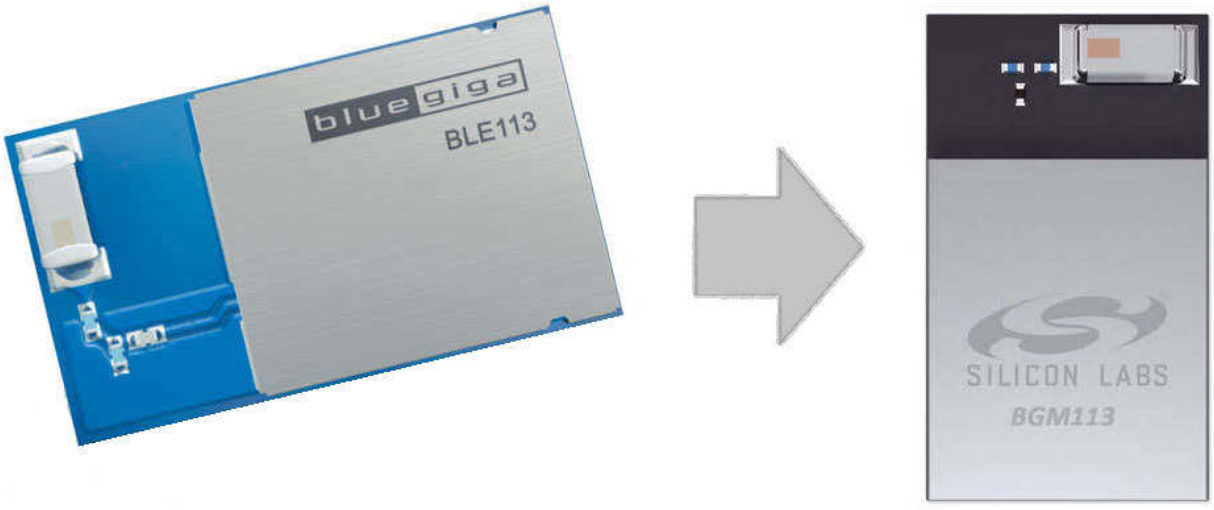
AN1036: BLE113 to BGM113 Migration Guide

This document provides guidance on how to migrate from the BLE113 Bluetooth® Module to the Blue Gecko BGM113 Bluetooth® Module. It highlights the major differences in terms of high-level features, module configuration, hardware design requirements, and software APIs with the purpose of ensuring the easiest possible migration.

KEY POINTS

- SDK and Tools
- BLE113 vs. BGM113 differences
  - Power supply
  - PCB layout
  - Configuration
  - API
- Simplifies migration



1. Introduction

This document provides guidance on how to migrate from the BLE113 Bluetooth Module to the Blue Gecko BGM113 Bluetooth Module. Section 2. SDK and Tools covers the SDK and tools that are used when working with BGM113. Section 3. Feature Differences summarizes the feature differences between BLE113 and BGM113. The hardware related differences are covered in Section 4. Hardware Related Differences.

BGM113 is designed to be footprint-compatible with BLE113 but there are some differences for example in the GPIO assignments that may prevent BGM113 being used as a direct drop-in replacement for BLE113.

Section 5. Software Related Differences explains the differences in project configuration options between BLE113 and BGM113. Section 6. Porting Application Code from BLE113 to BGM113 compares the BLE113 and BGM113 in terms of the application programming interface (BGAPI).

2. SDK and Tools

As of September 2016, the latest SDK for BLE113 is version 1.4.2.-130.

For Blue Gecko based products (including BGM113) the latest SDK is version 2.0.0 (build 1391). This SDK is integrated into Simplicity Studio v4. After installing the latest version of Simplicity Studio v4 you can add the Bluetooth Smart SDK by running the *Update Software* function in Studio.

More information about the development tools for Blue Gecko modules can be found in document QSG108: Blue Gecko Bluetooth® Smart Software Quick-Start Guide.

The BLE113 SDK includes a test program named BLE GUI that can be used to test different features of the BLExxx modules. For BGM113 development the corresponding tool is **BGTool**.

Note: BLEGUI cannot be used with BGM113.

3. Feature Differences

This section describes main feature differences between BLE113 and BGM113.

Table 3.1. Feature Differences between BLE113 and BGM113

Feature	BLE113	BGM113
Bluetooth features	Bluetooth 4.0	Bluetooth 4.2[1]
Max TX power	0 dBm	+3 dBm
Sensitivity	–93 dBm	–93 dBm
TX current (at 0 dBm)	18.2 mA (14.3 mA with DC-DC)	8.8 mA
RX current	14.3 mA	8.7 mA
Sleep current	0.4 µA (Sleep mode 3)	1.4 µA (EM2), 1.1 µA (EM3 stop current)
MCU	Single-cycle 8051-compatible core	ARM® Cortex®-M4, 38.4 MHz
RAM	8 kB	32 kB
FLASH	128 kB / 256 kB	256 kB
Number of GPIO	19	14
DC-DC	No DC-DC	Integrated DC-DC converter
Supply voltage range	2V to 3.6V	1.85 V to 3.8 V
Operating temperature range	–40 °C to 85 °C	–40 °C to 85 °C
Dimensions	9.15 × 15.75 × 2.1 mm	9.15 × 15.73 × 1.9 mm

Note:
1. Software upgradable to Bluetooth 4.2.

4. Hardware Related Differences

This section describes the hardware-related differences between BLE113 and BGM113.

4.1 Pinout Compatibility

The pinout for BGM113 Module is shown in the figure below. BGM113 is **footprint-compatible** with BLE113.

Ground and power supply pins, reset, programming and GPIO pins are mapped to the same pins.

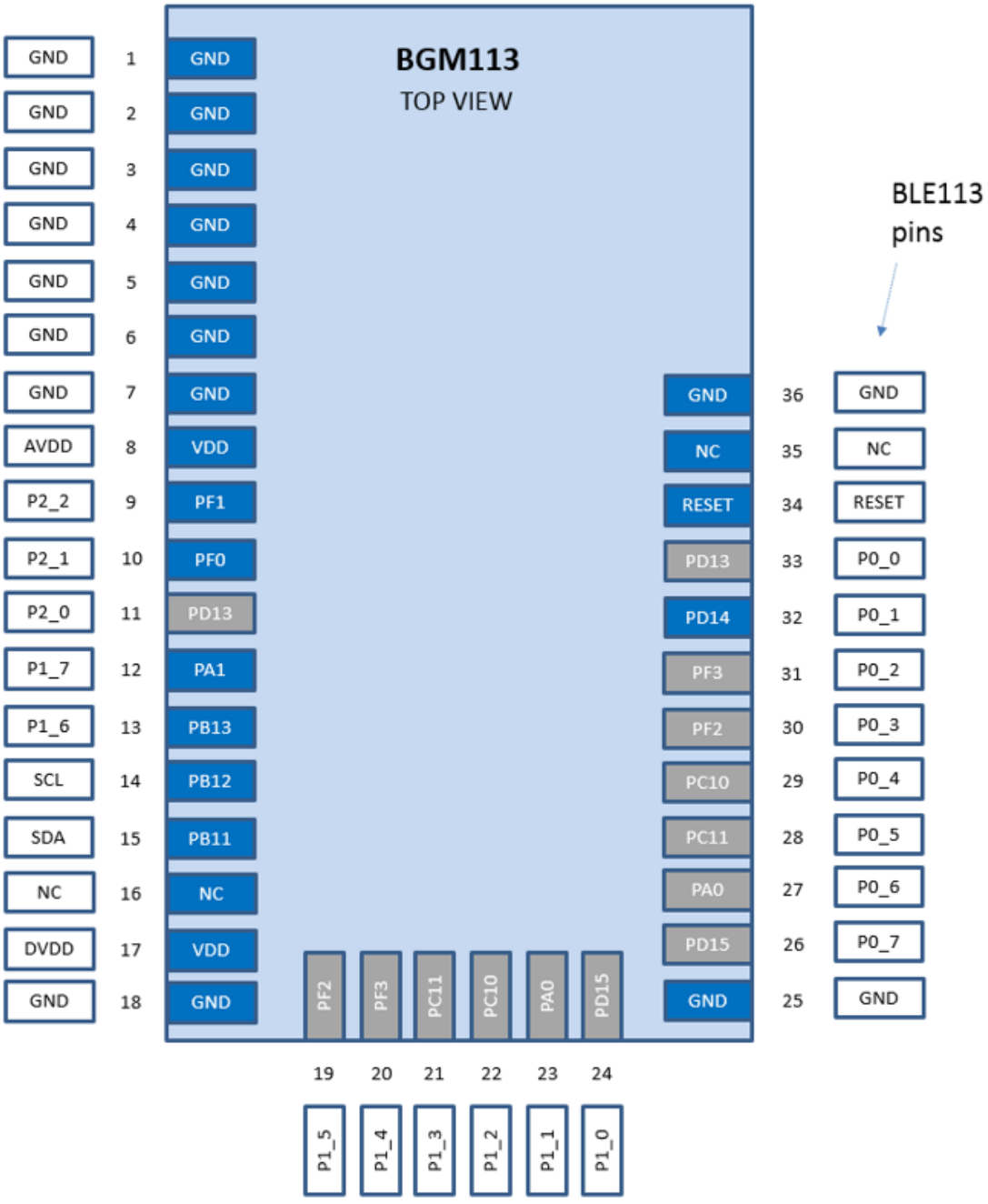


Figure 4.1. BLE113 and BGM113 Pinouts

Note: Some of the GPIO signals in BGM113 are connected to multiple pins. This means that the maximum number of individual GPIO lines is smaller in BGM113 than in BLE113. These pins are shaded gray in the figure above.

4.1.1 Debug and Programming Pins

The debug port is mapped to the same pins on both BGM113 and BLE113. This allows using the same debug connector for both modules. However, it is important to note that the programming interface is completely different. BGM113 uses an ARM SWD interface while BLE113 uses a TI CC debugger interface.

The debugger pins for BLE113 and BGM113 are indicated in the table below.

Table 4.1. BLE113 and BGM113 Debugger Pins

Pin Number	BLE113 Pin Name / Function	BGM113 Pin Name / Function
9	P2_2 / PROG_DC	PF1 / SWDIO
10	P2_1 / PROG_DD	PF0 / SWCLK

4.1.2 Power Supply and Reset Pins

Supply voltage is connected to pins 8 and 17. BLE113 has separate pins for analog and digital supply (AVDD, DVDD) while BGM113 has only single supply voltage VDD that is common for analog and digital domains.

Reset pin is active low and has internal pull-up resistor both in BGM113 and BLE113.

4.1.3 GPIO Pins

GPIO signals are mapped to same pins in BGM113 and BLE113. However, as shown in Figure 4.1 BLE113 and BGM113 Pinouts on page 4 some GPIO signals are mapped to two module pins in BGM113. This may potentially cause some conflicts when replacing BLE113 with BGM113.

Full list of all GPIO pins in BLE113 and BGM113 is included in Section 7. Appendix — BLE113 and BGM113 Pin Comparison. Note that in BLE113 pins 14 and 15 are hardwired for I^2C but in BGM113 these can be used either for I^2C or as GPIO signals.

4.1.4 Unused Pins

Unused pins must not be left floating. In BLE113 the internal pull-up or pull-down resistors defined will effect the whole port in question. For example, all pins in port P2 can be pulled either high or low. In BGM113 the pull-resistors can be individually programmed for each pin.

4.1.5 Pins with High Current Drive Capability

BLE113 has two I/O pins with 20 mA drive strength, these are pins **P1_0** and **P1_1**. In BGM113 there are no dedicated pins with higher drive strength.

4.1.6 Pins with Dedicated Functions

In BLE113 the I^2C SCL and SDA signals are fixed to pins 14 and 15. In BGM113 the I^2C signals can be freely mapped to any available GPIO. They can be mapped to same pins as in BLE113 for compatibility. If I^2C is not needed then these pins can be used by the application as general purpose I/O.

4.2 Peripherals

This section describes differences between peripherals in BLE113 and BGM113.

4.2.1 UART/SPI

BLE113 and BGM113 both have two USART blocks that can be configured either into UART or SPI mode. In BLE113 there are two alternate pin mappings for USART pins. The signals are mapped as a group, meaning that for example all UART0 signals use either **alt1** or **alt2** mapping.

In BGM113 the USART signals can be routed to any available GPIO individually which allows more flexible pin assignments.

4.2.2 ADC

In BLE113 there are eight possible pins that can be used as ADC input, these are **P0_0** – **P0_7**. In BGM113 any available GPIO input can be used as input to ADC.

ADC Reference

The possible ADC references in BLE113 are:

- Internal reference (1.24 V)
- External reference on AIN7 pin
- AVDD pin
- External reference on AIN6 – AIN7 differential input

In BGM113 the possible ADC references are:

- Internal 1.25 V reference
- Internal 2.5 V reference
- Buffered VDD
- Internal differential 5 V reference
- Single ended external reference from pin 6
- Differential external reference, 2x (pin 6 – pin 7)
- Unbuffered 2xVDD

For more details, refer to the *EFR32xG1 Wireless Gecko Reference Manual*.

Measuring Supply Voltage

In BLE113 there is a special ADC channel VDD/3 that can be used to measure supply voltage level with ADC.

In BGM113 there is no possibility to connect a downscaled VDD to the ADC input. However, VDD measurement is still possible by setting the ADC reference at internal 5 V differential and sampling the AVDD channel. For more information, see command `cmd_hardware_read_adc_channel` in the *Bluetooth® Smart Software API Reference Manual*.

4.2.3 Timers

In BLE113 there are three timers accessible by the user application, timer1, timer3 and timer4. These timers have timer/counter/PWM functionality. Timer 1 is 16-bit while timers 3 and 4 are 8-bit.

The BLE113 datasheet lists the possible mappings from timer channels to output pins. For each timer there is two alternate mappings, alt1 and alt2. The total number of timer channels that can be mapped to module pins is 5+2+2 = 9.

BGM113 features two 16-bit general purpose timers with 7 (3+4) compare/capture channels for input capture and compare/ Pulse-Width Modulation (PWM).

In BGM113 the timer channels can be mapped to any available GPIO pin. They do not need to be mapped in groups (like the alt1 / alt2 mapping in BLE113).

In general the timer configuration in BGM113 is at least as flexible as in BLE113. As the timer channels can be freely mapped to available pins it should be not a problem to port a design from BLE113 to BGM113, assuming that there are no conflicts in the GPIO pin assigments (because in BGM113 some signals are routed to multiple pins as discussed earlier).

In addition to the generic 16-bit timers, BGM113 features two special timer peripherals, CRYOTIMER and LETIMER.

The CRYOTIMER is a 32-bit counter which operates on a low frequency oscillator and is capable of running in all Energy Modes and wake up the module from sleep even from the deepest sleep mode EM4.

The LETIMER is a 16-bit down-counter, running on a 32.768 Hz clock. It can keep track of time and output configurable waveforms. LETIMER is available down to sleep mode EM2.

Note: At the time of writing, the timer functions in BGM113 (both the basic timers and LETIMER, CRYOTIMER) are only accessible in C-based applications, using the EMLIB drivers to access the MCU peripherals directly. The timer functions (such as PWM generation) are not accessible via BGAPI or BGScript.

5. Software Related Differences

This section contains information about the software-related differences between BLE113 and BGM113.

5.1 Converting Project Configuration Files

This section explains how to convert BGScript™ project configuration files from BLE113 to BGM113 format.

The set of project structure is same for BLE113 and BGM113:

- *Project_name.bgproj* (defines device type and list of XML files included)
  - *hardware.xml* (hw configurations)
  - *gatt.xml* (GATT database definition)
  - *script.bgs* (BGScript script)

The table below shows two example project files (**.bgproj*), the left side is for BLE113 and the right side is for BGM113.

Table 5.1. BLE113 and BGM113 Debugger Pins

BLE113 Project File	BGM113 Project File
<?xml version="1.0" encoding="UTF-8" ?> <project> <gatt in="gatt.xml" /> <hardware in="hardware.xml" /> <script in="keyboard.bgs" /> <image out="out-ble113.hex" /> <device type="ble113" /> <boot fw="bootuart" /> </project>	<?xml version="1.0" encoding="UTF-8" ?> <project device="bgm113"> <!-- GATT service database --> <gatt in="gatt.xml" /> <!-- Local hardware configuration file --> <hardware in="hardware.xml" /> <!-- BGScript source code --> <script in="keyboard.bgs" /> <!-- Firmware output file --> <image out="bgm113demo.bin" /> </project>

There are some differences that need to be taken into account when converting porting the project file from BLE113 to BGM113:

- In BGM113, target device type is defined in the `<project>` tag
- The BGScript source file is put inside `<scripting>` tag in BGM113 project

Note: `<boot>` option is not available for BGM113.

5.2 Hardware Configuration File

The following subsections describe hardware configuration related differences between BLE113 and BGM113.

5.2.1 GPIO Configuration

In BLE113 projects the `<port>` tag is used to configure settings GPIO settings that are valid for GPIO inputs only. Below is an example from a BLE113 project:

Example: Using the `<port>` Tag for Configuring GPIO Settings

```
<port index="0" tristatemask="0" pull="down" />
```

The above line configures pull direction to “down” for all pins in bank 0. Note that the pull direction is defined for the whole bank and there is no possibility to use both pull-up and pull-down in the same bank. Tristatemask can be used to disable the pull resistor for selected pins in the bank.

In BGM113 the GPIO configuration is completely different. The following example shows how to configure two output pins (PA6, PA3) and one input pin (PF6):

Example: Configuring Two Output Pins (PA6, PA3) and One Input Pin (PF6)

```
<gpio port="A" pin="6" mode="pushpull" out="1" />
<gpio port="A" pin="3" mode="pushpull" out="0" />
<gpio port="F" pin="6" mode="input" out="0" interrupt="both" />
```

In BGM113 projects it is possible to configure both input and output pin settings in the *hardware.xml* file. Pull-up resistors are defined individually for each pin and for GPIO outputs it is also possible to define initial state.

Basic GPIO settings for inputs and outputs can be also configured at runtime using BGScript or BGAPI commands. The function is hardware_configure_gpio, see the *Bluetooth® Smart Software API Reference Manual* for more details.

Detailed description of the available GPIO pin configuration options is found in *UG119: Blue Gecko Bluetooth® Smart Device Configuration Guide*.

5.2.2 UART Configuration

The basic usage of UART for BGAPI commands or application specific input/output is similar in BLE113 and BGM113. However, in the UART configuration there are several differences that need to be taken into account when porting BLE113 project to BGM113.

Below are examples of UART configuration, the first for BLE113 and the second for BGM113:

Example: UART Configuration with BLE113

```
<usart channel="1" alternate="1" baud="115200" endpoint="none" />
```

Example: UART Configuration with BGM113

```
<uart index="1" baud="115200" flowcontrol="false" bgapi="false" />
```

Some differences in UART configuration between BLE113 and BGM113 are listed below:

- Tag name is `<usart>` in BLE113 and `<uart>` in BGM113
- BLE113 has parameter “channel” while BGM113 has parameter “index” to select UART instance
- “Flow” parameter in BLE113 is replaced with “flowcontrol” in BGM113

For detailed list of UART configuration options, see CONFIG_GUIDE.

Note that the BLE113 UART example above has parameter “alternate” that is used to select between two alternative pin mappings. In BGM113 there is no such parameter because the UART pins can be individually mapped to any available GPIO.

Below is an example of UART configuration for BGM113 that specifies the location of **RTS** and **CTS** pins explicitly:

Example: UART Configuration with BGM113 with RTS and CTS pins explicitly defined

```
<uart index="1" baud="115200" flowcontrol="true" rts_pin="PA3" cts_pin="PA2" bgapi="true"/>
```

5.2.3 Wakeup Pin

The wakeup pin functionality is similar in both BLE113 and BGM113. There are some slight differences in the syntax how the wakeup pin is defined.

Example: Wakeup pin definition with BLE113 (pin P0_0)

```
<wakeup_pin enable="true" port="0" pin="0" state="up" />
```

Example: Wakeup pin definition with BGM113 (pin PD13)

```
<wake_up port="d" pin="13" state="up" />
```

Functionally both of the above examples do the same: they configure wakeup pin to P0_0 / PD13 and set the polarity as active-high. Only the syntax is a bit different. For full details, see *UG119: Blue Gecko Bluetooth® Smart Device Configuration Guide*.

5.2.4 Host Wakeup Pin

The function of the host wakeup pin is similar both in BLE113 and BGM113. The syntax for configuring this pin is a bit different, see examples below. For full details, please refer to the *UG119: Blue Gecko Bluetooth® Smart Device Configuration Guide*.

Example: Configure P1_7 as host wakeup pin with BLE113

```
<host_wakeup_pin enable="true" port="1" pin="7" state="up" />
```

Example: Configure PA1 as host wakeup pin with BGM113:

```
<host_wake_up port="a" pin="1" state="up" />
```

5.2.5 Sleep Enable

Configuring sleep enable is almost identical in BLE113 and BGM113. The difference is that BGM113 does not use the attribute `max_mode` at all. Following examples show how to enable sleep for BLE113 and BGM113, respectively.

Example: Enabling sleep with BLE113 (max mode 2, powermode 3 not enabled)

```
<sleep enable="true" max_mode="2" />
```

Example: Enabling sleep with BGM113 (max mode is implicitly EM2)

```
<sleep enable="true" />
```

5.2.6 TX Power Setting

In BLE113 projects the TX power can be set in the hardware configuration file using the `<txpower>` tag. Alternatively, the BLE113 TX power can be set dynamically at runtime by calling function `hardware_set_txpower`.

Example: Set maximum power level (0dBm) in BLE113 at runtime

```
call hardware_set_txpower(14)
```

In BGM113 there is **no option to configure TX power statically** in the configuration files. You will need to set the TX power level dynamically using function `system_set_tx_power`. For details, see the *Bluetooth® Smart Software API Reference Manual*.

Example: Set maximum power level (+3 dBm) in BGM113 at runtime

```
call system_set_tx_power(30)
```

Note the difference in the units when adjusting TX power. BLE113 uses values 0..14 where 0 is the lowest value (about -24 dBm) and 14 is the highest setting that equals roughly to 0 dBm.

In BGM113 the parameter is given 0.1 dBm steps. For example, setting 15 corresponds to 1.5 dBm. Function `system_set_tx_power` will return a value that is the actual power setting used by the stack. Note that the power is not adjustable in 0.1 dBm steps, the stack will use a pre-defined power level that is closest to the value that is requested by user.

5.2.7 Obsolete HW Related Settings

Following HW related settings are not applicable in BGM113:

- `<sleeposc>`
- `<slow_clock>`
- `<lock_debug>`
- `<pmux>`

6. Porting Application Code from BLE113 to BGM113

This chapter describes the differences between BGAPI commands in BLE113 and BGM113. Some of the BGAPI commands and events are almost identical between BLE113 and BGM113 stacks. On the other hand, some parts (like GATT service discovery) are quite different in BGM113 compared to BLE113.

The following topics will be covered in this chapter:

- System_boot event
- Enabling advertisements and changing advertising data/parameters
- Opening/closing of connections
- Sending notifications and indications
- Soft timers
- Enabling discovery
- Connecting to a peripheral device
- GATT service discovery
- Subscribing to notifications

6.1 System Boot

Both in BLE113 and BGM113, ”system_boot” is the first event that is raised after module reboots. The BGScript prototypes are shown below:

```
# BLE113:
event system_boot(major, minor, patch, build, ll_version, protocol_version, hw)
# BGM113:
event system_boot(major, minor, patch, build, bootloader, hw)
```

In both cases the first four parameters that identify the stack version (e.g. 1.0.2-755) are similar. The last parameters related to HW version etc are slightly different.

6.2 Enabling Advertisements

Starting and stopping of advertisements is basically identical in BLE113 and BGM113, it is done using command `gap_set_mode`:

```
# BLE113:
call gap_set_mode(discover, connect)(result)
# BGM113:
call le_gap_set_mode(discover,connect)(result)
```

The only difference here is that the BGM113 command has a prefix “le_” in the name (“le” stands for “low energy”). This same convention is used throughout most BGM113 commands and events.

Setting of advertising parameters and advertising data is almost identical in BLE113 and BGM113. The function and the parameters are the same (excluding the “le_” prefix for BGM113):

```
# BLE113:
call gap_set_adv_parameters(adv_interval_min, adv_interval_max, adv_channels)(result)
call gap_set_adv_data(set_scanrsp, adv_data_len, adv_data_data)(result)
# BGM113:
call le_gap_set_adv_parameters(interval_min, interval_max, channel_map)(result)
call le_gap_set_adv_data(scan_rsp,adv_data_len, adv_data_data)(result)
```

6.3 Connection Status Changes

The BGAPI events that indicate changes in connection status are completely different in BLE113 and BGM113. This will likely require some porting effort basically for any application that accepts incoming connections.

In BLE113, the connection status change is indicated with one single event “connection_status” that has following prototype:

```
# BLE113:
event connection_status(connection, flags, address, address_type, conn_interval, timeout, latency, bonding)
```

From the `flags` parameter application can decode what was the reason why this event was raised.

In BGM113 there is on `connection_status` event at all. Instead, the connection status changes are indicated with several different events that have are targeted for more specific use (e.g. handle incoming connection). Some relevant events in BGM113 stack are listed below:

```
# BGM113:
event le_connection_opened(address,address_type,master,connection,bonding)
event le_connection_closed(reason,connection)
event le_connection_parameters(connection,interval,latency,timeout,security_mode)
```

When porting application from BLE113 to BGM113, the functionality in `connection_status` event handler needs to be moved to suitable events that are provided by the BGM113 API.

6.4 Sending Notifications or Indications

Many applications use notifications or indications to push data to client. The API functions that are used to send a notification in BLE113 and BGM113 are shown below.

```
# BLE113:
call attributes_write(handle, offset, value_len, value_data)(result)

# BGM113:
call gatt_server_send_characteristic_notification(connection,characteristic,value_len, value_data)(result)
# Optional:
call gatt_server_write_attribute_value(attribute, offset, value_len, value_data)(result)
```

There is a fundamental difference in how notifications are implemented in BLE113 and BGM113. In BLE113 the application simply writes to the local GATT database using `attributes_write` command. If there is a client connected that has subscribed to notifications on this attribute then stack will automatically generate a notification.

In BGM113 there is no automatic triggering of notifications. The application may update the value in local database, using command `gatt_server_write_attribute_value`. To trigger notification, the application must explicitly use command `gatt_server_send_characteristic_notification`.

In many cases, if the application needs to just send a notification and the client does not need to be able to read the value from database then it is enough to replace the call to **attribute_write** with `gatt_server_send_characteristic_notification`.

The above description explains how notifications are generated in BGM113, note that the same applies to *indications*, too.

6.5 Soft Timers

The API call to configure a soft timer is same both in BLE113 and BGM113:

```
call hardware_set_soft_timer(time, handle, single_shot)(result)
```

For both modules the timer interval is defined in hardware clock ticks so that 1 second is equal to 32768 ticks.

In BLE113 there can be only one repeating timer at a time. If you start a new repeating timer, the existing repeating timer will be removed (if such timer exists). In BGM113 the number of repeating soft timers is not limited.

6.6 Enabling Discovery

The API calls for setting scan parameters, starting discovery and ending discovery are listed below for both BLE113 and BGM113. The prototypes are identical except for the “le_” prefix that is used in BGM1113.

```
# BLE113:
call gap_set_scan_parameters(scan_interval, scan_window, active)(result)
call gap_discover(mode)(result)
call gap_end_procedure()(result)

# BGM113:
call le_gap_set_scan_parameters(scan_interval,scan_window,active)(result)
call le_gap_discover(mode)(result)
call le_gap_end_procedure()(result)
```

The default scan parameters are different for BLE113 and BGM113. This must be taken into account if the application does not explicitly call `gap_set_scan_parameters` but uses the stack defaults. In BLE113 the defaults for scan interval and window are (75, 50) ms and in BGM113 the default values are (10,10) ms.

The event that is raised for each scan response is similar in both modules:

```
# BLE113:
event gap_scan_response(rssi, packet_type, sender, address_type, bond, data_len, data_data)
# BGM113:
event le_gap_scan_response(rssi, packet_type, address, address_type, bonding, data_len, data_data)
```

Note that the packet type enumeration is slightly different in BLE113 vs BGM113:

BLE113:

- **0**: Connectable advertisement packet
- **2**: Non-connectable advertisement packet
- **4**: Scan response packet
- **6**: Discoverable advertisement packet

BGM113:

- **0**: Connectable undirected advertising
- **2**: Scannable undirected advertising
- **3**: Non-connectable undirected advertising
- **4**: Scan response

6.7 Opening a Connection

The API call used to open a connection to an advertising device is quite different in BLE113 and BGM113, the function prototypes are shown below:

```
# BLE113:
call gap_connect_direct(address, addr_type, conn_interval_min, conn_interval_max, timeout, latency)(result, con
nection_handle)

# BGM113:
call le_gap_open(address,address_type)(result,connection)
```

In BLE113 the connection parameters are passed as parameters to `gap_connect_direct` while in BGM113 the function `le_gap_open` only takes the target address and address type as parameters.

In BGM113 if you want to change connection parameters then the following API calls can be used:

```
# BGM113:
call le_gap_set_conn_parameters(min_interval,max_interval,latency,timeout)(result)
call le_connection_set_parameters(connection,min_interval,max_interval,latency,timeout)(result)
```

These functions are used to set the same parameters that you would define as part of `gap_connect_direct` command in BLE113. The difference between the two API calls listed above is that `le_gap_set_conn_parameters` sets the default parameters that are used for all subsequent connections while `le_connection_set_parameters` is used to modify parameters for a single conncetion that has already been opened.

The events generated after connection is opened are listed below:

```
# BLE113:
event connection_status(connection, flags, address, address_type, conn_interval, timeout, latency, bonding)

# BGM113:
event le_connection_opened(address,address_type,master,connection,bonding)
event le_connection_parameters(connection,interval,latency,timeout,security_mode)
```

6.8 Closing a Connection

Connection may be closed either unexpectedly (e.g. due to a supervision timeout) or deliberately by the application.

In case of an unexpected connection close the following events are raised in BLE113 and BGM113:

```
# BLE113:
event connection_disconnected(connection, reason)

#BGM113:
event le_connection_closed(reason,connection)
```

Both events listed above include the same information, only the syntax is different.

If the application needs to close an active connection then the following API calls are used in BLE113 and BGM113:

```
# BLE113:
call connection_disconnect(connection)(connection, result)

# BGM113:
call endpoint_close(endpoint)(result,endpoint)
```

Again, the basic usage is the same but just the syntax is different.

6.9 GATT Service Discovery

There are many differences in the way service discovery is done in BLE113 and BGM113. Below is a summary of typical API calls and events related to service discovery for both modules. It is not practical to explain all the differences here, the purpose of this list is just to point reader to the relevant parts in the BGAPI documentation to help porting their application from BLE113 to BGM113.

Finding services in BLE113:

1. Call `attclient_read_by_group_type`.
2. This will generate events of type `attclient_group_found` (one for each primary or secondary service).
3. After last event the stack will raise event `attclient_procedure_completed`.

Finding attributes for given service in BLE113:

1. Call `attclient_find_information` (the handle range is passed as parameters).
2. This will generate events of type `attclient_find_information_found`.
3. After last event the stack will raise event `attclient_procedure_completed`.

Finding services in BGM113:

1. Call `gatt_discover_primary_services` (or `gatt_discover_primary_services_by_uuid`).
2. This will generate events of type `gatt_service` (one for each service).
3. After last event the stack will raise event `gatt_procedure_completed`.

Finding attributes for given service in BGM113:

1. Call `gatt_discover_characteristics` (service handle is passed as parameter).
2. This will generate events of type `gatt_characteristic`.
3. After last event the stack will raise event `gatt_procedure_completed`.

6.10 Subscribing to Notifications or Indications

To subscribe to notifications (or indications) the GATT client needs to write value 0x0001 (or 0x0002) to the client characteristic configuration (CCC) of the given characteristic. Typically the CCC handle value will be +1 or +2 steps relative to the characteristic itself. (Note that the client must not make any assumption that this is the case)

In BLE113, this can be done by calling `attclient_attribute_write` to change the CCC value directly. The client will need to know what is the handle associated with the CCC and this is detected as part of the service discovery (see above).

In BGM113, the client does not need to know the CCC handle. Instead, it will just call `gatt_set_characteristic_notification` and pass the value of the characteristic (not the CCC) to the function. The stack will automatically discover the characteristic client configuration.

Function `gatt_set_characteristic_notification` is used to enable/disable both notifications and indications, this is selected with the 'flags' parameter passed to the function. For details, see *Bluetooth® Smart Software API Reference Manual*.

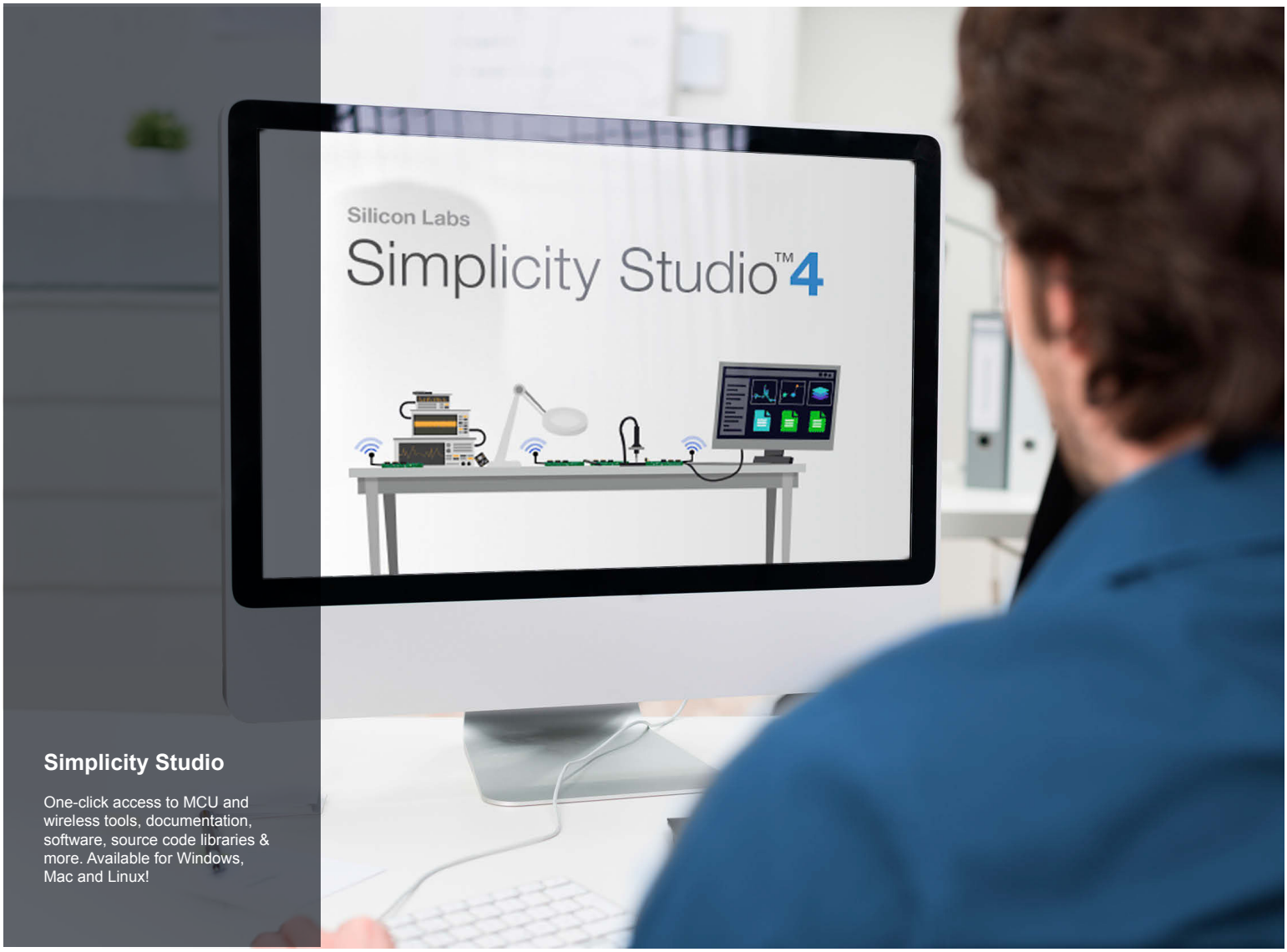
7. Appendix — BLE113 and BGM113 Pin Comparison

The table below lists all pins for both BLE113 and BGM113.

Table 7.1. BLE113 and BGM113 Pin Comparison

Pin Number	BLE113	BLE113 Notes	BGM113	BGM113 Notes
9	P2_2		PF1	
10	P2_1		PF0	
11	P2_0		PD13	Same as pin 33
12	P1_7		PA1	
13	P1_6		PB13	
14	SCL	I^2C only	PB12	
15	SDA	I^2C only	PB11	
19	P1_5		PF2	Same as pin 30
20	P1_4		PF3	Same as pin 31
21	P1_3		PC11	Same as pin 28
22	P1_2		PC10	Same as pin 29
23	P1_1	20 mA	PA0	Same as pin 27
24	P1_0	20 mA	PD15	Same as pin 26
26	P0_7		PD15	Same as pin 24
27	P0_6		PA0	Same as pin 23
28	P0_5		PC11	Same as pin 21
29	P0_4		PC10	Same as pin 22
30	P0_3		PF2	Same as pin 19
31	P0_2		PF3	Same as pin 20
32	P0_1		PD14	
33	P0_0		PD13	Same as pin 11

Notes:
1. BLE113 max number of GPIO lines is 19 (+ 2 dedicated pins for I2C).
2. BGM113 max number of GPIO lines is 14.



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 and Community
community.silabs.com

Disclaimer
Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Labs shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any Life Support System without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

Trademark Information
Silicon Laboratories Inc.® , Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, Clockbuilder®, CMEMS®, DSPLL®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, ISOmodem®, Precision32®, ProSLIC®, Simplicity Studio®, SiPHY®, Telegesis, the Telegesis Logo®, USBXpress® and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

SILICON LABS

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

http://www.silabs.com