AN984: *BLUETOOTH* LOW ENERGY SOFTWARE

Implementing Over-the-Air Firmware Upgrade

Wednesday, 02 December 2020

Version 1.10

VERSION HISTORY

Version	Comment
1.0	First version
1.1	Hardware reference added
1.2	Minor updates
1.3	Updated reference schematic
1.4	Updated Introduction chapter
1.5	Updated to match the Bluetooth Smart Software v.1.2.2 - external SPI flash board instructions added - 256kB instructions added
1.6	BLE113-A-256 instructions added
1.7	OTA Data Attribute length fixed to 20 bytes.
1.8	Added partial OTA update instructions
1.9	BGScript code examples updated
1.10	Renamed "Bluetooth Smart" to "Bluetooth Low Energy" according to the official Bluetooth SIG nomenclature

TABLE OF CONTENTS

1 Introduction

This application note discusses how to enable and perform Over-the-Air (OTA) firmware upgrade using the Bluegiga *Bluetooth* Low Energy Modules and Software

The application note contains a practical example of how to build *Bluetooth* Low Energy GATT based OTA update services with the profile toolkit, how to make a BGScript application that performs the firmware upgrade.

An assumption is made that the reader of this application note is already somewhat familiar with the *Bluetooth* Low Energy SDK.

The OTA feature is available in the *Bluetooth* Low Energy Software and SDK v.1.2.2 and newer.

In v.1.3 software a partial OTA update feature was added, which allows only BGScript and/or GATT database to be updated via the OTA firmware update. Although the partial OTA update does not allow the Bluetooth stack to be updated the benefit is that it does works with the Bluetooth Low Energy modules that have 128kB flash and the update file is significantly smaller than the full OTA update file, making the update procedure much faster and lower power.

The full OTA firmware update either requires an external SPI flash connected to the SPI interface of BLE112 or BLE113 or 256kB flash version of BLE113 (part number: BLE113-A-256). BLE121LR Bluetooth Low Energy Long Range module has built-in 256kB flash. The partial OTA update works with all the Bluegiga Bluetooth Low Energy Modules.

2 Implementation of OTA Firmware Upgrade

In this chapter we describe and discuss an actual implementation of the OTA firmware upgrade and the necessary steps. The implementation consists of following steps:

1. Prerequisites
2. Installing the tools
3. Setting up the project
4. Defining hardware configuration
5. Building a GATT based OTA server service database with profile toolkit
6. Writing a simple BGScript that performs the firmware upgrade
7. Compiling the GATT data base and BGScript into a binary firmware
8. Installing the firmware into BLE112 or BLED112 hardware
9. Testing it out

2.1 Limitations and Features of OTA firmware update

At the moment the OTA firmware update has the following limitations:

- A full OTA firmware update can be used to update following components of the firmware:
    - The Bluetooth Low Energy Stack
    - The GATT database
    - The BGScript Application
- A partial OTA firmware update on the other hand can be used to update following components of the firmware:
    - GATT database and/or
    - BGScript application
- OTA or DFU firmware update cannot however be used to update:
    - Hardware configuration of the module
    - Boot loader
- The full OTA update requires at least 256kB flash memory (internal or external SPI flash) whereas the partial OTA update works also with the Bluetooth Low Energy modules that have 128kB flash.
- At the moment only the Winbond SPI flash chips are supported by the OTA boot loader.

2.2 Prerequisites

In order to perform the OTA firmware update at least 256kB of flash memory is needed. The standard BLE112 and BLE113 *Bluetooth* Low Energy Modules only have a 128kB flash memory, but you can simply connect a low cost SPI flash memory to one of the SPI interfaces in the BLE112 or BLE113 modules. A variant of BLE113 exist with on-board 256kB flash and it does not require an external SPI flash and the BLE121LR module has 256kB of flash.

External SPI flash memories are typically very low cost and can be as cheap as $0.2-0.3.

2.3 Installing the Tools

1. Download the latest install the Bluegiga *Bluetooth* Low Energy SDK from the Bluegiga web site
2. Run the executable
3. Follow the on-screen instructions and install the SDK to the desired directory
4. Perform a full Installation

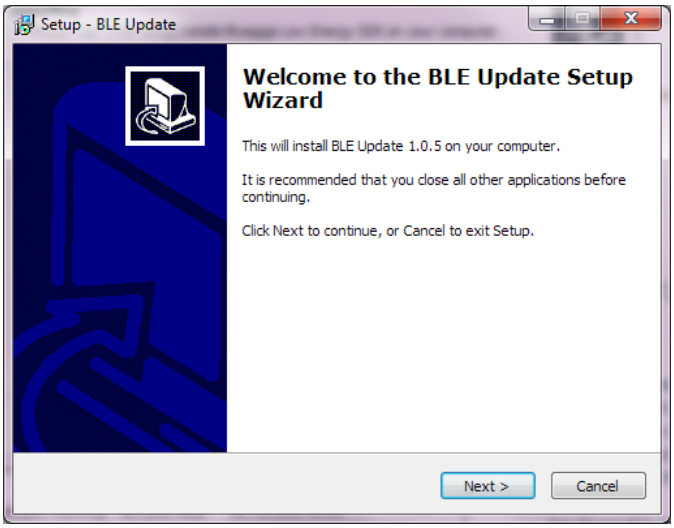

Figure 1: Installing Bluegiga *Bluetooth* Low Energy SDK

2.4 Implementing OTA with Products That Have 128kB Flash

This section discusses how to do the OTA firmware update with products that have built-in 128kB flash memory. These products are BLE112 Bluetooth Low Energy Module and BLE113 Bluetooth Low Energy Module.

Products that have only 128kB built-in flash can still be updated with the OTA firmware update, but you need to use an external SPI flash memory for storing the firmware update or alternatively make only the partial OTA firmware update which requires less flash.

The Bluetooth Low Energy Stack, the GATT database and you application typically require 90-100kB of flash memory and therefore a full OTA firmware update requires at least 256kB of flash. The partial OTA firmware update, which does not contain the Bluetooth Low Energy stack, can be as small as 3-5kB and can be made with products that only have 128kB built-in flash memory.

2.4.1 Adding OTA Support in Your Project

When you want to include OTA firmware update in your Bluetooth Low Energy project you need to make a few configurations into the project file. The project file defines the basic properties and resources used in your Bluetooth Low Energy project and it's a simple XML formatted file as can be seen below.

Below is an example project file for BLE113 Bluetooth Low Energy module (with 128kB built-in flash) with OTA firmware update support.

```
<?xml version="1.0" encoding="UTF-8" ?>
<project>
    <gatt in="gatt.xml" />
    <hardware in="hardware.xml" />
    <script in="ota.bgs" />
    <device type="ble113" />
    <boot fw="bootota" />
    <image out="out-ble113.hex" />
    <ota out="ble113.ota" firmware="true" />
</project>
```

Figure 2: Project file

The project configuration is described within the <project> tags

- <gatt> tag defines the .XML file containing the GATT data base
- <hardware> tag defines the .XML file containing the hardware configuration
- <script> tag defines the .BGS file containing the BGScript code.
- <image> tag defines the output .HEX file containing the firmware image
- <device type> tag defines if the project is meant for BLE113 hardware
- <boot fw> tag defines which interface is enabled for DFU firmware upgrades. **bootota** means the OTA boot loader will be used.
- <image out> tag defines the firmware update file which can be updated to the module over the debug programming interface or over the UART/USB interface using the DFU protocol
- <ota> tag defines the OTA firmware update file which is the actual firmware update file uploaded to the device over a Bluetooth LE connection. When you want to create a full OTA firmware update, which includes the Bluetooth Low Energy stack, GATT database and possible BGScript application use **firmware="true"** option within the <ota> tags.

The exact syntax and options of the project file can be found from the *Bluetooth Low Energy Module Configuration Guide* and the syntax is not fully described in this document.

2.4.2 GATT Service for OTA

This section discusses the implementation of OTA GATT service using the Profile Toolkit™.

The figure below shows the OTA service required in the GATT database for the OTA update to work the service uses a 128-bit manufacturer specific UUID for both the service and the characteristics.

```
<service uuid="1d14d6ee-fd63-4fa1-bfa4-8f47b42119f0">

    <description>Bluegiga OTA</description>

    <characteristic uuid="f7bf3564-fb6d-4e53-88a4-5e37e0326063" id="ota_control">
        <properties write="true" />
        <value length="1" type="user" />
    </characteristic>

    <characteristic uuid="984227f3-34fc-4045-a5d0-2c581f81a153" id="ota_data">
        <properties write_no_response="true" />
        <value length="20" />
    </characteristic>
```

Figure 3: OTA GATT Service

OTA service UUID: 1d14d6ee-fd63-4fa1-bfa4-8f47b42119f0

Two characteristics are required in the OTA service and they are:

Characteristic	UUID	Type	Length	Support	Security	Properties
OTA Control Point Attribute	f7bf3564-fb6d-4e53-88a4-5e37e0326063	hex	1 byte	Mandatory	none	Write User
OTA Data Attribute	984227f3-34fc-4045-a5d0-2c581f81a153	hex	20 bytes	Mandatory	none	Write without response

Table 1: OTA service characteristics description

The OTA control point attribute is used to control the firmware upgrade process between the device that will be updated and the device that performs the update. It is a write only attribute to the control can only made by the device that performs the update. The OTA control attribute has the **user** property enabled, which means that the BGScript application will need to read to data and acknowledge it to the sender and the acknowledgement is NOT automatically handled by the *Bluetooth* stack.

The 2nd attribute is on the other hand used to transmit the data from the device that performs the update to the device that is being updated. It's a **write without response,** so data transfers on the receiving end will not be acknowledged, increasing potential throughput. Note that it is not a "user" characteristic, so the stack will automatically handle the GATT data transfer seamlessly.

2.4.3

2.4.4 Hardware Configuration for External SPI flash

When an external SPI flash is used, a proper hardware configuration needs to be made for the Bluetooth Low Energy software to be able to access the flash chip over one of the SPI interfaces.

Below is an example for a hardware configuration that works with the DKBLE development kits and the setup used in them.

```
<?xml version="1.0" encoding="UTF-8" ?>

<hardware>
    <sleeposc enable="true" ppm="30" />
    <sleep enable="true" />
    <usart channel="0" mode="spi_master" alternate="2" polarity="negative"
    phase="0" endianness="msb" baud="2000000" endpoint="none" />
    <otaboot cs_port="0" cs_pin="7" power_port="1" power_pin="1" uart="0" />
    <pmux regulator_pin="7" />
    <txpower power="15" bias="5" />
    <script enable="true" />
</hardware>
```

Figure 4: Hardware configuration for the external SPI flash

The hardware configuration is described within the <hardware> tags

- <sleeposc> tag defines whether the sleep oscillator is enabled, which allows the low power sleep modes to be used. The BLE113 does incorporate the sleep oscillator so this value should be set to **true** especially in the applications where power consumption matters. The PPM value defines the sleep oscillator accuracy and MUST not be changed.
- <sleep> tag is used to enable the low power sleep modes on the device
- <usart> tag is used to enable the SPI master interface at 2Mbps. The OTA firmware is uploaded to an external flash and the SPI interface is used as the interface to it.
- <otaboot> tag is used to select the SPI interface, the chip select and power-on/off pins for the OTA boot loader, so the boot loader know which SPI interface and configuration is used.
- <pmux> configuration defines which GPIO pin is used to control an external DC/DC converter. An external DC/DC converter can be used to lower the peak power consumption during radio activity and the *Bluetooth* Low Energy software will automatically enable or disable the DC/DC based on the software status. The DKBLE, DKBLE112 and DKBLE113 development kits have the DC/DC converter, so this feature is enabled.
- <txpower> tag defines the TX power level and the value 15 configures the maximum TX power level.
- <script> tag defines if BGScript application is present. Since the example uses a BGScript application to perform the firmware upgrade we set this value to **true.**

The exact syntax and options of the project file can be found from the *Bluetooth Low Energy Module Configuration Guide* and the syntax is not fully described in this document.

2.4.5 Reference Schematic for External SPI Flash

In case you want to perform a full OTA update with products that have only 128kB flash you need to have an external SPI flash memory connected to one of the SPI interfaces of the Bluetooth Low Energy module. Below is a simple example schematic how to connect a SPI flash either to BLE112 or BLE113 Bluetooth Low Energy module

The external flash is powered from one of the high current IO's on the BLE112 or BLE113. In this reference the flash supply is taken from P1_0. When the external flash is used, P1_0 is first driven high by the software in order to power up to power up the flash. When the flash is not used, all the lines connected to the flash are driven low to avoid leakage currents.

When the flash is not used the chip select output of the module is in high impedance and requires a pull-up resistor. The modules MISO line also requires a pull-up resistor to make sure it is pulled low instantly with the supply voltage when the flash is not used.

DKBLE Bluetooth Low Energy development kits contain a SPI flash memory so you can also refer to DKBLE development kit schematics for more information.

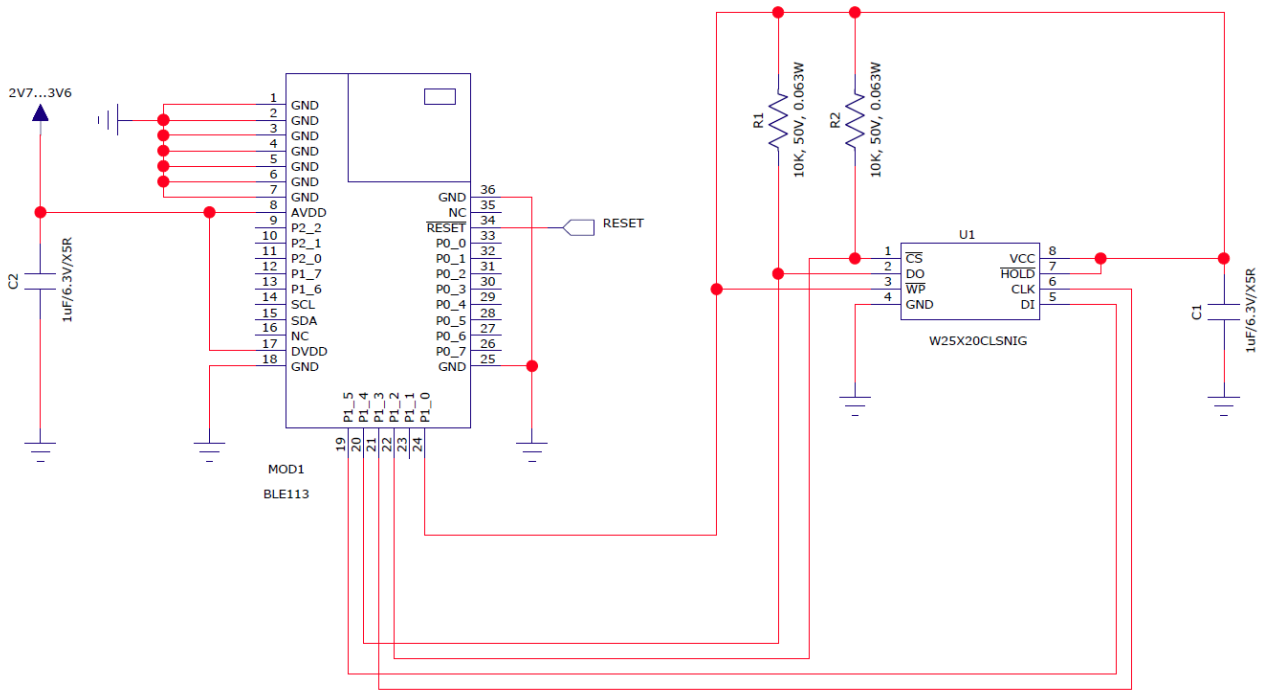

Figure 5: Example schematic

2.4.6 BGScript Application for External Flash

The OTA example application delivered with the Bluetooth Low Energy SDK implements a simple BGScript application that demonstrates the OTA firmware update capability and it can be used as an example and starting point to integrate the OTA firmware update to real applications. The application is implemented with BGScript scripting language and the code is explained in this chapter.

2.4.6.1 Boot Event and Initializations

The code is executed when the device is powered-up and it enables advertisements so the local can be discovered and connected by remote devices. The code also initializes the I/O pins used to control the external flash chip.

```
# Boot event listener - Starts advertisements, initializes variables and
# configures the I/O pins used for external flash
event system_boot(major, minor, patch, build, ll_version, protocol_version, hw)

    #Set device to advertisement mode and allow undirected connections
    call gap_set_mode(2,2)

    # Initialize the DFU pointer
    dfu_pointer             = 0

    # Inti Flash retry counter and MAX retries
    retry_counter           = 0
    max_retries             = 10

    # set power pin as output and pull down
    # also set p1.1 to output (does not have internal pull-resistor)
    call hardware_io_port_write(1,flash_power|p1_out,p1_out)
    call hardware_io_port_config_direction(1,flash_power|p1_out)

    call hardware_io_port_write(0, flash_cs, $ff)                   # deassert flash cs
    call hardware_io_port_config_direction(0, flash_cs)             # flash cs as output
end
```

2.4.6.2 Handling OTA Commands and Data

To handle the incoming OTA commands and data from the remote device (OTA application) an event handler for received data must be written. The data received from the remote end can be handled with the **`attributes_value`** (…) event listener, which will catch an event whenever a GATT characteristic is written.

In the example application the handler only checks of the **OTA Control Point Attribute** or **OTA Data Attribute** are written. The **OTA Control Point Attribute** carries commands such as flash erase or DFU boot and the **OTA Data Attribute** carries the actual firmware update.

The event handled code is below and more detailed information how the code works can be found from the comments.

```
# Incoming data event listener
# Handles OTA Control Point Attribute (commands) and
# OTA Data Attribute (firmware update) writes and
# performs the necessary actions
event attributes_value(connection, reason, handle, offset, value_len, value_data)

    #save connection handle, is always 0 if only slave
    curr_connection=connection

    # Check if OTA control point attribute is written by the remote device and execute the command
        # Command 0 : Erase flash block 0 (0x0-0x1FFFF)
        # Command 1 : Erase flash block 1 (0x10000-0x3FFFF)
        # Command 2 : Reset DFU data pointer
        # Command 3 : Boot to DFU mode
    # In case of errors application error code 0x80 is returned to the remote device
    # In case the flash comms fails error code 0x90 is returned to the remote device
    if handle = ota_control then
        #attribute is user attribute, reason is always write_request_user
        if value_len >1 || offset >0 then
            # Not a valid command -> report application error code : 0x80
            call attributes_user_write_response(connection, $80)
        else
            command=value_data(0:1)
            if command=0 then   # Command 0 received -> Erase block 0

                #reset retry counter
                retry_counter=0

                #write enable, cs down
                call hardware_io_port_write(0, flash_cs, 0) # assert flash cs
                call hardware_spi_transfer(0,1,"\x06")
                call hardware_io_port_write(0, flash_cs, $ff) # deassert flash cs

                # erase block 0 : 0-1ffff
```

```
            call hardware_io_port_write(0, flash_cs, 0) # assert flash cs
            call hardware_spi_transfer(0,4,"\xd8\x00\x00\x00")
            call hardware_io_port_write(0, flash_cs, $ff) # deassert flash cs

            #start timer to poll for erase complete
            call hardware_set_soft_timer(6000,0,1)
        end if

        if command=1 then # Command 1 received -> Erase block 1

            #reset retry counter
            retry_counter=0

            #write enable
            call hardware_io_port_write(0, flash_cs, 0) # assert flash cs
            call hardware_spi_transfer(0,1,"\x06")
            call hardware_io_port_write(0, flash_cs, $ff) # deassert flash cs

            # erase block 1 : 10000-3ffff
            call hardware_io_port_write(0, flash_cs, 0) # assert flash cs
            call hardware_spi_transfer(0,4,"\xd8\x01\x00\x00")
            call hardware_io_port_write(0, flash_cs, $ff) # deassert flash cs

            #start timer to poll for erase complete
            call hardware_set_soft_timer(6000,0,1)
        end if
        if command=2 then # Command 2 received -> Erase DFU pointer
            dfu_pointer=0
            call attributes_user_write_response(curr_connection, 0)
        end if
        if command=3 then # Command 3 received -> Boot to DFU mode
            call system_reset(1)
        end if
        if command=4 then  # Command 4 received -> power up the SPI flash
            #pull power and chip select pins up
            call hardware_io_port_write(1,flash_power,flash_power)
            call attributes_user_write_response(curr_connection, $0)
        end if
        if command>4 then # Unknown command -> report application error code : 0x80
            call attributes_user_write_response(curr_connection, $80)
        end if

    end if
end if
```

```
# Check if OTA data attribute is written which carries the firmware update
# and store the data to the external SPI flash
if handle = ota_data then

    # NOTE: when programming page, address cannot wrap over 256 byte boundary.
    # This must be handled in the remote DFU application
    # This is write no response attribute, no need to handle response to other end
    # TODO: handle zero length writes
    spi_response(0:1)=2 #page program command

    #flip endianness for address
    tmp(0:4)=dfu_pointer
    spi_response(1:1)=tmp(2:1)
    spi_response(2:1)=tmp(1:1)
    spi_response(3:1)=tmp(0:1)

    # enable SPI flash write mode
    #write enable
    call hardware_io_port_write(0, flash_cs, 0) # assert flash cs
    call hardware_spi_transfer(0,1,"\x06")
    call hardware_io_port_write(0, flash_cs, $ff) # deassert flash cs

    #write data
    call hardware_io_port_write(0, flash_cs, 0) # assert flash cs
    call hardware_spi_transfer(0,4,spi_response(0:4))

    #send data in next transfer, leave chip select asserted
    call hardware_spi_transfer(0,value_len,value_data(0:value_len))
    call hardware_io_port_write(0, flash_cs, $ff) # deassert flash cs

    #it can take up to 800 us for full page to program
    #loop couple of times for write to complete
    call hardware_io_port_write(0, flash_cs, 0) # assert flash cs
    #start polling
    call hardware_spi_transfer(0,2,"\x05\x00")(spiresult,channel,spi_len,spi_response(0:2))
    a=spi_response(1:1)
    while a&1
        call hardware_spi_transfer(0,1,"\x00")(spiresult,channel,spi_len,spi_response(0:1))
        a=spi_response(0:1)
    end while
    call hardware_io_port_write(0, flash_cs, $ff) # deassert flash cs
```

```
        #increase DFU offset
        dfu_pointer=dfu_pointer+value_len
    end if
end
```

An additional event handler is needed to check if the flash writes are ready and more data can be received from the remote end. The event handler below checks if the flash is ready and waits if it's not. Once the flash is ready a status code 0 is sent to the remote device indicating that more data can be received. In case of a flash error, code 0x90 is sent to the application.

```
# Software timer expired event handler
# Poll flash and if it's ready, and send response to the remote device (DFU application)
event hardware_soft_timer(handle)

    if handle = 0 then
        call hardware_io_port_write(0, flash_cs, 0) # assert flash cs
        call hardware_spi_transfer(0,2,"\x05\x00")(spiresult,channel,spi_len,spi_response(0:3))
        call hardware_io_port_write(0, flash_cs, $ff) # deassert flash cs

        # Check if MAX retries have been reached
        if (retry_counter < max_retries) then
            # Increase retry counter
            retry_counter = retry_counter + 1
        else
            # Could not talk to the flash : Report error core 0x90
            call attributes_user_write_response(curr_connection, $90)
        end if
        # Flash was not ready - check again later
        if spi_response(1:1) & 1 then
            call hardware_set_soft_timer(6000,0,1)
        else
            # Flash was ready, send response to the remote device (DFU application)
            call attributes_user_write_response(curr_connection, 0)
        end if

    end if
end
```

Note:

The description of the BGScript functions and events can be found from the *Bluetooth Low Energy Software API reference* document.

2.4.7 Partial OTA update with 128kB flash

Below is an example of a project file for a partial OTA update. This project files generates an OTA update file which contains only the GATT database and a BGScript application.

```
<?xml version="1.0" encoding="UTF-8" ?>
<project>
    <gatt in="gatt.xml" />
    <hardware in="hardware.xml" />
    <config in="config128.xml" />
    <script in="ota.bgs" />
    <device type="ble113" />
    <boot fw="bootota" />
    <image out="out-ble113.hex" />
    <ota out="ble113.ota" firmware="true"/>
</project>
```

Figure 6: Project file for partial OTA update

- <config> file defines an additional application configuration file. This file can be used to define additional application properties for the Bluetooth stack and in this case is used to allocate space from the flash for the partial OTA firmware update file
- <ota> tag defines the OTA firmware update file which is the actual firmware update file uploaded to the device over a Bluetooth LE connection. When you want to create a partial OTA firmware update, which excludes the Bluetooth Low Energy stack and only contains GATT database and possible BGScript application use **firmware="false"** option within the <ota> tags.

An example of the application configuration file is shown below to be used with the above project file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<config>
    <!-- Allocates 5120 bytes of flash for OTA update -->
    <user_data size="0x1400" />
    <!-- Increase script time-out to allow flash erase in loop -->
    <script_timeout value="10000" />
</config>
```

Figure 7: Application configuration file for partial OTA update

- <user data> configuration allocates 5kB of flash for user data from the built-in flash memory. The OTA firmware update will be uploaded and stored to this flash space and you need to make sure you allocate enough flash for the OTA firmware update so it will fit into user data space.
- <script timeout> tag defines a timeout value for a BGScript while-loops and how many iterations they can run. Since the flash needs to be erased as part of the OTA update the value is increased from the default value.

Note:

No special hardware configuration is needed with the partial OTA update, since the internal flash is used.

2.5 OTA Update with Products with 256kB Flash

This section discusses how to do the OTA firmware update with products that have built-in 256kB flash memory. These products are BLE113-A-M256K Bluetooth Low Energy Module and BLE121LR Bluetooth Low Energy Module.

Products that have 256kB built-in flash can support the full as well the partial OTA update firmware update.

2.5.1 Including OTA Support in Your Project

Below is an example of a project file for BLE113-A-M256K with OTA firmware update support.

```
<?xml version="1.0" encoding="UTF-8" ?>
<project>
    <gatt in="gatt.xml" />
    <hardware in="hardware256.xml" />
    <config in="config256.xml" />
    <script in="ota256.bgs" />
    <device type="ble113-m256k" />
    <boot fw="bootota" />
    <image out="out-ble113-m256k.hex" />
    <ota out="BLE113-256.ota" firmware="true"/>
</project>
```

Figure 8: Project file with OTA firmware update support

- <config> file defines an additional application configuration file. This file can be used to define additional application properties for the Bluetooth stack and in this case is used to allocate space from the flash for the partial OTA firmware update file
- <ota> tag defines the OTA firmware update file which is the actual firmware update file uploaded to the device over a Bluetooth LE connection. When you want to create a partial OTA firmware update, which excludes the Bluetooth Low Energy stack and only contains GATT database and possible BGScript application use **firmware="false"** option within the <ota> tags. In case you want to make a full firmware update use **firmware="true".**

2.5.2 Application Configuration

An example of the application configuration file is shown below to be used with the above project file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<config>
    <!-- Allocates 128kB of flash for OTA update -->
    <user_data size="0x20000" />
    <!-- Increase script timeout from default to allow flash erase in loop -->
    <script_timeout value="10000" />
</config>
```

Figure 9: Application configuration file for full OTA update

- <user data> configuration allocates 128kB of flash for user data from the built-in flash memory. The OTA firmware update will be uploaded and stored to this flash space. For partial OTA update smaller flash allocation can be used.
- <script timeout> tag defines a timeout value for a BGScript while-loops and how many iterations they can run. Since the flash needs to be erased as part of the OTA update the value is increased from the default value.

Note:

No special hardware configuration is needed with the partial OTA update, since the internal flash is used.

A partial OTA update can also be used with products that have built-in 256kB flash. In this case **firmware="false"** option should be used in the project file and less flash space than 128kB can be allocated in the application configuration file.

2.5.3 GATT Service for OTA

Please see chapter: 2.4.2.

2.6 BGScript Application for Internal Flash

A separate BGScript example is included in the SDK for OTA update using internal flash. This application is slightly simpler since there is no need to manage the external SPI flash.

2.6.1 Boot Event and Initializations

The code is executed when the device is powered-up and it enables advertisements so the local can be discovered and connected by remote devices. The code also erases the internal flash area dedicated for OTA update.

```
# Boot event listener - erases internal flash and starts advertisements
event system_boot(major, minor, patch, build, ll_version, protocol_version, hw)

    # Erase internal flash dedicated for OTA
    # For this to work, the script timeout has to be increased from the default
    # value in application configuration file (config.xml)
    erase_page = 0
    while erase_page < max_erase_page
        call flash_erase_page(erase_page)
        erase_page = erase_page + 1
    end while

    # Set device to advertisement mode and allow undirected connections
    call gap_set_mode(gap_general_discoverable, gap_undirected_connectable)

    # Initialize the DFU pointer
    dfu_pointer = 0
end
```

2.6.2 Handling OTA Commands and Data

To handle the incoming OTA commands and data from the remote device (OTA application) an event handler for received data must be written. The data received from the remote end can be handled with the **`attributes_value`** (…) event listener, which will catch an event whenever a GATT characteristic is written.

In the example application the handler only checks of the **OTA Control Point Attribute** or **OTA Data Attribute** are written. The **OTA Control Point Attribute** carries commands such as flash erase or DFU boot and the **OTA Data Attribute** carries the actual firmware update.

The event handled code is below and more detailed information how the code works can be found from the comments.

```
# Incoming data event listener
# Handles OTA Control Point Attribute (commands) and OTA Data Attribute (firmware update) writes
# and performs the necessary actions
event attributes_value(connection, reason, handle, offset, value_len, value_data)
    # Save connection handle, is always 0 if only slave
    curr_connection = connection

    # Check if OTA control point attribute is written by the remote device and execute the command
        # Command 0 : N/A
        # Command 1 : N/A
        # Command 2 : N/A
        # Command 3 : Boot to DFU mode
        # Command 4 : N/A
    # In case of errors application error code 0x80 is returned to the remote device
    # In case the flash comms fails error code 0x90 is returned to the remote device
    if handle = ota_control then
        # Attribute is user attribute, reason is always write_request_user
        if value_len > 1 || offset > 0 then
            # Not a valid command -> report application error code : 0x80
            call attributes_user_write_response(connection, $80)
        else
            command = value_data(0:1)

            if command > 4 then # Unknown command -> report application error code : 0x80
                call attributes_user_write_response(curr_connection, $80)
            else
                if command = 3 then # Command 3 received -> Boot to DFU mode
                    call system_reset(1)
                else
                    # Other commands are not used, but still accepted in order
                    # to be compatible with the external flash OTA
                    # implementation
                    call attributes_user_write_response(curr_connection, $0)
```

```
                end if
            end if
        end if
    end if

    # Check if OTA data attribute is written which carries the firmware update
    # and store the data to the internal flash
    if handle = ota_data then
        call flash_write_data(dfu_pointer, value_len, value_data(0:value_len))
        dfu_pointer = dfu_pointer + value_len
    end if
end
```

Note:

The description of the BGScript functions and events can be found from the *Bluetooth Low Energy Software API reference* document.

2.7 Compiling and Installing the Firmware

2.7.1 Using BLE Update Tool

When you want to test your project, you need to compile the hardware settings, the GATT data base 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* Low Energy Module using a CC debugger tools

In order to compile and install the project:

1. Connect the Bluetooth Low Energy Development Kit to the PC via the DEBUGGER port
2. Make sure the DEBUGGER switch on the development kit is tirned to MODULE position
3. Press the reset DEBUGGER button and make sure the DEBUGGER led turns green
4. Start **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 **BLE112-project.bgproj**)
7. Press **Update**

BLE Update tool will compile the project and install it into the target device.

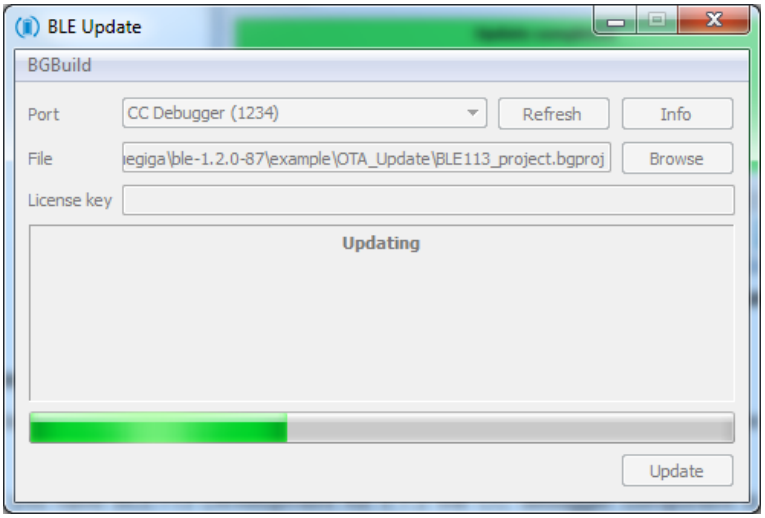

Figure 10: Compile and install with BLE Update tool

Note:

You can also double clikc the .BGPROJ file and it will automatically open the BLE Update tool.

The **View Build Log** opens up a dialog that shows the bgbuild compilere output and the RAM and Flash memory allocations.

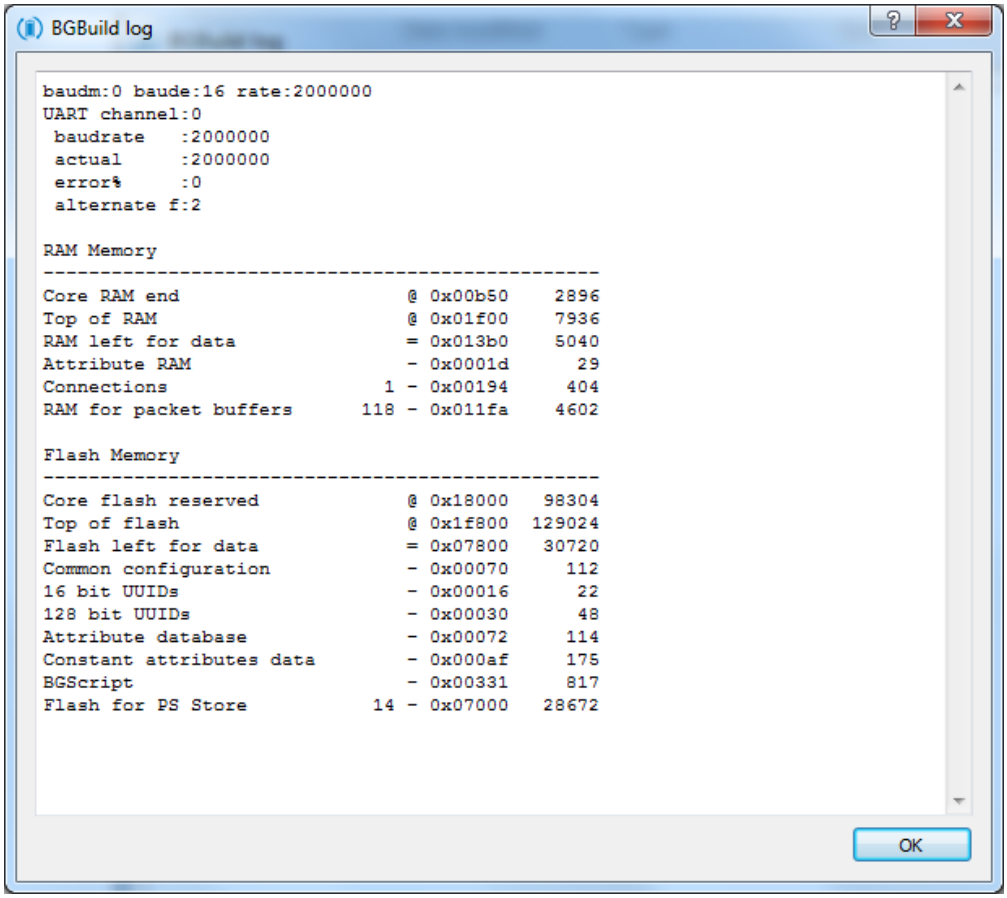

Figure 11: BLE Update build log

2.7.2 Compiling Using 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 to the BLE112 or BLE113.

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>*

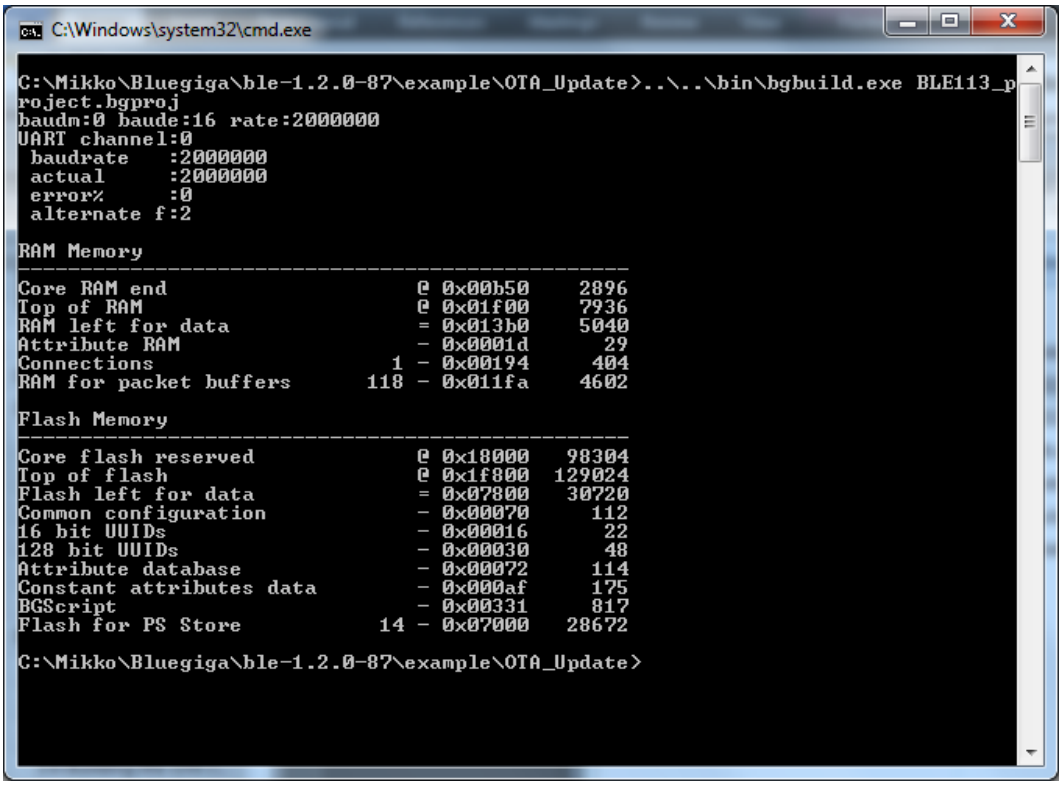

Figure 12: Compiling with BGBuild.exe

If the compilation is successful a .HEX file is generated, which can be installed into a *Bluetooth* Low Energy Module.

On the other hand if the compilation fails due to syntax errors in the BGScript or GATT files, and error message is printed.

3 Testing the OTA Update with BLEGUI

3.1 Using BLEGUI

This section describes how to test the OTA update example using BLEGUI software.

BLEGUI is a simple PC utility that can be used to control a Bluegiga *Bluetooth* Low Energy device over UART or USB. BLEGUI software sends the BGAPI commands to the device and parses the responses and has a simple user interface to display device data.

3.1.1 Discovering the OTA Device

- Connect for example a BLED112 USB dongle to your PC
- Make use the USB/CDC driver gets installed and a Virtual COM port gets created
- Open BLEGUI software and attach the device in the virtual COM port to the BLEGUI

As soon as the OTA example device is powered on it starts to advertise. A BLED112 USB dongle can for example be used to scan for the sensor.

- Enable **Active Scanning**
- Press **Set Scan Parameters**
- Select **Generic** scan mode
- Press **Scan**

If the OTA device is powered on and the OTA example application is installed to is you should see the device in the BLEGUI software.

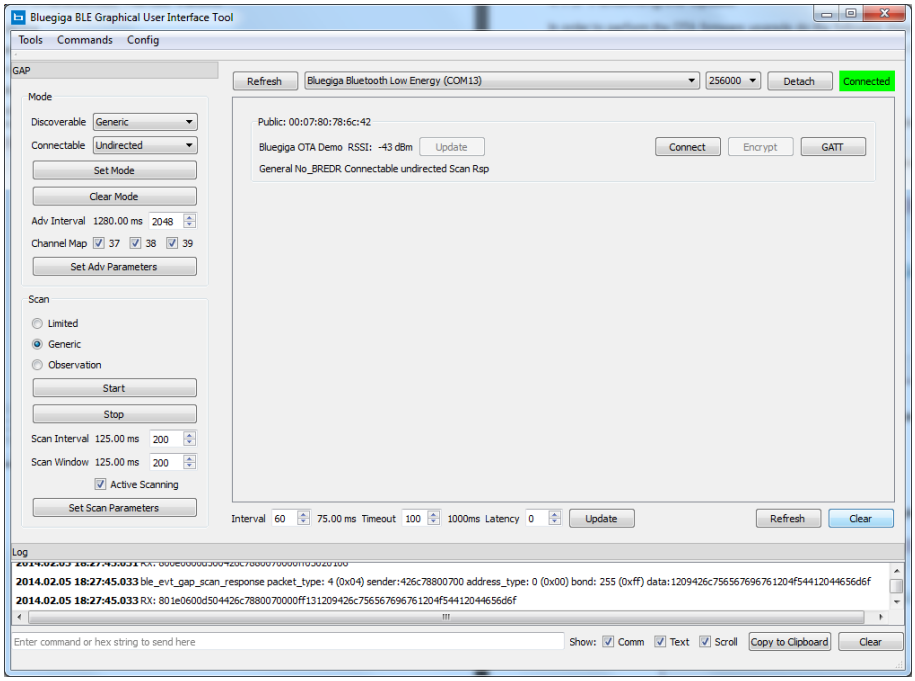

Figure 13: Discovering the OTA device

3.1.2 Checking the OTA Characteristic Handle Values

- **Connect** the OTA device
- Perform a GATT **service discovery**
- Select the OTA service (UUID: 1d14d6ee-fd63-4fa1-bfa4-8f47b42119f0)
- Perform a **descriptors discovery**
- Note the characteristic handle values for the **OTA Control Point Attribute** and **OTA Data Attribute** (15 and 18 in the example application)

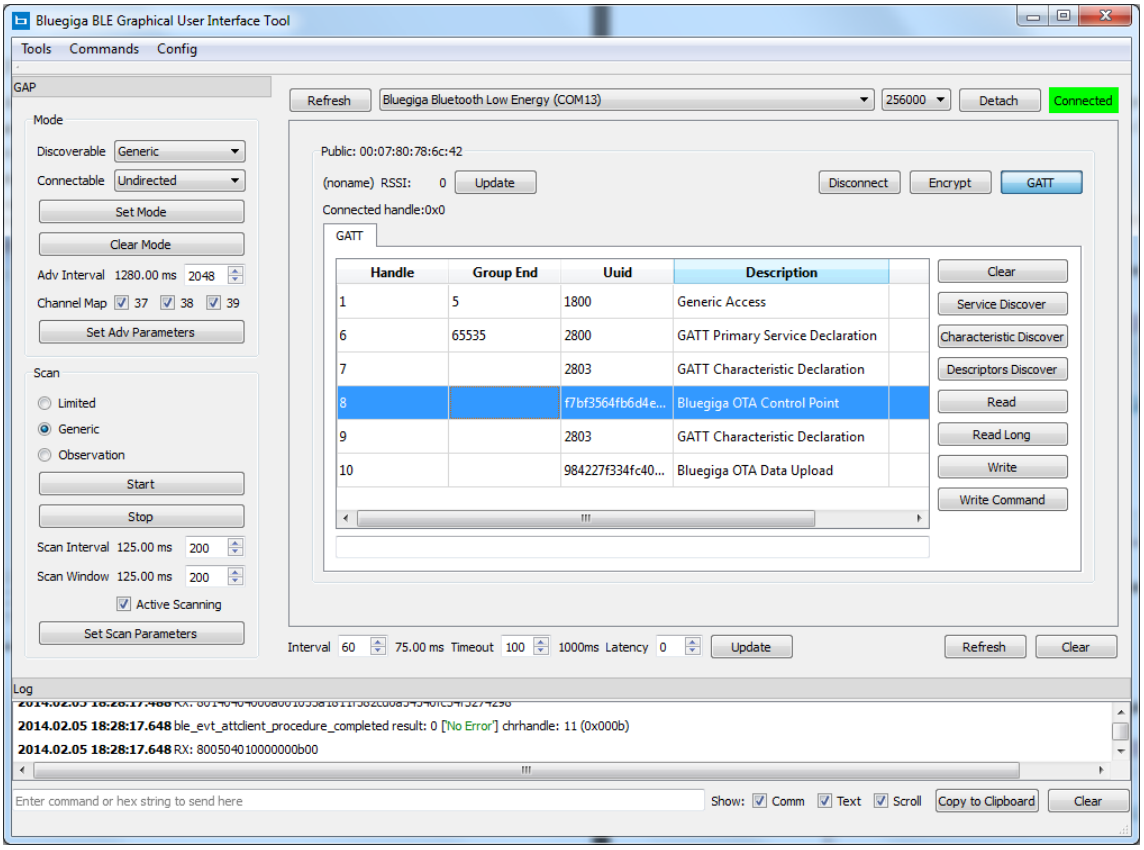

Figure 14: OTA characteristic handle values

3.1.3 Performing the Update

In order to perform the OTA firmware upgrade do the following steps

- Go to **Commands -> DFU -> Over the Air** Upgrade menu
- Select the desired firmware file (.OTA file)
- Select the connection handle of the device you want to update
    - Double check that the connection exists
- Write the **OTA Control Point Attribute** and **OTA Data Attribute handles** to the dialog
- Press **Upload**

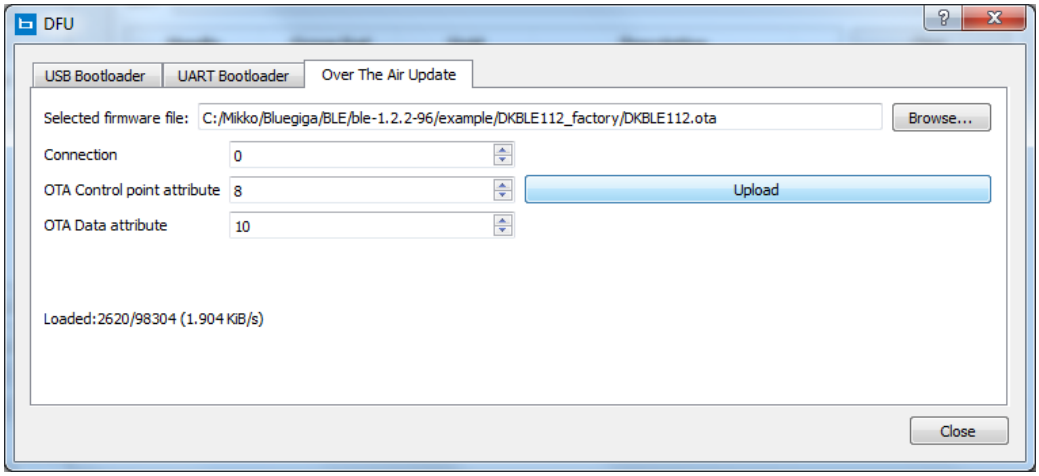

Figure 15: Performing OTA Update

3.1.4 Verifying the Update

Wait for the update the finish and verify you see a message **OTA Completed** message in the dialog and not an error message.

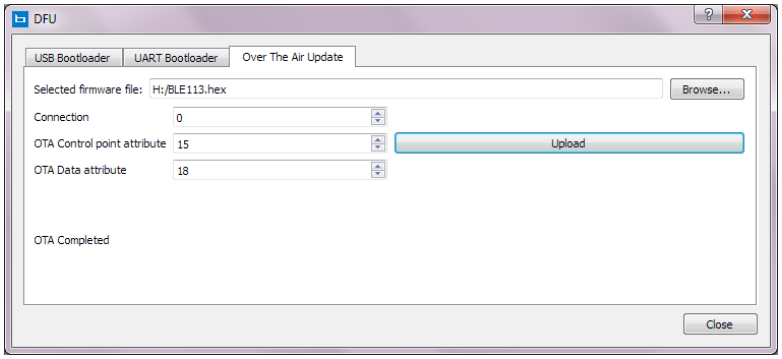

Figure 16: Successful OTA Firmware Update

4 Current Consumption

The average current consumption of BLE113 during OTA firmware update with slow clock disabled is 10.5 mA. The peak current is 27 mA. This assumes maximum TX power and the use of external DC/DC converter.

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com