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 describes the SDK API and peripheral device drivers and interfaces.

Contents

 © 2020 Dialog Semiconductor

Figures

 © 2020 Dialog Semiconductor

 © 2020 Dialog Semiconductor

1 Terms and Definitions

API	Application Programming Interface
AP	Access Point
BLE	Bluetooth Low Energy
FW	Firmware
GTL	Generic Transport Layer
IDE	Integrated Development Environment
IoT	Internet of Things
SDK	Software Development Kit
SoC	System on Chip
USB	Universal Serial Bus

2 References

[1] UM-WI-002, DA16200 SDK Programmer Guide, User Manual, Dialog Semiconductor

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

[3] DA14531 Getting Started Guide (http://lpccs-docs.dialog-semiconductor.com/UM-B-117-DA14531-Getting-Started-With-The-Pro-Development-Kit/index.html)

[4] DA14531 SW Platform Reference (http://lpccs-docs.dialog-semiconductor.com/UM-B-119_DA14585-DA14531_SW_Platform_Reference/index.html)

 © 2020 Dialog Semiconductor

3 Introduction

The Dialog DA16600 module is also known as 'Combo'. Combo and DA16600 may be used interchangeably in this document. The Dialog DA16600 module is comprised of the DA16200 (Wi-Fi) and DA14531 (BLE) SoC. This document describes the test steps and code walkthrough of four Combo example applications.

3.1 DA16600 Module Evaluation Board

The Combo Module evaluation board (see Figure 1) has the following features:

- **A**: Combo (DA16200+DA14531) module
- **B**: USB port to power up the Combo module
- **C**: USB port to use the Keil IDE (JTAG) for the DA14531
- **D**: JTAG connector to use the I-JET debugger for the DA16200
- **E**: RTC_PWR_KEY switch to power on/off DA16200. This switch does not switch on/off DA14531, therefore, for full reset, either type "reboot" or plug-out/in USB port B
- **F**: RTC_WAKE_UP switch to wake up DA16200 during sleep
- **G**: SW3, PIN configuration for DA16200/DA14531
- **H**: SW7, PIN configuration for DA16200/DA14531

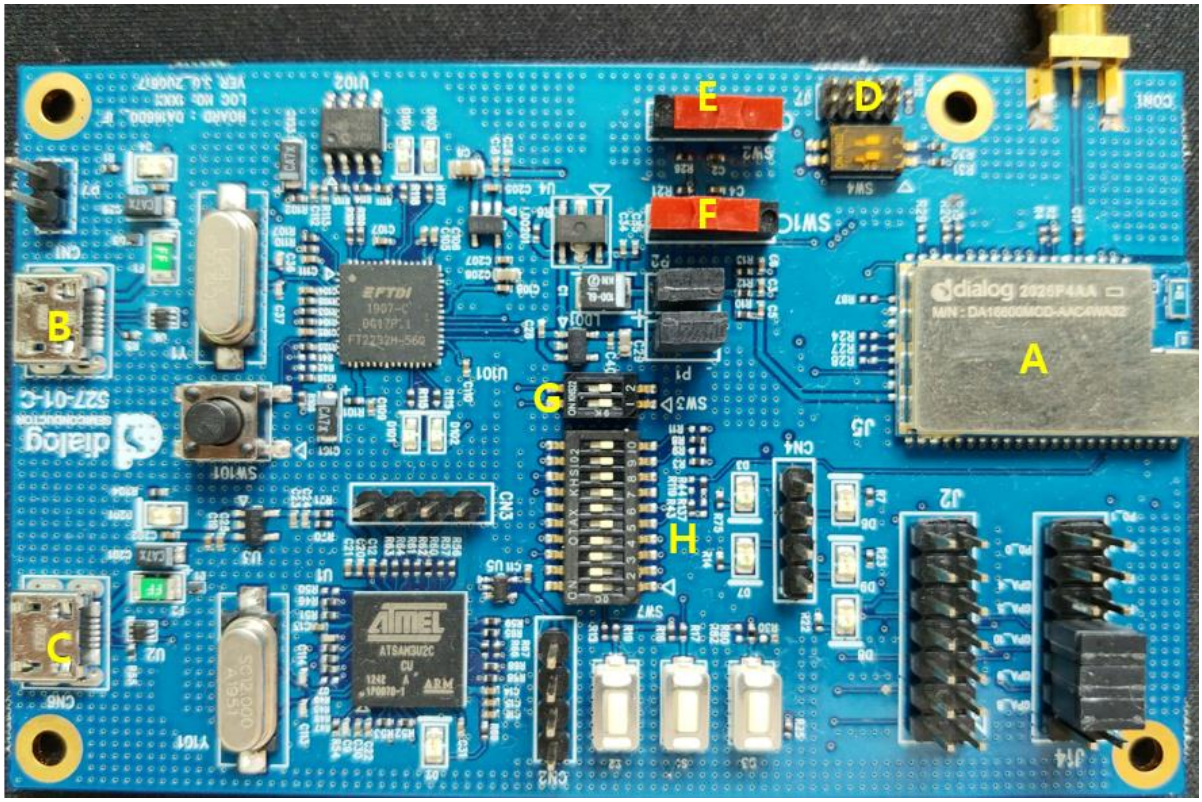


Figure 1: DA16600 (Combo) EVB

© 2020 Dialog Semiconductor

3.2 Example 1: BLE Assisted Wi-Fi Provisioning

In this example, the Combo module may be used in a product such as a "Wi-Fi door lock" where Wi-Fi plays the main role, and BLE assists with "Wi-Fi Provisioning" for the product's initial setup (Out-of-Box). A BLE peer provisioning mobile application (e.g. Android / IOS mobile App) needs to be developed for this example (see Section 6.4.1.5), which allows users to configure Wi-Fi (i.e. Wi-Fi Home router's SSID, password, server information, etc) for the Combo module.

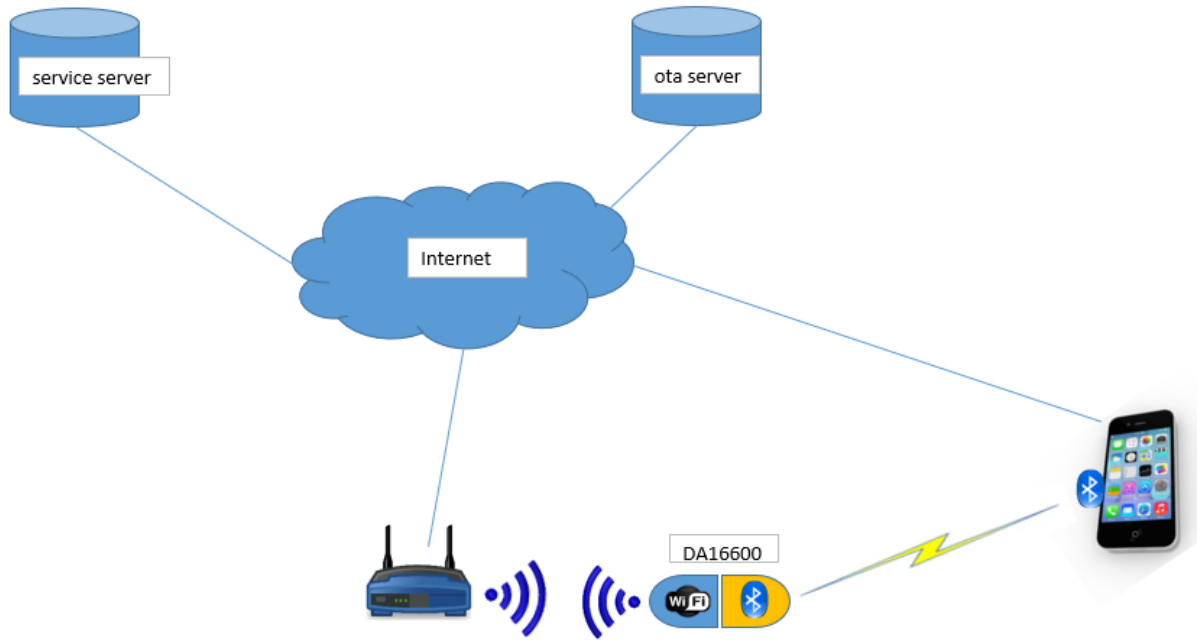


Figure 2: BLE Assisted Wi-Fi Provisioning

3.3 Example 2: BLE Firmware OTA Download via Wi-Fi

After Wi-Fi is successfully configured (provisioned) and up and running (in operational mode, for example the Wi-Fi door lock function is running), the Combo device can receive notifications (from a service server on the Internet/Cloud) that there is new firmware (Wi-Fi firmware or BLE firmware). Upon receipt (via Wi-Fi), the Combo module can download the new firmware from an OTA server and store the data in its flash memory and trigger the firmware update.

© 2020 Dialog Semiconductor

3.4 Example 3: Gas Leak Detection Sensor

This example assumes a Combo module that interacts with a standalone Gas Leak Detection Sensor.

This virtual gas density check sensor is attached to a BLE chip (for sensing work in low power mode) that periodically (at whatever specified interval a user wants) checks the gas density level. When a certain gas density level value is reached, BLE wakes up Wi-Fi, creates a "gas leak" event and sends the event to the Wi-Fi chip (provisioned), which then posts the "leak" event to a cloud server that notifies a user of the event. See Figure 3.

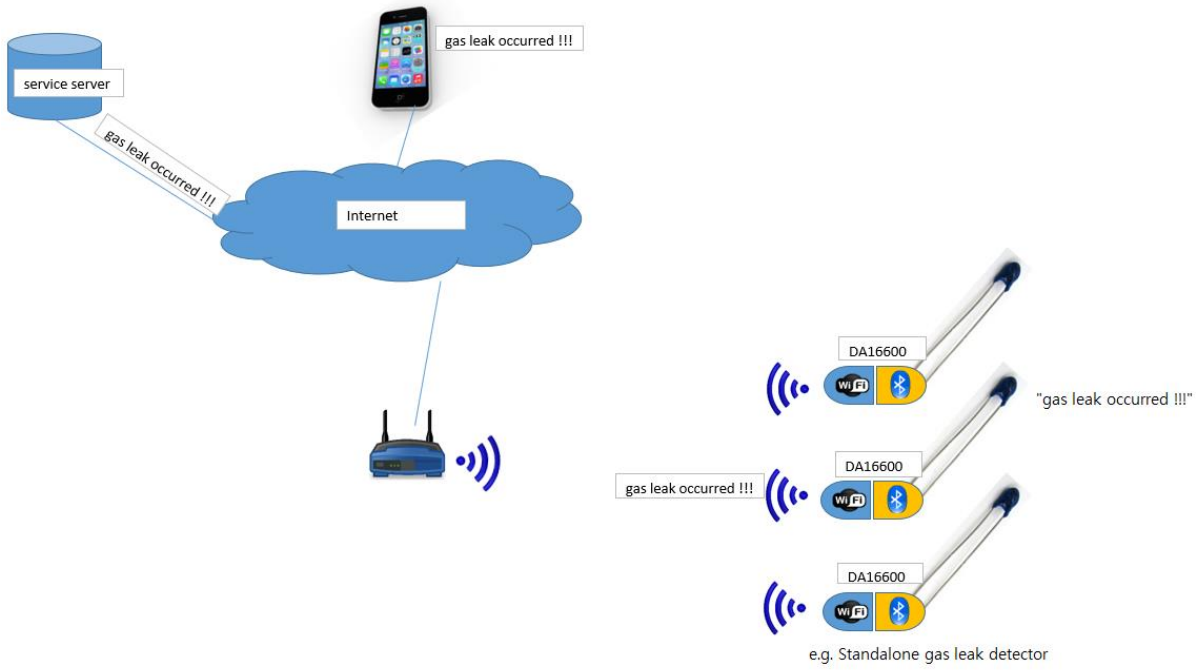


Figure 3: Standalone Gas Leak Detection Sensor

3.5 Example 4: DA14531 Peripheral Driver Examples

This example shows how to use peripheral devices that are connected to DA14531 from DA16200.

Programs in DA16200 can configure and run peripheral driver functions through GTL.

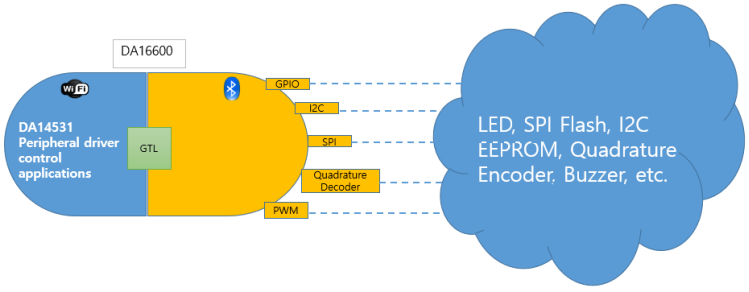


Figure 4: DA14531 Peripheral Device Control

© 2020 Dialog Semiconductor

3.6 Example 5: TCP DPM Client

In this example, DA16600 module runs TCP client in low power mode; DA16200 stays in DPM mode, and DA14531 stays in Extended sleep mode.

This example shows that DA16600 module in sleep mode can receive a Wi-Fi packet from a network peer or a BLE data from a BLE peer. After either Wi-Fi packet or BLE data is handled, DA16600 enters sleep mode again to save power.

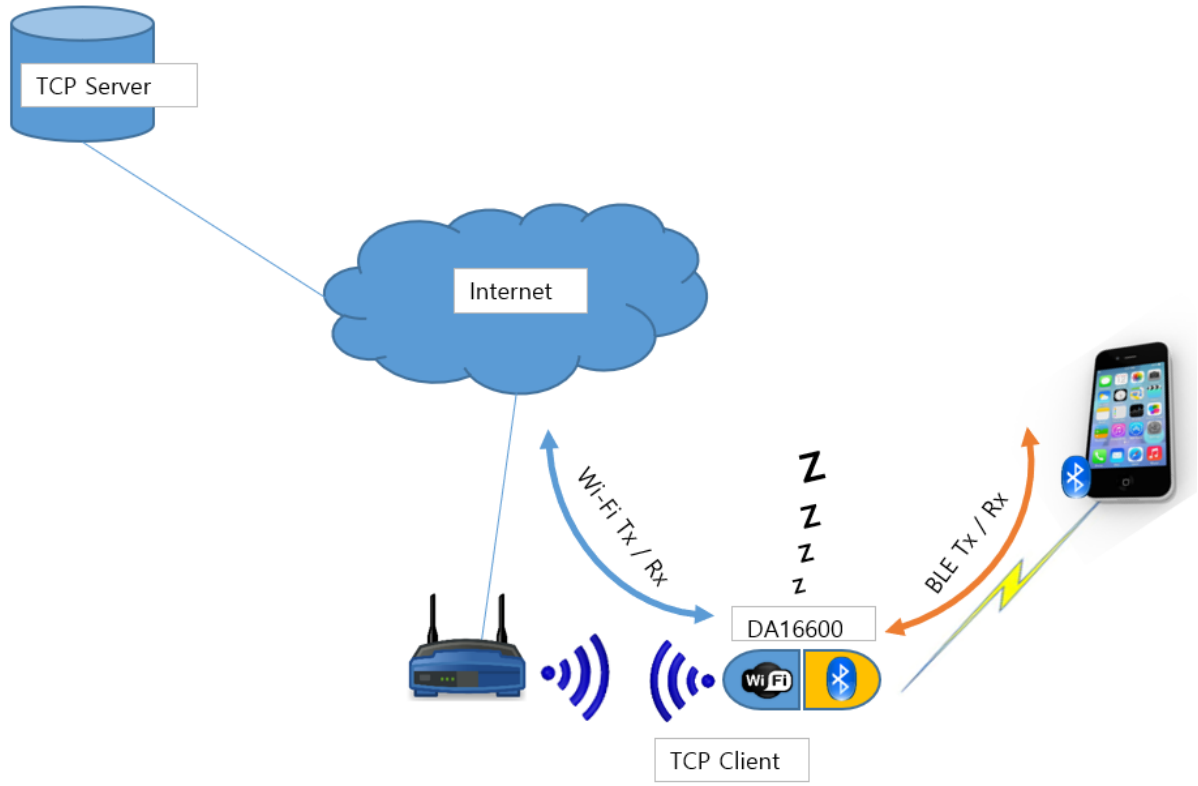


Figure 5: DA16600 DPM Communication

© 2020 Dialog Semiconductor

3.7 Example 6: IoT Sensor Gateway

In this example, the Combo module plays the role of a gateway device for multiple BLE temperature sensors.

A BLE sensor posts the current temperature value periodically via BLE's notify function. The BLE chip of the Combo module gathers the information and asks Wi-Fi to (periodically) post notifications to a service server in the cloud. See Figure 6.

NOTE
DA14531 can maintain a maximum of 3 connections.

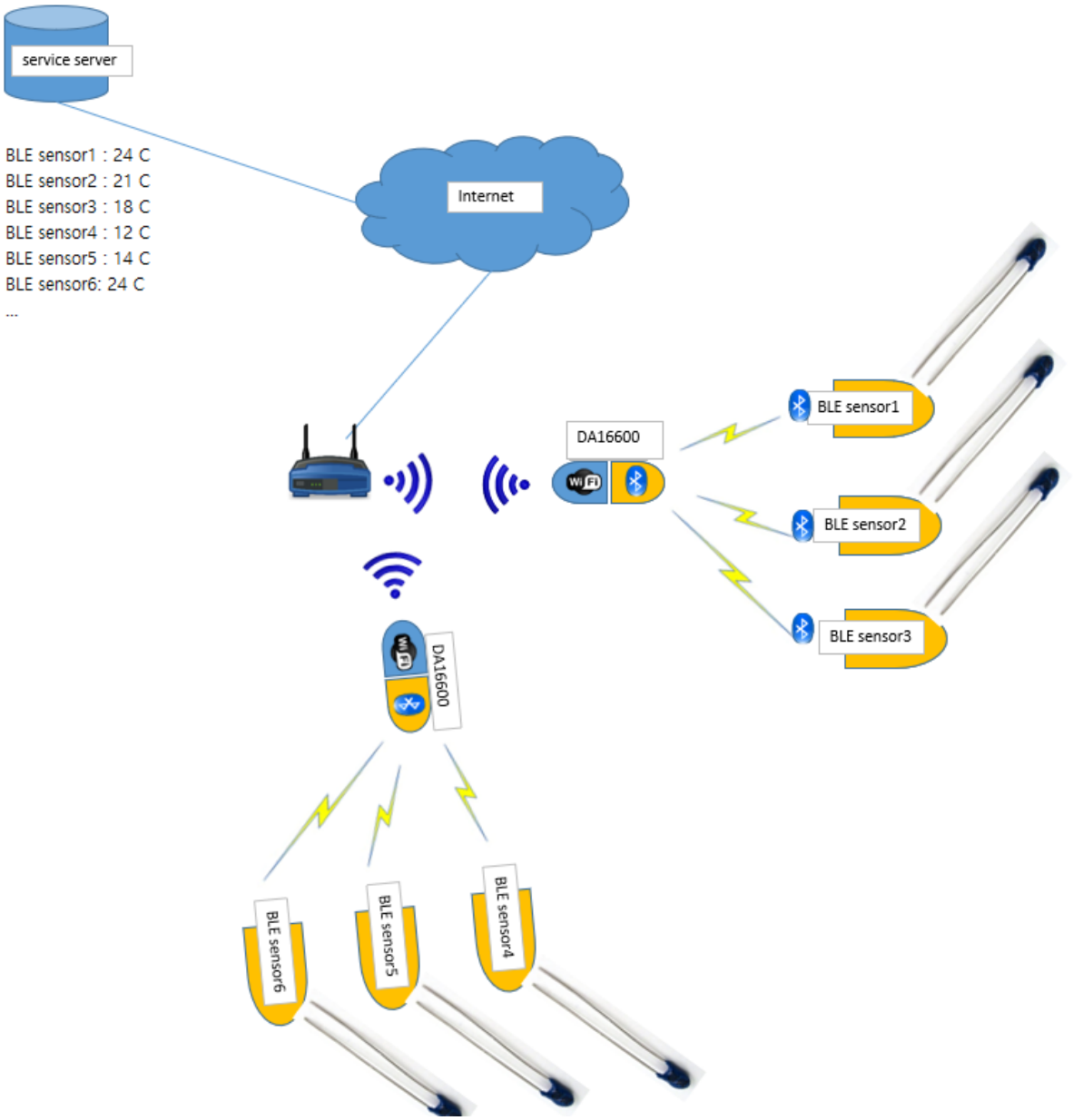


Figure 6: IoT Sensor Gateway

© 2020 Dialog Semiconductor

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 low power operation of DA16200)
  - **RTOS** image: main operation software into which user applications are built
- Code storage memory
  - sflash of DA16200

DA14531: BLE 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 example code below shows the sflash memory map defined for the DA16600 module.

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

#ifdef __FOR_4MB_SFLASH__

/*
 * 0x0000_0000      2nd Bootloader                        36 KB
 * 0x0000_9000      Boot Index                             4 KB
 * 0x0000_A000      RTOS #0                             1536 KB
 * 0x0018_A000      SLIB #0 ( RamLib + TIM )              64 KB
 * 0x0019_A000      User Area #0                         364 KB
 * 0x001F_5000      Debug/RMA Certificate                  4 KB
 * 0x001F_6000      TLS Certificate Key #0                16 KB
 * 0x001F_A000      TLS Certificate Key #1                16 KB
 * 0x001F_E000      NVRAM #0                               4 KB
 * 0x001F_F000      NVRAM #1                               4 KB
 *
 * 0x0020_0000      RTOS #1                       1536 KB
 * 0x0038_0000      SLIB #1 ( RamLib + TIM )        64 KB
 * 0x0039_0000      User Area #1                   448 KB
 */

  /* RTOS #0 */
  #define SFLASH_RTOS_0_BASE            0x0000A000    // RTOS #0
  /* SLIB #0 ( RamLib + TIM ) */
```


© 2020 Dialog Semiconductor

```
  #define SFLASH_CM_0_BASE                0x0018A000    // RamLib + TIM #0
  /* User Area #0 */
  #define SFLASH_USER_AREA_0_START        0x0019A000    // 364 KB
  /* Debug/RMA Certificate */
  #define SFLASH_RMA_CERTIFICATE  0x001F5000    // 4 KB

  /* TLS Certificate Key #0 */
  #define SFLASH_ROOT_CA_ADDR1            0x001F6000    // 16 KB
  #define SFLASH_CERTIFICATE_ADDR1 (SFLASH_ROOT_CA_ADDR1 + 0x1000)
  #define SFLASH_PRIVATE_KEY_ADDR1 (SFLASH_ROOT_CA_ADDR1 + 0x2000)
  #define SFLASH_DH_PARAMETER1            (SFLASH_ROOT_CA_ADDR1 + 0x3000)
  /* TLS Certificate Key #1 */
  #define SFLASH_ROOT_CA_ADDR2            0x001FA000    // 16 KB
  #define SFLASH_CERTIFICATE_ADDR2 (SFLASH_ROOT_CA_ADDR2 + 0x1000)
  #define SFLASH_PRIVATE_KEY_ADDR2 (SFLASH_ROOT_CA_ADDR2 + 0x2000)
  #define SFLASH_DH_PARAMETER2            (SFLASH_ROOT_CA_ADDR2 + 0x3000)
  /* NVRAM #0 */
  #define SFLASH_NVRAM_ADDR               0x001FE000    // 4KB
  /* NVRAM #0 */
  #define SFLASH_NVRAM_BACKUP             0x001FF000
  #define SFLASH_NVRAM_BACKUP_SZ   (SFLASH_NVRAM_BACKUP - SFLASH_NVRAM_ADDR)
  /* RTOS #1 */
  #define SFLASH_RTOS_1_BASE              0x00200000    // RTOS #1
  /* SLIB #1 ( RamLib + TIM ) */
  #define SFLASH_CM_1_BASE                0x00380000    // RamLib + TIM #1
  /* User Area #1 */
  #define SFLASH_USER_AREA_1_START        0x00390000    // 448 KB
  #define SFLASH_USER_AREA_1_END          0x003FD600

  /* User alias */
  #define SFLASH_USER_AREA_START          SFLASH_USER_AREA_0_START
  #define SFLASH_USER_AREA_END            SFLASH_RMA_CERTIFICATE
  #define SFLASH_USER_AREA_EXT_START      SFLASH_USER_AREA_1_START
  #define SFLASH_USER_AREA_EXT_END        SFLASH_USER_AREA_1_END

  /* Cert - Korea */
  #define SFLASH_ADDR_CERT_KR_MQTT        SFLASH_ROOT_CA_ADDR1
  #define SFLASH_ADDR_CERT_KR_HTTP        SFLASH_ROOT_CA_ADDR2
  /* Cert - Other */
  #define SFLASH_ADDR_CERT_OTHER_MQTT     SFLASH_USER_AREA_1_START                /*
0x00390000 */
  #define SFLASH_AREA_CERT_OTHER_HTTP     (SFLASH_ADDR_CERT_OTHER_MQTT + 0x1000) /*
0x00391000 */

 /* User Firmware */ ← BLE firmware is stored
  #define SFLASH_USER_FW_1         (SFLASH_AREA_CERT_OTHER_HTTP + 0x1000) /* -
0x00392000 ~ 0x00400000 */
```

The map has two image banks allocated for each type of DA16200 image (RTOS and SLIB), and also 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.

© 2020 Dialog Semiconductor

4.2 Build the DA16200 Software

In the DA16600 Combo SDK, use the IAR IDE to open the IAR workspace file (*[DA16600_SDK_ROOT]*\build\DA16xxx.eww).

For information on how to install and use the IAR IDE on your laptop, see the DA16200 SDK Programmer Guide [1] (Section 2.2), which is included in *[DA16600_SDK_ROOT]*\doc\.

You can configure the software before you start to build. Open the file "*[DA16600_SDK_ROOT]*\src\customer**config_combo_module_sdk.h**" to configure a Combo example application.

To test **Example 1**, **2**, or **3**, make the following changes in **config_combo_module_sdk.h** to build "DA16200_SW_1". If security needs to be enabled, then enable _WIFI_SVC_SECURITY__ as well ("DA16200_SW_1b"):

- DA16200_SW_1
  - Enable __**COMBO_SAMPLE_BLE_PERI_WIFI_SVC**__
  - Disable the another 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 BLE peer mobile application tries to connect to the DA16600 (with DA16200_SW_1b), a pairing procedure is requested.

There are two pairing modes 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 PINCODE 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 on the **OK** button to connect the DA16600 to your mobile application

The DA16600 example currently can save bond information for up to 10 BLE peers.

Once bond information is put in your mobile phone, your mobile phone's BLE 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

 © 2020 Dialog Semiconductor

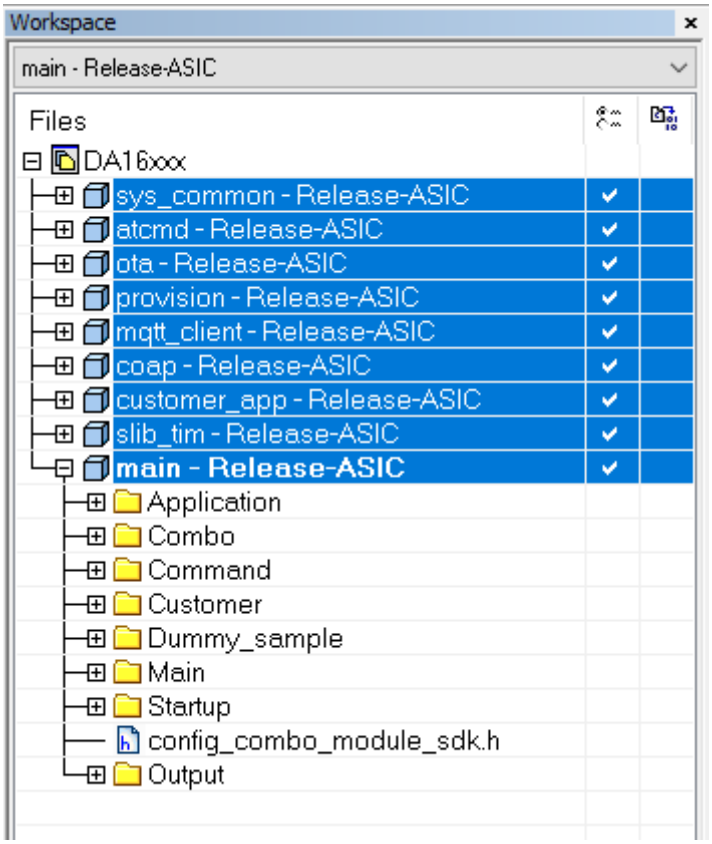


Figure 7: Project View

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 example, 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

 © 2020 Dialog Semiconductor

4.3 Build DA14531 Software

The DA14531 software (BLE 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 [*DA16600_SDK_ROOT*]\util\da14531_sdk*.zip.

Depending on which example you want to test, you need to use a different example BLE 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**.

© 2020 Dialog Semiconductor

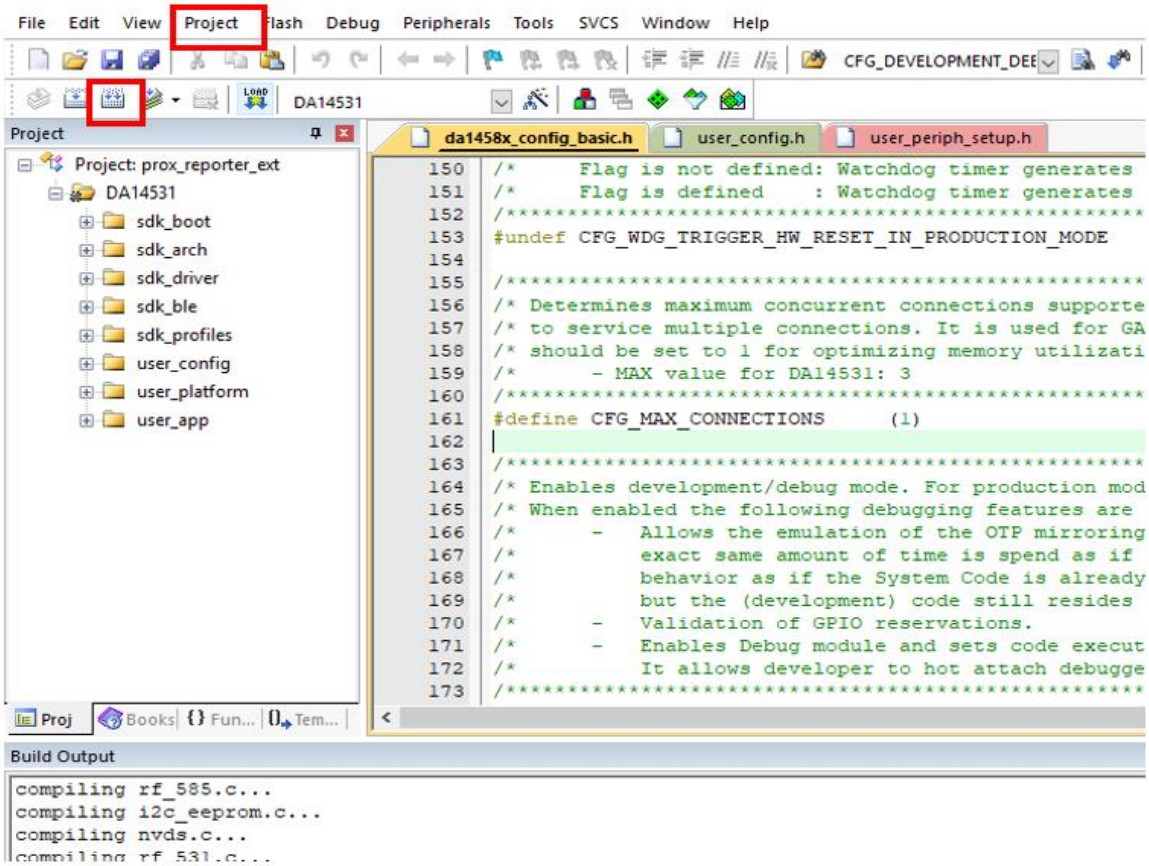


Figure 8. Keil - Build

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\".

© 2020 Dialog Semiconductor

4.4 Program sFlash and Run

4.4.1 Create a DA14531 Image from .bin

To create images you can use the tool named **mkimage.exe** available in the "*[DA16600_SDK_ROOT]\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 yourself. The tool mkimage.exe can be built in either GCC or MSVC environment. The mkimage tool source folder is: **[DA14531_SDK_ROOT]\utilities\mkimage**. For **GCC**: e.g. Linux, Cygwin, etc. [DA14531_SDK_ROOT]\utilities\mkimage\gcc : run "make" at the prompt. For **Windows**: e.g. 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 img 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.

© 2020 Dialog Semiconductor

4.4.1.2 Create a Multi-Part Image

You can create a multi-part 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 in your PC, you can create an image following these steps:

1. In Python 3.6.x, go to "[DA16600_SDK_ROOT]\util\ble_img_creator\" with the following files:
   - mkimage.exe
   - app_version.h
   - **mkimage_multi.py**
2. Copy the DA14531 bin file to "[DA16600_SDK_ROOT]\util\ble_img_creator\".
3. Open app_version.h, and change the version to your own 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_multi.py", and then follow the instruction.
   a. If you want to give your own name to .img, type it here, 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 yourself, 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 command prompt:

1. Create a folder (e.g. C:\temp_folder) and copy the following files in that folder:
   - [DA16600_SDK_ROOT]\util\ble_img_creator**mkimage.exe**
   - [DA16600_SDK_ROOT]\util\ble_img_creator**app_version.h**
   - [DA16600_SDK_ROOT]\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 "[DA16600_SDK_ROOT]\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 ("[DA16600_SDK_ROOT]\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.

© 2020 Dialog Semiconductor

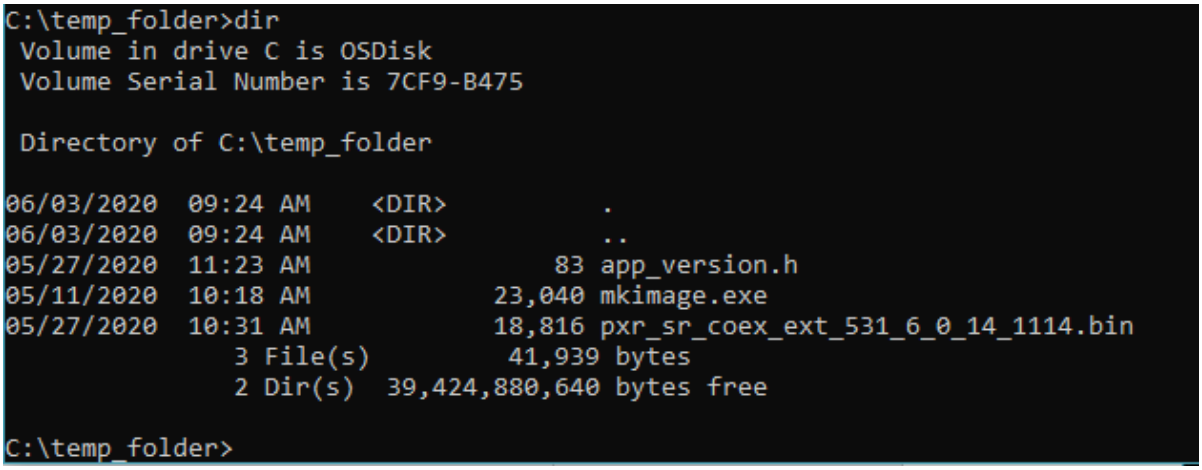


Figure 9. BLE_FW Image Generation - Preparation

3. At the command prompt, type the following commands in this 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.

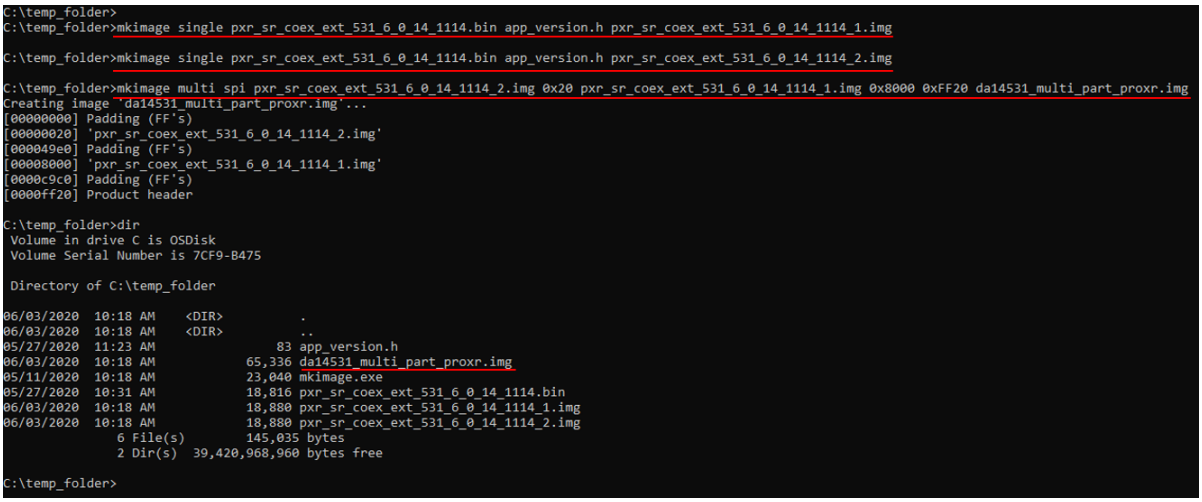


Figure 10. BLE_FW Image Generation - Multi-Part Image

© 2020 Dialog Semiconductor

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 in your PC, you can create an image following these steps:

1. In Python 3.6.x, go to "[DA16600_SDK_ROOT]\util\ble_img_creator\" with the following files:
   - mkimage.exe
   - app_version.h
   - **mkimage_single.py**
2. Copy the DA14531 bin file to "[DA16600_SDK_ROOT]\util\ble_img_creator\".
3. Open app_version.h, change the version to your own 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 own name to .img, type it here, 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 command prompt, do the following:

1. Create two copies of **app_version.h** and rename them: "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 this 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`.

© 2020 Dialog Semiconductor

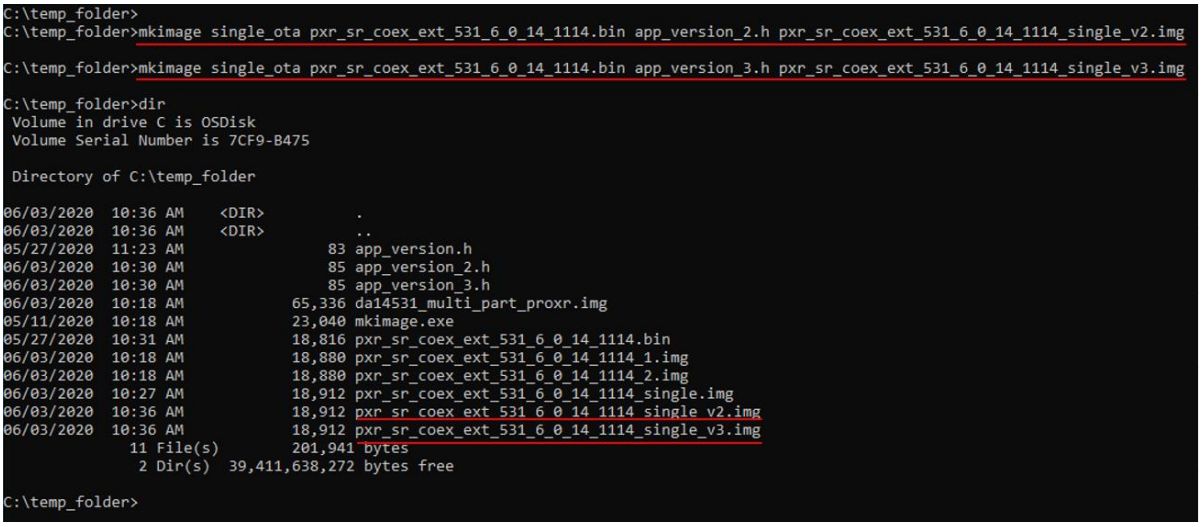


Figure 11. BLE_FW Image Generation - Single Image

4.4.2 Program sFlash

To program sFlash:

1. Make sure that all 4 images are ready:
   - DA16200_**BOOT**-GEN01-01-XXXXX-000000_W25Q32JW.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 Combo board (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 **Teraterm**.
   Windows will detect two USB ports (for example COM11 and COM12).
4. Connect Teraterm 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 command, copy the location path to the folder where the four Combo 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 (i.e. if you select a file too slow, the Teraterm'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** (see Figure 12).
    You can also use this shortcut key: keep the **Alt** key pressed and type **F**, **T**, **Y**, **S**.

© 2020 Dialog Semiconductor

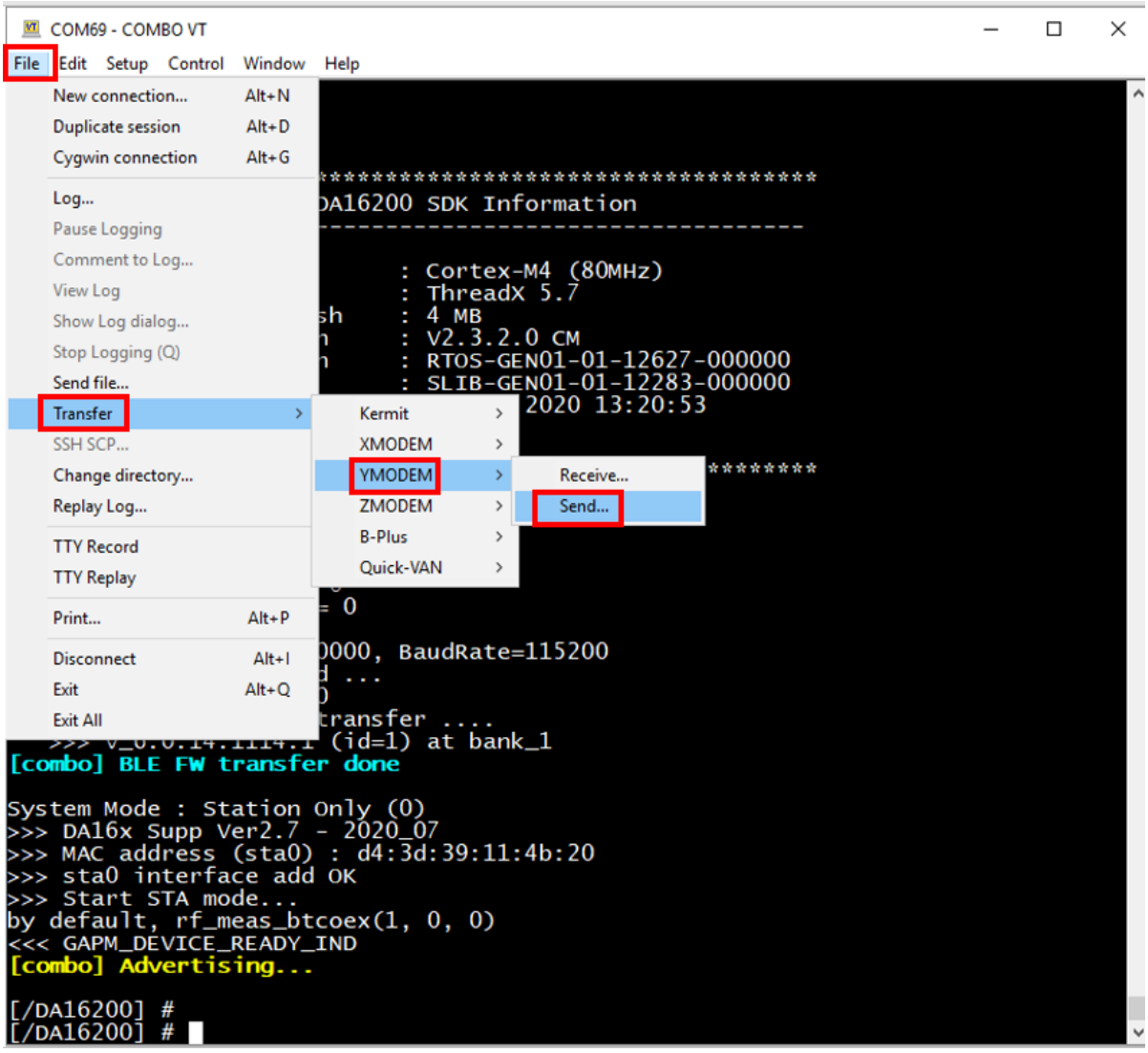


Figure 12: Teraterm

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 / BLE image) may not be correctly programmed.

14. After all 4 images are transferred to sflash, type `boot`, and then press **ENTER**.
    Some debug messages are printed in Teraterm.

© 2020 Dialog Semiconductor

15. Press **ENTER** several times until the prompt `[/DA16200] #` shows**.**
16. **Important**: **switch off** and **switch on** the USB port (if your 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.
17. Wait until your Teraterm is connected to COM11. (Or simply reconnect with serial. This step is not needed if you are not going to use Teraterm during the test.)

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

18. Now you are ready to do the test.

© 2020 Dialog Semiconductor

4.5 Run DA16600 with J-TAG

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 J-TAG to run DA16600. Because DA16600 has two chips - Wi-Fi (DA16200) and BLE (DA14531) - and each chip has its own J-TAG port.

4.5.1 Run DA16200 with J-TAG

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

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

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

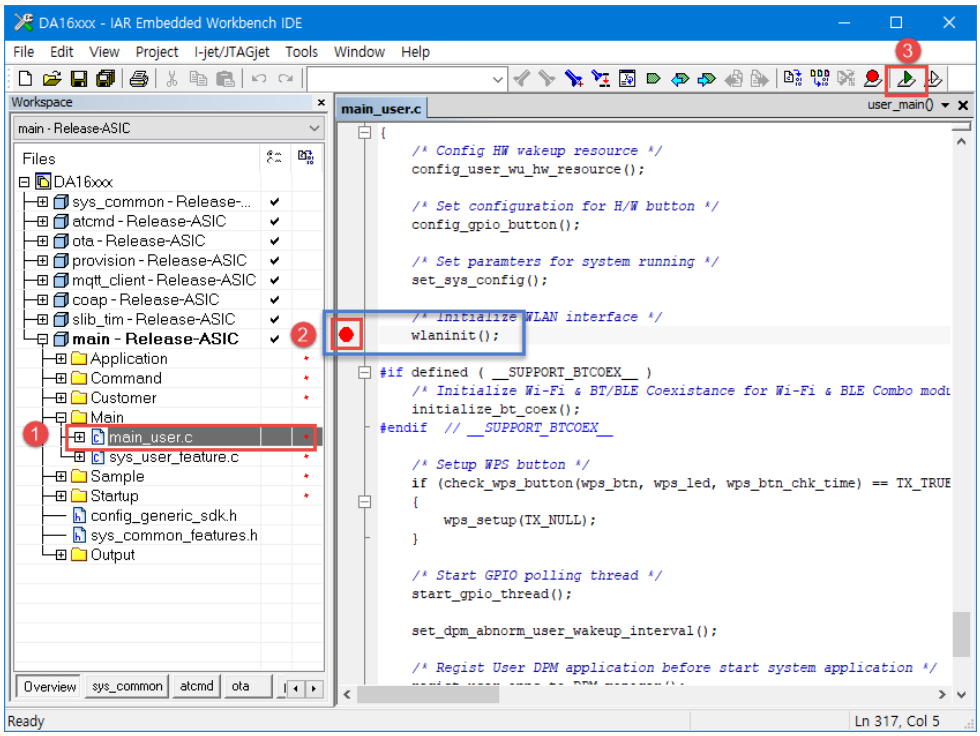


Figure 13: Run DA16200 by J-TAG

© 2020 Dialog Semiconductor

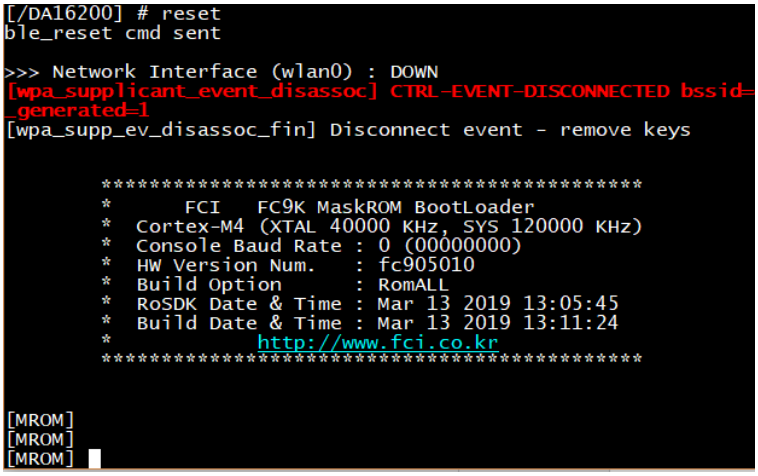


Figure 14: Run DA16200 by J-TAG - MROM

If you want to boot the DA16600 EVB in "non" J-TAG 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 J-TAG 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).

© 2020 Dialog Semiconductor

4.5.2 Run DA14531 with J-TAG

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

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

1. Build the DA16600 SDK with __DA14531_BOOT_FROM_UART__ disabled (see "[DA16600_SDK_ROOT]\src\customer\config_combo_module_sdk.h"), and program sFlash with the three DA16200 images (BOOT, SLB, and RTOS).
   The DA14531 image does not need be programmed.
2. Make sure DA16600 EVB's DIP switch configuration (SW7 and SW3) looks like Figure 25.
3. Connect a USB cable to USB Port **C** (DA14531 J-TAG Port) of the DA16600 EVB. See Figure 1.
4. Connect a USB cable to USB Port **B** of the DA16600 EVB, for a Teraterm 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.

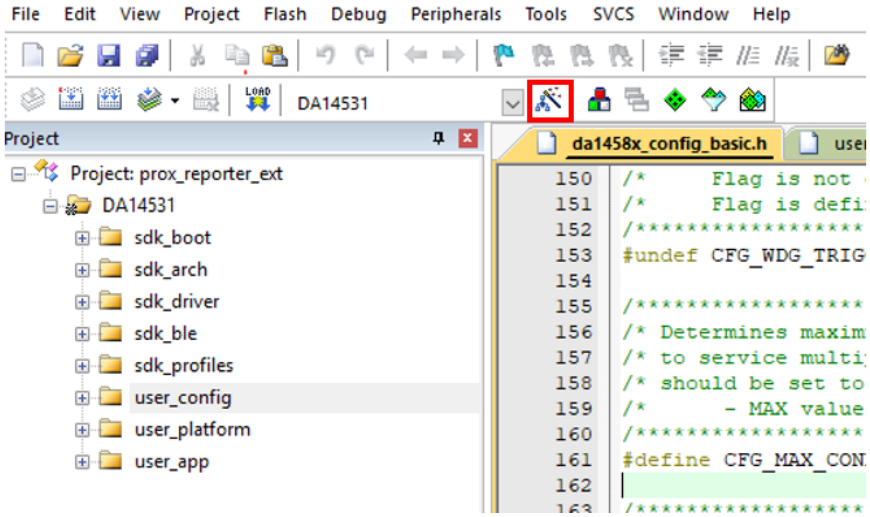


Figure 15: Keil - Option

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

© 2020 Dialog Semiconductor

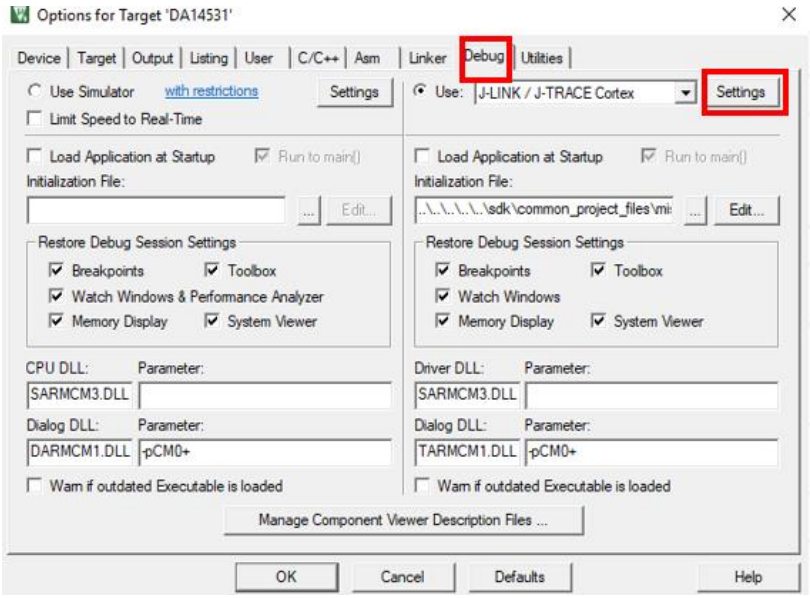


Figure 16: Keil - Debug

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

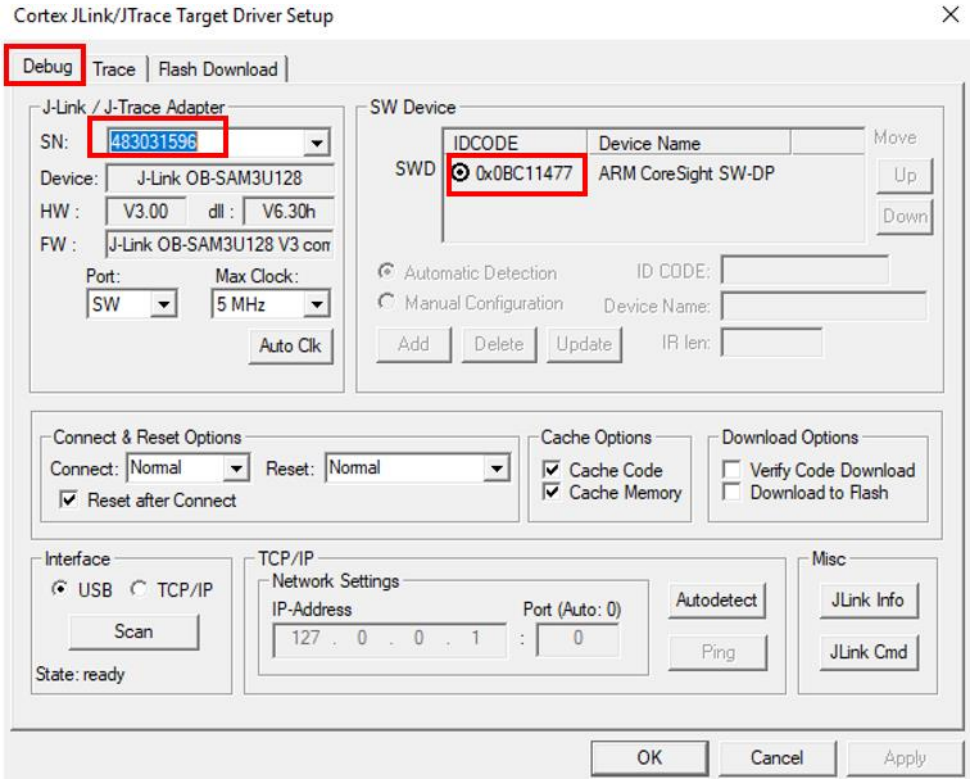


Figure 17: Keil – J-TAG Device

© 2020 Dialog Semiconductor

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

10. If the DA14531 J-TAG 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 Teraterm (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=921600
>>> UART1 : DMA Enabled ...
wakeup_src = 17
BLE_BOOT_MODE_0
```

Figure 18: Teraterm - DA16200 Waiting for DA14531 to Connect

14. Enable J-TAG 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 build is done, click the **Start Debugger** button. See Figure 19.

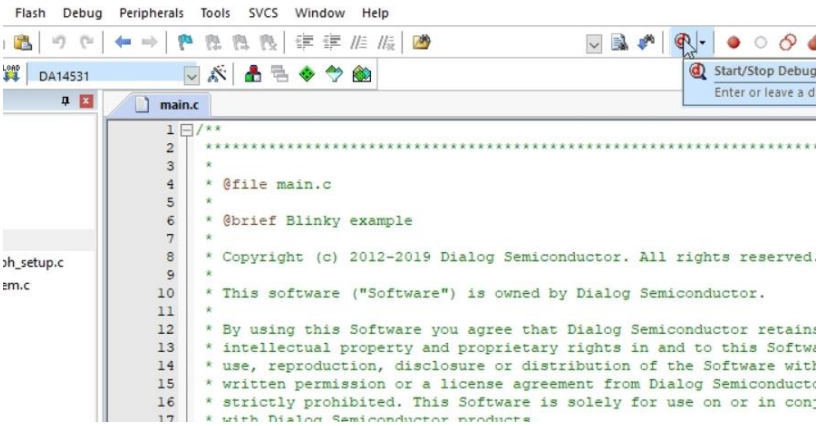


Figure 19: Keil - Start Debugger

16. Click **OK**.

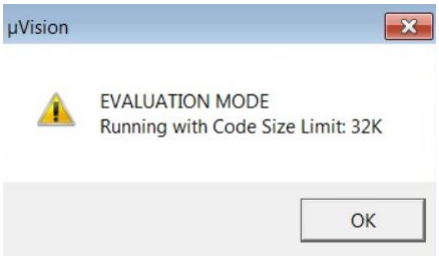


Figure 20: Keil - Evaluation Mode Popup

© 2020 Dialog Semiconductor

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

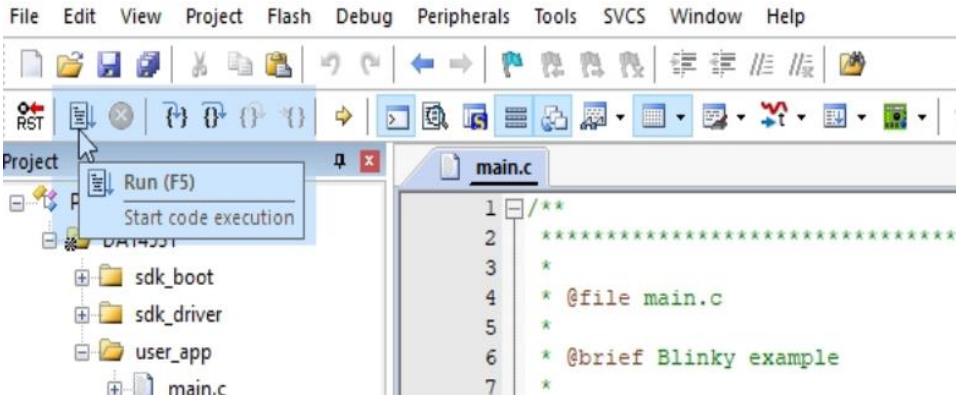


Figure 21: Keil - Run

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

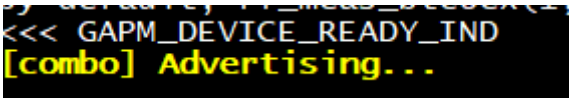


Figure 22: Teraterm - Advertising Successful

Now the DA16600 starts BLE advertising and can allow a BLE peer to connect to the DA16600.

© 2020 Dialog Semiconductor

5 Combo Example Test Procedure

5.1 Test Environment Setup

The following items are used in the example test:

- Wi-Fi Access Point
- BLE Peers
- Laptop: to Control BLE Peers and Combo Boards
- DA16600 EVB Boards

5.1.1 Wi-Fi Access Point

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

5.1.2 BLE Peers

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

5.1.2.1 BLE Mobile App

Dialog provides a sample mobile application (Android App) called "WiFi Provisioning" to test examples 1 to 4. This mobile application is used to give Wi-Fi provision information (i.e. 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 BLE 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 BLE Mobile App as well to send a command.

5.1.2.2 BLE Sensors

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

These BLE peer devices should implement a simple GATT server application as described in Figure 47 to be able to work with the sensor gateway application of DA16600.

For example, as Raspberry Pi 3 is common and capable of BLE communication, it is used in the example test. If you have another BLE development board / device to act as a peer, write simple GATT server as described in Figure 47.

Suppose we have two BLE 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 BLE Stack and Python used in RBP3: bluez-5.51, python 3.7.3.

© 2020 Dialog Semiconductor

5.1.3 Laptop: to Control BLE Peers and Combo Boards

1. Use a LAN cable to connect your laptop 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 Teraterm window and connect to the COM port (the lower port number of the two, with Baudrate 230400) of the DA16600. Let us say that this Teraterm window is called "**da16_tera_win**" from here onwards.
7. Open another Teraterm window and connect to the other COM port (the higher port number of the two, with Baudrate 115200) of the DA16600. Let us say that this Teraterm window is called "**da14_tera_win**" from here onwards. (this Teraterm 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: BLE Assisted Wi-Fi Provisioning

A BLE mobile application is used for the Wi-Fi provisioning test. This mobile application is a BLE 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 (Power On Reset boot: plug out, and then plug in the USB cable).
   b. After boot, run 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 application (install [SDK_ROOT]\util\provisioning_app\app-debug.apk).
3. Configure Wi-Fi as shown on Figure 23 with screen shots of the provisioning mobile application.

© 2020 Dialog Semiconductor

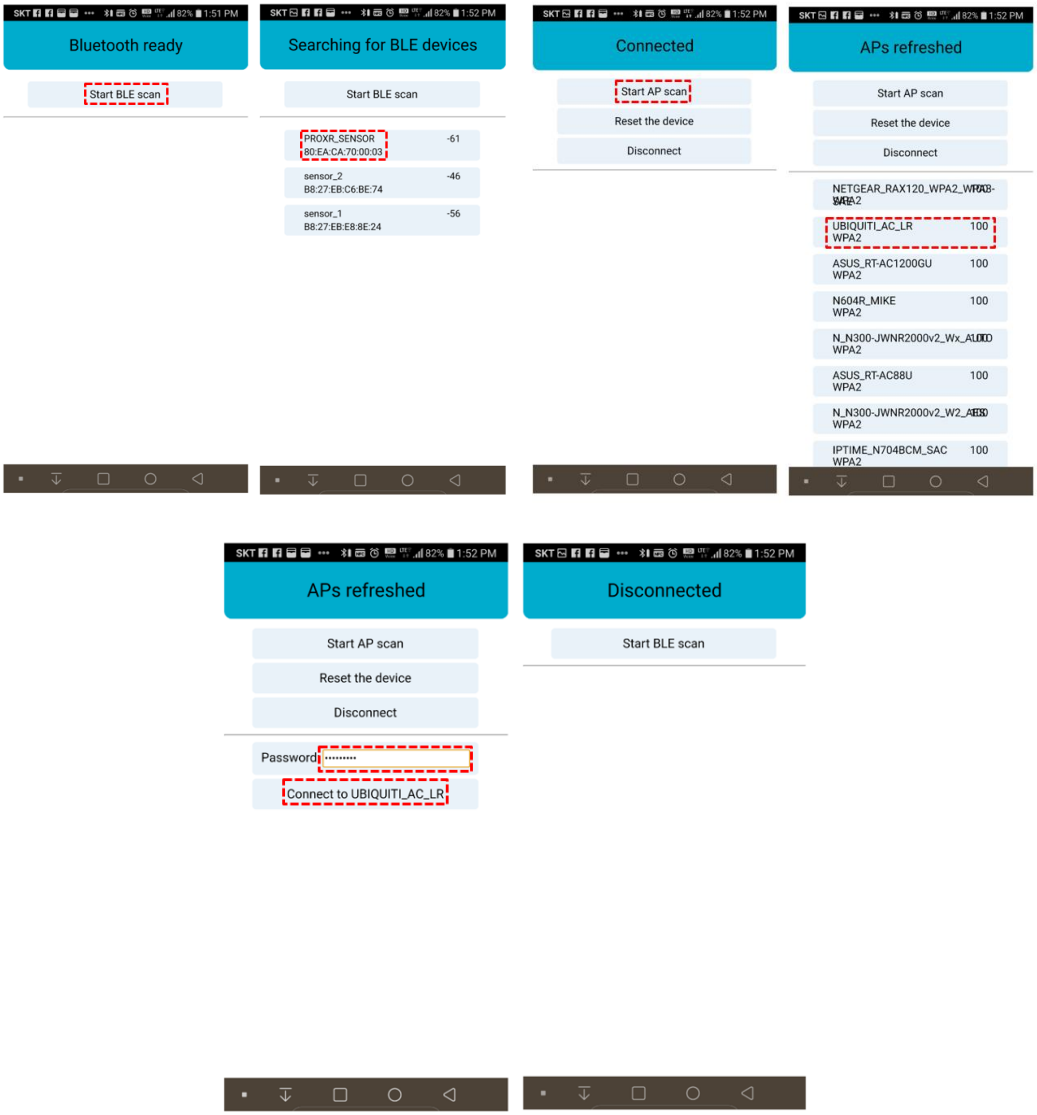


Figure 23: Provisioning App (Android)

4. If **Connect to [AP name]** is clicked, then the provisioning information is transferred to DA16600, which saves the provisioned information in NVRAM and reboots. After reboot, Wi-Fi is connected to the selected AP.

To remove the provisioning information:

1. Establish a BLE connection again with DA16600_EVB_1.
2. Click the **Reset the device** button.

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

Now you can start provisioning again.

© 2020 Dialog Semiconductor

5.3 Example 2: BLE 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 setup a simple http server. For example:

1. Install an http server on a laptop with Apache as 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 laptop 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 ...
```


© 2020 Dialog Semiconductor

```
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 BLE firmware is
loaded (from either bank_1 or bank_2). The current version of BLE 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 // BLE FW transferred successfully to DA14531 via UART
<<< GAPM_DEVICE_READY_IND
Advertising... // BLE 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 successfully 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
```


© 2020 Dialog Semiconductor

- BLE Mobile App

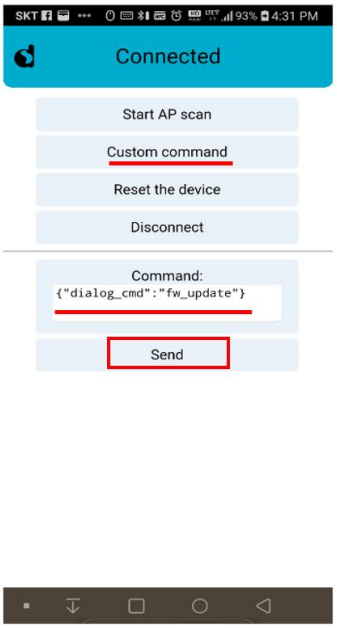


Figure 24: Provisioning Application - Custom Command

  - BLE 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 BLE Mobile Application

- da16_tera_win
  What 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 BLE 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
BLE 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
```


© 2020 Dialog Semiconductor

```
 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 BLE 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)
...
```


© 2020 Dialog Semiconductor

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)
```


© 2020 Dialog Semiconductor

```
          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!!!
```

What is happening:

- If you run 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

© 2020 Dialog Semiconductor

- 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
  - DA14531 image (pre-built image)
    - [DA16600_SDK_ROOT]\img\DA14531_1**da14531_multi_part_proxr.img**
- Teraterm: open two Teraterm windows
  - Teraterm_1 (da16_tera_win): connect to COMxx (lower one) with 230400 as baud rate. This is debug console of DA16200. 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 commands below to see which samples are available.

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

…
```


© 2020 Dialog Semiconductor

```
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
```


© 2020 Dialog Semiconductor

```
peri              : Run peripheral driver sample

[/DA16200/ble] # peri

------------------------------------------------
peri : Run DA14531 Peripheral Driver Sample
       type a command below
------------------------------------------------
[1] peri blinky     : blinking LED sample
[2] peri systick    : systick timer sample
[3] peri timer0_gen : timer0 general sample
[4] peri timer0_buz : timer0 PWM buzzer sample
[5] peri timer2_pwm : timer2 PWM LED array sample
[6] peri batt_lvl   : battery level read sample
[7] peri i2c_eeprom : I2C EEPROM read/write sample
[8] peri spi_flash  : SPI_flash read/write sample
[9] peri quad_dec   : Quadrature Decoder sample
------------------------------------------------
[/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
    - Use "Configuration_1"
    - By default, P0_8 is used to connect to LED. Connect J14:P0_8 to any PIN in CN4
  - Run command as below. The LED connected to the GPIO specified will blink

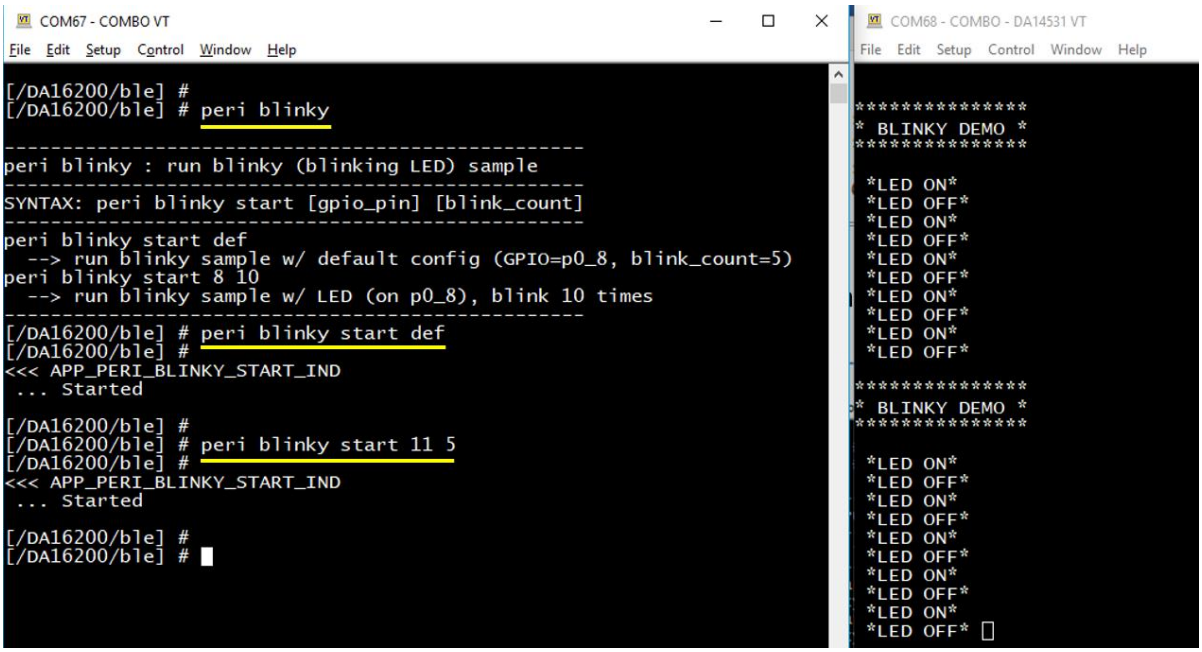


Figure 28: Peri Blinky

© 2020 Dialog Semiconductor

- peri systick
  - This sample uses 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
    - Use "Configuration_1"
    - By default, P0_8 is used to connect to LED. Connect J14:P0_8 to any PIN in CN4
  - Run command as below. Whenever systick timer is expired, it toggles the LED state

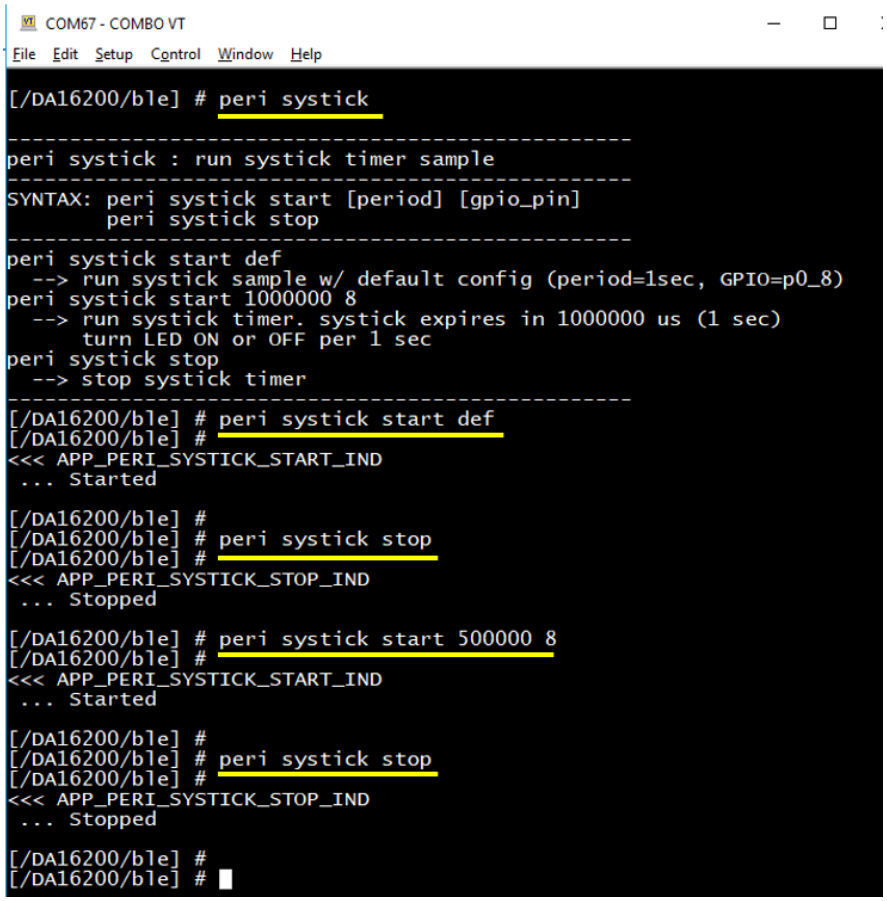


Figure 29: Peri Systick

© 2020 Dialog Semiconductor

- 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
    - Use "Configuration_1"
    - By default, P0_8 is used to connect to LED. Connect J14:P0_8 to any PIN in CN4
  - Run command as below, and check LED

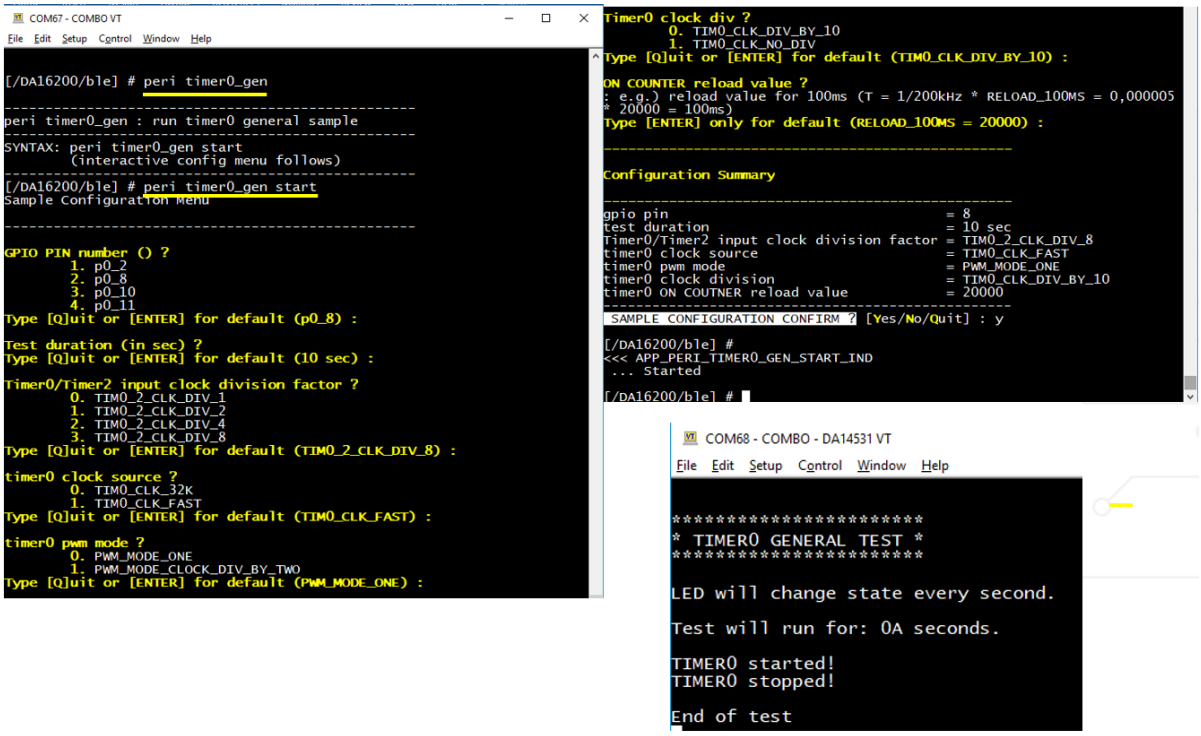


Figure 30: Peri Timer0_gen

© 2020 Dialog Semiconductor

- 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
    - Use "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 below

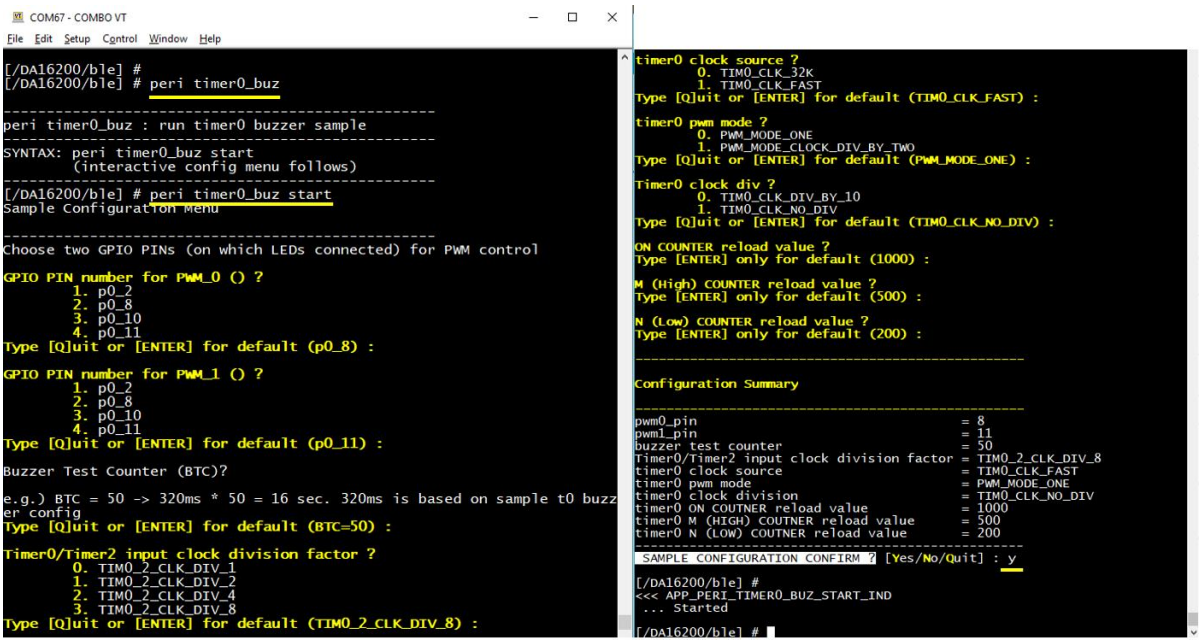


Figure 31: Peri Timer0_buz 1/2

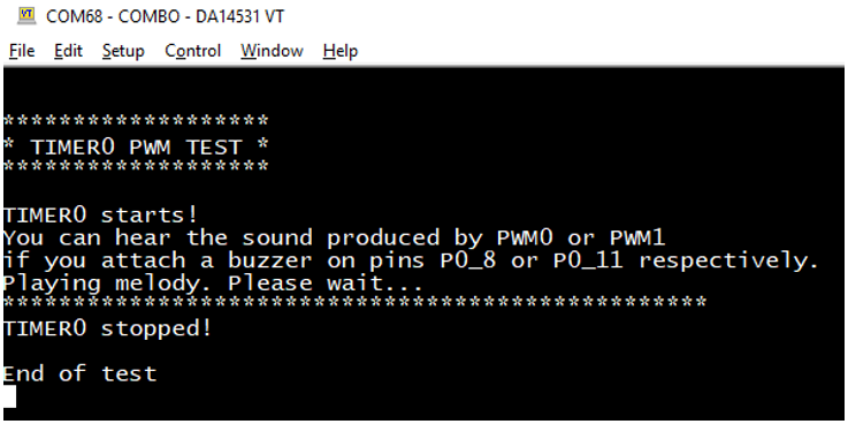


Figure 32: Peri Timer0_buz 2/2

© 2020 Dialog Semiconductor

- 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
    - Use "Configuration_1"
    - By default, P0_8, P0_11, 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 below

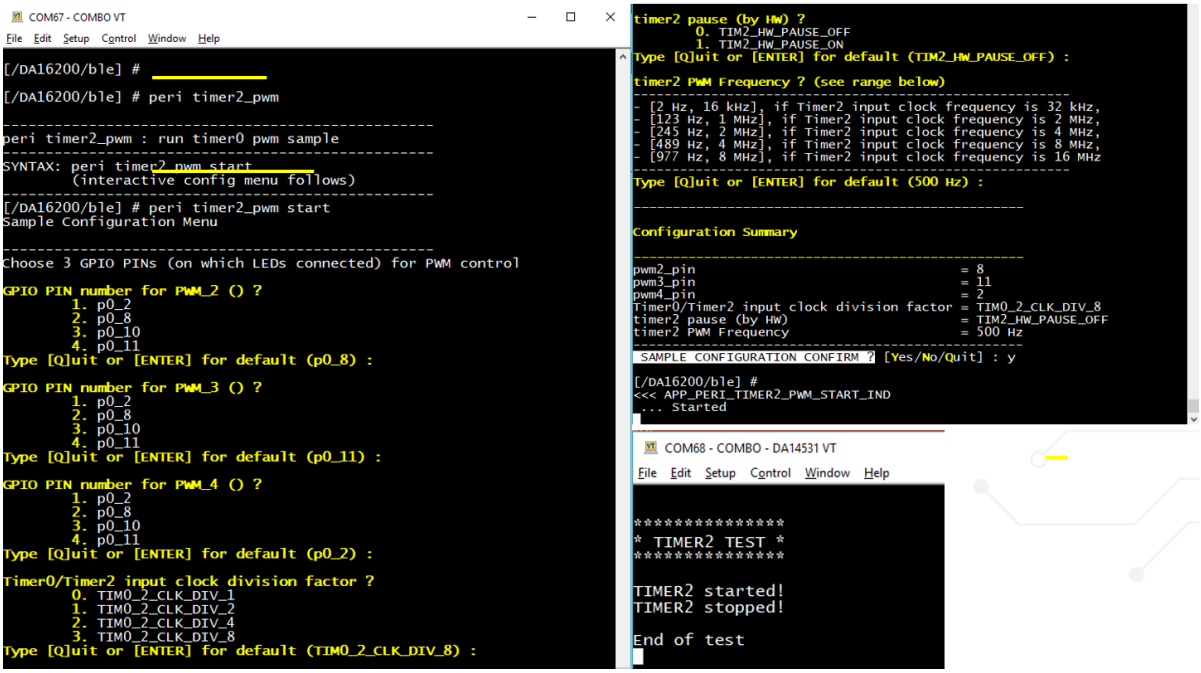


Figure 33: Peri Timer2_pwm

© 2020 Dialog Semiconductor

- peri batt_lvl
  - The Battery example demonstrates how to read the level of battery that is connected to DA14531
  - DA16600 EVB Configuration
    - Use "Configuration_1"
  - Run command as below

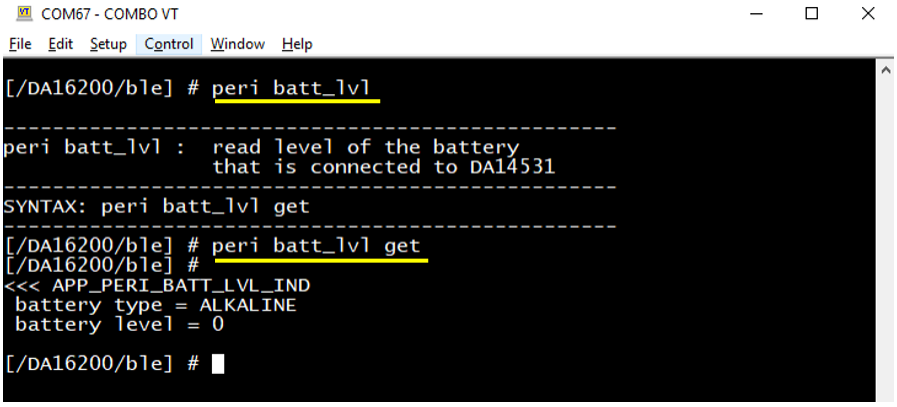


Figure 34: Peri Batt_lvl

- 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
    - Use "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 below

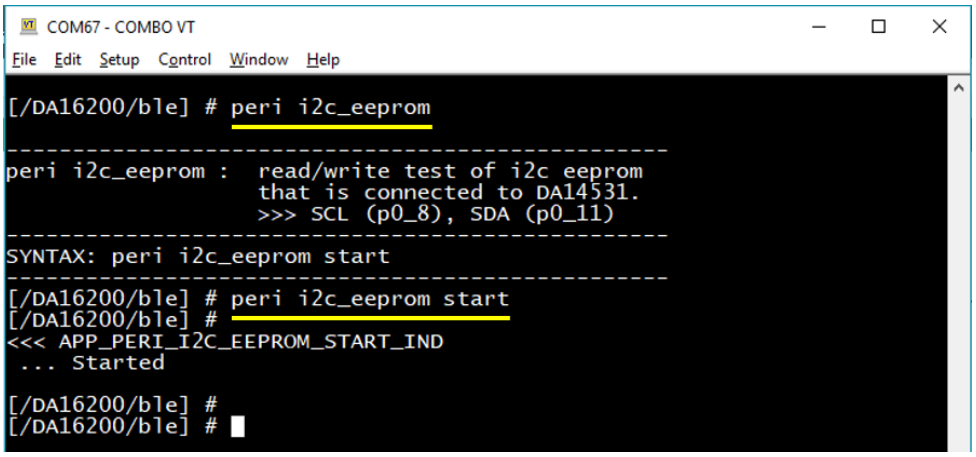


Figure 35: Peri I2c_eeprom

© 2020 Dialog Semiconductor

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 DA14531 img
    - #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
    - Use "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 image below for this test

      [DA16600_SDK_ROOT]\img\DA14531_1\peri_spi_flash**da14531_multi_part_proxr_s.img**
    - If you don't use the image above, you will get the following error

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

Figure 37: Reri Spi_flash - Wrong Image Warning

    - After booting with a correct BLE test image, you can find the following version string printed at boot

© 2020 Dialog Semiconductor

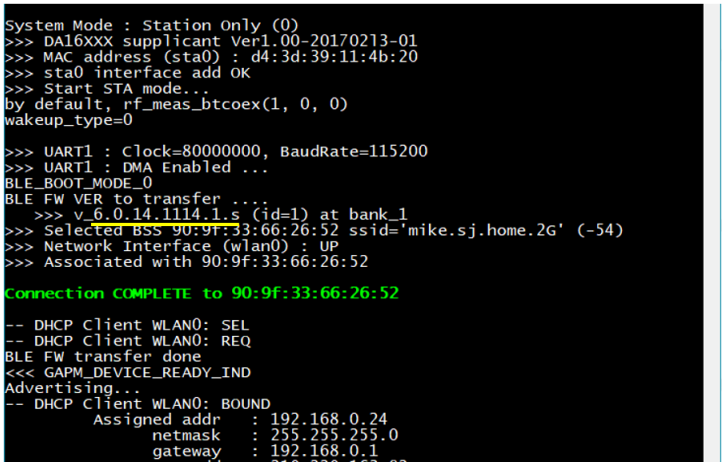


Figure 38: Correct Image Version for Reri Spi_flash Sample

- Run command as below

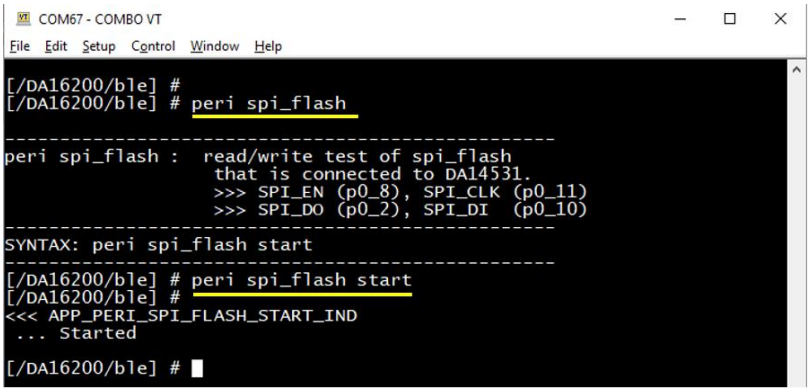


Figure 39: Reri Spi_flash

Figure 40: Peri Spi_flash Read/Write

 © 2020 Dialog Semiconductor

- 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
    - Use "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 don't use the image above, you will get the following error

```
[/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 BLE test image, you can find the following version string printed at boot

```
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 Reri Quad_dec Sample

© 2020 Dialog Semiconductor

- Run command as below

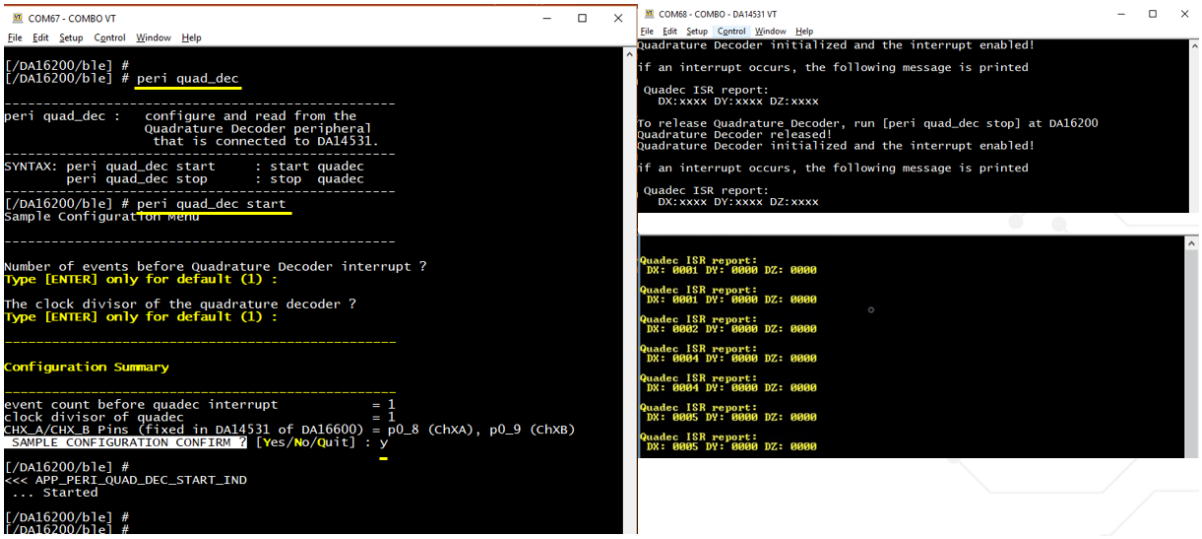


Figure 43: Peri Quad_dec - Start

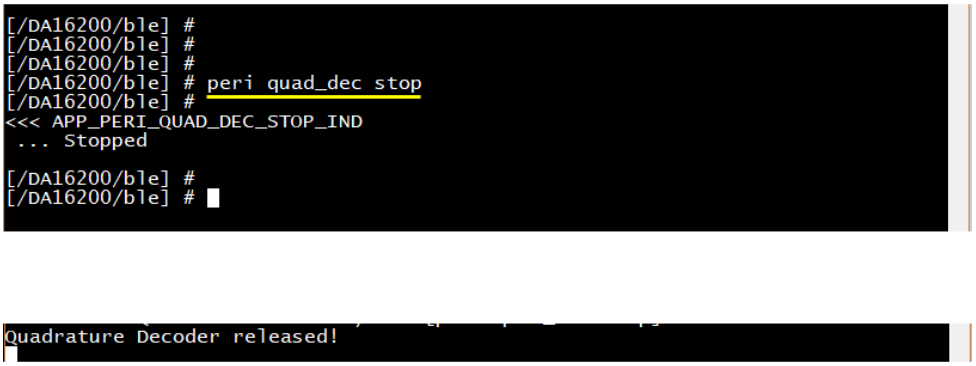


Figure 44: Peri Quad_dec Stop

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

© 2020 Dialog Semiconductor

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 Teraterm 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 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 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 below.

```
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 45: TCP Client in DPM Sleep

© 2020 Dialog Semiconductor

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

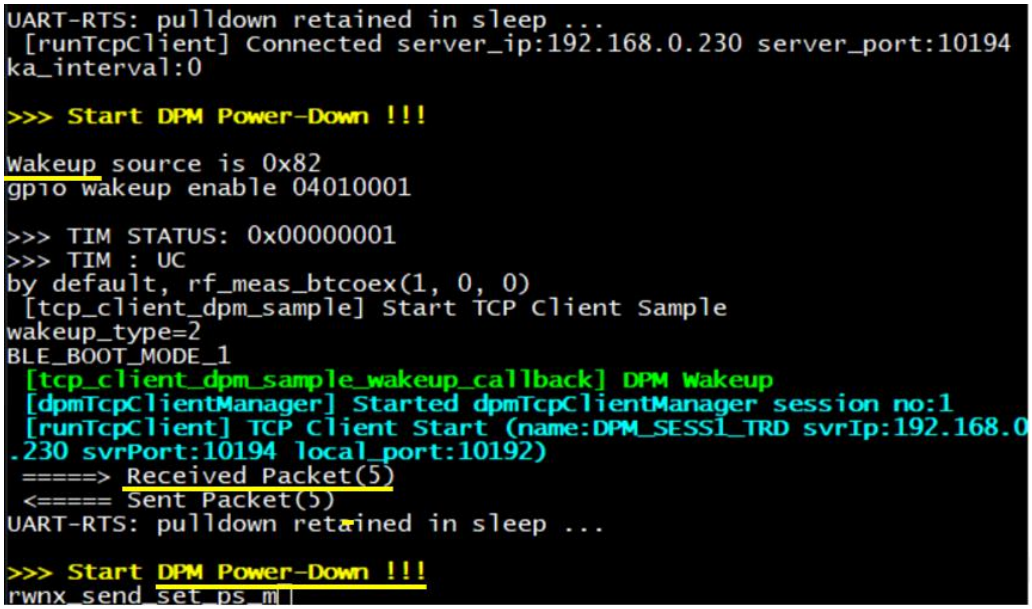


Figure 46: TCP Client - Wakeup from DPM Sleep

- Wi-Fi Provisioning while DA16600 is in sleep
  - Wi-Fi Provisioning APP: scan → connect to "DA16600" → DA16600 wakes up and connects to Wi-Fi Provisioning → do provision to an AP if needed

© 2020 Dialog Semiconductor

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 above you started two BLE 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.

- da16_tera_win
  - Power off DA16600_EVB_1 and power ON DA16600_EVB_2
  - Connect Teraterm to DA16600_EVB_2 (sensor_gateway)
  - By default, the sensor gateway is running as "BLE 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 BLE 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 BLE LE scanning

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

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

 © 2020 Dialog Semiconductor

```
          ################################################

#  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
...

#########################################################################
```


© 2020 Dialog Semiconductor

```
#########################   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 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: 31
temperature value: 40
temperature value: 27
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
```


© 2020 Dialog Semiconductor

```
temperature value: 12
temperature value: 37
temperature value: 2
temperature value: 32
temperature value: 19
temperature value: 34
temperature value: 0
temperature value: 17
temperature value: 18
temperature value: 5
temperature value: 29
temperature value: 30
temperature value: 37
temperature value: 4
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 messsage ... 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
...
```

What is happening:

- DA16600_EVB_2 acts as BLE GAP Central. You will see it start LE scanning when it is booted
- We have two BLE (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
- Anyway, when the 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**

© 2020 Dialog Semiconductor

- An RBP3 Sensor has a simple GATT service running as shown in Figure 47

```
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 47: Sensor_1/2 BLE Peer Sample Implementation (in Python)

- Sensor Service detail: implement the following GATT service on your BLE 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 5 sec (the temperature value is notified every 5 seconds)
- Every 5 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 10th message from a sensor, so you can see the UDP Server print the message received from DA16600_EVB_2 at ssh_win_1

© 2020 Dialog Semiconductor

6 Code Walkthrough

6.1 SDK Source Structure

In the Combo module, the Wi-Fi (DA16200) and BLE (DA14531) chips are connected to via a 4-wire UART (Tx, Rx, RTS, and CTS, baud rate is 115200 by default) and communicate with each other over Dialog Semiconductor's proprietary GTL interface. In the GTL architecture, a BLE application is running on the external host (DA16200).

As the GTL architecture is used in Combo, a Combo user application developer should understand and know how to use the user APIs for both external host platforms (DA16200) and BLE 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

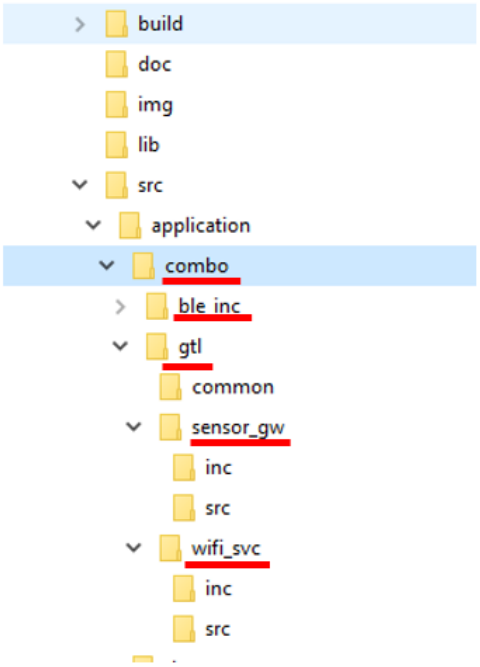


Figure 48: Source Folder Structure

- **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, the header files in this folder should be updated by 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 BLE 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 BLE peer applications (WiFi Provisioning Mobile APP)

© 2020 Dialog Semiconductor

  - **Gas leak detection sensor application** works with a gas leak sensor (virtual) that is locally connected to 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 BLE 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 BLE peer applications (WiFi 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 BLE 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 BLE peer applications (WiFi 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 BLE example "proximity monitor" (*projects\host_apps\windows\proximity***monitor**\). This is a GATT Client application used in example 4

6.2 Startup and GTL Initialization

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

int system_start(void) {
…
      /* combo: init 4-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[] = {
...
```


© 2020 Dialog Semiconductor

```
{ 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 Combo Application Development

The four BLE example applications (**Gas leak sensor, TCP Client, DA14531 Peri Driver**, and **Sensor Gateway**) included 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 BLE application flow (initialization and operation) is the same as for those examples.

A Combo user application (that may want to use both Wi-Fi and BLE functions) needs the development of functions that use both Wi-Fi APIs and BLE APIs.

To develop a function that talks to a BLE 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 reference document [4] > Section 2. Software Platform Overview, or the API document 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].

© 2020 Dialog Semiconductor

6.4 Examples

6.4.1 BLE 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 (BLE GATT Service) application/service is included in
all the examples. Below we explain the code using one (/wifi_svc) of examples.

[COMBO_SDK_ROOT]\src\customer\config_combo_module_sdk.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();
```


© 2020 Dialog Semiconductor

```
}

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;
...
```


© 2020 Dialog Semiconductor

6.4.1.3 GTL Message Flow

// Initialization until advertising

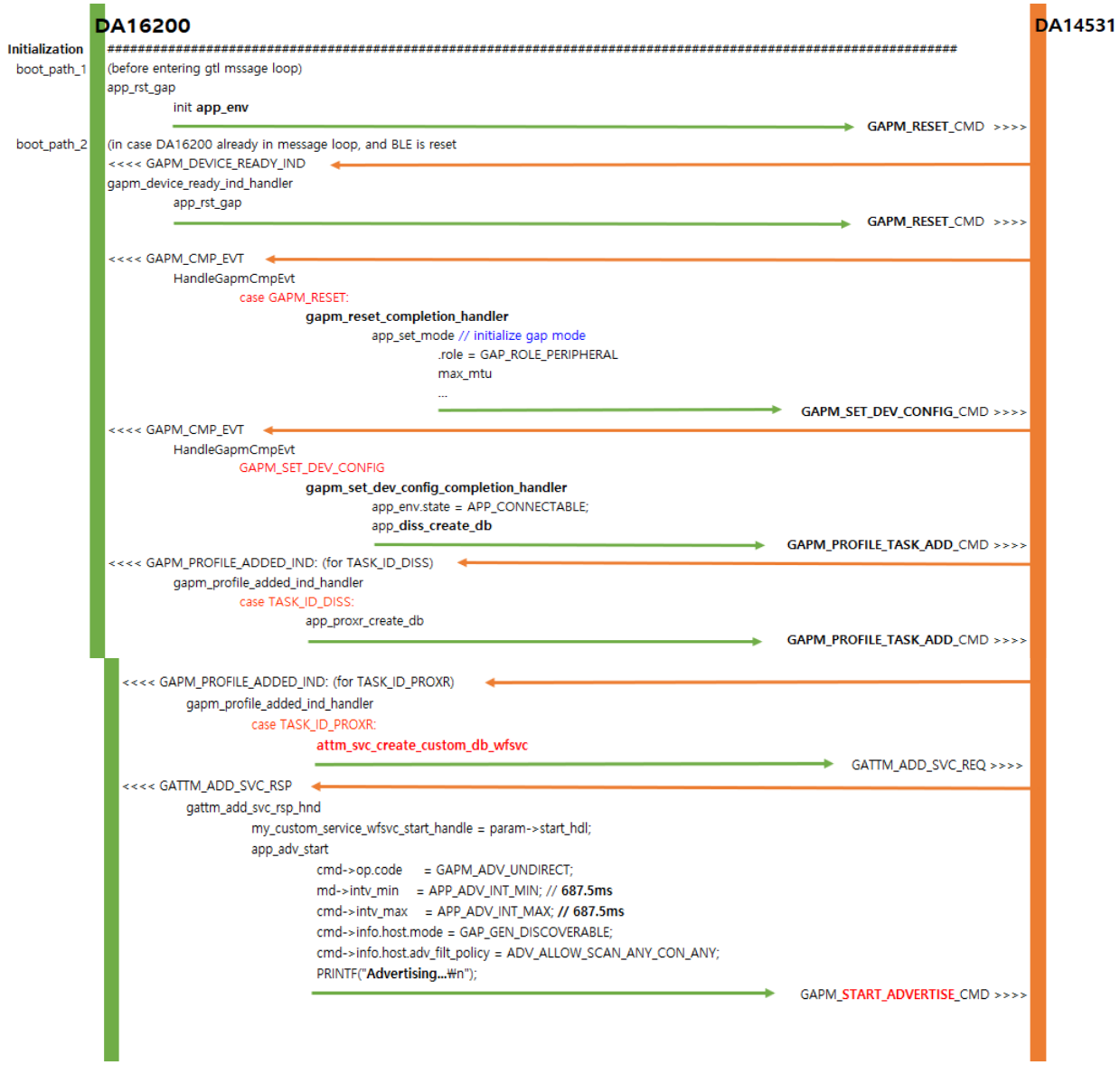


Figure 49: GTL Message Sequence Chart - Initialization

© 2020 Dialog Semiconductor

// Connection Request and Characteristic "Write" Request

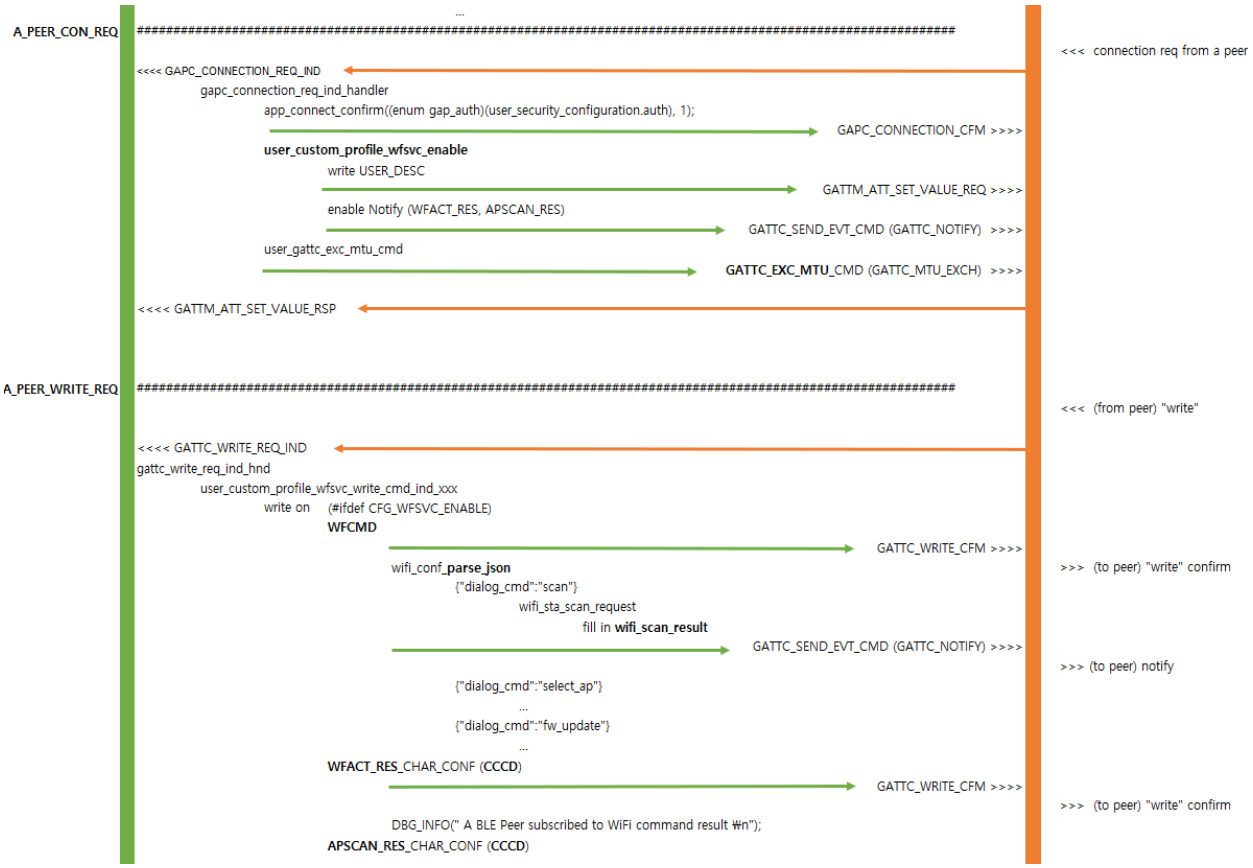


Figure 50: GTL Message Sequence Chart - Connect and Write

// Characteristic "Read" request

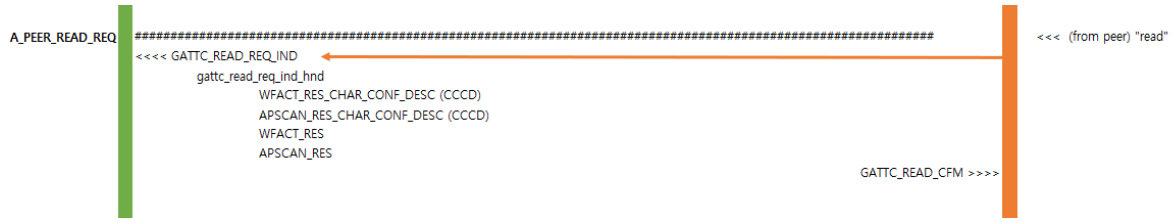


Figure 51: GTL Message Sequence Chart - Read

© 2020 Dialog Semiconductor

6.4.1.4 Wi-Fi Service GATT Database Design

The sample "Wi-Fi Service" GATT Server database is designed as below. 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.

att idx		att type	uuid	prop (read/write by ble peer)	value size	RI	user description	value to read / write
Wi-Fi Service			9161b201-1b4b-4727-a3ca-47b35cdcf5c1					
1		decl				No		

2	**Wi-Fi cmd**	**value**	**9161b202-1b4b-4727-a3ca-47b35cdcf5c1**	write	244	-		{
								"dialog_cmd":"**scan**"
								}
								
								{
								"dialog_cmd":"select_ap"
								"SSID":"linksys",
								"security_type": 3
								....
								}
								
								{
								"dialog_cmd":"fw_update"
								}
								see function wifi_conf_parse_json( )

3		descr (user)		read		No	"Wi-Fi Cmd"	
4		decl				No		

5	**Wi-Fi action result**	**value**	**9161b203-1b4b-4727-a3ca-47b35cdcf5c1**	read
				notify
					2	**Yes**		// 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 enum WIFI_ACTION_RESULT

6		descr(cccd)		read write write w/o rsp		**Yes**		
7		descr(user)		read		No	"Wi-Fi Action Result"	
8		decl				No		
9	**AP Scan result**	**value**	**9161b204-1b4b-4727-a3ca-47b35cdcf5c1**	read	244	**Yes**		
10		descr(user)		read		No	"AP Scan Result"	

Figure 52: Wi-Fi SVC GATT Service Design

6.4.1.5 Application Protocol Between a BLE 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 BLE peer that wants the Combo device to scan Wi-Fi routers

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

  - "select_ap" command can be sent by a BLE Peer to tell the Combo device which AP to connect to (plus other customer proprietary information, if needed, such as customer server information)
    Upon receipt of this command, Combo 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 Link Local IP */
    "password":"123456789",
```

 © 2020 Dialog Semiconductor

```
    "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 BLE Peer to tell Combo to download a new firmware from a specified OTA server

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

- "factory_reset" command can be sent by a BLE Peer to tell Combo to remove the Wi-Fi network profile saved. The Combo board will reboot after the reset

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

- "reboot" command can be sent by a BLE Peer to tell Combo 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 BLE Peer to find out the Wi-Fi connection status (connected or disconnected). The BLE 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 BLE Peer to disconnect a Wi-Fi connection from the currently connected AP. If the BLE 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

© 2020 Dialog Semiconductor

- Characteristic: "Wi-Fi Action Result" (2 byte), **READ, NOTIFY**
  - A BLE Peer is supposed to enable notification on this characteristic on connection
  - Then the BLE 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 BLE 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 BLE 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 BLE Peer gets the first read response (with payload), the payload included in the "read" response includes a 4-byte '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 2 bytes of the header, includes the **remaining length** of the total JSON_STR
    - H_2: the second 2 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 BLE 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 BLE Peer needs to combine and parse the whole JSON_STR to get the necessary infornation (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
```


© 2020 Dialog Semiconductor

```
},
{
"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 BLE Peer App is implemented, a BLE 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 BLE Firmware OTA Download via Wi-Fi

OTA FW Download Service (Wi-Fi download of FW) is included 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 BLE 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 Combo device (DA16200 Wi-Fi chip included) is 'always' in connected state with a Wi-Fi router, which has the Internet connection, a Service Server / Cloud Server is able to talk with this Combo 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 Combo 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 Combo device, and the Combo device restarts to boot with the new software.

For **Option 2**, a BLE 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;
       }
```


© 2020 Dialog Semiconductor

```
...
       else if (cmd == COMBO_WIFI_CMD_FW_UPDATE) {
              DBG_INFO(" COMBO_WIFI_CMD_FW_BLE_DOWNLOAD received \n");

              // download ble firmware 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 Combo Software 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 BLE 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:
```


© 2020 Dialog Semiconductor

```
                    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 BLE (DA14531) which starts
sensor reading task (periodically reads a gas-leak sensor).
```

For the external host (DA16200) to be able to talk to the BLE (DA14531) with application messages, the following should be implemented on both Wi-Fi and BLE 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 SDKE - anable 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
```


© 2020 Dialog Semiconductor

```
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 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.

© 2020 Dialog Semiconductor

```
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
…
}
```


© 2020 Dialog Semiconductor

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

[COMBO_SDK_ROOT]\src\customer\config_combo_module_sdk.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)

[DA16600_SDK_ROOT]\src\application\combo\gtl\wifi_svc_peri\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;

[DA16600_SDK_ROOT]\src\application\combo\gtl\wifi_svc_peri\src\wifi_svc_peri_consol
e.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
/*
```

 © 2020 Dialog Semiconductor

```
// 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);
...
}

[DA16600_SDK_ROOT]\src\application\combo\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,
```


© 2020 Dialog Semiconductor

```
...
} 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;
```

 © 2020 Dialog Semiconductor

```
    }
...
}
```

[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();
```


© 2020 Dialog Semiconductor

```
      // 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. Function is
changing
 *        "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++;
}
```


© 2020 Dialog Semiconductor

6.4.4.3 About GPIO PINs in DA14531

- DA14531 has 12 GPIOs. Among them, 7 GPIOs are reserved and being used by GTL (4-wire UART, thus 4 PINs) and BT-WiFi Coex (3-wire, thus 3 PINs), therefore, there are 5 GPIOs free for peripheral devices. Depending on 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 as sensor reading purpose. (Other 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 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 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

```
[COMBO_SDK_ROOT]\src\customer\config_combo_module_sdk.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

// [DA16600_SDK_ROOT]\src\customer\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 },
...
}
```


© 2020 Dialog Semiconductor

```
//
[DA16600_SDK_ROOT]\src\application\combo\gtl\wifi_svc_tcp_client_dpm\src\wifi_svc_tc
pc_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;
```


© 2020 Dialog Semiconductor

```
	//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_CTS__)
	enable_wakeup_by_ble_cts();
#endif
	dpm_mng_job_done(); //Done opertaion
}
```


© 2020 Dialog Semiconductor

The TCP Client application is a network application, and the Provisioning application is a BLE application. Both applications should register to the DPM sub-system 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

```
[COMBO_SDK_ROOT]\src\customer\config_combo_module_sdk.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
```

 © 2020 Dialog Semiconductor

```
/*
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 BLE 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]);
...
      }
```


© 2020 Dialog Semiconductor

```
}

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

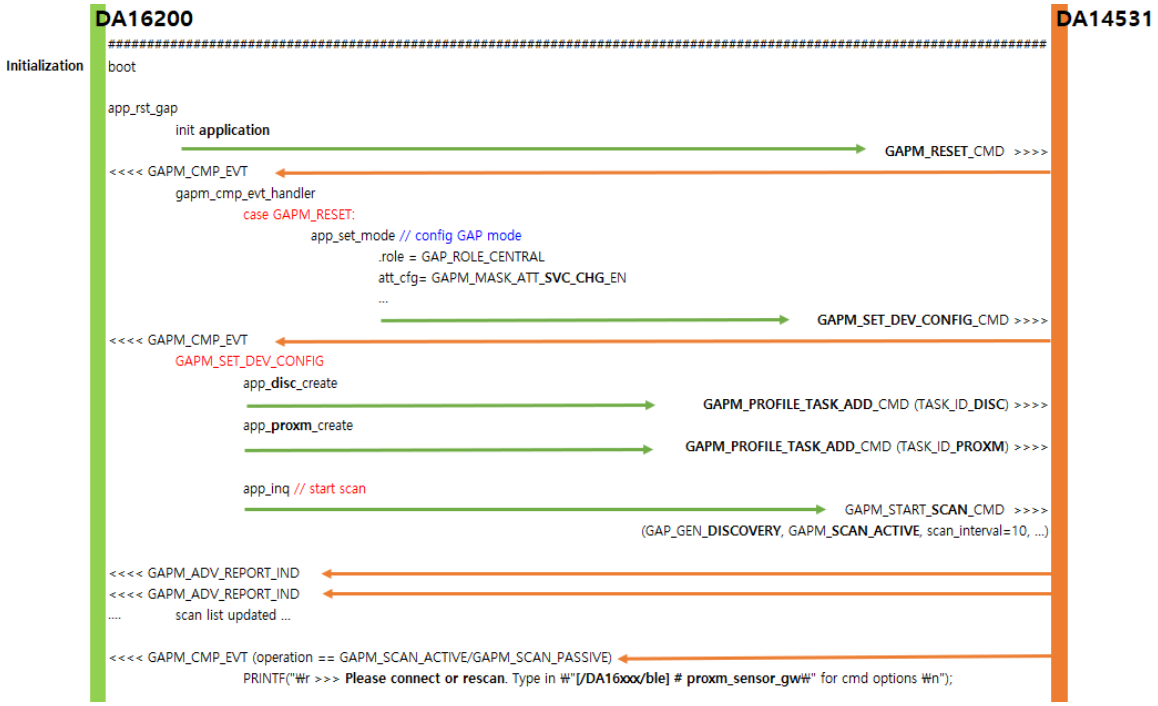


Figure 53: GTL Message Sequence Chart - Initialization

© 2020 Dialog Semiconductor

// Provisioning mode

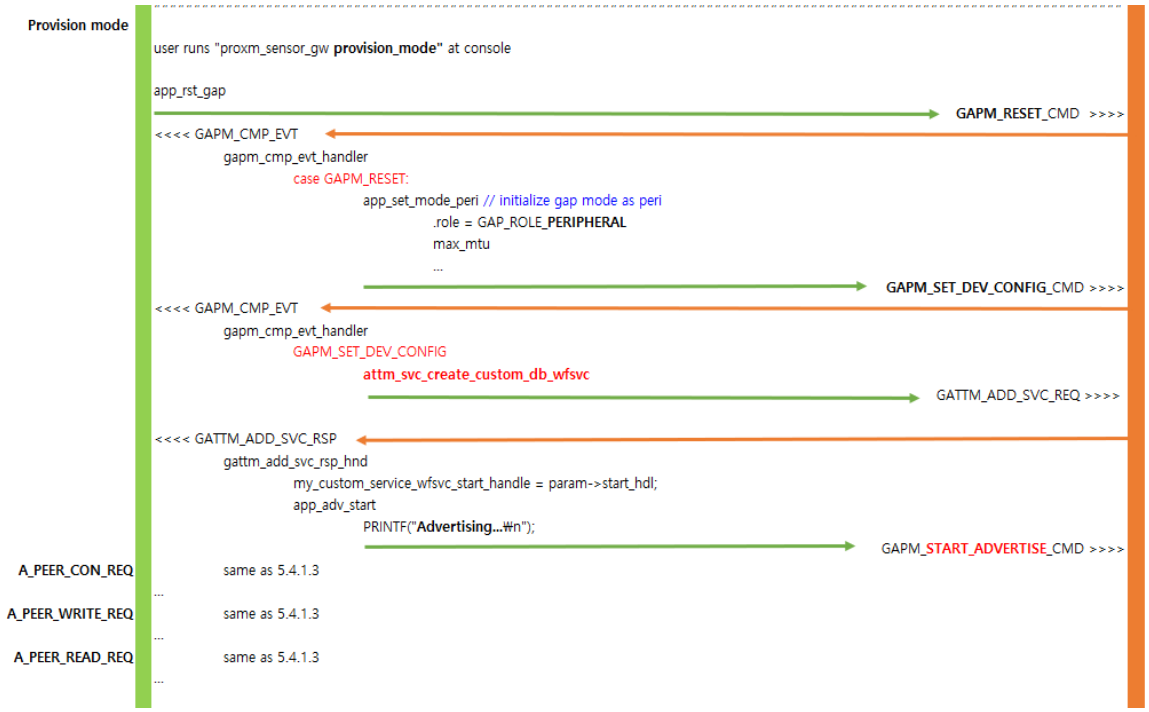
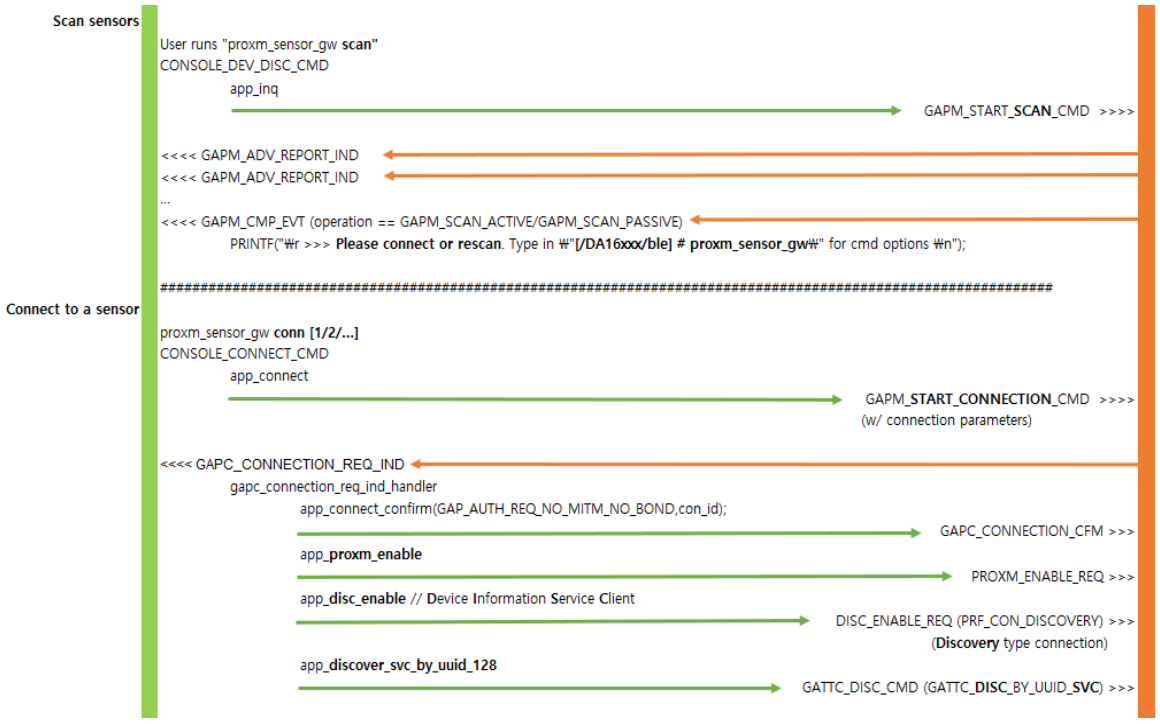


Figure 54: GTL Message Sequence Chart - Provisioning Mode

// Scan and Connect to Sensor



 © 2020 Dialog Semiconductor

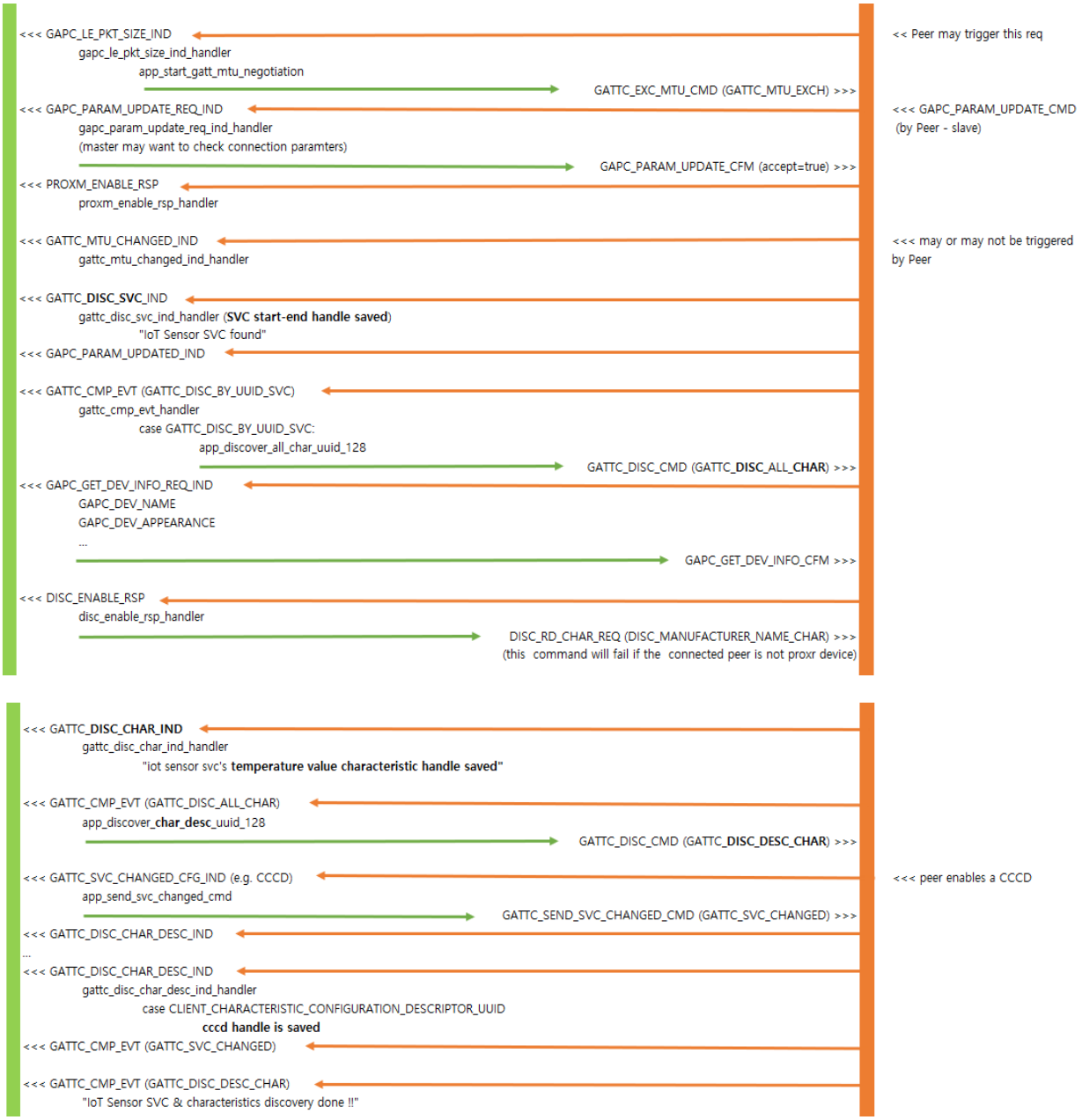


Figure 55: GTL Message Sequence Chart - Scan and Connect

© 2020 Dialog Semiconductor

// Enable Sensor Posting

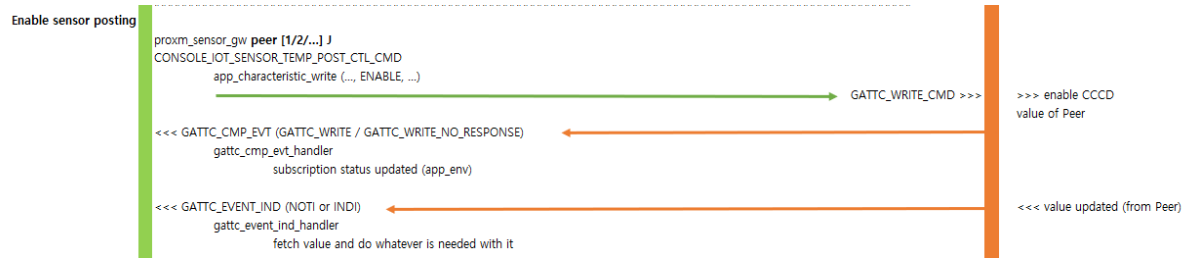


Figure 56: GTL Message Sequence Chart - Enable Sensor Posting

// Disable Sensor Posting

Figure 57: GTL Message Sequence Chart - Disable Sensor Posting

© 2020 Dialog Semiconductor

Revision History

Revision	Date	Description
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 Official Release.

© 2020 Dialog Semiconductor

Status Definitions

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

Disclaimer

Unless otherwise agreed in writing, the Dialog Semiconductor products (and any associated software) referred to in this document are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of a Dialog Semiconductor product (or associated software) can reasonably be expected to result in personal injury, death or severe property or environmental damage. Dialog Semiconductor and its suppliers accept no liability for inclusion and/or use of Dialog Semiconductor products (and any associated software) in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Information in this document is believed to be accurate and reliable. However, Dialog Semiconductor does not give any representations or warranties, express or implied, as to the accuracy or completeness of such information. Dialog Semiconductor furthermore takes no responsibility whatsoever for the content in this document if provided by any information source outside of Dialog Semiconductor.

Dialog Semiconductor reserves the right to change without notice the information published in this document, including, without limitation, the specification and the design of the related semiconductor products, software and applications. Notwithstanding the foregoing, for any automotive grade version of the device, Dialog Semiconductor reserves the right to change the information published in this document, including, without limitation, the specification and the design of the related semiconductor products, software and applications, in accordance with its standard automotive change notification process.

Applications, software, and semiconductor products described in this document are for illustrative purposes only. Dialog Semiconductor makes no representation or warranty that such applications, software and semiconductor products will be suitable for the specified use without further testing or modification. Unless otherwise agreed in writing, such testing or modification is the sole responsibility of the customer and Dialog Semiconductor excludes all liability in this respect.

Nothing in this document may be construed as a license for customer to use the Dialog Semiconductor products, software and applications referred to in this document. Such license must be separately sought by customer with Dialog Semiconductor.

All use of Dialog Semiconductor products, software and applications referred to in this document is subject to Dialog Semiconductor's Standard Terms and Conditions of Sale, available on the company website (www.dialog-semiconductor.com) unless otherwise stated.

Dialog, Dialog Semiconductor and the Dialog logo are trademarks of Dialog Semiconductor Plc or its subsidiaries. All other product or service names and marks are the property of their respective owners.

© 2020 Dialog Semiconductor. All rights reserved.

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.

Contacting Dialog Semiconductor

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

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

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

Email:
enquiry@diasemi.com

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

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

Taiwan
Dialog Semiconductor Taiwan
Phone: +886 281 786 222

Web site:
www.dialog-semiconductor.com

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

Korea
Dialog Semiconductor Korea
Phone: +82 2 3469 8200

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

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