User Manual

DA16600 Example Application Manual

UM-WI-018

Abstract

DA16200/DA16600 is a highly integrated ultra-low-power Wi-Fi system on a chip (SoC) and allows users to develop the Wi-Fi solution on a single chip. This document is an SDK guide document intended for developers who want to program using the DA16200 chipset. And it describes the SDK API and peripheral device drivers and interfaces.

Contents

Figures

1 Terms and Definitions

AP	Access Point
API	Application Programming Interface
BD	Bluetooth Device
COM	Communication Port
CTS	Clear to Send
DHCP	Dynamic Host Configuration Protocol
DPM	Dynamic Power Management
EVB	Evaluation Board
FW	Firmware
HW	Hardware
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GPIO	General Purpose Input / Output
GTL	Generic Transport Layer
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I2C	Inter-Integrated Circuit (bus)
IDE	Integrated Development Environment
iOS	iPhone Operating System
IoT	Internet of Things
JLINK	JTAG Link
JSON	JavaScript Object Notation
JTAG	Joint Test Action Group
LED	Light emitting diode
LAN	Local Area Network
OTA	Over the Air
OTP	One Time Programmable
PC	Personal Computer
POR	Power on Reset
PWM	Pulse Width Modulation
RBP	Raspberry Pi
RF	Radio Frequency
RTOS	Real Time Operating System
RTS	Request to Send
SDK	Software Development Kit
SoC	System on Chip
SPI	Serial Peripheral Interface
SVC	Service
SW	Software
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
USB	Universal Serial Bus

UUID	Universally Unique Identifier
Wi-Fi	Wireless Fidelity
XIP	Execute in Place

2 References

[1] UM-WI-002, DA16200 SDK Programmer Guide

[2] UM-B-143, Dialog External Processor Interface

[3] UM-B-117, DA14531 Getting Started Guide

[4] UM-B-119, DA14531 SW Platform Reference

[5] UM-WI-044, DA16600 Provisioning the Mobile App for Android/iOS

[6] UM-WI-035, DA16200 SDK Memory Map

3 Introduction

The Dialog DA16600 module is also known as "Combo". The terms "Combo" and "DA16600" may be used interchangeably in this document. The Dialog DA16600 module is comprised of the DA16200 (Wi-Fi) and DA14531 (Bluetooth® LE) SoCs. This document describes the test steps and code walkthroughs of several Combo example applications.

3.1 DA16600 Module Evaluation Board

Figure 1 shows the hardware configuration of the DA16600 EVB.

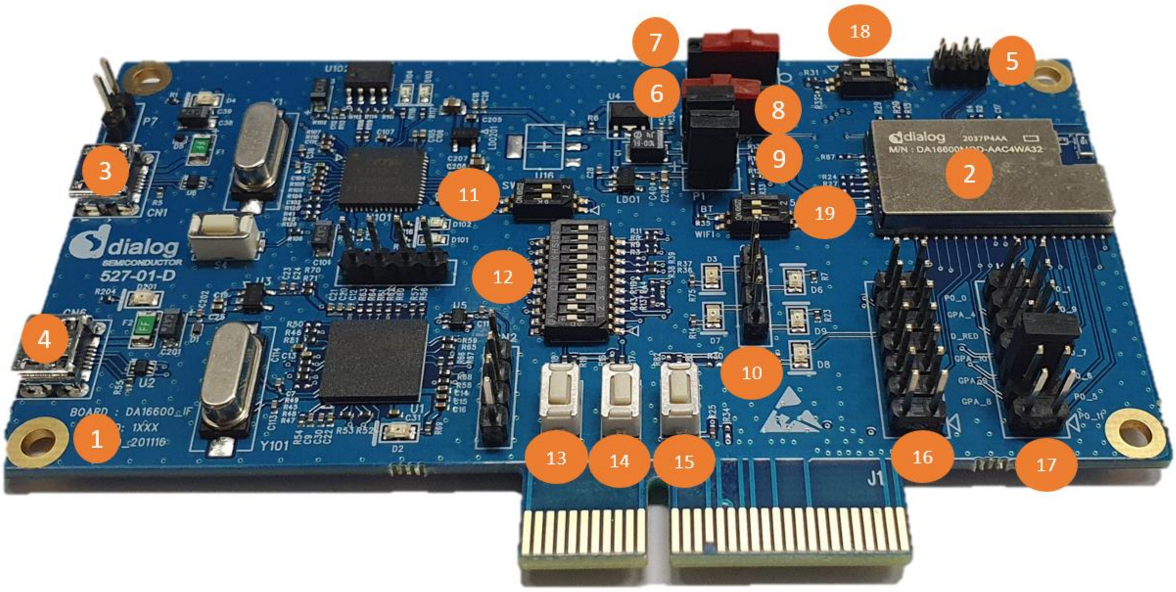
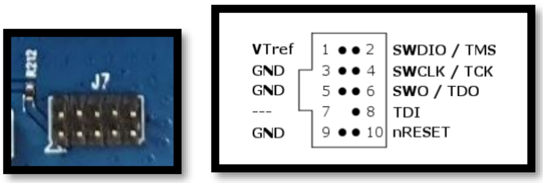


Figure 1: DA16600 EVB

Refer to the following description for the components on the DA16600 EVB components:

Table 1: Components Description on the DA16600 EVB

ID	Name	Description
1	**Main board**	DA16600 module EVB (DA16600MOD-AAC)
2	**DA16600MOD-AAC**	Dialogs **Wi-Fi and Bluetooth® LE Module**
3	**USB Port (WiFi)**	Provides UART0 for debug and UART1 for test
4	**USB Port (BLE)**	Connect directly to DA14531 for debug only Note: Do not use this port during normal operation
5	**JTAG Connector**	Connector for IARs I-jet JTAG Debugger

ID	Name	Description
		 J7 VTref 1 2 SWDIO / TMS GND 3 4 SWCLK / TCK GND 5 6 SWO / TDO --- 7 8 TDI GND 9 10 nRESET Note: Pin 7 on the I-Jet debugger cable is keyed with a white plug so pin 7 must be removed on EVB
6	**RTC Wake up2 key**	Switch to wake up the board from Sleep mode
7	**RTC Power key**	Switch to turn the EVB on and off
8	**Jumper (P2)**	Jumper to measure the current used by the Wi-Fi device For normal operation, this jumper must be shorted
9	**Jumper (P1)**	Jumper to measure the current used by the Bluetooth® LE device For normal operation, this pin must be shorted
10	**Connector CN4**	GPIO test purpose connector To test GPIO in J2 and J14, connect to LED by this connector
11	**Switch SW3**	Switch to connect directly to DA14531 and use UART to check Bluetooth® LE performance Set this switch off for normal operation
12	**Switch SW7**	Multipurpose switch Set this switch off for normal operation
13	**Switch S2**	Factory reset button using GPIOA7 To enable this button, set Pin2 of SW7 to on
14	**Switch S1**	WPS button using GPIOA6 To enable this button, set Pin1 of SW7 to on
15	**Switch S3**	The reset button of DA14531 in test mode
16	**Connector J2**	GPIO connector
17	**Connector J14**	GPIO connector
18	**Switch SW4**	Switch to control RF switch in DA16600MOD at test mode
19	**Switch SW5**	Switch to check the current consumption using a power meter kit

3.2 Example 1: Bluetooth® LE Assisted Wi-Fi Provisioning

This example demonstrates how the DA16600 can be used in a product such as a "Wi-Fi door lock" where the Wi-Fi is the main communication interface to the server and the Bluetooth® LE interface only assists with the "Wi-Fi Provisioning" during the product's initial "Out-of-Box" setup.

NOTE
This example requires a Bluetooth® LE peer provisioning application described in Ref. [5], it is to configure the DA16600 Wi-Fi parameters including the Wi-Fi home router's SSID and password and the server's

NOTE
connection information. See Section 6.4.1.5 and Ref. [5] for details on the provisioning protocol required for the mobile app.

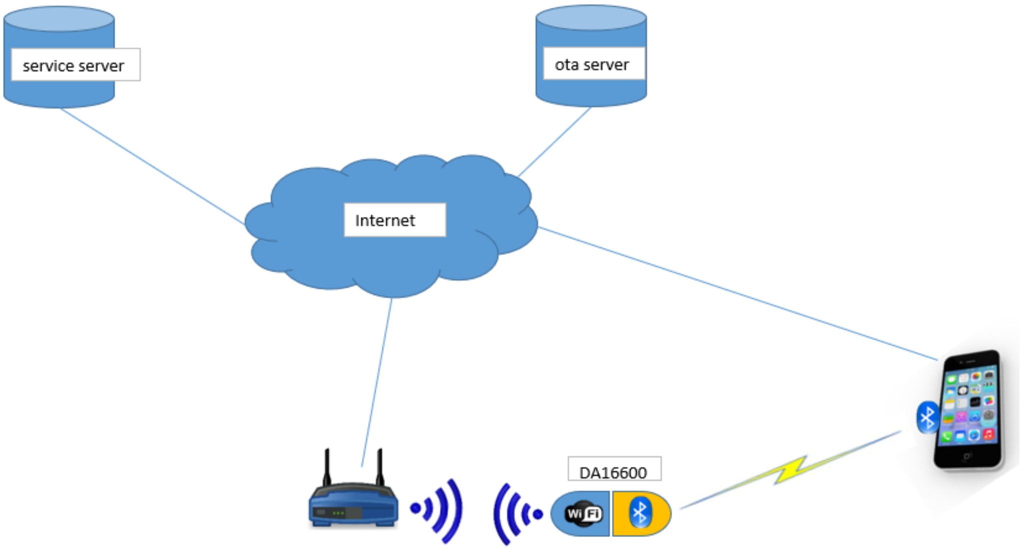


Figure 2: Bluetooth® LE Assisted Wi-Fi Provisioning

3.3 Example 2: Bluetooth® LE Firmware OTA Download via Wi-Fi

Once the Wi-Fi has been successfully provisioned and is up and running as described in Example 1, the DA16600 device can receive notifications over Wi-Fi from a remote service server indicating that there is new Wi-Fi or Bluetooth® LE firmware available for the DA16600. Upon receipt of the notification, the DA16600 can securely download the new firmware from an OTA server via Wi-Fi, store the firmware in its flash memory and then trigger the firmware update.

3.4 Example 3: Gas Leak Detection Sensor

This example demonstrates how a DA16600 can wirelessly interact with a standalone Gas Leak Detection Sensor via Bluetooth® LE and communicate events to a server over Wi-Fi.

A virtual gas density check sensor is attached to the Bluetooth® LE wireless interface of the DA16600. The DA16600 is operating in low-power mode and periodically checks the gas density level at a time interval defined by the user. When a certain gas density level value is reached, the Wi-Fi device is activated and a "gas leak" event is created and sent to the Wi-Fi device. The Wi-Fi device then posts the "gas leak" event to a cloud server where a user is notified, and action can be taken.

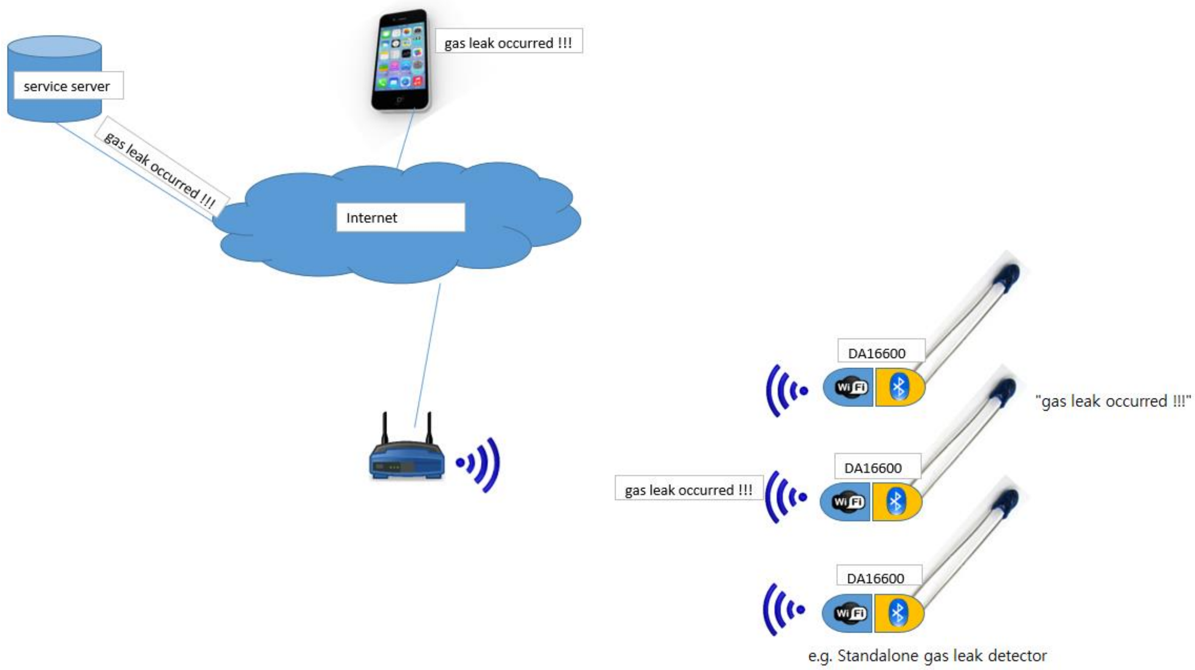


Figure 3: Standalone Gas Leak Detection Sensor

3.5 Example 4: DA14531 Peripheral Driver Examples

This example shows how the DA16200 uses peripheral devices which are connected to the DA14531. An application running in the DA16200 can configure and run peripheral driver functions through GTL.

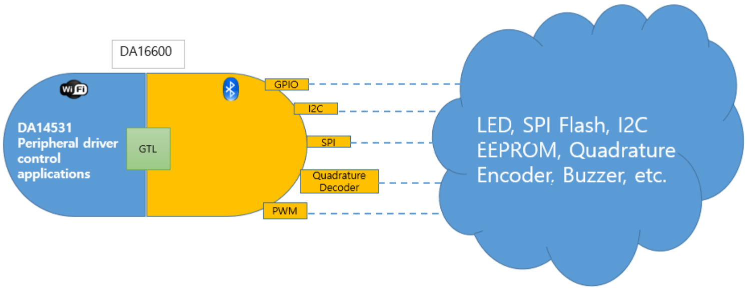


Figure 4: DA14531 Peripheral Device Control

3.6 Example 5: TCP DPM Client

This example demonstrates how the DA16600 module runs a TCP client in a low-power mode where the DA16200 stays in DPM mode and the DA14531 stays in Extended sleep mode.

A key feature of the DA16600 module is that while in sleep mode it can receive and process a Wi-Fi packet from a network peer or Bluetooth® LE data from a Bluetooth® LE peer. After either a Wi-Fi packet or Bluetooth® LE data has been handled, the DA16600 enters sleep mode again to save power.

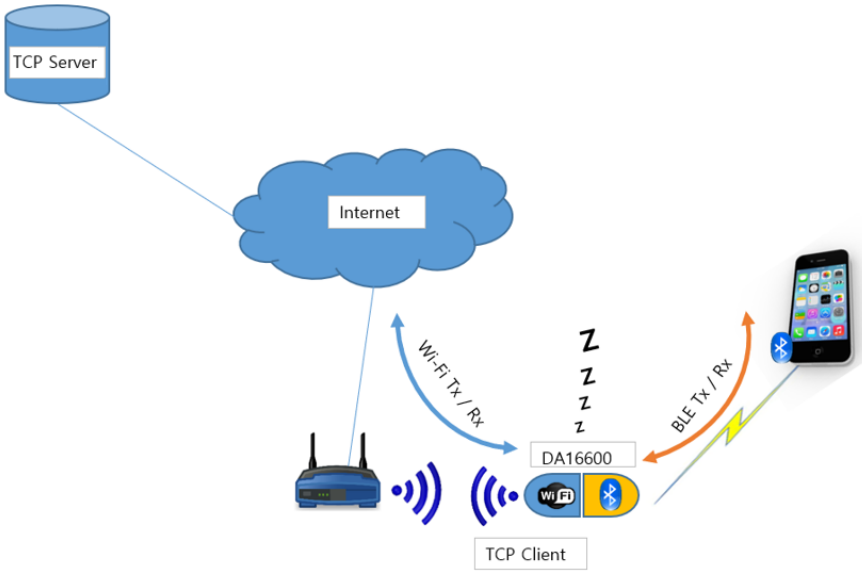


Figure 5: DA16600 DPM Communication

3.7 Example 6: IoT Sensor Gateway

In this example, the DA16600 plays the role of a gateway device for multiple Bluetooth® LE temperature sensors. A Bluetooth® LE sensor posts the current temperature value periodically via the notify function of Bluetooth® LE. The Bluetooth® LE chip of the DA16600 gathers the information and asks Wi-Fi to (periodically) post notifications to a service server in the cloud.

NOTE
DA14531 can maintain a maximum of three connections.

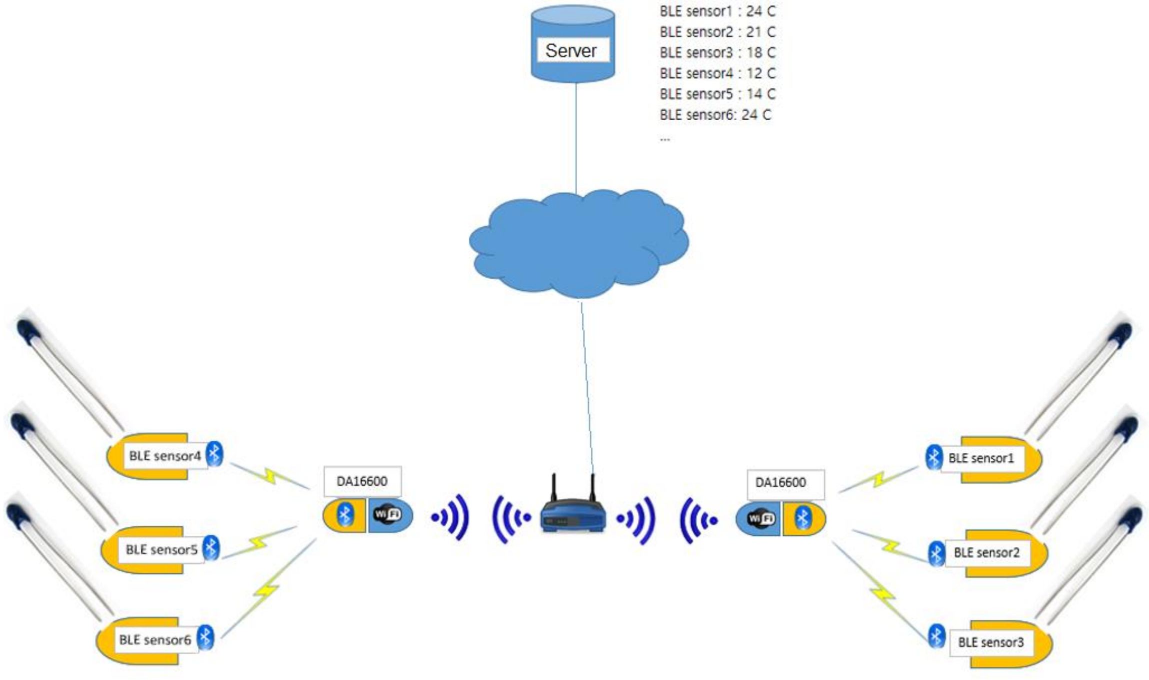


Figure 6: IoT Sensor Gateway

4 Software Build and Flashing Guide

The DA16600 module includes two SoC chips (DA16200 and DA14531). The firmware images of each SoC are stored in the SPI flash (SFLASH onwards) of the DA16600 module. The SFLASH is only accessible by DA16200. There is no separate flash memory installed that DA14531 can access directly. This means that a firmware image for DA14531 should be written in the SFLASH of the DA16200 and transferred to DA14531 at runtime by DA16200.

The list of firmware images needed to run each SoC chip:

DA16200: Wi-Fi chip

- Three image files:
  - **BOOT** image: secondary bootloader
  - **SLIB** image: RF RAM library and TIM image (which is for the low-power operation of DA16200)
  - **RTOS** image: main operation software into which user applications are built
- Code storage memory
  - SFLASH of DA16200

DA14531: Bluetooth® LE chip

- Single image file:
  - Main image: main operation software into which user applications are built
- Code storage memory
  - SFLASH of DA16200 or OTP memory (32 kB allocated for OTP image)
    - OTP memory can be used to burn a default image. If an OTA firmware update is needed, then the SFLASH of the DA16200 should be used

4.1 SFLASH Memory Map

The SFLASH memory map is described in Section 3.2 of Ref. [6]. And here are some more details.

```
[DA16600_SDK_ROOT]\core\hal\inc\da16200_map.h

/* User Area #1 */
#define SFLASH_USER_AREA_1_START          0x00390000
#define SFLASH_USER_AREA_1_END            0x003FD600

/* User Firmware */ ← DA14531 firmware is stored
#define SFLASH_USER_FW_1    SFLASH_USER_AREA_1_START        /* 0x390000 - 448kbyte */
#define SFLASH_BLE_FW_START  (SFLASH_USER_AREA_1_START+0x2000)  /* 0x00392000 64KB */
```

The map has two image banks allocated for each type of DA16200 image (RTOS and SLIB), and for the DA14531 image. Two slots are used for an OTA operation taking turns.

There are two user areas: one is 364 kB, the other is 448 kB.

The next sections describe how to build software for the DA16200 and the DA14531.

4.2 Build the DA16200 Software

In the DA16600 SDK, use the IAR IDE to open the IAR workspace file (`~SDK/apps/da16600/get_started/DA16xxx.eww`).

For information on how to install and use the IAR IDE on your PC, see Section 2.2 in Ref. [1]. You can configure the software before you start to build. Open the file

"~SDK/apps/da16600/get_started/inc/ble_combo_features.h" to configure an example application.

To test **Example 1**, **2**, or **3**, make the following changes in **ble_combo_features.h** to build "DA16200_SW_1". If security needs to be enabled, then enable _WIFI_SVC_SECURITY__ as well ("A16200_SW_1b"):

- DA16200_SW_1
  - Enable **__COMBO_SAMPLE_BLE_PERI_WIFI_SVC__**
  - Disable the other sample defines (__COMBO_SAMPLE_XXX)
- DA16200_SW_1**b**
  - Enable **__COMBO_SAMPLE_BLE_PERI_WIFI_SVC__**
  - Enable **__WIFI_SVC_SECURITY__**
  - Disable the other sample defines (__COMBO_SAMPLE_XXX)

NOTE
The difference between DA16200_SW_1b and DA16200_SW_1 is when a Bluetooth® LE peer mobile application tries to connect to the DA16600 (with DA16200_SW_1b), a pairing procedure is requested. Two pairing modes are depending on your mobile phone's Bluetooth authentication capability configuration: ● **Legacy Pairing**: the mobile application may show an input box and ask you to enter a passkey that can be found on the display of the DA16600 HW, and then you need to enter the exact passkey on your smartphone to successfully connect to the DA16600 ● **Secure Connection Pairing** (SC Pairing): the mobile application may show a PIN code on your mobile and ask you to compare the pin code with the one printed on the display of the DA16600 HW. If the pin codes match, click the **OK** button to connect the DA16600 to your mobile application The DA16600 example currently can save bond information for up to ten Bluetooth® LE peers. Once bond information is put in your mobile phone, your mobile phone's Bluetooth® LE peer application can connect to the DA16600 without the need to repeat the pairing process. If either party lost the pairing credential, the pairing process starts again when you reconnect.

For **Example 4**, make the following changes to build **DA16200_SW_2**:

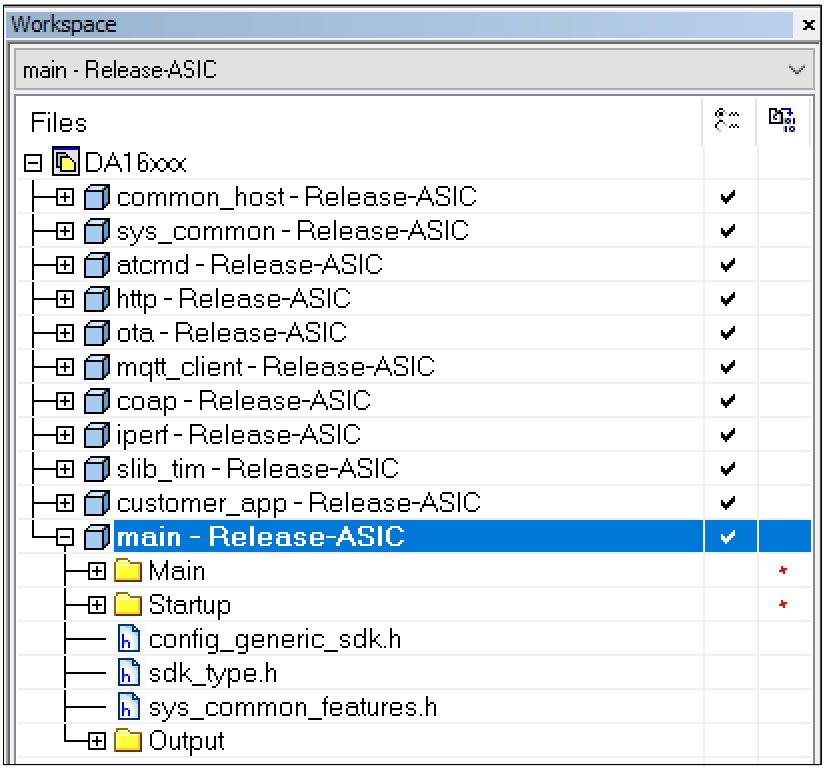
- Enable __COMBO_SAMPLE_BLE_PERI_WIFI_SVC_PERIPHERAL_SAMPLE__
- Disable the other sample defines (__COMBO_SAMPLE_XXX)

For **Example 5**, make the following changes to build **DA16200_SW_3**:

- Enable __COMBO_SAMPLE_BLE_PERI_WIFI_SVC_TCP_DPM_SAMPLE__
- Disable the other sample defines (__COMBO_SAMPLE_XXX)

For **Example 6**, make the following changes to build DA16200_SW_4:

- Enable __COMBO_SAMPLE_BLE_CENT_SENSOR_GW__
- Disable the other sample defines (__COMBO_SAMPLE_XXX)



```
Workspace
main - Release-ASIC
Files
DA16xxx
├ common_host - Release-ASIC        ✓
├ sys_common - Release-ASIC         ✓
├ atcmd - Release-ASIC              ✓
├ http - Release-ASIC               ✓
├ ota - Release-ASIC                ✓
├ mqtt_client - Release-ASIC        ✓
├ coap - Release-ASIC               ✓
├ iperf - Release-ASIC              ✓
├ slib_tim - Release-ASIC           ✓
├ customer_app - Release-ASIC       ✓
└ main - Release-ASIC               ✓
    ├ Main                              *
    ├ Startup                           *
    ├ config_generic_sdk.h
    ├ sdk_type.h
    ├ sys_common_features.h
    └ Output
```

Figure 7: Project View

To build the software for the first time, select all projects, right-click, and then select **Rebuild All**. For the second and next builds, rebuild only projects that were changed. For example, if the "main" project was changed, then rebuild "main". If you work with other examples, you should rebuild both "customer_app" and "main" projects.

After the build, the folder "`[DA16600_SDK_ROOT]\img`" should have the following three DA16200 images:

- DA16200_**SLIB**-GEN01-01-XXXXX-000000.img
- DA16200_**RTOS**-GEN01-01-XXXXX-000000.img
- DA16200_**BOOT**-GEN01-01-XXXXX-000000_W25Q32JW.img
  or DA16200_**BOOT**-GEN01-01-XXXXX-000000_AT25SL321.img

4.3 Build DA14531 Software

The DA14531 software (Bluetooth® LE SW) used in the DA16600 SDK is based on the DA14531 SDK version 6.0.14.1114. To build the DA14531 software for the examples, you need a DA14531 SDK 6.0.14.1114 that is specifically adapted for DA16600. This SDK is available in `~SDK\util\da14531_sdk\*.zip`.

Depending on which example you want to test, you need to use a different example Bluetooth® LE target project.

NOTE
The example projects mentioned below are not included in the original 6.0.14.1114. The projects below (Prj1_proxr and Prj2_proxm) are created and modified for the four examples of DA16600.

For **Examples 1**, **2**, and **3**, use the **Prj1_proxr** project:

- `[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex\Keil_5\prox_reporter_ext.uvprojx`

For **Example 4**, use the **Prj2_proxm** project:

- `[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_monitor_aux_ext_coex\Keil_5\prox_monitor_ext.uvprojx`

4.3.1 Install Keil

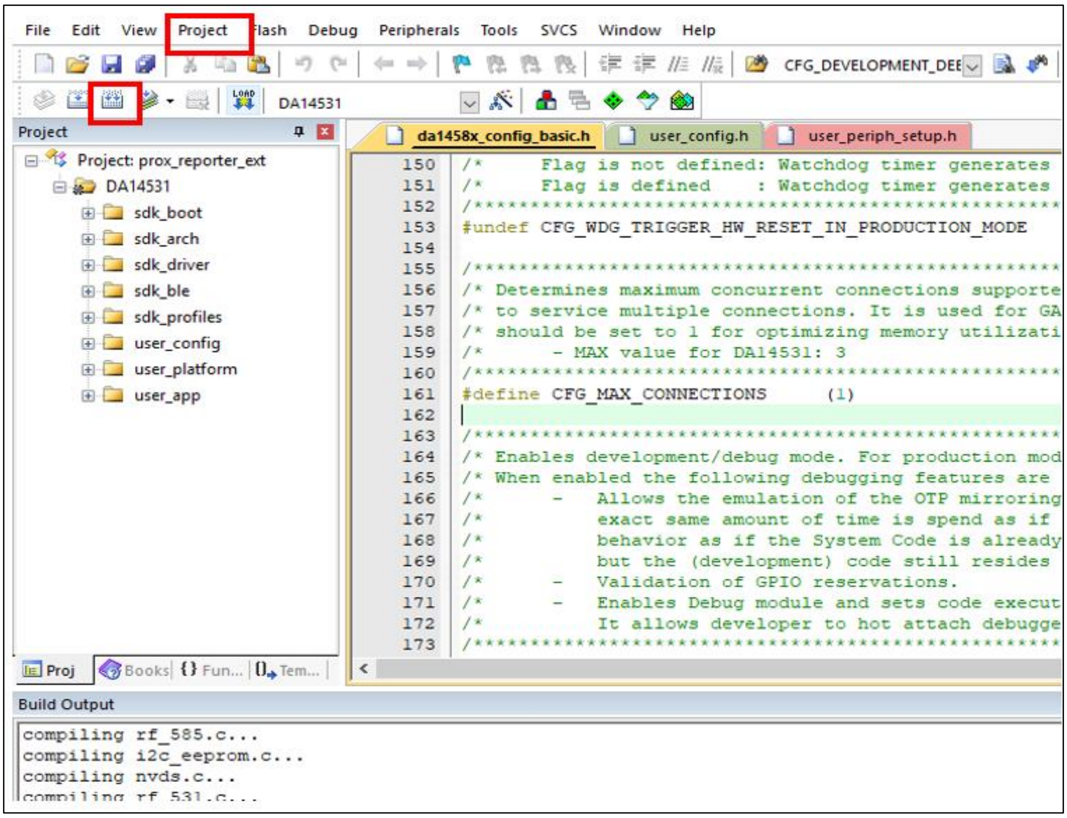
For information on the Keil installation, see the corresponding section in http://lpccs-docs.dialog-semiconductor.com/UM-B-117-DA14531-Getting-Started-With-The-Pro-Development-Kit/05_Software_Development_Tools/Software_Development_Tools.html.

NOTE
The Keil IDE download URL is https://www.keil.com/download/product/.

4.3.2 Build a Project

To build a project in Keil:

1. In Keil, go to **Project** > **Open Project**, and then select the "**.uvprojx"** file.
   Keil opens the project. For example, open `[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex\Keil_5\prox_reporter_ext.uvprojx`.
2. Go to **Project** > **Clear Targets**.
3. Click **Rebuild**.



```
/*      Flag is not defined: Watchdog timer generates
/*      Flag is defined     : Watchdog timer generates
/**************************************************
#undef CFG_WDG_TRIGGER_HW_RESET_IN_PRODUCTION_MODE

/**************************************************
/* Determines maximum concurrent connections supporte
/* to service multiple connections. It is used for GA
/* should be set to 1 for optimizing memory utilizati
/*       - MAX value for DA14531: 3
/**************************************************
#define CFG_MAX_CONNECTIONS      (1)

/**************************************************
/* Enables development/debug mode. For production mod
/* When enabled the following debugging features are
/*       -   Allows the emulation of the OTP mirroring
/*           exact same amount of time is spend as if
/*           behavior as if the System Code is already
/*           but the (development) code still resides
/*       -   Validation of GPIO reservations.
/*       -   Enables Debug module and sets code execut
/*           It allows developer to hot attach debugge
/**************************************************
```

```
compiling rf_585.c...
compiling i2c_eeprom.c...
compiling nvds.c...
compiling rf_531.c...
```

Figure 8: Keil – Build

NOTE

To quickly test an example without building the DA14531 software, use the pre-built DA14531 image. See the folder "`[DA16600_SDK_ROOT]\img\DA14531_...\`":

- DA14531_1 ← **Prj1_proxr**
- DA14531_2 ← **Prj2_proxm**

If you want to run multiple DA16600 boards at the same time (with the same example project), you need to build the DA14531 projects with different BD addresses. Ensure that each DA16600 board's BD address is unique to avoid an address conflict. You can change the BD address of the DA14531 in the **da1458x_config_advanced.h** file (search for **CFG_NVDS_TAG_BD_ADDRESS** at the bottom of the source).

After **Prj1_proxr** or **Prj2_proxm** is built with the Keil IDE, there is a **.bin** file in directory "`[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex\Keil_5\out_DA14531\Objects\`".

4.4 Program SFLASH and Run

4.4.1 Create a DA14531 Image from .bin

After code build, the images can be found in the following folder:

`[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex\Keil_5\out_img\`.

or

`[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_monitor_aux_ext_coex\Keil_5\out_img\`.

The images required in the examples are generated automatically by create_img.bat in the Keil project folder above. But to create images manually, you can use the tool named **mkimage.exe** available in the "`~SDK\util\ble_img_creator\`" folder of the DA16600 SDK.

NOTE

The mkimage.exe included in the DA16600 SDK is built in Visual Studio 2019 release mode. Therefore, if you run mkimage.exe from the command prompt window and get some errors like "*… vcruntime140.dll is missing …*", then install Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019 (see https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads).

If your PC still does not run mkimage.exe, you can also build the tool. The tool mkimage.exe can be built in either GCC or MSVC environments.

The mkimage tool source folder is: `[DA14531_SDK_ROOT]\utilities\mkimage\`.

For **GCC**: for example, Linux, Cygwin, and so on.

`[DA14531_SDK_ROOT]\utilities\mkimage\gcc` : run "make" at the prompt.

For **Windows**: for example, Visual Studio

`[DA14531_SDK_ROOT]\utilities\mkimage\` : add mkimage.c to a win32 console application project and build.

4.4.1.1 Use mkimage.exe

There are two types of .img files to be created with mkimage.exe:

- **multi-part format**: used in a normal development phase and in production

```
ble_multi_part_header_t     + // multi-part header
ble_img_hdr_t               + // bank1: fw_1 header
ble fw1 bin                 + //            fw_1 bin
ble_img_hdr_t               + // bank2: fw_2 header
```

ble fw2 bin + // fw_2 bin
ble_product_header_t // product header: bank1/bank2 offset

- **single-part format**: used for an OTA firmware update. A new version of a firmware image should be created in a single-part format and published on an OTA Server

  ble_multi_part_header_t +
  ble_img_hdr_t +
  ble fw bin (new version)

The SFLASH programming procedure is explained in the next section.

4.4.1.2 Create a Multi-Part Image

In Section 4.4.1, the multi-part image is generated already but it can be created in two ways using:

- Option 1 – the Python script included in SDK
- Option 2 – raw commands in command prompt

Option 1

If you have Python 3.6.x installed on your PC, you can create an image following these steps:

1. Make a temporary folder (for example, C:\temp). Locate and copy the following files from "`~SDK\util\ble_img_creator\`" to the temporary folder:
   - mkimage.exe
   - app_version.h
   - mkimage_multi.py
2. Copy the DA14531 bin file to the temporary folder.
3. Open **app_version.h**, and change the version to your version, and then save it.
   For example: change #define SDK_VERSION "6.0.14.1114.**1**" to #define SDK_VERSION "6.0.14.1114.**2**"
4. In Windows Explorer, double-click **mkimage_multi.py**, and then follow the instruction.
   a. If you want to give your name to .img, type the name in the command prompt window, then press **Enter**.

The following image is created: **da14531_multi_part_proxr.img**.

NOTE

- The number of characters in the version string (in app_version.h) is less than or equal to 15
- If you build DA14531 SDK, synchronize the version string in "`[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex\include\app_version.h`" with "`[DA16600_SDK_ROOT]\util\ble_img_creator\ app_version.h`"

Option 2

To create a multi-part image in a command prompt:

1. Create a folder (for example, C:\temp_folder) and copy the following files in that folder:
   - `~SDK\util\ble_img_creator\mkimage.exe`
   - `~SDK\util\ble_img_creator\app_version.h`
   - `~SDK\apps\da16600\get_started\img\DA14531_1\pxr_sr_coex_ext_531_6_0_14_1114.bin`

   In this demo, a "Prj1_proxr" image is created.

NOTE

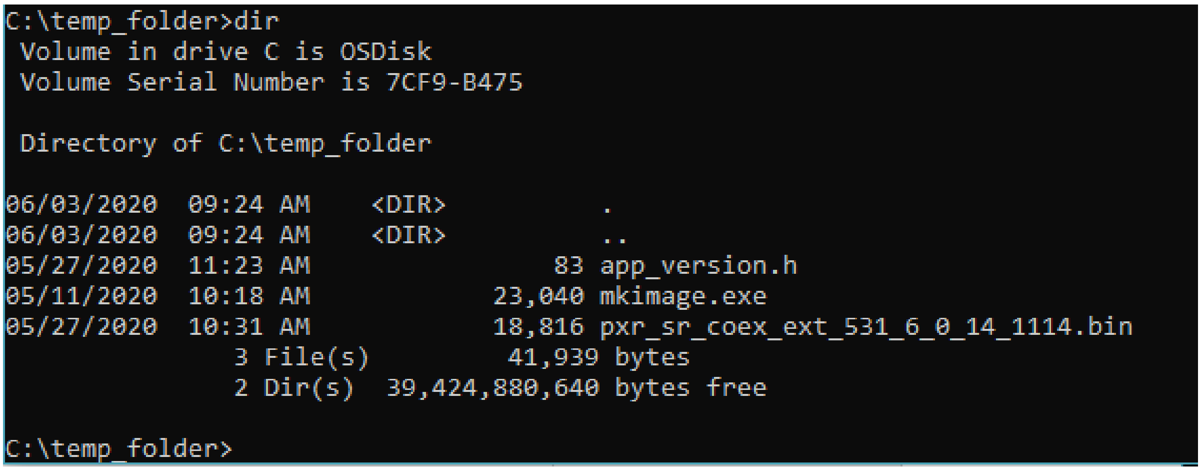
The version string in "~SDK\util\ble_img_creator\ app_version.h" should be the same as that of "[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex\include\app_version.h".

Instead of using the pre-compiled .bin ("~SDK\apps\da16600\get_started\img\DA14531_1\pxr_sr_coex_ext_531_6_0_14_1114.bin"), you can use one built in Keil.

2. Open the **app_version.h** file and change the version as follows:

   ```
   app_version.h
   #define SDK_VERSION "6.0.14.1114.1"
   …
   ```

   This version is written to **ble_img_hdr_t** of a multi-part image.

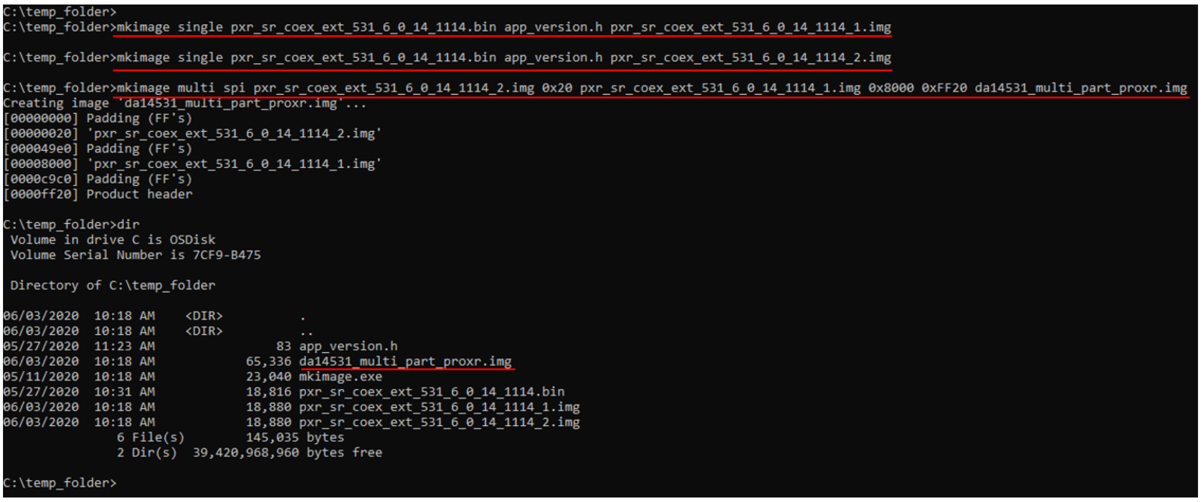


```
C:\temp_folder>dir
 Volume in drive C is OSDisk
 Volume Serial Number is 7CF9-B475

 Directory of C:\temp_folder

06/03/2020  09:24 AM    <DIR>          .
06/03/2020  09:24 AM    <DIR>          ..
05/27/2020  11:23 AM                83 app_version.h
05/11/2020  10:18 AM            23,040 mkimage.exe
05/27/2020  10:31 AM            18,816 pxr_sr_coex_ext_531_6_0_14_1114.bin
               3 File(s)         41,939 bytes
               2 Dir(s)  39,424,880,640 bytes free

C:\temp_folder>
```

Figure 9: Bluetooth® LE FW Image Generation – Preparation

3. At the command prompt, type the following commands in the following order:
   a. `mkimage single pxr_sr_coex_ext_531_6_0_14_1114.bin app_version.h pxr_sr_coex_ext_531_6_0_14_1114_1.img`.
   b. `mkimage single pxr_sr_coex_ext_531_6_0_14_1114.bin app_version.h pxr_sr_coex_ext_531_6_0_14_1114_2.img`.
   c. `mkimage multi spi pxr_sr_coex_ext_531_6_0_14_1114_2.img 0x20 pxr_sr_coex_ext_531_6_0_14_1114_1.img 0x8000 0xFF20 da14531_multi_part_proxr.img`.

Figure 10 shows how to create a demo multi-part image.



```
C:\temp_folder>
C:\temp_folder>mkimage single pxr_sr_coex_ext_531_6_0_14_1114.bin app_version.h pxr_sr_coex_ext_531_6_0_14_1114_1.img

C:\temp_folder>mkimage single pxr_sr_coex_ext_531_6_0_14_1114.bin app_version.h pxr_sr_coex_ext_531_6_0_14_1114_2.img

C:\temp_folder>mkimage multi spi pxr_sr_coex_ext_531_6_0_14_1114_2.img 0x20 pxr_sr_coex_ext_531_6_0_14_1114_1.img 0x8000 0xFF20 da14531_multi_part_proxr.img
Creating image 'da14531_multi_part_proxr.img'...
[00000000] Padding (FF's)
[00000020] 'pxr_sr_coex_ext_531_6_0_14_1114_2.img'
[000049e0] Padding (FF's)
[00008000] 'pxr_sr_coex_ext_531_6_0_14_1114_1.img'
[0000c9c0] Padding (FF's)
[0000ff20] Product header

C:\temp_folder>dir
 Volume in drive C is OSDisk
 Volume Serial Number is 7CF9-B475

 Directory of C:\temp_folder

06/03/2020  10:18 AM    <DIR>          .
06/03/2020  10:18 AM    <DIR>          ..
05/27/2020  11:23 AM                83 app_version.h
06/03/2020  10:18 AM            65,336 da14531_multi_part_proxr.img
05/11/2020  10:18 AM            23,040 mkimage.exe
05/27/2020  10:31 AM            18,816 pxr_sr_coex_ext_531_6_0_14_1114.bin
06/03/2020  10:18 AM            18,880 pxr_sr_coex_ext_531_6_0_14_1114_1.img
06/03/2020  10:18 AM            18,880 pxr_sr_coex_ext_531_6_0_14_1114_2.img
               6 File(s)        145,035 bytes
               2 Dir(s)  39,420,968,960 bytes free

C:\temp_folder>
```

Figure 10: Bluetooth® LE FW Image Generation – Multi-Part Image

4.4.1.3 Create a Single Image

You can create a single image in two ways using:

- Option 1 – the Python script included in SDK
- Option 2 – raw commands in command prompt

Option 1

If you have Python 3.6.x installed on your PC, you can create an image following these steps:

1. In Python 3.6.x, go to "`~SDK\util\ble_img_creator\`" with the following files:
   - mkimage.exe
   - app_version.h
   - **mkimage_single.py**
2. Copy the DA14531 bin file to "`~SDK\util\ble_img_creator\`".
3. Open **app_version.h**, change the version to your version, and then save it.
   For example: #define SDK_VERSION "6.0.14.1114.**1**" → #define SDK_VERSION "6.0.14.1114.**2**"
4. In Windows Explorer, double-click **mkimage_single.py**, and then follow the instructions.
   a. If you want to give your name to .img, type the name in the command prompt window, then press ENTER.

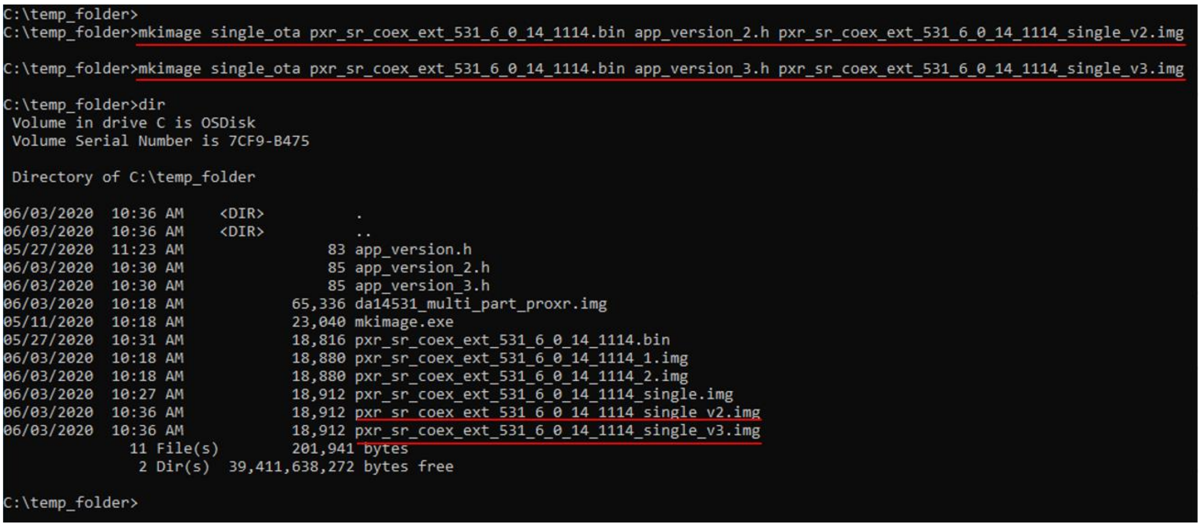
The following image is created: **da14531_pxr_single.img**

Option 2

If you want to create two single images (ver2 and ver3) using a command prompt, do the following:

1. Create two copies of **app_version.h** and rename them to **app_version_2.h** and **app_version_3.h** correspondingly.
2. Edit the two files as follows:
   a. App_version_2.h:
   `#define SDK_VERSION "v_6.0.14.1114.2"`
   b. App_version_3.h:
   `#define SDK_VERSION "v_6.0.14.1114.3"`

3. At the command prompt, type the following commands in the following order (see Figure 11):
   a. `mkimage single_ota pxr_sr_coex_ext_531_6_0_14_1114.bin app_version_2.h pxr_sr_coex_ext_531_6_0_14_1114_single_v2.img.`
   b. `mkimage single_ota pxr_sr_coex_ext_531_6_0_14_1114.bin app_version_3.h pxr_sr_coex_ext_531_6_0_14_1114_single_v3.img.`



```
C:\temp_folder>
C:\temp_folder>mkimage single_ota pxr_sr_coex_ext_531_6_0_14_1114.bin app_version_2.h pxr_sr_coex_ext_531_6_0_14_1114_single_v2.img

C:\temp_folder>mkimage single_ota pxr_sr_coex_ext_531_6_0_14_1114.bin app_version_3.h pxr_sr_coex_ext_531_6_0_14_1114_single_v3.img

C:\temp_folder>dir
 Volume in drive C is OSDisk
 Volume Serial Number is 7CF9-B475

 Directory of C:\temp_folder

06/03/2020  10:36 AM    <DIR>          .
06/03/2020  10:36 AM    <DIR>          ..
05/27/2020  11:23 AM                83 app_version.h
06/03/2020  10:30 AM                85 app_version_2.h
06/03/2020  10:30 AM                85 app_version_3.h
06/03/2020  10:18 AM            65,336 da14531_multi_part_proxr.img
05/11/2020  10:18 AM            23,040 mkimage.exe
05/27/2020  10:31 AM            18,816 pxr_sr_coex_ext_531_6_0_14_1114.bin
06/03/2020  10:18 AM            18,880 pxr_sr_coex_ext_531_6_0_14_1114_1.img
06/03/2020  10:18 AM            18,880 pxr_sr_coex_ext_531_6_0_14_1114_2.img
06/03/2020  10:27 AM            18,912 pxr_sr_coex_ext_531_6_0_14_1114_single.img
06/03/2020  10:36 AM            18,912 pxr_sr_coex_ext_531_6_0_14_1114_single_v2.img
06/03/2020  10:36 AM            18,912 pxr_sr_coex_ext_531_6_0_14_1114_single_v3.img
              11 File(s)        201,941 bytes
               2 Dir(s)  39,411,638,272 bytes free

C:\temp_folder>
```

Figure 11: Bluetooth® LE FW Image Generation – Single Image

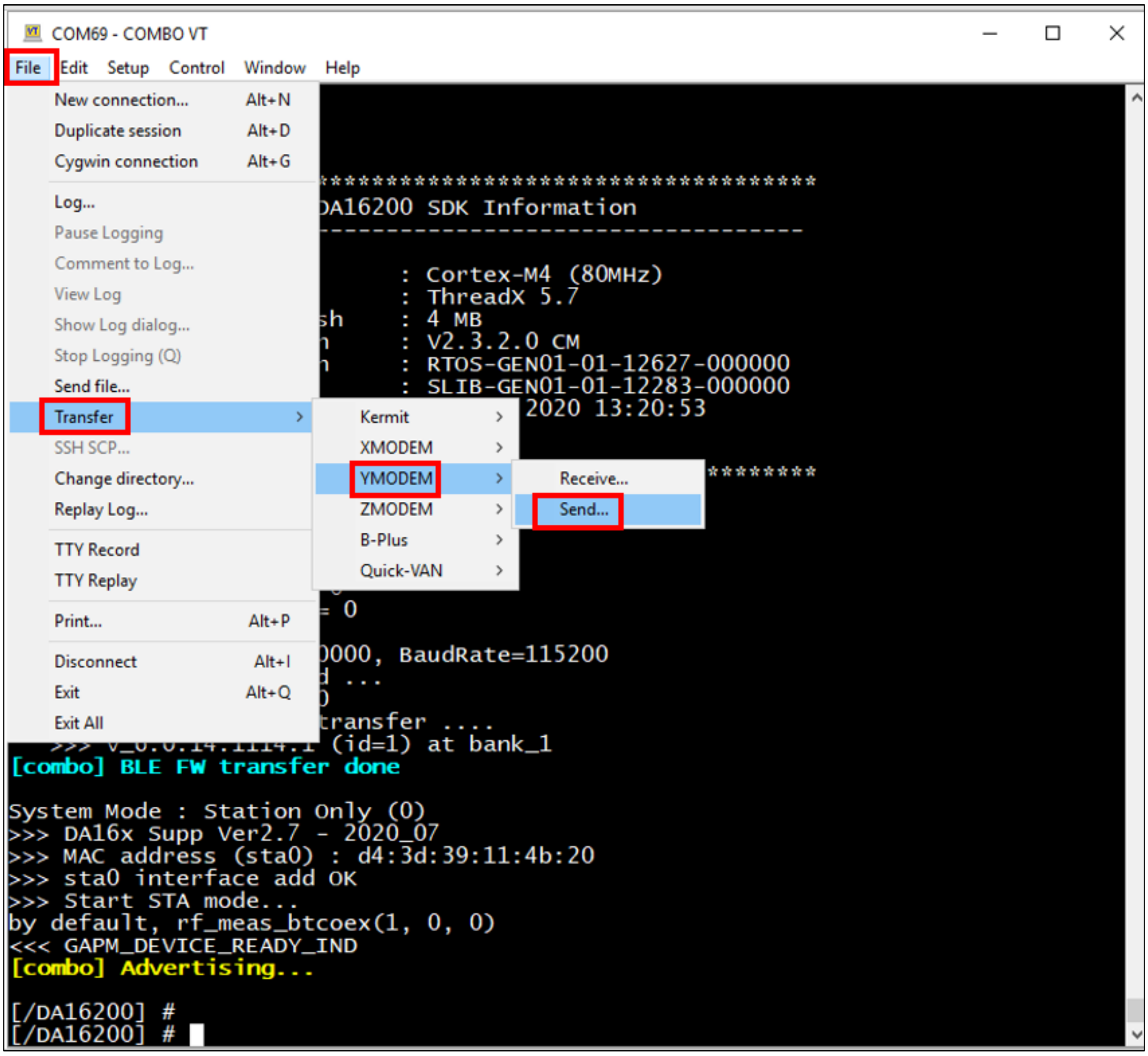
4.4.2 Program SFLASH

To program SFLASH:

1. Make sure that the following four images are ready:
   - DA16200_**BOOT**-GEN01-01-XXXXX-000000_W25Q32JW.img (or DA16200_**BOOT**-GEN01-01-XXXXX-000000_AT25SL321.img)
   - DA16200_**SLIB**-GEN01-01-XXXXX-000000.img
   - DA16200_**RTOS**-GEN01-01-XXXXX-000000.img
   - **da14531_multi_part_proxr.img**
2. Put a micro-USB cable in USB socket **B** of the EVB (see Figure 1).
   The recommendation is to put the other end of this USB cable in a USB hub with a power switch attached per port and connect that USB hub to the PC to make a “power cycle” of the board easy during the test.
3. Run **Tera Term**.
   Windows will detect two USB ports (for example, COM11 and COM12).
4. Connect Tera Term to the lowest of the two COM port numbers that were detected (for example, COM11, which is UART0 of DA16200).
5. Make sure that the baud rate is 230400.
6. Press **Enter** several times to check that you are online with the DA16600 EVB.
7. Type `reset` and then press **Enter**.
   You will see the `[MROM]` prompt.
8. Before you type any commands, copy the location path to the folder where the four firmware images are stored (BOOT, SLB, RTOS, and bin.out) in advance, so that you do not get a timeout error while running the `loady` command. If you select a file too slow, the Tera Term's ymodem transfer progress dialog box is stuck or not working, then you need to re-run the `loady` command.

9. Type `loady boot` and press **Enter**.

10. Go to **File** > **Transfer** > **YMODEM** > **Send** to quickly find file **DA16200_BOOT-GEN01-01-XXXXX-000000_W25Q32JW.img** (or **DA16200_BOOT-GEN01-01-XXXXX-000000_AT25SL321.img**) (see Figure 12).
    You can also use this shortcut key: keep the **Alt** key pressed and type **F**, **T**, **Y**, **S**.



```
*************************************
)A16200 SDK Information
-----------------------------------
          : Cortex-M4 (80MHz)
          : ThreadX 5.7
sh        : 4 MB
n         : V2.3.2.0 CM
n         : RTOS-GEN01-01-12627-000000
          : SLIB-GEN01-01-12283-000000
             2020 13:20:53

                             ********

= 0

)000, BaudRate=115200
d ...
)
transfer ....
>>> V_0.0.14.1114.1 (id=1) at bank_1
[combo] BLE FW transfer done

System Mode : Station Only (0)
>>> DA16x Supp Ver2.7 - 2020_07
>>> MAC address (sta0) : d4:3d:39:11:4b:20
>>> sta0 interface add OK
>>> Start STA mode...
by default, rf_meas_btcoex(1, 0, 0)
<<< GAPM_DEVICE_READY_IND
[combo] Advertising...

[/DA16200] #
[/DA16200] #
```

Figure 12: Tera Term

The download (to serial flash) of the selected image file takes time.

11. Similar to step 10, type `loady 18a000`, press **Enter**, and then select **DA16200_SLIB-GEN01-01-XXXXX-000000.img**.

12. Similar to steps 10, type `loady a000`, press **Enter**, and then select **DA16200_RTOS-GEN01-01-XXXXX-000000.img**.
    This ymodem transfer takes the longest time to complete.

13. Similar to step 10, type `loady 392000 1000 bin` , press **Enter**, and then select **da14531_multi_part_proxr.img**.

NOTE
Next time, when you want to, for example, download only one image file (RTOS, SLIB, or the da14531 image), you must always download the BOOT image first (`loady boot`), and then run either `loady a000 / loady 18a000 / loady 392000 1000 bin`. If you do not download the BOOT image beforehand, your image (RTOS/SLIB/Bluetooth® LE image) may not be correctly programmed.

14. After all images are transferred to SFLASH, type `boot`, and then press **Enter**.
    Some debug messages are printed in Tera Term.
15. Press **Enter** several times until the prompt `[/DA16200] #` shows**.**

IMPORTANT
Switch off and switch on the USB port (if a user USB hub has a power switch per port, then toggle it) or remove the USB cable completely and then put it back in. This is needed to change the DA14531 to Bootloader mode and wait to get an image from the DA16200.

16. Wait until your Tera Term is connected to COM11. (Or simply reconnect with serial. This step is not needed if you are not going to use Tera Term during the test.)

NOTE
Once Tera Term is connected, you can type `reboot` in the Tera Term console if you want to check early boot messages.

Now you are ready to do the test.

4.5 Run DA16600 with JTAG

After the steps in Section 4.4 are done, you are ready to run the example applications that are described in Section 5.

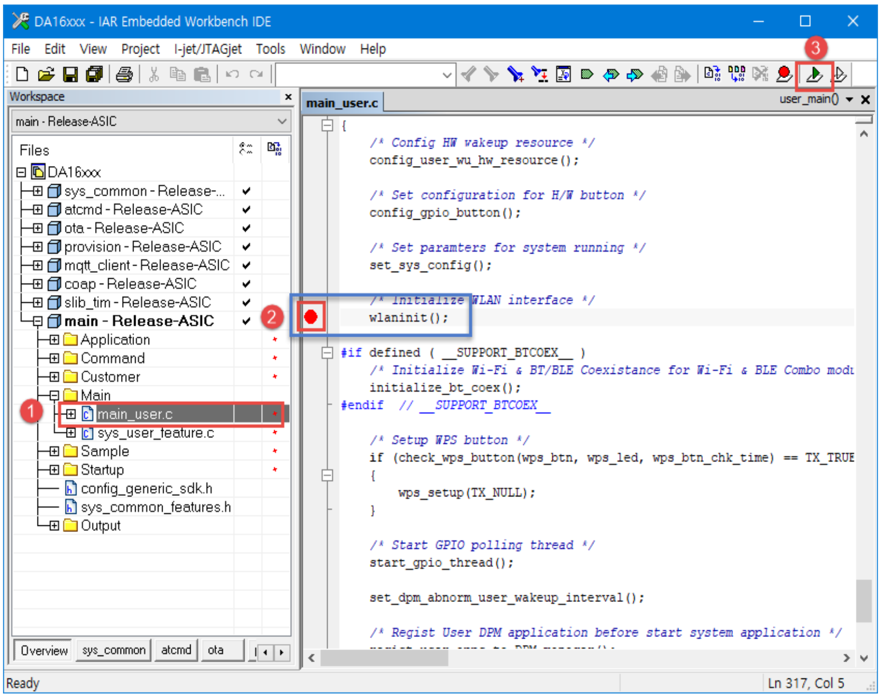
You can also use JTAG to run DA16600. Because DA16600 has two chips – Wi-Fi (DA16200) and Bluetooth® LE (DA14531) – and each chip has its own JTAG port.

4.5.1 Run DA16200 with JTAG

See the DA16200 SDK Programmer Guide [1] Appendix D "How to use I-Jet Debugger".

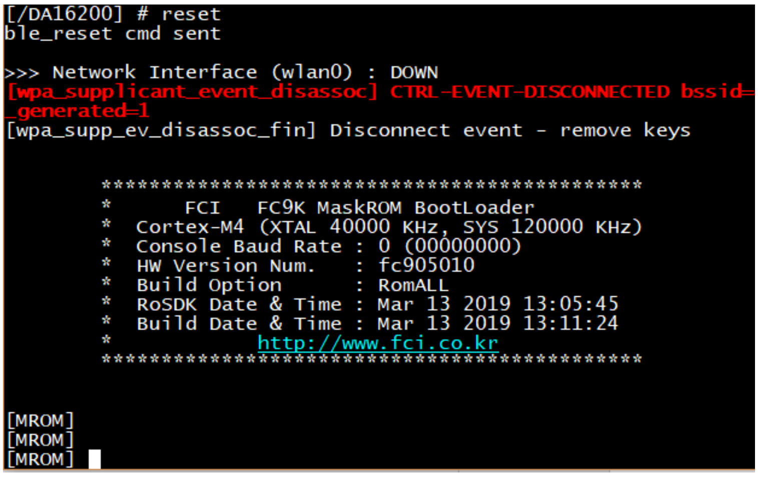
The JTAG cable should be connected to jumper **B** (J7) of the DA16600 EVB (Figure 1).

If you want to do debugging, enter the [MROM] prompt in Tera Term (Figure 14), and then click the debug start button (button 3 in Figure 13, which is taken from the DA16200 SDK Programmer Guide).



```
{
    /* Config HW wakeup resource */
    config_user_wu_hw_resource();

    /* Set configuration for H/W button */
    config_gpio_button();

    /* Set paramters for system running */
    set_sys_config();

    /* Initialize WLAN interface */
    wlaninit();

#if defined ( __SUPPORT_BTCOEX__ )
    /* Initialize Wi-Fi & BT/BLE Coexistance for Wi-Fi & BLE Combo modu
    initialize_bt_coex();
#endif  // __SUPPORT_BTCOEX__

    /* Setup WPS button */
    if (check_wps_button(wps_btn, wps_led, wps_btn_chk_time) == TX_TRUE
    {
        wps_setup(TX_NULL);
    }

    /* Start GPIO polling thread */
    start_gpio_thread();

    set_dpm_abnorm_user_wakeup_interval();

    /* Regist User DPM application before start system application */
```

Figure 13: Run DA16200 by JTAG



```
[/DA16200] # reset
ble_reset cmd sent

>>> Network Interface (wlan0) : DOWN
[wpa_supplicant_event_disassoc] CTRL-EVENT-DISCONNECTED bssid=
_generated=1
[wpa_supp_ev_disassoc_fin] Disconnect event - remove keys


        **********************************************
        *      FCI    FC9K MaskROM BootLoader
        *   Cortex-M4 (XTAL 40000 KHz, SYS 120000 KHz)
        *   Console Baud Rate : 0 (00000000)
        *   HW Version Num.   : fc905010
        *   Build Option      : RomALL
        *   RoSDK Date & Time : Mar 13 2019 13:05:45
        *   Build Date & Time : Mar 13 2019 13:11:24
        *             http://www.fci.co.kr
        **********************************************


[MROM]
[MROM]
[MROM] 
```

Figure 14: Run DA16200 by JTAG – MROM

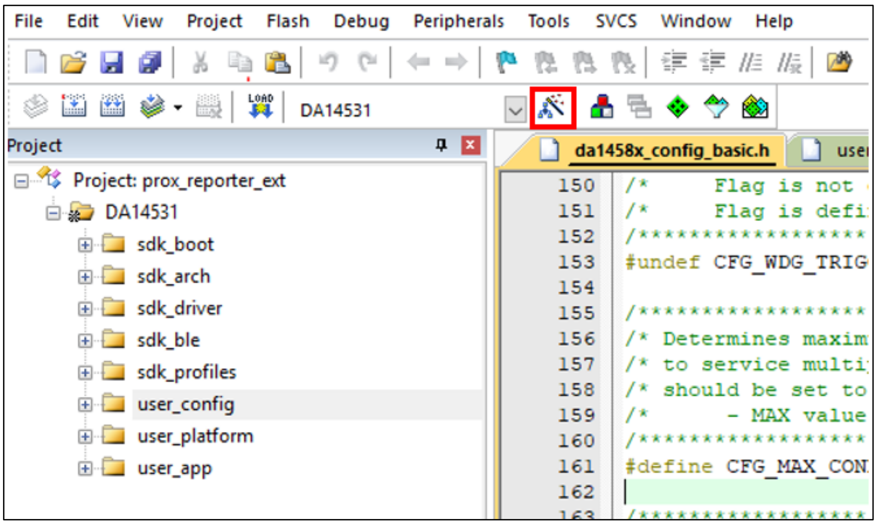
If you want to boot the DA16600 EVB in "non" JTAG mode again after J-Tag is used, then SPI re-programming with three DA16200 images is required. This is because the memory map is different for JTAG boot and normal boot – as DA16200 is using XIP: J-tag writes code in SFLASH by own memory map which is not the same as the DA16600's memory map (see Sections 4.1 and 4.4.2).

4.5.2 Run DA14531 with JTAG

To load a DA14531 image (.bin) with the JTAG function in the Keil IDE:

NOTE
The default DA16200 software loads and transfers a DA14531 image to DA14531 at boot. Disable this Bluetooth® LE image transfer feature before starting the procedure.

1. Build the DA16600 SDK with __DA14531_BOOT_FROM_UART__ disabled (see `~SDK/apps/da16600/get_started/inc/ble_combo_features.h`), and program SFLASH with the three DA16200 images (BOOT, SLB, and RTOS).
   The DA14531 image does not need to be programmed.
2. Make sure DA16600 EVB's DIP switch configuration (SW7 and SW3) looks like Figure 26.
3. Connect a USB cable to USB Port **C** (DA14531 JTAG Port) of the DA16600 EVB. See Figure 1.
4. Connect a USB cable to USB Port **B** of the DA16600 EVB for a Tera Term connection.
5. Switch **ON** (in a USB hub) the two USB cable connections.
6. Run the Keil IDE and open a DA14531 project.
7. Click the icon shown in Figure 15.

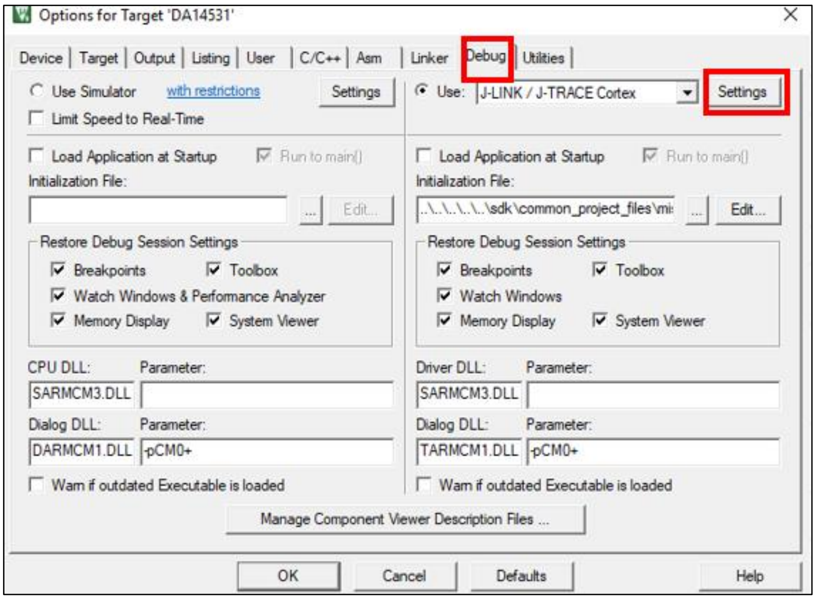


- Project: prox_reporter_ext
  - DA14531
    - sdk_boot
    - sdk_arch
    - sdk_driver
    - sdk_ble
    - sdk_profiles
    - user_config
    - user_platform
    - user_app

```
/*     Flag is not
/*     Flag is defi
/*****************
#undef CFG_WDG_TRIG

/*****************
/* Determines maxim
/* to service multi
/* should be set to
/*       - MAX value
/*****************
#define CFG_MAX_CON

/*****************
```

Figure 15: Keil – Option

8. Select the **Debug** tab and click **Settings**. See Figure 16.



○ Use Simulator
□ Limit Speed to Real-Time

□ Load Application at Startup ☑ Run to main()

Initialization File:

Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watch Windows & Performance Analyzer
☑ Memory Display ☑ System Viewer

CPU DLL: SARMCM3.DLL Parameter:

Dialog DLL: DARMCM1.DLL Parameter: -pCM0+

□ Warn if outdated Executable is loaded

● Use: J-LINK / J-TRACE Cortex

□ Load Application at Startup ☑ Run to main()

Initialization File: ..\..\..\..\sdk\common_project_files\mi

Restore Debug Session Settings
☑ Breakpoints ☑ Toolbox
☑ Watch Windows
☑ Memory Display ☑ System Viewer

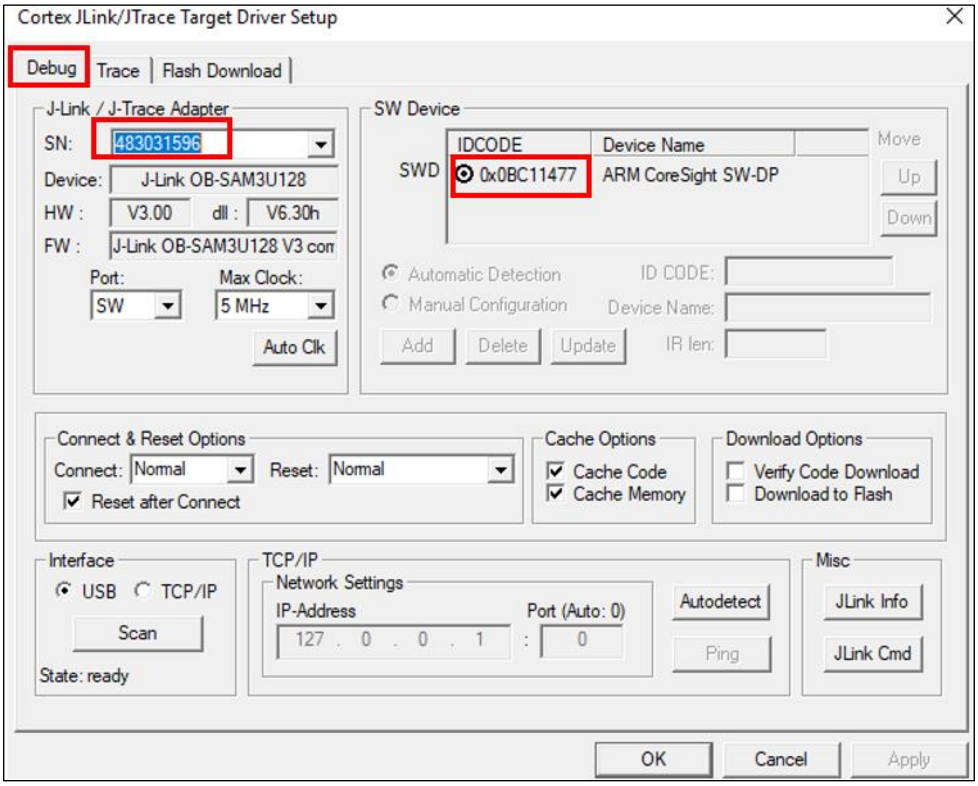
Driver DLL: SARMCM3.DLL Parameter:

Dialog DLL: TARMCM1.DLL Parameter: -pCM0+

□ Warn if outdated Executable is loaded

Figure 16: Keil – Debug

9. Make sure that the fields **SN** and **SWD** have valid values. See Figure 17.



SN: 483031596

Device: J-Link OB-SAM3U128

HW: V3.00 dll: V6.30h

FW: J-Link OB-SAM3U128 V3 com

Port: SW Max Clock: 5 MHz

	IDCODE	Device Name
SWD	⊙ 0x0BC11477	ARM CoreSight SW-DP

Connect: Normal Reset: Normal

☑ Reset after Connect

☑ Cache Code

☑ Cache Memory

☐ Verify Code Download

☐ Download to Flash

● USB ○ TCP/IP

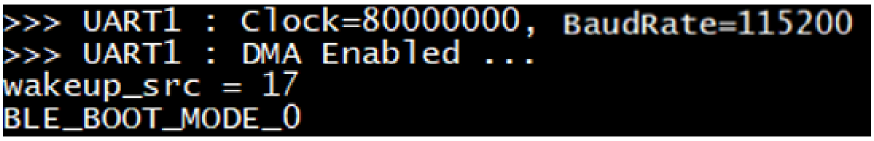
State: ready

IP-Address 127 . 0 . 0 . 1 Port (Auto: 0) : 0

Figure 17: Keil/JTAG Device

NOTE
If there is no valid value for **SN** or **SWD**, then this means that the JTAG/JLINK firmware does not exist or is not enabled in the JTAG chip of the DA16600, or the JTAG is not working for some reason. In this case, contact Dialog Semiconductor to update your JTAG firmware on the DA16600 EVB.

10. If the DA14531 JTAG is successfully recognized, click **OK**.
11. Switch **OFF** the power to the two USB cables.
12. Switch **ON** the power to the two USB cables.
13. In Tera Term (to which the lower number COM port is connected), make sure that the DA16200 boots successfully. If successful, you will see messages as shown in Figure 18.



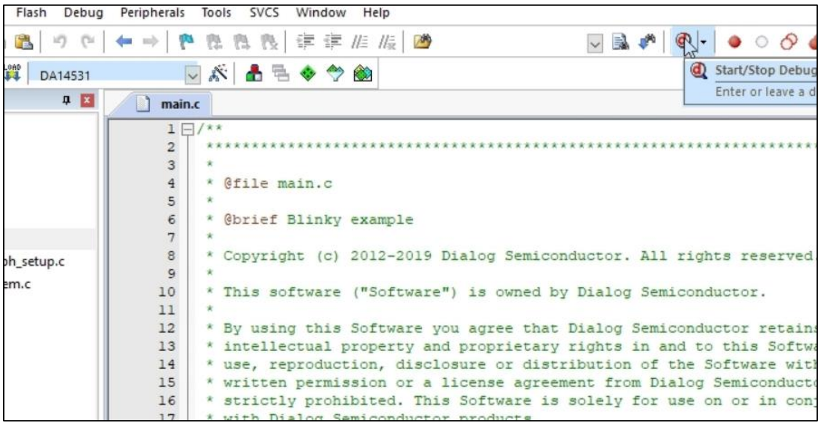
```
>>> UART1 : Clock=80000000, BaudRate=115200
>>> UART1 : DMA Enabled ...
wakeup_src = 17
BLE_BOOT_MODE_0
```

Figure 18: Tera Term – DA16200 Waiting for DA14531 to Connect

14. Enable JTAG SWD pins by changing a compiler flag:

```
[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex
\include\ext_host_ble_aux_task.h
…
#undef __DISABLE_J-TAG_SWD_PINS_IN_BLE__
…
```

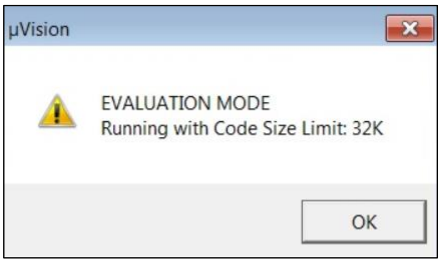
15. After the build is done, click the **Start Debugger** button. See Figure 19.



```
/**
 ***********************************************************************
 *
 * @file main.c
 *
 * @brief Blinky example
 *
 * Copyright (c) 2012-2019 Dialog Semiconductor. All rights reserved
 *
 * This software ("Software") is owned by Dialog Semiconductor.
 *
 * By using this Software you agree that Dialog Semiconductor retain
 * intellectual property and proprietary rights in and to this Softwa
 * use, reproduction, disclosure or distribution of the Software with
 * written permission or a license agreement from Dialog Semiconducto
 * strictly prohibited. This Software is solely for use on or in con
```

Figure 19: Keil – Start Debugger

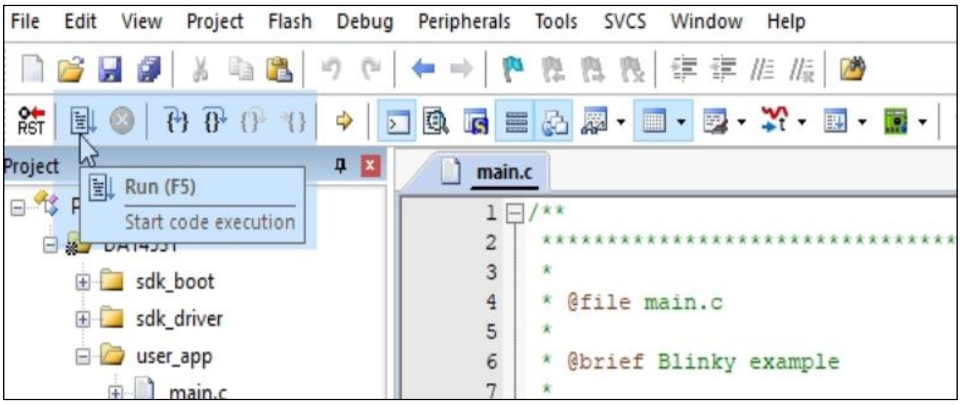
16. Click **OK**.


µVision

EVALUATION MODE
Running with Code Size Limit: 32K

Figure 20: Keil – Evaluation Mode Dialog Box

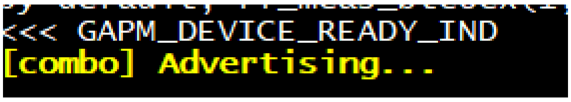
17. Click the **Run** button. See Figure 21.



```
/**
 ********************************
 *
 * @file main.c
 *
 * @brief Blinky example
 *
```

Figure 21: Keil – Run

If you see the message shown in Figure 22 being printed in Tera Term, then the DA14531 is successfully started.



```
<<< GAPM_DEVICE_READY_IND
[combo] Advertising...
```

Figure 22: Tera Term – Advertising Successful

Now the DA16600 starts Bluetooth® LE advertising and can allow a Bluetooth® LE peer to connect to the DA16600.

5 DA16600 Example Test Procedure

5.1 Test Environment Setup

The following items are used in the example test:

- Wi-Fi Access Point
- Bluetooth® LE Peers
- PC: to Control Bluetooth® LE Peers and DA16600 Boards
- DA16600 EVB Boards

5.1.1 Wi-Fi Access Point

Any Wi-Fi routers are acceptable. The Wi-Fi Access Point is called **MyAP** from here onwards.

5.1.2 Bluetooth® LE Peers

Bluetooth® LE peers are used for all the examples. Several types of Bluetooth® LE peers are used, listed in the following sub-sections.

5.1.2.1 Bluetooth® LE Mobile App

Dialog provides a sample mobile application (Android/iOS App) called "Wi-Fi Provisioning" to test examples 1 to 4. This mobile application is used to give Wi-Fi provision information (Wi-Fi router connection information plus any customer proprietary information to configure the DA16600) to the DA16600 board.

NOTE
You can also download (from App Store) and use a general-purpose Bluetooth® LE mobile application that supports a GATT Client (that can read/write a GATT characteristic of a GATT Server). If you understand the Wi-Fi SVC GATT Server database structure and JSON application protocols (see Section 6.4.1.5), you can use a general Bluetooth® LE Mobile App as well to send a command.

5.1.2.2 Bluetooth® LE Sensors

Two or three Bluetooth® LE peer devices are needed to test example 4 (sensor gateway).

These Bluetooth® LE peer devices should implement a simple GATT server application as described in Figure 48 to be able to work with the sensor gateway application of DA16600.

For example, as Raspberry Pi 3 is common and capable of Bluetooth® LE communication, it is used in the example test. If you have another Bluetooth® LE development board/device to act as a peer, write a simple GATT server as described in Figure 48.

Suppose we have two Bluetooth® LE sensors ready and called **RBP3_1** and **RBP3_2**. Connect both RBP3s to MyAP (with the LAN interface of RBP3). Now RBP3_1 and RBP3_2 are on the same local network as MyAP.

In RBP3_1, we run a UDP server application as well to communicate with the Wi-Fi part of the DA16600. If you have another UDP test client/utility, use whatever network utility you like.

NOTE
The version of Bluetooth® LE Stack and Python used in RBP3: bluez-5.51, python 3.7.3.

5.1.3 PC: to Control Bluetooth® LE Peers and DA16600 Boards

1. Use a LAN cable to connect your PC to **MyAP**.
2. Use PuTTy to open two ssh windows (to RBP3_1) ← **login as root**.
3. Enter `cd /home/pi` for both two ssh windows (let us say **ssh_win_1**, **ssh_win_2**).
4. Use PuTTy to open one ssh window to RBP3_2 ← **login as root**.
5. Enter `cd /home/pi` for the ssh window (let us say **ssh_win_3**).
6. Open one Tera Term window and connect to the COM port (the lower port number of the two, with baud rate 230400) of the DA16600. Let us call this Tera Term window **da16_tera_win** from here onwards.
7. Open another Tera Term window and connect to the other COM port (the higher port number of the two, with baud rate 115200) of the DA16600. Let us call this Tera Term window **da14_tera_win** from here onwards. (this Tera Term is connected to UART2 of DA14531 chip)

5.1.4 DA16600 EVB Boards

- **DA16600_EVB_1** to program DA16200_SW_1 + DA14531_IMG (Prj1_proxr)
  (Depending on an example, replace DA16200_SW_1 with DA16200_SW_1 / DA16200_SW_2 / DA16200_SW_3)
- **DA16600_EVB_2** to program DA16200_SW_4 + DA14531_IMG (Prj2_proxm)

5.2 Example 1: Bluetooth® LE Assisted Wi-Fi Provisioning

A Bluetooth® LE mobile application is used for the Wi-Fi provisioning test. This mobile application is a Bluetooth® LE application and can talk to the DA14531 of the DA16600 EVB board to do Wi-Fi provisioning.

1. Power **ON** DA16600_EVB_1.
   a. Do a POR boot (plug out, and then plug in the USB cable).
   b. After boot, run the command `factory` to clear any existing NVRAM content.
      The command `factory` also triggers a reboot after NVRAM is cleared.
   c. Make sure that "*Advertising …*" is printed on da16_tera_win.
2. Run the provisioning mobile Wi-Fi provisioning App as shown in Figure 23 that can be found and installed from the Google Play Store or iOS App Store.

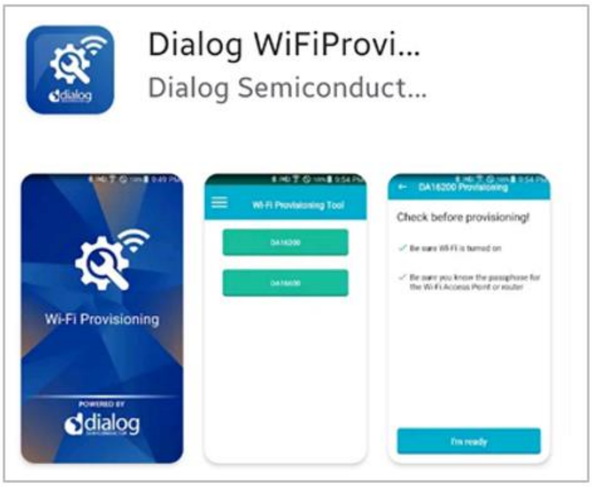


Figure 23: Provisioning App (Android)

3. Configure Wi-Fi as shown in Section 5.1 of the Ref. [5].
4. If a user clicks **Start DA16600-based > Start > Select 'DA16600-BLE' > Start Wi-Fi network scan > select [MyAP name]** `(input Password if need)` **> Connect to [MyAP name]**, then the provisioning information is transferred to DA16600, which saves the information into NVRAM and rebooted. After rebooted, Wi-Fi is connected to the selected AP.

To remove the provisioning information:

1. Establish a Bluetooth® LE connection again with DA16600_EVB_1.
2. Click the **Reset the device** button.

NOTE
As an alternative, you also can use the command `factory` to clear any provisioning information.

Now you can start provisioning again.

5.3 Example 2: Bluetooth® LE Firmware OTA Download via Wi-Fi

- DA16200_SW_1: make sure that the DA16200 RTOS image is built with __FORCE_LOADY_BANK_1_BLE_BOOT__ **disabled**
- For this test, make sure that an HTTP(s) server is running on the MyAP network

You can set up a simple HTTP server. For example:

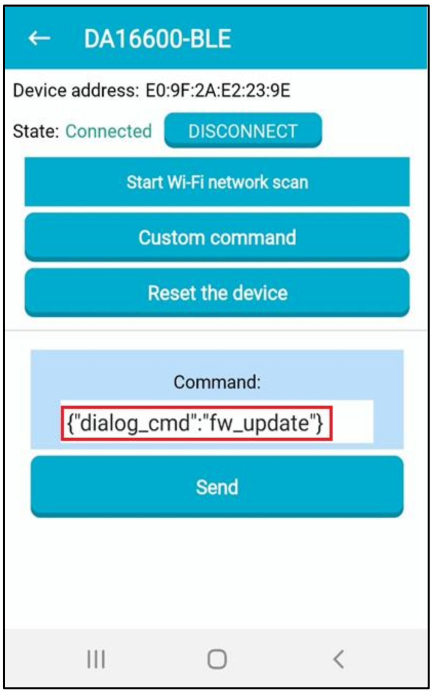
1. Install an HTTP server on a PC with Apache as a web server that is connected to MyAP. See https://https.apache.org/ to download Apache and for instructions.
2. Copy file **pxr_sr_coex_ext_ver3_single.img** to the **htdocs** folder of the Apache server. This assumes that you have generated a ver3 image as well in Section 4.4.1.
3. Make sure that the PC is connected to MyAP, and make sure that the **.img** file is downloadable via a web browser with the link http://192.168.0.230/pxr_sr_coex_ext_ver3_single.img.

- da16_tera_win

```
// User command in bold font, commentary / messages to note in blue font

// Add one NVRAM parameter as below. This may be a part of provision information or
hardcoded in source depending on implementation.

[/DA16200] # nvram
    Command-List is changed, "NVRAM"
```

```
[/DA16200/NVRAM] setenv URI_BLE http://192.168.0.230/pxr_sr_coex_ext_ver3_single.img
[/DA16200/NVRAM] reboot
...
Reboot...

ble_reset cmd sent

Wakeup source is 0x00

        **************************************************
        *              DA16200 SDK Information
        * -----------------------------------------------
        *
        * - CPU Type        : Cortex-M4 (80MHz)
        * - OS Type         : ThreadX 5.7
        * - Serial Flash    : 4 MB
        * - SDK Version     : V2.3.2.0 CM
        * - F/W Version     : RTOS-GEN01-01-12616-000000
        *                   : SLIB-GEN01-01-12283-000000
        * - F/W Build Time  : Sep 22 2020 18:46:28
        * - Boot Index      : 0
        *
        **************************************************
gpio wakeup enable 00010000

System Mode : Station Only (0)
>>> DA16XXX supplicant Ver1.00-20170213-01
>>> MAC address (sta0) : d4:3d:39:10:f4:92
>>> sta0 interface add OK
>>> Start STA mode...
by default, rf_meas_btcoex(1, 0, 0)

>>> UART1 : Clock=80000000, BaudRate=115200
>>> UART1 : DMA Enabled ...
wakeup_src = 17
BLE_BOOT_MODE_0
BLE FW VER to transfer ....
   >>> v_6.0.14.1114.2 (id=1) at bank_1 // During boot, the latest DA14531 firmware
is loaded (from either bank_1 or bank_2). The current version of Bluetooth® LE FW:
6.0.14.1114.2. the current bank: bank_1
>>> Selected BSS 88:36:6c:4e:a1:28 ssid='N604R_MIKE' (-89)
BLE FW transfer done // Bluetooth® LE FW transferred successfully to DA14531 via
UART
<<< GAPM_DEVICE_READY_IND
Advertising... // Bluetooth® LE starts advertising ...
          [wpa_scan_res_match] skip- blacklisted (count=1 limit=0)

!!! No proper APs found - It will be try again !!!

>>> Selected BSS 88:36:6c:4e:a1:28 ssid='N604R_MIKE' (-55)
>>> Network Interface (wlan0) : UP
>>> Associated with 88:36:6c:4e:a1:28

Connection COMPLETE to 88:36:6c:4e:a1:28

-- DHCP Client WLAN0: SEL
-- DHCP Client WLAN0: REQ
-- DHCP Client WLAN0: BOUND
         Assigned addr   : 192.168.0.28 // DA16200 attached to MyAP network
               netmask   : 255.255.255.0
               gateway   : 192.168.0.1
               DNS addr  : 168.126.63.1

         DHCP Server IP  : 192.168.0.1
         Lease Time      : 02h 00m 00s
         Renewal Time    : 01h 00m 00s
iot_sensor connected to server

[/DA16200] #
[/DA16200] #
[/DA16200] #
[/DA16200] #
...
[/DA16200/NVRAM] # getenv // check nvram by making sure valid URI_BLE exists

Total length (271)
N0_Profile (STR,02) .......... 1
N0_ssid (STR,13) ............. "N604R_MIKE"
N0_psk (STR,12) .............. "N12345678"
N0_key_mgmt (STR,08) ......... WPA-PSK
SYSMODE (STR,02) ............. 0
0:NETMODE (STR,02) ........... 1
UART1_FLOWCTRL (STR,02) ...... 1
UART1_BAUDRATE (STR,07) ...... 115200
ble_fw_act_bank (STR,02) ..... 1
provisioned (STR,02) ......... 1
URI_BLE (STR,53) ............. http://192.168.0.230/pxr_sr_coex_ext_ver3_single.img
```

- Bluetooth® LE Mobile App



Device address: E0:9F:2A:E2:23:9E

State: Connected

Command:

{"dialog_cmd":"fw_update"}

Figure 24: Provisioning Application – Custom Command

- ○ Bluetooth® LE Mobile Application > Scan > Connect to "DA16600" > tap **Custom command** (see Figure 24), fill in the command **{"dialog_cmd":"fw_update"}** and tap **Send**
- ○ To enter the command easily, open a notepad on your smartphone to type the command, and then copy and paste the command on the Bluetooth® LE Mobile Application

- da16_tera_win
  The following is happening on DA16600_EVB_1 when command "fw_update" is received:
  - ○ An attempt is made to connect to an OTA server (**192.168.0.230**) to download a DA14531 firmware file

```
// User command in bold font, commentary / messages to note in blue font
...
[/DA16200/NVRAM] #
[/DA16200/NVRAM] # <<< GAPC_CONNECTION_REQ_IND // indication of connection req from a
Bluetooth® LE peer (mobile application)
    peer bd addr type = 0x1 (0x0:public, 0x1:random)
    connection event intval = 36, connection latency = 0

                ######################################################
                #    DA14531 Proxr IoT Sensor demo application    #
                ######################################################

                         Connected to Device // connected to a peer

BLE Peer BDA: 45:23:32:3c:f9:61 Bonded: NO
    GATTC_EXC_MTU_CMD (GATTC_MTU_EXCH) sent !
...
<<< GATTC_WRITE_REQ_IND
    start_handle = 33
    param->handle = 34
 Receive - FW_UPDATE
 COMBO_WIFI_CMD_FW_BLE_DOWNLOAD received // "fw_update" received
```

```
[ota_fw_update_combo] uri_slib =
[ota_fw_update_combo] uri_rtos =
[ota_fw_update_combo] uri_ble = http://192.168.0.230/pxr_sr_coex_ext_ver3_single.img
[BLE_OTA] New FW: ver = v_6.0.14.1114.3, timestamp = 1587376260
[BLE_OTA] bank_1 (act): ver = v_6.0.14.1114.2, timestamp = 1588134660
[BLE_OTA] bank_2 : ver = v_6.0.14.1114.1, timestamp = 1588134660
...
   >> HTTP(s) Client Downloading... 100 %(17232/17232 Bytes)
...
- OTA Update : <BLE_FW> Download - Success

[BLE_OTA] CASE_1: BLE FW Update Only ...
ble_reset cmd sent
- OTA Update (BLE FW) : 0 seconds left to REBOOT....

>>> Network Interface (wlan0) : DOWN
[wpa_supplicant_event_disassoc] CTRL-EVENT-DISCONNECTED bssid=88:36:6c:4e:a1:28
reason=3 locally_generated=1
[wpa_supp_ev_disassoc_fin] Disconnect event - remove keys

Wakeup source is 0x00 // rebooted ...

        **************************************************
        *             DA16200 SDK Information
        * -----------------------------------------------
        *
        * - CPU Type        : Cortex-M4 (80MHz)
        * - OS Type         : ThreadX 5.7
        * - Serial Flash    : 4 MB
…
        **************************************************
gpio wakeup enable 00010000

System Mode : Station Only (0)
>>> DA16XXX supplicant Ver1.00-20170213-01
>>> MAC address (sta0) : d4:3d:39:10:f4:92
>>> sta0 interface add OK
>>> Start STA mode...
by default, rf_meas_btcoex(1, 0, 0)

>>> UART1 : Clock=80000000, BaudRate=115200
>>> UART1 : DMA Enabled ...
wakeup_src = 17
BLE_BOOT_MODE_0
BLE FW VER to transfer ....
   >>> v_6.0.14.1114.3 (id=2) at bank_2 // new Bluetooth® LE FW boots from bank_2
(bank_1 has the previous fw version)
BLE FW transfer done
<<< GAPM_DEVICE_READY_IND
Advertising...
>>> Selected BSS 88:36:6c:4e:a1:28 ssid='N604R_MIKE' (-42)
...
```

5.4 Example 3: Gas Leak Detection Sensor

- ssh_win_1

```
// User command in bold font, commentary / messages to note in blue font

// Type in the following command to start the UDP server

root@raspberrypi:/home/pi# python udp_server.py
UDP Server: waiting for a messsage ... at 172.16.30.136:10954
```

- da16_tera_win
  For Test Example 3, the provisioning command JSON string "select_ap" of the Provisioning App (see Section 6.4.1.5) should have the following data:
  - "dpm_mode": 1
  - "svr_addr": "172.16.30.136"
  - "svr_port": 10954

```
// User command in bold font, commentary / messages to note in blue font
Rebooted …
…
[/DA16200] # ble
    Command-List is changed, "ble"
[/DA16200/ble] # iot_sensor start
[ConsoleEvent] user command exists in queue
[/DA16200/ble] # APP_GAS_LEAK_SENSOR_START_CFM
sleep (rtm ON) entered // Wi-Fi enters sleep (while in sleep, keyboard input is not
working, wait for some minutes)
...
Wakeup source is 0x90 // Wi-Fi wakes up
gpio wakeup enable 00000402
[combo][iot_sensor]
   is_provisioned = 1
   is_sensor_started = 1
[combo] dpm_boot_type = 2
[combo] BLE_BOOT_MODE_1

>>> TIM STATUS: 0x00000000
by default, rf_meas_btcoex(1, 0, 0)
>>> Network Interface (wlan0) : DOWN
[wpa_supplicant_event_disassoc] CTRL-EVENT-DISCONNECTED bssid=2c:4d:54:dc:c8:90
reason=3 locally_generated=1
OK
OK
-- DHCP Client WLAN0: REQ(4)
>>> Network Interface (wlan0) : UP
>>> Associated with 2c:4d:54:dc:c8:90
<<< APP_GAS_LEAK_EVT_IND
gas leak occurred !!!
[UMAC DPM] beacon int(100) , dtim_period(3)

Connection COMPLETE to 2c:4d:54:dc:c8:90
L2 Packet process completed, Set DPM Sleep !!!
-- DHCP Client WLAN0: BOUND(5)
         Assigned addr   : 192.168.0.205
         Lease Time      : 24h 00m 00s
         Renewal Time    : 20h 00m 00s
[dpm_save_dhcpc_info] DHCPC Cancel
[combo] iot_sensor connected to server
>>> [msg_sent] : gas_leak occurred, plz fix it !!! // Wi-Fi posts a message to server
```

```
sleep (rtm ON) entered // Wi-Fi enters sleep again after message posting
```

- ssh_win_1

```
// User command in bold font, commentary / messages to note in blue font
UDP Server: waiting for a messsage ... at 172.16.30.136:10954
 >>> sensor_connected
 >>> [Gas Leak Sensor]: gas_leak occurred, go home and fix it!!!
```

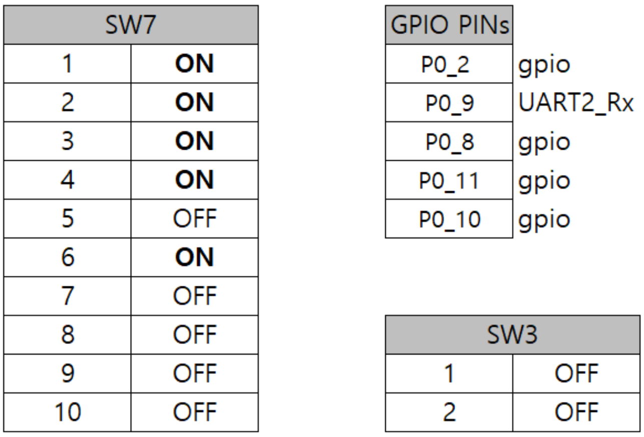
The following is happening:

- If you run the command "iot_sensor start", the command is sent over (via GTL) to DA14531 which starts a timer task that is supposed to read a gas leak density sensor periodically
- If the DA14531 reads that the density is above the threshold "gas leak" level, then DA14531 wakes up DA16200 and sends the event ("gas leak occurred!") to DA16200
  The DA16200, on receipt of the alert from the DA14531, sends an alert message to a UDP server where you can see the alert message printed

5.5 Example 4: DA14531 Peripheral Driver Example

5.5.1 Test Environment Setup

- DA16600 EVB Configuration
  You can use three configurations to test a sample:
  - Configuration_1



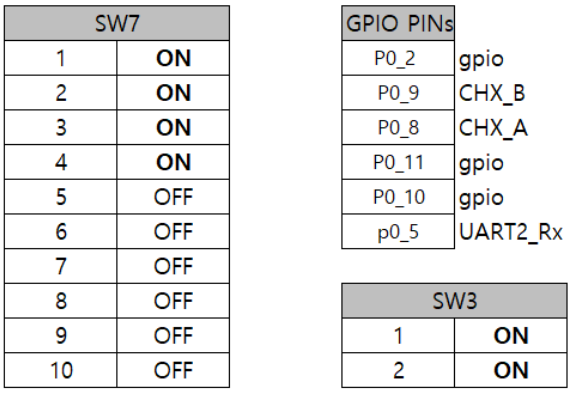
SW7	
1	**ON**
2	**ON**
3	**ON**
4	**ON**
5	OFF
6	**ON**
7	OFF
8	OFF
9	OFF
10	OFF

GPIO PINs	
P0_2	gpio
P0_9	UART2_Rx
P0_8	gpio
P0_11	gpio
P0_10	gpio

SW3	
1	OFF
2	OFF

Figure 25: DA16600 EVB Config. 1

  - Configuration_2



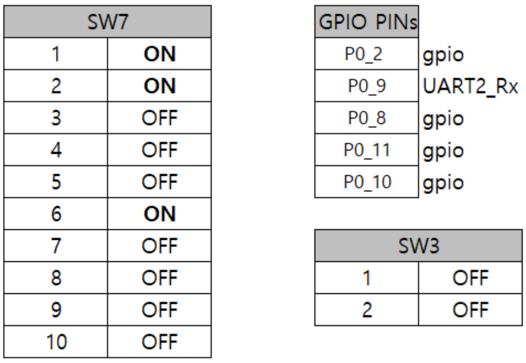
SW7	
1	**ON**
2	**ON**
3	**ON**
4	**ON**
5	OFF
6	OFF
7	OFF
8	OFF
9	OFF
10	OFF

GPIO PINs	
P0_2	gpio
P0_9	CHX_B
P0_8	CHX_A
P0_11	gpio
P0_10	gpio
p0_5	UART2_Rx

SW3	
1	**ON**
2	**ON**

Figure 26: DA16600 EVB Config. 2

- ○ Configuration_3



SW7	
1	**ON**
2	**ON**
3	OFF
4	OFF
5	OFF
6	**ON**
7	OFF
8	OFF
9	OFF
10	OFF

GPIO PINs	
P0_2	gpio
P0_9	UART2_Rx
P0_8	gpio
P0_11	gpio
P0_10	gpio

SW3	
1	OFF
2	OFF

Figure 27: DA16600 EVB Config. 3

- DA16200 image download for test
  Program SFLASH with the 4 images below. See Section 4.4.2
  - DA16200_SW_2 (build SDK):
    - DA16200_SLIB-GEN01-01-XXXXX-000000.img
    - DA16200_RTOS-GEN01-01-XXXXX-000000.img
    - DA16200_BOOT-GEN01-01-XXXXX-000000_W25Q32JW.img
      (or DA16200_BOOT-GEN01-01-XXXXX-000000_AT25SL321.img)
  - DA14531 image (pre-built image):
    - `[DA16600_SDK_ROOT]\img\DA14531_1\`**da14531_multi_part_proxr.img**
- Tera Term: open two Tera Term windows
  - Teraterm_1 (da16_tera_win): connect to COMxx (lower one) with 230400 as baud rate. This is debug console of DA16200. The user command is entered here
  - Teraterm_2 (da14_tera_win): connect to COMxx (higher one) with 115200 as baud rate. This is debug console of DA14531. Test progress are printed
- List of DA14531 Peripheral Driver samples
  Once DA16600 EVB is powered on, type the following commands to see which samples are available.

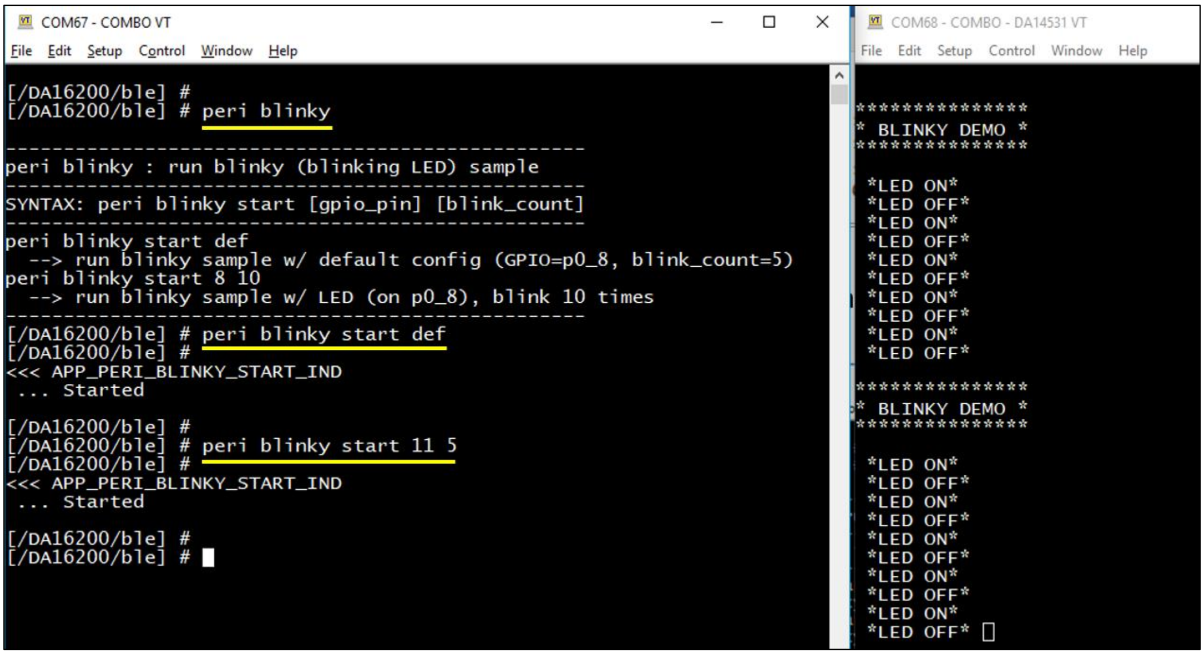
```
// User command in bold font, commentary / messages to note in blue font
```

```
…
System Mode : Station Only (0)
>>> DA16XXX supplicant Ver1.00-20170213-01
>>> MAC address (sta0) : d4:3d:39:11:4b:20
>>> sta0 interface add OK
>>> Start STA mode...
by default, rf_meas_btcoex(1, 0, 0)
wakeup_type=0

>>> UART1 : Clock=80000000, BaudRate=115200
>>> UART1 : DMA Enabled ...
BLE_BOOT_MODE_0
BLE FW VER to transfer ....
   >>> v_6.0.14.1114.1 (id=1) at bank_1
>>> Selected BSS 90:9f:33:66:26:52 ssid=xxxxxx' (-43)
>>> Network Interface (wlan0) : UP
>>> Associated with 90:9f:33:66:26:52

Connection COMPLETE to 90:9f:33:66:26:52

-- DHCP Client WLAN0: SEL
-- DHCP Client WLAN0: REQ
BLE FW transfer done
<<< GAPM_DEVICE_READY_IND
Advertising...
-- DHCP Client WLAN0: BOUND
         Assigned addr   : 192.168.0.24
               netmask   : 255.255.255.0
               gateway   : 192.168.0.1
               DNS addr  : 210.220.163.82

         DHCP Server IP  : 192.168.0.1
         Lease Time      : 02h 00m 00s
         Renewal Time    : 01h 00m 00s

[/DA16200] #
[/DA16200] # ble
    Command-List is changed, "ble"
[/DA16200/ble] # help

--------------------------------------------------
 Current CMD-List is "ROOT/ble"
--------------------------------------------------
...
- Sub-Commands - : --------------------------------
net              : Network commands
nvram            : nvram commands
sys              : system commands
user             : User commands
ble              : Combo BLE App commands

--------------------------------------------------

ble              : BLE application command
-------          : --------------------------------
ble_gapm_reset   : Reset BLE GAPM
ble_fw_ver       : Get BLE FW version
```

```
ble_disconnect    : Disconnect BLE connection
peri              : Run peripheral driver sample

[/DA16200/ble] # peri

---------------------------------------------------
peri : Run DA14531 Peripheral Driver Sample
       type a command below
---------------------------------------------------
[01] peri blinky     : blinking LED sample
[02] peri systick    : systick timer sample
[03] peri timer0_gen : timer0 general sample
[04] peri timer0_buz : timer0 PWM buzzer sample
[05] peri timer2_pwm : timer2 PWM LED array sample
[06] peri batt_lvl   : battery level read sample
[07] peri i2c_eeprom : I2C EEPROM read/write sample
[08] peri spi_flash  : SPI_flash read/write sample
[09] peri quad_dec   : Quadrature Decoder sample
[10] peri gpio       : GPIO contorl(High/Low)
---------------------------------------------------
[/DA16200/ble] #
```

5.5.2 User Commands

- peri blinky
  - This sample is using 1 GPIO to blink the LED that is connected to the GPIO
  - DA16600 EVB Configuration:
    - Configuration_1
    - By default, P0_8 is used to connect to LED. Connect J14:P0_8 to any pins in CN4
  - Run command as in Figure 28. The LED connected to the GPIO specified will blink

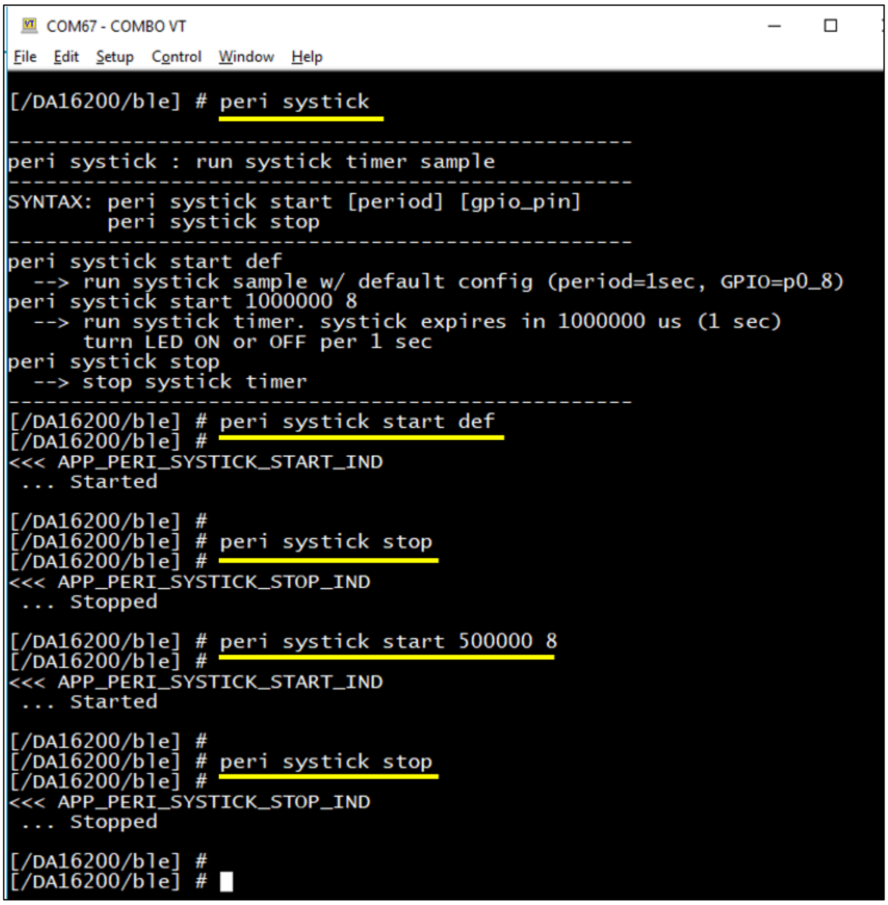

COM67 - COMBO VT
File Edit Setup Control Window Help
COM68 - COMBO - DA14531 VT
File Edit Setup Control Window Help

```
[/DA16200/ble] #
[/DA16200/ble] # peri blinky

---------------------------------------------------
peri blinky : run blinky (blinking LED) sample
---------------------------------------------------
SYNTAX: peri blinky start [gpio_pin] [blink_count]
---------------------------------------------------
peri blinky start def
  --> run blinky sample w/ default config (GPIO=p0_8, blink_count=5)
peri blinky start 8 10
  --> run blinky sample w/ LED (on p0_8), blink 10 times
---------------------------------------------------
[/DA16200/ble] # peri blinky start def
[/DA16200/ble] #
<<< APP_PERI_BLINKY_START_IND
 ... Started

[/DA16200/ble] #
[/DA16200/ble] # peri blinky start 11 5
[/DA16200/ble] #
<<< APP_PERI_BLINKY_START_IND
 ... Started

[/DA16200/ble] #
[/DA16200/ble] #
```

```
***************
* BLINKY DEMO *
***************

 *LED ON*
 *LED OFF*
 *LED ON*
 *LED OFF*
 *LED ON*
 *LED OFF*
 *LED ON*
 *LED OFF*
 *LED ON*
 *LED OFF*

***************
* BLINKY DEMO *
***************

 *LED ON*
 *LED OFF*
 *LED ON*
 *LED OFF*
 *LED ON*
 *LED OFF*
 *LED ON*
 *LED OFF*
 *LED ON*
 *LED OFF*
```

Figure 28: Peri Blinky

- peri systick
  - This sample uses a systick timer of DA14531
  - This sample is using 1 GPIO to change the state of the LED that is connected to the GPIO
  - DA16600 EVB Configuration:
    - Configuration_1
    - By default, P0_8 is used to connect to LED. Connect J14:P0_8 to any pins in CN4
  - Run command as in Figure 29. Whenever systick timer is expired, it toggles the LED state

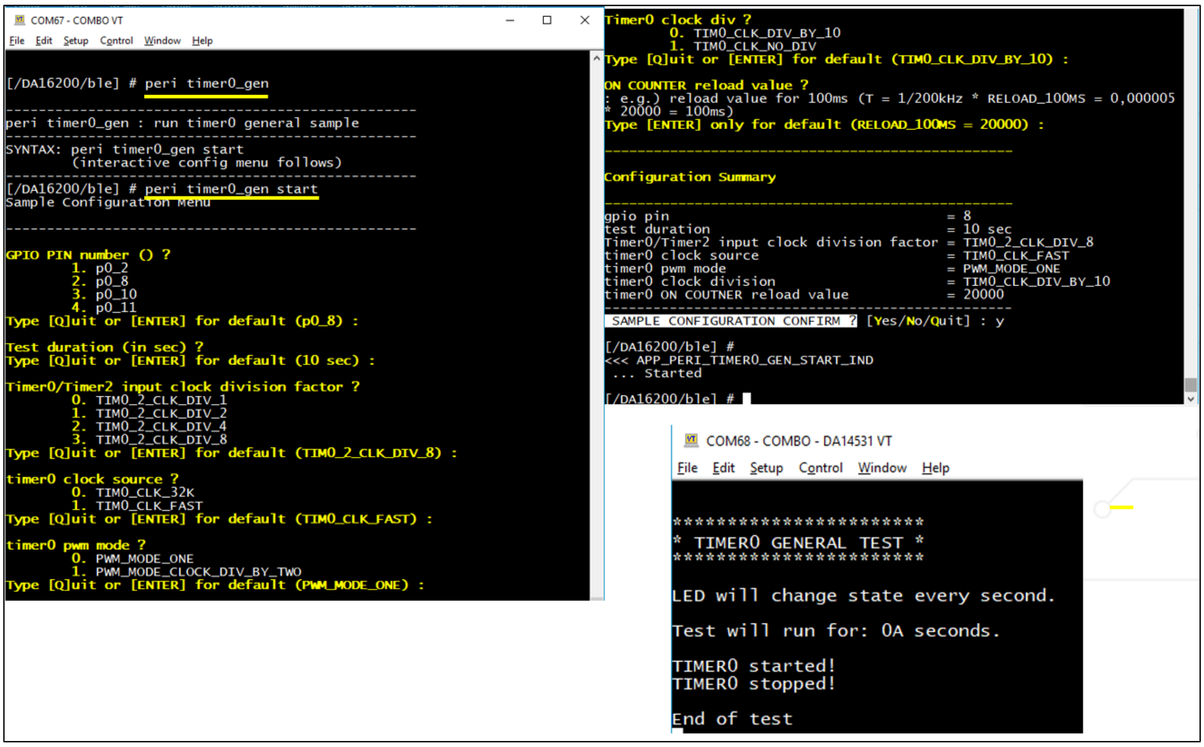


```
[/DA16200/ble] # peri systick

---------------------------------------------------
peri systick : run systick timer sample
---------------------------------------------------
SYNTAX: peri systick start [period] [gpio_pin]
        peri systick stop
---------------------------------------------------
peri systick start def
  --> run systick sample w/ default config (period=1sec, GPIO=p0_8)
peri systick start 1000000 8
  --> run systick timer. systick expires in 1000000 us (1 sec)
      turn LED ON or OFF per 1 sec
peri systick stop
  --> stop systick timer
---------------------------------------------------
[/DA16200/ble] # peri systick start def
[/DA16200/ble] #
<<< APP_PERI_SYSTICK_START_IND
 ... Started

[/DA16200/ble] #
[/DA16200/ble] # peri systick stop
[/DA16200/ble] #
<<< APP_PERI_SYSTICK_STOP_IND
 ... Stopped

[/DA16200/ble] # peri systick start 500000 8
[/DA16200/ble] #
<<< APP_PERI_SYSTICK_START_IND
 ... Started

[/DA16200/ble] #
[/DA16200/ble] # peri systick stop
[/DA16200/ble] #
<<< APP_PERI_SYSTICK_STOP_IND
 ... Stopped

[/DA16200/ble] #
[/DA16200/ble] #
```

Figure 29: Peri Systick

- peri timer0_gen
  - The TIMER0 general example demonstrates how to configure TIMER0 to count a specified amount of time and generate an interrupt. A LED is changing state upon each timer interrupt

    This sample is using 1 GPIO that is connected to an LED to show how TIMER0 can be used
  - DA16600 EVB Configuration:
    - Configuration_1
    - By default, P0_8 is used to connect to LED. Connect J14:P0_8 to any pins in CN4
  - Run command as in Figure 30 and check LED


COM67 - COMBO VT
File Edit Setup Control Window Help
COM68 - COMBO - DA14531 VT
File Edit Setup Control Window Help

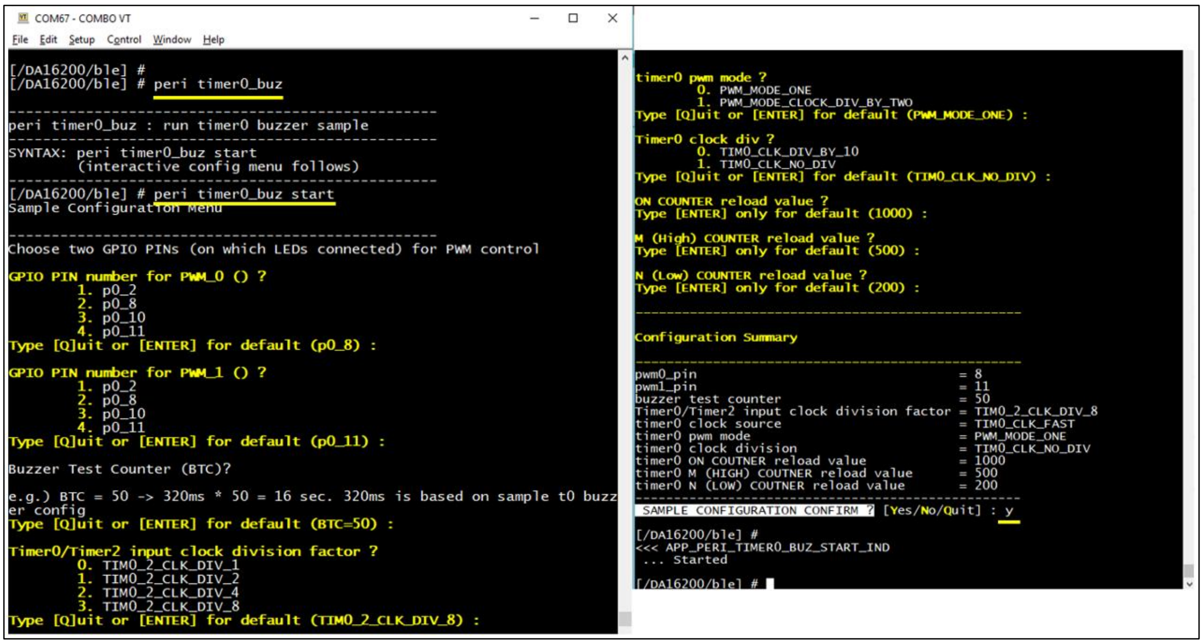
```
[/DA16200/ble] # peri timer0_gen

-------------------------------------------------
peri timer0_gen : run timer0 general sample
-------------------------------------------------
SYNTAX: peri timer0_gen start
        (interactive config menu follows)
-------------------------------------------------
[/DA16200/ble] # peri timer0_gen start
Sample Configuration Menu

-------------------------------------------------

GPIO PIN number () ?
        1. p0_2
        2. p0_8
        3. p0_10
        4. p0_11
Type [Q]uit or [ENTER] for default (p0_8) :

Test duration (in sec) ?
Type [Q]uit or [ENTER] for default (10 sec) :

Timer0/Timer2 input clock division factor ?
        0. TIM0_2_CLK_DIV_1
        1. TIM0_2_CLK_DIV_2
        2. TIM0_2_CLK_DIV_4
        3. TIM0_2_CLK_DIV_8
Type [Q]uit or [ENTER] for default (TIM0_2_CLK_DIV_8) :

timer0 clock source ?
        0. TIM0_CLK_32K
        1. TIM0_CLK_FAST
Type [Q]uit or [ENTER] for default (TIM0_CLK_FAST) :

timer0 pwm mode ?
        0. PWM_MODE_ONE
        1. PWM_MODE_CLOCK_DIV_BY_TWO
Type [Q]uit or [ENTER] for default (PWM_MODE_ONE) :
```

```
Timer0 clock div ?
        0. TIM0_CLK_DIV_BY_10
        1. TIM0_CLK_NO_DIV
Type [Q]uit or [ENTER] for default (TIM0_CLK_DIV_BY_10) :

ON COUNTER reload value ?
: e.g.) reload value for 100ms (T = 1/200kHz * RELOAD_100MS = 0,000005
* 20000 = 100ms)
Type [ENTER] only for default (RELOAD_100MS = 20000) :

-------------------------------------------------

Configuration Summary

-------------------------------------------------
gpio pin                                  = 8
test duration                             = 10 sec
Timer0/Timer2 input clock division factor = TIM0_2_CLK_DIV_8
timer0 clock source                       = TIM0_CLK_FAST
timer0 pwm mode                           = PWM_MODE_ONE
timer0 clock division                     = TIM0_CLK_DIV_BY_10
timer0 ON COUTNER reload value            = 20000
-------------------------------------------------
 SAMPLE CONFIGURATION CONFIRM ? [Yes/No/Quit] : y

[/DA16200/ble] #
<<< APP_PERI_TIMER0_GEN_START_IND
 ... Started

[/DA16200/ble] #
```

```
************************
* TIMER0 GENERAL TEST *
************************

LED will change state every second.

Test will run for: 0A seconds.

TIMER0 started!
TIMER0 stopped!

End of test
```

Figure 30: Peri Timer0_gen

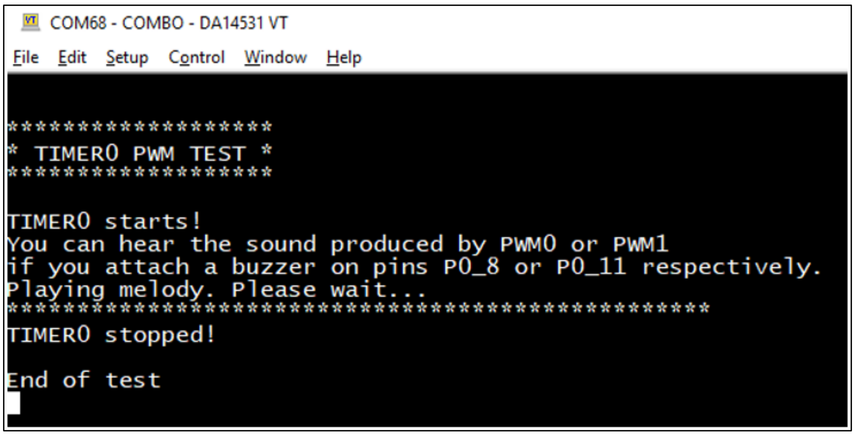
- peri timer0_buz
  - This is TIMER0 (PWM0, PWM1) example that demonstrates how to configure TIMER0 to produce PWM signals. A melody is produced on an externally connected buzzer if connected
  - This sample is using 2 GPIOs that are connected to an external buzzer
  - DA16600 EVB Configuration:
    - Configuration_1
    - By default, P0_8 and P0_11 are used to connect to a buzzer. Connect J14:P0_8, and J14:P0_11 to a buzzer
  - Run command as in Figure 31 and Figure 32



```
[/DA16200/ble] #
[/DA16200/ble] # peri timer0_buz

--------------------------------------------------
peri timer0_buz : run timer0 buzzer sample
--------------------------------------------------
SYNTAX: peri timer0_buz start
        (interactive config menu follows)
--------------------------------------------------
[/DA16200/ble] # peri timer0_buz start
Sample Configuration Menu

--------------------------------------------------
Choose two GPIO PINs (on which LEDs connected) for PWM control

GPIO PIN number for PWM_0 () ?
        1. p0_2
        2. p0_8
        3. p0_10
        4. p0_11
Type [Q]uit or [ENTER] for default (p0_8) :

GPIO PIN number for PWM_1 () ?
        1. p0_2
        2. p0_8
        3. p0_10
        4. p0_11
Type [Q]uit or [ENTER] for default (p0_11) :

Buzzer Test Counter (BTC)?

e.g.) BTC = 50 -> 320ms * 50 = 16 sec. 320ms is based on sample t0 buzz
er config
Type [Q]uit or [ENTER] for default (BTC=50) :

Timer0/Timer2 input clock division factor ?
        0. TIM0_2_CLK_DIV_1
        1. TIM0_2_CLK_DIV_2
        2. TIM0_2_CLK_DIV_4
        3. TIM0_2_CLK_DIV_8
Type [Q]uit or [ENTER] for default (TIM0_2_CLK_DIV_8) :
```

```
timer0 pwm mode ?
        0. PWM_MODE_ONE
        1. PWM_MODE_CLOCK_DIV_BY_TWO
Type [Q]uit or [ENTER] for default (PWM_MODE_ONE) :

Timer0 clock div ?
        0. TIM0_CLK_DIV_BY_10
        1. TIM0_CLK_NO_DIV
Type [Q]uit or [ENTER] for default (TIM0_CLK_NO_DIV) :

ON COUNTER reload value ?
Type [ENTER] only for default (1000) :

M (High) COUNTER reload value ?
Type [ENTER] only for default (500) :

N (Low) COUNTER reload value ?
Type [ENTER] only for default (200) :

--------------------------------------------------

Configuration Summary

--------------------------------------------------
pwm0_pin                                   = 8
pwm1_pin                                   = 11
buzzer test counter                        = 50
Timer0/Timer2 input clock division factor = TIM0_2_CLK_DIV_8
timer0 clock source                        = TIM0_CLK_FAST
timer0 pwm mode                            = PWM_MODE_ONE
timer0 clock division                      = TIM0_CLK_NO_DIV
timer0 ON COUTNER reload value             = 1000
timer0 M (HIGH) COUTNER reload value       = 500
timer0 N (LOW) COUTNER reload value        = 200
--------------------------------------------------
 SAMPLE CONFIGURATION CONFIRM ? [Yes/No/Quit] : y

[/DA16200/ble] #
<<< APP_PERI_TIMER0_BUZ_START_IND
 ... Started

[/DA16200/ble] #
```

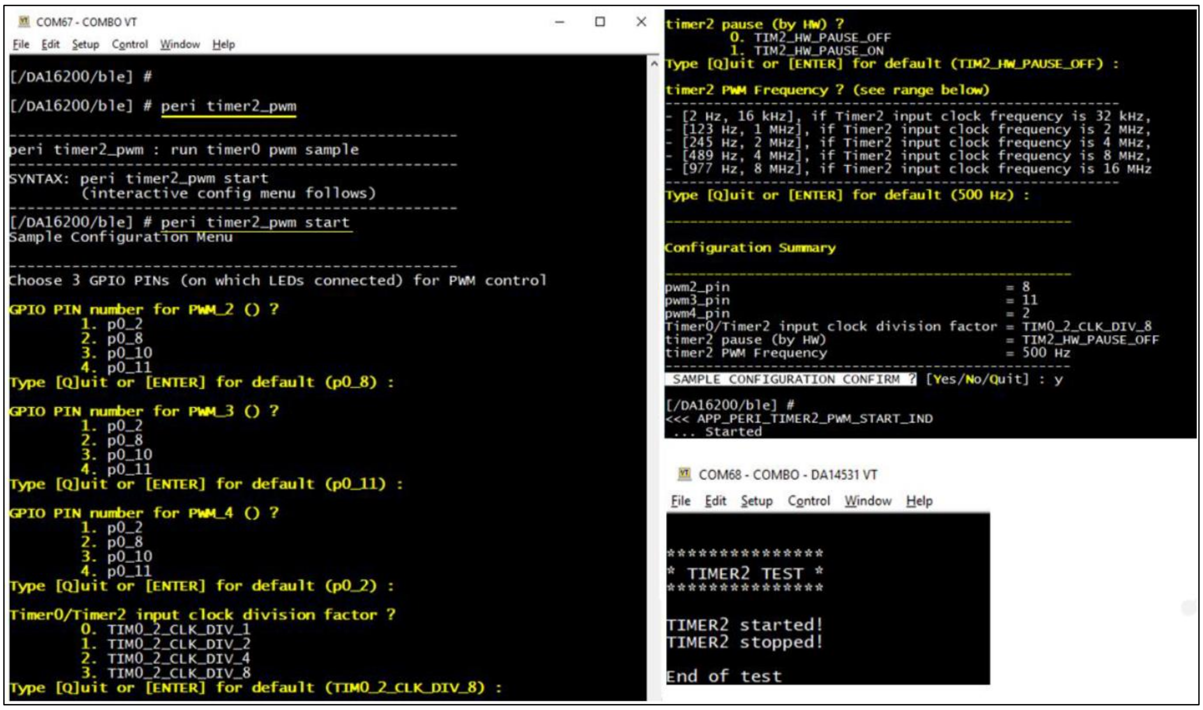
Figure 31: Peri Timer0_buz 1/2


COM68 - COMBO - DA14531 VT
File Edit Setup Control Window Help

```
*******************
* TIMER0 PWM TEST *
*******************

TIMER0 starts!
You can hear the sound produced by PWM0 or PWM1
if you attach a buzzer on pins P0_8 or P0_11 respectively.
Playing melody. Please wait...
**************************************************
TIMER0 stopped!

End of test
```

Figure 32: Peri Timer0_buz 2/2

- peri timer2_pwm
  - The TIMER2 (PWM2, PWM3, PWM4) example demonstrates how to configure TIMER2 to produce PWM signals. The PWM outputs are used to change the brightness of the LEDs in this example
  - This sample is using 3 GPIOs that are connected to an LED segment array to show how TIMER2 PWM can be used
  - DA16600 EVB Configuration:
    - Configuration_1
    - By default, P0_8, P0_11, and P0_2 are used to connect to an LED segment array. Connect J14:P0_8, J14:P0_11, and J2:P0_2 (pin at the bottom left of J2) to a LED segment array
  - Run command as in Figure 33


COM67 - COMBO VT
File Edit Setup Control Window Help
COM68 - COMBO - DA14531 VT
File Edit Setup Control Window Help

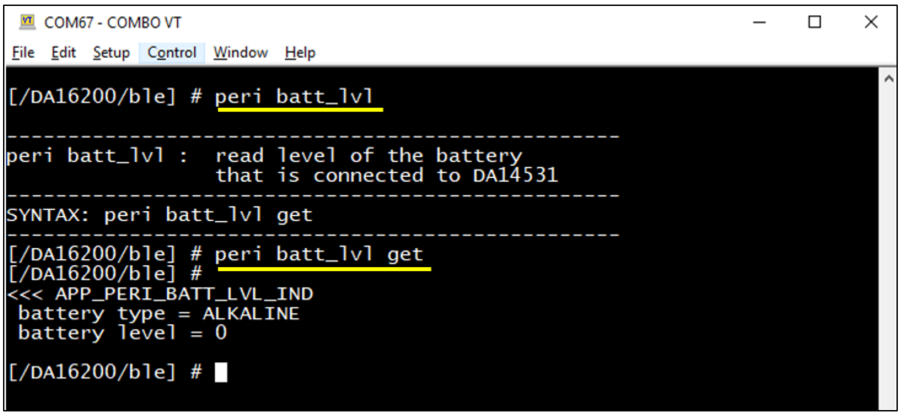
```
[/DA16200/ble] #

[/DA16200/ble] # peri timer2_pwm

-----------------------------------------------------
peri timer2_pwm : run timer0 pwm sample
-----------------------------------------------------
SYNTAX: peri timer2_pwm start
        (interactive config menu follows)
-----------------------------------------------------
[/DA16200/ble] # peri timer2_pwm start
Sample Configuration Menu

-----------------------------------------------------
Choose 3 GPIO PINs (on which LEDs connected) for PWM control

GPIO PIN number for PWM_2 () ?
        1. p0_2
        2. p0_8
        3. p0_10
        4. p0_11
Type [Q]uit or [ENTER] for default (p0_8) :

GPIO PIN number for PWM_3 () ?
        1. p0_2
        2. p0_8
        3. p0_10
        4. p0_11
Type [Q]uit or [ENTER] for default (p0_11) :

GPIO PIN number for PWM_4 () ?
        1. p0_2
        2. p0_8
        3. p0_10
        4. p0_11
Type [Q]uit or [ENTER] for default (p0_2) :

Timer0/Timer2 input clock division factor ?
        0. TIM0_2_CLK_DIV_1
        1. TIM0_2_CLK_DIV_2
        2. TIM0_2_CLK_DIV_4
        3. TIM0_2_CLK_DIV_8
Type [Q]uit or [ENTER] for default (TIM0_2_CLK_DIV_8) :
```

```
timer2 pause (by HW) ?
        0. TIM2_HW_PAUSE_OFF
        1. TIM2_HW_PAUSE_ON
Type [Q]uit or [ENTER] for default (TIM2_HW_PAUSE_OFF) :

timer2 PWM Frequency ? (see range below)
-----------------------------------------------------------
- [2 Hz, 16 kHz], if Timer2 input clock frequency is 32 kHz,
- [123 Hz, 1 MHz], if Timer2 input clock frequency is 2 MHz,
- [245 Hz, 2 MHz], if Timer2 input clock frequency is 4 MHz,
- [489 Hz, 4 MHz], if Timer2 input clock frequency is 8 MHz,
- [977 Hz, 8 MHz], if Timer2 input clock frequency is 16 MHz
-----------------------------------------------------------
Type [Q]uit or [ENTER] for default (500 Hz) :

-----------------------------------------------------

Configuration Summary

-----------------------------------------------------
pwm2_pin                                    = 8
pwm3_pin                                    = 11
pwm4_pin                                    = 2
Timer0/Timer2 input clock division factor = TIM0_2_CLK_DIV_8
timer2 pause (by HW)                        = TIM2_HW_PAUSE_OFF
timer2 PWM Frequency                        = 500 Hz
-----------------------------------------------------
 SAMPLE CONFIGURATION CONFIRM ? [Yes/No/Quit] : y

[/DA16200/ble] #
<<< APP_PERI_TIMER2_PWM_START_IND
 ... Started
```

```
***************
* TIMER2 TEST *
***************

TIMER2 started!
TIMER2 stopped!

End of test
```

Figure 33: Peri Timer2_pwm

- peri batt_lvl
  - ○ The Battery example demonstrates how to read the level of battery that is connected to DA14531
  - ○ DA16600 EVB Configuration:
    - – Configuration_1
  - ○ Run command as in Figure 34



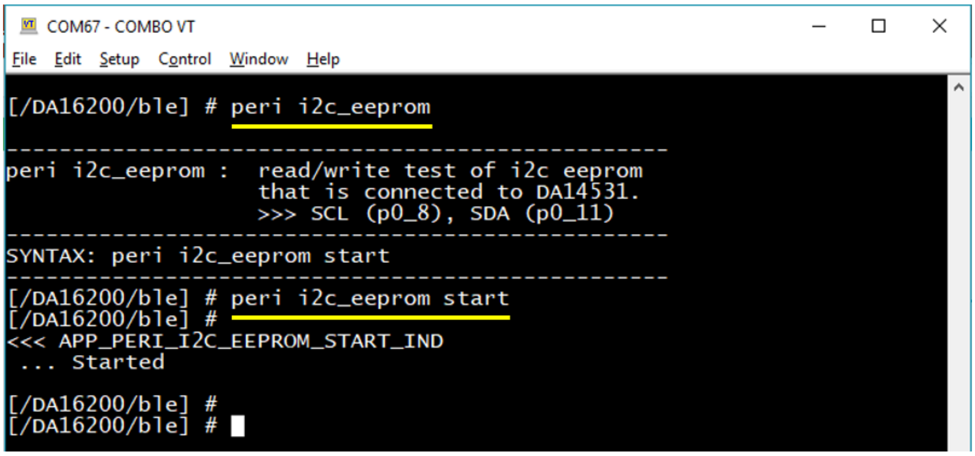
```
[/DA16200/ble] # peri batt_lvl

-----------------------------------------------------
peri batt_lvl :  read level of the battery
                 that is connected to DA14531
-----------------------------------------------------
SYNTAX: peri batt_lvl get
-----------------------------------------------------
[/DA16200/ble] # peri batt_lvl get
[/DA16200/ble] #
<<< APP_PERI_BATT_LVL_IND
 battery type = ALKALINE
 battery level = 0

[/DA16200/ble] # █
```

Figure 34: Peri Batt_lvl

- peri i2c_eeprom
  - ○ The I2C EEPROM example demonstrates how to initiate, read, write, and erase an I2C EEPROM memory. This example works if the user connects an external memory
  - ○ DA16600 EVB Configuration:
    - – Configuration_1

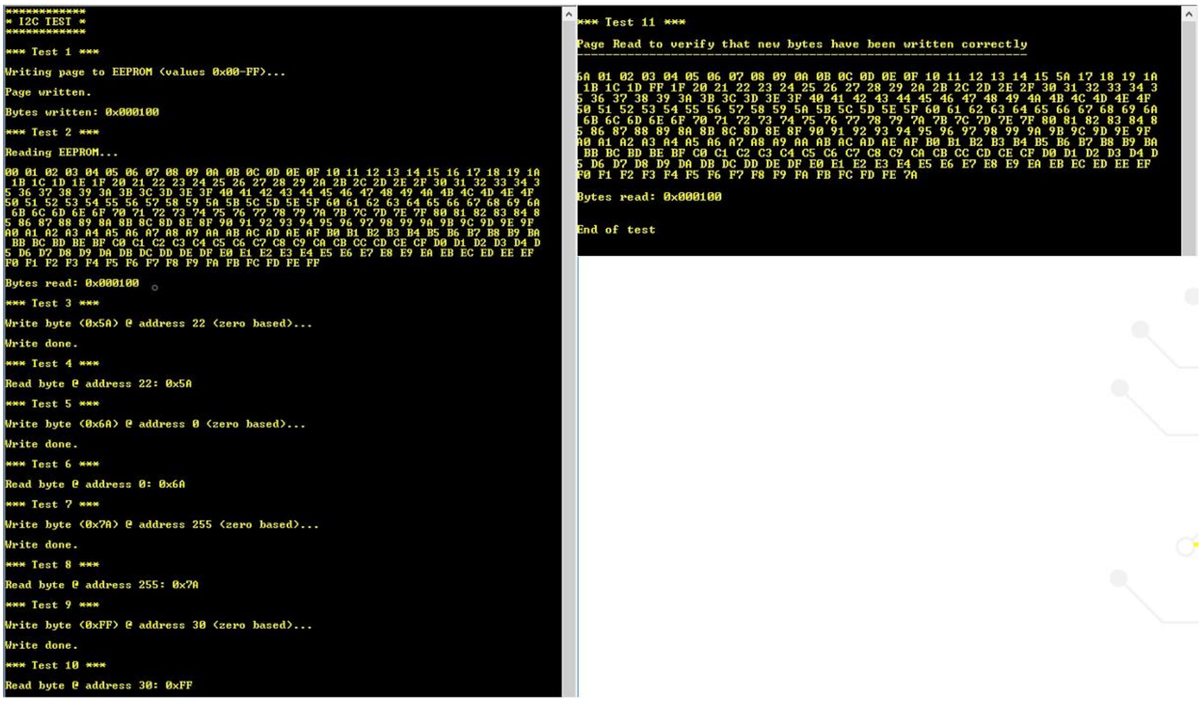
- By default, P0_8 (SCL) and P0_11 (SDA) are used. Connect J14:P0_8, J14:P0_11 to an external I2C_EEPROM

- Run command as in Figure 35 and Figure 36



```
[/DA16200/ble] # peri i2c_eeprom

-------------------------------------------------
peri i2c_eeprom :  read/write test of i2c eeprom
                   that is connected to DA14531.
                   >>> SCL (p0_8), SDA (p0_11)
-------------------------------------------------
SYNTAX: peri i2c_eeprom start
-------------------------------------------------
[/DA16200/ble] # peri i2c_eeprom start
[/DA16200/ble] #
<<< APP_PERI_I2C_EEPROM_START_IND
 ... Started

[/DA16200/ble] #
[/DA16200/ble] #
```

Figure 35: Peri I2c_eeprom



```
*************
* I2C TEST *
*************

*** Test 1 ***

Writing page to EEPROM (values 0x00-FF)...

Page written.

Bytes written: 0x000100

*** Test 2 ***

Reading EEPROM...

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A
 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 3
5 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68 69 6A
 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 8
5 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA
 BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D
5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF

Bytes read: 0x000100

*** Test 3 ***

Write byte (0x5A) @ address 22 (zero based)...

Write done.

*** Test 4 ***

Read byte @ address 22: 0x5A

*** Test 5 ***

Write byte (0x6A) @ address 0 (zero based)...

Write done.

*** Test 6 ***

Read byte @ address 0: 0x6A

*** Test 7 ***

Write byte (0x7A) @ address 255 (zero based)...

Write done.

*** Test 8 ***

Read byte @ address 255: 0x7A

*** Test 9 ***

Write byte (0xFF) @ address 30 (zero based)...

Write done.

*** Test 10 ***

Read byte @ address 30: 0xFF
```

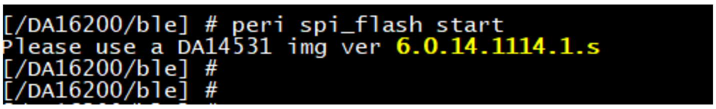
```
*** Test 11 ***

Page Read to verify that new bytes have been written correctly
--------------------------------------------------------------

6A 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 5A 17 18 19 1A
 1B 1C 1D FF 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 3
5 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68 69 6A
 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 8
5 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA
 BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D
5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE 7A

Bytes read: 0x000100


End of test
```

Figure 36: Peri I2c_eeprom Read/Write

- peri spi_flash
  - The SPI Flash memory example demonstrates how to initiate, read, write, and erase an SPI Flash memory with the SPI Flash driver
  - The following is the pre-defined characteristics configured in the DA14531 image:
    - #define SPI_MS_MODE   SPI_MS_MODE_MASTER
    - #define SPI_CP_MODE   SPI_CP_MODE_0
    - #define SPI_WSZ   SPI_MODE_8BIT

  - #define SPI_CS   SPI_CS_0
  - #define SPI_FLASH_DEV_SIZE   (256 * 1024)
- DA16600 EVB Configuration:
  - Configuration_3
  - By default, 4 GPIO pins are used; SPI_EN (J14: p0_8), SPI_CLK (J14:p0_11), SPI_DO (p0_2: pin at J2 Left-bottom), SPI_DI (p0_10: J2 Right-bottom)
- DA14531 image used:
  - Download the following image for this test

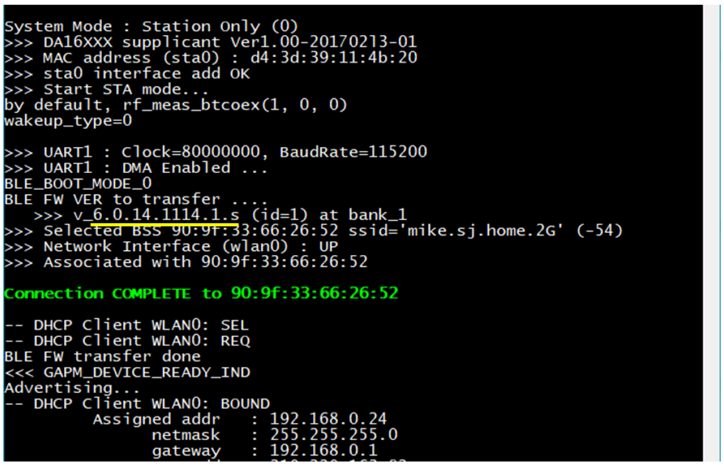
    ```
    [DA16600_SDK_ROOT]\img\DA14531_1\peri_spi_flash\
       da14531_multi_part_proxr_s.img
    ```
  - If you do not use the image above, you will get the message as in Figure 37



```
[/DA16200/ble] # peri spi_flash start
Please use a DA14531 img ver 6.0.14.1114.1.s
[/DA16200/ble] #
[/DA16200/ble] #
```

Figure 37: Peri Spi_flash – Wrong Image Warning

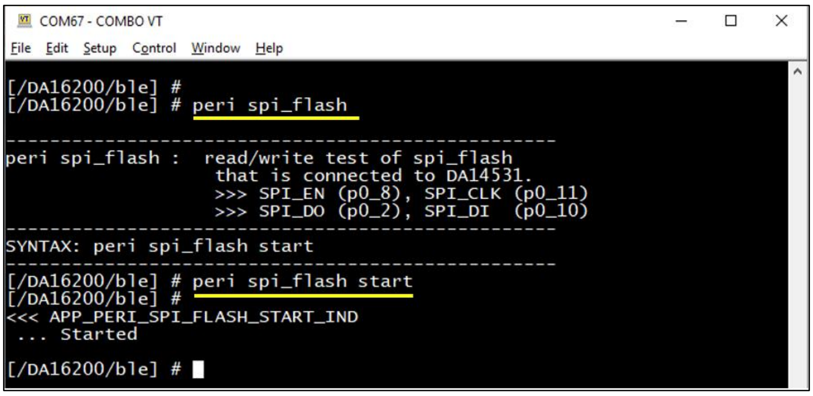
  - After booting with a correct Bluetooth® LE test image, you can find the following version string printed at boot as in Figure 38



```
System Mode : Station Only (0)
>>> DA16XXX supplicant Ver1.00-20170213-01
>>> MAC address (sta0) : d4:3d:39:11:4b:20
>>> sta0 interface add OK
>>> Start STA mode...
by default, rf_meas_btcoex(1, 0, 0)
wakeup_type=0

>>> UART1 : Clock=80000000, BaudRate=115200
>>> UART1 : DMA Enabled ...
BLE_BOOT_MODE_0
BLE FW VER to transfer ....
   >>> v_6.0.14.1114.1.s (id=1) at bank_1
>>> Selected BSS 90:9f:33:66:26:52 ssid='mike.sj.home.2G' (-54)
>>> Network Interface (wlan0) : UP
>>> Associated with 90:9f:33:66:26:52

Connection COMPLETE to 90:9f:33:66:26:52

-- DHCP Client WLAN0: SEL
-- DHCP Client WLAN0: REQ
BLE FW transfer done
<<< GAPM_DEVICE_READY_IND
Advertising...
-- DHCP Client WLAN0: BOUND
         Assigned addr   : 192.168.0.24
               netmask   : 255.255.255.0
               gateway   : 192.168.0.1
```

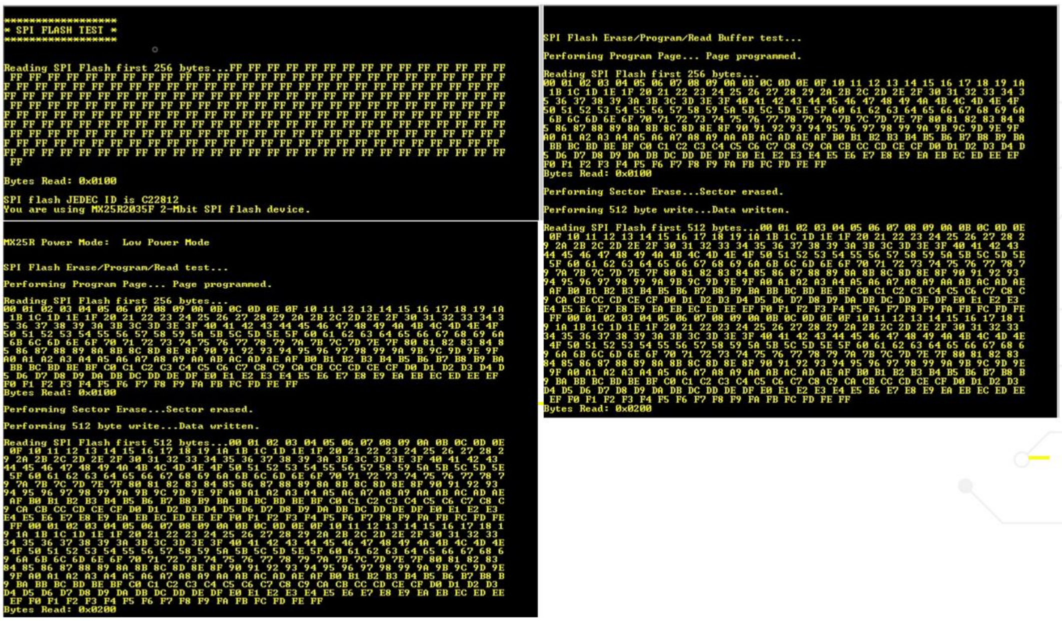
Figure 38: Correct Image Version for Peri Spi_flash Sample

- Run command as in Figure 39 and Figure 40



```
[/DA16200/ble] #
[/DA16200/ble] # peri spi_flash

-------------------------------------------------------------
peri spi_flash :  read/write test of spi_flash
                   that is connected to DA14531.
                   >>> SPI_EN (p0_8), SPI_CLK (p0_11)
                   >>> SPI_DO (p0_2), SPI_DI  (p0_10)
-------------------------------------------------------------
SYNTAX: peri spi_flash start
-------------------------------------------------------------
[/DA16200/ble] # peri spi_flash start
[/DA16200/ble] #
<<< APP_PERI_SPI_FLASH_START_IND
 ... Started

[/DA16200/ble] #
```

Figure 39: Peri Spi_flash



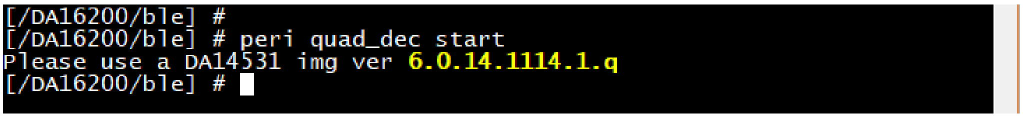
```
******************
* SPI FLASH TEST *
******************

Reading SPI Flash first 256 bytes...FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF F
F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF F
F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF F
F FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
 FF

Bytes Read: 0x0100

SPI flash JEDEC ID is C22812
You are using MX25R2035F 2-Mbit SPI flash device.
```

```
MX25R Power Mode:  Low Power Mode

SPI Flash Erase/Program/Read test...

Performing Program Page... Page programmed.

Reading SPI Flash first 256 bytes...
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A
 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 3
5 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68 69 6A
 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 8
5 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA
 BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D
5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
Bytes Read: 0x0100

Performing Sector Erase...Sector erased.

Performing 512 byte write...Data written.

Reading SPI Flash first 512 bytes...00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E
 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 2
9 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43
44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E
 5F 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 7
9 7A 7B 7C 7D 7E 7F 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93
94 95 96 97 98 99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE
 AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C
9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3
E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE
 FF 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 1
9 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33
34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E
 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68 6
9 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 81 82 83
84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E
 9F A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B
9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3
D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE
 EF F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
Bytes Read: 0x0200
```

```
SPI Flash Erase/Program/Read Buffer test...

Performing Program Page... Page programmed.

Reading SPI Flash first 256 bytes...
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A
 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 3
5 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68 69 6A
 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 8
5 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA
 BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D
5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
Bytes Read: 0x0100

Performing Sector Erase...Sector erased.

Performing 512 byte write...Data written.

Reading SPI Flash first 512 bytes...00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E
 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 2
9 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43
44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E
 5F 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 7
9 7A 7B 7C 7D 7E 7F 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93
94 95 96 97 98 99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE
 AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C
9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3
E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE
 FF 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 1
9 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33
34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E
 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68 6
9 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 81 82 83
84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E
 9F A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B
9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3
D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE
 EF F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
Bytes Read: 0x0200
```

Figure 40: Peri Spi_flash Read/Write

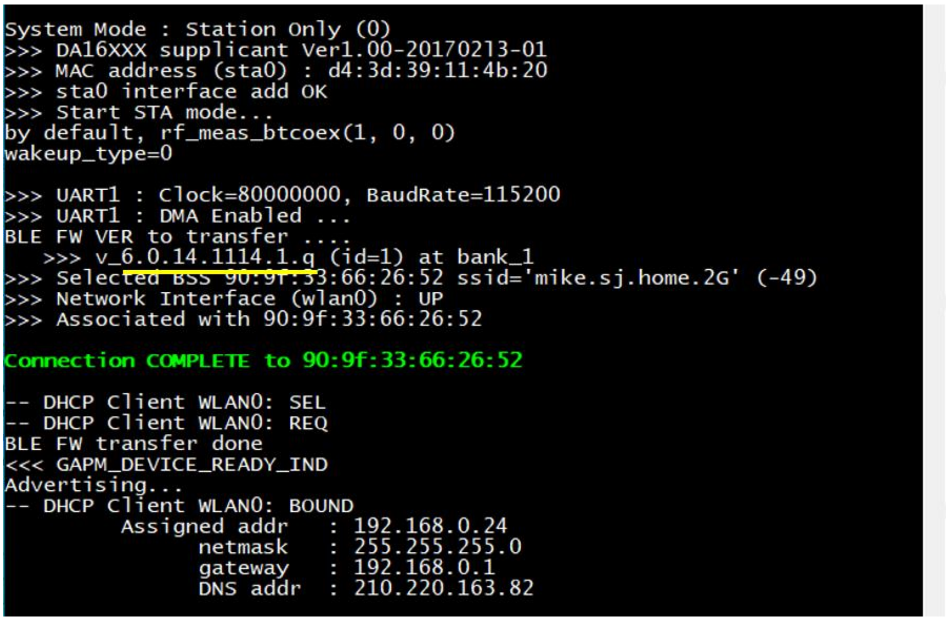
- peri quad_dec
  - ○ The Quadrature decoder example demonstrates how to configure and read from the quadrature decoder peripheral. This example works if the user connects an external quad encoder device
  - ○ DA16600 EVB Configuration:
    - – Configuration_2
    - – By default, 2 GPIO pins are used; CHX_A (J14: p0_8), CHX_B (J14: p0_9)
  - ○ DA14531 image used:
    - – Download the image below for this test
      `[DA16600_SDK_ROOT]\img\DA14531_1\peri_quad_dec\da14531_multi_part_proxr_q.img`
    - – If you do not use the image above, you will get the message as in Figure 41



```
[/DA16200/ble] #
[/DA16200/ble] # peri quad_dec start
Please use a DA14531 img ver 6.0.14.1114.1.q
[/DA16200/ble] #
```

Figure 41: Peri Quad_dec – Wrong Image Warning

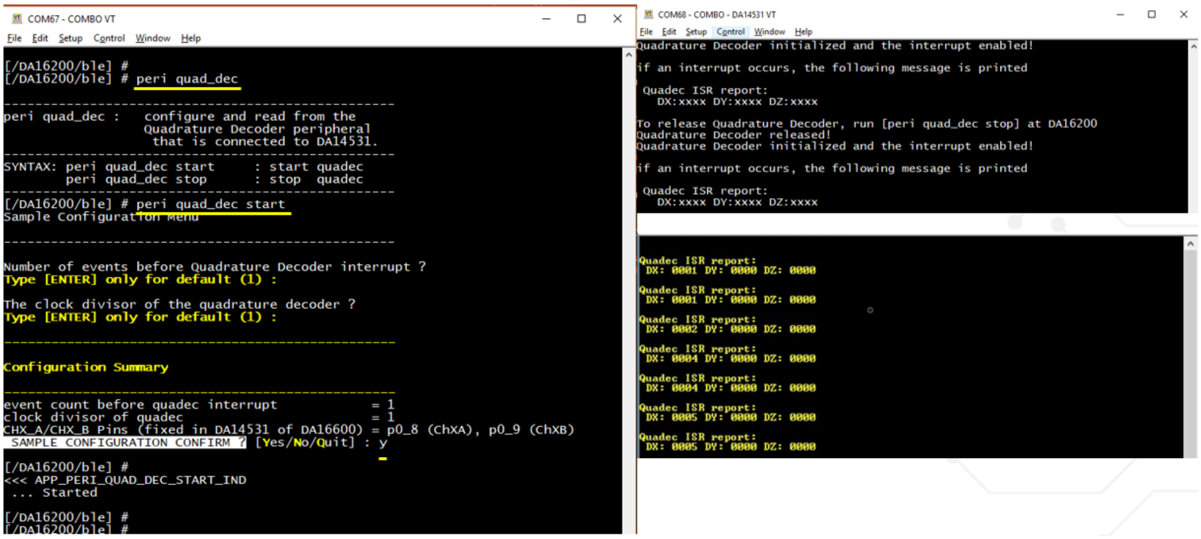
– After booting with a correct Bluetooth® LE test image, you can find the following version string printed at boot as in Figure 42



```
System Mode : Station Only (0)
>>> DA16XXX supplicant Ver1.00-20170213-01
>>> MAC address (sta0) : d4:3d:39:11:4b:20
>>> sta0 interface add OK
>>> Start STA mode...
by default, rf_meas_btcoex(1, 0, 0)
wakeup_type=0

>>> UART1 : Clock=80000000, BaudRate=115200
>>> UART1 : DMA Enabled ...
BLE FW VER to transfer ....
   >>> v_6.0.14.1114.1.q (id=1) at bank_1
>>> Selected BSS 90:9f:33:66:26:52 ssid='mike.sj.home.2G' (-49)
>>> Network Interface (wlan0) : UP
>>> Associated with 90:9f:33:66:26:52

Connection COMPLETE to 90:9f:33:66:26:52

-- DHCP Client WLAN0: SEL
-- DHCP Client WLAN0: REQ
BLE FW transfer done
<<< GAPM_DEVICE_READY_IND
Advertising...
-- DHCP Client WLAN0: BOUND
         Assigned addr   : 192.168.0.24
               netmask   : 255.255.255.0
               gateway   : 192.168.0.1
               DNS addr  : 210.220.163.82
```

Figure 42: Correct Image Version for Peri Quad_dec Sample

○ Run command as in Figure 43 and Figure 44



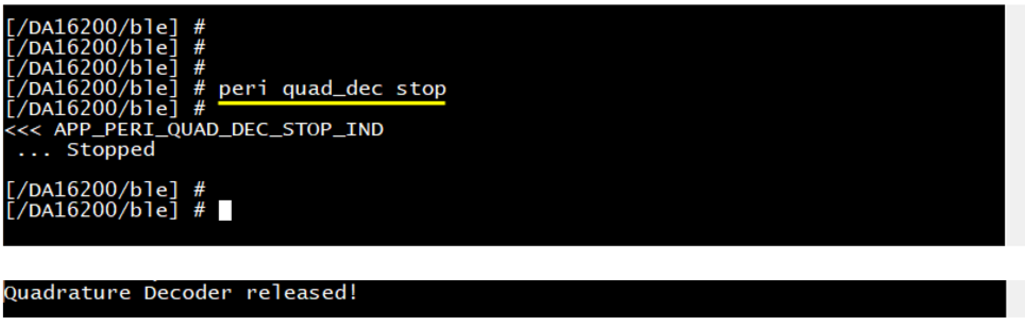
```
[/DA16200/ble] #
[/DA16200/ble] # peri quad_dec

-------------------------------------------------
peri quad_dec :   configure and read from the
                  Quadrature Decoder peripheral
                   that is connected to DA14531.
-------------------------------------------------
SYNTAX: peri quad_dec start     : start quadec
        peri quad_dec stop      : stop  quadec
-------------------------------------------------
[/DA16200/ble] # peri quad_dec start
Sample Configuration Menu

-------------------------------------------------

Number of events before Quadrature Decoder interrupt ?
Type [ENTER] only for default (1) :

The clock divisor of the quadrature decoder ?
Type [ENTER] only for default (1) :

-------------------------------------------------

Configuration Summary

-------------------------------------------------
event count before quadec interrupt             = 1
clock divisor of quadec                         = 1
CHX_A/CHX_B Pins (fixed in DA14531 of DA16600) = p0_8 (ChXA), p0_9 (ChXB)
 SAMPLE CONFIGURATION CONFIRM ? [Yes/No/Quit] : y

[/DA16200/ble] #
<<< APP_PERI_QUAD_DEC_START_IND
 ... Started

[/DA16200/ble] #
[/DA16200/ble] #
```

COM68 - COMBO - DA14531 VT
File Edit Setup Control Window Help

```
Quadrature Decoder initialized and the interrupt enabled!

if an interrupt occurs, the following message is printed

 Quadec ISR report:
   DX:xxxx DY:xxxx DZ:xxxx

To release Quadrature Decoder, run [peri quad_dec stop] at DA16200
Quadrature Decoder released!
Quadrature Decoder initialized and the interrupt enabled!

if an interrupt occurs, the following message is printed

 Quadec ISR report:
   DX:xxxx DY:xxxx DZ:xxxx
```

```
Quadec ISR report:
 DX: 0001 DY: 0000 DZ: 0000

Quadec ISR report:
 DX: 0001 DY: 0000 DZ: 0000

Quadec ISR report:
 DX: 0002 DY: 0000 DZ: 0000

Quadec ISR report:
 DX: 0004 DY: 0000 DZ: 0000

Quadec ISR report:
 DX: 0004 DY: 0000 DZ: 0000

Quadec ISR report:
 DX: 0005 DY: 0000 DZ: 0000

Quadec ISR report:
 DX: 0005 DY: 0000 DZ: 0000
```

Figure 43: Peri Quad_dec – Start



```
[/DA16200/ble] #
[/DA16200/ble] #
[/DA16200/ble] #
[/DA16200/ble] # peri quad_dec stop
[/DA16200/ble] #
<<< APP_PERI_QUAD_DEC_STOP_IND
 ... Stopped

[/DA16200/ble] #
[/DA16200/ble] #
```

```
Quadrature Decoder released!
```

Figure 44: Peri Quad_dec – Stop

After all tests, revert to Configuration_1 for the next sample test.

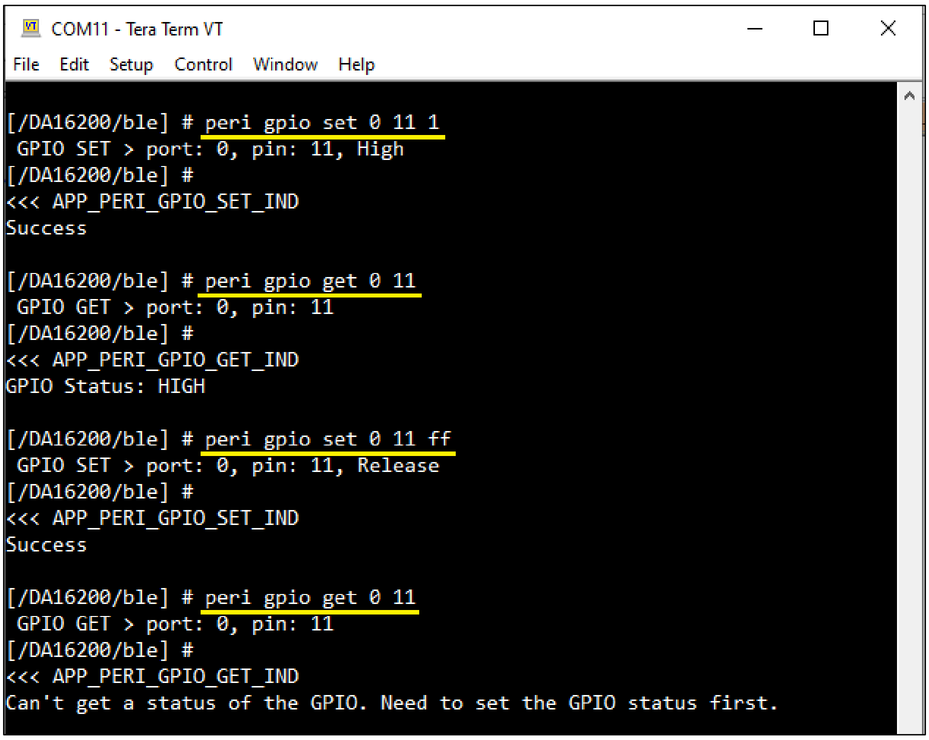
- peri gpio
  - ○ The GPIO example demonstrates how to set/get the state of a GPIO of DA14531. If you set the state of a GPIO of DA14531 to either HIGH or LOW, the state will be kept although DA14531 is in sleep. If you do not want to control the GPIO state of DA14531 anymore, then you need to set the GPIO to 0xFF
  - ○ DA16600 EVB Configuration:
    - – Configuration_1
    - – See the Section 6.4.4.3 to check available GPIOs in DA14531
  - ○ DA14531 image used:
    - – Download the image below for this test.
    - – `~SDK\apps\da16600\get_started\img\DA14531_1\ da14531_multi_part_proxr.img`.
  - ○ Run command as below:
    - – SYNTAX: SET

      – peri gpio set port_no pin_no 1 | 0 | FF : 1(High), 0(Low), FF(Release)

      – ex) peri gpio set 0 8 1 ; Set GPIO0_8 to High
    - – SYNTAX: GET

      – peri gpio get port_no pin_no

      – ex) peri gpio get 0 8 ; Get GPIO0_8 status



```
[/DA16200/ble] # peri gpio set 0 11 1
 GPIO SET > port: 0, pin: 11, High
[/DA16200/ble] #
<<< APP_PERI_GPIO_SET_IND
Success

[/DA16200/ble] # peri gpio get 0 11
 GPIO GET > port: 0, pin: 11
[/DA16200/ble] #
<<< APP_PERI_GPIO_GET_IND
GPIO Status: HIGH

[/DA16200/ble] # peri gpio set 0 11 ff
 GPIO SET > port: 0, pin: 11, Release
[/DA16200/ble] #
<<< APP_PERI_GPIO_SET_IND
Success

[/DA16200/ble] # peri gpio get 0 11
 GPIO GET > port: 0, pin: 11
[/DA16200/ble] #
<<< APP_PERI_GPIO_GET_IND
Can't get a status of the GPIO. Need to set the GPIO status first.
```

Figure 45: Peri GPIO Set/Get

5.6 Example 5: TCP DPM Client

5.6.1 Test Environment Setup

- Wi-Fi Router: MyAP
- TCP Client
  - H/W: **DA16600_EVB_1**
  - S/W: program SFLASH with **DA16200_SW_3** + **da14531_multi_part_proxr.img**
- Two Tera Term windows: **da16_tera_win**, and **da14_tera_win**
- TCP Server: any TCP Server utility is OK, for example, IONINJA, or an Android/iOS TCP network tool

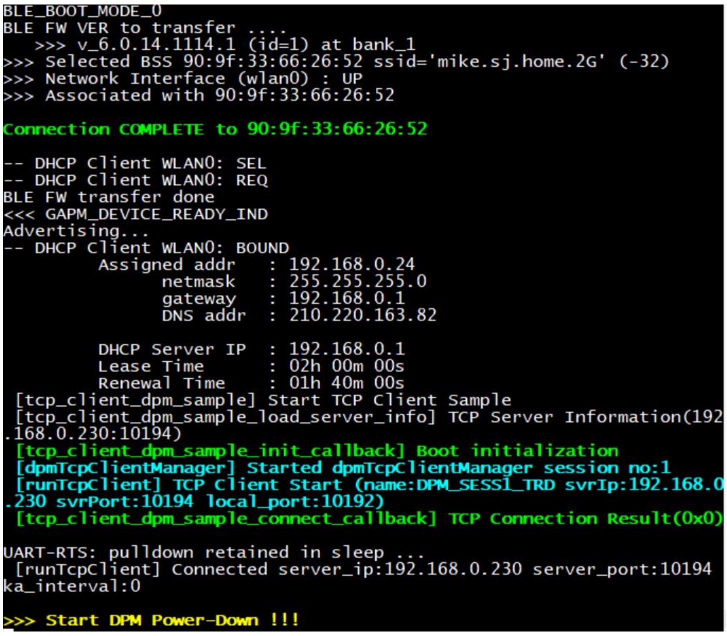
5.6.2 Test Steps

- TCP Server machine (Windows utility/mobile application)
  - Connect to MyAP (either through a wired port or Wi-Fi port – wired connection preferred)
  - Run TCP Server tool (with the port number set to 10194)
  - Take note of TCP Server information: IP = 192.168.0.230, Port = 10194
- TCP Client
  - da16_tera_win
    - type "factory" > type "y"
    - Run Wi-Fi Provisioning to connect to MyAP. See Section 5.2
    - type
      nvram.setenv TCPC_SERVER_IP 192.168.0.230

    nvram.setenv TCPC_SERVER_PORT 10194

- type "dpm on"

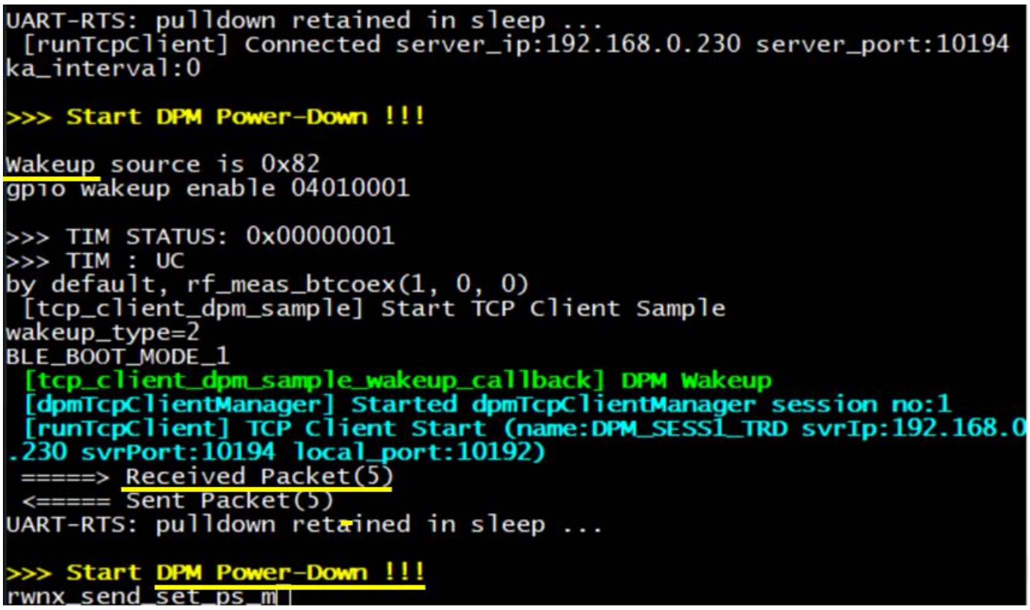
DA16600_EVB_1 is rebooted and enters DPM Sleep as in Figure 46.



```
BLE_BOOT_MODE_0
BLE FW VER to transfer ....
   >>> v_6.0.14.1114.1 (id=1) at bank_1
>>> Selected BSS 90:9f:33:66:26:52 ssid='mike.sj.home.2G' (-32)
>>> Network Interface (wlan0) : UP
>>> Associated with 90:9f:33:66:26:52

Connection COMPLETE to 90:9f:33:66:26:52

-- DHCP Client WLAN0: SEL
-- DHCP Client WLAN0: REQ
BLE FW transfer done
<<< GAPM_DEVICE_READY_IND
Advertising...
-- DHCP Client WLAN0: BOUND
         Assigned addr   : 192.168.0.24
               netmask   : 255.255.255.0
               gateway   : 192.168.0.1
               DNS addr  : 210.220.163.82

         DHCP Server IP  : 192.168.0.1
         Lease Time      : 02h 00m 00s
         Renewal Time    : 01h 40m 00s
 [tcp_client_dpm_sample] Start TCP Client Sample
 [tcp_client_dpm_sample_load_server_info] TCP Server Information(192
.168.0.230:10194)
 [tcp_client_dpm_sample_init_callback] Boot initialization
 [dpmTcpClientManager] Started dpmTcpClientManager session no:1
 [runTcpClient] TCP Client Start (name:DPM_SESS1_TRD svrIp:192.168.0
.230 svrPort:10194 local_port:10192)
 [tcp_client_dpm_sample_connect_callback] TCP Connection Result(0x0)

UART-RTS: pulldown retained in sleep ...
 [runTcpClient] Connected server_ip:192.168.0.230 server_port:10194
ka_interval:0

>>> Start DPM Power-Down !!!
```

Figure 46: TCP Client in DPM Sleep

- TCP Server Tool
    - Send a text to TCP Client
- TCP Client
    - TCP Client wakes up, receives, processes data, and enters sleep



```
UART-RTS: pulldown retained in sleep ...
 [runTcpClient] Connected server_ip:192.168.0.230 server_port:10194
ka_interval:0

>>> Start DPM Power-Down !!!

Wakeup source is 0x82
gpio wakeup enable 04010001

>>> TIM STATUS: 0x00000001
>>> TIM : UC
by default, rf_meas_btcoex(1, 0, 0)
 [tcp_client_dpm_sample] Start TCP Client Sample
wakeup_type=2
BLE_BOOT_MODE_1
 [tcp_client_dpm_sample_wakeup_callback] DPM Wakeup
 [dpmTcpClientManager] Started dpmTcpClientManager session no:1
 [runTcpClient] TCP Client Start (name:DPM_SESS1_TRD svrIp:192.168.0
.230 svrPort:10194 local_port:10192)
 =====> Received Packet(5)
 <===== Sent Packet(5)
UART-RTS: pulldown retained in sleep ...

>>> Start DPM Power-Down !!!
rwnx_send_set_ps_m
```

Figure 47: TCP Client – Wakeup from DPM Sleep

- Wi-Fi Provisioning while DA16600 is in sleep mode
  - Wi-Fi Provisioning APP: Start > Connect to "DA16600" > DA16600 wakes up and connects to Wi-Fi Provisioning, then do provision to an AP if needed

5.7 Example 6: IoT Sensor Gateway

- ssh_win_2 (RBP3_1)
  - root@raspberrypi:/home/pi# **sh ./run-iot-sensor.sh**
  - >> sensor_1 starts running
- ssh_win_3 (RBP3_2)
  - root@raspberrypi:/home/pi# **sh ./run-iot-sensor.sh**
  - >> sensor_2 starts running

NOTE
With the above you started two Bluetooth® LE temperature sensors (RBP3_1 and RBP3_2). Two sensors (sensor_1 and sensor_2) are now advertising, which means waiting to be connected by a GAP Central which is DA16600_EVB_2.

- Instead of the RBP3_1 and RBP3_2, extra two DA16600 boards – **DA16600_EVB_1** that programed DA16200_SW_1 and DA14531_IMG (Prj1_proxr) – can be used for this test as IoT sensor devices as well.

  This is described in Section 5.1.4, the images are used for Example 1,2, or 3.
- da16_tera_win
  - Power off DA16600_EVB_1 and power ON DA16600_EVB_2
  - Connect Tera Term to DA16600_EVB_2 (sensor_gateway)
  - By default, the sensor gateway is running as "Bluetooth® LE GAP Central" that scans neighbor GAP Peripheral devices
  - Once the scan is finished, enter "Provisioning" mode as shown below
    NOTE: wait until *">>> Please connect or rescan. Type in …."* is fully displayed before you do this action

```
// User command in bold font, commentary / messages to note in blue font
…

          #####################################################
          # DA14531 Proxm Sensor Gateway demo application    #
          #####################################################

#  No.  bd_addr                 Name                    Rssi            #
#  1    d9:6c:3f:82:7b:1c       Mi Band 3               -61 dB          #
#  2    b8:27:eb:e8:8e:24       sensor_1                -56 dB          #
 >>> Please connect or rescan. Type in "[/DA16xxx/ble] # proxm_sensor_gw" for cmd
options // LE scan finished

[/DA16200] #
[/DA16200] # ble
    Command-List is changed, "ble"
[/DA16200/ble] # proxm_sensor_gw provision_mode
[/DA16200/ble] # [ConsoleEvent] user command exists in queue
CONSOLE_ENABLE_WFSVC_CMD
[ConsoleEvent] user command handled
Advertising... // starts Bluetooth® LE advertising …
```

- ○ Now DA16600_EVB_2 (sensor_gateway) is in advertising mode (GAP Peripheral). Follow instructions from Section 5.2 to do provisioning
- ○ When DA16200 is provisioned, DA16600_EVB_2 is rebooted and gets back to GAP Central mode, and LE Scanning starts automatically

- ● Gateway device in Bluetooth® LE scanning

```
// User command in bold font, commentary / messages to note in blue font

...
          #####################################################
          #    DA14585 Proximity Monitor demo application    #
          #####################################################

#  No.  bd_addr                 Name                    Rssi            #
#  1    b8:27:eb:e8:8e:24       sensor_1                -48 dB          #
#  2    b8:27:eb:c6:be:74       sensor_2                -42 dB          #
#  3    ec:b3:07:60:6a:a4       Mi Smart Band 4         -62 dB          #
#  4    80:ea:ca:80:00:01       Dialog SOC Demo         -58 dB          #
Scanning... Please waitGAPM_ADV_REPORT_IND
 >>> Please connect or rescan. Type in "[/DA16xxx/ble] # proxm_sensor_gw" for cmd
options

[/DA16200/] # ble
    Command-List is changed, "ble"
[/DA16200/ble] #
[/DA16200/ble] # proxm_sensor_gw
Usage: BLE PROXM & Sensor Gateway Sample Application command
Name
        proxm_sensor_gw - Proximity Monitor and Sensor GW cmd
SYNOPSIS
        proxm_sensor_gw [OPTION]
OPTION DESCRIPTION
         scan
                 Scan BLE peers around
         show_conn_dev
                 shows connected BLE peers with status
         rd_rssi_conn_dev
                 read rssi for all connected devices
         read_temp
                 read temperature sensor values from all connected devices
         conn [1~9]
                 connect to a ble peer from the scan list
                 choose index from the scan list
         peer [1~9] [A|B|...|Z]
                 take an action on a connected BLE peer
                 ------------proxm cmd ----------------
                 A: Read Link Loss Alert Level
                 B: Read Tx Power Level
                 C: Start High Level Immediate Alert
                 D: Start Mild Level Immediate Alert
                 E: Stop Immediate Alert
                 F: Set Link Loss Alert Level to None
                 G: Set Link Loss Alert Level to Mild
                 H: Set Link Loss Alert Level to High
                 I: Show device info
                 ------------custom cmd ----------------
                 J: Enable iot sensor's temperature posting
                 K: Disable iot sensor's temperature posting
                 ------------common cmd ----------------
                 Z: Disconnect from the device
         exit
                 all peers are disconnected
[/DA16200/ble] # proxm_sensor_gw conn 1
...
[/DA16200/ble] # proxm_sensor_gw conn 2
...

#########################################################################
#########################   Connected Devices   #########################
#########################################################################

#  No. Model No.        BDA                Bonded  RSSI   LLA  TX_PL  Temp
#  * 1 iot_sensor       b8:27:eb:c6:be:74   NO

#    2 iot_sensor       b8:27:eb:e8:8e:24   NO

#    3                --   Empty Slot --
#
#########################################################################


[/DA16200/ble] # proxm_sensor_gw peer 1 J
...

[/DA16200/ble] # proxm_sensor_gw peer 2 J
...
#########################################################################
#########################   Connected Devices   #########################
#########################################################################

#  No. Model No.        BDA                Bonded  RSSI   LLA  TX_PL  Temp
#  * 1 iot_sensor       b8:27:eb:c6:be:74   NO                          35

#    2 iot_sensor       b8:27:eb:e8:8e:24   NO                          13

#    3                --   Empty Slot --
#
#########################################################################
```

NOTE
Depending on your RF signal environment, sensor_1 or sensor_2 may not easily get connected. In such a case, run the scan again and try to connect.

- ssh_win_2

```
...
Registering GATT application...
GetManagedObjects
GATT application registered
temperature value: 33 // command "J" let sensor start temperature posting
temperature value: 31
temperature value: 0
temperature value: 25
...
```

- ssh_win_3

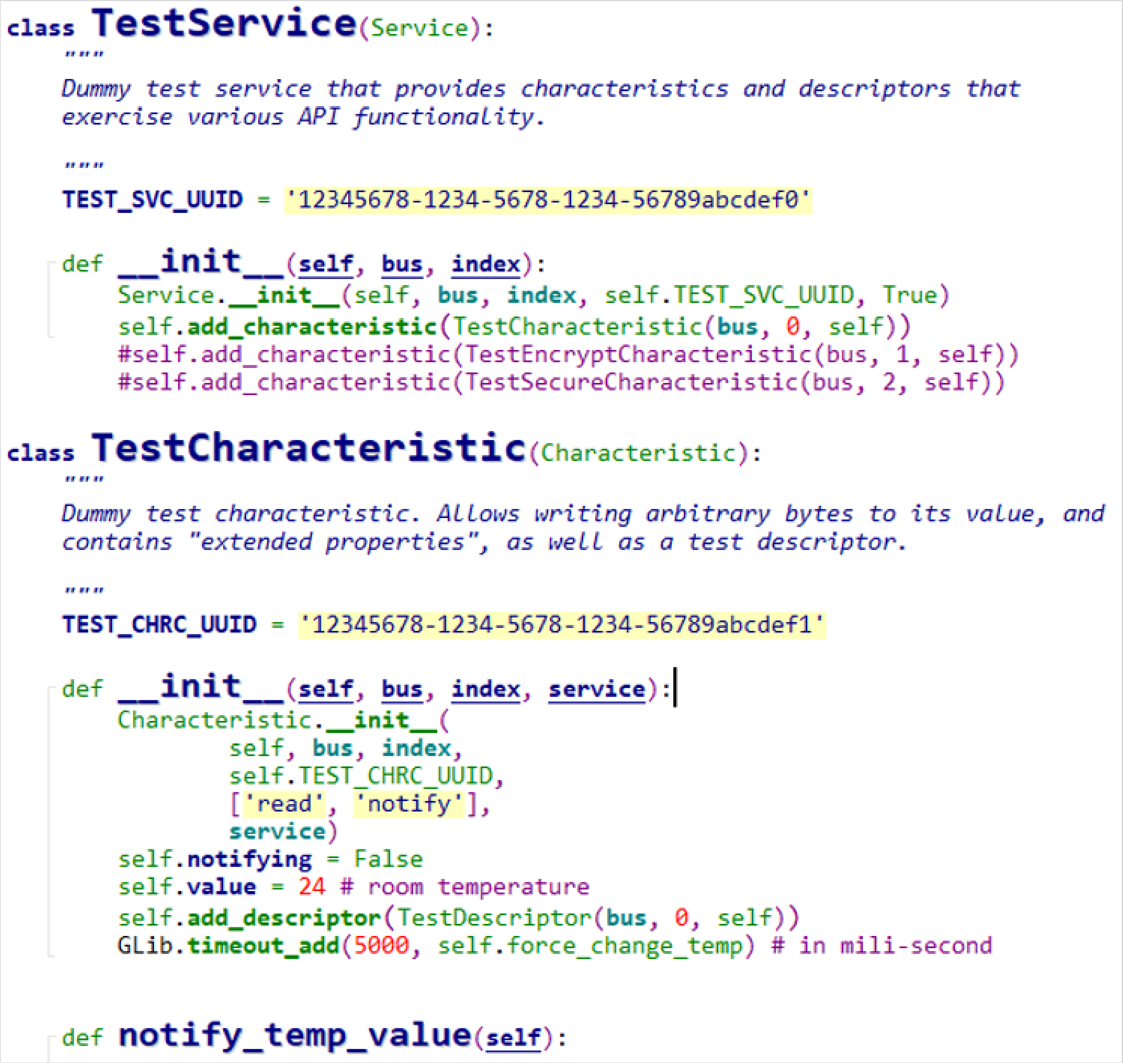
```
...
Registering GATT application...
GetManagedObjects
GATT application registered
temperature value: 8 // command "J" let sensor start temperature posting
temperature value: 12
temperature value: 37
temperature value: 2
temperature value: 32
temperature value: 19
temperature value: 30
temperature value: 37
temperature value: 4
temperature value: 28
temperature value: 39
...
```

- ssh_win_1

```
...
root@raspberrypi:/home/pi# python udp_server.py
UDP Server: waiting for a message ... at 172.16.30.136:10954
...
>>> iot_sensor[1], temperature value = 32
 >>> iot_sensor[2], temperature value = 17
 >>> iot_sensor[1], temperature value = 17
 >>> iot_sensor[2], temperature value = 21
...
```

The following is happening:

- DA16600_EVB_2 acts as Bluetooth® LE GAP Central. You will see it start LE scanning when it is booted
- We have two Bluetooth® LE (virtual) temperature sensors running: sensor_1 (RBP3_1) and sensor_2 (RBP3_2). And act as GAP Peripheral devices
- DA16600_EVB_2 finds two sensors during LE Scan
- If you run **proxm_sensor_gw conn 1**, DA16600_EVB_2 initiates connection to sensor_1. Same way, you also initiate a connection to sensor_2
- Once a connection is made, depending on the RF environment, you may need to try the scan command **proxm_sensor_gw scan** several times to get sensor 1 or 2 correctly listed
- When two sensors are connected, you can now set each sensor to start reading and posting temperature values with command **proxm_sensor_gw peer [1/2] J**
- An RBP3 Sensor has a simple GATT service running as shown in Figure 48



```
class TestService(Service):
    """
    Dummy test service that provides characteristics and descriptors that
    exercise various API functionality.

    """
    TEST_SVC_UUID = '12345678-1234-5678-1234-56789abcdef0'

    def __init__(self, bus, index):
        Service.__init__(self, bus, index, self.TEST_SVC_UUID, True)
        self.add_characteristic(TestCharacteristic(bus, 0, self))
        #self.add_characteristic(TestEncryptCharacteristic(bus, 1, self))
        #self.add_characteristic(TestSecureCharacteristic(bus, 2, self))

class TestCharacteristic(Characteristic):
    """
    Dummy test characteristic. Allows writing arbitrary bytes to its value, and
    contains "extended properties", as well as a test descriptor.

    """
    TEST_CHRC_UUID = '12345678-1234-5678-1234-56789abcdef1'

    def __init__(self, bus, index, service):
        Characteristic.__init__(
                self, bus, index,
                self.TEST_CHRC_UUID,
                ['read', 'notify'],
                service)
        self.notifying = False
        self.value = 24 # room temperature
        self.add_descriptor(TestDescriptor(bus, 0, self))
        GLib.timeout_add(5000, self.force_change_temp) # in mili-second


    def notify_temp_value(self):
```

Figure 48: Sensor_1/2 Bluetooth® LE Peer Sample Implementation (in Python)

- Sensor Service detail: implement the following GATT service on your Bluetooth® LE device
  - SVC: UUID = '12345678-1234-5678-1234-56789abcdef0'
  - Temperature characteristic: UUID = '12345678-1234-5678-1234-56789abcdef1', Permission = READ | NOTIFY, Value size = 1 byte
  - If subscription (on CCCD) is requested by a peer, periodic notification starts every five sec (the temperature value is notified every five seconds)
- Every five seconds, sensor_1 and sensor_2 each posts the current temperature to the sensor_gateway (DA16600_EVB_2)
- In the meantime, sensor_gateway is programmed to post one temperature message per every tenth message from a sensor, so you can see the UDP Server print the message received from DA16600_EVB_2 at ssh_win_1

6 Code Walkthrough

6.1 SDK Source Structure

In the DA16600, the Wi-Fi (DA16200) and Bluetooth® LE (DA14531) chips are connected via four-wire UART (Tx, Rx, RTS, and CTS, the baud rate is 115200 by default) and communicate with each other over Dialog Semiconductor's proprietary GTL interface. In the GTL architecture, a Bluetooth® LE application is running on the external host (DA16200).

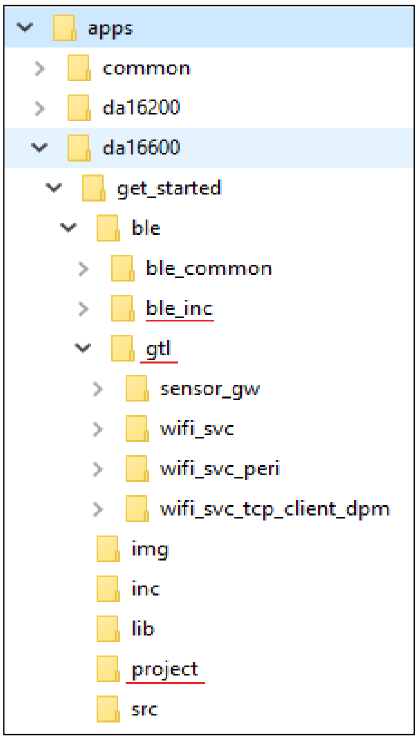
As the GTL architecture is used in DA16600, a user application developer should understand and know how to use the user APIs for both external host platforms (DA16200) and Bluetooth® LE platform (DA14531). Read the Programmer Guides for both platforms to get familiar with user APIs:

[1]: DA16200 SDK Programmer Guide

[2]: UM-B-143, Dialog External Processor Interface

[3]: DA14531 Getting Started Guide

[4]: DA14531 SW Platform Reference



```
apps
  common
  da16200
  da16600
    get_started
      ble
        ble_common
        ble_inc
        gtl
          sensor_gw
          wifi_svc
          wifi_svc_peri
          wifi_svc_tcp_client_dpm
      img
      inc
      lib
      project
      src
```

Figure 49: Source Folder Structure

- **ble_inc** folder contains the header files for the DA14531 SDK
  You can get the DA14531 SDK version being used at **\ble_inc**. If you need to migrate to a newer DA14531 SDK version, update the header files in this folder with those of the newer DA14531 SDK. Note that some header files may need to be modified for the host platform (DA16200)
- **gtl** folder contains four example applications:
  - **wifi_svc**: GAP Peripheral example application based on a Bluetooth® LE example "proximity reporter" (`[DA14531_SDK_ROOT] projects\host_apps\windows\proximity\reporter\`)
    This sample includes two applications:

    - **Wi-Fi Provisioning application**: GATT Server implementation to communicate with Bluetooth® LE peer applications (Wi-Fi Provisioning Mobile APP)
    - **Gas leak detection sensor application** works with a gas leak sensor (virtual) that is locally connected to the DA14531 chip. When a gas leak event occurs, this application posts a message to a network server in TCP/IP network
  - **wifi_svc_tcp_client_dpm**: GAP Peripheral example application based on a Bluetooth® LE example "proximity reporter" (`[DA14531_SDK_ROOT] projects\host_apps\windows\proximity \reporter\`)
    This sample includes two applications:
    - **Wi-Fi Provisioning application**: GATT Server implementation to communicate with Bluetooth® LE peer applications (Wi-Fi Provisioning Mobile APP)
    - **TCP Client DPM application**: a pure TCP/IP network application that communicates with a TCP Server in the network connected
  - **wifi_svc_peri**: GAP Peripheral example application based on a Bluetooth® LE example "proximity reporter" (`[DA14531_SDK_ROOT] projects\host_apps\windows\proximity \reporter\`)
    This sample includes two applications
    - **Wi-Fi Provisioning application**: GATT Server implementation to communicate with Bluetooth® LE peer applications (Wi-Fi Provisioning Mobile APP)
    - **DA14531 Peripheral driver sample**: this application configures and runs some peripheral devices locally attached to DA14531
  - **sensor_gw**: GAP Central example application based on a Bluetooth® LE example "proximity monitor" (`projects\host_apps\windows\proximity\monitor\`). This is a GATT Client application used in example 4
- **project** folder contains the project-related files to build with IAR tool.

6.2 Startup and GTL Initialization

```
// code highlighted in bold font, commentary in blue font

int system_start(void) {
…
      /* combo: init four-wire UART, dpm_boot_type, ble_boot_mode, download ble fw.
dpm_boot_type identifies system state like NO_DPM/DPM/DPM_WAKEUP/…. ble_boot_mode
decides whether or not ble downloading is needed or not. */
        combo_init();
        ...
        Wlaninit(); // wlan is initialized.
        ...
        start_sys_apps();
        ...
        start_user_apps();
}

// When external host (DA16200) boots, BLE user application thread defined in
user_apps_table[ ] is run with user defined config
```

```
// user_apps.c
...
const app_thread_info_t    user_apps_table[] = {
...
{ APP_GTL_INIT,         gtl_init,             2048, OAL_PRI_APP(1), FALSE,      FALSE,
UNDEF_PORT,             RUN_STA_MODE  },
...
}
// see DA16200 Programmer Guide [1] > Section 2.4

// wifi_svc_app.c
void gtl_init(ULONG arg) {
...
host_uart_init(); // UART init
...
boot_ble_from_uart(); // transfer ble firmware to DA14531
...
tx_thread_create(...,gtl_main, ...) // the main gtl message handler
...
}

void gtl_main(ULONG arg)
{
...
tx_thread_create(...,gtl_host_ifc_mon, ...) // a thread monitoring GTL message Rx
...
While(1) {

BleReceiveMsg(); // this function is the main GTL message handler, ensure that
HandleBleMsg() is invoked to check which GTL message is explicitly handled.
}
```

6.3 DA16600 Application Development

The four Bluetooth® LE example applications (**Gas leak sensor, TCP Client, DA14531 Peri Driver**, and **Sensor Gateway**) are based on the two basic external host examples included in the DA14531 SDK:

- [DA14531_SDK_ROOT]\projects\host_apps\windows\proximity\monitor
- [DA14531_SDK_ROOT]\projects\host_apps\windows\proximity\reporter

The Bluetooth® LE application flow (initialization and operation) is the same as for those examples.

A DA16600 user application (that may want to use both Wi-Fi and Bluetooth® LE functions) needs the development of functions that use both Wi-Fi APIs and Bluetooth® LE APIs.

To develop a function that talks to a Bluetooth® LE Peer (for example, can talk to the Provisioning Mobile Application), the application should be developed based on UM-B-143, Dialog External Processor Interface [2].

To develop a local function (for example, driver function) in DA14531 (for example, to handle a custom GTL message that should be handled in DA14531), the user also needs to understand the local APIs of the DA14531. See Section 2 of Ref. [4]. Software Platform Overview, or the API documentation included in the DA14531 SDK.

To develop a function that talks to a networked TCP/UDP peer (for example, can talk to RBP3's UDP Server Application), the application should be developed based on the DA16200 SDK Programmer Guide [1].

6.4 Examples

6.4.1 Bluetooth® LE Assisted Wi-Fi Provisioning

6.4.1.1 Application Build and Run

```
// code highlighted in bold font, commentary in blue font

// NOTE: Wi-Fi Provisioning (Bluetooth® LE GATT Service) application/service is
included in all the examples. Below we explain the code using one (/wifi_svc) of
examples.

apps/da16600/get_started/inc/ble_combo_features.h
...
#define __COMBO_SAMPLE_BLE_PERI_WIFI_SVC__
#undef __WIFI_SVC_SECURITY__ // if you enable this option, when a peer tries to
access one of your characteristics (1 time), a pop-up window asking pin code is
shown at a peer app side.
...
#undef __COMBO_SAMPLE_BLE_CENT_SENSOR_GW__
...
```

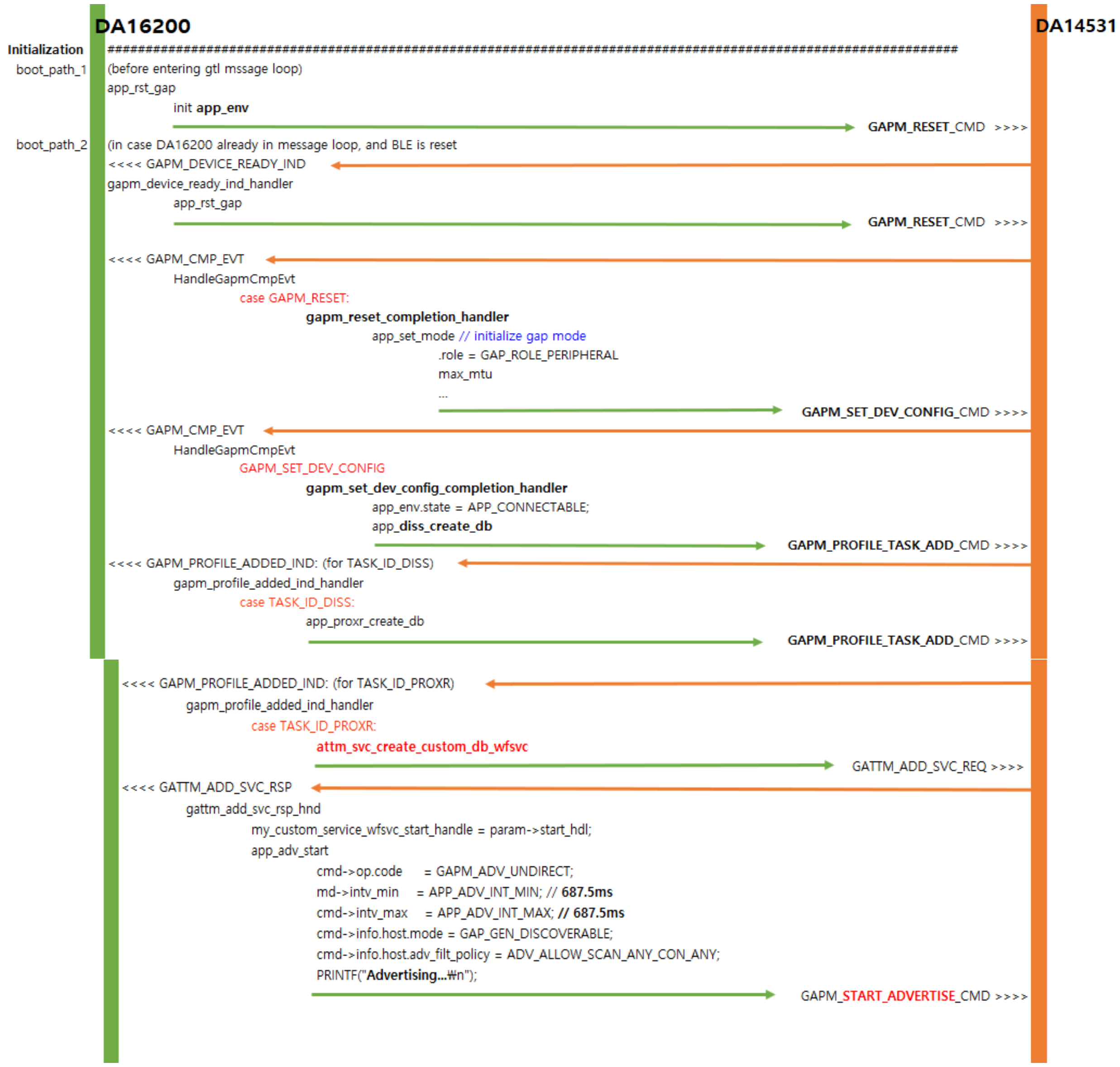
6.4.1.2 Application Initialization

```
// code highlighted in bold font, commentary in blue font

/*
There are two main user threads created …
- To read GTL messages from UART: gtl_host_ifc_mon
- To handle a GTL message: gtl_main
*/

// wifi_svc_app.c
void gtl_main(ULONG arg)
{
...
      status = tx_thread_create(tx_gtl_host_ifc_mon, // GTL message reader thread
                     "gtl_host_ifc_mon",
                     gtl_host_ifc_mon,
                     NULL,
                     gtl_host_ifc_mon_stack,
                     GTL_HOST_IFC_MON_STACK_SZ,
                     OAL_PRI_APP(0),
                     OAL_PRI_APP(0),
                     TX_NO_TIME_SLICE,
                     TX_AUTO_START);
...

      // GTL message handler loop
      while(1)
      {
        BleReceiveMsg(); // once a GTL message is available in the queue, this loop
```

```
                         // resumes and handles it.
      }
...
void gtl_host_ifc_mon(ULONG arg)
{
      UARTProc();
}

void UARTProc(void)
{
...
while(1)
    {
       tmp = getchar_uart1(OAL_SUSPEND); // waiting for data from DA14531
       ...
         switch(bReceiveState)
        {
         case 0:    // Receive FE_MSG
            if(tmp == FE_MSG_PACKET_TYPE)
            {
               bReceiveState = 1;
          ...
                   // once a known type of message - GTL type - is received, it is
                   // EnQueued (GtlRxQueue)
        SendToMain((unsigned short) (wReceive232Pos-1), &bReceive232ElementArr[1]);
...
      }
}

void BleReceiveMsg(void)
{
  ...
  msg = (ble_msg*) DeQueue(&GtlRxQueue); // a message is dequeued
  ...
  HandleBleMsg(msg); // main GTL message handler function
  ...
}

void HandleBleMsg(ble_msg *blemsg)
{
// for each GTL message type, a handler defined is invoked. This message type is
defined in DA14531 SDK headers.
...
case GAPM_CMP_EVT:
   HandleGapmCmpEvt(blemsg->bType, (struct gapm_cmp_evt *)blemsg->bData, blemsg-
>bDstid, blemsg->bSrcid);
...
case GAPM_DEVICE_READY_IND:
  DBG_INFO("<<< GAPM_DEVICE_READY_IND \n");
  gapm_device_ready_ind_handler(blemsg->bType, (struct gap_ready_evt *)blemsg-
>bData, blemsg->bDstid, blemsg->bSrcid);
  break;
...
```

6.4.1.3 GTL Message Flow

Initialization until Advertising



```
DA16200                                                                              DA14531
Initialization  ################################################################################
boot_path_1     (before entering gtl mssage loop)
                app_rst_gap
                          init app_env
                          ------------------------------------------------> GAPM_RESET_CMD >>>>
boot_path_2     (in case DA16200 already in message loop, and BLE is reset
                <<<< GAPM_DEVICE_READY_IND <------------------------------------------------
                gapm_device_ready_ind_handler
                          app_rst_gap
                          ------------------------------------------------> GAPM_RESET_CMD >>>>

                <<<< GAPM_CMP_EVT <---------------------------------------------------------
                          HandleGapmCmpEvt
                                    case GAPM_RESET:
                                              gapm_reset_completion_handler
                                                        app_set_mode // initialize gap mode
                                                                  .role = GAP_ROLE_PERIPHERAL
                                                                  max_mtu
                                                                  ...
                                                                  -------------> GAPM_SET_DEV_CONFIG_CMD >>>>
                <<<< GAPM_CMP_EVT <---------------------------------------------------------
                          HandleGapmCmpEvt
                                    GAPM_SET_DEV_CONFIG
                                              gapm_set_dev_config_completion_handler
                                                        app_env.state = APP_CONNECTABLE;
                                                        app_diss_create_db
                                                        ---------------> GAPM_PROFILE_TASK_ADD_CMD >>>>
                <<<< GAPM_PROFILE_ADDED_IND: (for TASK_ID_DISS) <----------------------------
                          gapm_profile_added_ind_handler
                                    case TASK_ID_DISS:
                                              app_proxr_create_db
                                              -------------------> GAPM_PROFILE_TASK_ADD_CMD >>>>

                  <<<< GAPM_PROFILE_ADDED_IND: (for TASK_ID_PROXR) <-------------------------
                            gapm_profile_added_ind_handler
                                      case TASK_ID_PROXR:
                                                attm_svc_create_custom_db_wfsvc
                                                --------------------------> GATTM_ADD_SVC_REQ >>>>
                  <<<< GATTM_ADD_SVC_RSP <--------------------------------------------------
                            gattm_add_svc_rsp_hnd
                                      my_custom_service_wfsvc_start_handle = param->start_hdl;
                                      app_adv_start
                                                cmd->op.code     = GAPM_ADV_UNDIRECT;
                                                md->intv_min    = APP_ADV_INT_MIN; // 687.5ms
                                                cmd->intv_max    = APP_ADV_INT_MAX; // 687.5ms
                                                cmd->info.host.mode = GAP_GEN_DISCOVERABLE;
                                                cmd->info.host.adv_filt_policy = ADV_ALLOW_SCAN_ANY_CON_ANY;
                                                PRINTF("Advertising...₩n");
                                                ---------------------> GAPM_START_ADVERTISE_CMD >>>>
```

Figure 50: GTL Message Sequence Chart – Initialization

Connection Request and Characteristic "Write" Request

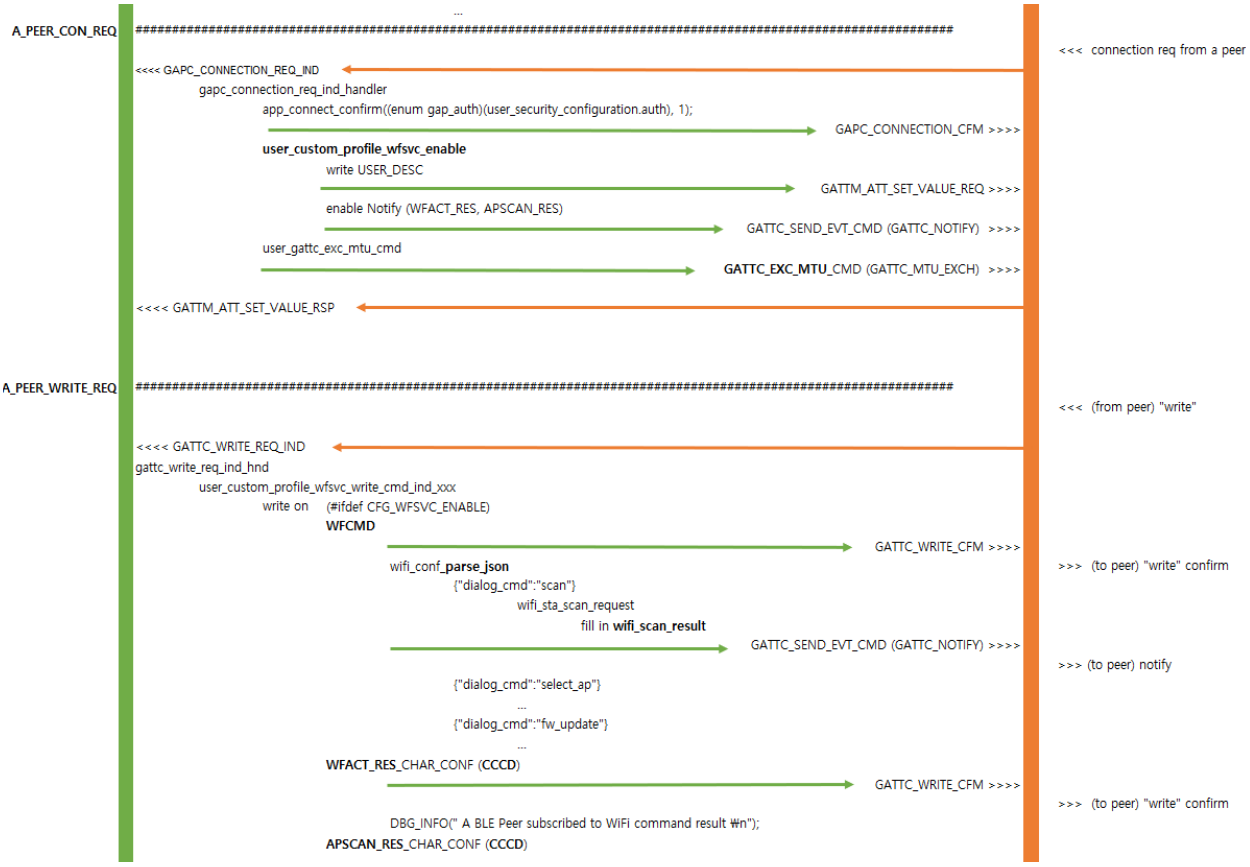

...
A_PEER_CON_REQ
###
<<< connection req from a peer
<<<< GAPC_CONNECTION_REQ_IND
gapc_connection_req_ind_handler
app_connect_confirm((enum gap_auth)(user_security_configuration.auth), 1);
GAPC_CONNECTION_CFM >>>>
user_custom_profile_wfsvc_enable
write USER_DESC
GATTM_ATT_SET_VALUE_REQ >>>>
enable Notify (WFACT_RES, APSCAN_RES)
GATTC_SEND_EVT_CMD (GATTC_NOTIFY) >>>>
user_gattc_exc_mtu_cmd
GATTC_EXC_MTU_CMD (GATTC_MTU_EXCH) >>>>
<<<< GATTM_ATT_SET_VALUE_RSP
A_PEER_WRITE_REQ
###
<<< (from peer) "write"
<<<< GATTC_WRITE_REQ_IND
gattc_write_req_ind_hnd
user_custom_profile_wfsvc_write_cmd_ind_xxx
write on (#ifdef CFG_WFSVC_ENABLE)
WFCMD
GATTC_WRITE_CFM >>>>
>>> (to peer) "write" confirm
wifi_conf_parse_json
{"dialog_cmd":"scan"}
wifi_sta_scan_request
fill in wifi_scan_result
GATTC_SEND_EVT_CMD (GATTC_NOTIFY) >>>>
>>> (to peer) notify
{"dialog_cmd":"select_ap"}
...
{"dialog_cmd":"fw_update"}
...
WFACT_RES_CHAR_CONF (CCCD)
GATTC_WRITE_CFM >>>>
>>> (to peer) "write" confirm
DBG_INFO(" A BLE Peer subscribed to WiFi command result ₩n");
APSCAN_RES_CHAR_CONF (CCCD)

Figure 51: GTL Message Sequence Chart – Connect and Write

Characteristic "Read" Request

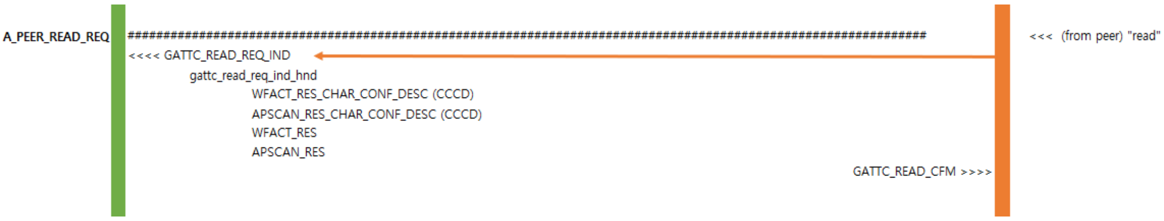


Figure 52: GTL Message Sequence Chart – Read

6.4.1.4 Wi-Fi Service GATT Database Design

The sample "Wi-Fi Service" GATT Server database is designed as in Table 2. This is just a sample database design, so SDK users that depend on their application use cases may want to create a different design. See file **wifi_svc_user_custom_profile.c/h** for more details.

Table 2: Wi-Fi SVC GATT Service Design

Att Idx		Att Type	UUID	Prop. (R/W)	Value Size	RI	User Desc.	Value to Read/Write
Wi-Fi Service			9161b2**01**-1b4b-4727-a3ca-47b35cdcf5c1					
1	Wi-Fi cmd	Declaration						

2		Value	9161b2**02**-1b4b-4727-a3ca-47b35cdcf5c1	Write	244	-		{
								"dialog_cmd":"scan"
								}
								{
								"dialog_cmd":"select_ap",
								"SSID":"linksys",
								"security_type":3,
								...
								}
								See the function wifi_conf_parse_json() for more details

3		Description (user)		Read		N	Wi-Fi Cmd	
4	Wi-Fi action result	Declaration				N		

5		value	9161b2**03**-1b4b-4727-a3ca-47b35cdcf5c1	Read Notify	2	Y		// Wi-Fi Action Result
								enum WIFI_ACTION_RESULT
								{
								COMBO_WIFI_CMD_SCAN_AP_SUCCESS = 1,
								COMBO_WIFI_CMD_SCAN_AP_FAIL,
								COMBO_WIFI_CMD_FW_BLE_DOWNLOAD_SUCCESS,
								COMBO_WIFI_CMD_FW_BLE_DOWNLOAD_FAIL,
								...
								};
								See the "enum WIFI_ACTION_RESULT" for more details

6		Description (cccd)		Read Write Write w/o.		Y		
7		Description (user)		Read		N		

Att Idx		Att Type	UUID	Prop. (R/W)	Value Size	RI	User Desc.	Value to Read/Write
8	AP Scan result	Declaration				N	Wi-Fi Action Result	
9		Value	9161b2**04**-1b4b-4727-a3ca-47b35cdcf5c1	Read	244	Y		
10		Description (user)		Read		N	AP Scan Result	

6.4.1.5 Application Protocol Between a Bluetooth® LE Peer and Wi-Fi SVC Application

Based on the Wi-Fi SVC GATT Database described in the previous section, the following protocols are used between the Wi-Fi SVC application and the Provisioning Mobile App application.

- Characteristic: "Wi-Fi Cmd" (244 bytes), **WRITE**
  - "scan" command can be sent by a Bluetooth® LE Peer that wants the DA16600 device to scan Wi-Fi routers

```
{
    "dialog_cmd":"scan"
}
```

  - "select_ap" command can be sent by a Bluetooth® LE Peer to tell the DA16600 device which AP to connect to (plus other customer proprietary information, if needed, such as customer server information)
    Upon receipt of this command, DA16600 stores the credentials in permanent storage (NVRAM) and reboots

```
{
    "dialog_cmd":"select_ap",
    "SSID":"linksys",
    "security_type":3,      /* 0 = none, 1 = WEP, 2 = WPA, 3 = WPA2 */
    "DHCP":0,               /* 0 = Static IP, 1 = DHCP, 2 = RFC3927 */
    "password":"123456789",
    "ipv4_address":"192.168.20.5",
    "mask":"255.255.255.0",
    "gateway":"192.168.20.1",
    "dns_address":"8.8.8.8",
    "dpm_mode":0 // for Demo 3, the value should be 1
     ...
     // additional paramters if needed
      "svr_addr":"172.16.0.100", // should be valid for Demo 3
      "svr_port":10925,          // should be valid for Demo 3
    ...
}
```

  - "fw_update" command can be sent by a Bluetooth® LE Peer to tell DA16600 to download new firmware from a specified OTA server

```
{
    "dialog_cmd":"fw_update"
}
```

- "factory_reset" command can be sent by a Bluetooth® LE Peer to tell DA16600 to remove the Wi-Fi network profile saved. The DA16600 EVB will reboot after the reset

```
{
    "dialog_cmd":"factory_reset"
}
```

- "reboot" command can be sent by a Bluetooth® LE Peer to tell DA16600 to reboot. On reboot, if a valid network profile exists, the DA16200 connects to the specified AP

```
{
    "dialog_cmd":"reboot"
}
```

- "wifi_status" command can be sent by a Bluetooth® LE Peer to find out the Wi-Fi connection status (connected or disconnected). The Bluetooth® LE peer can be notified of, or read, the status via the "Wi-Fi Action Result" characteristic

```
{
    "dialog_cmd":"wifi_status"
}
```

- "disconnect" command can be sent by a Bluetooth® LE Peer to disconnect a Wi-Fi connection from the currently connected AP. If the Bluetooth® LE Peer wants to reconnect to the same AP, it should send the command "reboot" to have the DA16600 EVB connect to the AP

```
{
    "dialog_cmd":"disconnect"
}
```

- If you want to add a new custom command, use the following
  - enum WIFI_CMD // define a new custom command
  - user_custom_profile_wfsvc_write_cmd_ind_xxx( ) // add the handler of the command
  - wifi_conf_parse_json // add the parser of the command

- Characteristic: "Wi-Fi Action Result" (two bytes), **READ, NOTIFY**
  - A Bluetooth® LE Peer is supposed to enable notification on this characteristic on connection
  - Then the Bluetooth® LE Peer is notified of the result of a Wi-Fi command sent

```
// Wi-Fi Action Result
enum WIFI_ACTION_RESULT {
      COMBO_WIFI_CMD_SCAN_AP_SUCCESS = 1,
      COMBO_WIFI_CMD_SCAN_AP_FAIL,

      COMBO_WIFI_CMD_FW_BLE_DOWNLOAD_SUCCESS,
      COMBO_WIFI_CMD_FW_BLE_DOWNLOAD_FAIL,

      COMBO_WIFI_CMD_INQ_WIFI_STATUS_CONNECTED,
      COMBO_WIFI_CMD_INQ_WIFI_STATUS_NOT_CONNECTED,

      COMBO_WIFI_PROV_DATA_VALIDITY_CHK_ERR,
      COMBO_WIFI_PROV_DATA_SAVE_SUCCESS,

      COMBO_WIFI_CMD_MEM_ALLOC_FAIL,

      COMBO_WIFI_CMD_UNKNOWN_RCV
};
```

  - Especially on receipt of "COMBO_WIFI_CMD_SCAN_AP_SUCCESS", a Bluetooth® LE Peer is supposed to initiate a "READ" request on the characteristic "AP Scan Result". Usually, the total list of AP Scan results is about 2 kB in size, therefore the Bluetooth® LE Peer should initiate a "READ" request on the characteristic multiple times until all SSIDs are fully read

- Characteristic: "AP Scan Result" (244 bytes), **READ**
  - When a Bluetooth® LE Peer gets the first read response (with payload), the payload included in the "read" response includes a 4 bytes 'application-specific custom' header (that you can modify freely to your application needs). See below the sample payload structure.
  - [H_1][H_2][JSON_STR]
    - H_1: first two bytes of the header, includes the **remaining length** of the total JSON_STR
    - H_2: the second two bytes of the header, includes the **total length** of JSON_STR. Normally the total size is over 2 kB
    - JSON_STR: JSON encoded "AP Scan result"
  - Upon receipt of a read response, a Bluetooth® LE Peer is supposed to keep triggering "read" requests on this characteristic until H_1 becomes 0. The read response message that contains 0 as H_1 contains the final fragment of JSON_STR
  - Next, a Bluetooth® LE Peer needs to combine and parse the whole JSON_STR to get the necessary information (in this case, the list of Wi-Fi routers)

```
Example

[
{
"SSID":"MyAP", // SSID of an AP
"security_type": 3 // 0=none, 1=WEP, 2=WPA, 3=WPA2
"signal_strength": 56 // in dBm
},
{
"SSID":"MyAP2", // SSID of an AP
"security_type": 0 // 0=none, 1=WEP, 2=WPA, 3=WPA2
"signal_strength": 24 // in dBm
},
{
"SSID":"MyAP3", // SSID of an AP
"security_type": 3 // 0=none, 1=WEP, 2=WPA, 3=WPA2
"signal_strength": 67 // in dBm
},
...
]
```

Depending on how a Bluetooth® LE Peer App is implemented, a Bluetooth® LE Peer App may let the user select an AP from the list and send a "write" request with {"dialog_cmd":"**select_ap**", ….} on the characteristic "Wi-Fi Cmd".

6.4.2 Bluetooth® LE Firmware OTA Download via Wi-Fi

OTA FW Download Service (Wi-Fi download of FW) is supported and enabled in all examples.

Depending on the customer use scenario, there may be a few ways to trigger an OTA FW Download Service. For example:

- Option 1: using a networked peer (a mobile application that is connected to the Internet or a customer cloud server application on the Internet)
- Option 2: using a Bluetooth® LE Peer

In this example, Option 2 is demonstrated.

NOTE

Option 1 can be used for unattended/automatic OTA operation. It works as follows:

As a DA16600 device (DA16200 Wi-Fi chip included) is 'always' in a connected state with a Wi-Fi router, which has the Internet connection, a Service Server/Cloud Server can talk with this device when there is a new firmware available on an OTA Server. A Service Server, if needed, can contact a user via 4G/Wi-Fi (in the form of a push message) for firmware update confirmation. Upon receipt of 'user confirmation', the Service Server may ask the DA16600 device to download a new firmware by giving an HTTP(s) URI to an OTA Server. Once the new firmware is received, that firmware is stored in the SFLASH of the DA16600 device, and the DA16600 device restarts to boot with the new software.

For **Option 2**, a Bluetooth® LE Peer App should 'write' the following command on the characteristic "Wi-Fi Cmd" to trigger an OTA FW update:

```
{
    "dialog_cmd":"fw_update"
}
```

Upon receipt of the command, GATTC_WRITE_REQ_IND is sent to the external host (DA16200), and then a handler is invoked.

```
// code highlighted in bold font, commentary in blue font

uint8_t user_custom_profile_wfsvc_write_cmd_ind_xxx(ke_msg_id_t const msgid,
                          struct gattc_write_req_ind const *param,
                          ke_task_id_t const dest_id,
                          ke_task_id_t const src_id)
{
...
       /* Parse JSON */
       if (wifi_conf_parse_json(value, &cmd)!= 0) {
              DBG_INFO(" UNKNOWN COMMAND!!! \n");
              wfsvc_wifi_cmd_result_nofity_user(COMBO_WIFI_CMD_UNKNOWN_RCV);
              goto finish;
       }
...
       else if (cmd == COMBO_WIFI_CMD_FW_UPDATE) {
              DBG_INFO(" COMBO_WIFI_CMD_FW_BLE_DOWNLOAD received \n");

              // download DA14531 image from the URL to a special area in sflash

              /* DEMO - TEST CODE */
              ota_fw_update_combo();

              // notify wifi-cmd execution result

     wfsvc_wifi_cmd_result_nofity_user(COMBO_WIFI_CMD_FW_BLE_DOWNLOAD_SUCCESS);
       }
...

UINT ota_fw_update_combo(void)
{
...
status = ota_update_start_download(fw_conf); // API for OTA FW Download service
...
}
```

```
// OTA_update_start_download() function creates an OTA thread - thread function :
ota_update_http_client_update_proc() - which handles http connection, http
download, and firmware renewing process.
```

6.4.3 Gas Leak Detection Sensor

For this example, the same application is used. For information on build and run, see Section 6.4.1.

NOTE

The Gas Leak Detection Sensor application does not need a connection with a Bluetooth® LE Peer.

6.4.3.1 User Interface

Normally, a user may want to use a Mobile Phone (a network peer that is always connected to a Service Server or Cloud Server which is then connected to a gas leak sensor) to control a Wi-Fi IoT device (gas leak sensor). But in this example, DA16200 console commands are used as a sensor controller. See Section 5.4 to know which commands to use (for example, iot_sensor start).

6.4.3.2 Operation

When the command **Iot_sensor start** or **iot_sensor stop** is used, the following function is invoked:

```
void ConsoleSendSensorStart(void)
{
    console_msg *pmsg = (console_msg *) malloc(sizeof(console_msg));
    pmsg->type = CONSOLE_IOT_SENSOR_START;
    EnQueueToConsoleQ(pmsg);
}
...
void ConsoleEvent(void)
{
...
       switch(msg->type)
        {
              case CONSOLE_IOT_SENSOR_START:
                     if (app_env.iot_sensor_state == IOT_SENSOR_INACTIVE) {
                            app_sensor_start();
                     }
              case CONSOLE_IOT_SENSOR_STOP:
                     if (app_env.iot_sensor_state == IOT_SENSOR_ACTIVE) {
                            app_sensor_stop();
                     }
            default:
                 break;
        }
...

void app_sensor_start(void) {
    // Allocate a message for DA14531 (BLE_AUX task - which is a user task)
    void *cmd = BleMsgAlloc(APP_GAS_LEAK_SENSOR_START, TASK_ID_EXT_HOST_BLE_AUX,
TASK_ID_GTL, 0);
    // Send the message
    BleSendMsg(cmd);
}
```

```
// the message " APP_GAS_LEAK_SENSOR_START" is sent to Bluetooth® LE(DA14531) which
starts sensor reading task (periodically reads a gas-leak sensor).
```

For the external host (DA16200) to be able to talk to the Bluetooth® LE (DA14531) with application messages, the following should be implemented on both Wi-Fi and Bluetooth® LE SDKs:

- For both DA16200 SDK and DA14531 SDK, define custom messages
  - DA16600 SDK – send a custom user-defined message via the GTL interface with TASK_ID_EXT_HOST_BLE_AUX as the destination task
  - DA14531 SDK – enable the BLE AUX task (TASK_ID_EXT_HOST_BLE_AUX) to receive a user-defined custom message
    Develop the message handlers

```
// code highlighted in bold font, commentary in blue font

// Define application-specific messages for your application

// app.h (in DA16600 SDK) and ext_host_ble_aux_task.h (in DA14531 SDK)
...
// the exact same enum should exist on DA14531 SDK
typedef enum {
...

    APP_BLE_SW_RESET,

    APP_GAS_LEAK_SENSOR_START,
    APP_GAS_LEAK_SENSOR_START_CFM,
    APP_GAS_LEAK_SENSOR_STOP,
    APP_GAS_LEAK_SENSOR_STOP_CFM,
    APP_GAS_LEAK_EVT_IND,
    APP_GAS_LEAK_SENSOR_RD_TIMER_ID,

    APP_CUSTOM_COMMANDS_LAST,
} ext_host_ble_aux_task_msg_t;
...

// DA16600 SDK: Develop a function to send the message
void app_sensor_start(void) {
    // Allocate a message for BLE_AUX
    void *cmd = BleMsgAlloc(APP_GAS_LEAK_SENSOR_START, TASK_ID_EXT_HOST_BLE_AUX,
TASK_ID_GTL, 0);
    // Send the message
    BleSendMsg(cmd);
}

// DA14531 SDK: enable BLE AUX task
// user_proxr.c
...
void user_on_init(void)
{
...
     ext_host_ble_aux_init();
     ext_host_ble_aux_task_init();
...
}

// DA14531 SDK: develop message handlers
// ext_host_ble_aux_task.c

int ext_host_ble_aux_task_handler (ke_msg_id_t const msgid,
                                          void const *param,
                                          ke_task_id_t const dest_id,
                                          ke_task_id_t const src_id)
{
…
if (msg->dest_id == TASK_EXT_HOST_BLE_AUX)
    {
        switch (msgid)
        {
...
            case APP_GAS_LEAK_SENSOR_START:
                app_gas_leak_sensor_start_cfm_send();
                break;

            case APP_GAS_LEAK_SENSOR_STOP:
                app_gas_leak_sensor_stop_cfm_send();
                break;
...
}

void app_gas_leak_sensor_start_cfm_send(void)
{
     ke_timer_set(APP_GAS_LEAK_SENSOR_RD_TIMER_ID, TASK_EXT_HOST_BLE_AUX,
APP_GAS_LEAK_SENSOR_RD_INTERVAL);

    app_gas_leak_sensor_start_cfm_t *req =
(app_gas_leak_sensor_start_cfm_t*)ke_msg_alloc(APP_GAS_LEAK_SENSOR_START_CFM,
TASK_GTL, TASK_EXT_HOST_BLE_AUX, sizeof(app_gas_leak_sensor_start_cfm_t));
     req->status = APP_NO_ERROR;
    ke_msg_send(req);
}
```

When the gas leak sensor starts, DA16600 enters sleep. Later, if a gas leak event occurs, DA14531 wakes up DA16200 which then sends a message to a server and enters sleep again.

```
int system_start(void)
{
…
sleep2_monitor_start(); // start sleep2_montor. Sleep2 monitor is used to
synchronize sleep between gas leak sensor app and wifi provisioning app,
…
}

void gtl_init(ULONG arg)
{
…
sleep2_monitor_regi(SLEEP2_MON_NAME_PROV, __func__, __LINE__); // register gas leak
application to sleep2_monitor
…
}

int app_sensor_event_ind_hnd(ke_msg_id_t msgid,
                                          app_gas_leak_evt_ind_t *param,
                                          ke_task_id_t dest_id,
                                             ke_task_id_t src_id)
{
…
       sleep2_monitor_set_state(SLEEP2_MON_NAME_IOT_SENSOR,
                     SLEEP2_CLR, __func__, __LINE__); // when gas leak happens, tell
sleep2_monitor to hold entering sleep
..
...
iot_sensor_data_info.is_gas_leak_happened = TRUE; // set "gas leak happened" flag
..
}

void  udp_client(char *ip_addr_str, int udp_server_port)
{
…
      while (1)
      {
        …
       if (iot_sensor_data_info.is_gas_leak_happened == FALSE)
       {
              OAL_MSLEEP(100);
              continue; // busy loop on gas leak sensor event
       }
       …
       // gas leak occurred
       …
       /* send gas leak message to server */
       status = nx_udp_socket_send(udp_client_sock,
                                   udp_client_pkt,
                                   udp_server_ip_addr,
                                   udp_server_port);
       …
       sleep2_monitor_set_state(SLEEP2_MON_NAME_IOT_SENSOR,
                            SLEEP2_SET, __func__, __LINE__); // job done. tell
sleep2_monitor to enter sleep
…
}
```

6.4.4 DA14531 Peripheral Driver Example

6.4.4.1 Build

```
// Enable the sample feature below and disable the other samples, and build SDK

apps\da16600\get_started\inc\ble_combo_features.h
...
#define __COMBO_SAMPLE_BLE_PERI_WIFI_SVC_PERIPHERAL_SAMPLE__
...
```

6.4.4.2 Example Code Flow

```
// code highlighted in bold font, commentary in blue font

// Take TIMER0_BUZ (Buzzer playback sample) as an example flow (the other samples
have the same flow sequence)
```

```
apps\da16600\get_started\ble\gtl\wifi_svc\inc\app.h

...

// GTL messages are defined for TIEMR0_BUZ
typedef enum
{
...
    APP_PERI_BLINKY_START,
    APP_PERI_BLINKY_START_IND,
    APP_PERI_BLINKY_RUN_TIMER_ID,

    APP_PERI_SYSTICK_START,
    APP_PERI_SYSTICK_START_IND,
    APP_PERI_SYSTICK_RUN_TIMER_ID,
    APP_PERI_SYSTICK_STOP,
    APP_PERI_SYSTICK_STOP_IND,

    APP_PERI_TIMER0_GEN_START,
    APP_PERI_TIMER0_GEN_START_IND,
    APP_PERI_TIMER0_GEN_RUN_TIMER_ID,

    APP_PERI_TIMER0_BUZ_START,
    APP_PERI_TIMER0_BUZ_START_IND,
    APP_PERI_TIMER0_BUZ_RUN_TIMER_ID,
...
    APP_CUSTOM_COMMANDS_LAST,
} ext_host_ble_aux_task_msg_t;

apps\da16600\get_started\ble\gtl\wifi_svc_peri\src\wifi_svc_peri_console.c
// user command "ble.peri timer0_buz start" brings an interactive setup menus
void cmd_peri_sample(int argc, char *argv[])
{
...
// collect timer0_buz driver configuration info (to fill in the structure below)
from user
/*
// timer0_buz
typedef struct app_peri_timer0_buz_start
{
    uint8_t t0_pwm0_port;
    uint8_t t0_pwm0_pin;
    uint8_t t0_pwm1_port;
    uint8_t t0_pwm1_pin;

    uint16_t buz_tst_counter;

    tim0_2_clk_div_t t0_t2_in_clk_div_factor;
    TIM0_CLK_SEL_t   t0_clk_src;
    PWM_MODE_t       t0_pwm_mod;
    TIM0_CLK_DIV_t   t0_clk_div;

    int t0_on_reld_val;
    int t0_high_reld_val;
    int t0_low_reld_val;
} app_peri_timer0_buz_start_t;
```

```
*/

...

// send the sample configuration info to DA14531
app_peri_timer0_buz_start_send(&buz_conf);
...
}

apps\da16600\get_started\ble\gtl\wifi_svc_peri\src\wifi_svc_peri_app.c

// timer0_buzzer ------------------------------------------------------------
void app_peri_timer0_buz_start_send(app_peri_timer0_buz_start_t* config_param)
{
    app_peri_timer0_buz_start_t *req =
(app_peri_timer0_buz_start_t*)BleMsgAlloc(APP_PERI_TIMER0_BUZ_START,
                                  TASK_ID_EXT_HOST_BLE_AUX,
                                  TASK_ID_GTL,
                                  sizeof(app_peri_timer0_buz_start_t));

    memcpy((void*)req, (void*)config_param, sizeof(app_peri_timer0_buz_start_t));

    BleSendMsg(req);
}

// Now let's look at DA14531 SDK to check how APP_PERI_TIMER0_BUZ_START is handled

[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex\
include\ext_host_ble_aux_task.h

// same GTL messages are defined in the same order

typedef enum
{
...
    APP_PERI_TIMER0_BUZ_START,
    APP_PERI_TIMER0_BUZ_START_IND,
    APP_PERI_TIMER0_BUZ_RUN_TIMER_ID,
...
} ext_host_ble_aux_task_msg_t;

[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex\
src\ext_host_ble_aux_task.c

int ext_host_ble_aux_task_handler (ke_msg_id_t const msgid,
                                         void const *param,
                                         ke_task_id_t const dest_id,
                                         ke_task_id_t const src_id)
{
...
if (msg->dest_id == TASK_EXT_HOST_BLE_AUX)
    {
        switch (msgid)
        {
...        // GTL message handler
          case APP_PERI_TIMER0_BUZ_START:
           app_peri_timer0_buz_start_ind_send((app_peri_timer0_buz_start_t*)msg-
          >param);
           break;
...
}

void app_peri_timer0_buz_start_ind_send(app_peri_timer0_buz_start_t* config_data)
{
      // run a timer to start the sample action in 2 seconds
     ke_timer_set(APP_PERI_TIMER0_BUZ_RUN_TIMER_ID, TASK_EXT_HOST_BLE_AUX,
APP_PERI_SAMPLE_RUN_AFTER);

     memcpy((void*)&conf_buz, (void*)config_data,
sizeof(app_peri_timer0_buz_start_t));
     clk_div_config.clk_div = conf_buz.t0_t2_in_clk_div_factor;

    // send back the GTL "APP_PERI_TIMER0_BUZ_START_IND" to DA16200 indicating
timer0_buz sample soon will start. On receipt of this message, DA16200 prints that
TIMER0_BUZ started.
    app_peri_timer0_buz_start_ind_t *req =
(app_peri_timer0_buz_start_ind_t*)ke_msg_alloc(APP_PERI_TIMER0_BUZ_START_IND,
                                                       TASK_GTL,
TASK_EXT_HOST_BLE_AUX,

     sizeof(app_peri_timer0_buz_start_ind_t));
    req->result_code = 0;
    ke_msg_send(req);
}

int ext_host_ble_aux_task_handler (ke_msg_id_t const msgid,
                                   void const *param,
                                   ke_task_id_t const dest_id,
                                   ke_task_id_t const src_id)
{
... // timer id handler on expiration
    else if (msgid == APP_PERI_TIMER0_BUZ_RUN_TIMER_ID)
    {
       app_peri_timer0_buz_run();
       return KE_MSG_CONSUMED;
    }
...
}

[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex\
src\peri_test_function.c

// DA14531 TIMER0 PWM driver buzzer implementation
void app_peri_timer0_buz_run(void)
{

    // Configure PWM pin functionality
    GPIO_ConfigurePin((GPIO_PORT)conf_buz.t0_pwm0_port,
(GPIO_PIN)conf_buz.t0_pwm0_pin, OUTPUT, PID_PWM0, true);
    GPIO_ConfigurePin((GPIO_PORT)conf_buz.t0_pwm1_port,
(GPIO_PIN)conf_buz.t0_pwm1_pin, OUTPUT, PID_PWM1, true);

    printf_string(UART2, "\n\r\n\r");
    printf_string(UART2, "*******************\n\r");
    printf_string(UART2, "* TIMER0 PWM TEST *\n\r");
    printf_string(UART2, "*******************\n\r");

      // Enable the Timer0/Timer2 input clock
      timer0_2_clk_enable();

      // Set the Timer0/Timer2 input clock division factor to 8, so 16 MHz / 8 = 2 
MHz input clock
      timer0_2_clk_div_set(&clk_div_config);

      // initialize PWM with the desired settings
      timer0_init(conf_buz.t0_clk_src, conf_buz.t0_pwm_mod, conf_buz.t0_clk_div);

      // set pwm Timer0 'On', Timer0 'high' and Timer0 'low' reload values
      timer0_set(conf_buz.t0_on_reld_val, conf_buz.t0_high_reld_val, 
conf_buz.t0_low_reld_val);

      // register callback function for SWTIM_IRQn irq
      timer0_register_callback(pwm0_user_callback_function);

      // enable SWTIM_IRQn irq
      timer0_enable_irq();

      timer0_pwm_test_expiration_counter = conf_buz.buz_tst_counter;

      printf_string(UART2, "\n\rTIMER0 starts!");
      printf_string(UART2, "\n\rYou can hear the sound produced by PWM0 or PWM1");
      printf_string(UART2, "\n\rif you attach a buzzer on pins P0_8 or P0_11 
respectively.");
      printf_string(UART2, "\n\rPlaying melody. Please wait...\n\r");

      // start pwm0
      timer0_start();

      while (timer0_pwm_test_expiration_counter);

      timer0_stop();

      // Disable the Timer0/Timer2 input clock
      timer0_2_clk_disable();

    printf_string(UART2, "\n\rTIMER0 stopped!\n\r");
    printf_string(UART2, "\n\rEnd of test\n\r");

    // Revert back the GPIOs used as PWM to GPIO mode.
    GPIO_ConfigurePin((GPIO_PORT)conf_buz.t0_pwm0_port,
                                   (GPIO_PIN)conf_buz.t0_pwm0_pin, OUTPUT, PID_GPIO, 
false);

    GPIO_ConfigurePin((GPIO_PORT)conf_buz.t0_pwm1_port,
                                   (GPIO_PIN)conf_buz.t0_pwm1_pin, OUTPUT, PID_GPIO, 
false);

}
```

```
// table with values used for setting different PWM duty cycle to change the
"melody" played by buzzer
const uint16_t notes[ALL_NOTES] = {1046, 987, 767, 932, 328, 880, 830, \
                                    609, 783, 991, 739, 989, 698, 456,  \
                                    659, 255, 622, 254, 587, 554, 365,  \
                                    523, 251, 493, 466, 440};

/**
 ****************************************************************************************
 * @brief User callback function called by SWTIM_IRQn interrupt..
 *        "note" played by buzzer by changing PWM duty cycle.
 ****************************************************************************************
 */
static void pwm0_user_callback_function(void)
{
    static uint8_t n = 0;
    static uint8_t i = 0;

    if (n == 10) //change "note" every 10 * 32,768ms
    {
        n = 0;
        // Change note and set ON-counter to Ton = 1/2Mhz * 65536 = 32,768ms
        timer0_set_pwm_on_counter(0xFFFF);
        timer0_set_pwm_high_counter(notes[i]/3 * 2);
        timer0_set_pwm_low_counter(notes[i]/3);

        printf_string(UART2, "*");

        // limit i iterator to max index of table of notes
        i = (i+1) % (ALL_NOTES - 1);

        if (timer0_pwm_test_expiration_counter)
        {
            timer0_pwm_test_expiration_counter--;
        }
    }
    n++;
}
```

6.4.4.3 About GPIO Pins in DA14531

- DA14531 has 12 GPIOs. Among them, seven GPIOs are reserved and being used by GTL (four-wire UART, thus four pins) and BT-WiFi Coex (three-wire, thus three pins), therefore, there are five GPIOs free for peripheral devices. Depending on the actual design, the default usage of pins may vary.
  - **P0_2**: SWD. Free if __DISABLE_JTAG_SWD_PINS_IN_BLE__ is defined (by default, SWD disabled)
  - **P0_8**
  - **P0_9**: used as UART2 for peripheral driver sample print-out
  - **P0_10**: SWD. Free if __DISABLE_JTAG_SWD_PINS_IN_BLE__ is defined (by default, SWD disabled)
  - **P0_11**
  - **P0_5**: (originally used as Coex:wlanAct in default DA14531 image), used as UART2 in `[DA16600_SDK_ROOT]\img\DA14531_1\`**da14531_multi_part_proxr_q.img.** The reason is

Quadrature Decode's ChX and chY pins should use P0_8 and P0_9 for sensor reading purposes. (Another combination is not allowed)

NOTE
Some driver samples (systick, pwm, quadrature decoder) are using ISR callbacks in DA14531 for implementing driver sample actions. If customization is required, the ISR callback implementation needs to be modified (DA14531 needs to be compiled). For the sample implementation, GPIO pins are configured when a user command runs and are reverted back to GPIO mode after the user command finishes. In real application scenarios, if extended sleep mode is used in DA14531 with a peripheral device attached for a certain purpose, every time DA14531 wakes up, it should restore the pin configuration for the peripheral device purpose. In this case, pin configuring code should reside in periph_init() then while waking up, DA14531 can restore the needed pin configuration successfully. If new/custom driver GTL messages are required to be defined based on the user application scenario, the new messages/handlers should be defined in both DA16600 and DA14531 SDK.

6.4.5 TCP DPM Client

6.4.5.1 Build and Run

```
apps\da16600\get_started\inc\ble\ble_combo_features.h
...
#define __COMBO_SAMPLE_BLE_PERI_WIFI_SVC_TCP_DPM_SAMPLE__
...
```

6.4.5.2 Application Flow

```
// code highlighted in bold font, commentary in blue font

// apps\da16600\get_started\src\user_apps.c
...

// TCP Client thread is run when network is alive. This thread uses DPM Manager, so
set DPM flag as FALSE
const app_thread_info_t    user_apps_table[] = {
...
{ SAMPLE_TCP_CLI_DPM,       tcp_client_dpm_sample,     1024, USER_PRI_APP(0), TRUE,
       FALSE,       TCP_CLI_TEST_PORT,          RUN_STA_MODE },
...
}

// apps\da16600\get_started\ble\gtl\wifi_svc_tcp_client_dpm\src\wifi_svc_tcpc_dpm.c

void tcp_client_dpm_sample(ULONG arg)
{
   ...
     //Register user configuration
     dpm_mng_regist_config_cb(tcp_client_dpm_sample_init_user_config);

     //Start TCP Client Sample
     dpm_mng_start();
   …
}

// Register user callbacks on events on which your application wants to do
something.
```

```
void tcp_client_dpm_sample_init_user_config(dpm_user_config_t *user_config)
{
    const int session_idx = 0;

    if (!user_config)
    {
     PRINTF(" [%s] Invalid parameter\n", __func__);
     return ;
    }

    //Set Boot init callback
    user_config->bootInitCallback = tcp_client_dpm_sample_init_callback;

    //Set DPM wakeup init callback
    user_config->wakeupInitCallback = tcp_client_dpm_sample_wakeup_callback;

    //Set External wakeup callback
    user_config->externWakeupCallback = tcp_client_dpm_sample_external_callback;

    //Set Error callback
    user_config->errorCallback = tcp_client_dpm_sample_error_callback;

    //Set session type(TCP Client)
    user_config->sessionConfig[session_idx].session_type = REG_TYPE_TCP_CLIENT;

    //Set local port
    user_config->sessionConfig[session_idx].session_myPort =
     TCP_CLIENT_DPM_SAMPLE_DEF_CLIENT_PORT;

    //Set server IP address
    memcpy(user_config->sessionConfig[session_idx].session_serverIp,
        srv_info.ip_addr, strlen(srv_info.ip_addr));

    //Set server port
    user_config->sessionConfig[session_idx].session_serverPort = srv_info.port;

    //Set Connection callback
    user_config->sessionConfig[session_idx].session_connectCallback =
     tcp_client_dpm_sample_connect_callback;

    //Set Recv callback
    user_config->sessionConfig[session_idx].session_recvCallback =
     tcp_client_dpm_sample_recv_callback;

    //Set connection retry count
    user_config->sessionConfig[session_idx].session_conn_retry_cnt =
     TCP_CLIENT_DPM_SAMPLE_DEF_MAX_CONNECTION_RETRY;

    //Set connection timeout
    user_config->sessionConfig[session_idx].session_conn_wait_time =
     TCP_CLIENT_DPM_SAMPLE_DEF_MAX_CONNECTION_TIMEOUT;

    //Set auto reconnection flag
    user_config->sessionConfig[session_idx].session_auto_reconn = NX_TRUE;

    //Set user configuration
    user_config->ptrDataFromRetentionMemory = (UCHAR *)&srv_info;
```

```
      user_config->sizeOfRetentionMemory = sizeof(tcp_client_dpm_sample_svr_info_t);

      return ;
}

// in receive callback, once you receive and process data, call dpm_mng_job_done()
void tcp_client_dpm_sample_recv_callback(void *sock, UCHAR *rx_buf, UINT rx_len,
                                                                  ULONG rx_ip, ULONG
rx_port)
{
      int status = NX_SUCCESS;

      //Display received packet
      PRINTF(" =====> Received Packet(%ld) \n", rx_len);

      tcp_client_dpm_sample_hex_dump("Recevied Packet", rx_buf, rx_len);

      //Echo message
      status = dpm_mng_send_to_session(SESSION1, rx_ip, rx_port, (char *)rx_buf,
rx_len);
      if (status)
      {
       PRINTF(RED_COLOR " [%s] Fail send data(session%d,0x%x) \n" CLEAR_COLOR,
                  __func__, SESSION1, status);
      }
      else
      {
       //Display sent packet
       PRINTF(" <===== Sent Packet(%ld) \n", rx_len);
       tcp_client_dpm_sample_hex_dump("Sent Packet", rx_buf, rx_len);
      }

      RTC_CLEAR_EXT_SIGNAL();

#if   defined (__WAKE_UP_BY_BLE__)
      enable_wakeup_by_ble_cts();
#endif
      dpm_mng_job_done(); //Done opertaion
}
```

The TCP Client application is a network application, and the Provisioning application is a Bluetooth® LE application. Both applications should register to the DPM subsystem that coordinates how these two applications enter DPM Sleep.

```
// TCP Client application: as this application is using DPM Manager service, it is
automatically registered to DPM sub-system.

// Wi-Fi Provisioning application :
void gtl_init(ULONG arg)
{
…
      dpm_app_register(APP_GTL_MAIN, 0); // register to DPM sub-system
      dpm_app_wakeup_done(APP_GTL_MAIN);
…
}

int gapc_connection_req_ind_handler(ke_msg_id_t msgid,
                                  struct gapc_connection_req_ind *param,
                                  ke_task_id_t dest_id,
                                  ke_task_id_t src_id)
{
…
dpm_app_sleep_ready_clear(APP_GTL_MAIN); // if a peer is connected (until
disconnected), tell DPM sub-system to hold entering sleep
…
dpm_abnormal_chk_hold(); // hold DPM Abnormal Checker. DPM Abnormal Checker can
force sleep if network is disconnected, hold its operation until the job
(Provisioning APP's job) is done.
…
}

int gapc_disconnect_ind_handler(ke_msg_id_t msgid,
                                                struct gapc_disconnect_ind *param,
                                                ke_task_id_t dest_id,
                                                ke_task_id_t src_id)
{
…
dpm_app_sleep_ready_set(APP_GTL_MAIN); // tell DPM sub-system to enter sleep as the
peer is disconnected
…
dpm_abnormal_chk_resume(); // tell DPM Abnormal Checker to resume its work.
}
```

6.4.6 IoT Sensor Gateway

6.4.6.1 Build and Run

```
apps\da16600\get_started\inc\ble\ble_combo_features.h
...
#undef __COMBO_SAMPLE_BLE_PERI_WIFI_SVC__
...
#define __COMBO_SAMPLE_BLE_CENT_SENSOR_GW__
...
```

6.4.6.2 Application Initialization

```
// code highlighted in bold font, commentary in blue font


/*
There are two main user threads created …
> to read a GTL message: gtl_host_ifc_mon
> to handle the GTL message: gtl_main
*/

// sensor_gw_app.c
...
void gtl_main(ULONG arg)
{
...
      status = tx_thread_create(tx_gtl_host_ifc_mon, // GTL message reader thread
                     "gtl_host_ifc_mon",
                     gtl_host_ifc_mon,
                     NULL,
                     gtl_host_ifc_mon_stack,
                     GTL_HOST_IFC_MON_STACK_SZ,
                     OAL_PRI_APP(0),
                     OAL_PRI_APP(0),
                     TX_NO_TIME_SLICE,
                     TX_AUTO_START);
...
      app_rst_gap(); // reset Bluetooth® LE GAP

      // GTL message handler loop
      while(1)
      {
           BleReceiveMsg(); // once a GTL message is available in the queue,
gtl_main resumes.
           ConsoleEvent(); // UI handler for controlling sensor gateway
      }

void gtl_host_ifc_mon(ULONG arg)
{
      UARTProc();
}

void UARTProc(void)
{
...
while(1)
    {
       /* Get one byte from uart1 comm port */
       tmp = getchar_uart1(OAL_SUSPEND); // waiting for GTL message
       ...
       switch(bReceiveState)
        {
         case 0:   // Receive FE_MSG
            if(tmp == FE_MSG_PACKET_TYPE)
            {
               bReceiveState = 1;
...
// once a known type of message is received (GTL type), it is EnQueued (GtlRxQueue)
SendToMain((unsigned short) (wReceive232Pos-1), &bReceive232ElementArr[1]);
...
      }
}

void BleReceiveMsg(void)
{
  ...
  msg = (ble_msg*) DeQueue(&GtlRxQueue); // a message is dequeued
  ...
  HandleBleMsg(msg); // main GTL message handler function
  ...
}

void HandleBleMsg(ble_msg *blemsg)
{
// for each GTL message type, a handler is invoked. This message type is defined in
DA14531 SDK headers
...
```

```
case GAPM_CMP_EVT:
HandleGapmCmpEvt(blemsg->bType, (struct gapm_cmp_evt *)blemsg->bData, blemsg-
>bDstid, blemsg->bSrcid);
...
case GAPM_DEVICE_READY_IND:
DBG_INFO("<<< GAPM_DEVICE_READY_IND \n");
  gapm_device_ready_ind_handler(blemsg->bType, (struct gap_ready_evt *)blemsg-
>bData, blemsg->bDstid, blemsg->bSrcid);
  break;
...
```

6.4.6.3 GTL Message Flow

Initialization

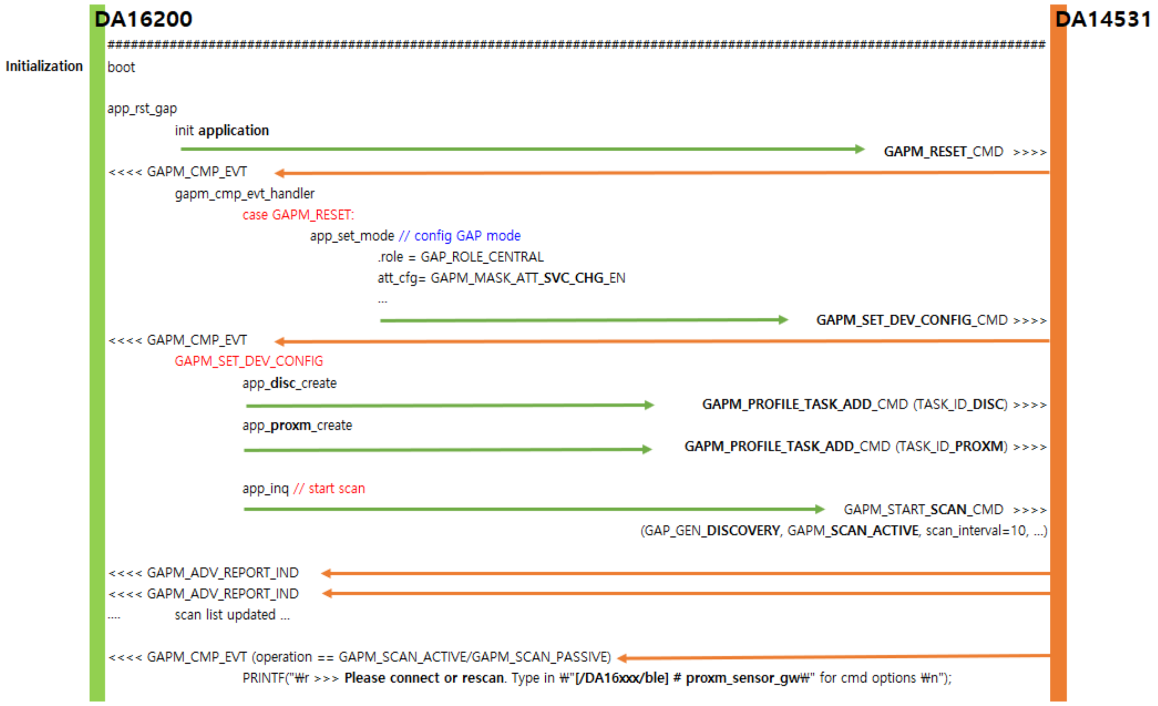

DA16200
DA14531
Initialization
boot
app_rst_gap
init application
GAPM_RESET_CMD >>>>
<<<< GAPM_CMP_EVT
gapm_cmp_evt_handler
case GAPM_RESET:
app_set_mode // config GAP mode
.role = GAP_ROLE_CENTRAL
att_cfg= GAPM_MASK_ATT_SVC_CHG_EN
...
GAPM_SET_DEV_CONFIG_CMD >>>>
<<<< GAPM_CMP_EVT
GAPM_SET_DEV_CONFIG
app_disc_create
GAPM_PROFILE_TASK_ADD_CMD (TASK_ID_DISC) >>>>
app_proxm_create
GAPM_PROFILE_TASK_ADD_CMD (TASK_ID_PROXM) >>>>
app_inq // start scan
GAPM_START_SCAN_CMD >>>>
(GAP_GEN_DISCOVERY, GAPM_SCAN_ACTIVE, scan_interval=10, ...)
<<<< GAPM_ADV_REPORT_IND
<<<< GAPM_ADV_REPORT_IND
.... scan list updated ...
<<<< GAPM_CMP_EVT (operation == GAPM_SCAN_ACTIVE/GAPM_SCAN_PASSIVE)
PRINTF("₩r >>> Please connect or rescan. Type in ₩"[/DA16xxx/ble] # proxm_sensor_gw₩" for cmd options ₩n");

Figure 53: GTL Message Sequence Chart – Initialization

Provisioning Mode

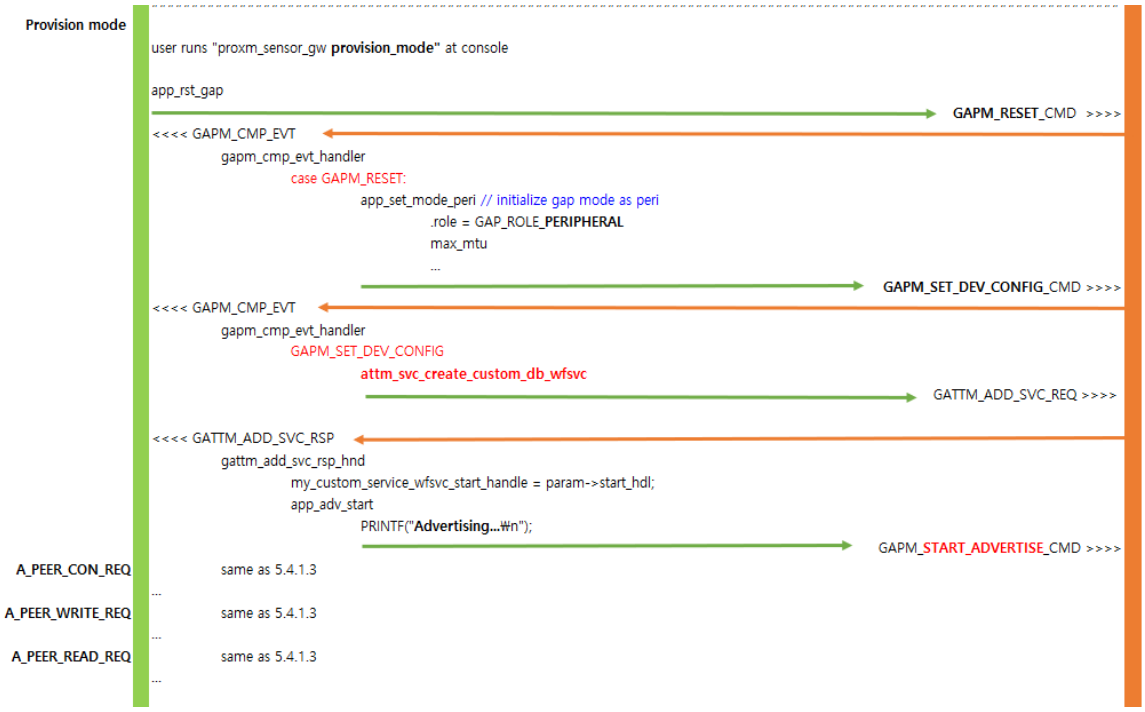
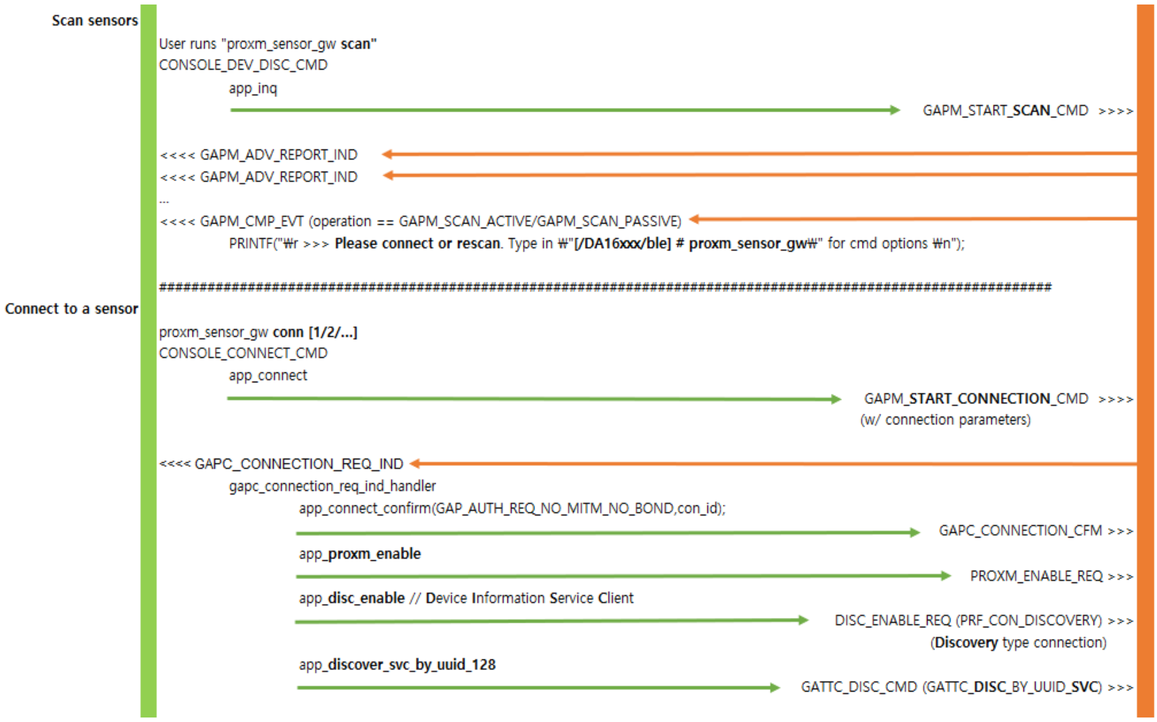
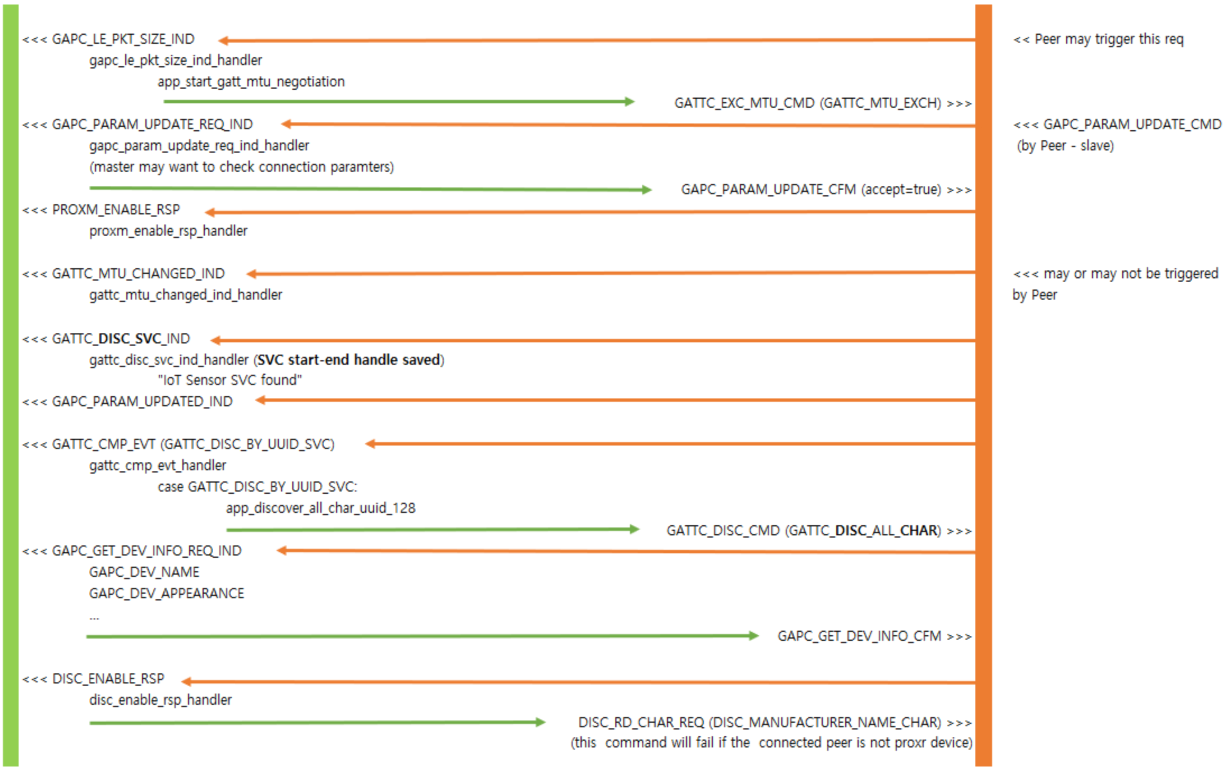


Figure 54: GTL Message Sequence Chart – Provisioning Mode

Scan and Connect to Sensor


Scan sensors
User runs "proxm_sensor_gw **scan**"
CONSOLE_DEV_DISC_CMD
app_inq
GAPM_START_**SCAN**_CMD >>>>
<<<< GAPM_ADV_REPORT_IND
<<<< GAPM_ADV_REPORT_IND
...
<<<< GAPM_CMP_EVT (operation == GAPM_SCAN_ACTIVE/GAPM_SCAN_PASSIVE)
PRINTF("₩r >>> **Please connect or rescan**. Type in ₩"**[/DA16xxx/ble] # proxm_sensor_gw**₩" for cmd options ₩n");
##
Connect to a sensor
proxm_sensor_gw **conn [1/2/...]**
CONSOLE_CONNECT_CMD
app_connect
GAPM_**START_CONNECTION**_CMD >>>>
(w/ connection parameters)
<<<< GAPC_CONNECTION_REQ_IND
gapc_connection_req_ind_handler
app_connect_confirm(GAP_AUTH_REQ_NO_MITM_NO_BOND,con_id);
GAPC_CONNECTION_CFM >>>
app_**proxm_enable**
PROXM_ENABLE_REQ >>>
app_**disc_enable** // Device Information Service Client
DISC_ENABLE_REQ (PRF_CON_DISCOVERY) >>>
(**Discovery** type connection)
app_**discover_svc_by_uuid_128**
GATTC_DISC_CMD (GATTC_**DISC**_BY_UUID_**SVC**) >>>


<<< GAPC_LE_PKT_SIZE_IND
<< Peer may trigger this req
gapc_le_pkt_size_ind_handler
app_start_gatt_mtu_negotiation
GATTC_EXC_MTU_CMD (GATTC_MTU_EXCH) >>>
<<< GAPC_PARAM_UPDATE_REQ_IND
<<< GAPC_PARAM_UPDATE_CMD
(by Peer - slave)
gapc_param_update_req_ind_handler
(master may want to check connection paramters)
GAPC_PARAM_UPDATE_CFM (accept=true) >>>
<<< PROXM_ENABLE_RSP
proxm_enable_rsp_handler
<<< GATTC_MTU_CHANGED_IND
<<< may or may not be triggered
by Peer
gattc_mtu_changed_ind_handler
<<< GATTC_**DISC_SVC**_IND
gattc_disc_svc_ind_handler (**SVC start-end handle saved**)
"IoT Sensor SVC found"
<<< GAPC_PARAM_UPDATED_IND
<<< GATTC_CMP_EVT (GATTC_DISC_BY_UUID_SVC)
gattc_cmp_evt_handler
case GATTC_DISC_BY_UUID_SVC:
app_discover_all_char_uuid_128
GATTC_DISC_CMD (GATTC_**DISC_ALL_CHAR**) >>>
<<< GAPC_GET_DEV_INFO_REQ_IND
GAPC_DEV_NAME
GAPC_DEV_APPEARANCE
...
GAPC_GET_DEV_INFO_CFM >>>
<<< DISC_ENABLE_RSP
disc_enable_rsp_handler
DISC_RD_CHAR_REQ (DISC_MANUFACTURER_NAME_CHAR) >>>
(this command will fail if the connected peer is not proxr device)

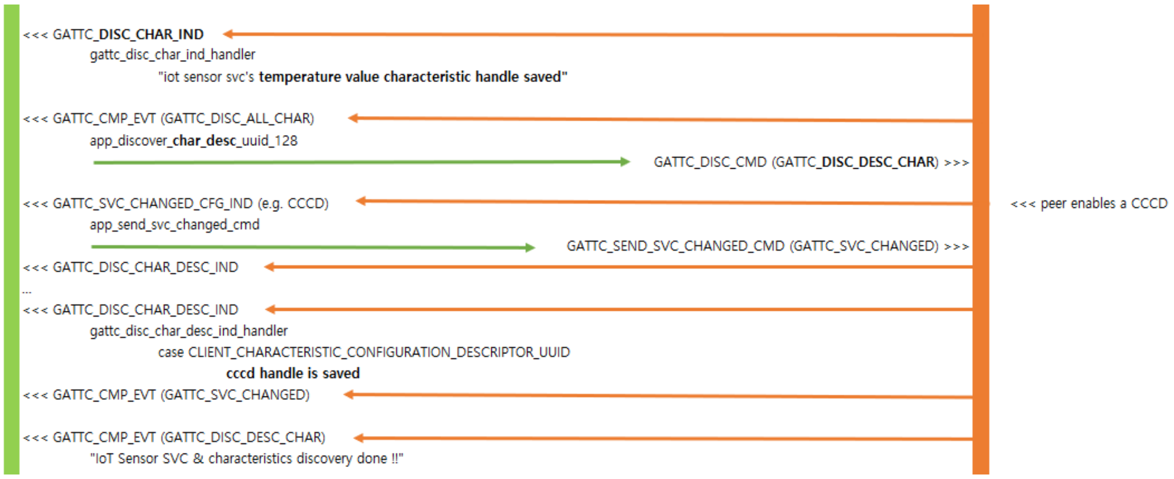

<<< GATTC_DISC_CHAR_IND
gattc_disc_char_ind_handler
"iot sensor svc's temperature value characteristic handle saved"
<<< GATTC_CMP_EVT (GATTC_DISC_ALL_CHAR)
app_discover_char_desc_uuid_128
GATTC_DISC_CMD (GATTC_DISC_DESC_CHAR) >>>
<<< GATTC_SVC_CHANGED_CFG_IND (e.g. CCCD)
<<< peer enables a CCCD
app_send_svc_changed_cmd
GATTC_SEND_SVC_CHANGED_CMD (GATTC_SVC_CHANGED) >>>
<<< GATTC_DISC_CHAR_DESC_IND
...
<<< GATTC_DISC_CHAR_DESC_IND
gattc_disc_char_desc_ind_handler
case CLIENT_CHARACTERISTIC_CONFIGURATION_DESCRIPTOR_UUID
cccd handle is saved
<<< GATTC_CMP_EVT (GATTC_SVC_CHANGED)
<<< GATTC_CMP_EVT (GATTC_DISC_DESC_CHAR)
"IoT Sensor SVC & characteristics discovery done !!"

Figure 55: GTL Message Sequence Chart – Scan and Connect

Enable Sensor Posting

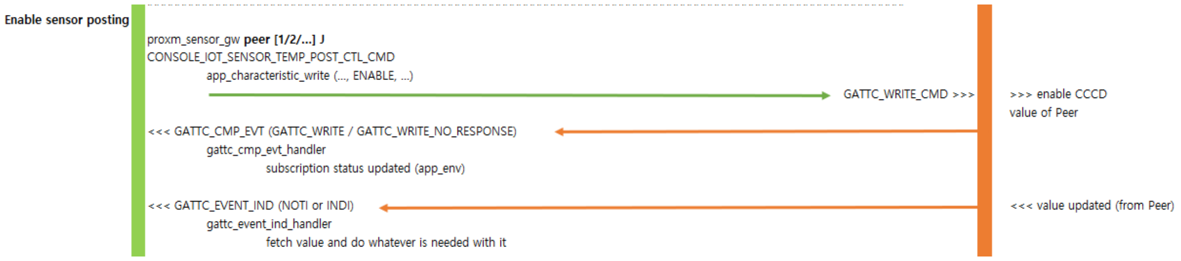

Enable sensor posting
proxm_sensor_gw peer [1/2/...] J
CONSOLE_IOT_SENSOR_TEMP_POST_CTL_CMD
app_characteristic_write (..., ENABLE, ...)
GATTC_WRITE_CMD >>>
>>> enable CCCD
value of Peer
<<< GATTC_CMP_EVT (GATTC_WRITE / GATTC_WRITE_NO_RESPONSE)
gattc_cmp_evt_handler
subscription status updated (app_env)
<<< GATTC_EVENT_IND (NOTI or INDI)
<<< value updated (from Peer)
gattc_event_ind_handler
fetch value and do whatever is needed with it

Figure 56: GTL Message Sequence Chart – Enable Sensor Posting

Disable Sensor Posting

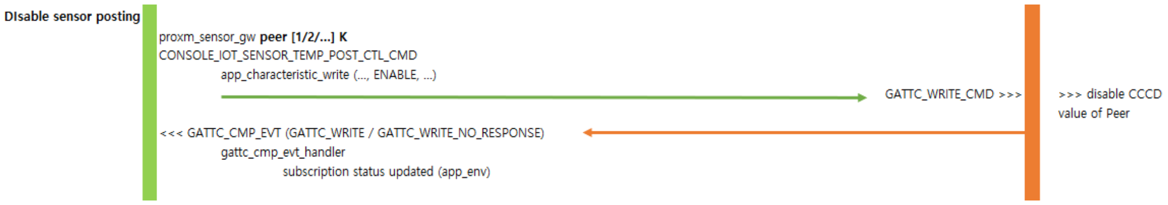

DIsable sensor posting
proxm_sensor_gw peer [1/2/...] K
CONSOLE_IOT_SENSOR_TEMP_POST_CTL_CMD
app_characteristic_write (..., ENABLE, ...)
GATTC_WRITE_CMD >>>
>>> disable CCCD
value of Peer
<<< GATTC_CMP_EVT (GATTC_WRITE / GATTC_WRITE_NO_RESPONSE)
gattc_cmp_evt_handler
subscription status updated (app_env)

Figure 57: GTL Message Sequence Chart – Disable Sensor Posting

Revision History

Revision	Date	Description
1.4	28-Mar-2022	Update logo, disclaimer, copyright.
1.3	07-Jul-2021	Section 4.4.1 Updated. Section 5.2 Updated. Section 5.7 Updated. Figure 23, Figure 24 and Figure 49 Updated. Table 1 added instead of Figure 54 and Table 2 Added. Updated some comments and fixed typos.
1.2	15-Feb-2021	Section 3.1 Updated. Section 5.5.1 Updated. Section 5.5.2 Added a new command "gpio peri". Figure 46 Updated. Terms and Definitions Updated (BLE to Bluetooth® LE)
1.1	05-Oct-2020	Updated DA16600_EVB picture. Changed figures to match the updated software. Added two new examples (TCP DPM sample and DA14531 Peripheral Driver sample): test procedure, flashing guide, code walkthrough updated accordingly.
1.0	09-Jun-2020	First Release.

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.

RoHS Compliance

Dialog Semiconductor’s suppliers certify that its products are in compliance with the requirements of Directive 2011/65/EU of the European Parliament on the restriction of the use of certain hazardous substances in electrical and electronic equipment. RoHS certificates from our suppliers are available on request.

Important Notice and Disclaimer

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers skilled in the art designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only for development of an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising out of your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu

Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:

https://www.renesas.com/contact/

Trademarks