Renesas Synergy™ Platform

BLE Framework

R30AN0309EU0104
Rev.1.04
Mar 22, 2018

Introduction

This document enables you to effectively use the Bluetooth® Low Energy (BLE) Framework module in your own design. On completion of this guide, you will be able to add the BLE Framework module to your own design, configure it correctly for the target application, and write code using the included application example code as a reference and efficient starting point. References to more detailed API descriptions and suggestions of other application projects that illustrate more advance uses of the module are available in the *Synergy Software Package (SSP) User's Manual* (see BLE Framework Next Steps section), and are valuable resources for creating more complex designs.

Currently, the BLE Framework is implemented and tested for the RL78G1D BLE module. Support for other BLE modules will be provided in later revisions.

The BLE Framework provides high-level API for BLE applications, and is implemented as `sf_ble_rl78g1d`. The BLE Framework uses the Synergy Software Package (SSP) communication framework which in turn enables UART driver for communication to the underlying BLE module. It also integrates the generic BLE profile framework (`g_sf_ble_onboard_profile`) which provides a uniform interface to BLE profiles. For the RL78G1D BLE hardware module, the generic BLE profiles are implemented by the BLE module firmware.

Required Resources

To build and run the BLE framework application example, you need:

- Renesas SK-S7G2 Synergy MCU Group or the PK-S5D9 Synergy MCU Group kits
- e² studio ISDE v5.4.0.023 or greater or IAR Embedded Workbench® for Renesas Synergy™ v7.71.3 or greater
- Synergy Software Package (SSP) 1.4.0 or later or Synergy Standalone Configurator (SSC) 5.4.0.023 or later
- Segger J-link® USB driver
- Micro USB cables
- USB 2.0 Flash drive
- Android phone with **BLE Scanner APK** installed
- Download all the required Renesas software from the Renesas Synergy™ Gallery (https://synergygallery.renesas.com).

Prerequisites and Intended Audience

This application note assumes you have some experience with the Renesas Synergy e² studio ISDE and Synergy Software Package (SSP). Before you perform the procedure in this application note, follow the procedure in the *SSP User Manual* to build and run the **Blinky** project. Doing so enables you to become familiar with the e² studio and the SSP, to ensure that the debug connection to your board functions properly. In addition, this application note assumes you have some knowledge on BLE and its communication protocols.

The intended audience are users who want to develop applications with BLE interface using Renesas Synergy™ S3, S5, S7 MCU Group Series.

Contents

1. BLE Framework Overview

Bluetooth® Low Energy (BLE), sometimes referred to as **Bluetooth Smart**, is a light-weight subset of Classic Bluetooth and was introduced as part of the Bluetooth 4.0 core specification. In contrast to Classic Bluetooth, BLE is designed to provide significantly lower power consumption. This allows Internet of Thing (IoT) devices that have stricter power capacity to transfer small amounts of data between nearby devices.

Application developers access the functionality provided by the BLE stack using its APIs. The BLE stack APIs provided by different vendors are not standardized. This results in application developers having to update their code when porting to different BLE stacks.

The Synergy BLE Framework handles this issue by providing a generic interface for the underlying BLE stack provided by various vendors thereby preventing coupling between application and vendor-specific BLE stack code. The use of generic APIs makes application development simpler and portable.

1.1 Supported features

The Synergy BLE framework supports the following features:

- ThreadX® RTOS Aware and thread safe
- Bluetooth v4.2 compliant framework.
- Generic Access Profile (GAP) Features
  - User-defined advertising data
  - Security modes 1 and 2
  - Peripheral and central roles
  - White list support up to 6 devices
  - Bonding support
- Generic Attribute Profile (GATT) features
  - GATT client and server
- Generic Attribute Profile (GATT) APIs
- Generic Access Profile (GAP) APIs
- Generic On-board Profiles APIs

2. BLE Framework Module Operational Overview

This section provides the Synergy BLE Framework software architecture overview and highlights the major SSP modules used as part of BLE framework along with the operational flow sequence from the user's application level.

2.1 BLE framework architecture overview

The BLE framework provides a common interface for the application. The implementation of the interface is specific for each module. The Synergy BLE framework currently defines an interface implemented for RL78G1D BLE module. Each implementation interacts with the corresponding BLE device driver. The BLE device driver uses the underlying SSP communication framework (`g_sf_comms`) which in turn interacts with the SSP HAL components such as Universal Asynchronous Receiver/Transmitter (UART), Data Transfer Controller (DTC), and General PWM Timer (GPT) drivers to communicate with the BLE module.

The following diagram shows a high-level software architecture overview of the BLE framework in the SSP.

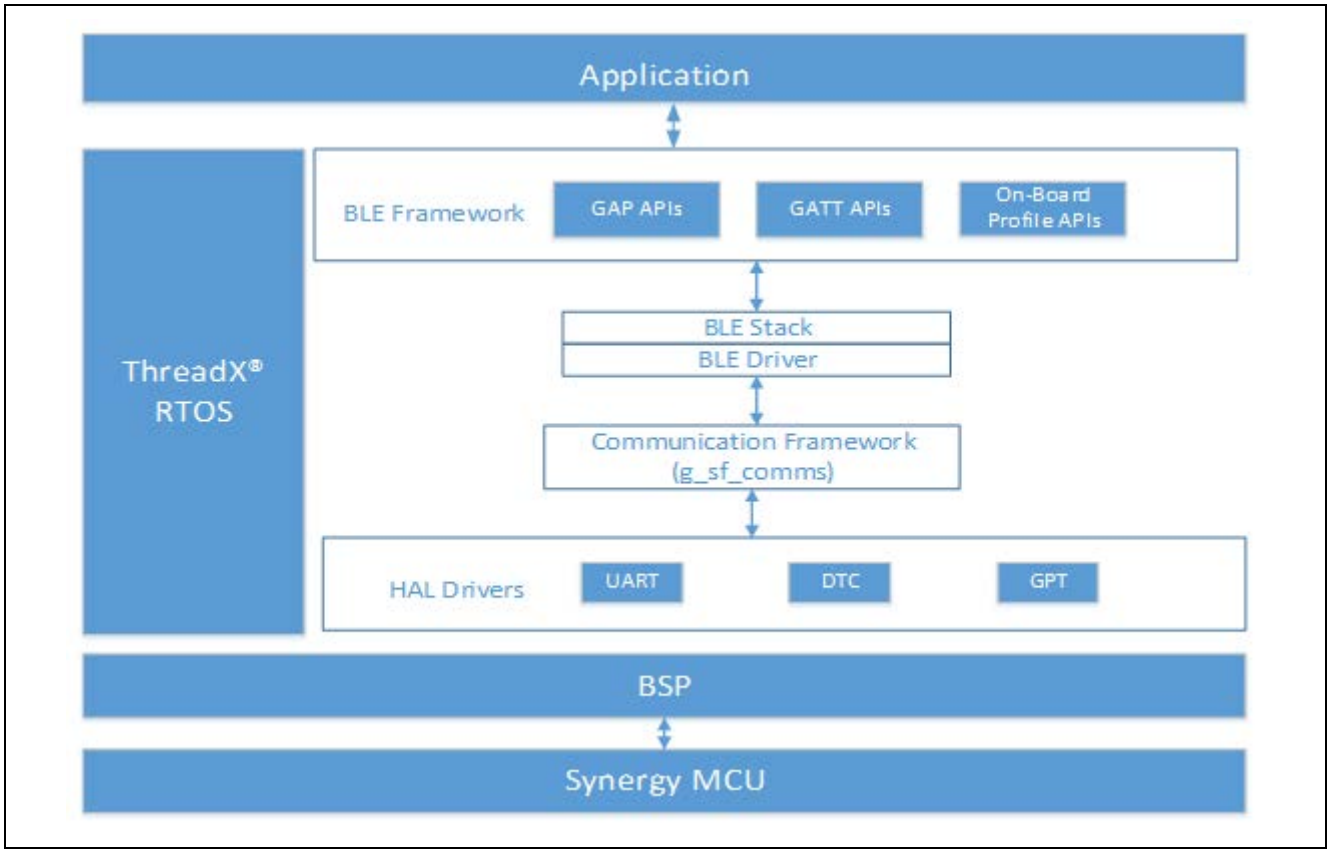


Figure 1 Typical BLE module architecture types

The Synergy BLE framework consists of the following blocks:

- GAP and GATT APIs
- On-board profiles APIs
- BLE stack.

GAP and GATT APIs

The BLE framework provides a generic interface for the application to configure and provision the BLE module. The BLE module has various configuration parameters as specified by the family of Bluetooth Smart standards. It is possible that individual device drivers and/or BLE modules might not support all configuration parameters. At a bare minimum, the provisioning API provides a mechanism to set the operating mode, security mode, security keys, and bonding mode of the BLE interface. It also provides an API for the GAP/GATT layers.

On-board Profiles APIs

The on-board profiles APIs provide a uniform interface to the BLE profiles implemented by the BLE module firmware.

BLE Stack

The BLE module host stack is typically provided by the BLE module vendor. The BLE module typically comes in three different flavors depending on the HW/SW partitioning between the host MCU and BLE module. The RL78G1D BLE module is part of the Network Controller Implementation architecture where the BLE chipset includes all the implementation for the BLE link layer, GAP, GATT, and on-board profiles. The module interfaces with the MCU over `sf_comms` framework provided by SSP.

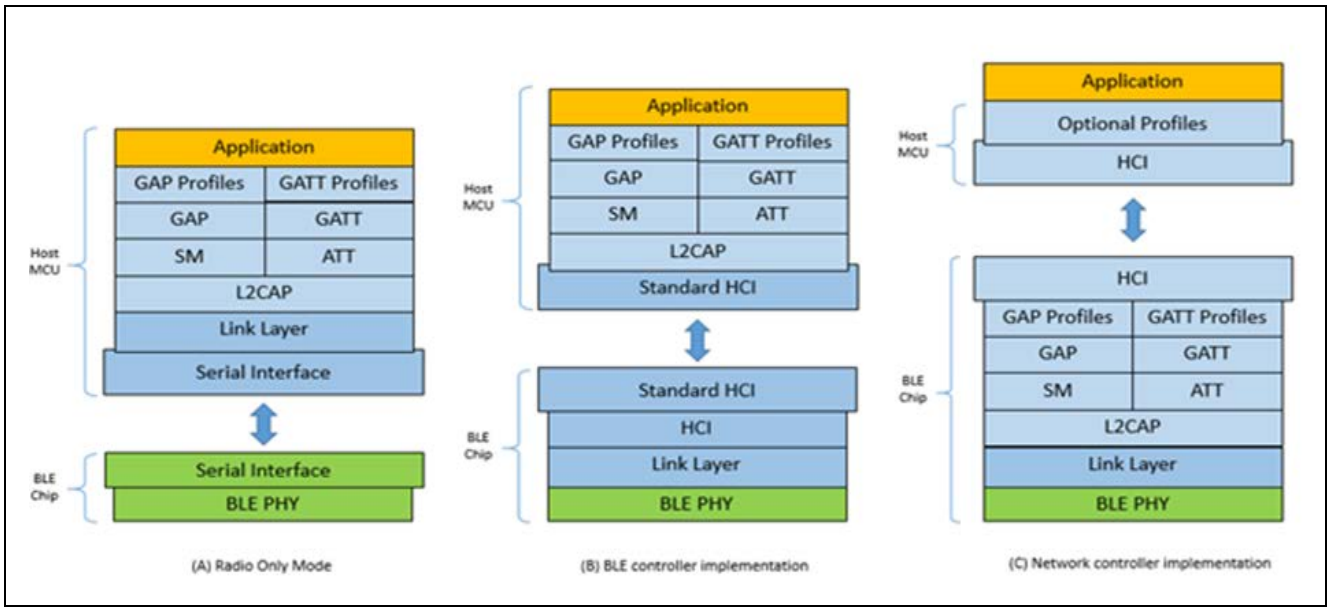


Figure 2 BLE module architecture types

1. BLE radio-only mode:

   Link layer, L2CAP, GATT, GAP layers, profiles, and application run on the host MCU. Physical layer runs on BLE chipset.

2. BLE controller implementation:

   Link layer runs on BLE chipset, L2CAP, and higher BLE protocol (GATT, GAP) layers. Profiles and application run on the host MCU.

3. Network controller implementation:

   Link layer, L2CAP, GATT, GAP layers, and generic profiles run on the BLE chipset. Optional profiles and application run on the host processor.

2.2 BLE framework instances

Application must define the BLE framework instance before using it. The instance is a structure that includes pointers to any of the following:

- BLE Framework control structure
- BLE Framework configuration structure
- BLE Framework APIs structure
- On-board profiles APIs structure.

```
/** BLE instance */
typedef struct st_sf_ble_instance
{
  sf_ble_ctrl        * p_ctrl;     ///<  Pointer to the control structure
  for this instance
  sf_ble_cfg   const * p_cfg;      ///<  Pointer to the configuration
                                        structure for this instance
  sf_ble_api_t const * p_api;      ///<  Pointer to the API structure for
                                        this instance
} sf_ble_instance_t;
```

The following structures are the Synergy BLE framework instance.

BLE Framework Control Structure:

This structure is used in all BLE framework APIs.

```
/** BLE Framework control structure */
typedef struct sf_ble_ctrl
{

void * p_driver_handle; ///< Storage for information needed for each BLE
device driver in the system
} sf_ble_ctrl_t;
```

This structure includes pointer to driver handle, that is used by framework for storing the required information by the BLE device driver.

BLE Framework Configuration Structure:

This structure is passed to open () API and you can use this structure to configure the BLE module. This configuration is applied either during initialization, such as open or provisioning such as `provisioningSet`. Configuration parameters that are not supported by the BLE module are ignored by the framework.

```
/** BLE configuration information */
typedef struct sf_ble_cfg
{
    uint8_t             bd_addr[SF_BLE_ADDR_LEN];   ///< BLE address
    sf_ble_addr_type_t  own_addr_type;              ///< self address type
    uint8_t             max_slaves;                 ///< Maximum slaves
allowed to be connected

uint8_t             update_bd_addr;         ///< Set this to true to
                                                update bluetooth address
                                                during SF_BLE_Open

    uint16_t            scan_interval;          ///< BLE scan interval
for receiving advertisement

    uint16_t            scan_window;          ///< Period of time during
                                              which advertising data is
                                              received at the scan interval.

uint16_t            disc_time;            ///< Duration for which
    the device remain discoverable

    uint16_t            con_interval;         ///< Interval for transmitting
                                              and receiving data periodically
                                              after connection establishment

    uint16_t            slave_latency;        ///< Period of time during
                                          which data is transmitted
              and received at the connection
              interval

    uint16_t            sup_timeout;          ///< Link loss time-out
    void const          * p_extend;           ///<  Instance specific
configuration
} sf_ble_cfg_t;
```

BLE Framework APIs Structure

This structure contains pointers to the BLE Framework APIs that are specific to a given module. See Section 3 BLE Framework Module API Overview for more details on these APIs.

2.3 BLE framework module operational flow

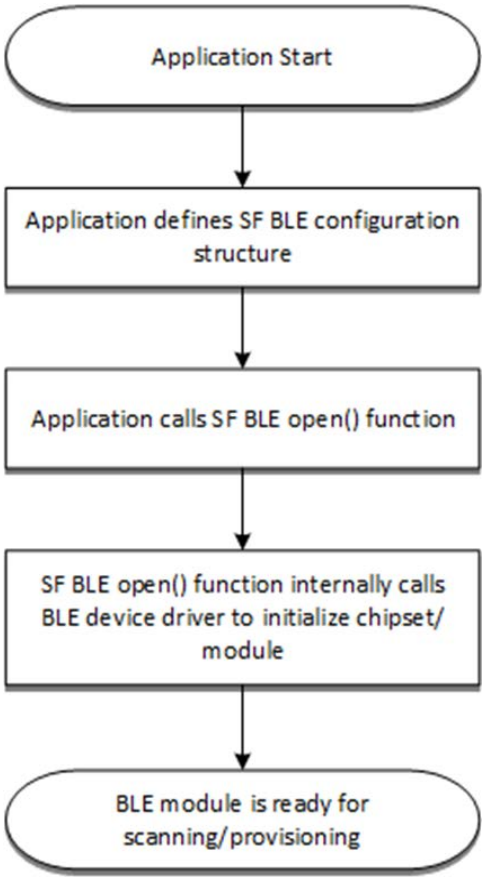
The steps for using the BLE framework module in an application are:

1. Initialize the BLE hardware module.
2. Select the GATT layer role such as GATT client or GATT server. It is most common for the slave (peripheral) device to be the GATT server and the master (central) device to be the GATT client.

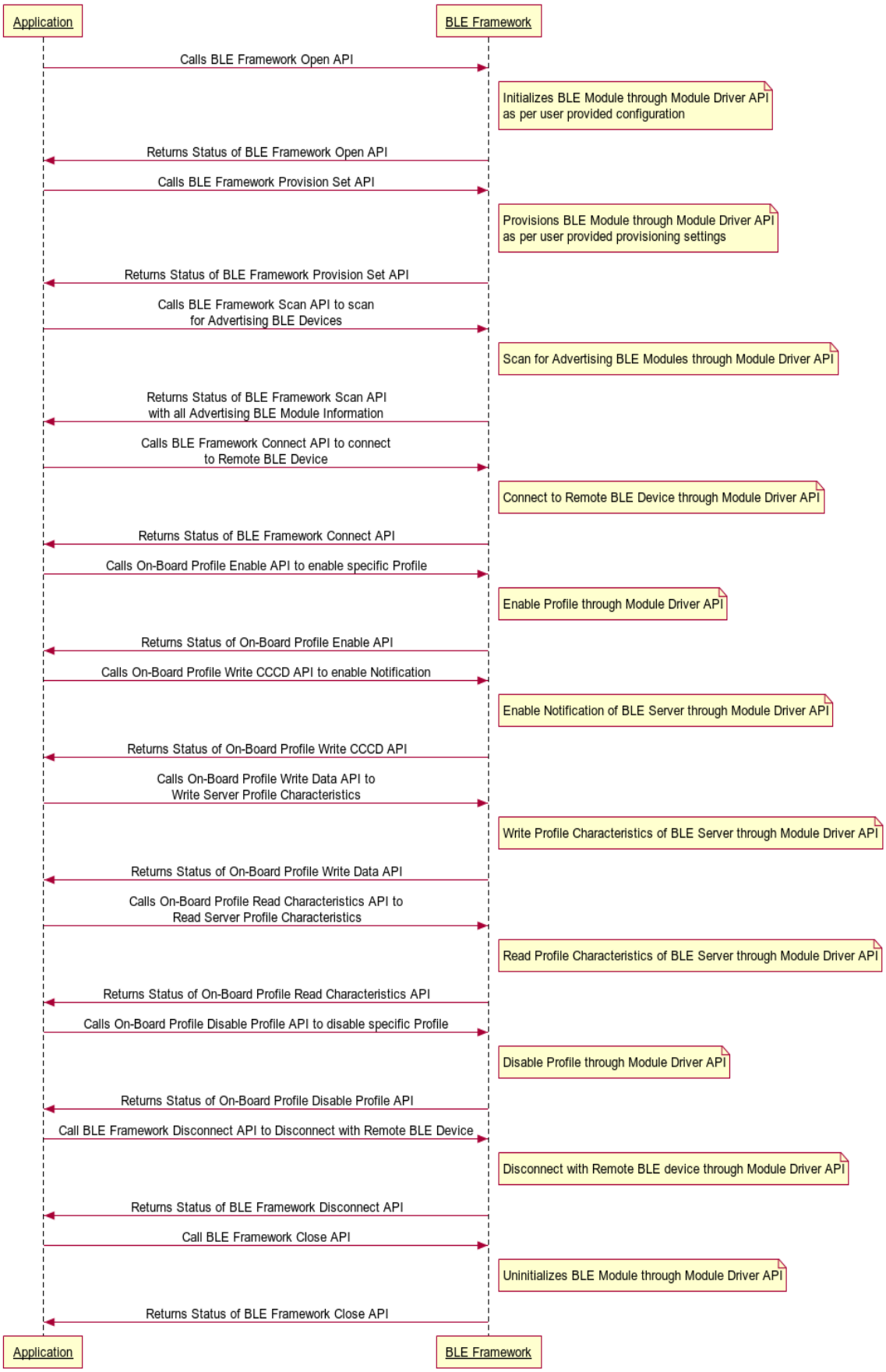
Develop application using generic (on-board) profile APIs or GAP/GATT APIs.

2.3.1 BLE module initialization flow sequence

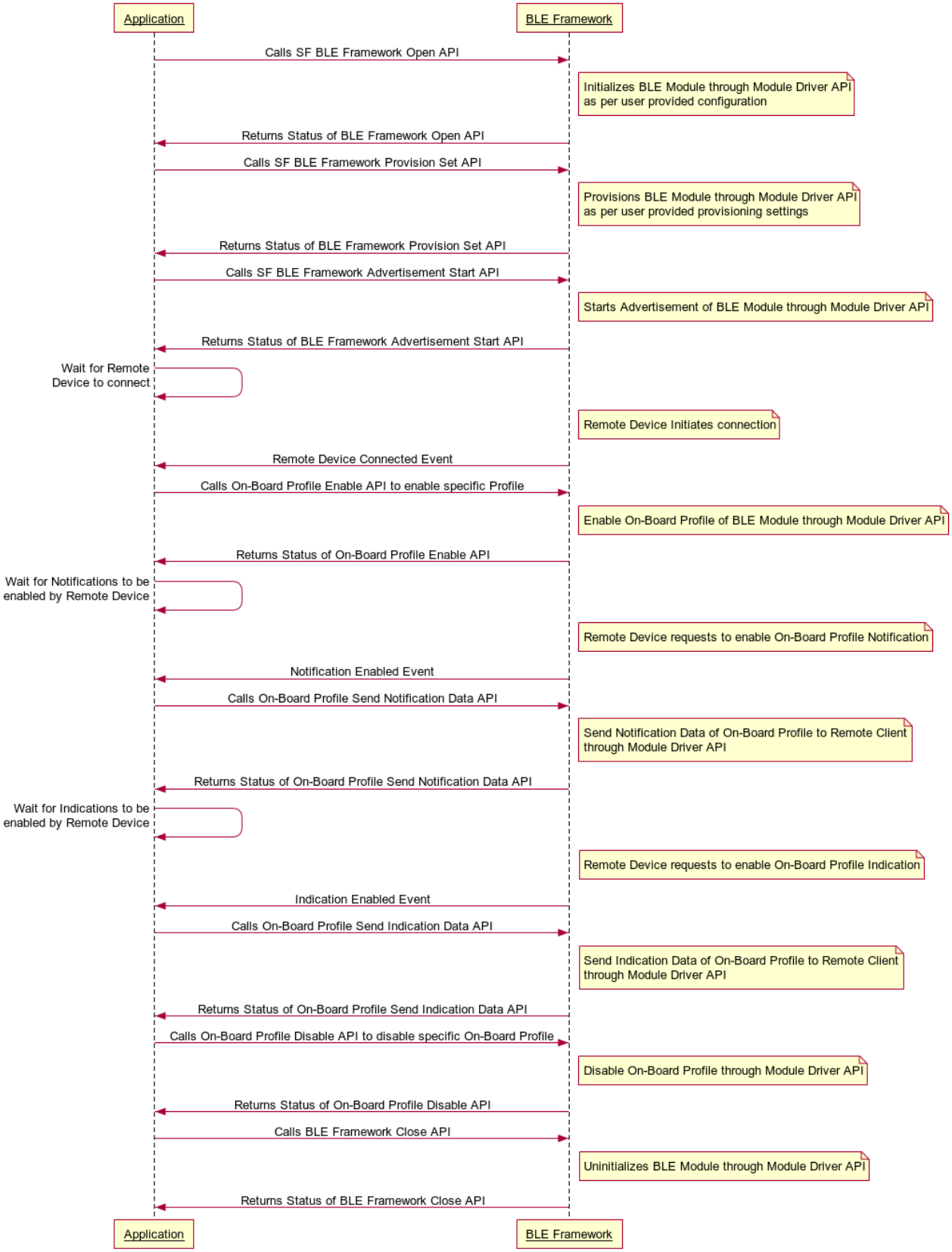
The following BLE module initialization sequence is part of the Synergy auto-generated code.



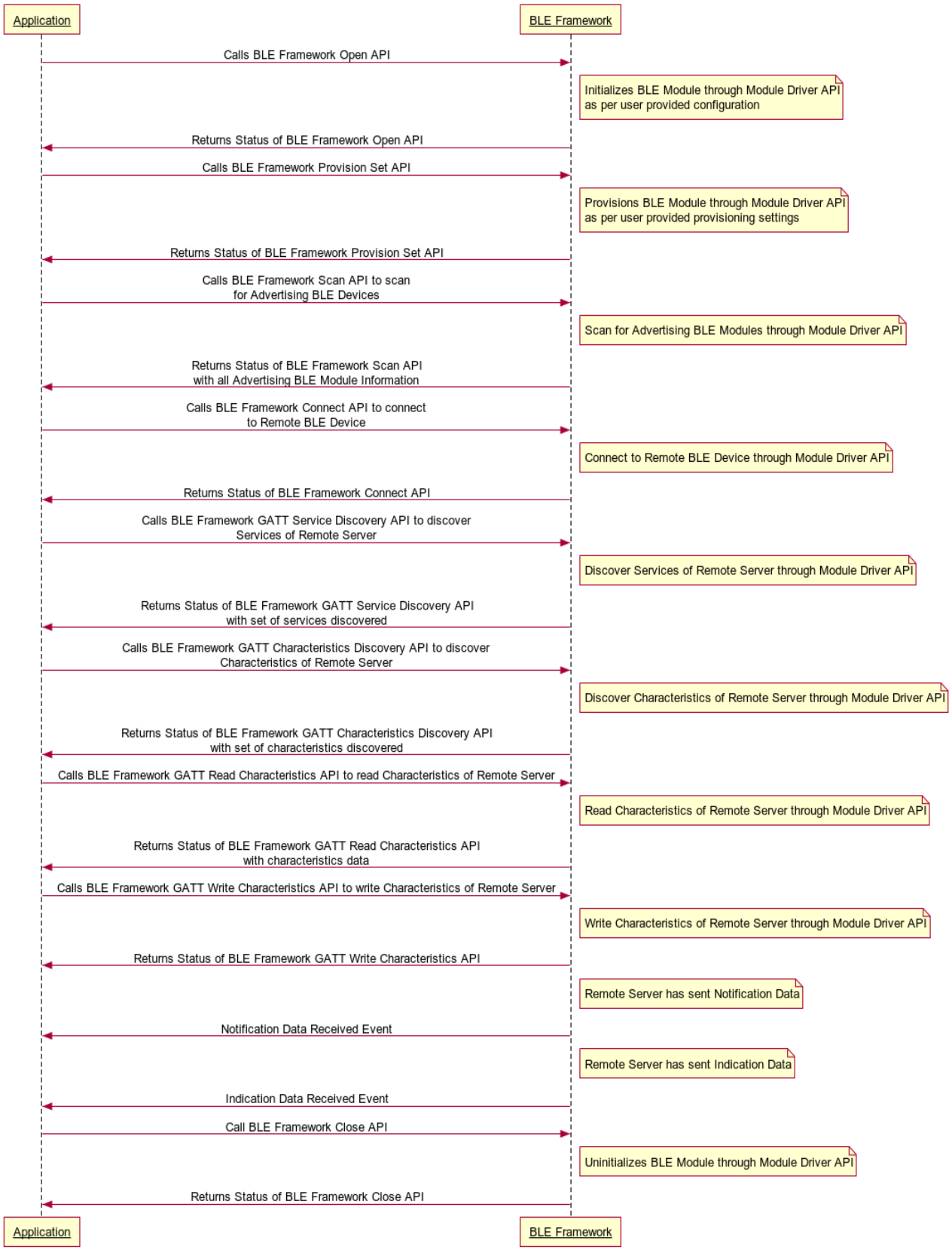
2.3.2 On-Board Profile based client application flow sequence



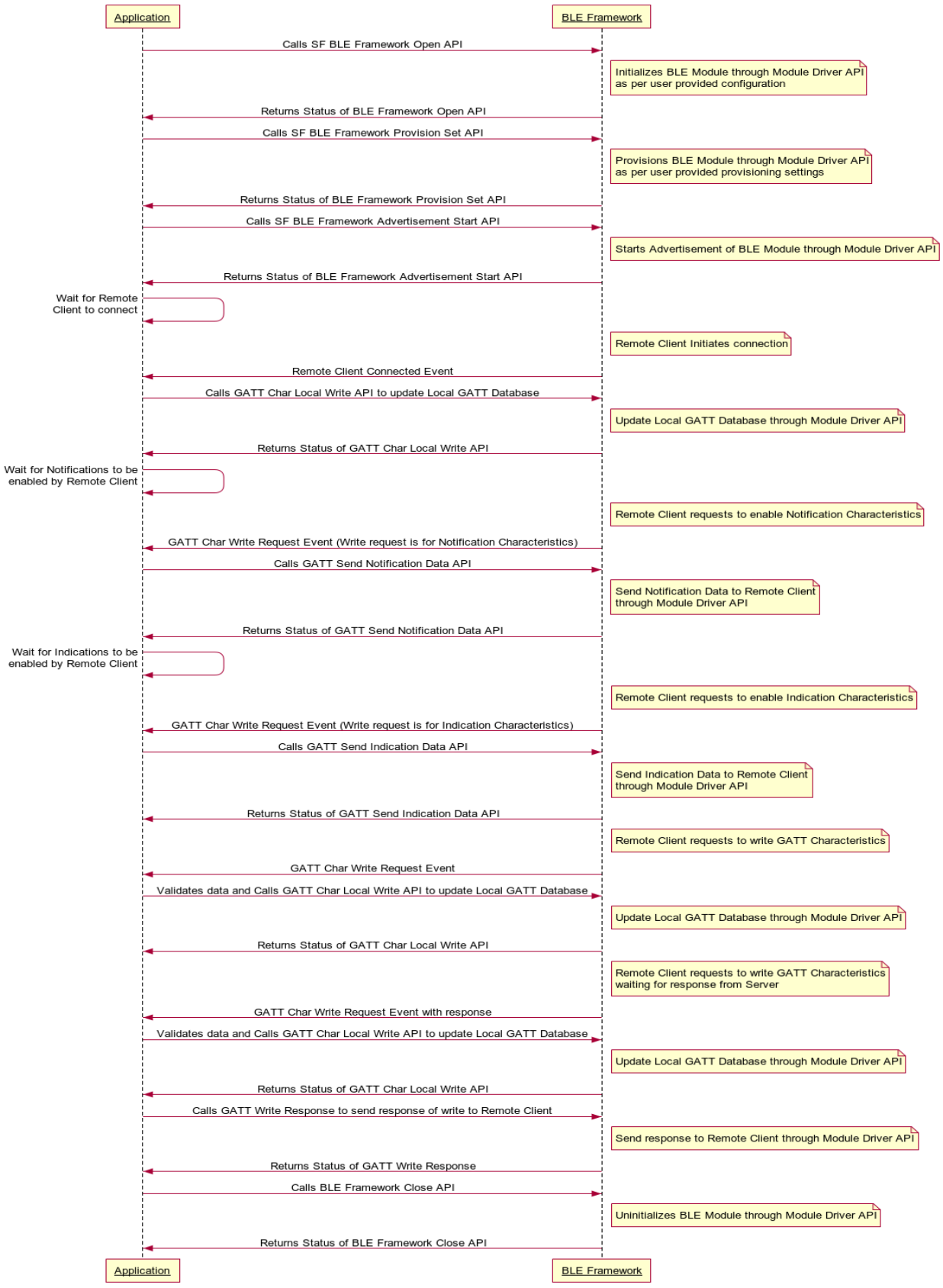
2.3.3 On-Board Profile based server application flow sequence



2.3.4 GAP/GATT based client application flow sequence



2.3.5 GAP/GATT based server application flow sequence



2.4 BLE framework security

Security Manager provides BLE protocol stack the ability to generate and exchange security keys that are used to encrypt communication link. The Security Manager has two functions:

- **Initiator**
  This is the GAP Master/Central device
- **Responder**
  This is the GAP Slave/Peripheral device

The initiator is the master device that initiate the security procedure, however the slave device can asynchronously request the initiator to begin the security procedure.

2.4.1 BLE security modes

BLE Security provides modes with levels associated with each mode. Security mode and level is a combination of support for authenticated or unauthenticated pairing, encryption or data signing. Pairing is required to satisfy various security requirements. Two types of pairing are available:

- Authenticated pairing where devices are protected from MITM (Man in The Middle) attacks
- Unauthenticated pairing where they are not protected from MITM.

Security Mode 1

Security Level 1: No Security

Security Level 2: Unauthenticated pairing with encryption

Security Level 3: Authenticated pairing with encryption

Security Level 4: Authenticated LE secure connections pairing with encryption

Security Mode 2

Security Level 1: Unauthenticated pairing with data signing

Security Level 2: Authenticated pairing with data signing

Note: RL78G1D BLE module does not support Security Mode 1 with Security Level 4.

2.4.2 BLE security procedure

BLE Security has the following procedures:

- **Pairing**
  This procedure is used to generate temporary encryption key to encrypt communication link.
  Permanent encryption keys can be shared over this encrypted communication link for additional communication.
- **Bonding**
  This is a combination of pairing and storing of permanent keys. After pairing, the permanent keys are stored in a non-volatile memory, which creates a permanent bond between two devices. For subsequent communication, it is not necessary for devices to perform the bonding procedure.
- **Encryption Establishment**
  Communication is encrypted using permanent keys

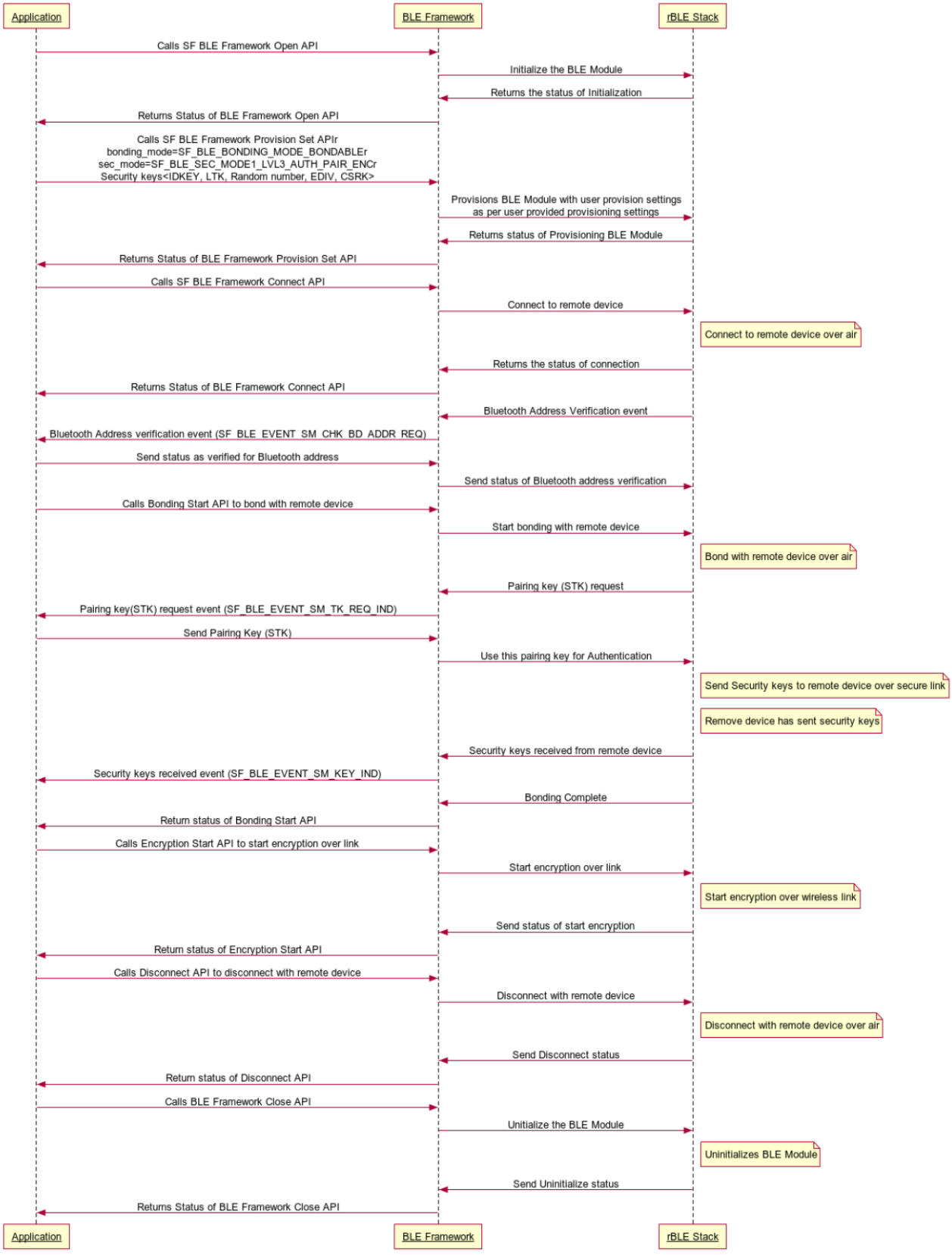
Pairing creates a secure link that lasts for the lifetime of the connection, whereas bonding creates a permanent association called bond.

2.4.3 BLE security phases

BLE Security goes through three phases as shown in the figure that follows. Two devices establish connection using the GAP connection procedure, followed by the three phases to establish a secure communication link:

- Phase 1 (Pairing Phase, Information Sharing)
  Initially in phase 1, all information required to generate the temporary keys are shared between two devices.
- Phase 2 (Pairing Phase, Temporary Key Sharing)
  In this phase, temporary encryption key (Short Term Key or STK) is generated on both devices. This is used to encrypt the connection. This encrypted link can be used for additional communication. This communication link remains encrypted until the peer devices stay connected.
- Phase 3 (Bonding, Sharing and Storage of Permanent keys)
  Devices enter this phase if bonding is required. In this phase, permanent keys (Long Term Key or LTK) is exchanged between two devices using the encrypted link which was established in phase 2 using temporary keys. These permanent keys are then stored in non-volatile memory to be made available for the devices over each connection.

2.4.4 BLE framework authentication flow sequence



2.5 BLE framework limitations

1. The BLE framework is tested only on RL78G1D BLE hardware module. Supported for different BLE modules will be added in later versions.
2. BLE Framework using RL78G1D will see compilation warnings. All the warnings are in the 3rd party RL78G1D driver code. The BLE framework files do not have any warning. These warnings should not impact the user applications.
3. The custom profile support in the BLE framework is limited to RL78G1D type BLE hardware module only.
4. HID profile client mode not supported by RL78G1D BLE hardware module. As a result, the BLE framework implementation of HID profile will also not support HID profile client mode. Applications using BLE framework for RL78G1D will not be able to use the HID profile in client mode.
5. Multiple slave BLE devices cannot be connected to RL78G1D BLE module.

3. BLE Framework Module API Overview

This section provides a list of available APIs and a short description of each API, including its functionality, parameters, and return values. For more detailed information, see the *SSP User's Manual*, API reference section.

3.1 BLE GAP APIs

3.1.1 open

Description:

This API initializes the interface for data transfers. It handles initial driver configuration, enables the driver link and interrupt, and makes the device ready for data transfer.

Parameters:

Name	Direction	Description
p_ctrl	In, out	Pointer to the control block for BLE module (see sf_ble_ctrl)
p_cfg	In	Pointer to BLE configuration structure sf_ble_cfg_t (see sf_ble_cfg)

Return Values:

SSP Error status

Function Prototype:

```
      ssp_err_t (*open)(sf_ble_ctrl_t * const p_ctrl,
 const sf_ble_cfg_t * p_cfg);
```

3.1.2 close

Description:

This API de-initialize the interface and may put the BLE module in low power mode or power it off. It also closes the driver, disables the driver link, disable the interrupt in the BLE module driver.

Parameter Name	Direction	Description
p_ctrl	In	Pointer to the control block for BLE module (see `sf_ble_ctrl`)

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*close)(sf_ble_ctrl_t * const p_ctrl);
```

3.1.3 infoGet

Description:

This API gets the BLE module information such as the chipset information and RSSI value.

Parameter Name	Direction	Description
p_ctrl	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
p_handle	In	Pointer to connection handle
p_ble_info	Out	Pointer to module information

Return Values:

It returns the following information obtained from the BLE module:

- Chipset/driver information string
- RSSI value (unsigned 16 bits integer)

Function Prototype:

```
ssp_err_t (*infoGet)(sf_ble_ctrl_t * const p_ctrl, sf_ble_conn_handle_t *
p_handle, sf_ble_info_t * p_ble_info);
```

3.1.4 provisionGet

Description:

The provisionGet() function gets the BLE GAP provisioning information.

Parameter Name	Direction	Description
p_ctrl	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
p_ble_provisioning	Out	Current provisioning information

Return Values:

It returns the following parameters:

- GAP Name
- Broadcast mode flag
- Bonding mode
- Security mode
- GAP role (Central/Master or Peripheral/Slave)
- GAP user event callback.

Function Prototype:

```
ssp_err_t (*provisionGet)(sf_ble_ctrl_t * const p_ctrl,  sf_ble_provisioning_t
* p_ble_provisioning);
```

3.1.5 provisionSet

Description:

The `provisionSet()` function provisions BLE module.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see `sf_ble_ctrl`)
`p_ble_provisioning`	in	Pointer to BLE provisioning structure

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*provisionSet)(sf_ble_ctrl_t * const p_ctrl, const
sf_ble_provisioning_t * p_ble_provisioning);
```

3.1.6 scan

Description:

This API scans for available BLE devices and returns the list to the caller.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`p_scan`	Out	Pointer to scan structure
`P_cnt`	Inout	Pointer to number of BLE devices scanned
`P_scan_info`	In	Pointer to scan information structure

Return Values:

The `scan()` function returns a list of BLE devices scanned by the BLE module with the following parameters:

- 48-bits Bluetooth address
- RSSI
- Scan data.

Function Prototype:

```
ssp_err_t (*scan)(sf_ble_ctrl_t * const p_ctrl, sf_ble_scan_t * p_scan,
uint8_t * p_cnt,  sf_ble_scan_info_t * p_scan_info);
```

3.1.7 advertisementStart

Description:

The `advertisementStart()` function start advertisement.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`p_advt_info`	In	Pointer to advertisement information structure

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*advertisementStart)(sf_ble_ctrl_t * const p_ctrl,
sf_ble_adv_info_t * const p_advt_info);
```

3.1.8 advertisementStop

Description:

The `advertisementStop()` function stops advertisement.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see `sf_ble_ctrl`)

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*advertisementStop)(sf_ble_ctrl_t * const p_ctrl);
```

3.1.9 whitelistAdd

Description:

The `whitelistAdd()`function adds devices to the whitelist for advertisements, scans, and connects requests.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see `sf_ble_ctrl`)
`p_bd_addr`	In	Pointer to BLE address

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*whitelistAdd)(sf_ble_ctrl_t * const p_ctrl, const uint8_t *
p_bd_addr);
```

3.1.10 whitelistDel

Description:

The `whitelistDel()` function deletes devices from the whitelist for advertisements, scans, and connects requests.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see `sf_ble_ctrl`)
`p_bd_addr`	In	Pointer to BLE address

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*whitelistDel)(sf_ble_ctrl_t * const p_ctrl, const uint8_t *
p_bd_addr);
```

3.1.11 bondingStart

Description:

The `bondingStart()` function starts bonding with a remote device.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see `sf_ble_ctrl`)
`p_bd_addr`	In	Pointer to BLE address
`p_handle`	In	Pointer to connection handle

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*bondingStart)(sf_ble_ctrl_t * const p_ctrl,
                          sf_ble_conn_handle_t * p_handle,
  const uint8_t *p_bd_addr,
  sf_ble_bonding_start_t *p_bonding_start);
```

3.1.12 bondingResponse

Description:

The `bondingResponse()` function responds to a bonding request.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`p_bd_addr`	In	Pointer to BLE address
`p_handle`	In	Pointer to connection handle
`P_bonding_resp`	In	Pointer to bonding address

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*bondingResponse)(sf_ble_ctrl_t * const p_ctrl,
                             sf_ble_conn_handle_t * p_handle,
                             const uint8_t * p_bd_addr,
                             sf_ble_bonding_response_t * p_bonding_resp);
```

3.1.13 connect

Description:

The `connect()` function connects to a remote device.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_conn`	In	Pointer to connection information
`p_handle`	out	Pointer to connection handle

Return Values:

Returns the connection handle.

Function Prototype:

```
ssp_err_t (*connect)(sf_ble_ctrl_t * const p_ctrl, sf_ble_connection_t
const * const p_conn,              sf_ble_conn_handle_t * p_handle);\
```

3.1.14 disconnect

Description:

The `disconnect()` function disconnects from a remote device.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`p_handle`	out	Pointer to connection handle

Return Values:

Returns the connection handle.

Function Prototype:

```
ssp_err_t (*disconnect)(sf_ble_ctrl_t * const p_ctrl,
                        sf_ble_conn_handle_t * p_handle);
```

3.1.15 listen

Description:

The `listen()`function listens for an incoming connection request from a remote device.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)

Return Values:

Returns the connection handle.

Function Prototype:

```
ssp_err_t (*listen)(sf_ble_ctrl_t * const p_ctrl);
```

3.2 BLE GATT APIs

3.2.1 gattCharWriteLocal

Description:

The `gattCharWriteLocal()` function updates the local GATT database.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`Char_handle`	In	Characteristics handle
`Data_length`	In	Length of data to write
`P_data`	In	Pointer to data

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (* gattCharWriteLocal)(sf_ble_ctrl_t * const p_ctrl, uint16_t
                          char_handle, uint16_t data_length,
                          uint8_t * const p_data);
```

3.2.2 gattServiceDiscovery

Description:

The `gattServiceDiscovery()` function discovers GATT services on a remote device.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Connection handle
`P_sf_ble_svc_dscv_req`	In	Pointer to service discovery request
`P_sf_ble_svc_dscv_rsp`	Out	Pointer to service discovery response
`P_rsp_cnt`	Inout	Input size specifying maximum number of service discovery results which can be stored in response, output specifying number of service discovery results stored in response

Return Values:

Returns pointer to service discovery response, outputs specifying number of service discovery results stored in response.

Function Prototype:

```
ssp_err_t (* gattServiceDiscovery)(sf_ble_ctrl_t * const p_ctrl,
                           sf_ble_conn_handle_t * p_handle,
                           sf_ble_service_discovery_req_t const
                           * const p_sf_ble_svc_dscv_req,
                           sf_ble_service_discovery_rsp_t *
                           const p_sf_ble_svc_dscv_rsp,
                           uint32_t * const p_rsp_cnt);
```

3.2.3 gattCharDiscovery

Description:

The gattCharDiscovery() function discovers GATT characteristics on a remote device.

Parameter Name	Direction	Description
p_ctrl	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
P_handle	In	Connection handle
P_sf_ble_char_dscv_req	In	Pointer to characteristics discovery request
P_sf_ble_char_dscv_rsp	Out	Pointer to characteristics discovery response
P_rsp_cnt	Inout	Input size specifying maximum number of service discovery results which can be stored in response, output specifying number of service discovery results stored in response

Return Values:

Returns pointer to characteristics discovery response, output specifying number of characteristics discovery results stored in response.

Function Prototype:

```
ssp_err_t (* gattCharDiscovery)(sf_ble_ctrl_t * const p_ctrl,
                                sf_ble_conn_handle_t * p_handle,
                                sf_ble_char_discovery_req_t const *
                                const p_sf_ble_char_dscv_req,
                                sf_ble_char_discovery_rsp_t * const
                                p_sf_ble_char_dscv_rsp, uint32_t * const
                                p_rsp_cnt);
```

3.2.4 gattCharDescDiscovery

Description:
The `gattCharDescDiscovery()` function discovers GATT characteristic descriptor on a remote device.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Connection handle
`Start_handle`	In	Start handle from set of handle ranges to be used in discovery
`End_handle`	In	End handle from set of handle ranges to be used in discovery
`P_sf_ble_chardesc_dscv_rsp`	out	Pointer to characteristics descriptor discovery response
`P_rsp_cnt`	Inout	Input size specifying maximum number of service discovery results which can be stored in response, output specifying number of service discovery results stored in response

Return Values:

Returns pointer to characteristics descriptor discovery response.

Function Prototype:

```
ssp_err_t (* gattCharDescDiscovery)(sf_ble_ctrl_t * const p_ctrl,
                                    sf_ble_conn_handle_t * p_handle,
                    uint16_t start_handle, uint16_t end_handle,
                    sf_ble_char_desc_discovery_rsp_t * const
         p_sf_ble_chardesc_dscv_rsp, uint32_t * const p_rsp_cnt);
```

3.2.5 gattCharWrite

Description:

The `gattCharWrite()` function writes GATT characteristics on a remote device.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Connection handle
`P_char_write_req`	In	Pointer to characteristic write request

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (* gattCharWrite)(sf_ble_ctrl_t * const p_ctrl,
                            sf_ble_conn_handle_t * p_handle,
                            sf_ble_char_write_req_t const * const
                            p_char_write_req);
```

3.2.6 gattCharRead

Description:

The `gattCharRead()` function reads GATT characteristics on a remote device.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Connection handle
`P_char_read_req`	In	Pointer to characteristic read request
`P_char_read_rsp`	Out	Pointer to characteristic read response

Return Values:

Returns pointer to characteristics read response.

Function Prototype:

```
ssp_err_t (* gattCharRead)(sf_ble_ctrl_t * const p_ctrl,
                           sf_ble_conn_handle_t * p_handle,
                           sf_ble_char_read_req_t const * const
                           p_char_read_req, sf_ble_char_read_rsp_t *
                           const p_char_read_rsp);
```

3.2.7 gattCharExecuteWrite

Description:

The `gattCharExecuteWrite()` function executes a write (commit) on GATT characteristics on a remote device.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Connection handle
`Execute_flag`	In	Flag specifying whether to execute or cancel pending writes

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (* gattCharExecuteWrite)(sf_ble_ctrl_t * const p_ctrl,
                                   sf_ble_conn_handle_t * p_handle,
                                   sf_ble_execute_write_t execute_flag);
```

3.2.8 gattSendNotify

Description:

The `gattSendNotify()` function sends notifications from local GATT server to remote GATT client.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Connection handle
`Char_handle`	In	Characteristics handle whose value will be notified

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (* gattSendNotify)(sf_ble_ctrl_t * const p_ctrl,
                             sf_ble_conn_handle_t * p_handle,
                             uint16_t char_handle);
```

3.2.9 gattSendIndicate

Description:

The `gattSendIndicate()` function sends indications from local GATT server to remote GATT client.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Connection handle
`Char_handle`	In	Characteristics handle whose value will be indicated

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (* gattSendIndicate)(sf_ble_ctrl_t * const p_ctrl,
                               sf_ble_conn_handle_t * p_handle,
                               uint16_t char_handle);
```

3.2.10 gattWriteResponse

Description:

The `gattWriteResponse()` function responds to the write characteristic value request from the remote GATT client.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Connection handle
`handle`	In	Characteristics handle used for write operation
`Error_code`	In	Characteristics write operation error code to be sent in response

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (* gattWriteResponse)(sf_ble_ctrl_t * const p_ctrl,
                                sf_ble_conn_handle_t * p_handle,
                                uint16_t handle,
                                sf_ble_attribute_error_code_t
                                error_code);
```

3.3 On-Board Profiles APIs

3.3.1 open

Description:

This API initializes the interface for data transfers.

Parameter Name	Direction	Description
`p_ctrl`	Inout	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_cfg`	In	Pointer to BLE configuration structure

Return Values:

SSP Error status

Funct**ion Prototype:**

```
ssp_err_t (*open)(sf_ble_onboard_profile_ctrl_t * const p_ctrl, const
                  sf_ble_onboard_profile_cfg_t *p_cfg);
```

3.3.2 close

Description:

This API de-initializes the interface and may put it in low power mode or power it off. The API closes the driver, and disables the driver link and interrupt.

Parameter Name	Direction	Description
p_ctrl	In	Pointer to the control block for BLE module (see sf_ble_ctrl)

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*close)(sf_ble_onboard_profile_ctrl_t * const p_ctrl);
```

3.3.3 onbpEnable

Description:

The `onbpEnable()` function enables the profile in server mode or client mode.

Parameter Name	Direction	Description
p_ctrl	Inout	Pointer to the control block for BLE module (see sf_ble_ctrl)
P_handle	In	Pointer to connection handle
Profile	In	Profile type to enable
P_prf_cb	In	User callback for profile
Sec	In	Security type for profile

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*onbpEnable)(sf_ble_onboard_profile_ctrl_t * const p_ctrl,
                        sf_ble_conn_handle_t * p_handle,
                        sf_onbp_t profile, sf_ble_profile_callback_t
                        p_prf_cb, sf_ble_prf_sec_t sec);
```

3.3.4 onbpServerWriteData

Description:

The `onbpServerWriteData()` function updates the value of the characteristic in the local database.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Pointer to connection handle
`Profile`	In	Profile type
`characteristics`	In	Profile characteristics
`P_data`	In	Pointer to data

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*onbpServerWriteData)(sf_ble_onboard_profile_ctrl_t * const
                                 p_ctrl, sf_ble_conn_handle_t *
                                 p_handle, sf_onbp_t profile,
                                 sf_ble_onbp_char_t characteristics,
                                 const void * p_data);
```

3.3.5 onbpServerSendNotification

Description:

The `onbpServerSendNotification()` function sends notifications.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Pointer to connection handle
`Profile`	In	Profile type
`characteristics`	In	Profile characteristics
`P_data`	In	Pointer to data

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*onbpServerSendNotification)(sf_ble_onboard_profile_ctrl_t *
                                const p_ctrl, sf_ble_conn_handle_t *
                                p_handle, sf_onbp_t profile,
                                sf_ble_onbp_char_t characteristics, const
                                void * p_data);
```

3.3.6 onbpServerSendIndication

Description:

The `onbpServerSendIndication()` function sends indications.

Parameter Name	Direction	Description
p_ctrl	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
P_handle	In	Pointer to connection handle
Profile	In	Profile type
characteristics	In	Profile characteristics
P_data	In	Pointer to data

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*onbpServerSendIndication)(sf_ble_onboard_profile_ctrl_t *
                                const p_ctrl,sf_ble_conn_handle_t *
                                p_handle, sf_onbp_t profile,
                                sf_ble_onbp_char_t characteristics, const
                                void * p_data);
```

3.3.7 onbpClientWriteCCCD

Description:

The `onbpClientWriteCCCD()` function sets the Client Configuration Control Descriptor on the remote device.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Pointer to connection handle
`Profile`	In	Profile type
`Cccd_char`	In	CCCD code
`Cccd_val`	In	Configuration data of CCCD

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*onbpClientWriteCCCD)(sf_ble_onboard_profile_ctrl_t * const
                                p_ctrl, sf_ble_conn_handle_t *
                                p_handle, sf_onbp_t profile,
                                sf_ble_onbp_char_t cccd_char,
                                sf_ble_cccd_val_t cccd_val);
```

3.3.8 onbpDisable

Description:

The `onbpDisable()` function disables the profile in server mode and client mode.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Pointer to connection handle
`Profile`	In	Profile type to disable

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*onbpDisable)(sf_ble_onboard_profile_ctrl_t * const p_ctrl,
      sf_ble_conn_handle_t * p_handle,
            sf_onbp_t profile);
```

3.3.9 onbpClientReadChar

Description:

The `onbpClientReadChar()` function reads a GATT characteristic associated with the profile or service.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Pointer to connection handle
`Profile`	In	Profile type
`Characteristics`	In	Profile characteristics

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*onbpClientReadChar)(sf_ble_onboard_profile_ctrl_t * const
                                p_ctrl, sf_ble_conn_handle_t *
                                p_handle, sf_onbp_t profile,
                                sf_ble_onbp_char_t characteristics);
```

3.3.10 onbpClientWriteChar

Description:

The `onbpClientWriteChar()` function writes a GATT characteristic associated with the profile or service.

Parameter Name	Direction	Description
`p_ctrl`	In	Pointer to the control block for BLE module (see sf_ble_ctrl)
`P_handle`	In	Pointer to connection handle
`Profile`	In	Profile type
`Characteristics`	In	GATT characteristics code
`P_data`	In	Pointer to data

Return Values:

SSP Error status

Function Prototype:

```
ssp_err_t (*onbpClientWriteChar)(sf_ble_onboard_profile_ctrl_t * const
         p_ctrl, sf_ble_conn_handle_t *
         p_handle, sf_onbp_t profile,
         sf_ble_onbp_char_t characteristics,
         const void * p_data);
```

4. Including BLE Framework in an Application

This section assumes that you have some experience with the Renesas e^2 studio ISDE and Synergy Software Package (SSP). Before you perform the procedure in this section, follow the procedure in the *SSP 1.4.0 User's Manual* to build and run the Blinky project. By doing so, you will become familiar with the e^2 studio ISDE and SSP.

SSP 1.4.0 User's Manual can be downloaded from the Renesas Synergy™ Gallery (http://www.renesassynergy.com/gallery)

The following procedure is used to include the Synergy BLE Framework in your application using the e^2 studio ISDE.

Step 1: Create a new project with RTOS included

1. Create a new Synergy project by clicking **File->New->Synergy C Project**.
2. Enter the project name and set up the Synergy license file.
3. Select the board (for example, SK-S7G2 : S7G2 SK).
4. Select the **BSP** option in the **Project Template Selection** window.

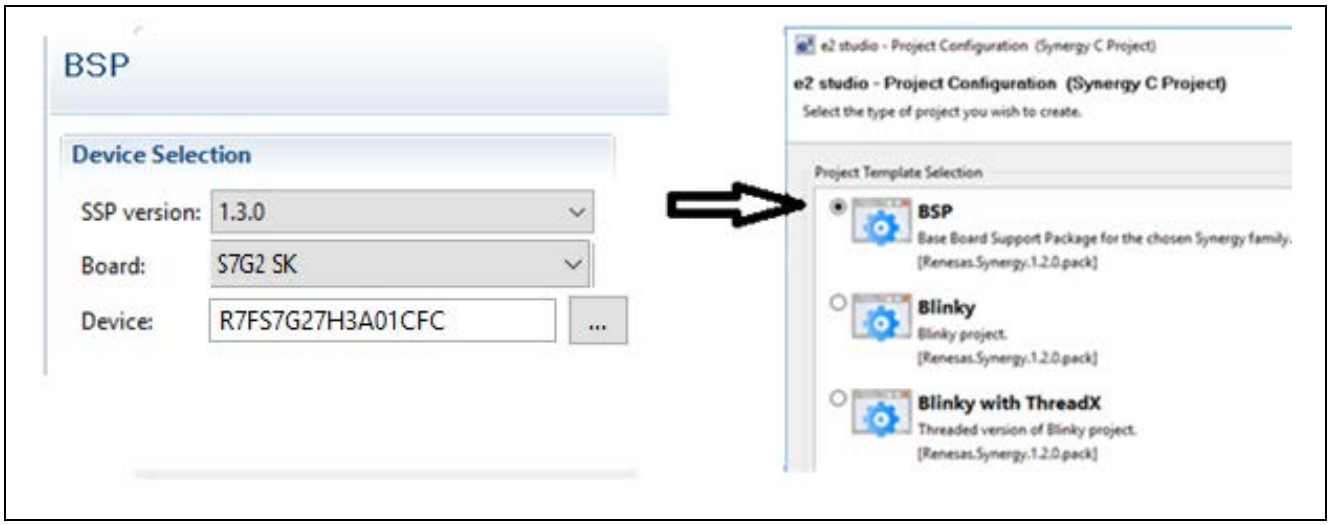


Figure 3 Synergy project creation

Step 2: Create a new thread to include BLE framework

1. BLE framework is ThreadX compliant. To include BLE framework in your application, create a new thread and then include the BLE framework module.
2. Select the **Thread** tab and click the + sign to create a new thread.
3. Refer to the below table that explains the **Thread Properties**.

Table 1 Thread Properties

Properties	Default value	Description
Symbol	New_thread0	Symbol name for the thread. This name will be used for creating the thread file. If the Symbol is set to template, then the thread file is created as template_entry.c
Name	New_thread	Name of the thread created
Stack size	1024	Stack size in bytes for this thread
Priority	1	Priority of this thread
Auto Start	Enabled	If Enabled, the thread starts to run once its created. If Disabled, the thread doesn't run once its created. User need to resume when needed.
Time slicing interval (ticks)	1	Thread execution interval in ticks

4. The below figure shows an example on how to create a BLE **Thread** and its **Properties** are updated.

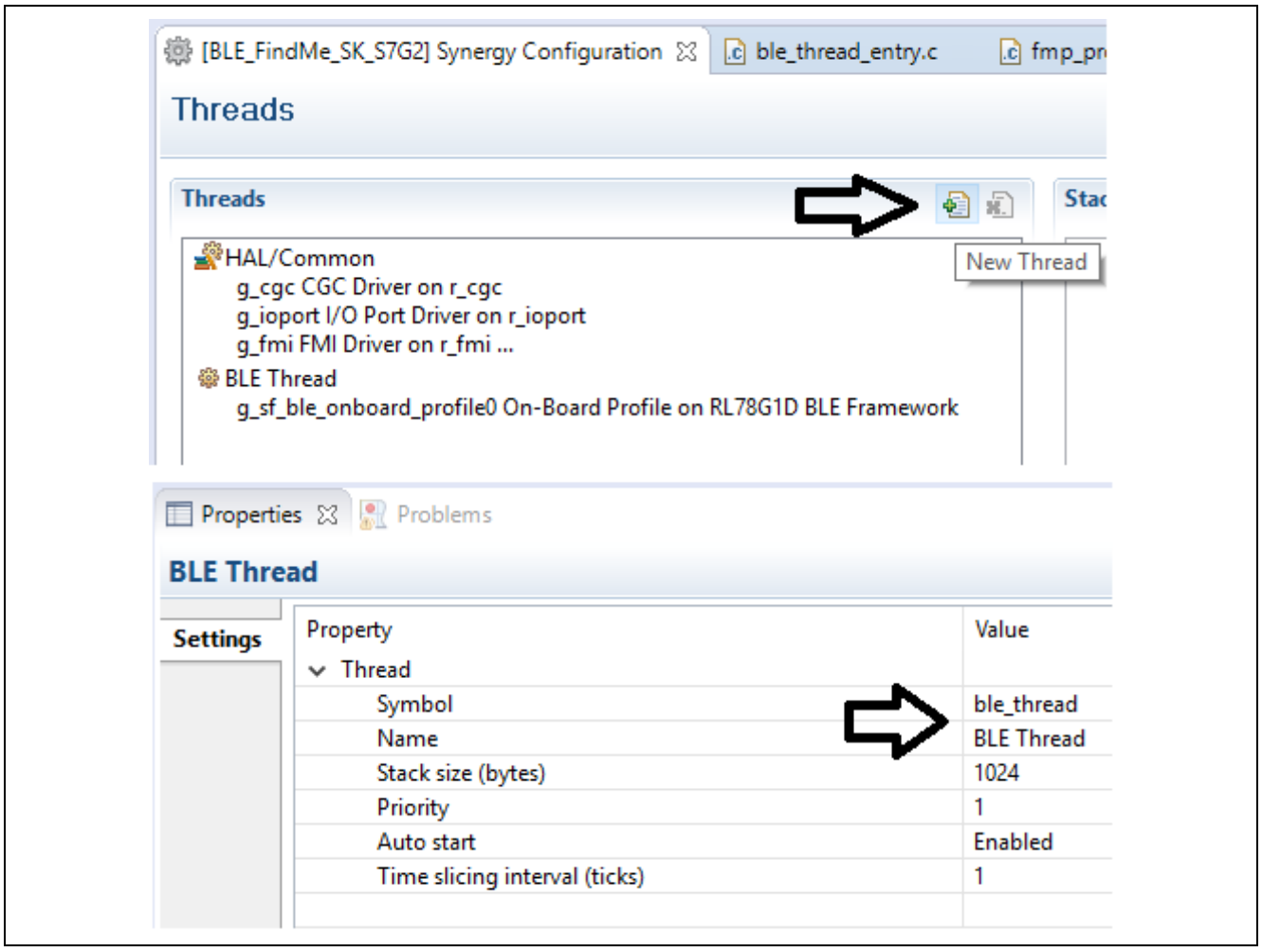


Figure 4 BLE Thread creation and Properties tab

Step 3: Add the BLE framework

1. Click the newly created BLE thread. In the **BLE Thread Stacks** window, click the + sign to add the BLE framework.
2. Select **Framework → Networking →BLE → On-Board Profile on RL78G1D BLE Framework**.

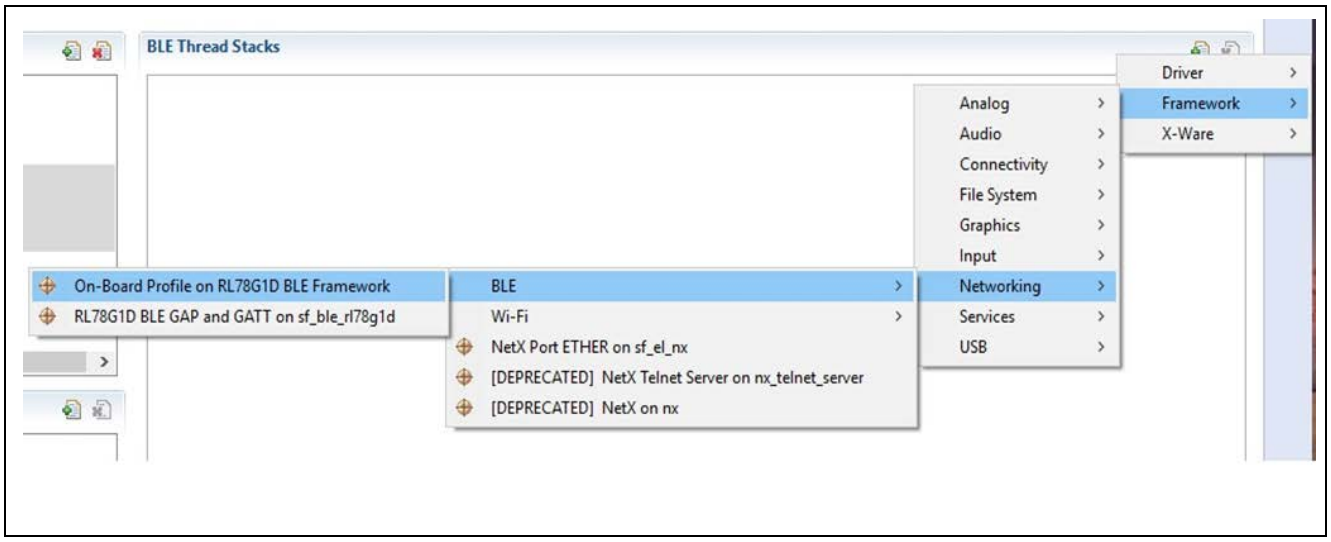


Figure 5 Adding BLE framework

3. The BLE framework uses the SSP Communication framework module to communicate with the underlying BLE hardware module. The communication in turn uses UART/USB for communicating to the underlying BLE hardware module.

4. Click the **Add Communication Framework** box →**New**, and select **Communications Framework on sf_uart_comms**.

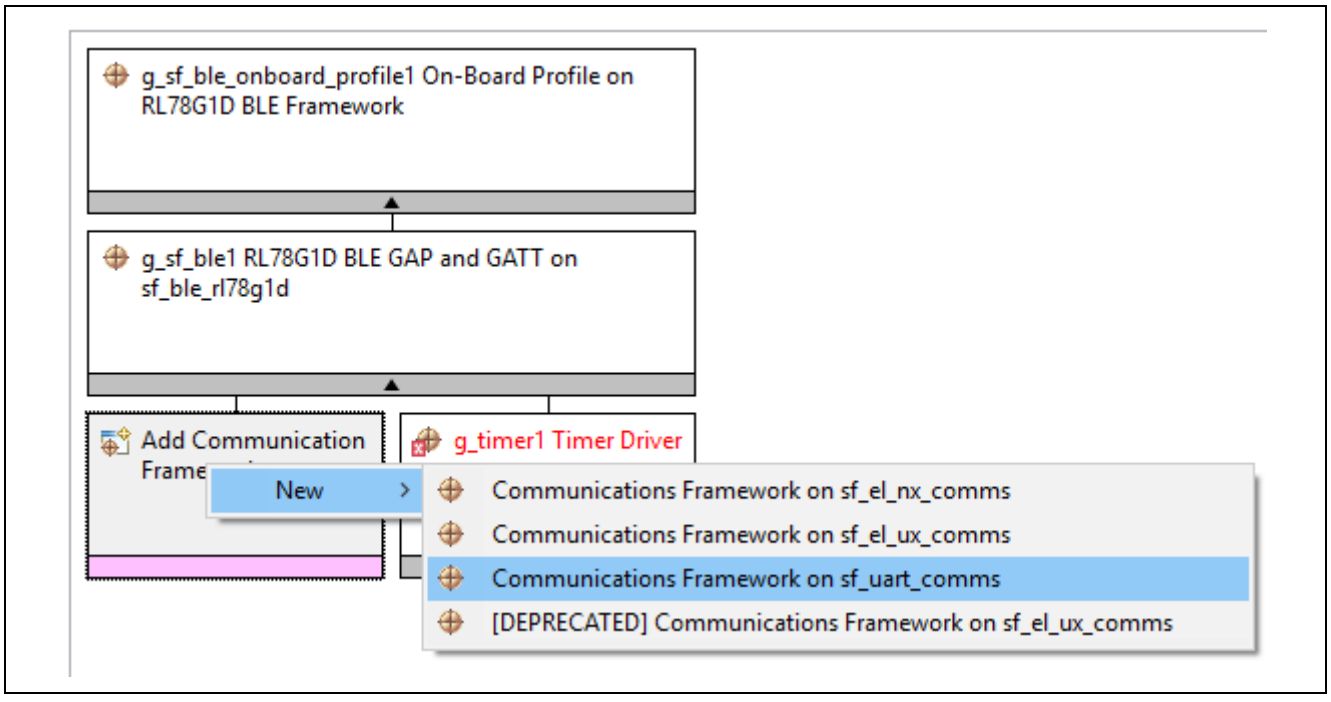


Figure 6 Adding communications framework

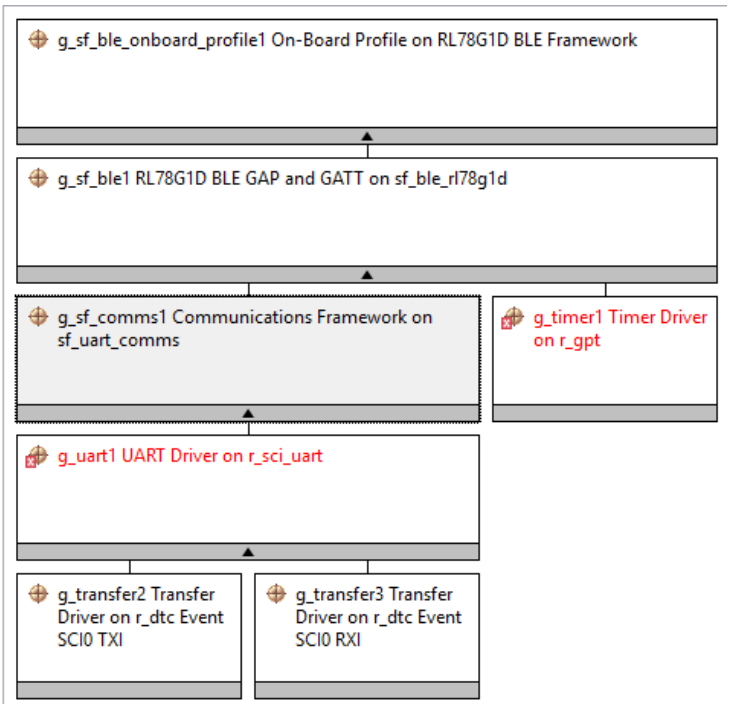


Figure 7 Communication Framework added

5. Configuring BLE Framework Module

This section provides detailed information about the configuration parameters associated with the BLE framework module.

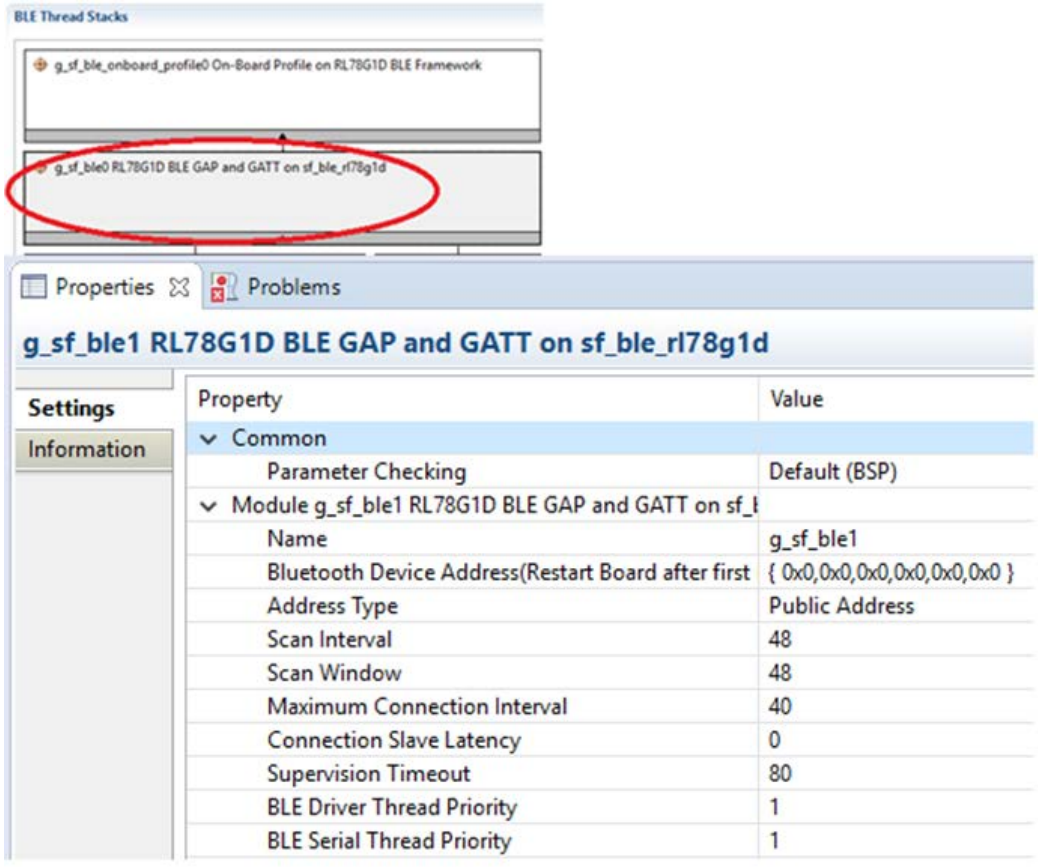


Table 2 Configuration properties of BLE framework module

Property	Default Value	Description
Name	g_sf_ble0	BLE framework instance
Bluetooth Device Address	{0x0, 0x0, 0x0, 0x0, 0x0, 0x0}	Bluetooth device address
Address Type	Public Address	Address type, can be Random Address or Public Address. Public Address: This is an address that includes an allocated 24-bit OUI (Organizationally Unique Identifier) registered with IEEE. Random Address: This is an address that contains a random number and belongs to one of the following 3 categories: Static Address Non-Resolvable Private Address Resolvable Private Address.
Scan Interval	48	This is the interval for receiving advertising data and is in units of 0.625 ms. The allowed range is between 2.5 to 10240 ms.
Scan Window	48	This is the period of time during which advertising data is received at the scan interval. It is in units of 0.625ms. The allowed range is between 2.5 to 10240 ms.
Maximum Connection Interval	40	This is the interval for transmitting and receiving data periodically following connection establishment. It is in units of 1.25 ms. The allowed range is between 7.5 ms to 4 s.
Connection Slave Latency	0	This is the period of time during which data is transmitted and received at the connection interval. The allowed range is between 0 and 500 ms.
Supervision Timeout	80	This is the timeout interval after which the link is considered to have been lost when no response is received from the peer device. It is in units of 10 ms. The allowed range is between 100 ms to 32 s.
BLE Driver Thread Priority	1	The BLE driver thread priority.
BLE Serial Thread Priority	1	The BLE serial thread priority.

The following screenshot, shows the configuration properties of the on-board generic BLE profile framework.

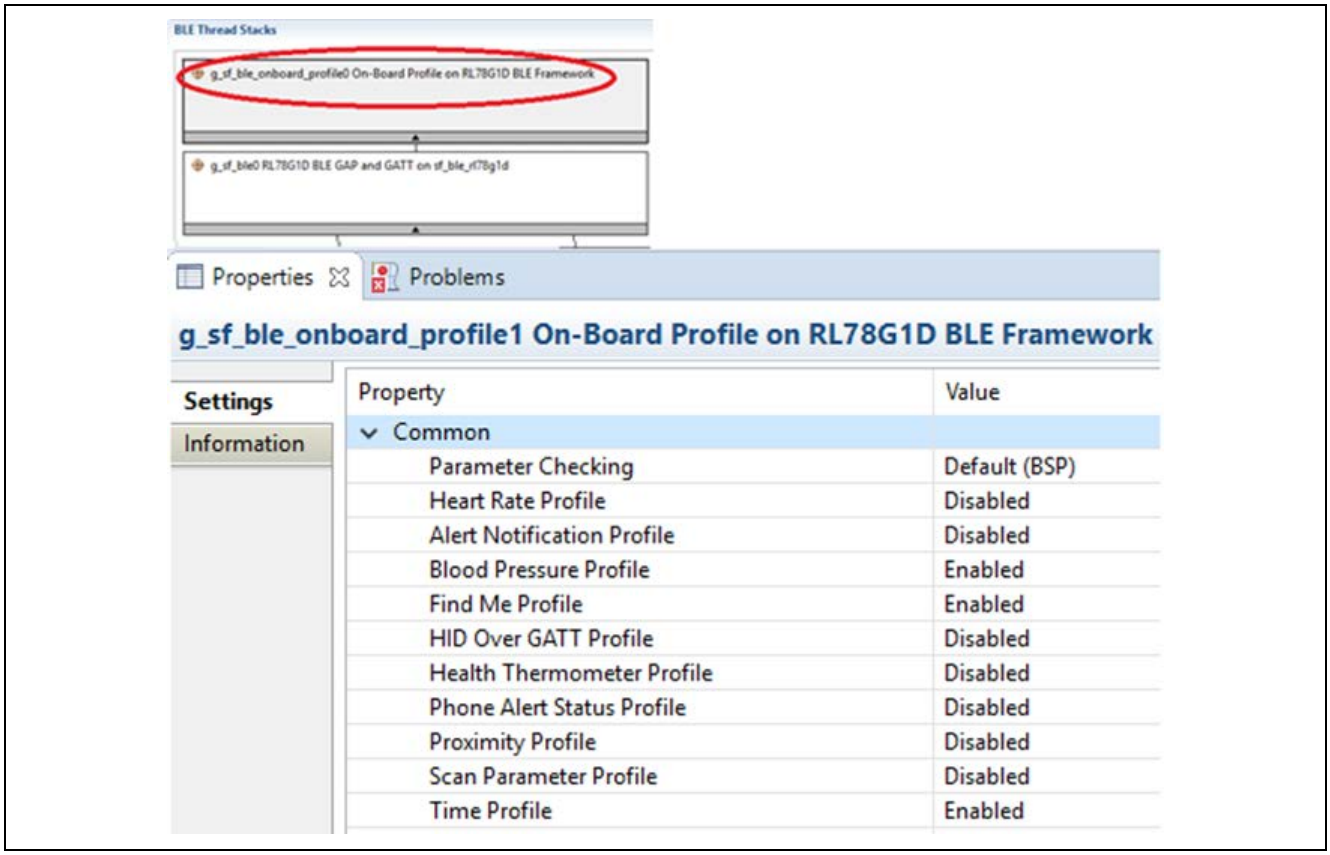


Table 3 Configuration properties of on-board profile framework

Property	Default Value	Description
Heart Rate profile	Disabled	BLE Heart Rate profile, can be enabled or disabled based on need
Alert Notification profile	Disabled	BLE Alert Notification profile, can be enabled or disabled based on need
Blood Pressure profile	Disabled	BLE Blood pressure profile, can be enabled or disabled based on need
Find Me profile	Enabled	BLE Find Me profile, can be enabled or disabled based on need
HID over GATT profile	Disabled	BLE HID over GATT profile, can be enabled or disabled based on need
Health Thermometer profile	Disabled	BLE Health Thermometer profile, can be enabled or disabled based on need
Phone Alert Status profile	Disabled	BLE Phone Alert Status profile, can be enabled or disabled based on need
Proximity profile	Disabled	BLE Proximity profile, can be enabled or disabled based on need
Scan Parameter profile	Disabled	BLE Scan Parameter Profile, can be enabled or disabled based on need
Time profile	Disabled	BLE Time Profile, can be enabled or disabled based on need

6. BLE Framework Module Application Example

6.1 Overview

This example application project demonstrates the Find Me Profile operation of Synergy BLE framework. The Find Me Target utilizes the Find Me Profile with one instance of the Immediate Alert Service to display alerts if the Client configured the device for modification. Find Me Target operates with other devices that implement the Find Me Locator Profile.

The Find Me profile defines 2 roles:

- The Find Me Target is the GATT server
- The Find Me Locator is the GATT client

The following figure shows the relationship between the services and the profile roles.

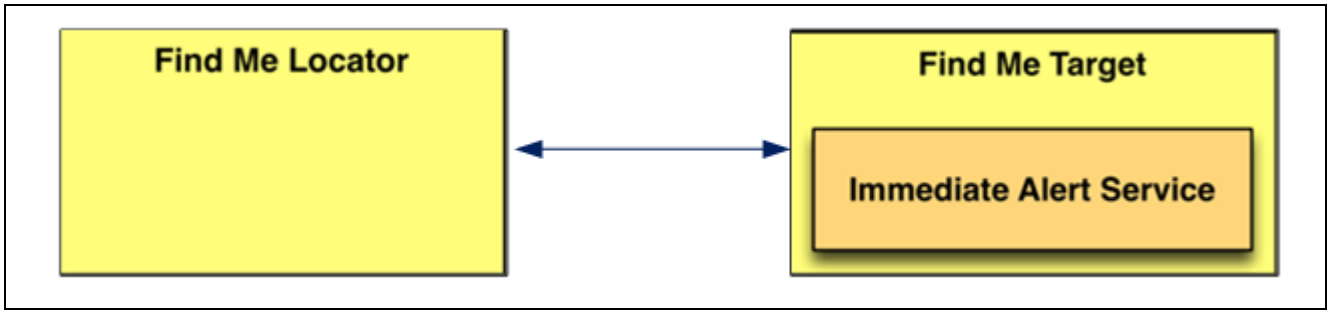


Figure 8 Role and service relationship

The Find Me Target has an instance of the Immediate Alert Service. In this BLE application example, the SK-S7G2 kit acts as the Find Me Target and the application like BLE Scanner APK running on Android phone or LightBlue APK running on iPhone acts as the Find Me Locator.

6.2 BLE application software architecture overview

This section provides the software architecture overview of the BLE Framework application example.

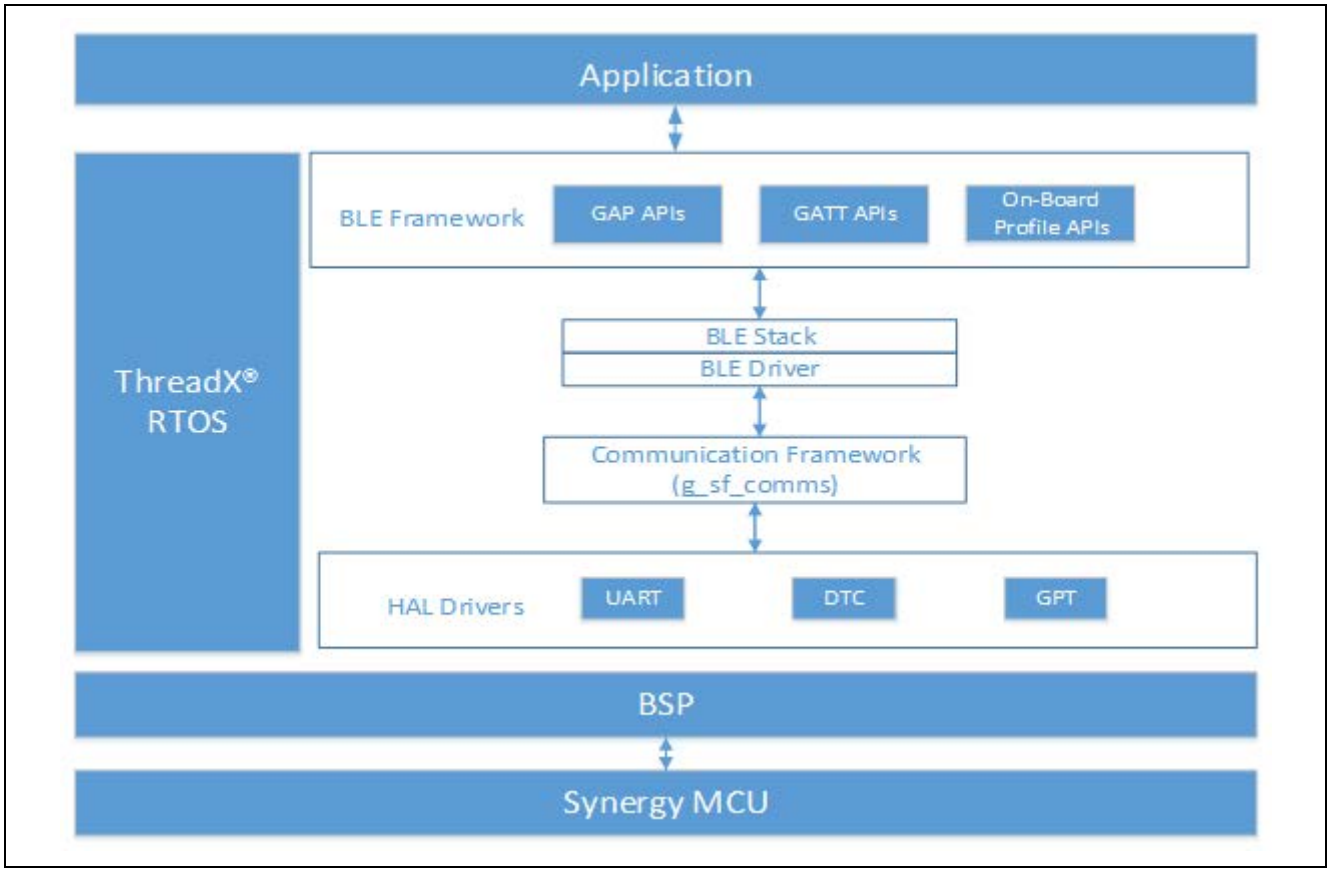


Figure 9 BLE Find Me Target application software architecture

The main software components of the BLE Find Me Target application are:

- BLE thread
- BLE framework
- Communication framework.

This BLE application example demonstrates the core functionality of Synergy BLE framework, configured using the Find Me Profile on RL78G1D BLE module. The RL78G1D BLE module acts as the Find Me Target. This project has GAP role configured as peripheral-only mode.

The BLE thread includes the Synergy BLE Framework for the RL78G1D module and its associated BLE stack and BLE device drivers, including SSP modules such as UART, DTC, and GPT to support data transfers between two BLE devices. The BLE thread handles all BLE communications using the underlying BLE framework such as initialization, provisioning, scanning, advertising, and data transferring between two BLE devices.

Callbacks:

There are two user callbacks registered to the underlying BLE framework:

1. User_ble_callback
2. Fmpt_callback

User_ble_callback:

During initialization, the BLE thread registers a callback to the BLE framework and receives notifications for events such as connect, disconnect, bonding, GATT notification/indication from the underlying BLE framework.

Fmpt_callback:

During the Find Me Profile Target enable, the BLE thread registers the `fmpt_callback` to receive notifications for Find Me profile specific events such as alert level change from the underlying BLE framework.

The BLE thread runs on a state machine with the following states. At any one point of time, the BLE thread will be in one of the following states.

1. Init state
2. Connect state
3. Activate profile
4. Handle profile events
5. Disconnect state

Init State:

You will need to manually fill in the `sf_ble_provisioning_t and sf_ble_adv_info_t` structures based on your application design.

During initialization, the BLE thread does provisioning and starts advertisement by passing the above structures to the underlying BLE framework. The `user_ble_callback` callback function will be registered to receive notifications from the BLE framework.

After initialization, the BLE thread sets to Connect state.

Connect State:

The device already starts advertising and waiting for the BLE client to initiate the connection. In this state, the BLE thread waits for the connect event from the BLE framework.

Once the connection event is received, the BLE thread sets to the Activate profile state.

Activate Profile:

The device is already connected to the BLE client device. The BLE thread will enable the Find Me profile and register the `fmpt_callback` routine to receive the Find Me profile specific notifications.

Once the profile is activated, the BLE thread sets to the Handle profile events state.

Handle profile events:

The BLE thread handles the Find Me profile events. The BLE thread stays in this state until it received the disconnect event from the BLE framework. Then it moves to the disconnect state.

The Find Me Profile defines the behavior when a button is pressed on a device to cause an immediate alert on a peer device. This can be used to locate devices that are misplaced.

The LED2 is used to demonstrate the Alert level. If the Alert level is set to `MILD_ALERT` from the client, the LED2 starts blinking. If the Alert level is set to `HIGH_ALERT` from the client, the LED2 is turned ON. To clear the alerts,

send a request from the client to set the Alert Level Characteristics to `NO_ALERT`. The alerts are also cleared when the connection with the client is canceled or lost.

Disconnect state:

At this state, the BLE device is disconnected from the BLE client and received the disconnect event from the BLE framework.

The BLE thread disables the Find Me profile, turns off user LED and sets the state to Init state.

6.3 Configuration

The following steps are used to configure the Synergy BLE Framework modules in this application example using the e^2 studio ISDE.

In this section, we take the SK-S7G2 Synergy MCU Group board as a reference kit and the configurations relevant to a hardware platform are done for the SK-S7G2 Synergy MCU Group board.

1. Set the `g_sf_ble0` Properties as shown in the following figure. You can set the Bluetooth address of your choice. To see the changed address, restart the board after the first run.

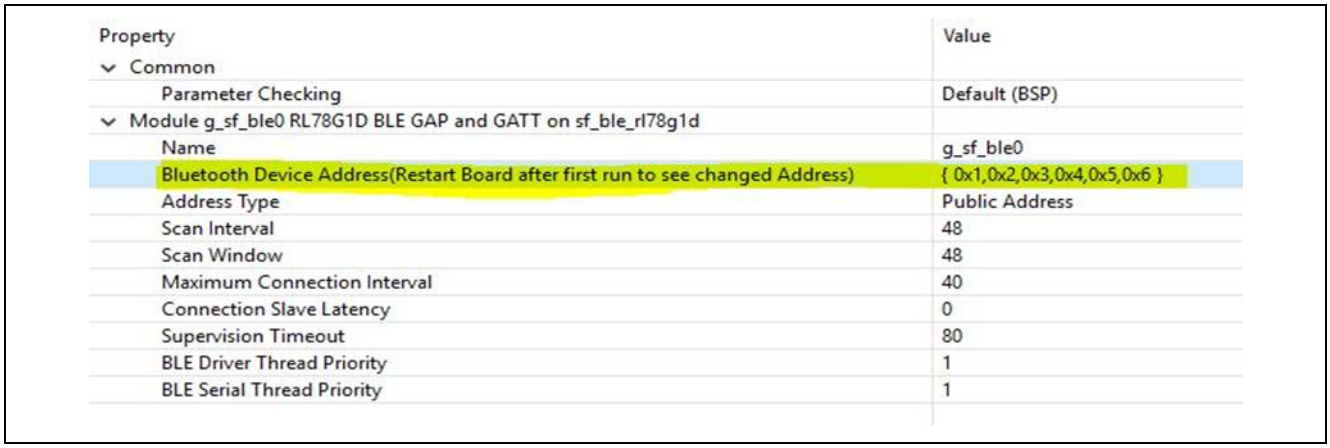


Figure 10 Properties configuration of g_sf_ble0

2. This project uses the Find Me Profile to demonstrate the Synergy BLE Framework functionality. Enable the Find Me Profile from the `g_sf_ble_onboard_profile0` Property window as shown in the figure below.

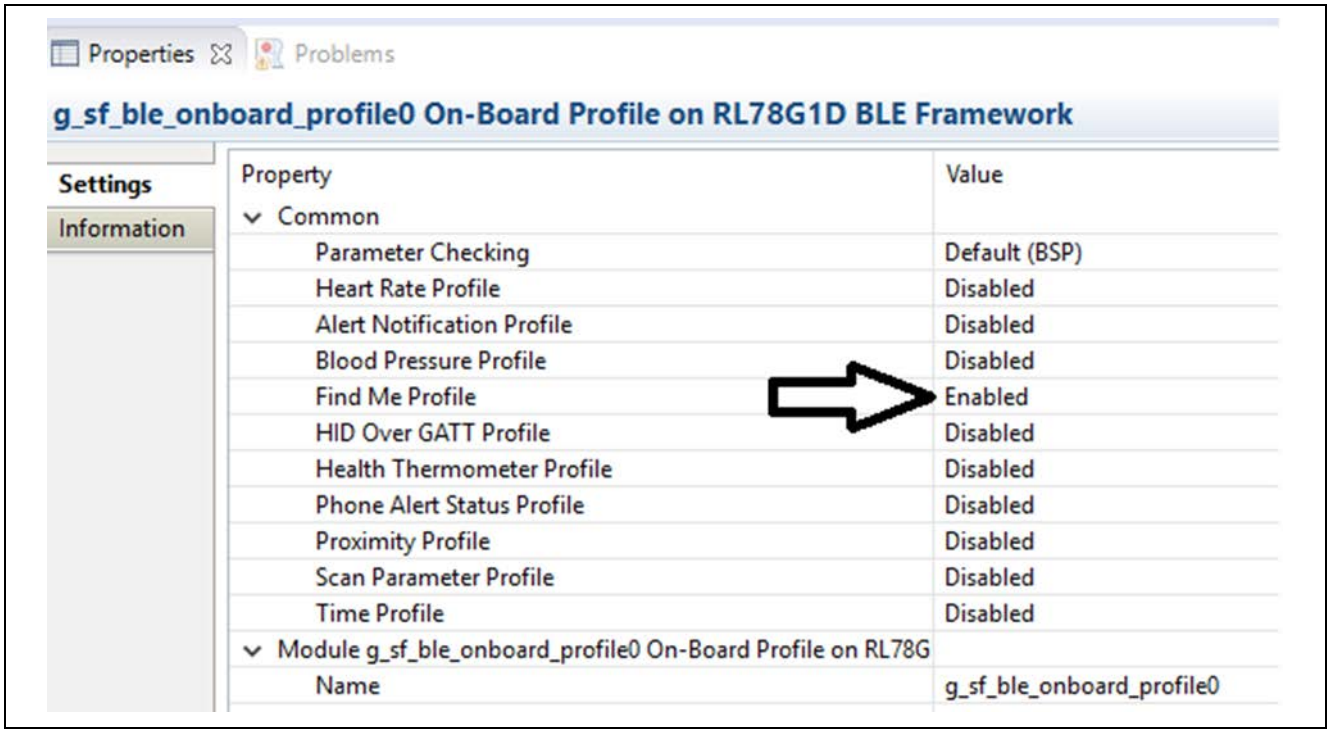


Figure 11 Properties configuration of g_sf_ble_onboard_profile

3. Set Channel to 6 and Baud Rate to 4800 in the Property window `for r_sci_uart` as shown in the following figure. Refer to the *UART Module Guide* for or more details on the UART driver properties. Use this link to download the *UART Module Guide*.

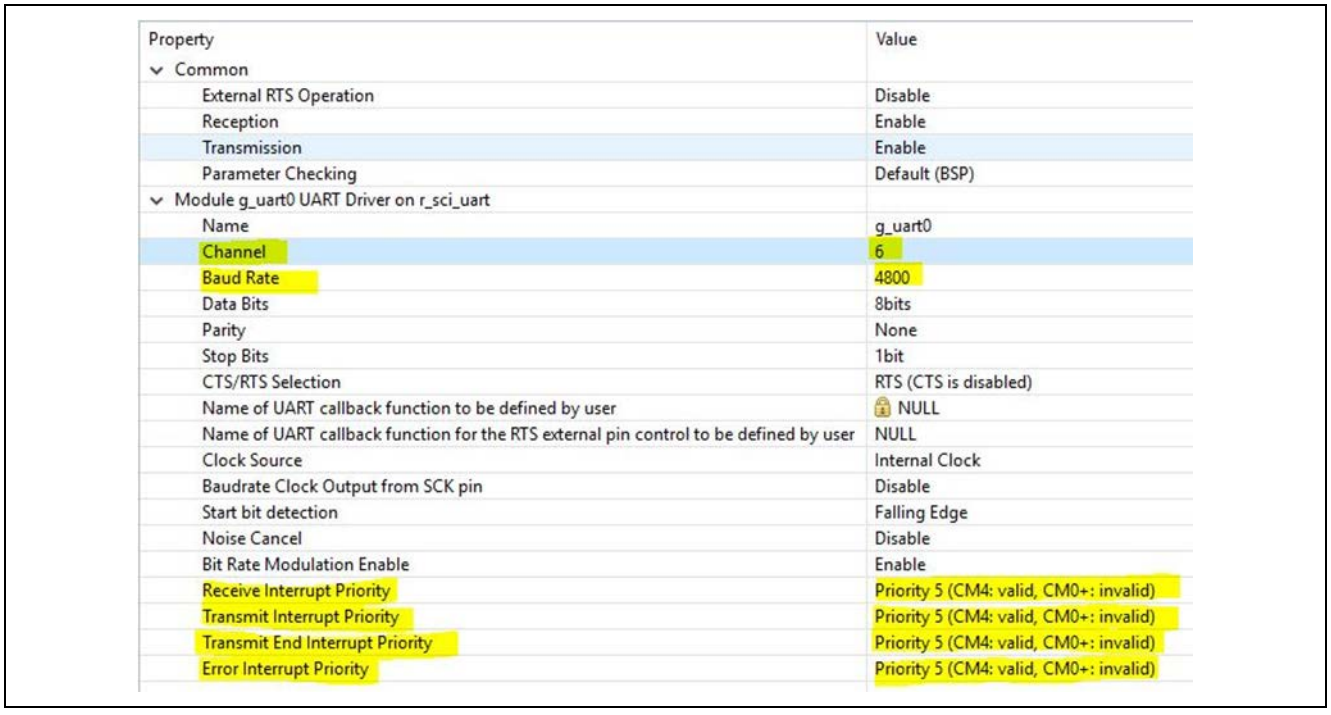


Figure 12 Properties configuration of g_uart0

4. Set the `g_timer0` properties as shown in the following figure. Refer to the *GPT Module Guide* for more details on the GPT Driver Properties. Use this link to download the *GPT Module Guide*.

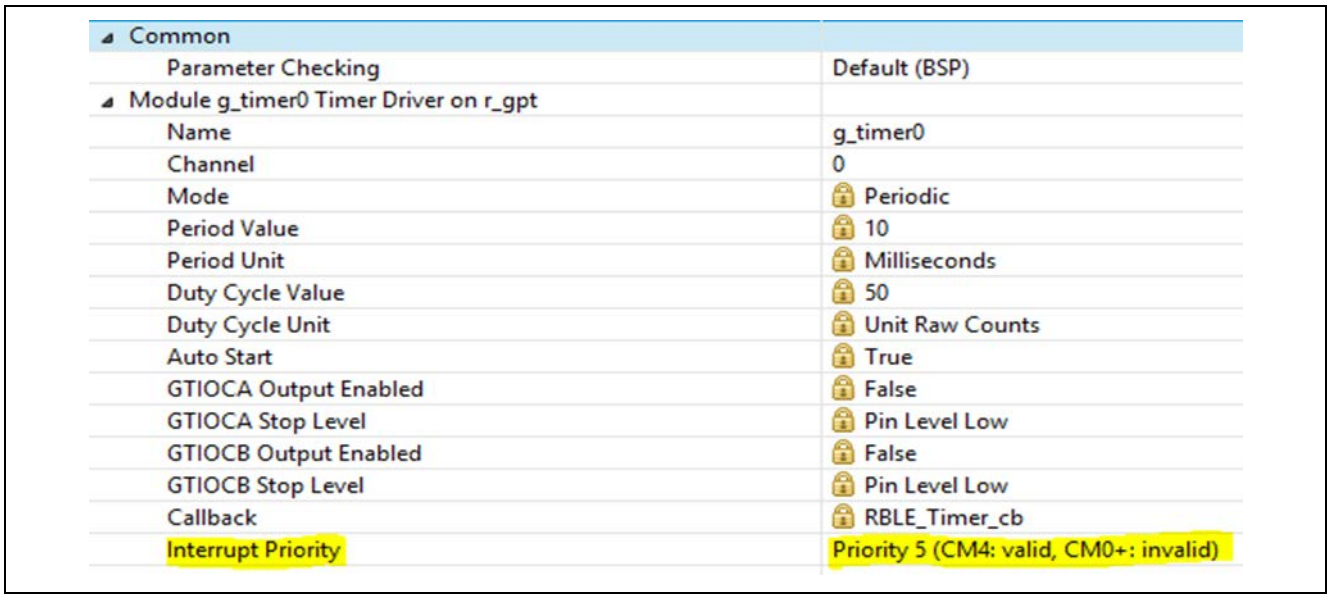


Figure 13 Properties configuration of g_timer

Pin Configuration:

Go to the Pins tab and set the following pin configuration for the SK-S7G2 board.

SCI pins

1. For SK-S7G2 board, SCI6 is used.
2. From the Pins tab, go to the Pin Selection section.
3. Go to Peripherals → Connectivity: SCI → SCI6
4. Set Operation Mode to Asynchronous UART. Select P304 and P305 for SCI use as shown in figure below.

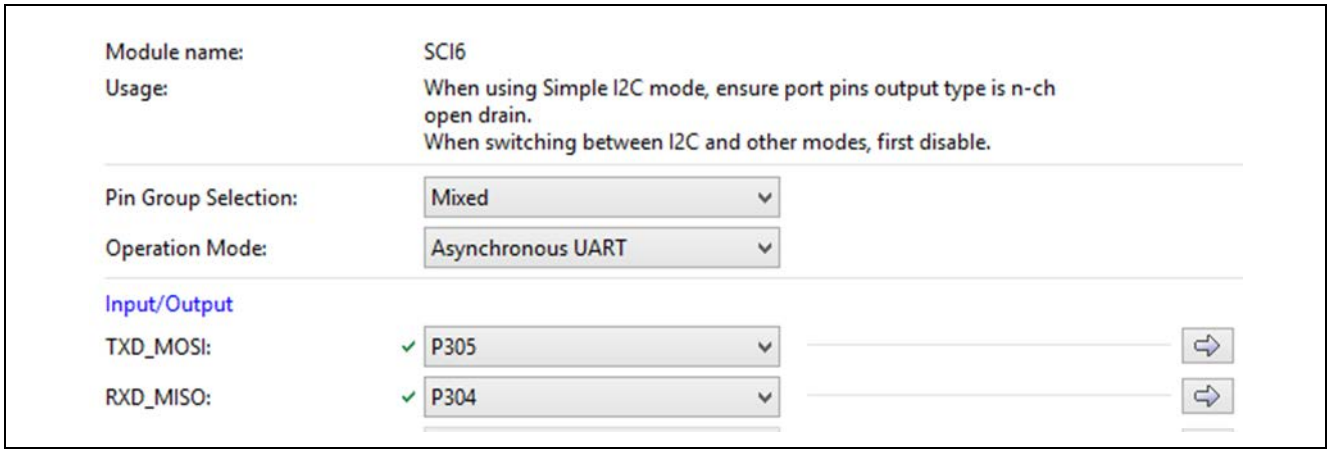


Figure 14 Setup for SCI6 pins configuration for SK-S7G2 board

Setup the RESET pin for SK-S7G2

1. From the Pins tab, go to the Pin Selection section
2. Go to Ports → P3 → P309.

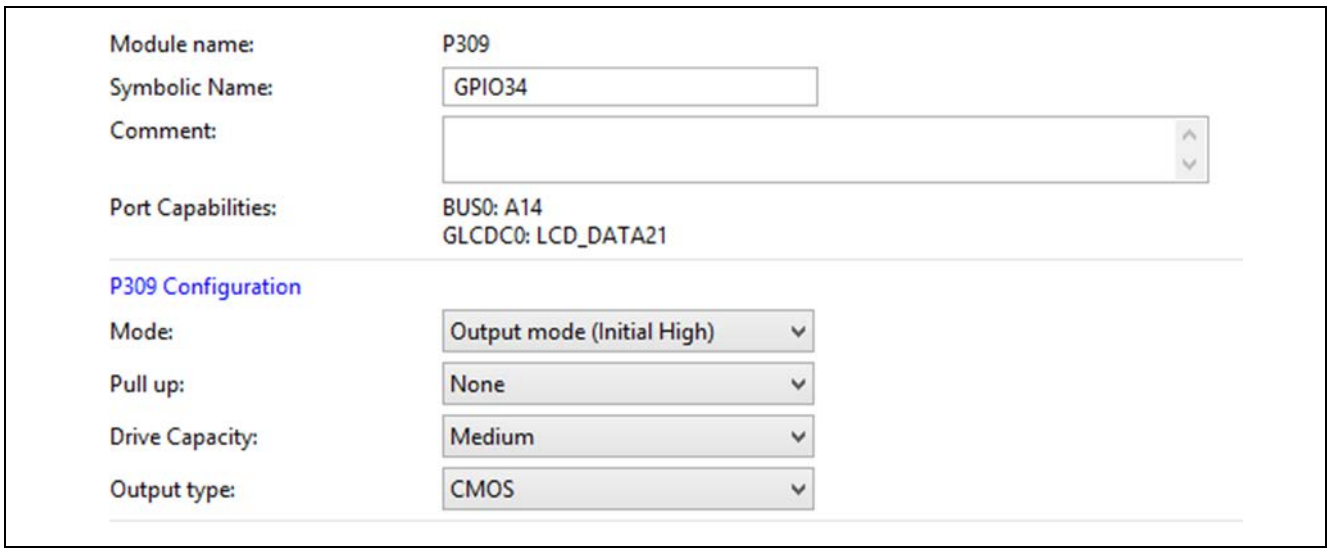


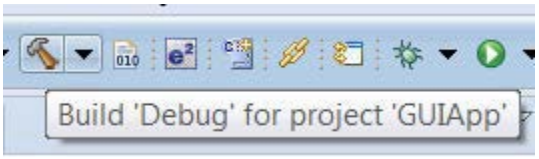
Figure 15 Setup for RESET pin configuration for SK-S7G2 board

When you finish the configuration for your application project, generate the project content by clicking **Generate Project Content**. This generates the project files using the configuration option you selected.

After the e^2 studio ISDE generated the application project files with the chosen configuration, go to the project explorer window under your project, and open the `src` folder to see the relevant files generated for this application project.

These files are place holder for adding the user application code. You can either write your own application functions or copy the existing source files from the `BLE_FindMe_SK_S7G2` demo application project to recreate this demo.

Build the application project by clicking the hammer icon from the menu bar.



7. Running the BLE Framework Module Application Example

7.1 Powering up the board

This section describes how to connect power to the board, the J-Link® debugger to the PC, the board to the PC USB port, and how to run the debug application.

To connect to the board:

1. Connect the micro USB end of the supplied USB cable to the SK-S7G2 board J19 connector (DEBUG_USB).

Note: The kit contains a SEGGER J-Link® On-board (OB). The J-Link provides full debug and programming capabilities for the SK-S7G2 board.

2. Connect the other end of the USB cable to the USB port on your workstation.

7.2 RL78G1D firmware programming

The RL78G1D BLE module must be programmed based on your application before running the BLE application demonstration. You need to manually copy one of the below .hex files based on your application into a USB flash drive.

The following steps show instructions to flash the firmware for the on-board RL78G1D onto the SK-S7G2 Synergy MCU Group board.

1. After you have successfully compiled the `BLE_FindMe_SK_S7G2` project:
   A. Go to debug→ debug configurations → Renesas GDB Hardware Debugging → Your project debug **BLE_FindMe_SK_S7G2 Debug**.
   B. Click the Browse button and select the Programmer.hex file stored in your PC as shown in the figure below. This file is given as part of the BLE Framework Module Application Example package.
2. Go to debugger and set the target device as R7FS7G2.
   A. Synergy → Synergy/CM4 → R7FS7G2.
3. Click the debug button.
4. Once the image is flashed, terminate the project.
5. Restart the device. When the display on the board reads **Looking for a USB device**, connect the USB loaded with the firmware file RL78_G1D_IM (FMP).hex. Follow the instructions on the display to flash the hex file for GATT or specified on-board profile.

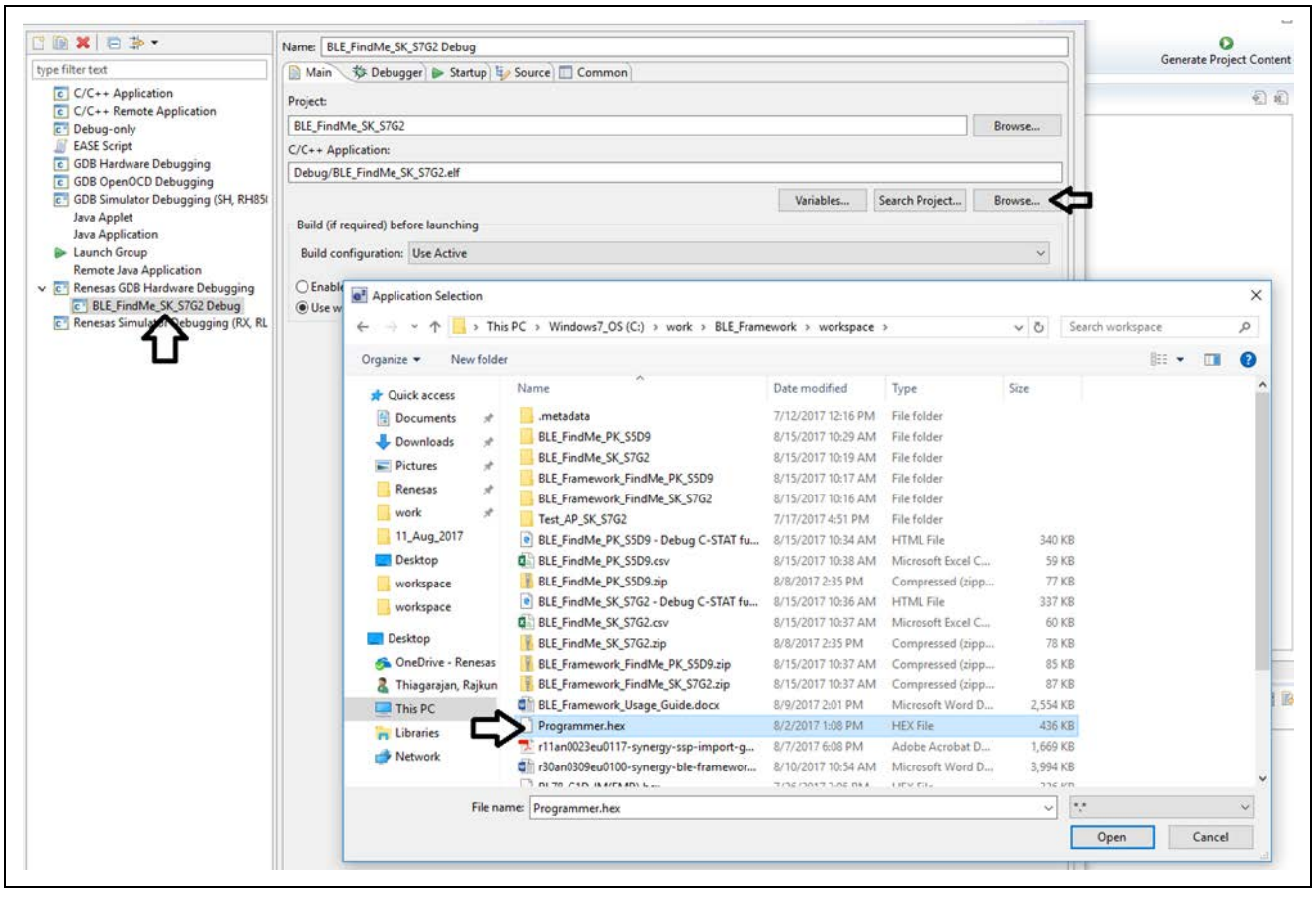


Figure 16 Programming RL78G1D BLE Hardware Module

For GATT and on-board profiles different files are flashed.

1. For GATT, flash the `RL78_G1D_IM (SCP).hex` file. Connect a USB device with `RL78_G1D_IM (SCP).hex` file loaded.
2. For on-board profiles, the following hex files are listed. Only one hex file can be flashed at a time. These files are given as part of the BLE framework module application example.

RL78_G1D_IM (GLP, PASP, TIP).hex
This file has the following combination of profiles:
- Glucose profile
- Phone Alert Status profile

RL78_G1D_IM (HOGP, ScPP).hex
This file has the following combination of profiles:
- HID over GATT profile
- Scan Parameter profile.

RL78_G1D_IM (HTP, BLP, HRP).hex
This file has the following combination of profiles:
- Health Thermometer profile
- Blood Pressure profile
- Heart Rate profile.

RL78_G1D_IM (PXP, FMP, ANP).hex
This file has the following combination of profiles:
- Proximity profile
- Find Me profile
- Alert Notification profile.

3. Once the code is flashed, unplug the USB device and restart the board.

Note: These .hex files support multiple profiles and their associated mandatory services. If you program one of these .hex files and enable one of the supported profiles, then all the services associated with other profiles that are part of the .hex file are enabled by default.

7.3 Importing, building, and running the project

Refer to the *SSP Import Guide* (r11an0023eu0119-synergy-ssp-import-guide.pdf) for instructions on importing the project into e^2 studio and building/running the project.

Note: You need to select **BLE_FindMe_SK_S7G2 Debug** GDB Hardware Debugging configuration for debugging.

7.4 Verifying the demonstration

See sections 5.2.1 to 5.2.3, and follow the steps to power up the SK-S7G2 MCU board, flash the RL78G1D firmware, and run the existing BLE Framework Application example project.

The client device can be either another board that runs the Find Me Locator application, or the standard BLE application such as the BLE Scanner, running on Android or IOS devices. In this document, the BLE Scanner APK running on Android devices is used as the BLE Client device.

Once the BLE application is running on the SK-S7G2 MCU board, open the BLE Scanner application on your android phone and scan for devices. The SK-S7G2 board is displayed as SynergyBLE device as shown in the following figure.

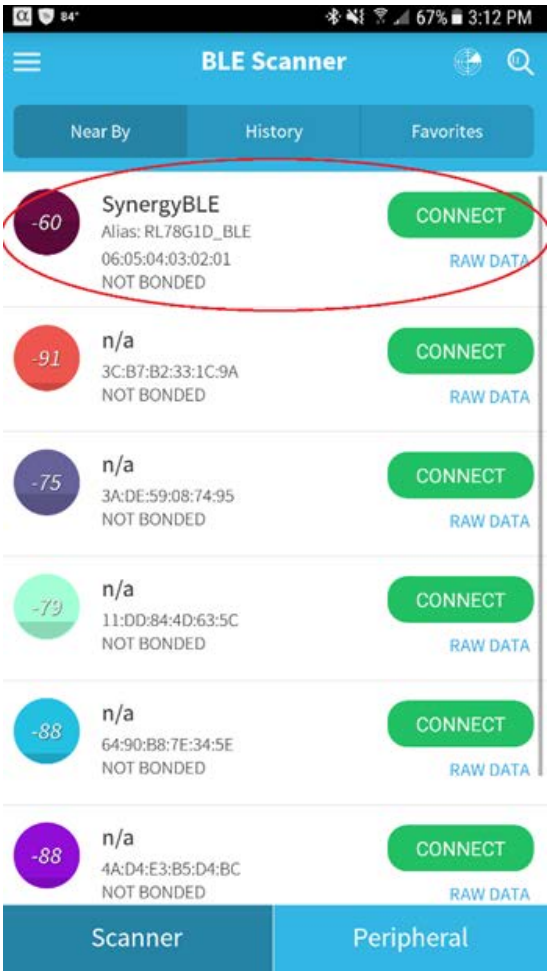


Figure 17 BLE scan window

When the Synergy BLE device is displayed in the window, connect to the device by clicking the CONNECT button. After successful connection, the BLE Scanner APK opens a new window with the list of services supported for this profile as shown below in the figure.

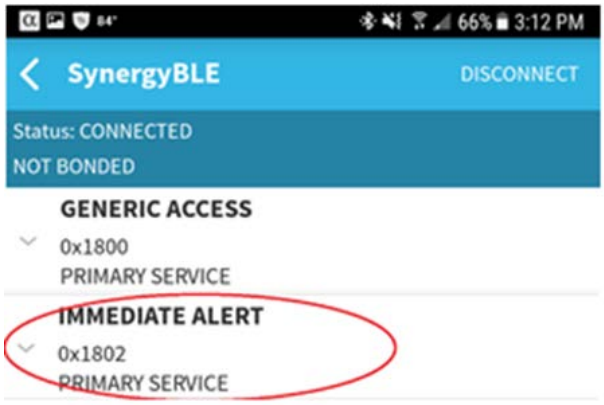


Figure 18 BLE Find Me Profile Services

Expand the IMMEDIATE_ALERT service by clicking the downward arrow next to the service. It shows the properties associated with that service. Click the W button to send an alert level to the BLE server (SK-S7G2 MCU board) as shown in the following figure.

Figure 19 Triggering BLE Find Me Profile Alert Level

A drop-down menu is displayed with the following ALERT LEVEL as shown in the following figure.

Figure 20 BLE Find Me profile Alert Level

Choose the alert level and press the **OK** button. The BLE Client sends out the alert level to the BLE Server application running on the SK-S7G2 MCU board.

Based on the ALERT LEVEL, the LED2 blinks as shown in the following table.

Table 4 BLE Find Me profile Alert Level

Alert Level	LED 2 status
No Alert	OFF
Mild Alert	Blinking continuously
High Alert	ON

8. Next Steps

1. Visit renesassynergy.com/tools to learn more about development tools & utilities.
2. Visit http://www.renesassynergy.com/gallery to download development tools & utilities.
3. To learn more about:
   - Synergy kits at http://www.renesassynergy.com/kits
   - Synergy Microcontrollers at http://www.renesassynergy.com/microcontrollers
   - Synergy Software at http://www.renesassynergy.com/software
   - Synergy Solutions at http://www.renesassynergy.com/solution
4. Procuring RL78G1D BLE module.

   The RL78G1D BLE module is mounted on SK-S7G2 and PK-S5D9 kits.
5. Renesas Synergy Module guides collateral link

   https://www.renesas.com/en-us/products/synergy/tools-kits.html#sampleCodes

9. References

1. *SSP 1.4.0 User Manual* can be downloaded from the Renesas Synergy™ Gallery (http://www.renesassynergy.com/gallery)
2. *BLE Find My Profile Specification*.

Website and Support

Support: https://synergygallery.renesas.com/support

Technical Contact Details:

- America: https://www.renesas.com/en-us/support/contact.html
- Europe: https://www.renesas.com/en-eu/support/contact.html
- Japan: https://www.renesas.com/ja-jp/support/contact.html

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Aug 30, 2017	-	Initial release
1.01	Sep 25, 2017	-	Added RL78G1D_HexFiles.zip to the download package. No changes to the document.
1.02	Oct 19, 2017	-	Repackaged with new zip file
1.03	Oct 27, 2017	-	Updated for v1.3.2 release
1.04	Mar 22, 2018	-	Updated for SSP v1.4.0

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.0