BLUETOOTH® SMART SOFTWARE

SPP-OVER-BLE APPLICATION NOTE

Friday, 15 November 2013

Version 1.0

bluegiga

VERSION HISTORY

Version	Comment
1.0	Initial release

TABLE OF CONTENTS

1 Introduction

This application note discusses how to prepare a *Bluetooth* Smart SPP-over-BLE device using Bluegiga's *Bluetooth* Smart SDK, for use with a DKBLE112 hardware evaluation board, DKBLE113 hardware evaluation board, BLED112 USB dongle, or other BLE11x-based hardware. This service is based on our proprietary "Cable Replacement" service and operates on the same principles, but it is technically different and will not work with that service due to the changes required for various features.

Note that this SPP-over-BLE profile is a Bluegiga proprietary profile and is not standardized by the *Bluetooth* SIG.

The Bluegiga SPP-over-BLE profile and projects are built to provide the simplest possible method to connect two of our modules together so that you can send arbitrary data back and forth, much like using SPP with a classic *Bluetooth* device. The BLE protocol is not built to allow this kind of data transfer in a very efficient way, due to the very small packet payload size (20 bytes max) and the relatively large delays required between packets (the BLE radio can only be on for less than 25% of the time even in the most aggressive transmit configuration). However, it is possible using normal *Bluetooth* Smart GATT operations to appear as though you have an open, transparent data pipe between each end of the connection.

Key features of the SPP-over-BLE project include:

- Transparent serial data connection
- AT-style command interface
- Act as either slave or master with the same firmware
- Configuration options stored in flash, persist across power cycles
- Automatic connection with matching device based on configurable ID
- Full source code available
- Pre-built binaries for BLE112, BLE113, and BLED112
- Reboot into DFU mode with a command (safe for BLED112)

Notable limitations of this implementation include:

- **Hardware flow control (RTS/CTS handshaking) is *required* for reliable data transmission**. While it is possible to use it without flow control, you ***will*** lose data unless you send only very small blocks of data (a few bytes) with long gaps between them (hundreds of milliseconds). With a proper flow control implementation, there should not be any lost data. This has been thoroughly tested; for details on flow control when used with our module, please see this Knowledge Base article:
    - https://bluegiga.zendesk.com/entries/23143152--REFERENCE-Using-or-bypassing-flow-control-with-UART-communication

- **Maximum throughput is approximately 1 kByte/sec**. The *Bluetooth* Low Energy protocol is not designed for high throughput, especially when reliable data transfer is important. Due to the small payload side of BLE packets (20 bytes maximum) and the interval between each packet (typically 10% duty cycle or less), it is not possible to obtain fast data rates. For details on why BLE throughput is limited in this way, please see the following Knowledge Base articles:
    - https://bluegiga.zendesk.com/entries/24646818-Throughput-with-Bluetooth-Smart-technology
    - https://bluegiga.zendesk.com/entries/22400867--HOW-TO-Maximize-throughput-with-the-BLE112-BLED112

- **This SPP-over-BLE firmware can only communicate with other Bluegiga hardware running the same firmware or any BLE-enabled device configured to use the same GATT structure and behaviour.** It is not possible to use this firmware to connect with just any random BLE device (e.g. a heart rate sensor), due to the way GATT works. For details on GATT structure and behaviour, please see the following Knowledge Base article:
    - https://bluegiga.zendesk.com/entries/25053373--REFERENCE-BLE-master-slave-GATT-client-server-and-data-RX-TX-basics

- **Sleep mode is not currently implemented in v1.0, which means consumption of the module will be roughly at least 8mA at all times.** This will be changed in future releases, and will require the use of a wake-up pin during UART transmissions to the module. For details on low-power modes and wake-up pin usage, please see the following Knowledge Base article:
    - https://bluegiga.zendesk.com/entries/23173106--REFERENCE-BLE11x-low-power-and-sleep-modes

2 What is *Bluetooth* Smart technology?

Bluetooth low energy (*Bluetooth* 4.0) is a new, open standard developed by the *Bluetooth* SIG. It's targeted to address the needs of new modern wireless applications such as ultra-low power consumption, fast connection times, reliability and security. *Bluetooth* low energy consumes 10-20 times less power and is able to begin transmitting data 50 times quicker than classical *Bluetooth* solutions.

Bluetooth low energy is designed for new emerging applications and markets, but it still embraces the very same benefits we already know from the classic well-established *Bluetooth* technology:

- **Robustness and reliability** - The adaptive frequency hopping technology used by *Bluetooth* low energy allows the device to quickly hop within a wide frequency band, not just to reduce interference but also to identify crowded frequencies and avoid them. On addition to broadcasting *Bluetooth* low energy also provides a reliable, connection oriented way of transmitting data.
- **Security** - Data privacy and integrity is always a concern is wireless, mission critical applications. Therefore *Bluetooth* low energy technology is designed to incorporate high level of security including authentication, authorization, encryption and man-in-the-middle protection.
- **Interoperability** - *Bluetooth* low energy technology is an open standard maintained and developed by the *Bluetooth* SIG. Strong qualification and interoperability testing processes are included in the development of technology so that wireless device manufacturers can enjoy the benefit of many solution providers and consumers can feel confident that equipment will communicate with other devices regardless of manufacturer.
- **Global availability** - Based on the open, license free 2.4GHz frequency band, *Bluetooth* low energy technology can be used in world wide applications.

There are two types of *Bluetooth* 4.0 devices:

- ***Bluetooth* 4.0 single-mode** devices that only support *Bluetooth* low energy and are optimized for low-power, low-cost and small size solutions.
- ***Bluetooth* 4.0 dual-mode** devices that support *Bluetooth* low energy as well as classic *Bluetooth* technologies and are interoperable with all the previous *Bluetooth* specification versions.

Key features of *Bluetooth* low energy wireless technology include:

- Ultra-low peak, average and idle mode power consumption
- Ability to run for years on standard, coin-cell batteries
- Low cost
- Multi-vendor interoperability
- Enhanced range

Bluetooth low energy is also meant for markets and applications, such as:

- Automotive
- Consumer electronics
- Smart energy
- Entertainment
- Home automation
- Security & proximity
- Sports & fitness

3 SPP-over-BLE profile

3.1 Description

The Bluegiga SPP-over-BLE profile enables two devices to connect and exchange data. The profile provides a reliable and transparent way to send data from each device's UART (or USB) interface over *Bluetooth* low energy connection to a collector device. Both devices may implement both the sensor and collector functionality to better allow for automated, transparent connections.

The SPP-over-BLE profile defines two roles:

1. **SPP DevA**

The SPP DevA device receives data from its UART (or USB) interface and transparently exposes it via the SPP-over-BLE Service. DevA can also receive data from the SPP DevB device and transparently route the data to the same UART (or USB) interface. The SPP DevA is the GATT server.

2. **SPP DevB**

The SPP DevB device accesses the information exposed by SPP DevA and outputs the received data. The SPP DevB is the GATT client, and can be either another Bluegiga module/dongle running the same SPP-over-BLE firmware or else another device such as a mobile phone, tablet, or PC that has been configured to operate within the structure and required behavior of the SPP-over-BLE profile.

The source code and pre-compiled firmware provided to run on our modules (BLE112, BLE113, BLED112) is built to support either the GATT server (DevA) or the GATT client (DevB) role, and can operate as desired based on the commands given over the UART or USB interface.

The figure below shows the relationship of these two roles.

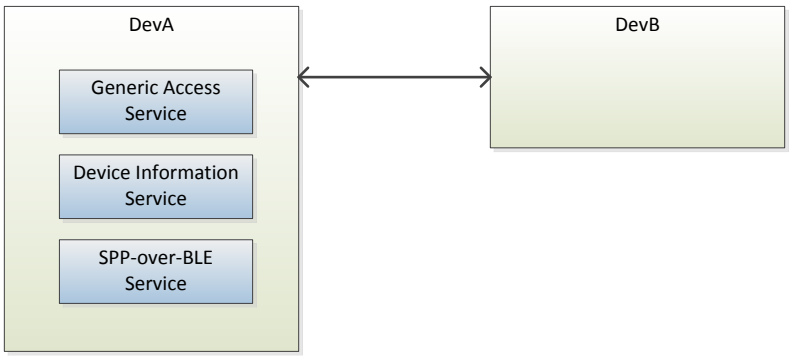

Figure 1: SPP-over-BLE Profile Roles

3.2 GATT Server: Service requirements

The table below describes the service requirements.

Service	UUID	SPP-over-BLE DevA
Generic Access Service	1800	Mandatory
SPP-over-BLE Service	f6ec37db-bda1-46ec-a43a-6d86de88561d	Mandatory
DFU Reboot Service	4f3edfe7-6f17-4f87-b2ee-ea2cdac0dd02	Optional

Table 1: Service requirements

3.3 GATT Server: Attribute requirements

The table below describes the structure and requirements for the attributes in the **SPP-over-BLE service**:

Characteristic	UUID	Length	Type	Support	Security	Properties
SPP-over-BLE Configuration	7fb03169-d527-4ca8-b9f0-c6f4a887c2fa	Variable (max 20B)	User Data	Mandatory	Optional	• Write • Indicate
SPP-over-BLE Data	af20fbac-2518-4998-9af7-af42540731b3	Variable (max 20B)	User Data	Mandatory	Optional	• Indicate • Write

Table 2: SPP-over-BLE Service structure

- **UUID** is a manufacturer proprietary 128-bit ID for each characteristic.
- Each characteristic's **length** is variable and has a maximum length of 20 bytes, since that is the maximum payload allowed for indicated values.
- By default security is not required, but it can be implemented if the application requires security .
- By default each characteristic has **indicate** and **write** properties value.

3.4 Other Requirements

The DevA device includes the SPP-over-BLE Service UUID in the advertisement data. This allows DevB devices to filter and recognize only compatible DevA devices during the scanning process.

The DevA device should include the device name in the advertisement or scan response data.

The DevA device may support the **write** property for the device's local name characteristic, allowing DevB to edit the friendly name. **Support for this is not currently built into the SPP-over-BLE firmware.**

4 Preparing SPP-over-BLE devices

The chapter contains step by step instructions how to flash SPP-over-BLE firmware with Bluegiga's *Bluetooth* 4.0 Software Development Kit. The chapter is split into following steps:

1. Installing the tools
2. Compiling the firmware using the latest SDK
3. Installing the firmware into BLE112, BLE113, or BLED112 hardware

4.1 Installing the Tools

1. Download the latest install the Bluegiga *Bluetooth* Smart SDK from the Bluegiga web site
2. Run the executable installer
3. Follow the on-screen instructions and install the SDK to the desired directory
4. Perform a Full Installation (BLE SDK and BLE Update)

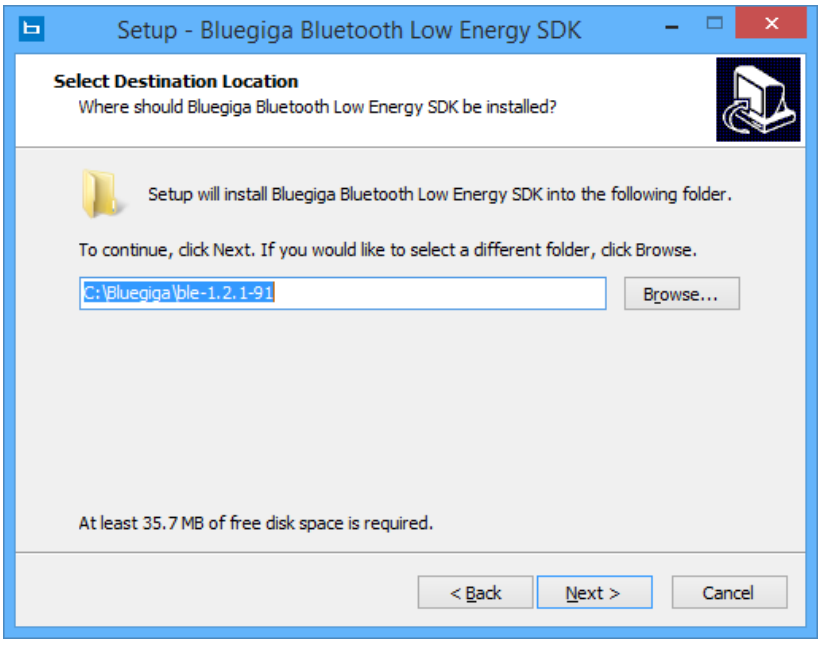

Figure 2: Installing Bluegiga *Bluetooth* Smart SDK

4.2 Compiling and Installing the Firmware

The SPP-over-BLE firmware can be downloaded from Bluegiga's online example project area. The collection of project files contains the full source code for the SPP-over-BLE firmware, including predefined hardware configurations to support the following deployment scenarios:

- BLE112 or BLE113 using UART1/Alt1
- BLED112 using USB

If you wish to use a different configuration, e.g. UART0 in either "alternate" setting or UART1/Alt2, then you will need to modify both the **hardware.xml** file and the BGScript source file accordingly. Note that **modifications to the BGScript source file are not trivial** due to the full utilization of specific GPIO pins, built around the assumption that P0_2/3/4/5 will be used for UART1/Alt1, and other pins will be free. Changing to a different UART configuration will require reworking the GPIO usage throughout the code.

The UART1/Alt1 configuration was chosen as the standard option because it works out of the box with the DKBLE112 devkit and DKBLE113 devkits.

This application note does not go into detail on precisely how the SPP-over-BLE application is built, due to its relative complexity.

4.2.1 Using BLE Update tool

In order to use the SPP-over-BLE firmware, you need to compile the hardware settings, GATT database, and BGScript code into a firmware binary file. The easiest way to do this is with the **BLE Update tool** that can be used to compile the project and install the firmware to a Bluetooth Smart Module using the CC debugger device. It is also possible to compile it manually using the "**bgbuild.exe**" console application, and then flash the resulting firmware onto a dongle using the "**dfutool.exe**" console application (or the USB DFU interface found in BLEGUI application).

To compile and install the project onto a BLE112 or BLE113 module with BLE Update:

1. Connect the CC debugger to the PC via USB
2. Connect the CC debugger to the debug interface on the BLE112 or BLE113
3. Press the button on CC debugger and make sure the LED turns green
4. Start the **BLE Update** tool
5. Make sure the CC debugger is shown in the **Port** drop down list
6. Use Browse to locate your **project** file (for example **project_ble112_u1a1.bgproj**)
7. Press **Update**
8. BLE Update tool will compile the project and install it into the target device.

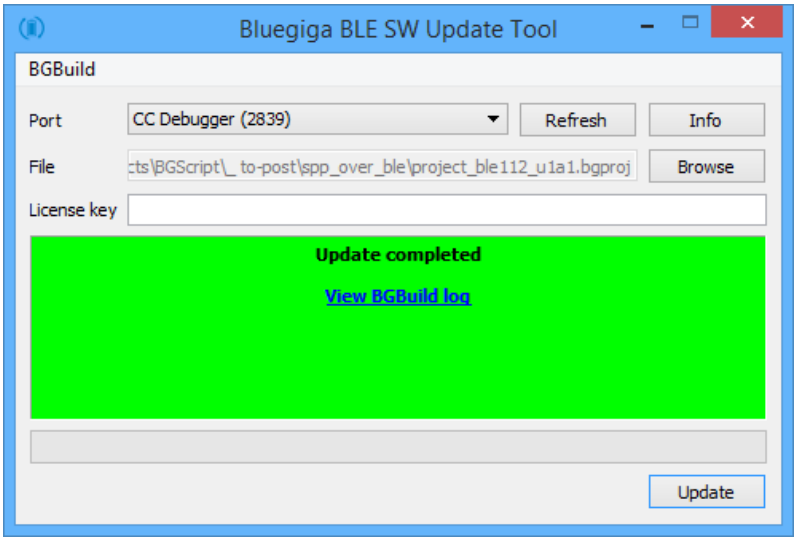

Figure 3: Compile and install with BLE Update tool

Note:

You can also double-click the desired **.bgproj** file and it will automatically open in the BLE Update tool.

If you have a DKBLE113 v1.2 or later, then the CC debugger hardware is integrated in the development kit, and you simply need to:

- Connect the **DEBUGGER** USB port to the PC
- Turn the **DEBUGGER** switch to **MODULE**
- Press the **RESET DEBUGGER** button and make sure the **DEBUGGER** LED turns green
- Continue from Step 4 above

The **View Build Log** button opens up a dialog that shows the “bgbuild.exe” compiler output, as well as the RAM and Flash memory allocations.

NOTE: when using Windows 8, the message “Qt: Untested Windows verion 6.2 detected!” is normal and may be ignored.

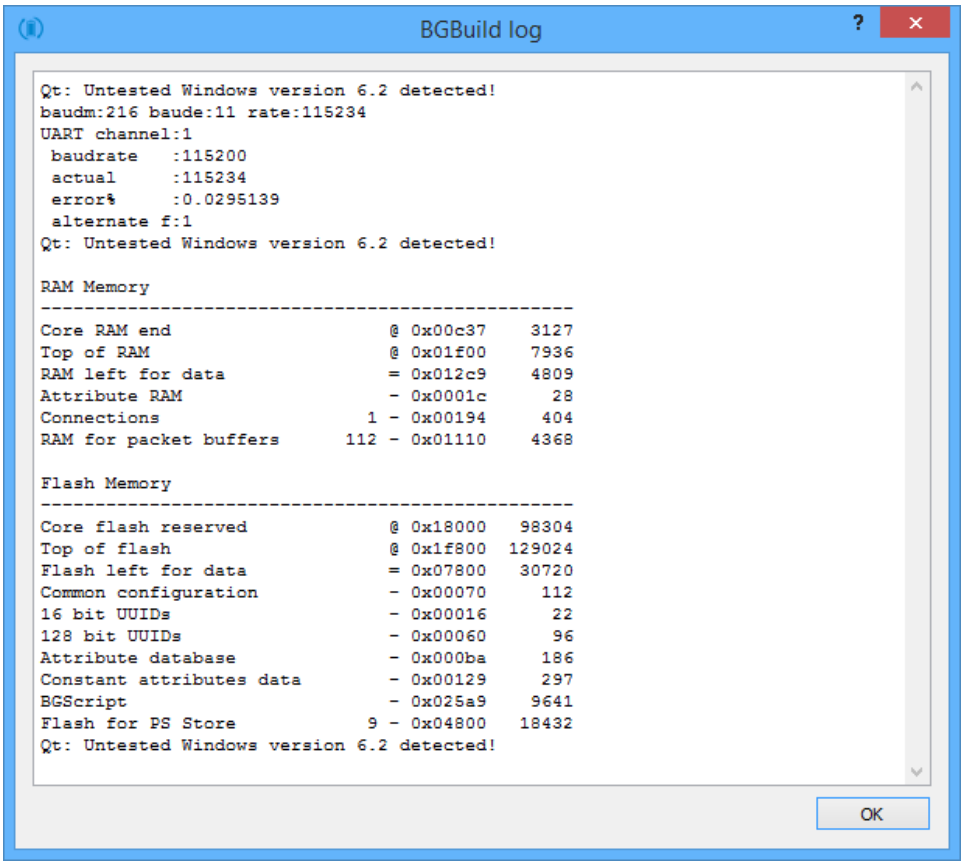

Figure 4: BLE Update build log

4.2.2 Compiling with bgbuild.exe

The project can also be compiled with the **bgbuild.exe** command line compiler. The BGBuild compiler simply generates the firmware image file, which can be installed on the BLE112, BLE113, or (if prepared correctly) the BLED112.

In order to compile the project using BGBuild:

1. Open Windows Command Prompt (cmd.exe)
2. Navigate to the directory where your project is
3. Execute BGbuild.exe compiler

   Syntax: *bgbuild.exe <project file>*

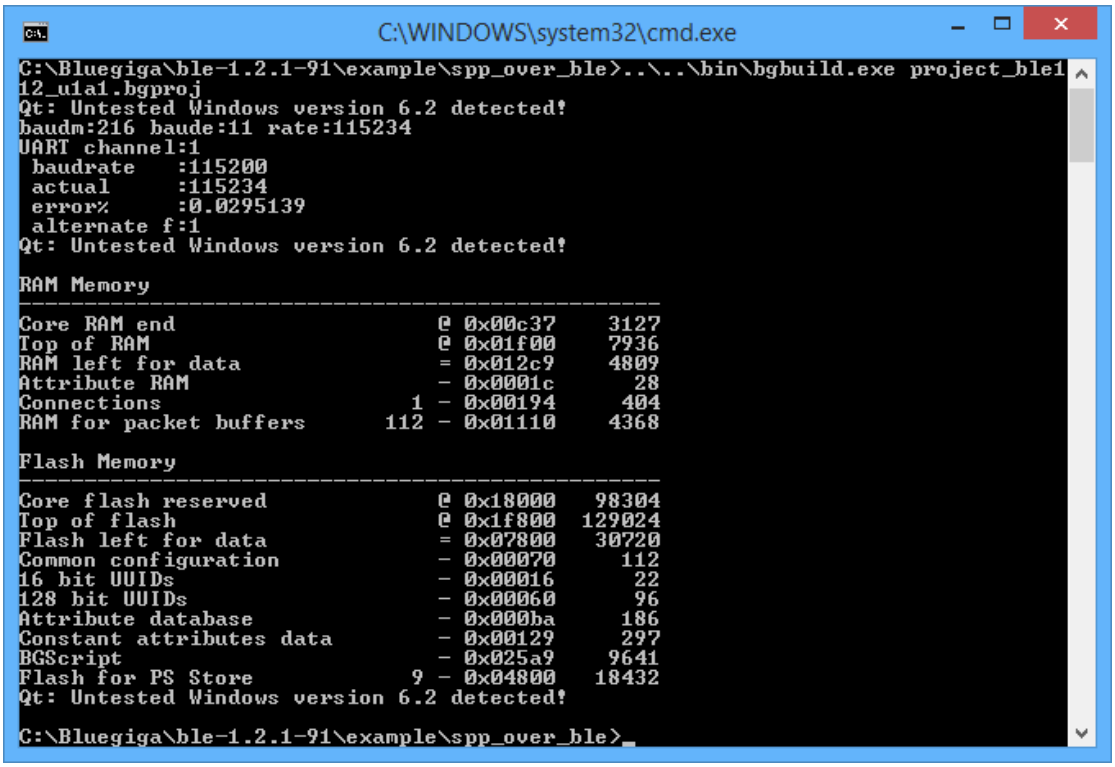

Figure 5: Compiling with BGBuild.exe

If the compilation is successful, a **.hex** file is generated which can be installed into a Bluetooth Smart module.

On the other hand, if the compilation fails due to syntax errors in the BGScript source or GATT file, an error message is displayed.

4.2.3 Installing the firmware with TI's SmartRF Flash Tool

TI Flash tool should **NOT** be used with projects compiled with the Bluegiga *Bluetooth* Smart SDK **v1.1 or newer**, and the **BLE Update** tool should be used instead. The BLE11x and BLED112 devices contain a license key which is needed for the firmware to operate, and if the device is programmed with the TI SmartRF Flash tool, this license key will be erased.

TI SmartRF should only ever be used to rewrite a new MAC address in case the old one has been lost (which does not happen under normal circumstances). It should never be used for completely reflashing a module with new firmware.

5 Testing the SPP-over-BLE device

5.1 Testing with BLEGUI software

This section describes how to test the DevA device with BLEGUI software. This requires manually acting as the DevB device using BLEGUI and a BLED112. Under normal circumstances, the other device will be programmed with matching firmware and will make this connection process much more automated (see later sections for testing with two devices both running the same firmware). **The use of a DKBLE112 dev kit is assumed for the DevA portion of this test.**

5.1.1 Discovering the SPP-over-BLE DevA device

With the default configuration, the DevA device **does not automatically start advertising when power is first applied**. Instead, it starts in an "idle" state, waiting for AT commands over the specified UART or USB interface. There are two ways to change this behaviour:

1. Connect over the UART or USB data interface and send an "ATA" or "ATP" command manually
2. Reset the BLE112 or BLE113 module while **P0_0** has 3.3v applied (automatic "ATP" on start-up)

The "ATA" command will cause the module to begin advertising, while the "ATP" command or the **P0_0** assertion will cause the module to enter ping-pong mode, which is a special mode that alternates between advertising and scanning every few seconds.

NOTE: booting in automatic ping-pong mode (with P0_0 asserted) but without the serial port opened in a terminal of some kind will result in a slow build-up of data in the TX buffer. If this buffer overflows (which can occur within 15 seconds), the module will freeze until the port is opened and the buffered data is read. This can be avoided by making sure the port is open and outgoing data is being read and processed whenever ping-pong mode is in use.

Preparations:

1. Serial Terminal
    a. Connect a UART or USB cable (appropriate to the firmware loaded onto the module)
    b. Open a serial terminal such as Realterm, PuTTY, TeraTerm, Coolterm, or minicom and select the correct port, using 115200 baud, 8/N/1, and RTS/CTS flow control.
    c. Power on or reset the module, **optionally holding down the P0_0 button on the dev kit to enable automatic ping-pong mode**.
    d. You should see a "READY" event in the serial terminal, possibly followed by alternating "ADV" and "SCAN" events if you entered ping-pong mode.
    e. If you did not enter ping-pong mode, type either "ATA" or "ATP" in the serial terminal to begin advertising or ping-pong mode, respectively. This is necessary in order to make the device connectable from BLEGUI.
2. BLEGUI
    a. Connect BLED112 USB dongle to a PC
    b. Start BLEGUI software
    c. Select the correct COM port from the drop down menu and press **Attach**
    d. Execute for example **Command – Info** menu item to make sure the communication works

Discovering DevA:

1. In BLEGUI, set desired scan parameters, check the **Active Scanning** box, and then press the **Set Scan Parameters** button
2. Select **Generic** scanning mode and **Start** scanning
3. If the DevA is powered on, in range, and advertising, it should appear in the main view as something like “BLE SPP 4F:E2:90” (the three hex bytes will be different).

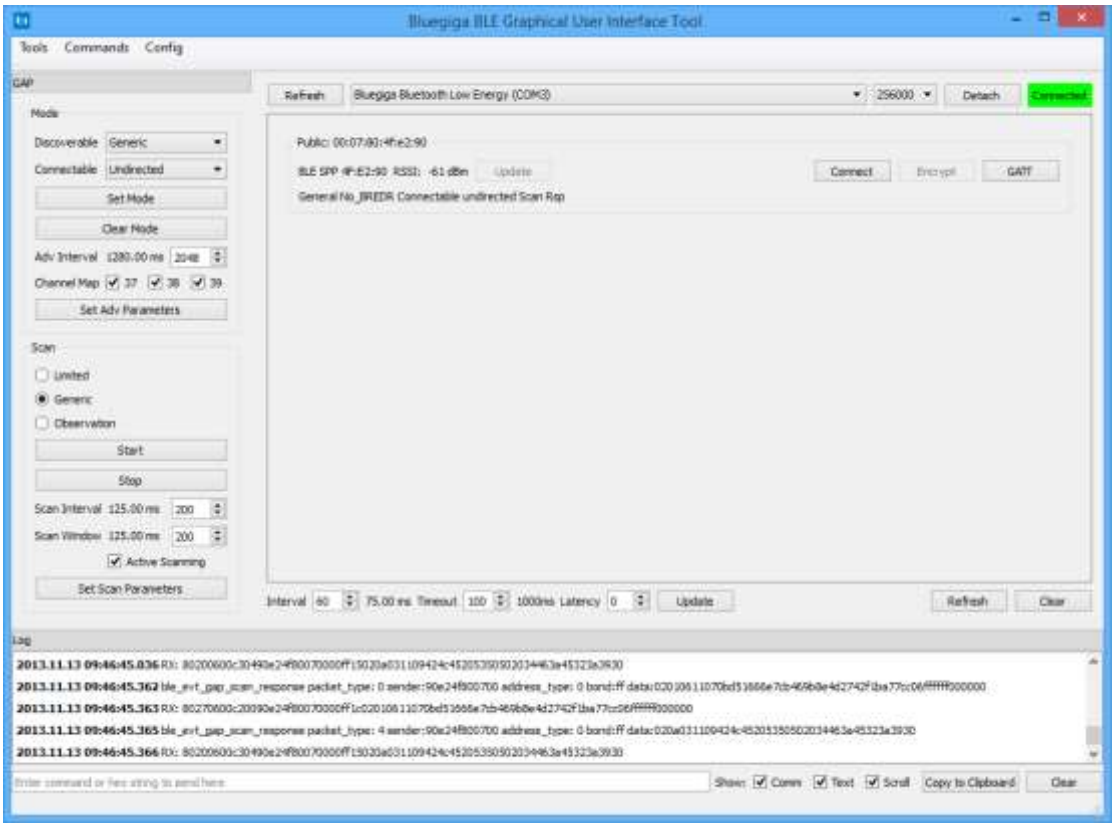

Figure 6: Scanning for SPP-over-BLE DevA

5.1.2 Establishing a *Bluetooth* Connection

1. Press the **Connect** button located next to the device you want to connect to

   a. If the connection is successful the connect button will change to **Disconnect**, and you should see the “RING” event appear in the separate serial terminal application.

   b. If the connection fails an error message is displayed in the **Log** view

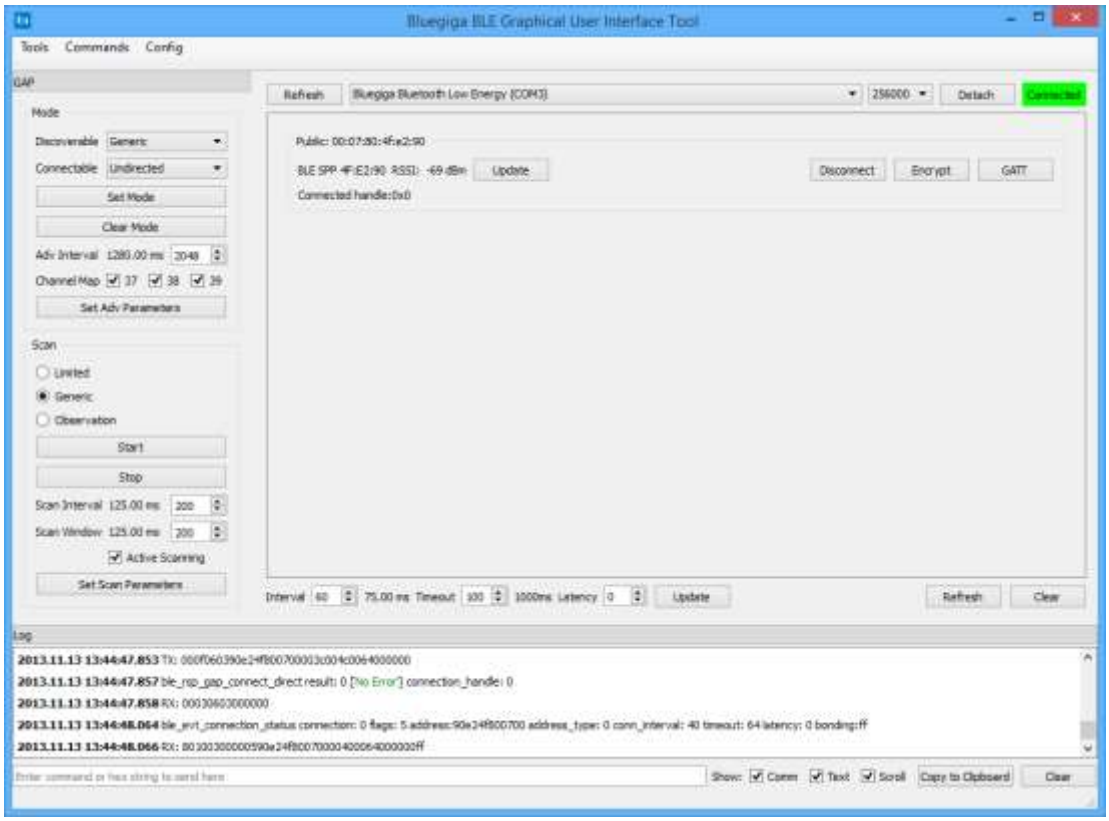

Figure 7: Connected to SPP-over-BLE DevA

5.1.3 Discovering Services

Once you've connected a device, you can use the GATT protocol to discover what services it supports. To discover the services of the SPP-over-BLE DevA:

1. Press the ***GATT*** button of the device you've just connected in order to open the GATT tool
2. Press ***Service Discover*** button to start a GATT primary service discovery procedure

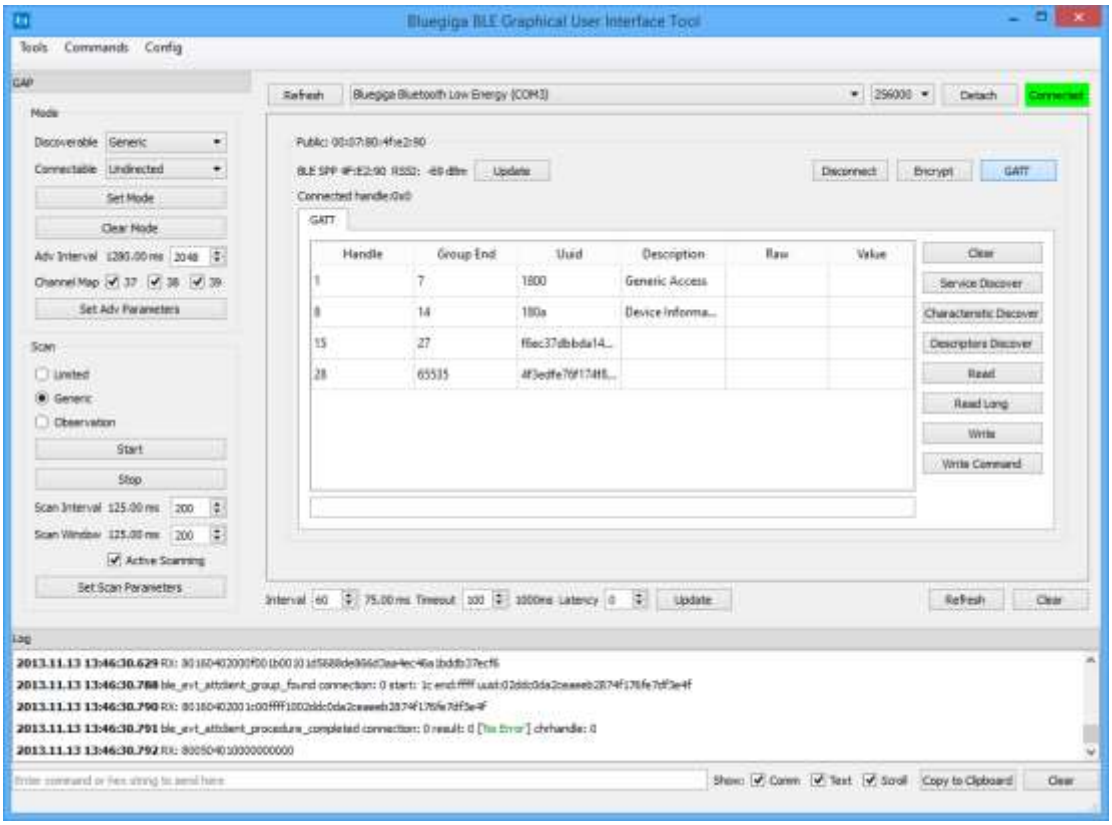

Figure 8: GATT service discovery

The four services defined in the **gatt.xml** should be visible in the GATT tool.

5.1.4 Receiving Data from DevA

Data can be received from DevA with the attribute protocol's **indicate** operation. Indication is a reliable way of transmitting data from the GATT server to the GATT client; new indication packets are generated whenever the GATT server has some new data to send to the client. Indications can carry up to 20 bytes of application data at a time.

Indications will not be sent by default, but instead the GATT client needs to enable them (a.k.a. "subscribe") first. In order to enable indications:

1. Select the SPP-over-BLE service in the BLEGUI's GATT view.
2. Press **Descriptors Discover** in order to see all characteristics and descriptors in the service.
3. Once the descriptors discovery is complete, locate the **Client Characteristics Configuration** (UUID: 2902) value that relates to the **SPP-over-BLE data** characteristic (UUID: af20fbac-2518-4998-9af7-af42540731b3) and select it. This attribute should have a Handle value of **23**, assuming no modifications.
4. In order to enable indications for the **SPP-over-BLE data** characteristic**,** use the **Write** button to write the value **2** to the **Client Characteristics Configuration**.
5. Finally make sure the write operation is executed properly (see log).

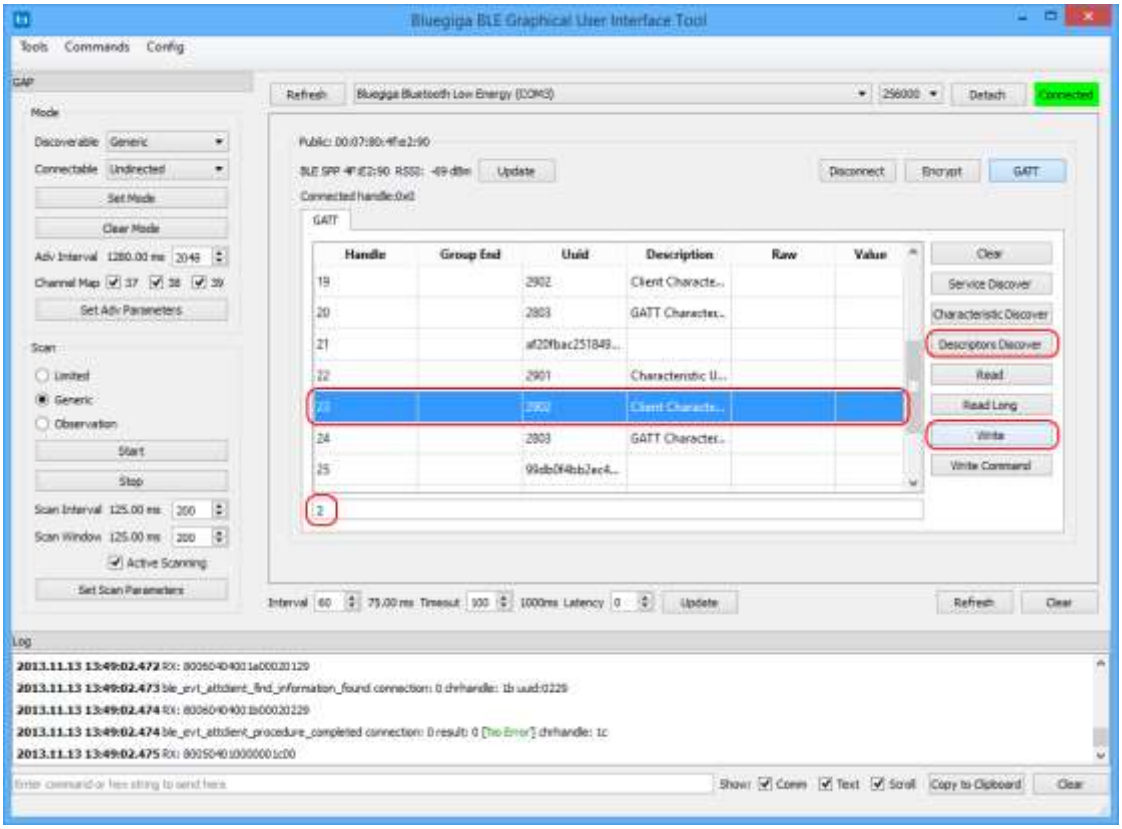

Figure 9: Enabling indications on SPP-over-BLE data characteristic

Once the indications have been enabled, data can be sent from DevA to DevB. In order to send data to BLEGUI, simply type characters into the previously opened serial terminal window connected to the DKBLE112. You can monitor BLEGUI software and the **SPP-over-BLE data** attribute for new data. Given most normal typing speeds, you will most likely see characters come in one at a time. Data is displayed in BLEGUI in hexadecimal, so if you type “abcd” into the terminal, you will see four updates showing **61**, **62**, **63**, and finally **64**.

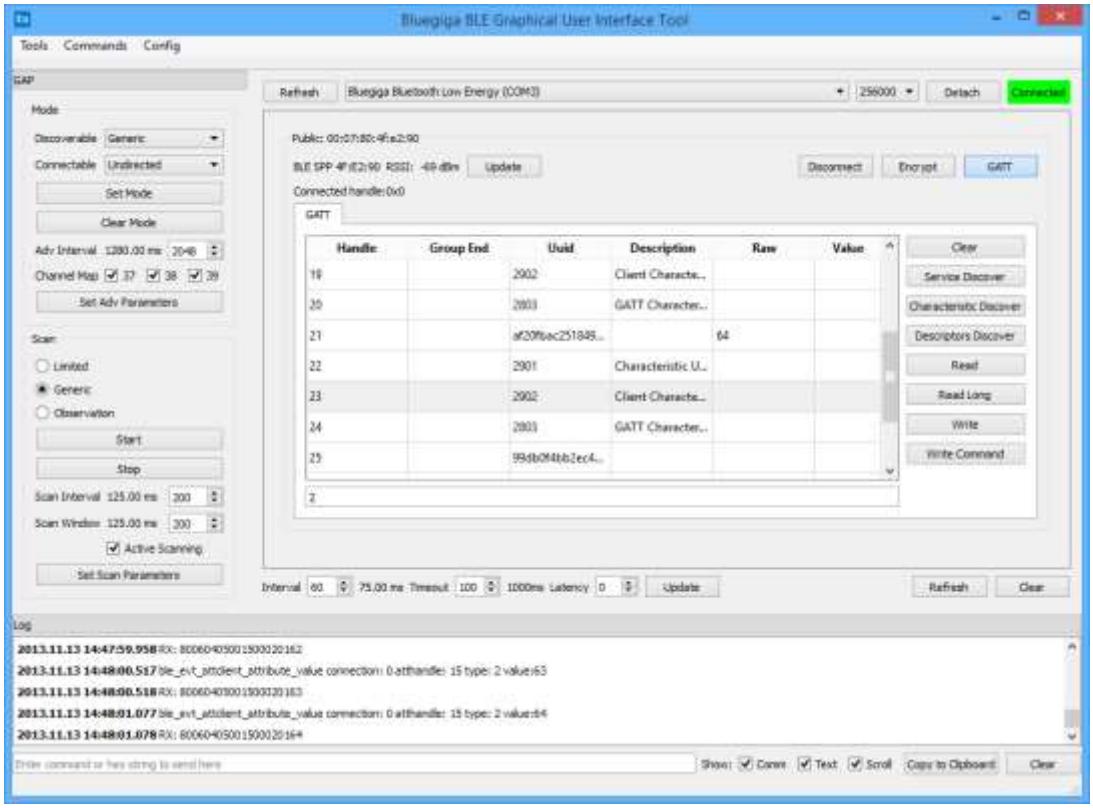

Figure 10: Data indication events

5.1.5 Sending Data to DevA

The **SPP-over-BLE data** attribute is writable, which allows you to reliably **write** up to 20 bytes of application data to DevA.

In order to send data to DevA:

1. Select the **SPP-over-BLE data** characteristic (UUID: af20fbac-2518-4998-9af7-af42540731b3) from the GATT entry list in BLEGUI. The correct attribute should have a Handle value of **21**, assuming no modifications.
2. Type the data (in hex) which you want to send to DevA in to the text field below the GATT table. For example, to send “abcd” you would enter “61 62 63 64” (spaces are optional).
3. Press the **Write** button to perform the “attribute write” procedure.
4. Verify from the log view below that no error is received.

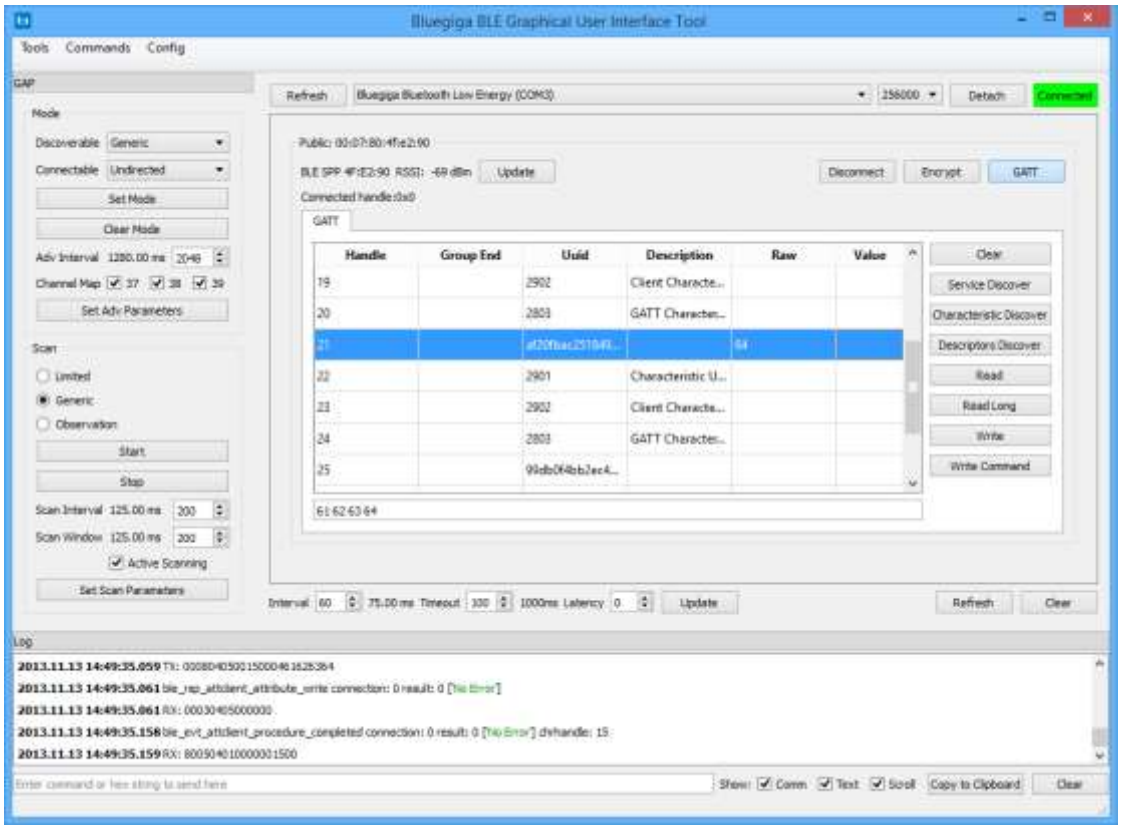

Figure 11: Sending data to DevA

If the DKBLE112 is used to receive data over the RS232 interface and the DKBLE112 is connected to terminal software as previously described, the transmitted data can be observed in the terminal window:

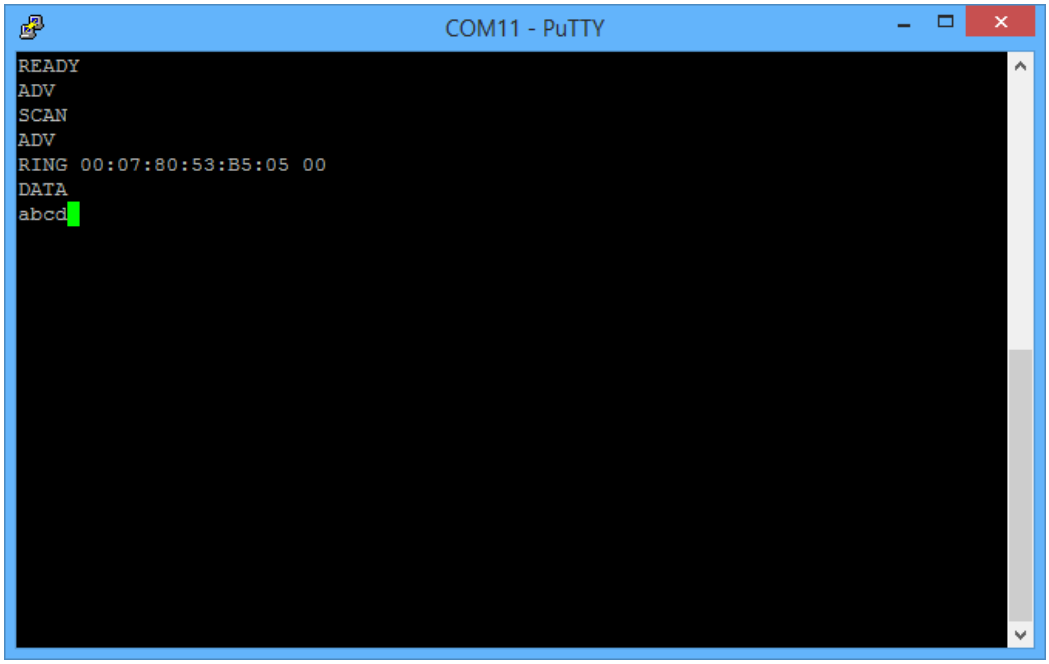

Figure 12: Terminal connected to DevA

This concludes the simple test of a single BLE-over-SPP DevA device using BLEGUI and a BLED112 as the DevB side of the connection. For a more “normal” test, use two BLE112 modules (or devkits) and two terminal windows, and employ the P0_0 assertion on boot to cause both modules to enter ping-pong mode automatically. They will connect on their own and open a data connection, ready for sending and receiving anything you wish to send.

6 GPIO Reference

With the default UART1/Alt1 endpoint configuration for streaming data, there are a number of GPIO pins left free for other purposes. These may be used for controlling the module's behavior (**Port 0** pins), as well as for detecting the current status of the module (**Port 1** pins). These pins are not available on the BLED112 dongle due to physical limitations.

6.1 GPIO Input Signals

In the UART1/Alt1 configuration, the P0_2/3/4/5 pins are used for CTS/RTS/TXD/RXD, leaving Port 0 pins 0/1/6/7 available for other purposes. These are used for the following purposes:

GPIO	Direction	Behavior
P0_0	**Input**	- **Hold HIGH** during boot to automatically enter ping-pong mode
P0_1	**Input**	- **Pulse HIGH** during connection to toggle between data and command modes
P0_6	**Input**	- **Hold HIGH** during boot/reset to prevent automatic entry into the configured role (advertising, scanning, or ping-pong mode) - **Pulse HIGH** after boot to disconnect or stop advertising/scanning
P0_7	**Input**	- **Hold HIGH** during boot to reset settings to factory default
NOTE: a DFU reboot can be triggered via GPIOs by holding all four of the above pins **HIGH** during boot. This is a failsafe in case you cannot communicate with the module via either UART or BLE.		

6.2 GPIO Output Signals

All of the Port 1 pins are available except for P1_7, which is configured in the BLE11x module projects to control an external DC/DC converter, if present. Therefore there are 7 pins which are used to indicate various behavior and link status bits. The logic state of these pins match exactly with the first byte of status reported by the **ATL** command; for details on that command, see Section 7.8.

GPIO	Direction	Behavior
P1_0	**Output**	Currently **advertising** (listening for incoming connection requests)
P1_1	**Output**	Currently **scanning** (searching for advertisement packets)
P1_2	**Output**	Connection pending (requested but not yet established)
P1_3	**Output**	Currently **connected**
P1_4	**Output**	Currently **encrypted** (only set if a connection is also present)
P1_5	**Output**	Connected as **master** (i.e. this device initiated the connection request)
P1_6	**Output**	Currently in **data mode** (only set if connection is also present)

6.2.1 GPIO Output Examples

By connecting a set of 7 LEDs to the **P1_0-6** pins of the BLE112 module (including current-limiting 220 ohm resistors), it is easy to see all of the status signals. In a real application, you would most likely connect these pins to MCU inputs, instead of or in additional to the LED indicators. Here are four example scenarios:

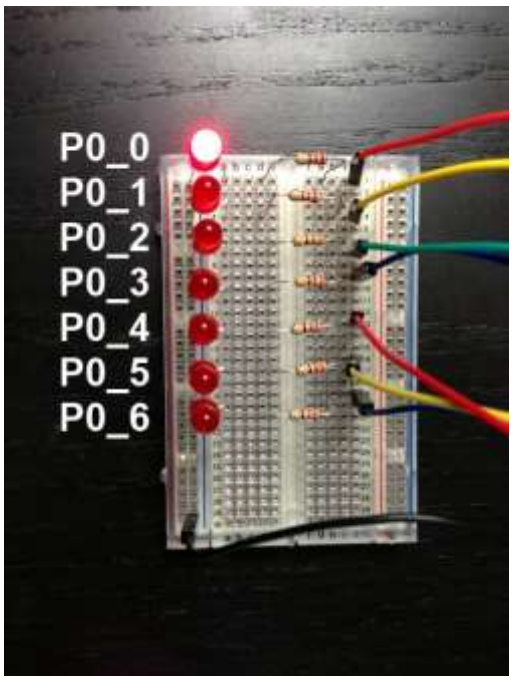

Figure 13: Advertising

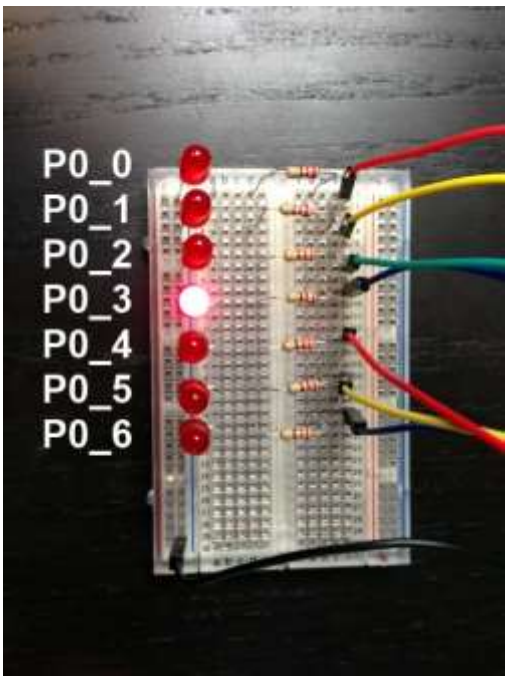

Figure 14: Connected as slave

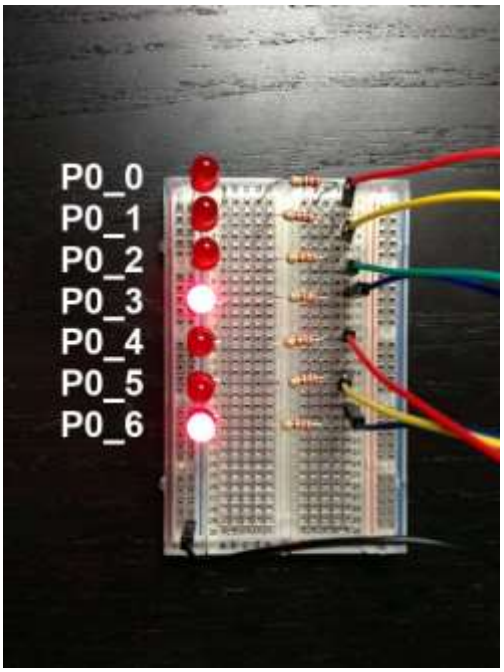

Figure 15: Connected as slave + data mode

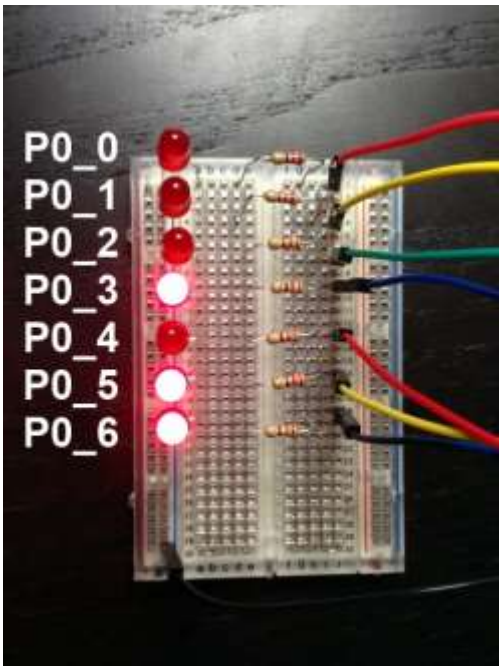

Figure 16: Connected as master + data mode

7 AT Command Reference

The SPP-over-BLE firmware designed to run on Bluegiga's BLE devices is controlled and configured via simple AT commands. The AT command parser behaves in the following way:

- Every command must end with a carriage return ("\r" or 0x0D), or a line feed ("\n", or 0x0A), or both.
- The maximum length of any command is **32** characters, not including the line ending(s). Any data that extends beyond 32 characters will be dropped.
- Commands are not case-sensitive.
- Every command is acknowledged with either "OK" or "ERR***n***", followed by CR+LF characters ("\r\n", or 0x0D 0x0A).
- The application must be in COMMAND mode (default boot state) in order to accept commands. If you have established a connection and seen the **DATA** event, then you are in DATA mode. It is possible to switch back and forth between DATA mode and COMMAND mode using a GPIO pulse on the P0_1 pin (not available on the BLED112 dongle). There is no support for an escape sequence over UART or USB at this time.

Possible error responses to commands sent are these:

Error	Description
ERR1	**Syntax error**, command not recognized. Verify that the command is formatted properly and that there are no leading or trailing characters.
ERR2	**Invalid state**, command cannot be executed. For example, you cannot call another device with the **ATD** command while you are currently connected.
ERR3	**Invalid parameter count**. This occurs if the "AT*x*" portion of the command is recognized, but the command requires a certain number of parameters which have not all been given.
ERR4	**Invalid parameter value**. This occurs if you supply a value that is outside of the acceptable range (mainly applies when using the **ATC** command to reconfigure certain aspects of the application).
ERR9 {code}	**BGAPI error**. This occurs if a specific error within the internal Bluegiga BGAPI protocol is detected. For detailed info, look up the meaning of the error code within the API Reference Guide, available to download from the Bluetooth Smart Software documentation page on our website.

A detailed list of all commands is found in the following sections.

7.1 AT – Test Serial Communication

The **AT** command is used to verify proper communication over UART with the local device, and to verify that it is in COMMAND mode.

7.1.1 Syntax

Synopsis:
AT

Response:
OK

7.1.2 Examples

```
AT
OK
```

7.2 ATA – Enter Advertising Mode

The **ATA** command will enter the **advertising** state (indicated by the **ADV** event). In this state, the BLE radio will periodicially send out advertisements which include the 128-bit SPP-over-BLE service UUID (**0bd51666-e7cb-469b-8e4d-2742f1ba77cc**) as well as a manufacturer data field containing the 3-byte connection key, which can be used for uniquely identifying one device among many, if desired.

NOTE: the default advertisement interval is 320ms (**0x0200** in hex, which is 512 * 0.625ms intervals). This value may be reconfigured with the ”ATC 03 05 {interval}” command.

NOTE: the default connection key is all zeros (**0x00 0x00 0x00**). This value may be reconfigured with the ”ATC 01 01 {key}” command.

7.2.1 Syntax

Synopsis:
ATA

Response:
OK

Events:
ADV

7.2.2 Examples

```
ATA
OK
ADV
```

7.3 ATC – Get or Set Configuration

The **ATC** command will display or update configuration options on the device. Note that any options which are updated only remain set that way until the device is rebooted or loses power, **unless** you write them to flash using the **ATW** command after setting them.

You can revert to all factory default settings with the **ATF** command.

The **ATC** command takes three parameters. The first one (**config_page**) is required. If you send the command with only this parameter, then the full page of options will be displayed in hexadecimal format, all on one line. This can be useful for quickly retreiving all configuration data for local parsing/storage on an external device.

The second parameter (**config_item**) specifies a single value to show or write. The item's value will be displayed if the third parameter (**value**) is ommitted, or it will be rewritten if the third parameter is supplied. Note that in all cases except for the Device Name option (which is a string of alphanumeric characters), all existing values are displayed in hexadecimal format, and all new values must be specified in this same format.

If you enter an incorrect number of parameters, you will see the **ERR3** event; or if you enter the correct number of parameters but the values are unknown or out of the allowed range, then you will see the **ERR4** event.

7.3.1 Syntax

Synopsis:
ATC {config_page} [config_item [value]]

Response:
OK

Page:	**Item:**	**Name:**	**Description**:
01	**01**	**Connection key**	The unique connection key for this module, advertised in a "manufacturer data" field in the main advertisement packet and used to identify matching devices in the auto-connecting **ATD** mode and the ping-pong **ATP** mode. This field may also be used for advertising up to three bytes of custom data for any purpose, if you have designed a unique master/central application (e.g. an iOS app) to read and process this data as desired. *3 bytes, default **00 00 00***
02	**01**	**Device name**	Device name in the local GATT structure and used in the scan response advertisement packet. This value is displayed (and rewritten) in **alphanumeric format**, not hexadecimal. Strings longer than 16 characters will be rejected. *16 bytes maximum, default **"BLE SPP AA:BB:CC"** (using last three bytes of device's MAC address)*

03	**01**	**Default role**	Defines the boot mode when powered on or reset: - 00 = Idle mode, no ad/scan - 01 = Scan + auto-connect as master (i.e. **ATD**) - 02 = Advertise as slave (i.e. **ATA**) - 03 = Ping-pong mode (i.e. **ATP**) *1 byte, default **00** (idle mode)*
03	**02**	**Ping-pong interval**	Number of seconds to split between advertising and scanning while in ping-pong mode. The exact split is not 50/50, but is instead somewhat randomized in order to ensure that an exact overlap cannot occur. This increases the possibility of a fast auto-connection between two devices which are both in ping-pong mode. *1 byte, default **04** (4 seconds)*
03	**03**	**Scan interval**	Number of 0.625ms units to wait between activating the BLE radio for RX during scan operations. This is the same value used for the "**gap_set_scan_parameters**" BGAPI command. *2 bytes, default **C8 00** (0x00C8 = 200 * 0.625ms = **125ms**)* *NOTE: 16-bit value specified in **little-endian** byte order*
03	**04**	**Scan window**	Number of 0.625ms units to enable the BLE radio for RX during scan operations. This is the same value used for the "**gap_set_scan_parameters**" BGAPI command. *2 bytes, default **C8 00** (0x00C8 = 200 * 0.625ms = **125ms**)* *NOTE: 16-bit value specified in **little-endian** byte order*
03	**05**	**Advertisement interval**	Number of 0.625ms units to wait between sending each advertisement packet. This is the same value used for the "**gap_set_adv_parameters**" BGAPI command. *2 bytes, default **00 02** (0x0200 = 512 * 0.625ms = **320ms**)* *NOTE: 16-bit value specified in **little-endian** byte order*
03	**06**	**Connection interval**	Number of 1.25ms units to wait between each connection maintenance or data transfer event. **THIS HAS A SIGNIFICANT EFFECT ON POTENTIAL THROUGHPUT** since only one 20-byte (maximum) data packet may be sent every other connection interval. This is a limitation of the BLE protocol. *2 bytes, default **10 00** (0x0010 = 16 * 1.25ms = **20ms**)* *NOTE: 16-bit value specified in **little-endian** byte order*
03	**07**	**Supervision timeout**	Number of 10ms units to wait after BLE communications stop before considering the link to be terminated. (Does not affect disconnections caused by hanging up on purpose, e.g. **ATH**.) *2 bytes, default **64 00** (0x0064 = 100 * 10ms = **1000ms**)* *NOTE: 16-bit value specified in **little-endian** byte order*

03	**08**	**Slave latency**	Number of connection intervals that a slave may skip without sending any connection maintenance packets to the master device. This can save some power on the slave device at the expense of making the very beginning of some data transfers slightly delayed, and making the connection slightly less reliable when the RF signal strength is weak. *2 bytes, default **00 00** (0x0000 = 0, **no latency**)* *NOTE: 16-bit value specified in **little-endian** byte order*
03	**09**	**Connection attempt timeout**	Number of seconds to allow a pending connection attempt (e.g. **ATD {mac_address}**) to continue before timing out and generating a **NORX** event. *1 byte, default **05** (0x05 = **5 seconds**)*
04	**01**	**Outgoing watermark**	Number of bytes that must be received from incoming UART/USB connection before those bytes are read and sent over BLE. The default value of "1" means that any data is sent whenever it is seen, but you can use a larger value if you only send packets that are a certain number of bytes in length, and you want them to be sent in a single BLE packet every time. Due to the transparent nature of the data flow in this application, it is often not necessary to change this, but you can if desired. *1 byte, default **01** (0x01 = 1 byte, e.g. **any/all data**)*
04	**02**	**UART baud rate**	Mantissa and exponent to set baud rate. Exact values may be calculated using the formula in section 17.4 of the CC2540 User Guide from TI, or you can refer to the commonly used baud rate table below. This value takes effect immediately, **before** the "OK\r\n" response is sent. *2 bytes, default **D8 0B** (mantissa=216, exponent=11 -> **115,200**)*
05	**01**	**TX power output**	Controls the TX power of the local BLE device. Acceptable values are between 0 and 15, and follow the same intervals that the **hardware_set_txpower** BGAPI command uses. Minimum (0) is -24 dBm, and maximum (15) is +3 dBm on the BLE112 or BLED112, or +0 dBm on the BLE113. *1 byte, default **0E** (0x0E = 15, **highest possible output power**)*

Commonly used baud rates:

Baud Rate	Mantissa	Exponent	Error	ATC 04 02 value
2400	**59**	**6**	*0.14%*	**3B06**
4800	**59**	**7**	*0.14%*	**3B07**
9600	**59**	**8**	*0.14%*	**3B08**
14,400	**216**	**8**	*0.03%*	**D808**
19,200	**59**	**9**	*0.14%*	**3B09**
28,800	**216**	**9**	*0.03%*	**D809**
38,400	**59**	**10**	*0.14%*	**3B0A**
57,600	**216**	**10**	*0.03%*	**D80A**
76,800	**59**	**11**	*0.14%*	**3B0B**
115,200	**216**	**11**	*0.03%*	**D80B**
230,400	**216**	**12**	*0.03%*	**D80C**

7.4 ATD – Call Device

The **ATD** command will initiate an outgoing connection process. When you enter the **ATD** command without any arguments, it will scan first and look for any advertising devices which have a matching 3-byte connection key. If a matching device is found, then it will automatically open a connection with that device.

If you supply the optional **mac_address** argument, then it will skip the scanning step and immediately try to connect to the supplied MAC address, assuming **address_type** = 0 (public). The **address_type** argument is also optional, but must be given if you are connecting to a device which uses random/private addressing. Note that iOS devices configured as peripherals operate in this fashion. The value specified for this argument (if supplied) must be the full two-character hexadecimal notation, either ”**00**” for public or ”**01**” for random/private.

If the call (automatic or manual) does not succeed within the configured timeout period, then the attempt will be cancelled and the **NORX** event will be displayed.

NOTE: the default timeout is 5 seconds (**0x05** in hex, value given in seconds). This value may be reconfigured with the ”ATC 03 09 {timeout}” command.

7.4.1 Syntax

Synopsis:
ATD [mac_address [address_type]]

Response:
OK

Events:
SCAN
RESP {mac_address} {address_type} {key} {rssi}
CONN {mac_address} {address_type}
DATA
NORX

7.4.2 Examples

Calling with auto-connection to matching devices:

```
ATD
OK
SCAN
RESP 00:07:80:3A:17:F8 00 000000
CALL 00:07:80:3A:17:F8 00
CONN 00:07:80:3A:17:F8 00
DATA
```

Calling a specific MAC address, using default address type (argument not supplied):

```
ATD 00:07:80:3A:17:F8
OK
CALL 00:07:80:3A:17:F8 00
CONN 00:07:80:3A:17:F8 00
DATA
```

Calling a specific MAC address, device out of range:

```
ATD 00:07:80:3A:17:F8
OK
CALL 00:07:80:3A:17:F8 00
NORX
```

7.5 ATE – Encrypt Connection

The **ATE** command will encrypt a currently established connection. Currently, the SPP-over-BLE firmware does not support **bonding**, but only per-connection encryption, so that this must be done each time a new connection is made. Future updates to this firmware will add bonding capability.

NOTE: the ”ENCRYPT” event will only be visible if you are in **command mode**. However, even if you are in **data mode**, the encryption will still occur

7.5.1 Syntax

Synopsis:
ATE

Response:
OK

Events:
ENCRYPT

7.5.2 Examples

```
ATE
OK
ENCRYPT
```

7.6 ATF – Factory Reset

The **ATF** command causes the device to reset **after first clearing any custom configuration values that have been set using the ATC command**. The configuration will revert to the factory default values. Note that an "OK" response will still be generated for acknowledgement before the reset occurs. The reset itself is performed internally using the "system_reset(0)" BGAPI command, so the module (or dongle) will be fully rebooted; it is not merely a software state initialization.

Once the reset completes, you should see a **READY** event occur, indicating that it is back in **command mode** and ready to accept AT commands.

NOTE: running this command on a BLED112 dongle (or a BLE112 configured for USB connectivity) will cause the device to momentarily disconnect and then re-enumerate on the USB port. **In Microsoft Windows, this will cause a loss of connectivity** if you have the port open in a terminal and do not **immediately** close it after sending the **ATZ** command. If you leave the port open while device re-enumerates, then you will need to close the port, physically disconnect the device from the USB port, then re-insert the device before you can begin using the USB-provided COM port again. **You cannot simply close and re-open the port in this case**.

7.6.1 Syntax

Synopsis:
ATF

Response:
OK

7.6.2 Examples

```
ATF
OK
[settings will be cleared and device will reset]
READY
```

7.7 ATH – Hang Up

The **ATH** command is used to close the current active connection, or exit from advertising, scanning, or ping-pong mode. If you are not in one of these modes (connected or ad/scan/ping-pong), then an **ERR2** message will be generated. This is not a terminal error and simply indicates that you were in the wrong state

Using this command while connected will generate a **DISC** event with the disconnect code.

7.7.1 Syntax

Synopsis:
ATH

Response:
OK

Events:
DISC {disconnect_code}

7.7.2 Examples

Exiting advertising mode:

```
ATA
OK
ADV
ATH
OK
```

Closing a connection:

```
CONN 00:07:80:3A:17:F8 00
DATA
READY
ATH
OK
DISC 0216
```

7.8 ATL – Get Link Status

The **ATL** command is used to display the current link status with the **LINK** event. This includes information on whether the device is currently advertising, scanning, connected, etc. The information is presented as two hexadecimal bytes (**little-endian byte order**) which represent a collection of status bits, as described below. The **ATL** command may be used at any time.

Bit:	Hex value:	Description:
0	**0100**	Currently **advertising** (listening for incoming connection requests)
1	**0200**	Currently **scanning** (searching for advertisement packets)
2	**0400**	Connection pending (requested but not yet established)
3	**0800**	Currently **connected**
4	**1000**	Currently **encrypted** (only set if a connection is also present)
5	**2000**	Connected as **master** (i.e. this device initiated the connection request)
6	**4000**	Currently in **data mode** (only set if connection is also present)
7	*Reserved*	*Reserved*
8	**0001**	Auto-connection enabled if scanning (using **ATD** or **ATP**)
9	**0002**	Ping-pong mode enabled (using ATP), used to re-enter after disconnection
10-15	*Reserved*	*Reserved*

NOTE: Bits **0-6** are masked directly onto the **Port 1** pins on the BLE112 and BLE113 modules. This means that you can connect them to a microcontroller or other device to detect any of that state information in real time and use it to control the MCU behaviour. This is particularly useful for the "**connected**" (P1_3) and "**data mode**" (P1_6) bits. Due to the hardware architecture, these pins are not available on the BLED112 dongle.

7.8.1 Syntax

Synopsis:
ATL

Response:
OK

Events:
LINK {status}

7.8.2 Examples

Link status when **advertising** is enabled:

```
ATA
OK
ADV
ATL
OK
LINK 0001
```

Link status when **scanning** is enabled:

```
ATS
OK
SCAN
ATL
OK
LINK 0002
```

Link status when **ping-pong mode** is enabled:

```
ATP
OK
ADV
ATL
OK
LINK 0101
```

7.9 ATM – Get MAC Address

The **ATM** command is used to display the local Bluetooth MAC address in hexadecimal format. This information is displayed with the **MAC** event.

7.9.1 Syntax

Synopsis:
ATM

Response:
OK

Events:
MAC {mac_address}

7.9.2 Examples

```
ATM
OK
MAC 00:07:80:3A:17:F8
```

7.10 ATP – Enter Ping-Pong Mode

The **ATP** command will enter **ping-pong** mode. This is one of the most powerful tools of the SPP-over-BLE application, since it automatically bounces back and forth between the **scanning** state and the **advertising** state every second or two (the exact timing is randomized at start-up based on the unique module serial number and the current ambient temperature reading).

When in ping-pong mode, the device will either advertise (as though you sent the **ATA** command) or scan with auto-connection (as though you sent the **ATD** command), and repeatedly switch between these two modes until a connection is established. This is particularly useful when used with a unique 3-byte connection key, since you can configure a pair of devices to have their own key and then put both of them in ping-pong mode. If they are in range, they will automatically connect without any further interaction as soon as one of them is advertising and the other is scanning.

7.10.1 Syntax

Synopsis:
ATP

Response:
OK

Events:
SCAN
ADV
RESP {mac_address} {address_type} {key} {rssi}
CONN {mac_address} {address_type}
RING {mac_address} {address_type}
DATA

7.10.2 Examples

Ping-pong mode where device answers as slave:

```
ATP
OK
ADV
SCAN
ADV
RING 00:07:80:4F:E2:90 00
DATA
```

Ping-pong mode where device calls as master:

```
ATP
OK
ADV
SCAN
ADV
SCAN
RESP 00:07:80:3A:17:F8 00 000000
CALL 00:07:80:3A:17:F8 00
CONN 00:07:80:3A:17:F8 00
DATA
```

7.11 ATS – Enter Scanning Mode

The **ATS** command will enter the **scanning** state (indicated by the **SCAN** event). In this state, the BLE radio will scan for any advertising devices which include the 128-bit SPP-over-BLE service UUID (**0bd51666-e7cb-469b-8e4d-2742f1ba77cc**) in their advertisment packets.

When scanning via the **ATS** command, the device will *not* automatically connect; it will only display the advertised information from detected devices, presented in the **RESP** event. as well as a manufacturer data field containing the 3-byte connection key, which can be used for uniquely identifying one device among many, if desired.

NOTE: the default scan interval is 125ms (**0x00c8** in hex, which is 200 * 0.625ms intervals). This value may be reconfigured with the ”ATC 03 03 {interval}” command.

NOTE: the default scan window is 125ms (**0x00c8** in hex, which is 200 * 0.625ms intervals). This value may be reconfigured with the ”ATC 03 04 {window}” command.

7.11.1 Syntax

Synopsis:
ATS

Response:
OK

Events:
SCAN **RESP {mac_address} {address_type} {key} {rssi}**

7.11.2 Examples

```
ATS
OK
SCAN
RESP 00:07:80:3A:17:F8 00 000000 -71
RESP 00:07:80:3A:17:F8 00 000000 -71
RESP 00:07:80:3A:17:F8 00 000000 -67
```

7.12 ATV – Get Firmware Version

The **ATV** command is used to display the SPP-over-BLE firmware version. This information is displayed with the **VER** event, which is formatted with a "`major.minor.patch.variant`" structure. The variant specifies which project source was used to build the firmware (1=BLE11x-U1A1, 2=BLE112-USB, 3=BLED112-USB).

7.12.1 Syntax

Synopsis:
ATV

Response:
OK

Events:
VER {version}

7.12.2 Examples

```
ATV
OK
VER 1.0.0.1
```

7.13 ATW – Write Configuration Values

The **ATW** command is used to write the current configuration settings to the module's PS keys so that they will automatically take effect on power-on or reset. The PS key space is in non-volatile flash memory, so settings stored here will persist until they are overwritten with **ATW** again, or until you reflash the module (which cleans out all PS key storage).

The **SAVED** event is displayed after the values have been written to flash.

7.13.1 Syntax

Synopsis:
ATW

Response:
OK

Events:
SAVED

7.13.2 Examples

```
ATW
OK
SAVED
```

7.14 ATY – DFU Reset

The **ATY** command causes the device to reset into DFU mode. Note that an “OK” response will still be generated for acknowledgement before the reset occurs. The reset is performed internally using the “system_reset(1)” BGAPI command.

Once the reset completes, **the device will no longer be accessible over the UART or USB port** for serial communication. You will need to use one of the Bluegiga-provided DFU mechanisms—most simply, the DFU interface within the **BLEGUI** application that is part of the BLE SDK—in order to perform the DFU operation. Alternatively, you can remove and reapply power to the device (power-cycle, hardware reset, or remove/reinsert USB) in order to return to a normal boot mode without

NOTE: DFU operations are not typically required as part of the normal operation of the SPP-over-BLE firmware. The DFU operation does not have anything to do specifically with SPP-over-BLE functionality, but it simply provides a way to change or update the firmware running on the Bluegiga device. This is particularly useful in the case of the BLED112 dongle, since there is no other way to update or change the firmware running on the device. For other devices (BLE112 or BLE113 module), the CC debugger is the preferred way to modify firmware.

7.14.1 Syntax

Synopsis:
ATY

Response:
OK

7.14.2 Examples

```
ATY
OK
[device will reset into DFU mode]
```

7.15 ATZ – Reset

The **ATZ** command causes the device to reset. Note that an “OK” response will still be generated for acknowledgement before the reset occurs. The reset is performed internally using the “system_reset(0)” BGAPI command, so the module (or dongle) will be fully rebooted; it is not merely a software state initialization.

Once the reset completes, you should see a **READY** event occur, indicating that it is back in **command mode** and ready to accept AT commands.

NOTE: running this command on a BLED112 dongle (or a BLE112 configured for USB connectivity) will cause the device to momentarily disconnect and then re-enumerate on the USB port. **In Microsoft Windows, this will cause a loss of connectivity** if you have the port open in a terminal and do not **immediately** close it after sending the **ATZ** command. If you leave the port open while device re-enumerates, then you will need to close the port, physically disconnect the device from the USB port, then re-insert the device before you can begin using the USB-provided COM port again. **You cannot simply close and re-open the port in this case**.

7.15.1 Syntax

Synopsis:
ATZ

Response:
OK

7.15.2 Examples

```
ATZ
OK
[device will reset]
READY
```

8 Appendix

8.1 External resources

- *Bluetooth* 4.0 software development kit is available at : www.bluegiga.com
- BLE112 and DKBLE112 hardware documentation is available at : www.bluegiga.com
- Project files for the SPP-over-BLE firmware are in the SDK archive and online at: www.bluegiga.com
- *Bluetooth* SIG's developer portal: https://developer.bluetooth.org/

9 Contact information

Sales: sales@bluegiga.com

Technical support: www.bluegiga.com/support

Orders: orders@bluegiga.com

WWW: www.bluegiga.com
www.bluegiga.hk

Head Office / Finland:

Phone: +358-9-4355 060
Fax: +358-9-4355 0660
Sinikalliontie 5A
02630 ESPOO
FINLAND

Postal address / Finland:

P.O. BOX 120
02631 ESPOO
FINLAND

Sales Office / USA:

Phone: +1 770 291 2181
Fax: +1 770 291 2183
Bluegiga Technologies, Inc.
3235 Satellite Boulevard, Building 400, Suite 300
Duluth, GA, 30096, USA

Sales Office / Hong-Kong:

Phone: +852 3972 2186
Bluegiga Technologies Ltd.
Unit 10-18
32/F, Tower 1, Millennium City 1
388 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong