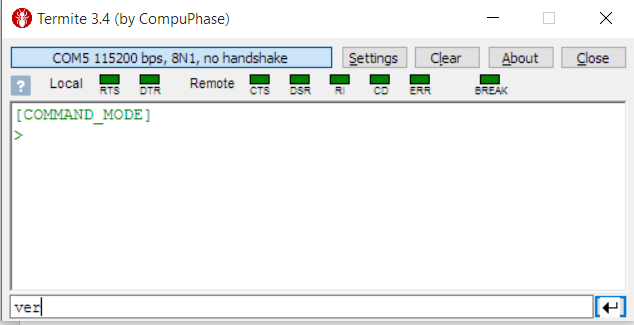
# Interfacing with BGX through its serial interface

We’ll use Termite as a terminal program to communicate with the serial interface of the BGX13. Termite simulates an embedded host processor that would interface with a BGX13 in an end customer product.

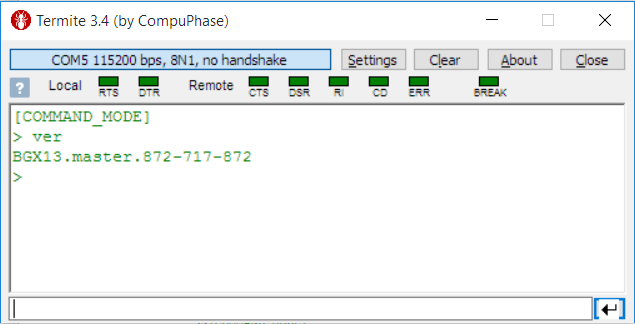
## Sending commands to the BGX13

In Termite, check the version of the BGX13P by running the ‘ver’ command.

In Termite, enter ‘ver’ in the text box at the bottom of the window.



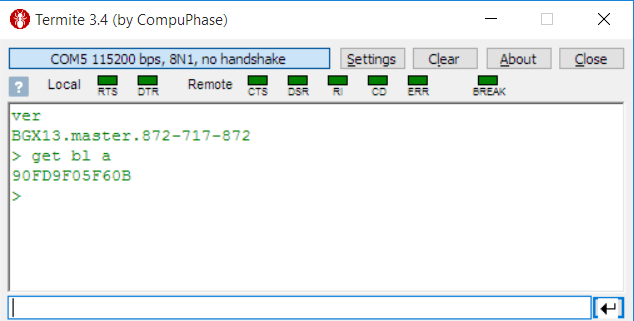
1. Hit enter and observe the output from the BGX.



## Read the BGX Bluetooth address

In steps to come, we will need the Bluetooth address of the BGX module on your EXP board. Let’s read this address, which is stored in the BGX as a variable, using the ‘get’ command to read this variable.

1. In Termite, enter the command ‘get bl a’
2. Record the resulting Bluetooth address for future use.



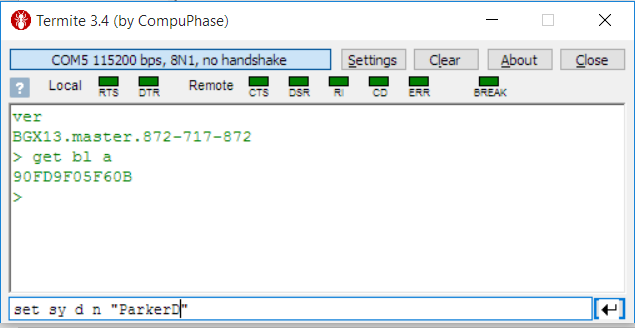
## Setting a device name

In the previous step, we used the ‘get’ command to read a variable. In this step, we’ll use the ‘set’ command, which writes a value to a variable. We’ll then use ‘save’ to store that value into the BGX13’s non-volatile memory.

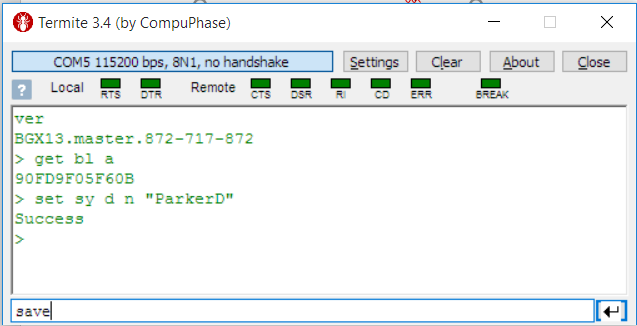
1. Execute the command “set sy d n <name>” to the BGX

**Customizing with ‘set’**

BGX13 performance characteristics like advertising intervals and duration using variables that can be customized using the ‘set’ command. These variables can be adjusted at runtime and also stored in non-volatile memory.



1. Execute the command ‘save’ to store the value in non-volatile memory.



**The Power of ‘save’**

The save command can be used at customer factory to store customized settings in non-volatile memory.

**BGX interface demo review**

**What we just learned:**

* The BGX13 uses a serial interface for configuration and communication.
* Performance characteristics can be saved to non-volatile memory to persist across power cycles.
* API documentation available online provides definitions and usage examples.

**Why it’s useful:**

* Variables enable customization to easily optimize BGX13 for power-sensitive or high-throughput applications
* In many applications where BGX13 is operating in a peripheral mode, **the command interface is not needed**. In those cases, the embedded host can monitor flow control signals to know when a link is established and communication can begin.

# BGX-to-Phone demo

In this section we’ll introduce the BGX commander mobile app and use it as a demo vehicle for BGX-to-phone interaction.

Before this portion of the demo, make sure you’ve installed either the iOS app or the Android app, using the links at the top of this document.

The screen captures in this section show the iOS app.

Note that you should keep the Termite program running on your laptop for this portion of the demo. Termite simulates the embedded host side of the BLE link. For this demo, the BGX will operate in a peripheral role, and the BGX app will function in the central role.

### Scanning for your BGX device

First we’ll scan for the BGX device.

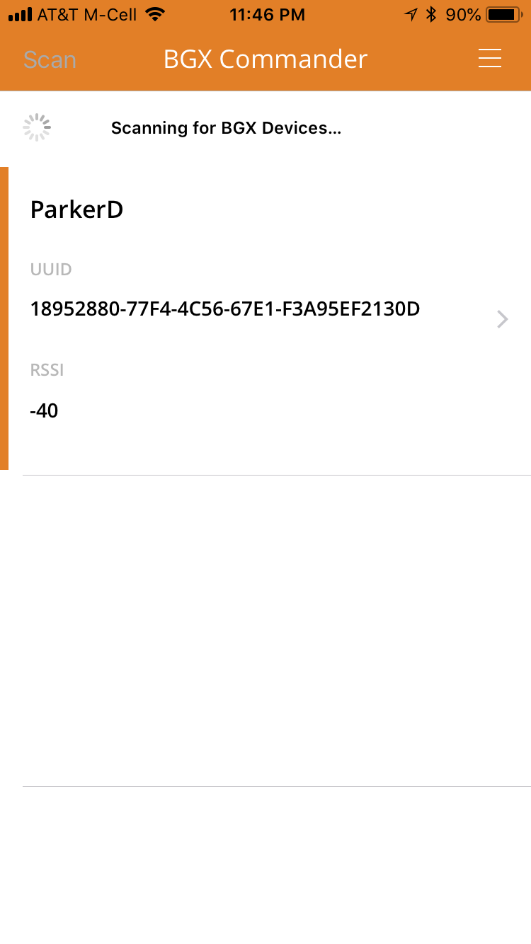
1. Run the BGX Commander app. Wait for the scan results to appear and find your BGX13 device. The device will use the name set in the previous demo steps. In this example, the device name ‘ParkerD’ appears in the scan results.

**Using the Xpress framework**

The BGX Commander app was developed using the Xpress framework for BGX13 connection and communication.

All of the BGX interactions in this demo call into this framework, which abstracts BLE complexity into a simple API.

The source code for this app and the framework will be available in Simplicity Studio at launch.



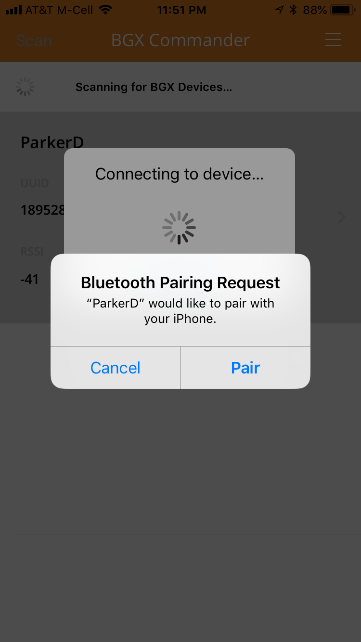
1. Click on your BGX device. If using the iOS app, accept the pairing request when the screen appears.

**Pairing, bonding, and encryption**

The BGX13 uses encryption for connecting and communicating.

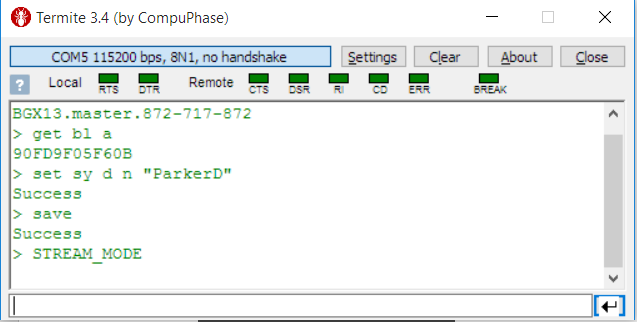
Bonding can be enabled or disabled through the command interface.

With bonding enabled, the ‘Bluetooth Pairing Request’ will only happen during the first connection with the BGX.



After pairing completes, the app switches to a view where we can communicate with the BGX13.

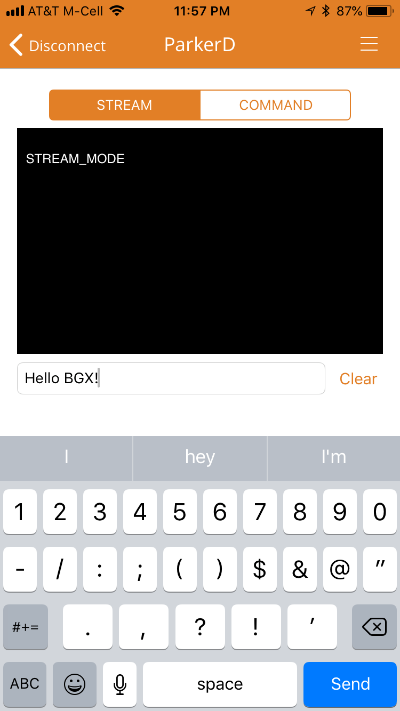
Note that the BGX has signaled through its serial interface that connection has completed successfully by showing that it has entered ‘STREAM\_MODE’.



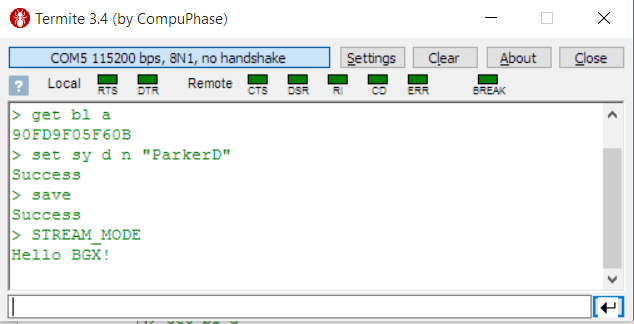
## Sending and receiving data

At this point, the interface is in stream mode, which means that the BLE link has created a raw data pipe between the embedded system and the app. Now we can send characters between the two points of the link.

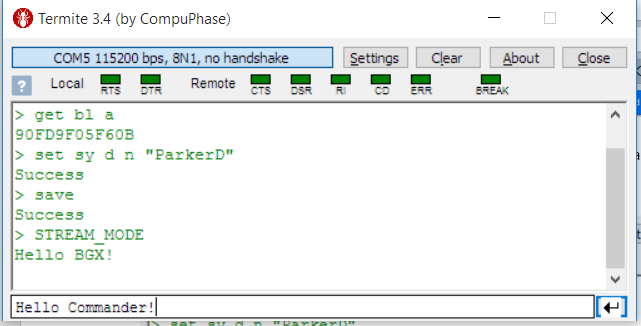
1. Type a string into the text box and hit ‘Send’ to transmit data to the BGX by entering a line in the app and sending.



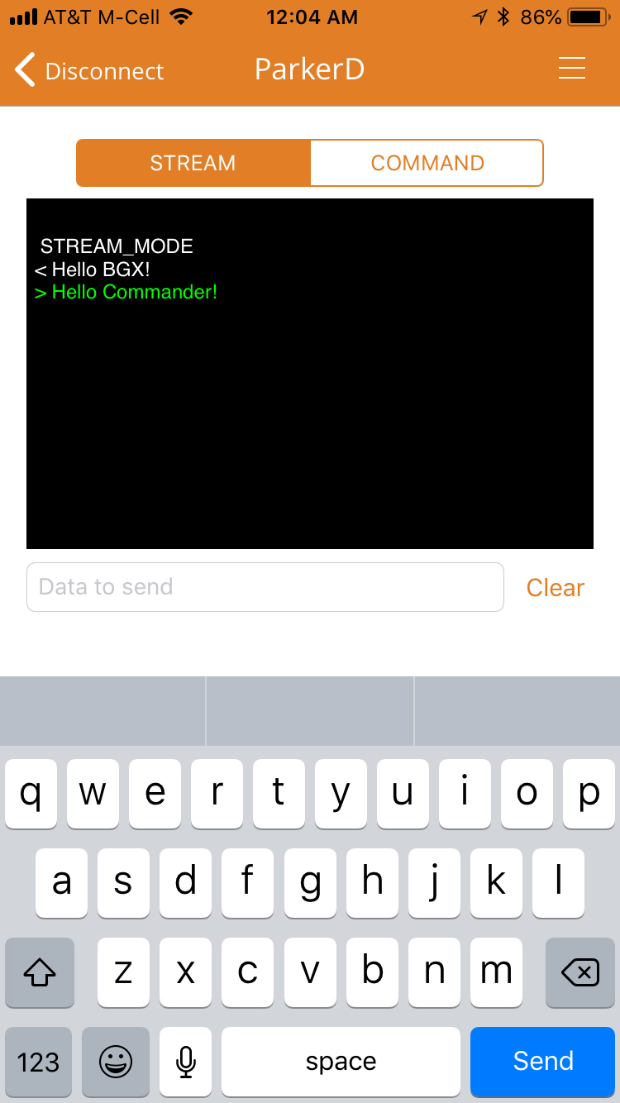
Note that the data was sent out of the TX pin on the BGX, and displayed in the terminal:



1. Send data to the app through the BGX by typing a string into Termite’s text box and hitting enter.



Note that in the app, the string gets received from the BGX and displayed.

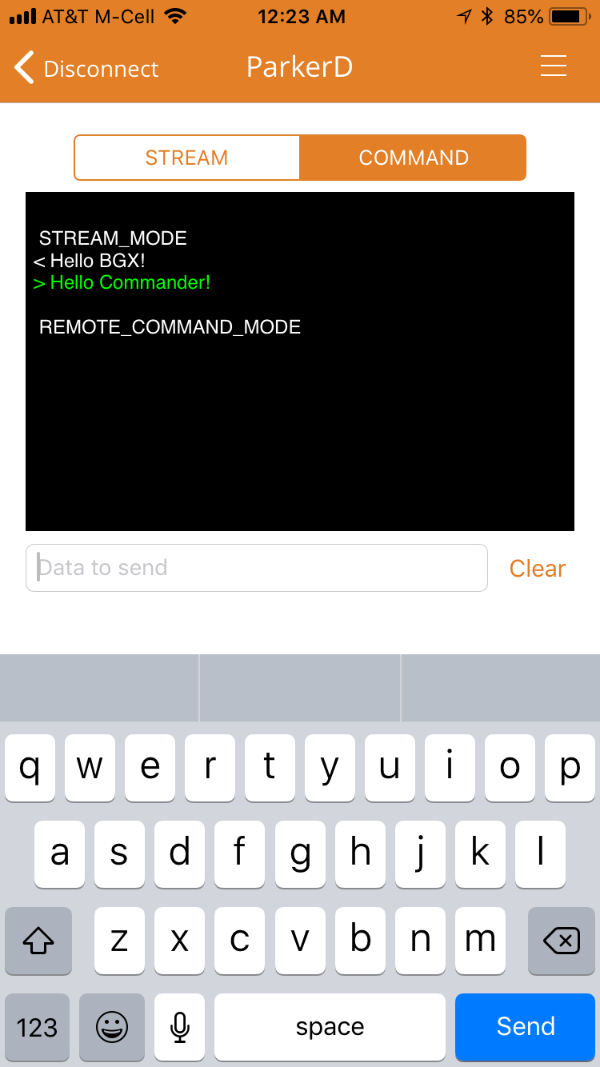


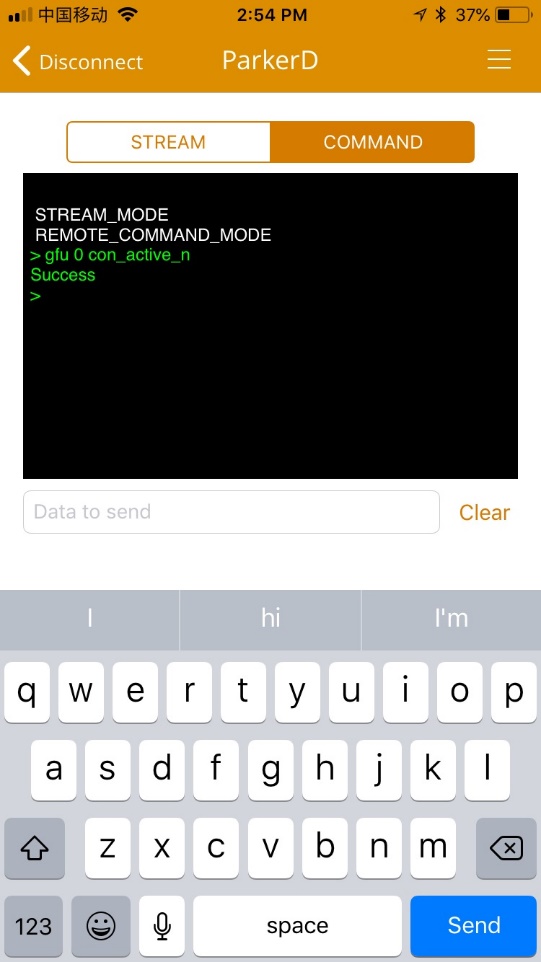
### Remote command execution

In addition to supporting stream mode, the BGX Commander app supports remote command mode. In this mode, all of the commands that can be executed through the BGX’s serial interface can also be executed remotely, through a connected Bluetooth link.

In this demo, we’ll execute commands to configure a BGX port pin to toggle an LED indicating BLE connection state.

1. In BGX commander, click ‘COMMAND’ to switch to remote command mode.



1. Configure GPIO 0 to be an output pin with “gfu 0 con\_active\_n”. Note that ‘0’ signifies GPIO 0, and ‘con\_active\_n’ configures the pin to digital output initialized to logic low. 

**BGX13 port pins**

The BGX13 offers a number LED control features. Pins can be configured to signify connection status and stream mode data activity.

Additionally, output pins can be controlled manually through commands.

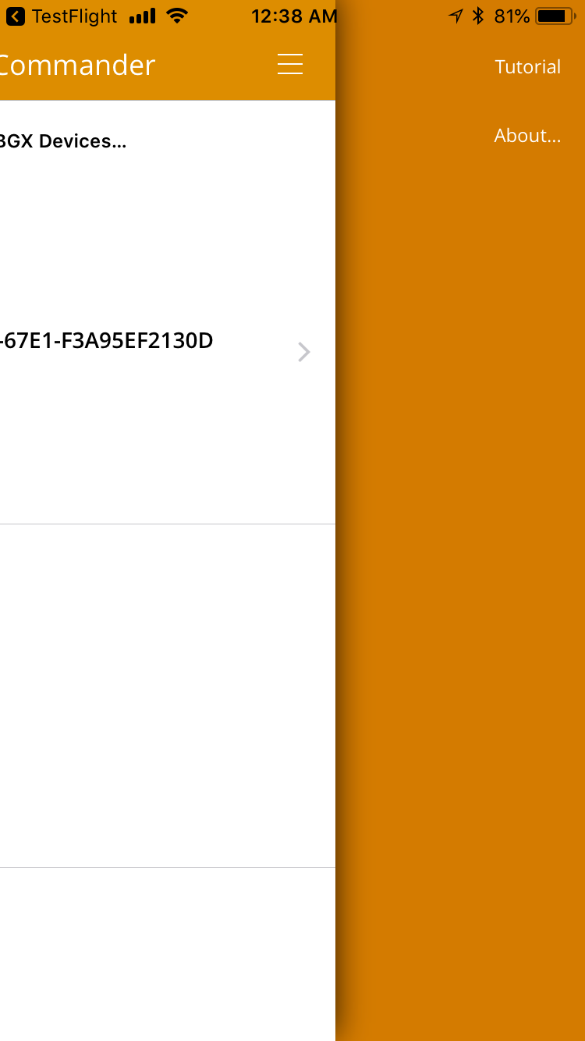
1. Tap ‘Disconnect’ in BGX Commander. Then select your BGX13 from the scan results list. Note that the LED on the BGX13 toggles as connection and disconnection occur.

Note that remote commands can be used to control general purpose I/O on the BGX13. An app can set the logic level of output pins and read the state of input pins through the BLE link.

### In-app Interactive Tutorial

The BGX commander app also includes an interactive tutorial that reveals how scanning, connection, and data transfer are made easy with the Xpress framework. We won’t step through this tutorial during this demo, but please try this feature out on your own, and consider highlighting it to customers interested in using the Xpress framework to simplify mobile app development.

To access the tutorial, disconnect from the BGX and click the ‘shelf’ in the upper right part of the screen. Options will slide in from the right, including a link to the ‘Tutorial’.



**BGX Commander demo review**

**What we just learned:**

* The BGX Commander iOS and Android example app is built on the Xpress framework, which simplifies BGX communication by abstracting low-level, core Bluetooth APIS.
* In addition to sending data in STREAM mode, the communications link enables remote command execution on the BGX from the mobile app.

**Why it’s useful:**

* Many BGX applications will need a mobile app, and many mobile app developers are not familiar with core Bluetooth APIs.
* With the Xpress framework, mobile app developers don’t have to become Bluetooth experts. Instead, they can call into the framework APIs for connection and communication.

# BGX-to-BGX demo

This section will demonstrate BGX-to-BGX communication. In this section, please partner up with a neighbor sitting next to you. We’ll use the BGX EXP board and the terminal app to communicate.

### Demo setup

In this demo, one person will configure their BGX13 to operate in a ‘central’ role, and the other person will operate in a peripheral role. Note that in all demos to this point, we’ve been operating the BGX in a peripheral role.

**BGX-to-BGX connection options**

In this example, the Bluetooth address of a peripheral is needed for connection by a central.

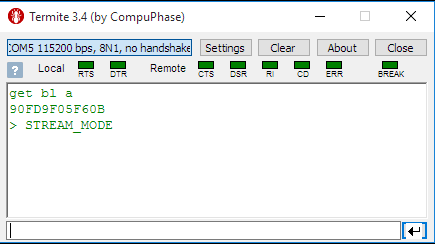
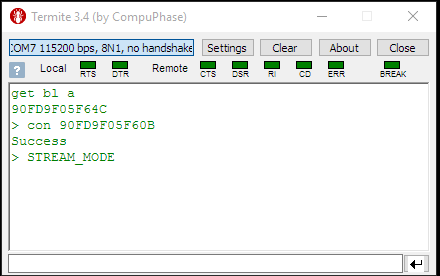
BGX also offers a ‘scan’ command with indexed results. The ‘con’ command can use the index value from scan results to connect to a peripheral.

The designated person operating as central needs to get the Bluetooth address of other BGX device. We read our Bluetooth addresses earlier in this demo, but if you need to read the address again, you can access it through Termite using the ‘get bl a’ command.

## Connecting and communicating

The ‘con’ command will switch a BGX from its default peripheral role to a central role and attempt to establish a secure connection with the requested peripheral.

For the device meant to be the central, execute ‘con <Bluetooth address of peripheral>’. In this example, the peripheral’s address is 90FD9F05F60B.



BGX in Central role

BGX in peripheral role

Note that ‘STREAM\_MODE’ is output from both central and peripheral upon successful connection.

1. Try to send data using Termite between the central and peripheral.

### 

Central, sent ‘Hello peripheral!’

Peripheral, sent ‘Hello central!’

**BGX-to-BGX demo review**

**What we just learned:**

* In addition to connecting to mobile apps, BGX13 modules can connect to each other
* In BGX-to-BGX communication, one device will be central, and that device is responsible for initiating the connection.
* BGX modules default to a peripheral state but switch to central with calls to ‘con’ or ‘scan’
* Once connection is established, the communications link is symmetrical, just as would be the case in a cable replacement design.

**Why it’s useful:**

* BGX to BGX operation supports wireless cable replacement opportunities
* A star network can be created with one central and multiple peripherals.
* Remote command execution is possible with BGX-to-BGX communication, which enables GPIO control of remote BGX devices as well as configuration adjustments.

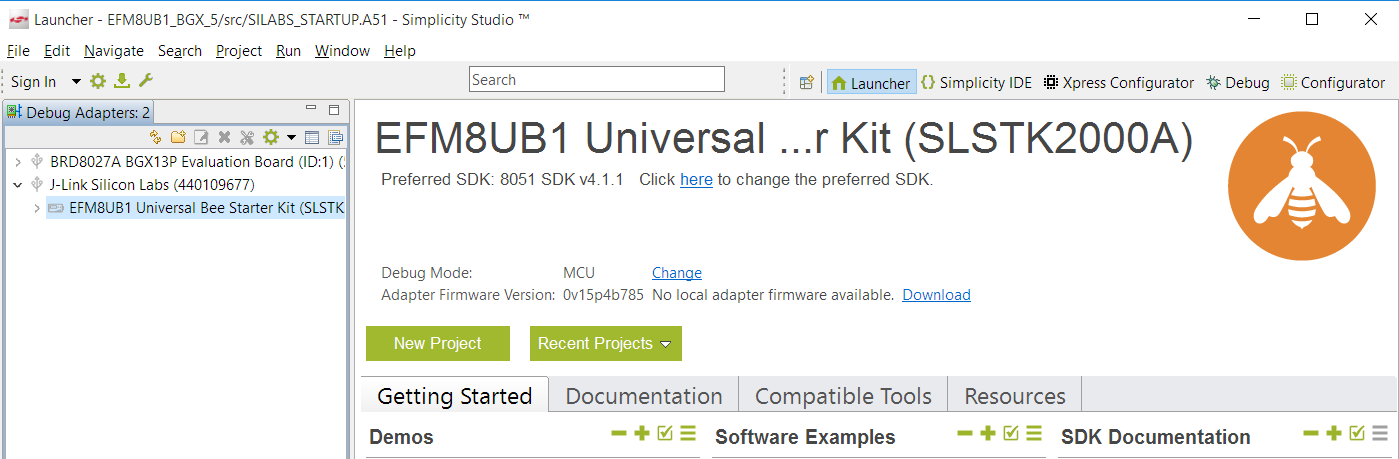
# Demoing with EFM8 STK

In this portion of the training, instead of using the Termite terminal app to simulate an embedded host, we’ll use an EFM8 STK. The EFM8UB1 STK has an EXP connector on one side of the board, and when the BGX EXP board is plugged into that connector, the EFM8UB1 on the board can access BGX UART and GPIO signals.

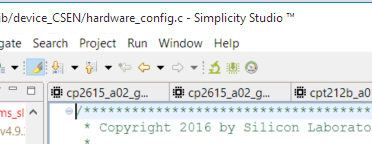
The source code for this example shows best practices when interfacing with the BGX13 and will be available in EFM8 SDK when the BGX13 launches.

### Flashing the EFM8 STK

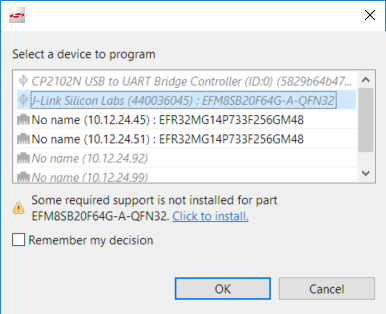
1. Disconnect the USB micro cable from the BGX EXP board.
2. Connect the BGX EXP board to the EFM8UB1 STK.
3. Connect mini USB cable to the EFMUB1 STK.
4. Run Simplicity Studio and wait for Studio to load.
5. In the launcher screen, choose ‘Simplicity IDE’.



1. Once the IDE loads, click on the Flash programming utility.

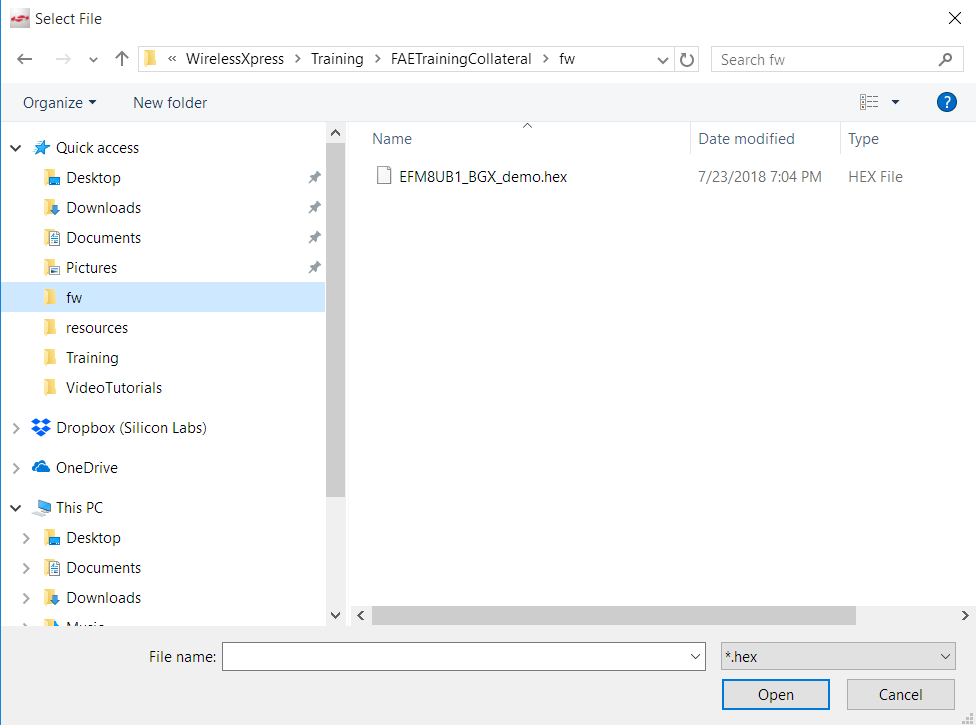


1. Choose the EFM8SB1 from the list of available devices to program.

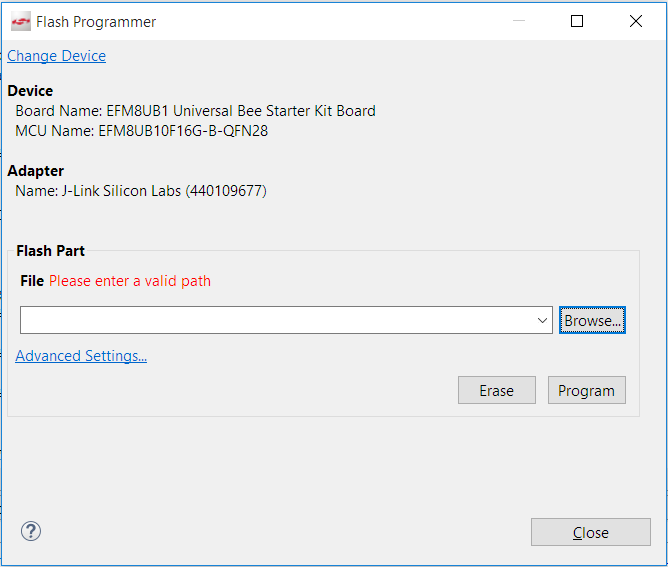


Note that Simplicity may need to install drivers to program the EFM8UB1 example. If you see this warning, choose ‘click to install’ to download drivers automatically.

1. Browse to find the hex image for the example.



1. Click ‘Program’.

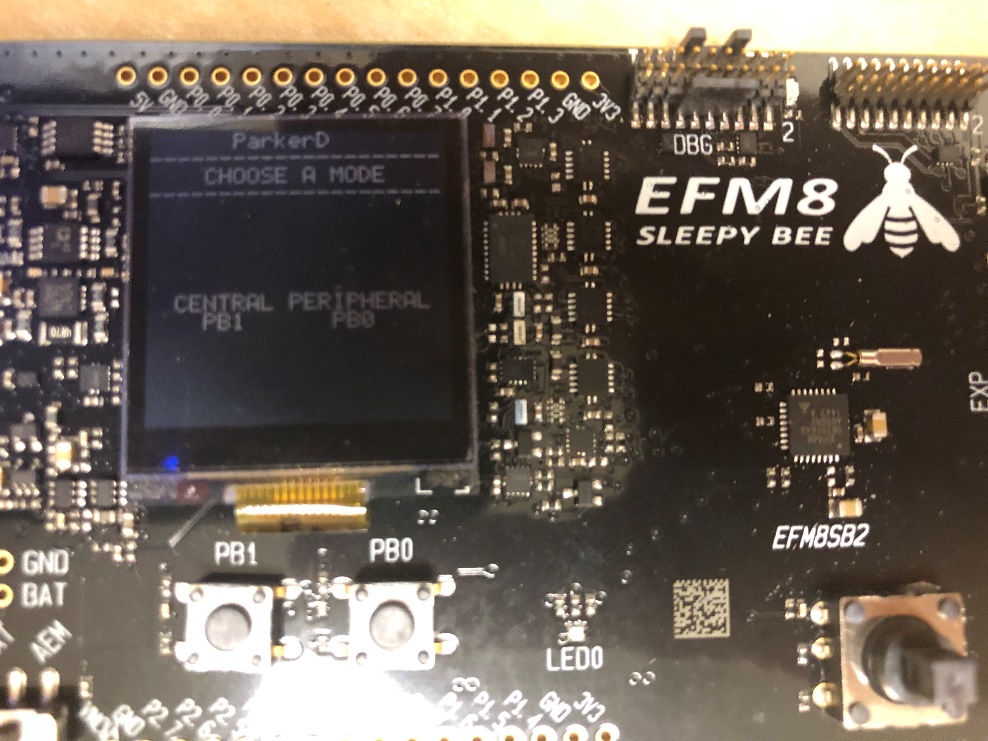


Note that you might see a ‘please enter valid path’ error, but ignore it and click ‘Program’ anyway.

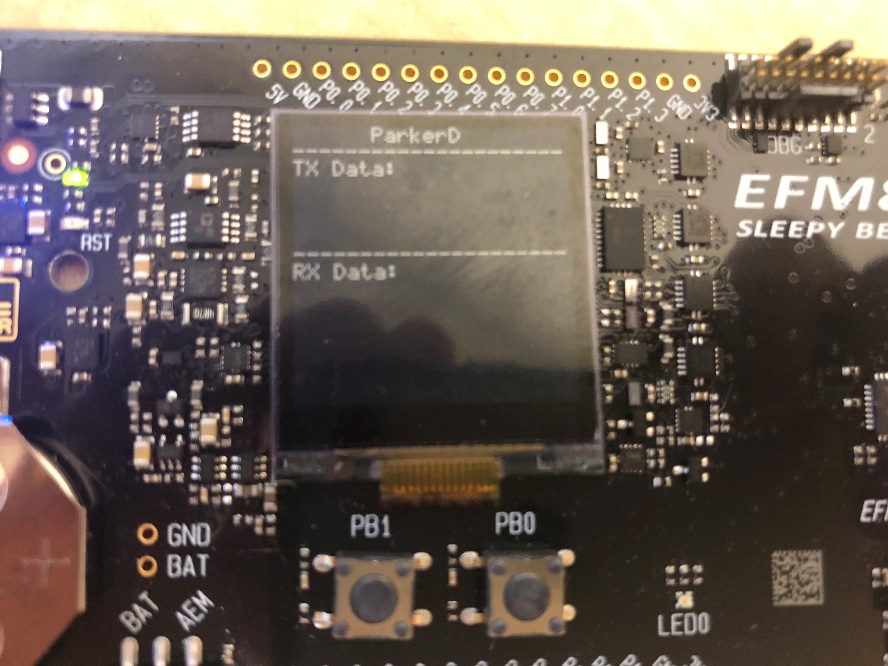
1. Close Simplicity Studio.
2. Remove the USB cable and then plug the cable back in to reset the system.

### Running the demo

By default, the STK puts the EXP board in a sleep state, and so it is not advertising. The startup screen on the demo looks as follows.

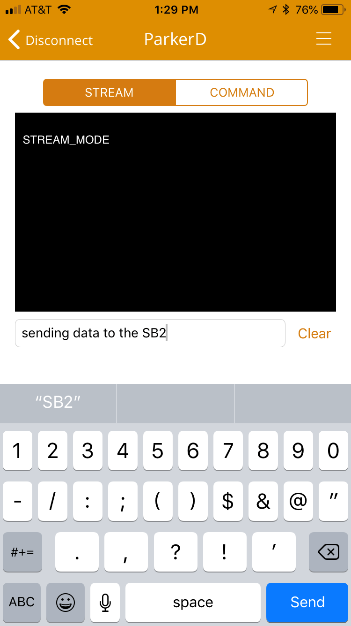


1. Click on the PB0 button to begin advertising as a peripheral. The example then displays a screen with a portion showing any transmitted data, and a portion showing received data.



Note that before making this choice, the BGX is in a sleep state. Choosing ‘peripheral’ wakes the device, which then automatically begins advertising.

1. In the BGX Commander mobile app, scan for the device and connect to it. Once in streaming mode, send a message to the BGX.



Note that data was received and displayed on the EFM8’s LCD.



Also, pressing either PB1 or PB0 will send a message through the BGX13 to the BGX command for display.

**EFM8 example demo review**

**What we just learned:**

* An EFM8 STK example included with the EFM8 SDK simulates an embedded host to the BGX13 EXP board
* This example demonstrates bi-directional communication between a mobile app and a BGX13.
* It can also be used to demonstrate BGX-to-BGX communication if central is selected instead of peripheral in the startup screen.

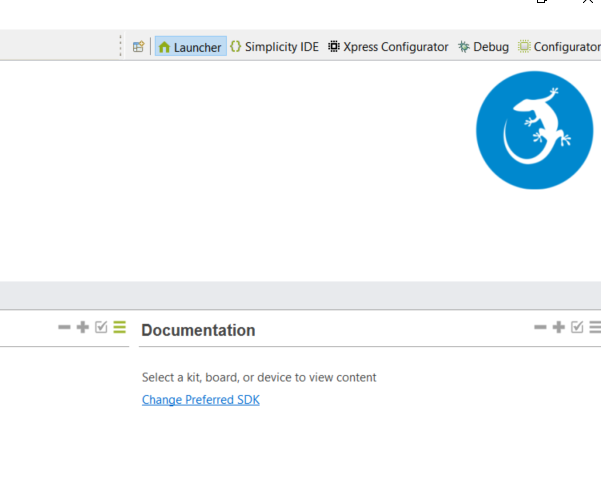
**Why it’s useful:**

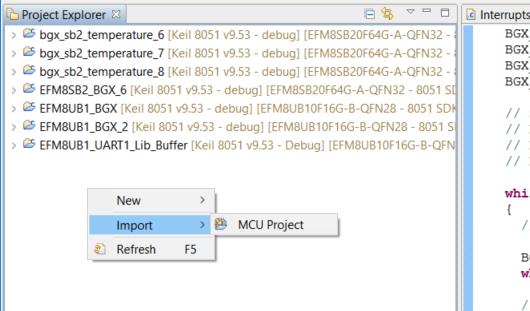
* The source code for this example illustrates best practices for BGX communication.
* The source code also shows how easy the host-to-BGX link is to implement, illustrating how low the requirements are for a host to use the BGX13 to add BLE functionality to an application.

# Editing the BGX temperature demo

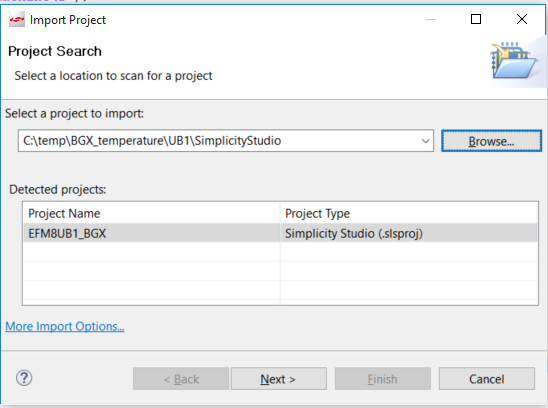
In this portion of the training, we’ll revisit the BGX temperature sensor example demoed during the presentation. We’ll open up the firmware, add some calls to BGX, and test performance with the BGX Commander app.

### Demo setup

1. Run Simplicity Studio.
2. Find ‘Simplicity IDE’ in the upper right corner and click to run. 
3. Once inside Simplicity IDE, right click on the Project Explorer window and choose ‘Import->MCU project…”



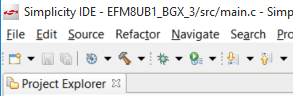
1. Click brows and navigate to the directory on your hard drive where the code examples from the Dropbox folder have been stored. Find the directory BGX\_temperature\UB1\SimplicityStudio.



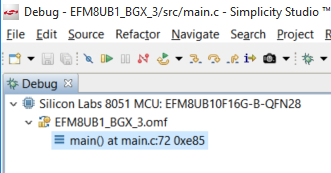
1. Click on ‘EFM8UB1\_BGX’ in the ‘Detected Projects’ box, then click next twice and ‘Finish’ to import the project.
2. In the project explorer window, navigate to the main.c file in the project in the src folder.



1. Connect the EFM8UB1 STK + BGX EXP board to your PC through the UB1’s USB port if it’s not already connected.
2. Build and download code by clicking here:



1. Run the code by clicking here:



1. Now run BGX commander on your phone. Scan for devices. Note: your device doesn’t show up because the code is turning off advertising. In the next sections, we’ll start editing code to enable BGX features.

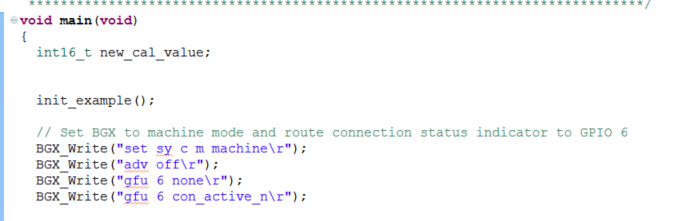
**Why your BGX may have appeared in scan results**

BGX13 was designed for ease-of-use. Out of reset, default behavior for the BGX is to begin advertising with the configuration stored in the device. For this reason, if you run the UB1 firmware, which toggles the BGX reset pin, and immediately click ‘Scan’ in BGX commander, your device may appear.

We’ll discuss this further in the next sections.

### Step 1 – Enabling advertising on a button click

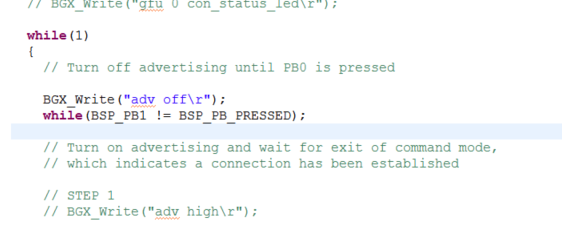
As we saw in the presentation earlier, a design goal of the firmware is to not advertise until a button is pressed. Out of reset, the BGX will attempt to advertise immediately. The UB1 project shuts off advertising as soon after reset as possible, in this line in main():



In that same configuration block, we also do the following:

* enable machine mode output from BGX, which is an optimized output style easier for embedded systems to parse
* Route the ‘con\_active\_n’ signal out BGX GPIO 6. This is a digital output from the BGX that signals logic high when no connection is active and logic low when a connection is active. As we’ll see, the firmware uses that signal to determine whether to output temperature data.

Right now, firmware waits for a button press on PB0, but does not turn on advertising after the button is pressed:

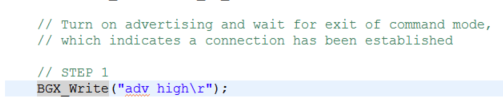


**Turn on advertising and after button press**

**Turn off advertising and wait for button press**

Let’s enable advertising by uncommenting the call to ‘adv high’.

1. Remove comments on the line BGX\_Write(“adv high\r”);



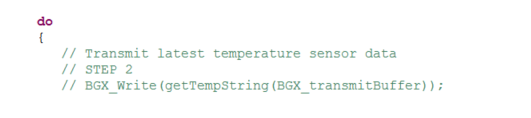
1. Build and download the project.
2. Run the project.
3. Go back to BGX Commander and run scan again.
4. Press PB0 on the UB1 STK.
5. Now note that your BGX device appears in the scan results.
6. Touch your BGX on the scan results to connect.
7. Once connected, note that no temperature data is transmitting. We’ll add that feature in the next step.

### Step 2: Transmitting temperature data

Once connected, the firmware enters a do while() loop that outputs a temperature measurement and then delays for 1 second before repeating. This loop continues until the MODE\_PIN, which is tied to con\_status\_n, changes state to indicate a disconnected BLE link:

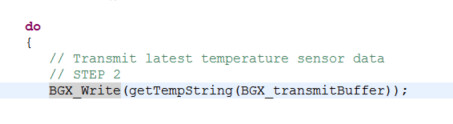


Right now, no temperature data is outputting because the line to read the on-chip temperature sensor and output results is commented out:

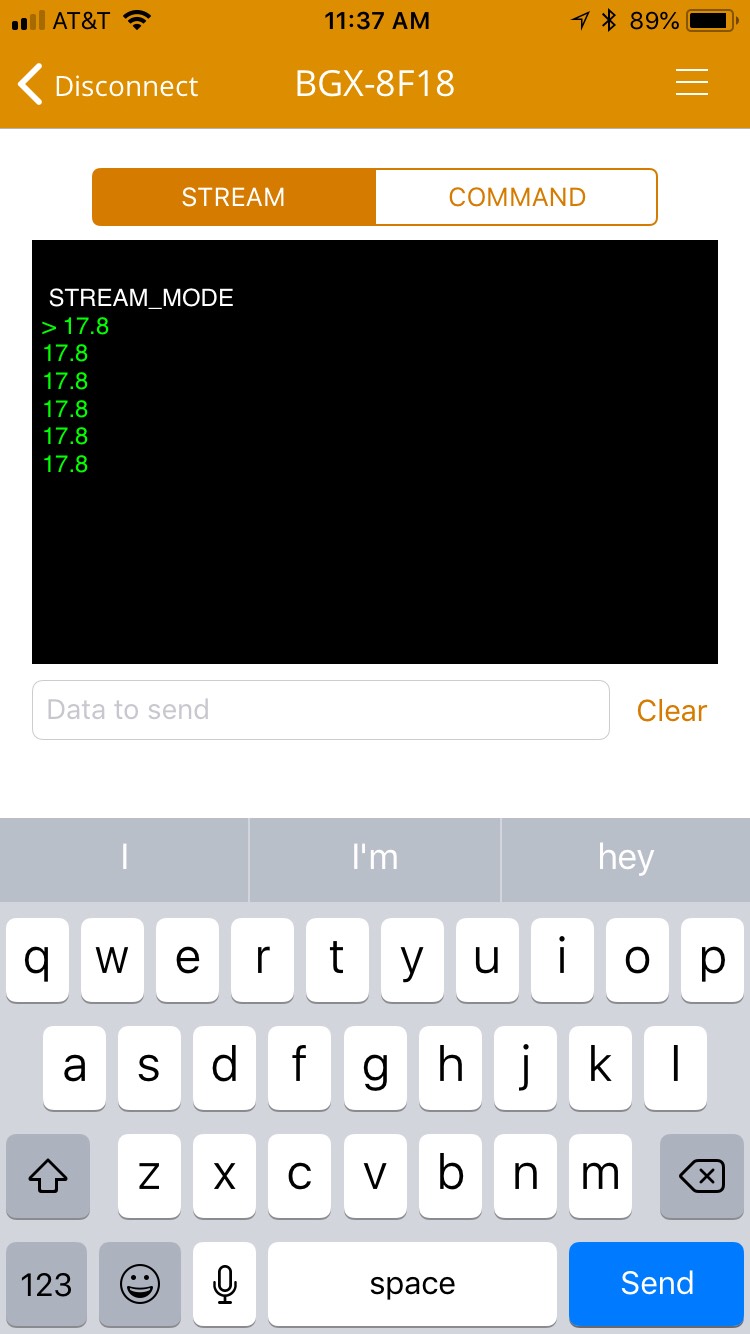


Let’s add that line into the build and test results.

1. Uncomment the line that begins “BGX\_Write(getTemp…”:



1. Build, download, and run code.
2. Open BGX commander. Press PB0 on the board to enable advertising, and connect to the BGX device. Now you’ll see temperature data streaming from the UB1:



### Step 3

Now let’s add something to the ‘product’ that was not demoed during the presentation. BGX13 offers a set of configurable LED outputs to visually indicate connection status and other information. The duty cycles for these outputs are configurable and set using the ‘sy i s’ variable.

‘sy i s’ takes an 8-character argument of the form AABBCCDD, where the components are defined as follows:

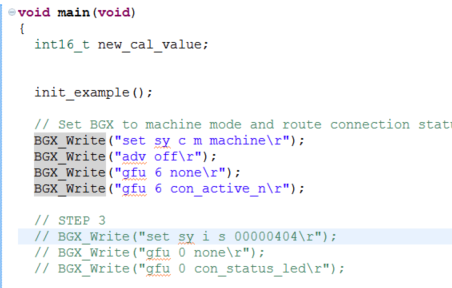
* AA: LED on duration when not connected
* BB: LED off duration when not connected
* CC: LED on duration when connected
* DD: LED off duration when connected

Each value is in units of 100 ms. So, a setting of “sy i s 04080100” would mean that the LED blink rate is:

* On for 400 ms then off for 800 ms when not connected
* Remain always on for when connected, because the off duration is set to 0.

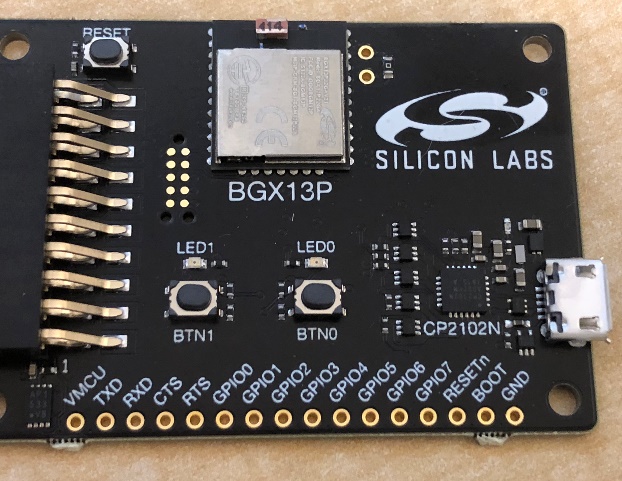
Let’s route the LED signal ‘con\_status\_led’ to one of the LEDs on the BGX EXP board.

1. Toward the top of the main() function, find the configuration section that configures con\_status\_led and uncomment the code.



This code sets the LED to remain off when not connected, and blink with an 800 ms period and 50% duty cycle when connected. The signal gets routed to LED0 on the BGX13P board.

1. Build, download, and run code.
2. Go back to BGX Commander. Click PB0 to enable advertising and connect to your BGX13P board while watching LED0 on the BGX13P.



Note that when connection completes successfully, the LED starts blinking.

We just accomplished in three lines of code what would take multiple development steps on unprogrammed wireless modules:

* Set a port pin as LED output
* Configure a timer to a update at a defined rate
* Enable a timer interrupt
* Write a state machine inside that interrupt to manage LED duty cycles as a function of connection state

**BGX temperature example demo review**

**What we just learned:**

* In less than 10 top-level lines of code, we can connect across BLE and transmit product-specific data to a mobile app
* The BGX offers functionality like LED control to simplify development of an expected BLE application design feature

**Why it’s useful:**

* This source code highlights just how easy it can be to connect and communicate across BLE in a product
* Note that this code is still more complex than is absolutely required. Remember that we saved the BGX name with the ‘save’ feature. All configuration we’re doing at runtime could be committed to nonvolatile memory. With configuration stored, only the calls to communicate and enable/disable advertising based on button press remain.
* Pushing toward simplification even further, if we had used the UFU command set, which can bind BGX GPIO pin state to command execution, we could tie advertising enable/disable to a BGX pin’s logic state. With this feature, the lines that enable/disable advertising could be replaced with a pin toggle.