Hardware Breakout

User Guide: Bluetooth Low Energy BoosterPack – V3.0

Description

The Bluetooth Low Energy (BLE) BoosterPack has been designed to make it as easy as possible to use BLE in your LaunchPad based projects. The BLE BoosterPack is built around the easy to use BLE112 module from Bluegiga. The BoosterPack can be powered by (and can charge) a lithium battery, making the entire system portable. It also has an ultra-high efficiency switching regulator that minimizes the power consumption of the system. The most important feature of the BLE BoosterPack is that no prior knowledge BLE is needed, the device is configured to be a simple UART pass-through. Using the example code provided for both Android and Python, you can get started with your BLE application without knowing needing to be familiar with BLE services or BLE characteristics.

The whole point of using BLE is to minimize the power consumption of portable projects. This BLE Booster has been built from the ground up for low-power operation. There are no status LEDs that burn through power, the BLE112 module can be reset and woken up by the MSP430, and the switching regulator has an efficiency of up to 95%. Even the voltage divider used to measure the battery voltage can be disabled.

Open source demo and example code is provided for the BLE BoosterPack on GitHub. The BLE BoosterPack's example firmware will work right out of the box, make getting started with Bluetooth Low Energy as easy and painless as possible.

Bluetooth Low Energy BoosterPack Pinout

V3.0 – Hardware Breakout Store

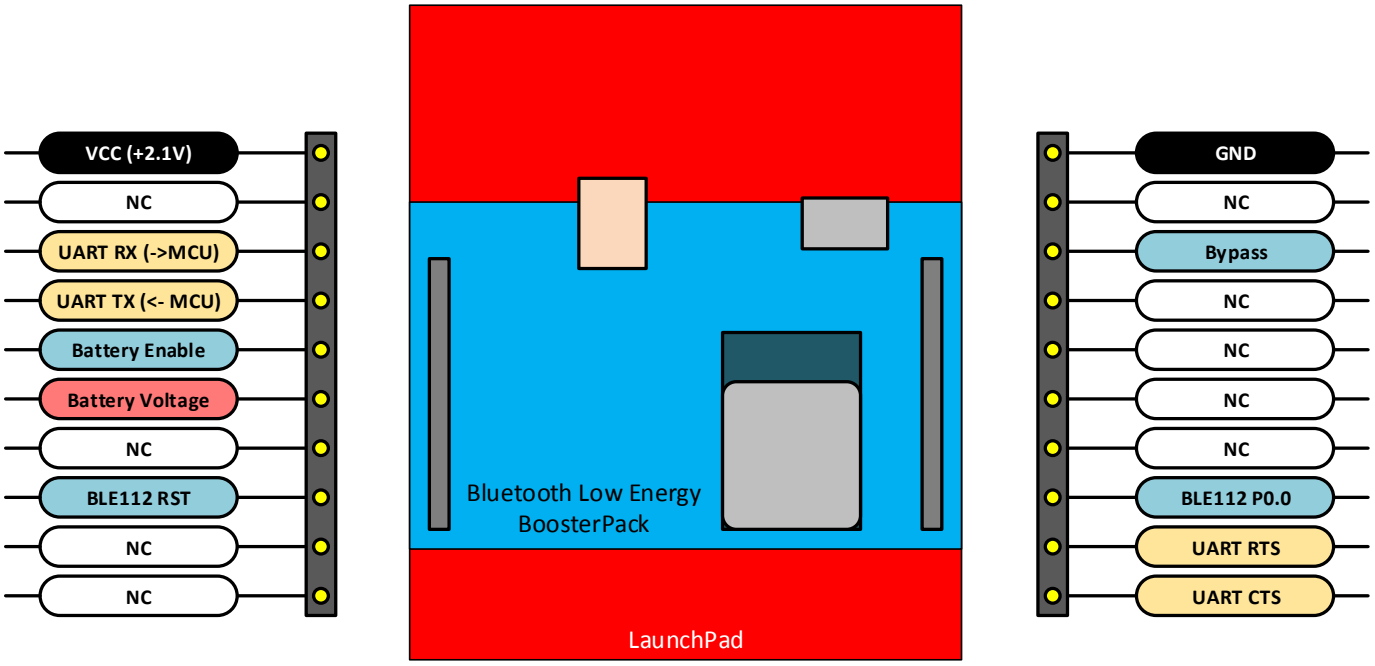


Table of Contents

Features

- Based on the BLE112 Bluetooth Low Energy Module
- BLE UART Pass-through firmware preprogrammed with optional flow control (CTS/RTS)
- Open source software examples
    - Example code for the Value Line LaunchPad and the 5529 LaunchPad
    - Firmware examples for the BLE112
    - Fully functional Android application (for supported devices)
    - Example Python script
- Lithium battery micro USB charger (MCP73831)
- 2.1V switching buck regulator (TPS62730) up to 95% efficiency
- Firmware update using USB DFU, removing the need for the CC Debugger
- Included CC Debugger programming header for easy interfacing
- Easily solderable parts – 0805 passives and an LDO alternative to the switching regulator
- Bluegiga 2.1 SDK for easy firmware development
- Micro USB interface so that the BoosterPack can act as a USB dongle
- Jumpers for enabling or disabling every I/O pin
- Adheres to the TI BoosterPack standard

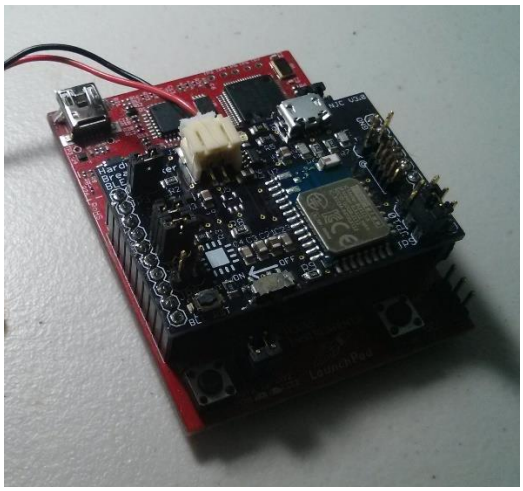
PCB Options

Note: Lithium batteries are not included in any option. These need to be purchased separately.

1. PCB Only – $10
    - Being so easy to assemble by hand, this is the cheapest way to get started with the BLE BoosterPack. All the information needed to purchase the correct parts and assemble the PCB is on github.
2. BLE Only – $40
    - If your project does not require battery power, this is best option. This board does not contain any power circuitry or connectors. This means there is no lithium battery charger, and no voltage regulator. The PCB is assembled with all of the parts needed to use and program the BLE module.
3. Complete Board with Linear Regulator – $50

- This is the best option to choose when a switching regulator is overkill. This is a full featured board, including the lithium battery connector and charger circuitry. Rather than using switching regulator, a linear regulator is used to reduce the overall cost.

4. Complete board with Switching Regulator – $65
    - Do you want all the bells and whistles? This is the option for you. Completely assembled and ready to go.



Useful links

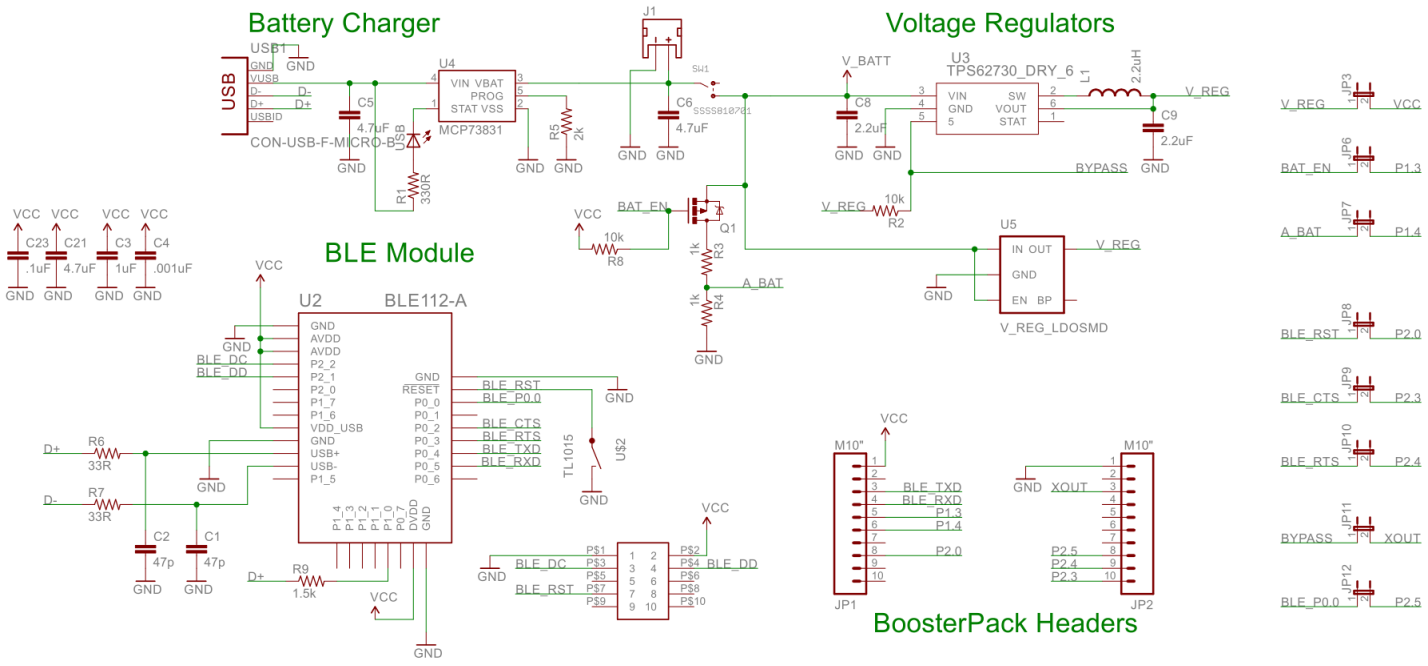
- Bluegiga's BLE112 product page
    - http://bluegiga.com/en-US/products/bluetooth-4.0-modules/ble112-bluetooth--smart-module/
- Bluegiga resources for BLE
    - https://bluegiga.zendesk.com/categories/20128526-Knowledgebase
- TSP62730 product page and datasheet
    - http://www.ti.com/product/tps62730
    - http://www.ti.com/lit/ds/symlink/tps62730.pdf
- BLED112 USB dongle product page
    - http://www.bluegiga.com/en-US/products/bluetooth-4.0-modules/bled112-bluetooth-smart-dongle/
- Programming the BLE112
    - http://blog.bluetooth-smart.com/2012/09/11/programming-the-ble112-with-c-code-using-iar/
- BLE on Wikipedia
    - http://en.wikipedia.org/wiki/Bluetooth_low_energy
- Basic overview of BLE
    - http://www.element14.com/community/groups/wireless/blog/2013/08/23/bluetooth-low-energy
- Basic BLE information and Arduino shield example
    - http://www.mkroll.mobi/?page_id=53
- BLE and Windows 8.1
    - http://www.ingenuitymicro.com/justins-blog/2013/7/11/bluetooth-le-and-windows-81.aspx
- Guide for a different BLE112 board
    - http://www.blelabs.com/blog/bl_t0003-ble112-protostick-connecting-characteristics-handles-services-and-notifications-under-windows/
- If you know of any other useful links, please send an email to store@hardwarebreakout.com.

Getting Started

Note: You must have either a BLED112 USB dongle, a Bluetooth Low Energy capable Android phone running android 4.3 or higher, or two BLE BoosterPacks for the demo code to work. A second BLE BoosterPack can be used as a USB dongle if you do not have the BLED112 USD dongle from Bluegiga.

1. For the blank boards, assemble using the BOM and the PCB layout file provided on github.
2. Plug the BoosterPack into the LaunchPad and hookup the battery to the BoosterPack (optional)
3. Load the example MSP430 code onto the LaunchPad using Code Composer Studio (CCS) from Texas Instruments.
    a. See this link for more information on programming the LaunchPad: http://www.msp430launchpad.com/2010/07/my-version-of-getting-started.html
4. Connect to the BLE112 device on the BoosterPack using either the Python example code or the Android application.
5. Play with the demo code! Press the button on the LaunchPad, measure the VCC voltage, the temperature, and the battery voltage.

Schematic

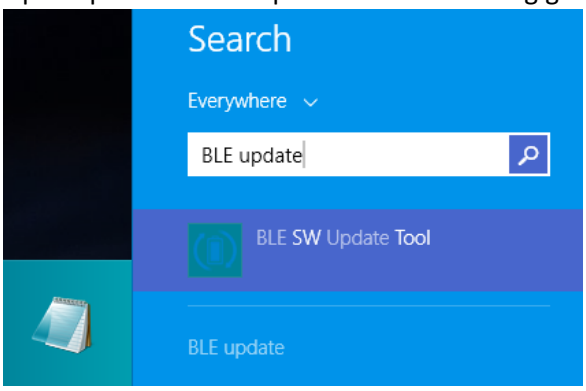


Programming the BLE112 with the CC Debugger

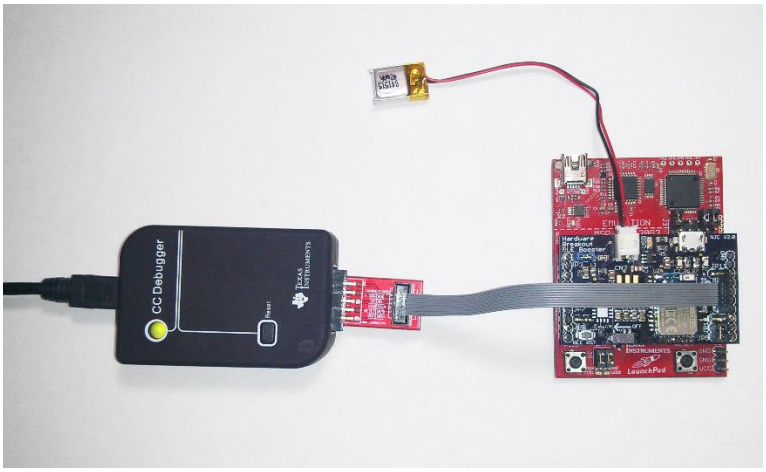
Note: The VCC pin on the CC Debugger is not connected to the BoosterPack VCC pin, so that the chance of providing power from two sources is reduced. In order for the device to be programmed via the CC Debugger, the BoosterPack will need to be powered either from the LaunchPad or from the on board 2.1 V regulator.

1. Hook up the CC Debugger to the BoosterPack with the correct orientation. Pin one is labeled with a bull's-eye on the BoosterPack.

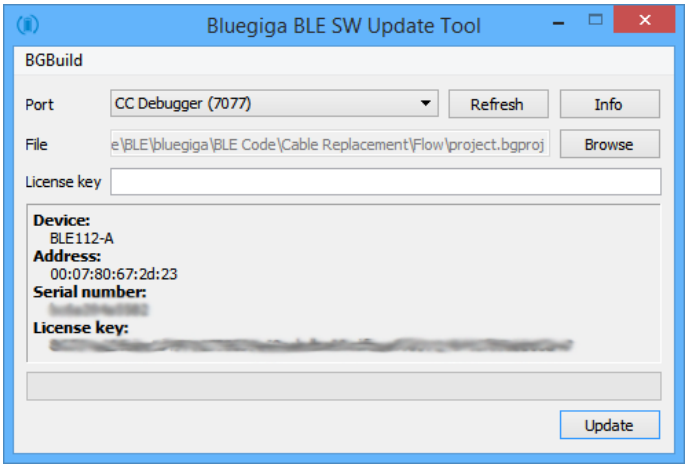
2. Open up the BLE SW Update Tool from Bluegiga



3. Ensure that the BoosterPack's power is turned on
4. Press the Reset button on the CC Debugger. The LED on the CC Debugger should now be green.



5. Check that the device is recognized correctly by pressing Info, then select Update to update the firmware.



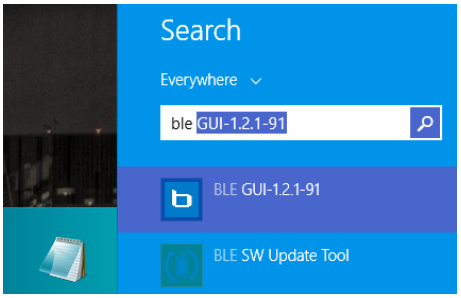
Programming the BLE112 via USB DFU

Note: If you have the BLED112 (BLE112 USB Dongle), it is recommended that you disconnect it from your computer.

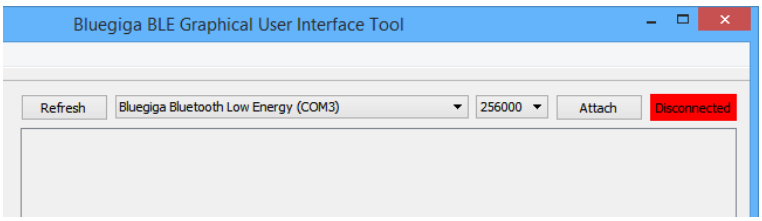
Note: This is only possible if USB is enabled on the BLE112 and the endpoint is set to "api". This is done in the hardware.xml file.

<usb enable="true" endpoint="api" />

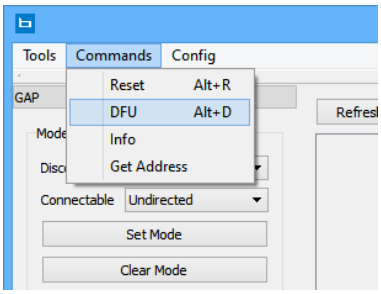
1. Make sure that the Bluetooth Smart Software and SDK (currently on v.1.2) is installed.
2. Open the Bluegiga BLE Graphical User Interface Tool



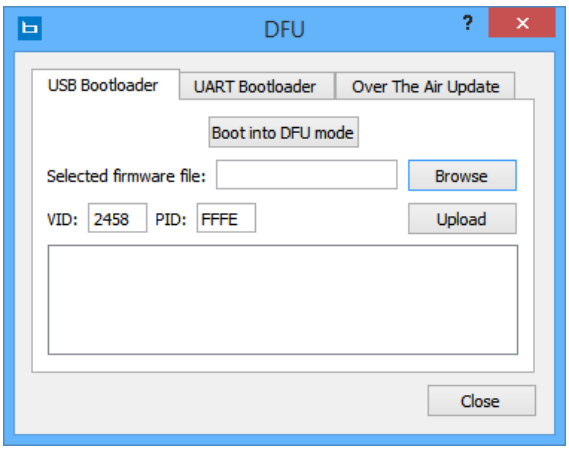
3. In the main GUI, select the COM port for your BLE112 module and click Attach. You may need to click Refresh first.



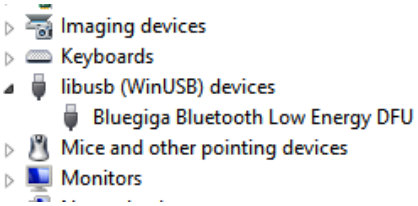
4. Select DFU under the command Commands menu.



5. Select “Boot into DFU mode”.



6. The BLE112 will reboot and show up in your device manager as a new device called Bluegiga Bluetooth Low Energy DFU.



7. Then select hex file that you would like to load onto the BLE112 using Browse.

8. Finally, click upload. The procedure should complete with no errors.

Test Code

The test code provided for the MSP430G2553 and the MSP430F5529 is designed to teach you how to utilize the BLE112 module. It is easily expandable and should be easy to follow. The test code contain the following demos for use with the example Python script or the example Android app (host):

- LED and Button Demo – The host can turn on and off the LED on the LaunchPad. A button press on the LaunchPad's can be recognized and counted by the host. This example demonstrates how basic digital input and output operations can be performed over BLE.
- Device Information Demo – The MSP430 can be prompted by the host to send the current VCC voltage, the on-board temperature, and the battery voltage (if installed) to the host. This example demonstrates how to read an analog value from an internal sensor or external pin via BLE.
- Terminal Demo – The MSP430 can be prompted to send multiple strings to the host, allowing for terminal like programs on the host to be tested. This example is for applications which would benefit from being able to send and receive arbitrary data from the host.

Android Application and Python Code

The example Android application is written in Eclipse using the Android ADT Bundle for windows. This application can be installed on your device manually, rather than from the Android Play Store. The application requires Android 4.3 or greater and an Android phone that contains Bluetooth Low Energy. The application was tested on a Nexus 4 running both Android 4.3 and 4.4.

The example Python code requires either the BLED112 USB dongle from Bluegiga or a second BLE BoosterPack. The BLE112 enumerates on the PC as a virtual COM port for the BGAPI data. The Python code includes a Python library for using the BGAPI. While not every BGAPI function is implemented, all of the ones which are needed for typical operation are.

Creating a New Android Fragment

Note: This is a brief overview of the steps required to modify the Android app to add a custom fragment, not a complete tutorial.

Note: In essence, a fragment is a tab in the Android app. For example, the "Terminal Demo" tab.

1. Create and design a new layout file – This dictates what the fragment will look like. Add any buttons and indicators you may need.
2. Add the new fragment to the navigation activity "NavigationActivity.java" – This allows the fragment to be opened within the app. Note the order in which the fragments are listed.
3. Add the new fragment title the string file under the "demo_titles_array" – The string file is under res/values. Be sure that the title is the same as the string array in the navigation activity.
4. Create the fragment's java file (e.g. "LedButtonFragment.java") – This is the java file where all the action takes place. You can copy an existing fragment, but you have to be sure to handle all events and modify the layout resource.