Tutorial

Advertising Concept

(For the DA1468x Devices)

Abstract

This tutorial should be used as a reference guide to gain a deeper understanding of the 'Advertising Concept'. As such, it covers a broad range of topics including a brief introduction to Bluetooth Low Energy (BLE) and the usage of the SmartSnippets Toolbox. Furthermore, it covers a number of sections containing in depth software analysis of various advertising concepts.

Figures

Figure 1: Bluetooth low energy protocol stack 3
Figure 2: Advertising data packet 4
Figure 3: First Step 5
Figure 4: Second Step 6
Figure 5: Third Step 6
Figure 6: Connection parameter update 12
Figure 7: Verifying the Bluetooth low energy device output using a scanner app 15
Figure 8: Exploring the various services after connecting with a remote peer (using a scanner app) 15
Figure 9: Initializing the SmartSnippets Toolbox 16
Figure 10: Verifying advertising interval 16
Figure 11: Verifying connection parameter update 17
Figure 12: Verifying connection parameter update 17
Figure 13: Verifying advertising channel map and interval 18
Figure 14: Verifying advertising channel map 18

Tables

Table 1: Advertising type enumeration 8
Table 2: Advertising ATT permissions 8
Table 3: Advertising intervals 10
Table 4: Advertising channels 11
Table 5: Advertising modes 11
Table 6: Update connection parameter values 14

Terms and Definitions

BD	Bluetooth
GAP	Generic Access Profile
ISM	Industrial Scientific Medical
LE	Low Energy
ms	Millisecond
PDU	Protocol Data Unit

References

[1] UM-B-044, DA1468x Software Platform Reference, User Manual, Dialog Semiconductor.
[2] Naresh Gupta, “Inside BLEUTOOTH LOW ENERGY”, ARTECH HOUSE, 2013.

Advertising Concept Tutorial

Introduction

1.1 Before You Start

Before you start you need to

- Install the latest SmartSnippets Studio
- Download the latest SDK (currently version 1.0.10.1072)

These can be downloaded from the Dialog Semiconductor support portal.

Additionally, for this tutorial either a Pro or Basic Development kit is required.

The key goals of this tutorial are to:

- Provide a basic understanding of the Advertising Concept
- Explain how to change the advertising contents and connection parameters with respect to Dialog SDK
- Verify the correct functionality of the Bluetooth low energy device using the SmartSnippets Toolbox (integrated in the SmartSnippets Studio)

1.2 Bluetooth Low Energy Protocol

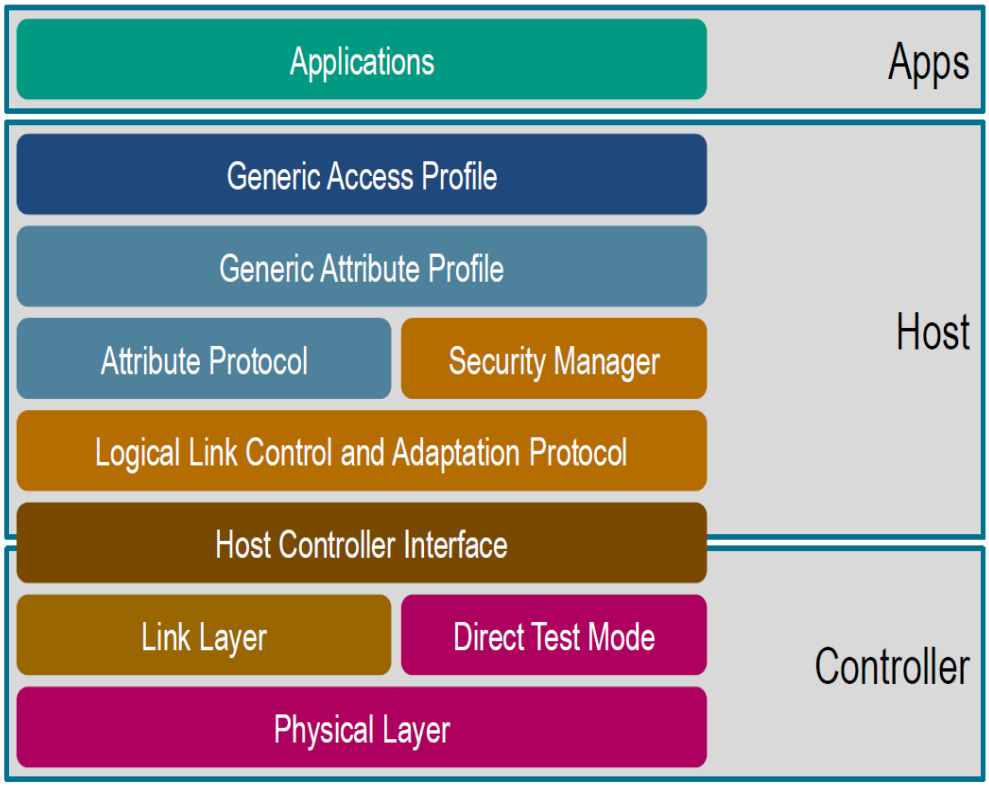

Figure 1: Bluetooth Low Energy Protocol Stack

1.3 Advertising Concept

Bluetooth low energy can broadcast small packets of data containing advertisements to peer devices.

- An Advertising packet is small and has a well-defined format. As a result, only a restricted amount of user data can be carried
- The Advertising mode also support transmission of a secondary scan Response packet which contains additional data. This data can be requested by a potential client using a Scan Request without establishing a permanent connection to the device.
- An Advertising packet is made up of a number of fields which typically includes:
    - The name of the device
    - Some or all of the Services supported by the device
- Advertising packets may also contain proprietary manufacturer-specific data and flags declaring the capabilities of the device

1.3.1 Advertising with Respect to Bluetooth Low Energy

Bluetooth low energy implements two communication methods:

- Advertisement: A Bluetooth low energy peripheral device broadcasts packets to every device around it. The receiving device can then act on this information without establishing any connection (scan request) or it may also connect to receive further information.
- Connected: Communication is setup to receive packets using a physical connection link, where both the peripheral and central send packets.

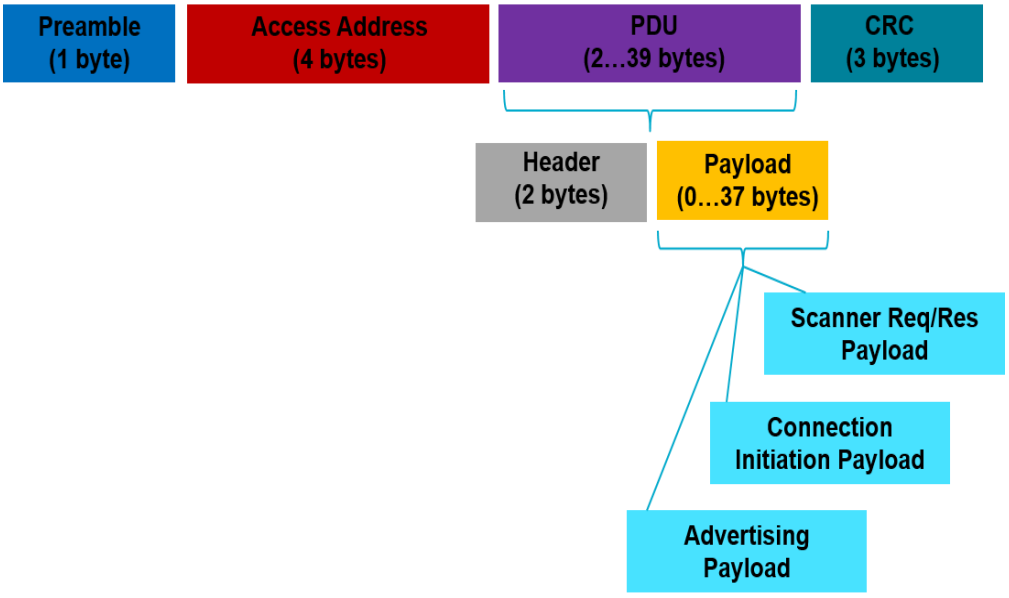

Figure 2: Advertising Data Packet

Advertising Concept

A **packet** can be 80 to 376 bits in length, and has the following fields:

Preamble

Used for internal protocol management. Advertising packets have 0xAA as preamble.

Access Address

This is always 0x8E89BED6 for advertising packets.

PDU

There are two PDU formats, one for advertising packets and one for data packets. The Advertising PDU consists of the 16-bit PDU header, and depending on the type of advertising, the device address (6 bytes) and up to 31 bytes of information. If the advertising mode allows it, the active scanner (through scan request) may request up to 31 bytes of additional information from the advertiser. This means that a sizeable portion of data can be received from the advertising device even without establishing a connection.

Setting the Bluetooth Address and Device Name

SmartBond™ device family uses a default Bluetooth Device (BD) address if the device developer has not assigned a specific address. This approach allows a device to be brought up quickly but proves inadequate as soon as multiple devices advertise using the same address. This section provides a step-by-step description of how to change the BD address and device name. It also explains the key parameters.

2.1 Importing a Project

The fastest way to get started with advertising is to examine the example application named *ble_adv* from our SDK. The First setup is to include the project in our current workspace:

1. In the SmartSnippet Welcome page, click on **Browse** under the **SOFTWARE RESOURCES** section.

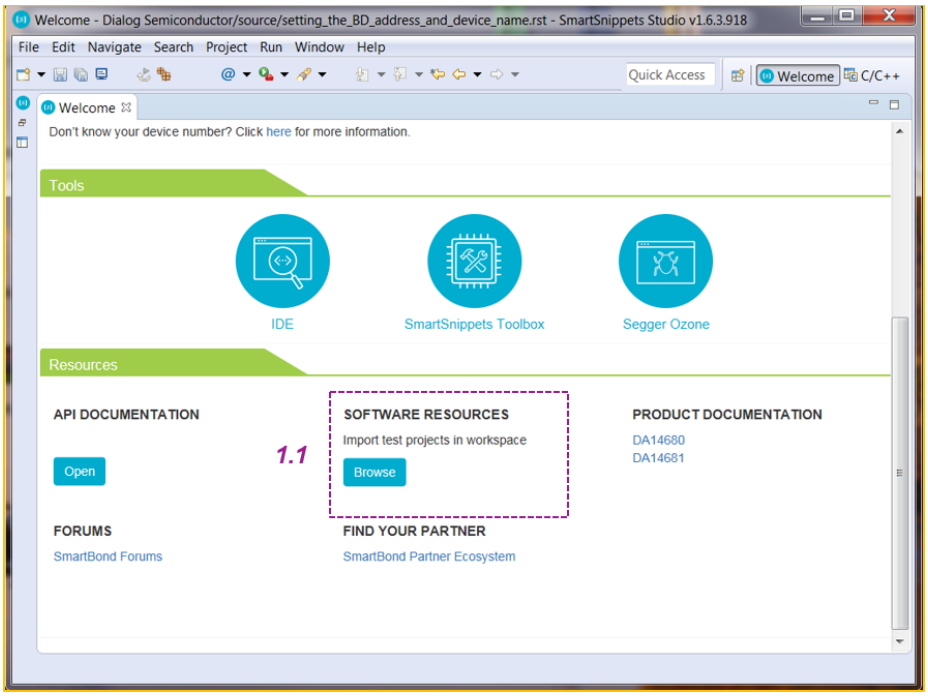

Figure 3: First Step

2. In the pop-up window, click **OK** as your current workspace folder should be automatically selected. If this is not the case, you must explicitly select it.

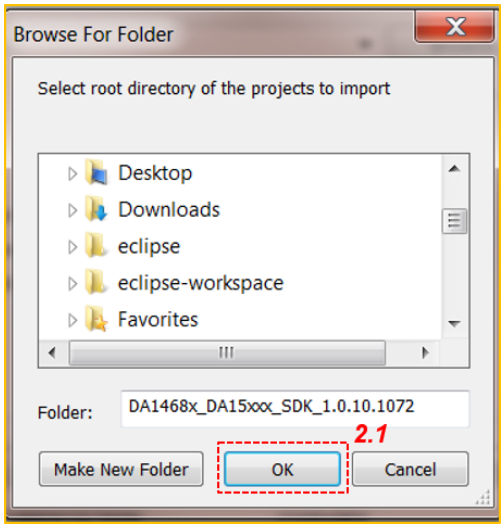

Figure 4: Second Step

3. The final step is to select the preferred project(s) to import in. By default all projects are selected. It is recommended to:

    1. Click **Deselect All.**
    2. Select the required projects by clicking on the respective **tick box**.
    3. Click **Finish.**

Now you are ready to start working with the project.

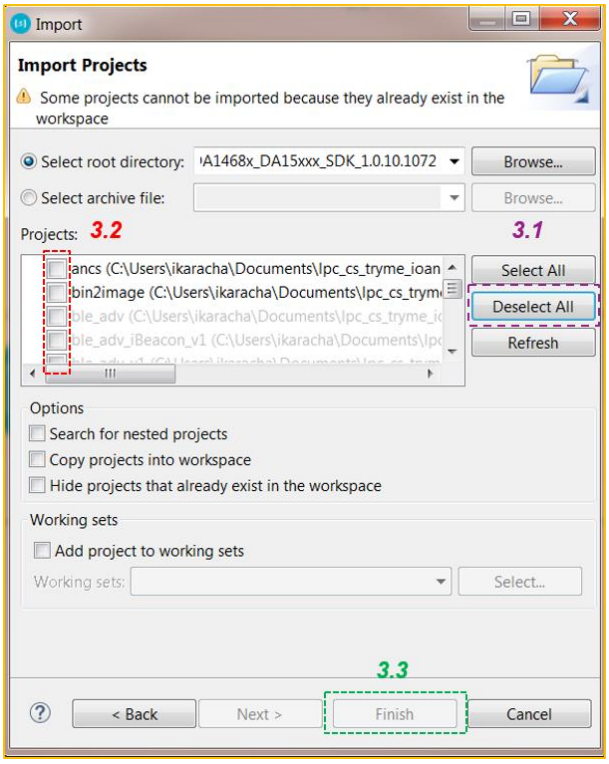

Figure 5: Third Step

2.2 Setting the BD Address

2.2.1 Using MACRO Definition

To change the BD address, follow the method in Using API Functions.

When no address is provided, the application uses the default address definition found in the */sdk/ble/config/ble_config.h* header file. It is not recommended to change this definition although, for some development purposes, it can be more practical than programming the address through the API. If possible, define the proper macro definition in the */ble_adv/config/custom_config_qspi.h* header file which is where all custom definitions should be declared.

Code snippet:

```
/* Add this macro definition in custom_config_qspi.h header file to overwrite the
 * default public address. The address will be displayed in reverse order
 * i.e. 06-05-04-03-02-01
 */

#define defaultBLE_STATIC_ADDRESS   {0x01, 0x02, 0x03, 0x04, 0x05, 0x06}
```

Note: The address in the above macro definition is the PUBLIC address. It is used if the user does not explicitly declare a custom address as described in Using API Functions.

2.2.2 Using API Functions

The recommended way of setting the device address is to use the GAP function which is responsible for setting the BD address

STEP #1 Add the following code snippet somewhere at the beginning of the *main.c* file.

Code snippet:

```
/*Initialize the BLE structure related to BD address value*/

static const own_address_t user_bd_address = {
  .addr_type = PRIVATE_STATIC_ADDRESS,
  .addr = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06}
};
```

Note: The device address type must be set to ***PRIVATE_STATIC_ADDRESS***. If this is not done, the default BD address is used (see Using MACRO Definition)

Table 1: *Advertising Type Enumeration*

Enumeration name	Value	Description
PUBLIC_STATIC_ADDRESS	0x0	Public Static Address
PRIVATE_STATIC_ADDRESS	0x1	Private Static Address
PRIVATE_RANDOM_RESOLVABLE_ADDRESS	0x2	Private Random Resolvable Address
PRIVATE_RANDOM_NONRESOLVABLE_ADDRESS	0x3	Private Random Non-resolvable Address

STEP #2 Call the corresponding GAP function to set the BD address with the defined value. (Place it immediately after the device name function.)

Code snippet:

```
/*Set BD address to the preferred value*/

ble_gap_address_set(&user_bd_address, 0x00FF);
```

Note: The second input parameter of this function, that is *0x00FF*, does not have any special meaning. It is only used if the address type is either ***PRIVATE_RANDOM_RESOLVABLE_ADDRESS*** or ***PRIVATE_RANDOM_NONRESOLVABLE_ADDRESS***.

2.3 Setting the Device Name

Open the *main.c* file in the *ble_adv* folder and change the device name as required.

Code snippet:

```
/*Set device name*/

ble_gap_device_name_set("Dialog TTT Demo", ATT_PERM_READ);
```

Table 2*: Advertising ATT Permissions*

Enumeration name	Value	Description
ATT_PERM_NONE	0x00	You are not permitted to take any action
ATT_PERM_READ	0x01	You are permitted READ only action
ATT_PERM_WRITE	0x02	You are permitted WRITE only action

ATT_PERM_READ_AUTH	0x04	You are permitted READ only action using authentication
ATT_PERM_WRITE_AUTH	0x08	You are permitted WRITE only action using authentication
ATT_PERM_READ_ENCRYPT	0x10	You are permitted READ only action using encryption
ATT_PERM_WRITE_ENCRYPT	0x20	You are permitted READ only action using encryption
ATT_PERM_KEYSIZE_16	0x80	Using Key size 16
ATT_PERM_RW	ATT_PERM_READ | ATT_PERM_WRITE	You are permitted both Read and Write action
ATT_PERM_RW_AUTH	ATT_PERM_READ_AUTH | ATT_PERM_WRITE_AUTH	You are permitted both Read and Write action using authentication
ATT_PERM_RW_ENCRYPT	ATT_PERM_READ_ENCRYPT | ATT_PERM_WRITE_ENCRYPT	You are permitted both Read and Write action using encryption

Changing Advertising Parameters

This section describes how to change the various advertising parameters. It covers all the necessary key elements related to advertising, specifically dealing with advertising data and intervals. It also describes the advertising channel map and mode.

3.1 Advertising Data

Step #1 Change the contents of the existing *adv_data[ ]* variable for the required advertising data.

Code snippet:

```
/*Bluetooth low energy adv demo advertising data*/

static const uint8_t adv_data[] = {
0x14, GAP_DATA_TYPE_LOCAL_NAME,
'H', 'o', 'w', ' ', 'A', 'r', 'e', ' ', 'Y', 'o', 'u', ' ', 'T', 'o', 'd', 'a', 'y', ' ', '?'
};
```

Note: The first element of the array is the size of the data to be sent plus an extra null character, that is, *19 + 1 = 20* elements or *0x14* in hexadecimal format. If the wrong value is given, it is likely that Bluetooth low energy device will not to advertise at all so care must be taken when calculating this value.

3.2 Advertising Interval

Step #2 The advertising interval is the period for which a Bluetooth low energy peripheral device advertises. For this scenario we have adapted two different time slots - one in a high-speed mode and another in a lower speed. To switch between them, set the preferred mode to “1”.

Code snippet:

```
/* Depending on the advertising interval mode the corresponding code segment
  * is selected
  */
#if (FAST_ADV_INTERVAL == 1)
   static const uint16_t min = BLE_ADV_INTERVAL_FROM_MS(80);
   static const uint16_t max = BLE_ADV_INTERVAL_FROM_MS(100);
#else
   static const uint16_t min = BLE_ADV_INTERVAL_FROM_MS(1000);
   static const uint16_t max = BLE_ADV_INTERVAL_FROM_MS(1500);
#endif
```

Table 3*: Advertising Intervals*

MACRO	Value	Description
FAST_ADV_INTERVAL	0x0	Fast advertising mode
POWER_ADV_INTERVAL	0x1	Slow advertising mode. Use this mode in case of need for reduce power consumption.

Step #3 Call the corresponding GAP function to set the min-max advertising intervals (before starting advertising and after starting the Bluetooth low energy module as a peripheral device).

Code snippet:

```
/*Set advertising interval*/

ble_gap_adv_intv_set(min,max);
```

3.3 Advertising Channel Map

Step #4 The channel defined in the Bluetooth Core Specification consists of 37 data communication channels and 3 advertising channels used for device discovery. The latter are allocated in different parts of the spectrum to prevent interference from concurrent activities in the ISM Band. Specifically a Bluetooth low energy device can advertise on channels *37*, *38* and *39* which correspond to frequencies of *2.402 MHz*, *2.2426 MHz* and *2.480 MHz* respectively. SmartBond™ devices advertise successively in all enabled channels. By default, all channels are enabled. To force the Bluetooth low energy device to use only one channel, for example channel 37, use the following GAP function:

Code snippet:

```
/* Set advertising channel map */
gap_adv_chnl_t channel_map = GAP_ADV_CHANNEL_37

ble_gap_adv_chnl_map_set(channel_map);
```

Table 4*: Advertising Channels*

Enumeration	Name	Value Description
GAP_ADV_CHANNEL_37	0x00	Select channel 37 for advertising
GAP_ADV_CHANNEL_38	0x02	Select channel 38 for advertising
GAP_ADV_CHANNEL_39	0x04	Select channel 39 for advertising

3.4 *Advertising Mode*

Step #5 The advertising mode is also customizable by changing the input parameter of the *ble_gap_adv_start()* function. The default mode in this application is depicted in the following code snippet and forces the Bluetooth low energy module to advertise toward all devices in the outside world.

Code snippet:

```
/* Start advertising */

ble_gap_adv_start(GAP_CONN_MODE_UNDIRECTED);
```

Table 5: *Advertising Modes*

Enumeration	Name	Value Description
GAP_CONN_MODE_NON_CON	0x00	The Bluetooth low energy device just advertises without permitting connection with another central device.
GAP_CONN_MODE_UNDIRECTED	0x01	The Bluetooth low energy device advertises towards all devices regardless of their BD value.
GAP_CONN_MODE_DIRECTED	0x02	The Bluetooth low energy device advertises towards a device with specific BD address
GAP_CONN_MODE_DIRECTED_LDC	0x03	The Bluetooth low energy device advertises towards a device with specific BD address using Low Duty Cycle

Connection parameter update

This section describes how to initiate a connection parameter update procedure.

4.1 Introduction

It is important to understand the central/peripheral (master/slave) concept at the heart of the connected mode in the Bluetooth low energy protocol. The peripheral device, which is advertising, assumes the role of a slave device, while the scanner device, which is searching for a device to connect to, assumes the role of a master device upon a connection process. The latter is responsible for the various mandatory settings including in which channel to transmit and what event interval to use. However, following a successful connection, the slave device can propose its own preferred parameters via an **update connection parameter request**. After that, the master responds to the slave if its demands has been approved.

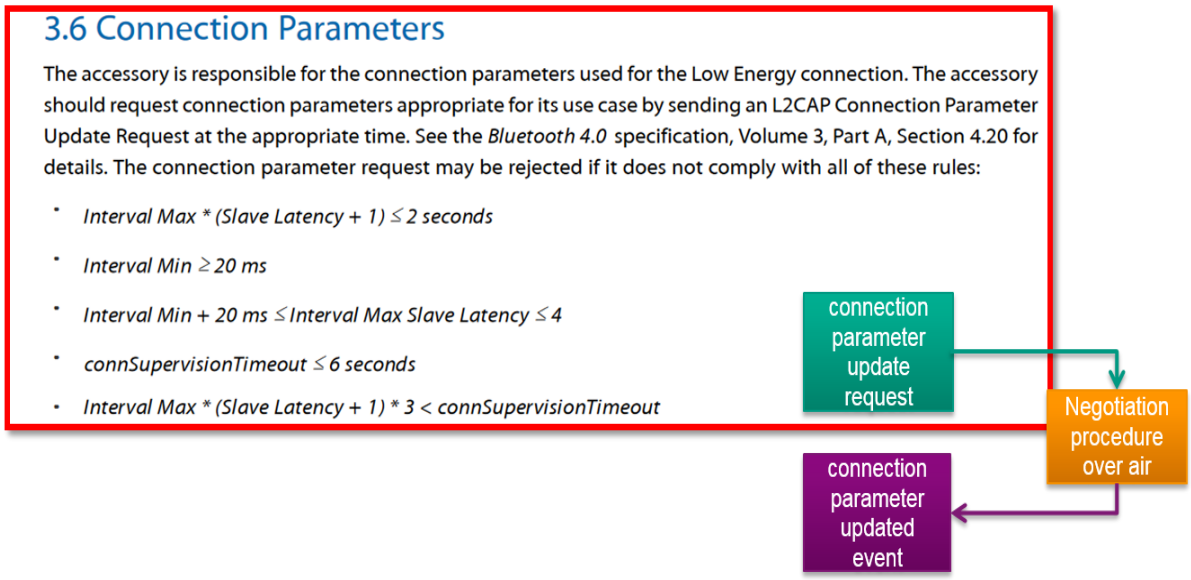

Figure 6: *Connection Parameter Update*

Note: There are some **restrictions** with regard to the timing intervals. For example, for an *iOS* platform the *Interval Min* must be different from the *Interval Max*.

4.2 Updating connection parameter

Step #1 In the *handle_evt_gap_connected()* handler, which is triggered when a connection is established, we create a software timer with a default timeout of *5* seconds. When it expires, a callback function is called and the update connection parameter process takes place.

```
/* Handler for successful connection establishment */
```

Advertising Concept

```
static void handle_evt_gap_connected(ble_evt_gap_connected_t *evt)
{
   /*
    * Manage behavior upon connection
    */
    connection_index = evt->conn_idx;

    /* Add a timer that when expired will renegotiate connection parameters. */
    update_timer = OS_TIMER_CREATE("conn_timer", OS_MS_2_TICKS(5000),  \
            OS_TIMER_FAIL , (uint32_t) OS_GET_CURRENT_TASK(), \
conn_param_timer_cb);
    OS_TIMER_START(update_timer, OS_TIMER_FOREVER);
}
```

Note: The second input parameter of the timer creation function (***OS_TIMER_CREATE***) is the timer period and has been set to 5 seconds. The last parameter is the callback function which is called when the timer expires.

Step #2 When timer expires, the *conn_params_timer_cb()* callback function is triggered. This function calls the *conn_param_update()* function which is responsible for initiating the update connection parameters process.

```
/* This timer callback notifies task that time for discovery, bonding and
 * encryption has elapsed, and connection parameters can be changed to the
 * preferred ones
 */
static void conn_param_timer_cb(OS_TIMER timer)
{
      /* Call the function which is responsible for the connection parameters
       * update
       */
      conn_param_update(connection_index);
}
```

Step #3 In the *conn_param_update()* function, change the parameters displayed in the code snippet:

```
/* Update connection parameters. */
static void conn_param_update(uint16_t conn_idx)
{
    gap_conn_params_t cp;

    cp.interval_min = defaultBLE_PPCP_INTERVAL_MIN;
    cp.interval_max = defaultBLE_PPCP_INTERVAL_MAX;
    cp.slave_latency = defaultBLE_PPCP_SLAVE_LATENCY;
    cp.sup_timeout = defaultBLE_PPCP_SUP_TIMEOUT;

    ble_gap_conn_param_update(conn_idx, &cp);
}
```

Step #4 In the */ble_adv/config/customer_config_qspi.h* header file, define the preferred update connection parameters values

```
/* Peripheral specific config */

#define defaultBLE_PPCP_INTERVAL_MIN   (BLE_CONN_INTERVAL_FROM_MS(500))
// 500ms

#define defaultBLE_PPCP_INTERVAL_MAX   (BLE_CONN_INTERVAL_FROM_MS(750))
// 750ms

#define defaultBLE_PPCP_SLAVE_LATENCY (0)  // 0 events

#define defaultBLE_PPCP_SUP_TIMEOUT (BLE_SUPERVISION_TMO_FROM_MS(6000))
//6000ms
```

Table 6: *Update Connection Parameter Values*

Macro name	Unit	Description
defaultBLE_PPCP_INTERVAL_MIN	ms	The Central device connecting to a Peripheral device needs to define the time interval for a connection to happen. This parameter is the minimum permissible connection time value to be used during a connection event.
defaultBLE_PPCP_INTERVAL_MAX	ms	The Central device connecting to a Peripheral device needs to define the time interval for a connection to happen. This parameter is the maximum permissible connection time value to be used during a connection event.
defaultBLE_PPCP_SLAVE_LATENCY	--	Defines the latency of the slave in responding to a connection event in consecutive connection events. This is expressed in terms of multiples of connection intervals, where only one connection event is allowed per interval.
defaultBLE_PPCP_SUP_TIMEOUT	ms	This parameter defines the LE link supervision timeout interval. It defines the timeout duration for which an LE link needs to be sustained in case of no response from peer device over the LE link.

Verify your advertiser

This section suggests some useful tools that can be used to verify if your SmartBond™ device is behaving as expected. It describes how to initialize the various tools and use them to verify the procedures described in previous sections of this document.

5.1 Advertised Data in the Air

When the project starts running, the Bluetooth low energy module will be shown up on the Bluetooth low energy scanner on your cell phone. In our case study, the following results were captured:

Advertising Concept

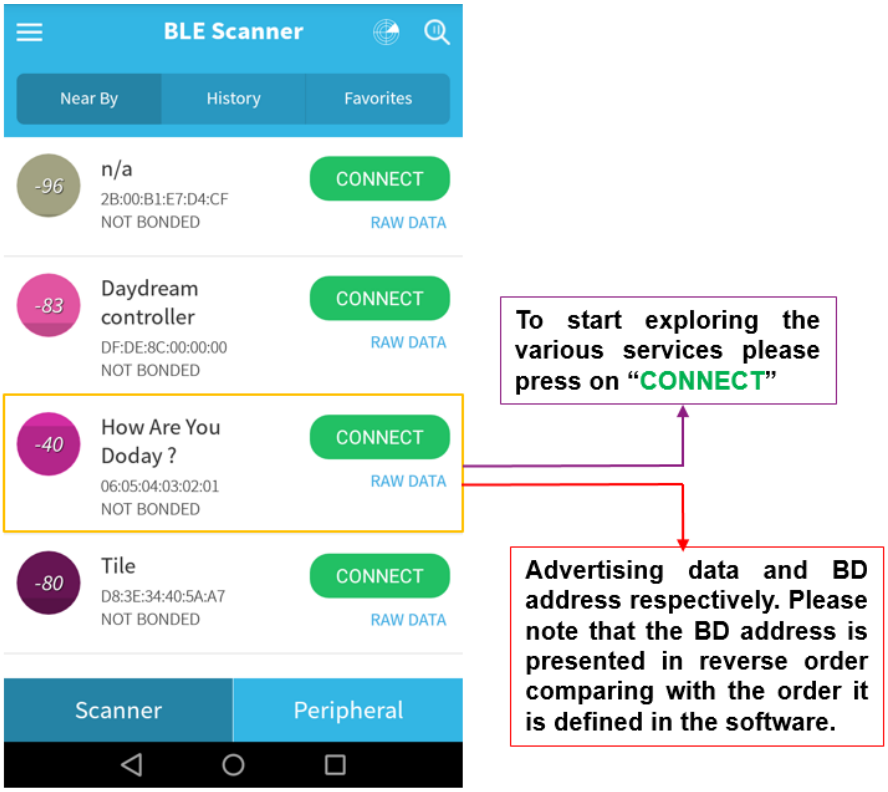

Figure 7: Verifying the Bluetooth Low Energy Device Output Using a Scanner App

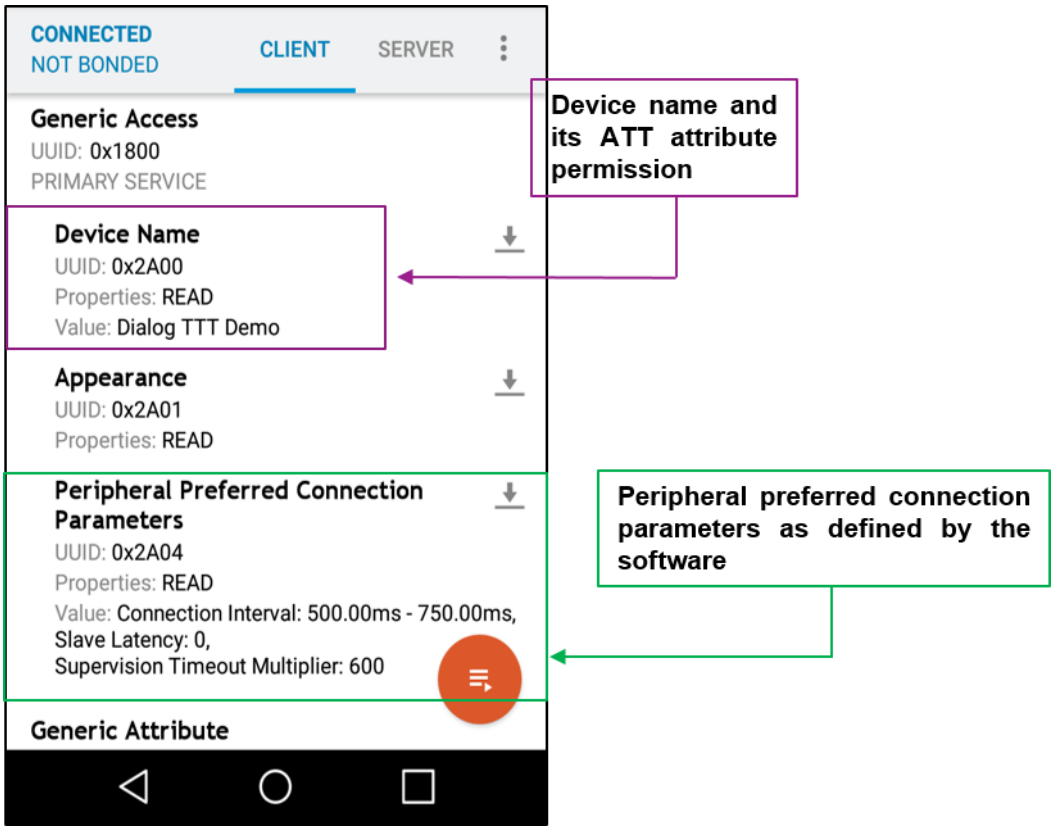

Figure 8: *Exploring the Various Services after Connecting to a Remote Peer*

5.2 Initializing the SmartSnippets Toolbox

Another useful tool that can be used both for debugging and measurement purposes is the SmartSnippets Toolbox bundled with the SmartSnippets Studio. To create a new session, in the SmartSnippets Welcome page, click on **SmartSnippets Toolbox** section. In the pop-up window displayed, do the following (with the number order):

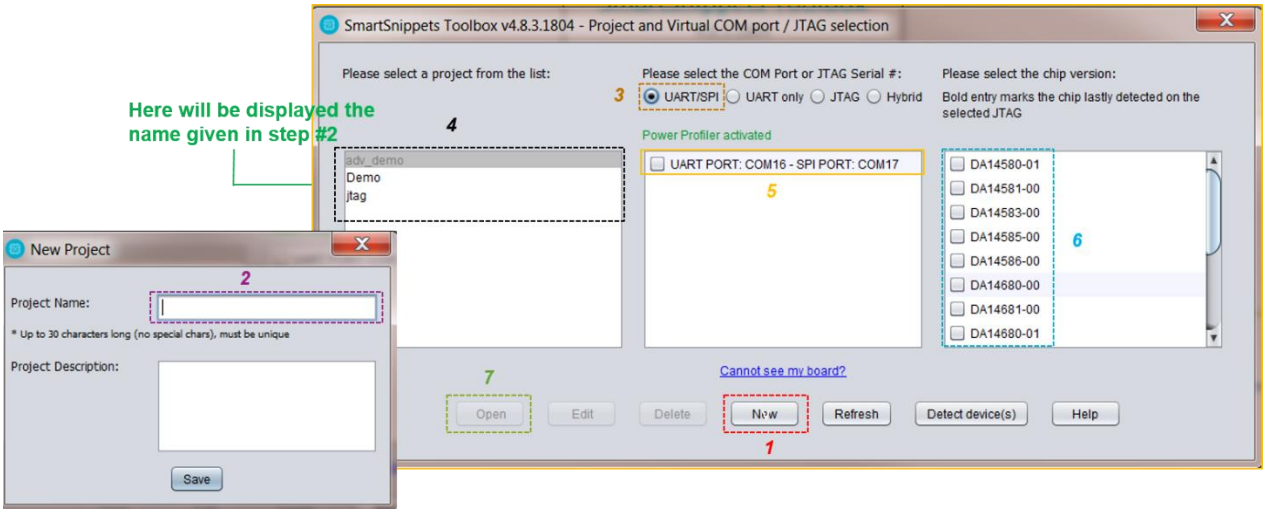

Figure 9: *Initializing the SmartSnippets Toolbox*

5.2.1 Using the SmartSnippets Toolbox

Step #1 Using the power profiler, check that the advertising interval has been set to power-save mode as selected. In this case, the Bluetooth low energy module uses the *max* value of this mode. (*1500 ms*). The alternative option would be the *min* value (*1000 ms*). You can also change the advertising interval to fast mode and verify the device behavior.

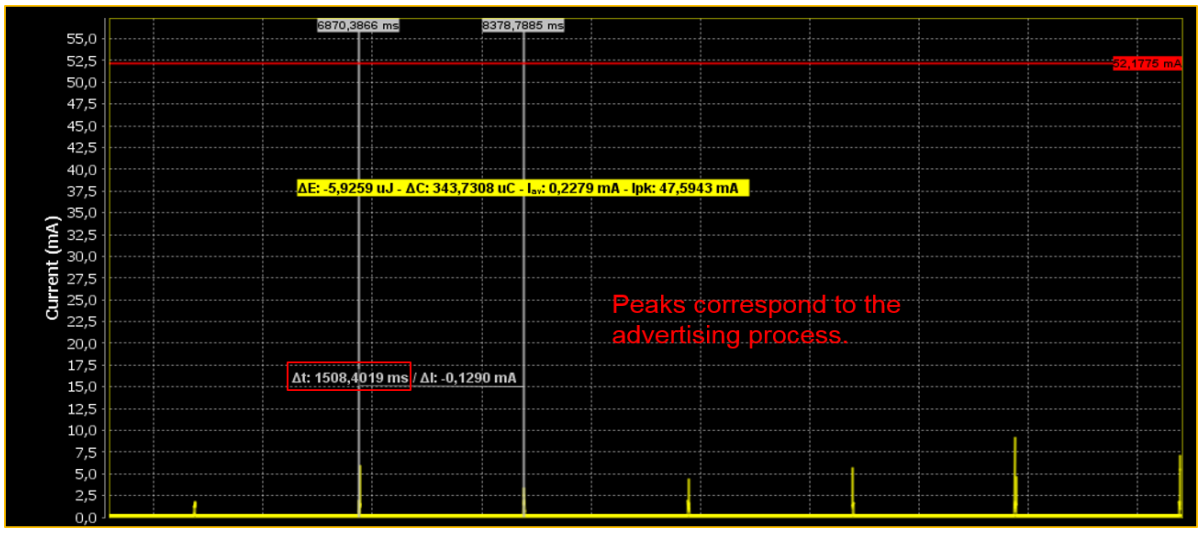

Figure 10: Verifying Advertising Interval

Step #2 Using the power profiler check that connection parameter has changed 5 seconds after a connection establishment. (Remember that the timer period was set to 5000 ms.)

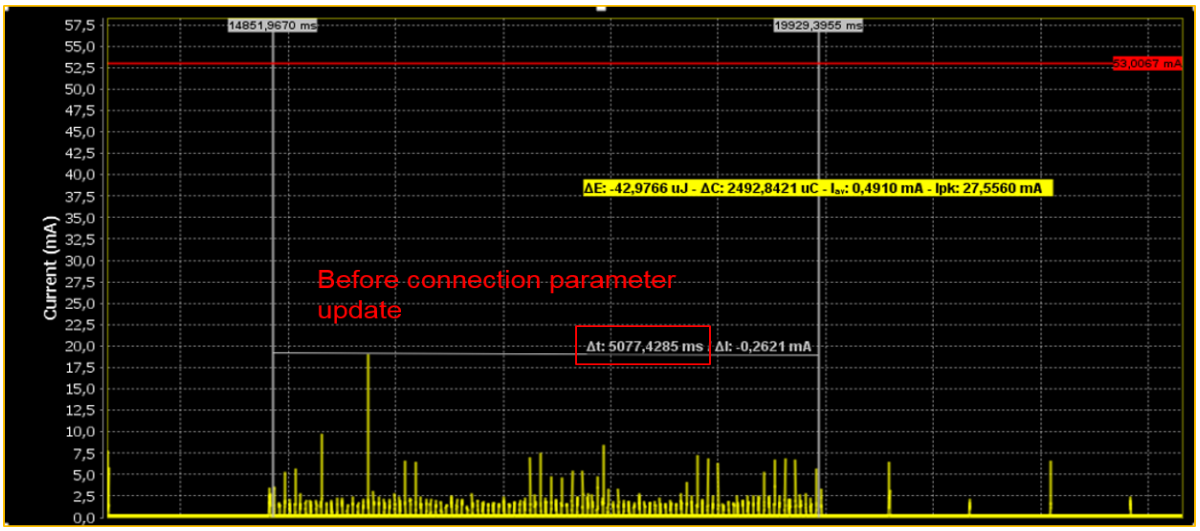

Figure 11: Verifying Connection Parameter Update

Step #3 Using the power profiler, check that connection parameter has been changed and is set to the new values. In this case, the *max* interval value is used by the Bluetooth low energy module (*750ms*). The alterative option would be the *slow* interval value (*500 ms*).

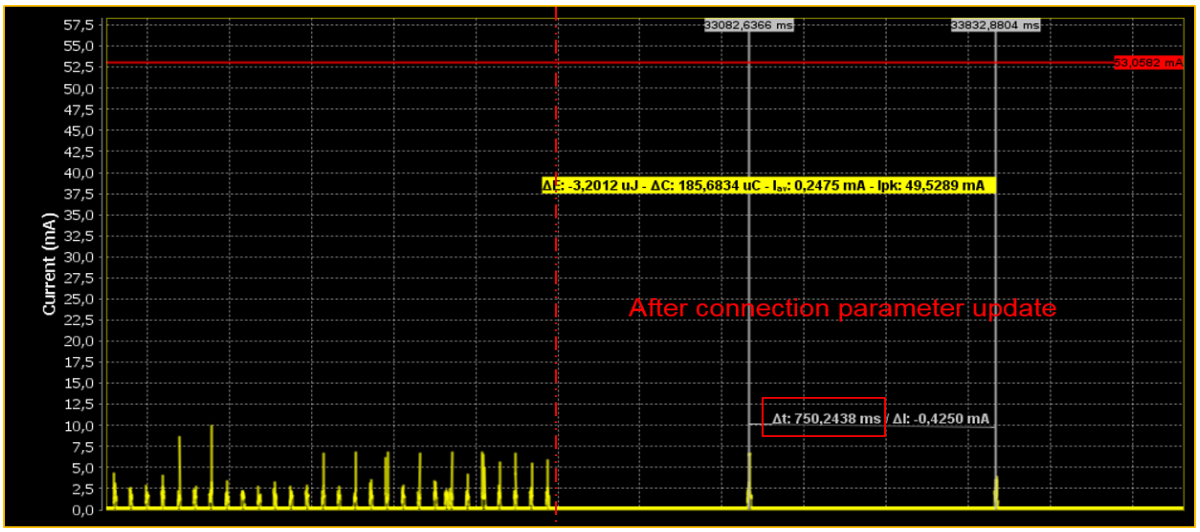

Figure 12: Verifying Connection Parameter Update

Advertising Concept

Step #4 Using the power profiler, check the default advertising channel map (channels 37, 38 and 39). Also note the advertising interval is set to high speed mode.

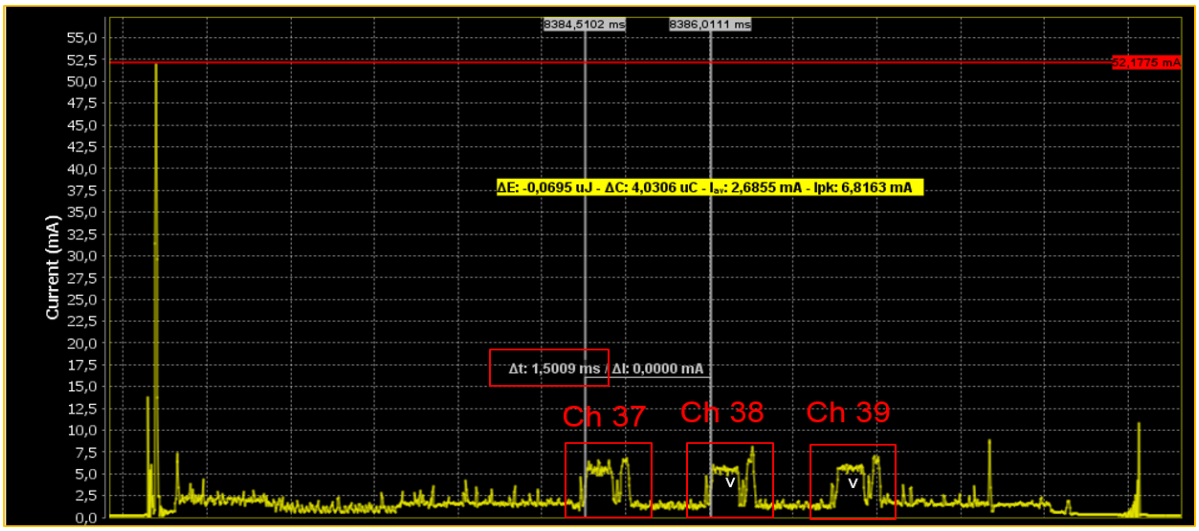

Figure 13: Verifying Advertising Channel Map and Interval

Step #5 Using the power profiler, check that the advertising channel map has been updated (only channel 37 advertises).

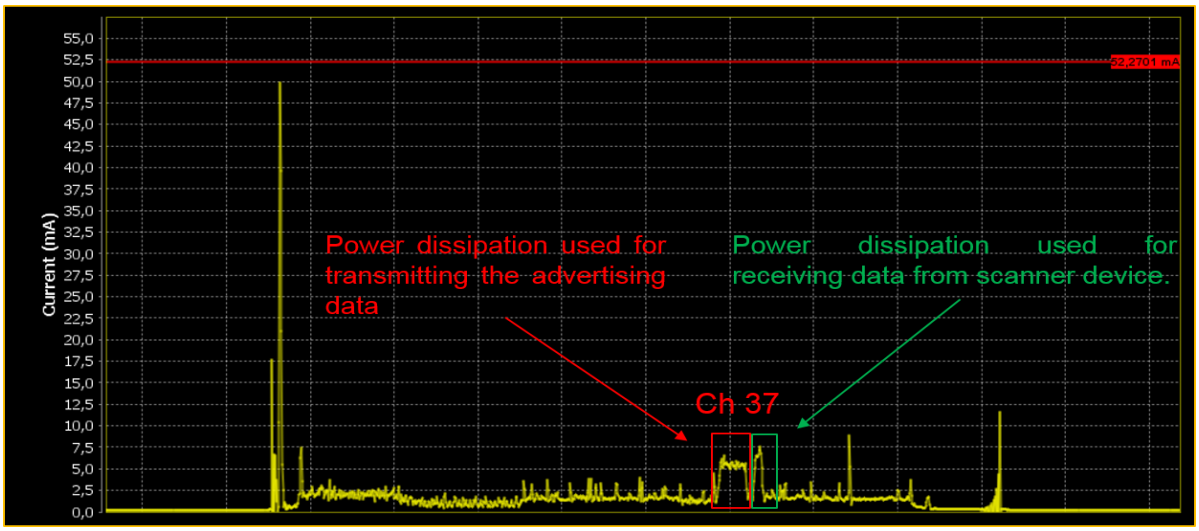

Figure 14: Verifying Advertising Channel Map

Code Overview

This section provides the code blocks needed to successfully execute this tutorial. It is recommended that you copy these blocks to the new project before reading the tutorial. Please watch the following video.

Advertising Concept

6.1 Initialization Code

Code snippet of the initialization code block:

```
/* Definitions for the advertising interval */
#define FAST_ADV_INTERVAL      (0)
#define POWER_ADV_INTERVAL  (1)


OS_TIMER update_timer;
static uint16_t connection_index;

/*
 * BLE adv demo advertising data
 */
static const uint8_t adv_data[] = {
      0x14, GAP_DATA_TYPE_LOCAL_NAME,
      'H', 'o', 'w', ' ', 'A', 'r', 'e', ' ', 'Y', 'o', 'u', ' ', 'D', 'o', 'd', 'a', 'y', ' ', '?'
};

/* Initialize the BLE structure related to BD address value. */
static const own_address_t user_bd_address = {
     .addr_type = PRIVATE_STATIC_ADDRESS,
     .addr = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06},
};


#if (FAST_ADV_INTERVAL == 1)
      static const uint16_t min = BLE_ADV_INTERVAL_FROM_MS(80); //80ms
      static const uint16_t max = BLE_ADV_INTERVAL_FROM_MS(100); //100ms
#else
      static const uint16_t min = BLE_ADV_INTERVAL_FROM_MS(1000); //1000ms
      static const uint16_t max = BLE_ADV_INTERVAL_FROM_MS(1500); //1500ms
#endif
```

6.2 Advertising Task Code

Code snippet of *ble_adv_demo_task()* function:

```
static void ble_adv_demo_task(void *pvParameters)
{
      int8_t wdog_id;

      gap_adv_chnl_t channel_map = GAP_ADV_CHANNEL_37;

      // Just remove compiler warnings about the unused parameter
      ( void ) pvParameters;

      /* register ble_adv_demo task to be monitored by watchdog */
      wdog_id = sys_watchdog_register(false);

      // Start BLE module as a peripheral device
      ble_peripheral_start();
```

Advertising Concept

```
// Set device name
ble_gap_device_name_set("Dialog TTT Demo", ATT_PERM_READ);

// Set BD address to the preferred value
ble_gap_address_set(&user_bd_address, 0x00FF);

// Set advertising interval
ble_gap_adv_intv_set(min,max);

ble_gap_adv_chnl_map_set(channel_map);

// Set advertising data
ble_gap_adv_data_set(sizeof(adv_data), adv_data, 0, NULL);

// Start advertising
ble_gap_adv_start(GAP_CONN_MODE_UNDIRECTED);

for (;;) {
    ble_evt_hdr_t *hdr;

    /* notify watchdog on each loop */
    sys_watchdog_notify(wdog_id);

    /* suspend watchdog while blocking on ble_get_event() */
    sys_watchdog_suspend(wdog_id);

    /*
     * Wait for a BLE event - this task will block
     * indefinitely until something is received.
     */
    hdr = ble_get_event(true);

    /* resume watchdog */
    sys_watchdog_notify_and_resume(wdog_id);

    if (!hdr) {
        continue;
    }

    switch (hdr->evt_code) {
    case BLE_EVT_GAP_CONNECTED:
        handle_evt_gap_connected((ble_evt_gap_connected_t *) hdr);
        break;
    case BLE_EVT_GAP_DISCONNECTED:
        handle_evt_gap_disconnected((ble_evt_gap_disconnected_t *) hdr);
        break;
    case BLE_EVT_GAP_PAIR_REQ:
        handle_evt_gap_pair_req((ble_evt_gap_pair_req_t *) hdr);
        break;
    default:
        ble_handle_event_default(hdr);
        break;
    }
```

```
            // Free event buffer
            OS_FREE(hdr);
        }
}
```

6.3 Connection Parameter Update Code

Code snippet of connection parameter update code block:

```
/* Update connection parameters. */
static void conn_param_update(uint16_t conn_idx)
{
    gap_conn_params_t cp;

    cp.interval_min = defaultBLE_PPCP_INTERVAL_MIN;
    cp.interval_max = defaultBLE_PPCP_INTERVAL_MAX;
    cp.slave_latency = defaultBLE_PPCP_SLAVE_LATENCY;
    cp.sup_timeout = defaultBLE_PPCP_SUP_TIMEOUT;

    ble_gap_conn_param_update(conn_idx, &cp);
}

/* This timer callback notifies task that time for discovery, bonding and
 * encryption has elapsed, and connection parameters can be changed to
 * the preferred values.
 */
static void conn_param_timer_cb(OS_TIMER timer)
{
      /* Call the function which is responsible for the connection parameters
       * update
       */
       conn_param_update(connection_index);
}

static void handle_evt_gap_connected(ble_evt_gap_connected_t *evt)
{
     /*
      * Manage behavior upon connection
      */
      connection_index = evt->conn_idx;

    /* Add a timer that when expired will renegotiate connection parameters. */
    update_timer = OS_TIMER_CREATE("conn_timer", OS_MS_2_TICKS(5000),  \
            OS_TIMER_FAIL , (uint32_t) OS_GET_CURRENT_TASK(), \
conn_param_timer_cb);
    OS_TIMER_START(update_timer, OS_TIMER_FOREVER);
}

static void handle_evt_gap_disconnected(ble_evt_gap_disconnected_t *evt)
{
     /*
      * Manage behavior upon disconnection
```

```
        */

        // Restart advertising
        ble_gap_adv_start(GAP_CONN_MODE_UNDIRECTED);
}
```

Note: Don't forget to delete the already existing ***handle_evt_gap_connected()*** and ***handle_evt_gap_disconnected()*** function.

6.4 Custom definitions code

Code snippet for macro definitions defined in *customer_config_qspi.h* header file:

```
/* Peripheral specific config */

#define defaultBLE_PPCP_INTERVAL_MIN   (BLE_CONN_INTERVAL_FROM_MS(500))
// 500ms

#define defaultBLE_PPCP_INTERVAL_MAX  (BLE_CONN_INTERVAL_FROM_MS(750))
// 750ms

#define defaultBLE_PPCP_SLAVE_LATENCY (0) // 0 events

#define defaultBLE_PPCP_SUP_TIMEOUT (BLE_SUPERVISION_TMO_FROM_MS(6000))
//6000ms
```

Revision History

Revision	Date	Description
1.0	30-Nov-2017	First released version

Advertising Concept

Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

Disclaimer

Contacting Dialog Semiconductor

United Kingdom (Headquarters)
Dialog Semiconductor (UK) LTD
Phone: +44 1793 757700

Germany
Dialog Semiconductor GmbH
Phone: +49 7021 805-0

The Netherlands
Dialog Semiconductor B.V.
Phone: +31 73 640 8822

Email:
enquiry@diasemi.com

North America
Dialog Semiconductor Inc.
Phone: +1 408 845 8500

Japan
Dialog Semiconductor K. K.
Phone: +81 3 5769 5100

Taiwan
Dialog Semiconductor Taiwan
Phone: +886 281 786 222

Web site:
www.dialog-semiconductor.com

Hong Kong
Dialog Semiconductor Hong Kong
Phone: +852 2607 4271

Korea
Dialog Semiconductor Korea
Phone: +82 2 3469 8200

China (Shenzhen)
Dialog Semiconductor China
Phone: +86 755 2981 3669

China (Shanghai)
Dialog Semiconductor China
Phone: +86 21 5424 9058