User Manual

DA16200/DA16600 FreeRTOS SDK Programmer Guide

The DA16200/DA16600 is a highly integrated ultra-low power Wi-Fi system on a chip (SoC) that allows users to develop a complete Wi-Fi solution on a single chip. This document is an SDK guide which describes the examples that are included in the SDK and is intended for developers who want to develop applications using the DA16200/DA16600 SDK.

Contents

Figures

Tables

1. Terms and Definitions

AP	Access Point
ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
API	Application Programming Interface
AT	Attention
AWS	Amazon Web Services
BSS	Basic Service Set
CCM	Counter with CBC-MAC
CLI	Command Line Interface
CRC	Cyclic Redundancy Check
CTR	Counter
DAC	Digital-To-Analog Converter
DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DHCP	Dynamic Host Configuration Protocol
DMA	Direct Memory Access

DNS	Domain Name Server
DPM	Dynamic Power Management
DRBG	Deterministic Random Bit Generator
DTLS	Datagram Transport Layer Security
DUT	Device Under Test
EAP	Extensible Authentication Protocol
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EVB	Evaluation Board
EVK	Evaluation Kit
GCM	Galois/Counter Mode
GPIO	General-Purpose Input/Output
HMAC	Hash(-based) Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
KDF	Key Derivation Function
LE	Low Energy
MQTT	Message Queuing Telemetry Transport
MD5	Message Digest 5
MCU	Microcontroller Unit
NVRAM	Non-volatile Random-Access Memory
OFB	Output Feedback
OTA	Over the Air
PEM	Privacy-Enhanced Mail
POR	Power-On Reset
PWM	Pulse Width Modulation
QoS	Quality of Service
RSA PKCS	RSA Public Key Cryptography Standards
RTC	Real-Time Clock
RTM	Retention Memory
RTOS	Real-Time Operating System
SD/eMMC	Secure Digital/Embedded Multimedia Card
SDIO	Secure Digital Input Output
SDK	Software Development Kit
SNTP	Simple Network Time Protocol
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
STA	Station
TCP	Transmission Control Protocol
TIM	Traffic Indication Map
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver-Transmitter
UDP	User Datagram Protocol
WPA	Wi-Fi Protected Access
WPA2	Wi-Fi Protected Access 2
WPA3	Wi-Fi Protected Access 3

2. References

[1] LwIP API. (n.d). Retrieved September 9, 2021. From Savannah: https://www.nongnu.org/lwip/2_0_x/raw_api.html.

[2] DA16200, Datasheet, Renesas Electronics.

[3] UM-WI-056, DA16200 DA16600 FreeRTOS Getting Started Guide, User Manual, Renesas Electronics.

[4] UM-WI-042, DA16200 DA16600 Provisioning Mobile App, User Manual, Renesas Electronics.

[5] UM-WI-011, DA16200 DA16600 Mass Production User Manual, User Manual, Renesas Electronics.

[6] UM-WI-030, DA16200 DA16600 DPM User Manual, User Manual, Renesas Electronics.

[7] UM-WI-003, DA16200 DA16600 Host Interface and AT Command User Manual, User Manual, Renesas Electronics.

[8] UM-B-117, DA14531 Getting Started with the Pro Development Kit, Renesas Electronics.

[9] UM-B-143, Dialog External Processor Interface, Renesas Electronics.

[10] UM-B-119, DA14531 SW Platform Reference, Renesas Electronics.

[11] UM-WI-039, DA16200 DA16600 MultiDownloader, Renesas Electronics.

Note 1 References are for the latest published version, unless otherwise indicated.

3. Introduction

This document provides an overview of the Software Development Kit (SDK) used for application development based on Wi-Fi solution using the DA16200/DA16600 devices and boards. This SDK includes DA16200/DA16600 generic projects, sample projects, a set of libraries, and drivers to facilitate the creation of various applications by exploiting the provided hardware resources of a connected DA16200/DA16600 devices.

3.1 Overview

The DA16200/DA16600 FreeRTOS SDK has six folders:

- **apps**: project files and source codes for generic and sample applications
  - **apps/common/examples**: sample applications and template
  - **apps/da16xxx/get_started**: generic application
- **core** : source codes for core funnctions
- **docs**: doxgen document and lincens file
- **library** : pre-compiled lib (.a) files
- **tools**: build tools/scripts, temporary build artifacts, or environment files
  - **version** : firmware version files
- **utility** : utilities for programming, debugging, DA14531 SDK, and network tools

The SDK can be used with different features according to the use case or applications, and the features can be changed in the SDK.

General features are defined in `~/FreeRTOS_SDK/apps/da16200/<app name>/include/user_main/config_generic_sdk.h` where the features can be enabled or disabled. And other system features are defined in `~/FreeRTOS_SDK/apps/da16200/<app name>/include/user_main/sys_common_features.h.`

NOTE
The main header files including configurable features are located in ./apps/da16xxx/<app name>/include/user_main for generic projects or ./apps/common/examples/<sample group name>/<sample name>/include for sample projects. All features in config_generic_sdk.h are configurable as required. Some features in the sys_common_feature.h can be changed also but need the support from Renesas Support Team.

The typical e² studio project for the DA16200/DA16600 SDK is shown in Figure 1.

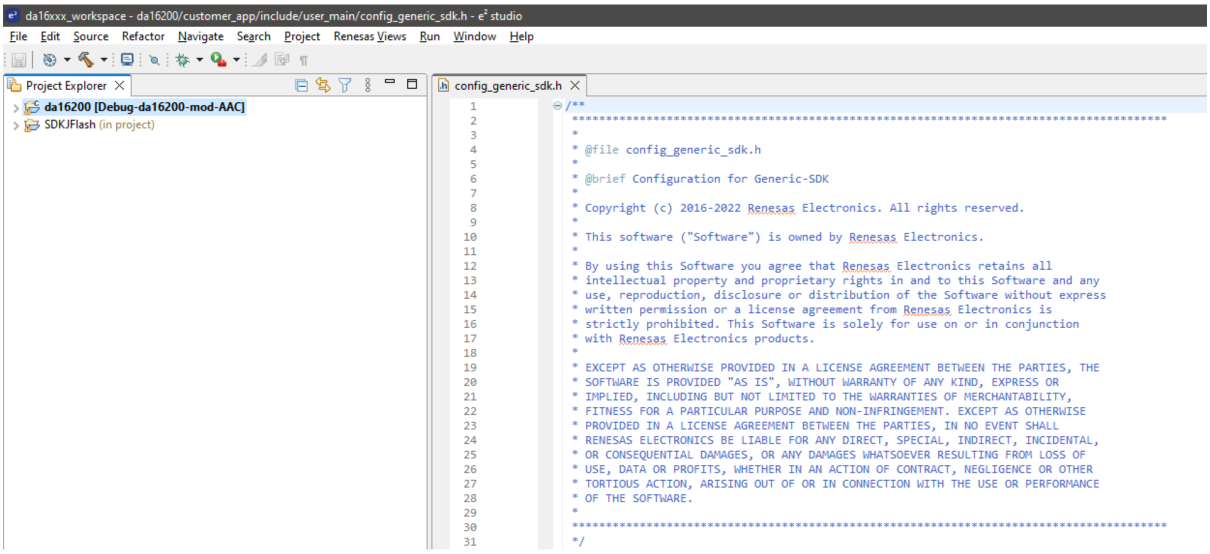


Figure 1. e² studio project configuration

3.2 Development Environment

The DA16200/DA16600 FreeRTOS SDK needs the Renesas e² studio IDE. See Ref. [3] for e² studio installation.

3.3 System and Application Startup

The `main()` is first startup function. After hardware resources (PIN_MUX, RTC, Console …) are initialized, `user_main()` in each project is called.

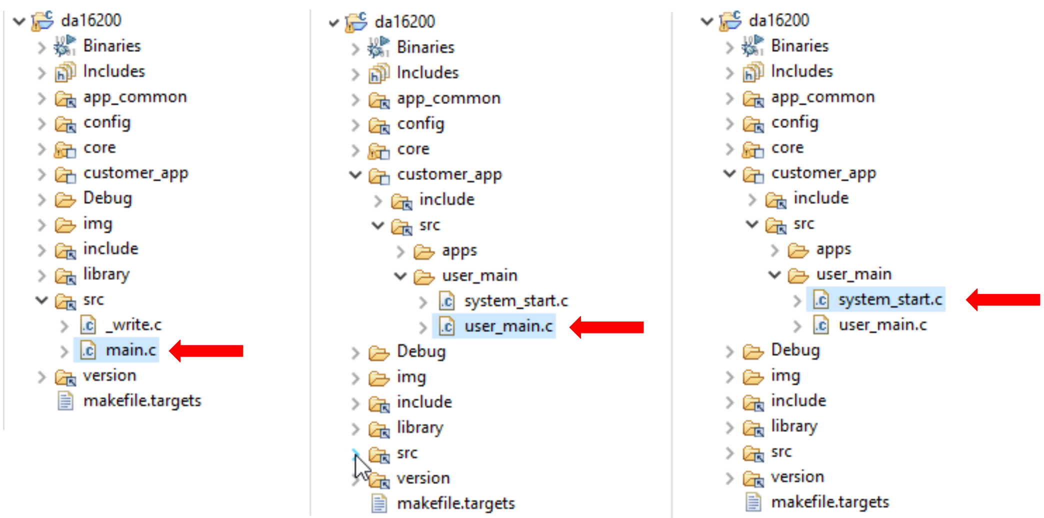


Figure 2. Startup files on DA16200/DA16600 project

```
[ ~/FreeRTOS_SDK/core/main/src/main.c ]
int main(char init_state)
{
    ...
    xTaskCreate(system_launcher,
                "system_launcher",
                256*3,                    // for SecureBoot
                (void *)NULL,
                (tskIDLE_PRIORITY+1),
                NULL);
    ...
    vTaskStartScheduler();
}

void system_launcher( void *pvParameters)
{
    ...
// Initialize and run system application
     // and run user application if needed.
    start_da16x();
    ...
}

static void start_da16x(void)
{
    ...
    /* Configure Pin-Mux of DA16200*/
    config_pin_mux();
   ...
    /* Start DA16200 IoT system layer*/
    user_main(ramlib_ptim_init_status); // USER main
}
```

`system_start()` in `user_main()` runs as follows:

- Configure hardware and software features
- Configure system resources for system clock and TX power

- Initialize Wi-Fi function in `wlaninit()`
- Start system applications in `start_sys_apps()`
- Start user applications in `start_user_apps()`

```
[~/FreeRTOS_SDK/apps/da16200/<app name>/src/user_main/user_main.c ]
int user_main(char init_state)
{
    ...
    /* Entry point for customer main */
    if (init_state == pdTRUE) {
     system_start();
    } else {
     Printf("\nFailed to initialize the RamLib or pTIM !!!\n");
    }

    return status;
}

[~/FreeRTOS_SDK/apps/da16200/<app name>/src/user_main/system_start.c ]
int system_start(void)
{
/* Config hardware wake-up resource */
    config_user_wu_hw_resource();

    /* Set configuration for hardware button */
    config_gpio_button();

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

    /* Initialize WLAN interface */
    wlaninit();

... ...

    /* Start system applications for DA16XXX */
    start_sys_apps();

    /*
     * Entry point of user's applications
     *      : defined in user_apps_table.c
     */
    /* Start system applications for DA16XXX */
    start_user_apps();
}
```

3.4 System Applications

After the startup function is run, each system application such as MQTT, HTTP server or AT command can be started according to the user defined features.

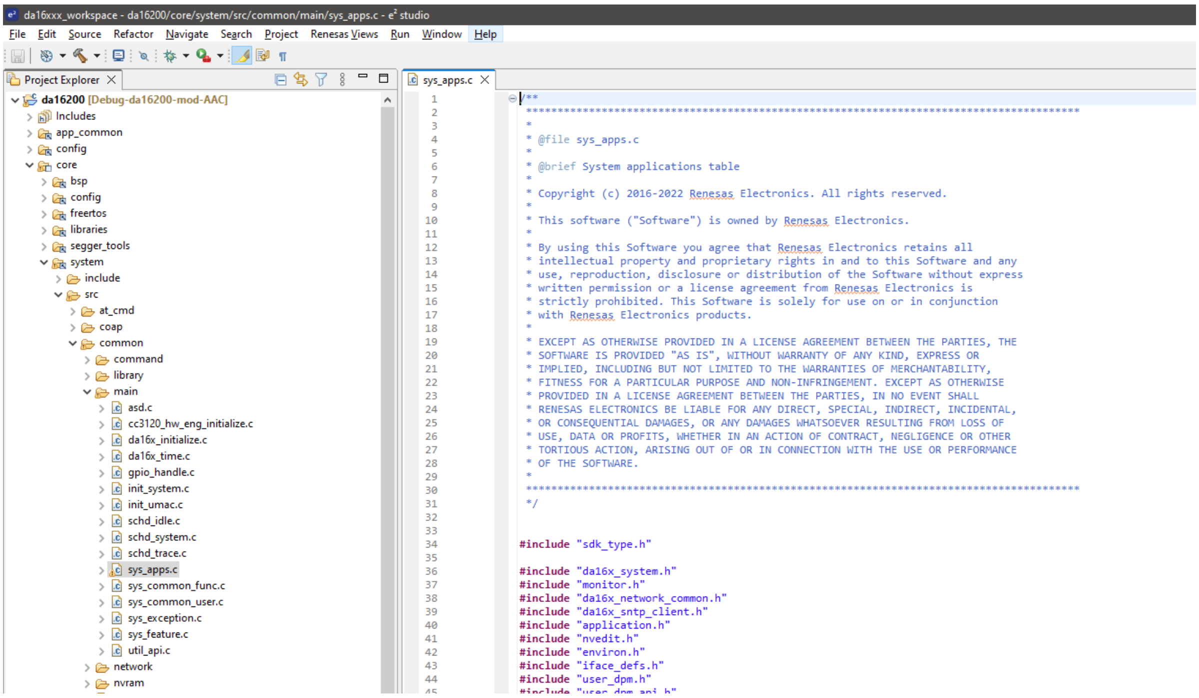


Figure 3. Applications on e² studio project

```
[ ~/FreeRTOS_SDK/core/system/src/common/main/sys_apps.c ]
void start_sys_apps(void)
{
...
    /* Start user application functions */
    run_sys_apps();
}
```

The system applications can run in two cases below:

- Applications run immediately regardless of network connection
- Applications run only after the network connection is completed.

```
static void run_sys_apps(void)
{
... ...
/* Create network independent apps */
create_sys_apps(sysmode, FALSE);

/* Create user's network independent apps */
create_user_apps(sysmode, FALSE);
    ...
    /* wait for network initialization */
    while (1) {
      if (check_net_init(iface) == pdPASS) {
             i = 0;
             break;
      }
      i++;
      vTaskDelay(1);
    }
...
    /* Check IP address resolution status */
    while (check_net_ip_status(iface)) {
      vTaskDelay(1);
    }
```

```
/* Create network apps */
create_sys_apps(sysmode, TRUE);
}
```

All system applications are in the `sys_apps_table[]` as shown in the example code below.

```
[ ~/FreeRTOS_SDK/core/system/src/common/main/sys_apps.c ]
static const app_task_info_t  sys_apps_table[] =
{
/* name, entry_func, stack_size, priority, timeslice, net_chk_flag, dpm_flag, port_no,
run_sys_mode */

    /****** For function features ***********************************/

    … …

#if defined ( __SUPPORT_MQTT__ )
{ APP_MQTT_SUB, mqtt_auto_start, 320, (U_PRIO), TRUE, TRUE, UNDEF_PORT, RUN_STA_MODE},
#endif       // __SUPPORT_MQTT__

    … …

/******* End of List *********************************************/
{ NULL, NULL,   0, 0, FALSE, FALSE, UNDEF_PORT, 0   }

};
```

The parameters of the `sys_apps_table[]` are as shown below.

```
[ ~/FreeRTOS_SDK/apps/da16200/get_started/include/apps/application.h ]
typedef struct  _app_task_info {
    /// Thread Name
    char     *name;

    /// Funtion Entry_point
    VOID     (*entry_func)(void *);

    /// Thread Stack Size
    USHORT   stksize;

    /// Thread Priority
    USHORT   priority;

    /// Flag to check network initializing
    UCHAR    net_chk_flag;

    /// Usage flag for DPM running
    UCHAR    dpm_flag;

    /// Port number for network communitation
    USHORT   port_no;

    /// Running mode of DA16xxx
    int            run_sys_mode;
} app_task_info_t;
```

- `name` Unique thread name
- `entry_func` Thread entry point
- `stksize` Stack size of thread
- `priority` Thread running priority

- `net_chk_flag` — `TRUE`: run application only after network connection is completed
  `FALSE`: run application immediately regardless of network connection
- `dpm_flag` — `TRUE`: register an application to DPM service
  `FALSE`: not register an application to DPM service
- `port_no` — Port number of network session for DPM service
- `run_sys_mode` — `RUN_STA_MODE`: run application only at Station mode
  `RUN_AP_MODE`: run application only at AP mode
  `RUN_STA_SOFTAP_MODE`: run application only at Concurrent (AP + Station) mode
  `RUN_ALL_MODE`: run application at any mode

NOTE
▪ Do not use malloc() or free() function to allocate or free memory. Use pvPortMalloc() or vPortFree() function for allocating or free memory ▪ There is no need to modify the system application tables in the DA16200/DA16600 SDK. However, if required, that can be modified with the support of Renesas Electronics ▪ See Ref. [6] for details about DPM service.

If sample projects in the SDK are used, sample applications also can be run. The sample applications are defined in `sample_apps_table[]` and the parameters are the same as the table of system applications.

```
[ ~/FreeRTOS_SDK/core/system/src/common/main/sys_apps.c ]
static void create_sys_apps(int sysmode, UCHAR net_chk_flag)
{
    … …

    /* Create test samples apps */
    if (sample_app_start_cb != NULL) {
        sample_app_start_cb(net_chk_flag);
    }
}
```

3.5 User Applications

After running the system applications, user applications run in two cases:

- Applications run immediately regardless of network connection

```
[ ~/FreeRTOS_SDK/core/system/src/common/main/sys_apps.c]
static void run_sys_apps(void)
{
... ...

/* Start user's network independent applications */
create_user_apps(sysmode, FALSE);
... ...
```

- Applications run after network connection is complete

```
[ ~/FreeRTOS_SDK/core/system/src/common/main/sys_apps.c ]
void start_user_apps(void)
{
int sysmode;
… …

/* Run user's network dependent apps */
create_user_apps(sysmode, TRUE);
}
```

All user applications are listed in the `user_apps_table[]` as shown in the example code below. There is a "hello_world" application in the SDK and the feature `__SUPPORT_HELLO_WORLD__` is defined in *~/FreeRTOS_SDK/apps/da16200/get_started/include/user_main/config_generic_sdk.h*.

```
[ ~/FreeRTOS_SDK/apps/da16200/get_started/src/apps/user_apps.c ]
const app_task_info_t user_apps_table[] = {
/* name, func, stack_size, pri, net_flag, dpm_flag, port_no, sys_mode */

#if defined (__SUPPORT_HELLO_WORLD__)
{ HELLO_WORLD_1, customer_hello_world_1, 64,(U_PRIO),FALSE,FALSE,UNDEF_PORT, ALL_MODE   },
{ HELLO_WORLD_2, customer_hello_world_2, 64,(U_PRIO),TRUE, FALSE,UNDEF_PORT, ALL_MODE   },
#endif  // __SUPPORT_HELLO_WORLD__

{ NULL, NULL,   0, 0, FALSE, FALSE, UNDEF_PORT, 0   }
};
```

- `HELLO_WORLD_1`: This application runs immediately regardless of network connection as shown in Figure 4.
- `HELLO_WORLD_2`: This application runs after network connection is completed as shown in Figure 4.

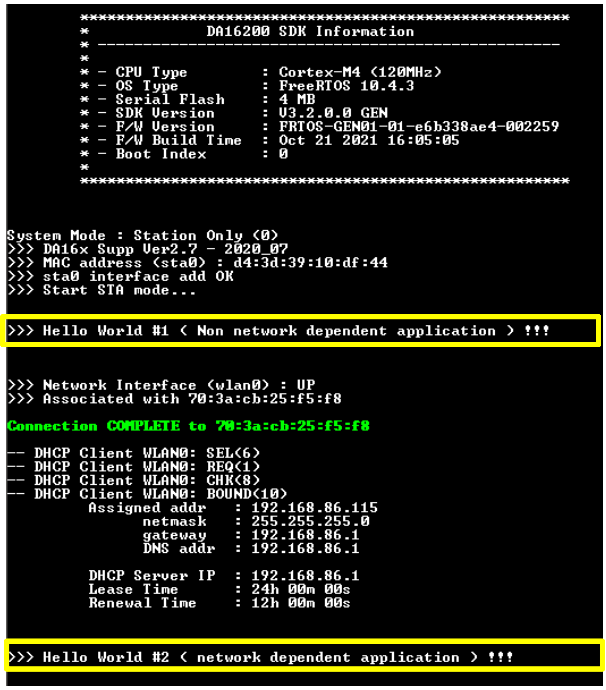


Figure 4. Results of running hello world applications

The applications described above can be reused or new source code can be added for new applications.

3.6 Sample Applications

The SDK contains various examples which demonstrate how to use DA16200 features. The examples included are:

- **Crypto**: Examples demonstrate how to use the cryptography and security capabilities
- **DPM**: Examples demonstrate how to use the various DPM low power mode
- **ETC**: Examples demonstrate how to get the current time, Access Point scan result
- **Network**: Examples demonstrate how to use various network protocols for either a client or server application
- **Peripheral**: Examples demonstrate how to use peripherals such as GPIO, I2C, and PWM

Before using the examples, set up the e^2 studio development environment. See Ref. [3] for details on setting up e^2 studio and importing the DA16200 SDK into that environment.

When the environment is set up, the examples can be found in the `apps/common/examples` directory. Each example directory has a similar structure and contains its own projects, one for da16200 and one for da16600, which can be imported into the e^2 studio environment.

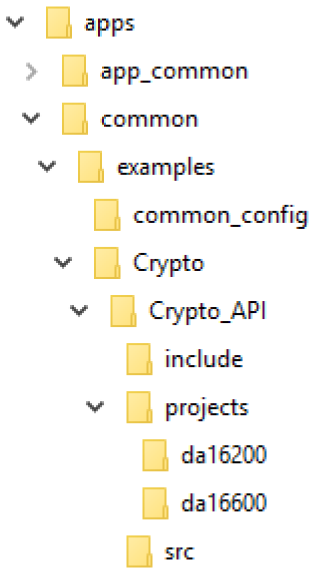


Figure 5. DA16200 SDK example

Select the desired example project folder and import it to e^2 studio. See Ref. [3] for how to import projects.

For example, the **Crypto_API** example project is located in the path below:

`~/SDK/Apps/common/examples/Crypto/Crypto_API/projects/da16200`

3.6.1 Wi-Fi Configuration for Sample Application

Each example using the Wi-Fi communication interface contains default configuration information. This information can be modified in the example code in the following location:

`[ ~/SDK/apps/common/examples/common_config/sample_preconfig.c ]`

NOTE
Each sample code runs with pre-configured Wi-Fi profile and environment variables in the NVRAM unless users want to add their codes in this file.

```
/* Sample for Customer's Wi-Fi configuration */
#define       SAMPLE_AP_SSID              "TEST_AP_SSID"
#define       SAMPLE_AP_PSK               "12345678"

// CC_VAL_AUTH_OPEN, CC_VAL_AUTH_WEP, CC_VAL_AUTH_WPA, CC_VAL_AUTH_WPA2, CC_VAL_AUTH_WPA_AUTO
#define SAMPLE_AP_AUTH_TYPE     CC_VAL_AUTH_WPA_AUTO

/* Required when WEP security mode */
#define SAMPLE_AP_WEP_INDEX     0

// CC_VAL_ENC_TKIP, CC_VAL_ENC_CCMP, CC_VAL_ENC_AUTO
#define SAMPLE_AP_ENCRPT_INDEX  CC_VAL_ENC_AUTO

void sample_preconfig(void)
{
    //
    // Need to change as Customer's profile information
```

```
    //


#if 0   // Example ...(Customer's code to config Wi-Fi profile for sample code)
    char reply[32];


    // Delete existed Wi-Fi profile
    da16x_cli_reply("remove_network 0", NULL, reply);


    // Set new Wi-Fi profile for sample test
    da16x_set_nvcache_int(DA16X_CONF_INT_MODE, 0);
    da16x_set_nvcache_str(DA16X_CONF_STR_SSID_0, SAMPLE_AP_SSID);
    da16x_set_nvcache_int(DA16X_CONF_INT_AUTH_MODE_0, SAMPLE_AP_AUTH_TYPE);

    if (SAMPLE_AP_AUTH_TYPE == CC_VAL_AUTH_WEP)
    {
        da16x_set_nvcache_str(DA16X_CONF_STR_WEP_KEY0 + SAMPLE_AP_WEP_INDEX, SAMPLE_AP_PSK);
        da16x_set_nvcache_int(DA16X_CONF_INT_WEP_KEY_INDEX, SAMPLE_AP_WEP_INDEX);
    }
    else if (SAMPLE_AP_AUTH_TYPE > CC_VAL_AUTH_WEP)
    {
        da16x_set_nvcache_str(DA16X_CONF_STR_PSK_0, SAMPLE_AP_PSK);
        da16x_set_nvcache_int(DA16X_CONF_INT_ENCRYPTION_0, SAMPLE_AP_ENCRPT_INDEX);
    }


    // Save new Wi-Fi profile to NVRAM area
    da16x_nvcache2flash();

    vTaskDelay(10);

    // Enable new sample Wi-Fi profile
    da16x_cli_reply("select_network 0", NULL, reply);

#endif  // 0
}
```

3.7 Build SDK

After the application is written, right-click on the project **DA16200/DA16600**, and then click **Build Project**. If building an SDK for the first time, Renesas recommends running command `Clean` first. See Figure 6.

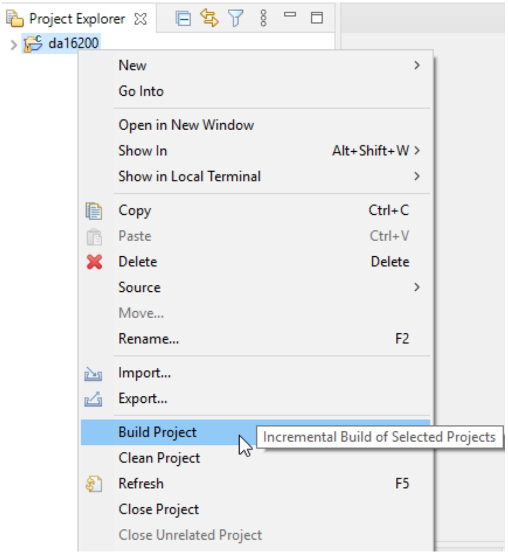


Figure 6. Build SDK on e² studio IDE

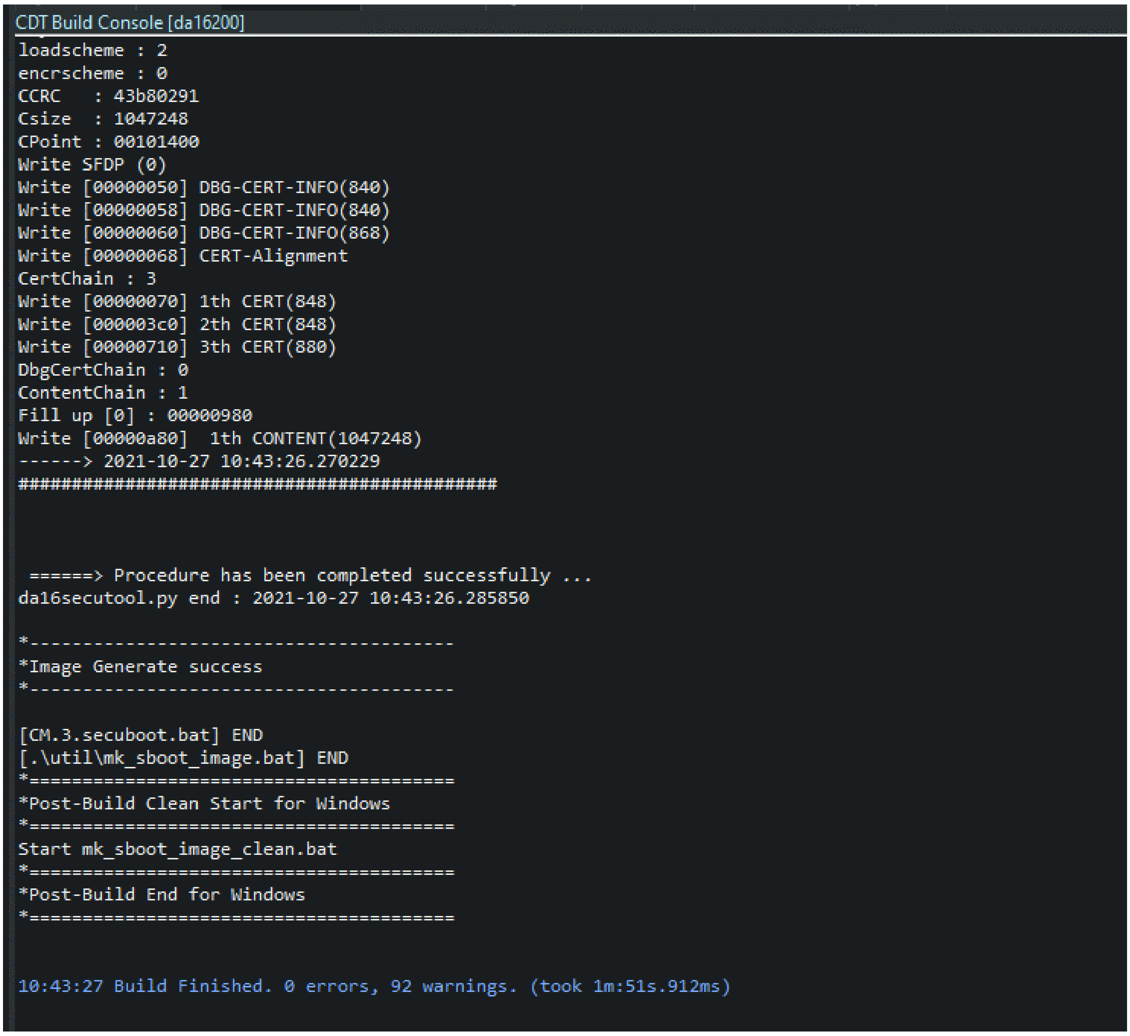


Figure 7. Build success on e² studio IDE

If the SDK is successfully built, two binary images are created in the `~/FreeRTOS_SDK/apps/da16200/get_started/img` folder. The names of the image files are:

- RTOS : DA16200_FRTOS-GEN01-01-XXXXXXXXX-000000.img
- 2nd Bootloader : DA16200_FBOOT-GEN01-01-XXXXXXXXX-000000_W25Q32JW.img (In case of Winbond W25Q32JW Sflash)

For more information about the firmware download, see the Programming Firmware Images section of Ref. [3].

3.7.1 Create RTOS Image for fcCSP

By default, the DA16200/DA16600 SDK provides a QFN-type RTOS Sflash image file. After building the DA16200/DA16600 SDK, the QFN-type RTOS image with filename **DA16200_FRTOS-GEN01-01-XXXXX-000000.img** is created in the **~/SDK/apps/da16200/get_started/img/** folder.

For fcCSP type package, to create an RTOS image with the DA16200/DA16600 SDK, change the build configurations to fcCSP_LP or fcCSP_NP. See the Build Configurations section of Ref. [3], and then follow the Build SDK instructions described in Section 3.7. For SDK version 3.2.7.1 or earlier and sample projects, see Appendix F.

When the programming is complete (see Ref. [3] for programming firmware), the SDK version shows "V3.2.x.0 CSP LP" for Low-Power or "V3.2.X.0 CSP NP" for Normal-Power. See Figure 8.

Figure 8. Boot logo with fcCSP-LP RTOS image

3.7.2 Build a Project Using Command Line

The command line of e^2 studio can be used to compile a project, see the following example command.

```
e2 studioc.exe -nosplash --launcher.suppressErrors -application
org.eclipse.cdt.managedbuilder.core.headlessbuild -data "c:\wksp" -cleanBuild MyProj (To
build the project in the workspace)
```

You can find the detailed information in the FAQ section: Command-line build of e^2 studio project.

4. Wake-Up Source

DA16200 DA16600 SDK supports various wake-up sources such as POR, system reset, external wake-up pin, and RTC wake-up counter. The wake-up sources (see Table 1) can be checked by calling `dpm_mode_get_wakeup_source()`,.and can be duplicated except power on reset, and each wake-up source can be categorized based on a defined Sleep mode and DPM. When the device wakes up from DPM, check the DPM wake-up types for further details (See Ref. [6]).

Table 1. Wake-up source

Wake-up source	Value	Wake-up from sleep	Description
WAKEUP_RESET	0x00		System reset
WAKEUP_SOURCE_EXT_SIGNAL	0x01	Sleep mode 2	External wake-up pin toggled
WAKEUP_SOURCE_WAKEUP_COUNTER	0x02	Sleep mode 2	RTC wake-up counter expired. (RTC wake-up counter sets the sleep period)
WAKEUP_EXT_SIG_WAKEUP_COUNTER	0x03	Sleep mode 2	External wake-up pin toggled and RTC wake-up counter expired.
WAKEUP_SOURCE_POR	0x04	Sleep mode 1	Power on reset
WAKEUP_WATCHDOG	0x08	Sleep mode 2	RTC watchdog expired. (RTC watchdog is not a CPU WDOG, and it wakes up if a device does not wake up when wake-up counter has expired.) (Note 1)
WAKEUP_WATCHDOG_EXT_SIGNAL	0x09	Sleep mode 2	RTC watchdog expired, and external wake-up pin toggled. (Note 1)
WAKEUP_SENSOR	0x10	Sleep mode 2	Wake-up GPIO toggled, pulse counter expired, or ADC sensor occurred. Return which wake-up source occurred by calling RTC_GET_AUX_WAKEUP_SOURCE() function. The return values are: 0x10: ADC sensor event 0x20: WAKEUP_PULSE 0x40: WAKEUP_GPIO See wakeup_sample.c in the SDK for details
WAKEUP_PULSE	0x20	Sleep mode 2	Pulse counter expired. The user can set the pulse count for wake-up, and when the count expires, the system wakes up. This wake-up source is subset of WAKEUP_SENSOR and should be read by calling RTC_GET_AUX_WAKEUP_SOURCE() when WAKEUP_SENSOR occurs.
WAKEUP_GPIO	0x40	Sleep mode 2	Wake-up GPIO toggled. This wake-up source is subset of WAKEUP_SENSOR and should be read by calling RTC_GET_AUX_WAKEUP_SOURCE() when WAKEUP_SENSOR occurs.
WAKEUP_SENSOR_EXT_SIGNAL	0x11	Sleep mode 2	External wake-up pin toggled and sensor (pulse or GPIO or ADC sensor) wake-up occurred.
WAKEUP_SENSOR_WAKEUP_COUNTER	0x12	Sleep mode 2	RTC wake-up counter expired and sensor (pulse or GPIO or ADC sensor) wake-up occurred.

Wake-up source	Value	Wake-up from sleep	Description
WAKEUP_SENSOR_EXT_WAKEUP_COUNTER	0x13	Sleep mode 2	RTC wake-up counter expired, external wake-up pin toggled, and sensor (pulse or GPIO or ADC sensor) wake-up occurred.
WAKEUP_SENSOR_WATCHDOG	0x18	Sleep mode 2	Sensor (pulse or GPIO or ADC sensor) wake-up occurred and RTC watchdog expired.
WAKEUP_SENSOR_EXT_WATCHDOG	0x19	Sleep mode 2	Sensor (pulse or GPIO or ADC sensor) wake-up occurred, RTC watchdog expired, and external wake-up pin toggled.
WAKEUP_RESET_WITH_RETENTION	0x80	N/A	System reset and the retention memory have valid data.
WAKEUP_EXT_SIG_WITH_RETENTION	0x81	Sleep mode 3 or DPM LPM	External wake-up pin toggled, and the retention memory has valid data.
WAKEUP_COUNTER_WITH_RETENTION	0x82	Sleep mode 3 or DPM LPM	RTC wake-up counter expired and the retention memory has valid data. (Note 2)
WAKEUP_EXT_SIG_WAKEUP_COUNTER_WITH_RETENTION	0x83	Sleep mode 3 or DPM LPM	External wake-up pin toggled, RTC wake-up counter expired, and the retention memory has valid data. (Note 2)
WAKEUP_WATCHDOG_WITH_RETENTION	0x88	Sleep mode 3 or DPM LPM	RTC watchdog expired, and the retention memory has valid data. (Note 1)
WAKEUP_SENSOR_WITH_RETENTION	0x90	Sleep mode 3 or DPM LPM	Sensor (pulse or GPIO or ADC sensor) wake-up occurred and the retention memory has valid data.
WAKEUP_SENSOR_EXT_SIGNAL_WITH_RETENTION	0x91	Sleep mode 3 or DPM LPM	Sensor (pulse or GPIO or ADC sensor) wake-up occurred, external wake-up pin toggled, and the retention memory has valid data.
WAKEUP_SENSOR_WAKEUP_COUNTER_WITH_RETENTION	0x92	Sleep mode 3 or DPM LPM	Sensor (pulse or GPIO or ADC sensor) wake-up occurred, RTC wake-up counter expired, and the retention memory has valid data. (Note 2)
WAKEUP_SENSOR_EXT_WAKEUP_COUNTER_WITH_RETENTION	0x93	Sleep mode 3 or DPM LPM	Sensor (pulse or GPIO or ADC sensor) wake-up occurred, RTC wake-up counter expired, external wake-up pin toggled, and the retention memory has valid data. (Note 2)
WAKEUP_SENSOR_WATCHDOG_WITH_RETENTION	0x98	Sleep mode 3 or DPM LPM	Sensor (pulse or GPIO or ADC sensor) wake-up occurred, RTC watch dog expired, and the retention memory has valid data. (Note 1)
WAKEUP_SENSOR_EXT_WATCHDOG_WITH_RETENTION	0x99	Sleep mode 3 or DPM LPM	Sensor (pulse or GPIO or ADC sensor) wake-up occurred, RTC watch dog expired, external wake-up pin toggled, and the retention memory has valid data. (Note 1)

Note 1 The wake-up source is deprecated.

Note 2 PTIM works through this wake-up source, thus users need to see the DPM wake-up types in Ref. [6] after waking up.

NOTE
There are exceptional cases for system faults and CPU watchdog which were set by SDK. ▪ 0x00: Bus fault or memory corruption. ▪ 0x04: CPU watchdog.

5. NVRAM

The DA16200/DA16600 has an NVRAM area on the flash memory to store system data and user data. NVRAM has various system configuration parameters to control the Wi-Fi function.

5.1 API

There are two types of NVRAM: integer and string. Use the following functions based on the datatype that is currently used.

Table 2. APIs for NVRAM

Item		Description
int write_nvram_int(const char *name, int val)		
Parameter	name	NVRAM item name to write.
	value	Integer value to write.
Return		If it succeeds, return 0. If it fails, return an error code.
Description		Write a specific NVRAM item with an integer value.
int write_nvram_string(const char *name, const char *val)		
Parameter	name	NVRAM item name to write.
	value	Pointer to the string buffer to write.
Return		If it succeeds, return 0. If it fails, return an error code.
Description		Write a specific NVRAM item with a string value.
int read_nvram_int(const char *name, int *_val)		
Parameter	name	NVRAM item name to read.
	value	Pointer to the integer value to read the value.
Return		If it succeeds, return 0. If it fails, return an error code.
Description		Read an integer value of a specific NVRAM item.
char *read_nvram_string(const char *name)		
Parameter	name	NVRAM item name to get.
	value	Pointer to the string buffer to read the value.
Return		If it succeeds, return 0. If it fails, return an error code.
Description		Read a string value of a specific NVRAM item.

6. TLS Certificate

Certificates are required to make secure connections and can be used in MQTT client, HTTPs client/server, WPA Enterprise, and TLS client/server. The secure applications except the TLS client/server are designed and implemented to read certificates from prefixed areas of SFlash map in Ref. [3]. In the case of the TLS client/server application, it can use the user area in serial flash as no prefixed area is allocated.

6.1 Certificate for MQTT Client

The DA16200/DA16600 has prefixed areas in flash for certificates of MQTT client. The address map of each certificate is defined in `da16200_map.h` of SDK. See `my_app_mqtt_user_config` in `mqtt_client_sample.c` in Section 13.5.

```
/* TLS Certificate Key #0 */
#define SFLASH_ROOT_CA_ADDR1                    0x003A3000
#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)
```

6.2 Certificate for WPA Enterprise

The DA16200/DA16600 has prefixed areas in flash for certificates of WPA Enterprise. The address map of each certificate is defined in `da16200_map.h` of SDK. The certificates can be stored using console commands in Section 6.6.1.

```
/* TLS Certificate WPA Enterprise */
#define SFLASH_ENTERPRISE_ROOT_CA               0x003ED000
#define SFLASH_ENTERPRISE_CERTIFICATE           (SFLASH_ENTERPRISE_ROOT_CA + 0x1000)
#define SFLASH_ENTERPRISE_PRIVATE_KEY           (SFLASH_ENTERPRISE_ROOT_CA + 0x2000)
#define SFLASH_ENTERPRISE_DH_PARAMETER          (SFLASH_ENTERPRISE_ROOT_CA + 0x3000)
```

6.3 Certificate for HTTPs Client/Server or OTA

The DA16200/DA16600 has prefixed areas in flash for certificates of HTTPs or OTA. The address map of each certificate is defined in `da16200_map.h` of SDK. See `http_client_read_certs` in `http_client_sample.c` in Section 14.5.

```
/* TLS Certificate Key #1 */
#define SFLASH_ROOT_CA_ADDR2                    0x003A7000
#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)
```

6.4 Certificate for TLS Client/Server

The DA16200/DA16600 does not have prefixed areas for certificates of TLS client/server. User area of flash can be used for certificates using flash APIs directly. See `tls_sever_samples.c` on how to use the certificate as constant data in Section 12.2.

6.5 APIs for Accessing Prefixed Area of SFlash

The prefixed area of flash for certificates can be accessed using APIs shown in Table 3, Table 4, and Table 5.

Table 3. APIs for reading certificate from flash

Item		**Description**
int da16x_cert_read(int module, int type, int *format, unsigned char *out, size_t *outlen)		
Parameter	module	Module ID: 0 – MQTT 1 – HTTPs client or OTA 2 – WPA Enterprise
	type	Certificate type: 0 – CA certificate 1 – Certificate 2 – Private key 3 – DH params
	format	Certificate format: 0 – DER 1 – PEM
	out	Pointer to read certificate.
	outlen	Length of certificate.
Return		If it succeeds, return 0. If it fails, return an error code.
Description		Read certificate from specific SFlash memory by module and type.
int da16x_cert_read_no_fopen(HANDLE flash_handler, int module, int type, int *format, unsigned char *out, size_t *outlen)		
Parameter	flash_handler	Handler to read certificate. It must be open.
	module	Module ID: 0 – MQTT 1 – HTTPs client or OTA 2 – WPA Enterprise
	type	Certificate type: 0 – CA certificate 1 – Certificate 2 – Private key 3 – DH params
	format	Certificate format: 0 – DER 1 – PEM
	out	Pointer to read certificates.
	outlen	Length of certificate
Return		If it succeeds, return 0. If it fails, return an error code.
Description		Read certificate from specific SFlash memory by module and type.

Table 4. API to write certificate to flash

Item		Description
int da16x_cert_write(int module, int type, int format, unsigned char *in, size_t inlen)		
Parameter	module	Module ID: 0 – MQTT 1 – HTTPs client or OTA 2 – WPA Enterprise
	type	Certificate type: 0 – CA certificate 1 – Certificate 2 – Private key 3 – DH params
	format	Certificate format: 0 – DER 1 – PEM
	in	Pointer to write certificate.
	inlen	Length of certificate.
Return		If it succeeds, return 0. If it fails, return an error code.
Description		Write certificate to specific SFlash memory address by module and type.

Table 5. APIs to delete certificate in flash

Item		Description
int da16x_cert_delete(int module, int type)		
Parameter	module	Module ID: 0 – MQTT 1 – HTTPs client or OTA 2 – WPA Enterprise
	type	Certificate type: 0 – CA certificate 1 – Certificate 2 – Private key 3 – DH params
Return		If it succeeds, return 0. If it fails, return an error code.
Description		Delete certificate from specific SFlash memory by module and type.
int da16x_cert_delete_no_fopen(HANDLE flash_handler, int module, int type)		
Parameter	flash_handler	Handler to read certificate. It must be open.
	module	Module ID: 0 – MQTT 1 – HTTPs client or OTA 2 – WPA Enterprise

Item		Description
int da16x_cert_delete(int module, int type)		
	type	Certificate type: 0 – CA certificate 1 – Certificate 2 – Private key 3 – DH params
Return		If it succeeds, return 0. If it fails, return an error code.
Description		Delete certificate from specific SFlash memory by module and type.

6.6 Store Certificates to Flash Using Console Command

The DA16200/DA16600 provides methods to store certificates in the serial flash with the use of console command.

6.6.1 Console Command for Certificate

Table 6. Console command for Certificate

Command	Parameters	Description
`cert`	`<action> <dest>`	Certificate console command. ▪ <action>: status | write | read | del • status: Certificate status. • write: write certificate in SFlash • read: read certificate in SFlash • del: del certificate in SFlash ▪ <dest>: Pre-fixed destination area in SFlash. • ca#: root CA (#1~3) • cert#: server/client certificate (#1~3) • key#: private key (#1~3) • dh#: DH parameter (#1~3) • all: all certificates for del in <action> #: 1:MQTT/CoAP, 2: HTTPs/OTA, 3: Enterprise

6.6.2 Store Certificates

1. Store a CA certificate.

```
[/DA16200/NET]# net
[/DA16200/NET]# cert write ca1 // ca1: MQTT/ CoAP , ca2: HTTPs/OTA, ca3: Enterprise
Typing data: (certificate value)
Cancel - CTRL+D, End of Input - CTRL+C or CTRL+Z
// Copy & paste certificate data in the terminal window and press "CTRL+C" or "CTRL+Z"
(see Section 6.6.3)
```

2. Store a client certificate.

```
[/DA16200/NET]# cert write cert1 // cert1: MQTT/CoAP, cert2: HTTPs/OTA, cert3: Enterprise
Typing data: (certificate value)
Cancel - CTRL+D, End of Input - CTRL+C or CTRL+Z
// Copy & paste certificate data in the terminal window and press "CTRL+C" or "CTRL+Z"
(see Section 6.6.3)
```

3. Store a client key.

```
[/DA16200/NET]# cert write key1 // key1: MQTT/CoAP, key2: HTTPs/OTA, key3: Enterprise
```

```
Typing data: (certificate value)
Cancel - CTRL+D, End of Input - CTRL+C or CTRL+Z
// Copy & paste certificate data in the terminal window and press "CTRL+C" or "CTRL+Z"
(see Section 6.6.3)
```

4. After adding cert/keys, check if they are successfully stored.

```
[/DA16200/NET]# cert status
#1:
  For MQTT, CoAPs Client
  - Root CA      : Found
  - Certificate : Found
  - Private Key : Found
  - DH Parameter: Empty
#2:
  For HTTPs, OTA
  - Root CA      : Empty
  - Certificate : Empty
  - Private Key : Empty
  - DH Parameter: Empty
#3:
  For Enterprise (802.1x)
  - Root CA      : Empty
  - Certificate : Empty
  - Private Key : Empty
  - DH Parameter: Empty
```

5. In case remove all the credentials stored:

```
[/DA16200/NET] # cert del all

all Delete success.
```

6.6.3 Root CA, Client Cert, and Private Key

Certificate format follows X.509 standard and should input the new line character in BEGIN and END lines.

Figure 9, Figure 10 and Figure 11 show the example certificates.

6.6.3.1 Root CA

```
-----BEGIN CERTIFICATE-----LF
MIIDLTCCApagAwIBAgIHFUYwCVQnWDANBgkqhkiG9w0BAQsFADCBpzELMAkGA1UELF
BhMCVFcxDzANBgNVBAgMBlRhaXdhbjEPMA0GA1UEBwwGVGFpcGVpMRYwFAYDVQQKLF
DA1TeW5vbG9neSBJbmMuMR4wHAYDVQQLDBVDZXJ0aWZpY2F0ZSBBdXRob3JpdHkxLF
GTAXBgNVBAMMEFN5bm9sb2d5IEluYy4gQ0ExIzAhBgkqhkiG9w0BCQEWFHByb2R1LF
Y3RAc3lub2xvZ3kuY29tMB4XDTE5MDEwMTAwMDIzNFoXDTM4MDkxODAwMDIzNFowLF
gZYxCzAJBgNVBAYTAlRXMQ8wDQYDVQQIDAZUYWl3YW4xDzANBgNVBAcMBlRhaXBlLF
aTEWMBQGA1UECgwNU3lub2xvZ3kgSW5jLjERMA8GA1UECwwIRlRQIFRlYW0xFTATLF
BgNVBAMMDHN5bm9sb2d5LmNvbTEjMCEGCSqGSIb3DQEJARYUcHJvZHVjdEBzeW5vLF
bG9neS5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBANG7EAgRg+8012IiLF
RV0CddJdeUCAlIDcwgqVPQZ9ox7vXL6jTunfbbsNQO5KHuoKSCmUqcjv9u7joNCyLF
52Rg4fxzs2dwsK6sLNNjkrwlCAKk0i0QAzenJvAHYkfp8ks+cHyWmevokGVL7y6oLF
Uux3sJpblmwcklNLeuO5odPJN3rVAgMBAAGjcjBwMB8GA1UdEQQYMBaBFHByb2R1LF
Y3RAc3lub2xvZ3kuY29tMDoGCWCGSAGG+EIBDQQtFittb2Rfc3NsIGdlbmVyYXRlLF
ZCBjdXN0b20gc2VydmVyIGNlcnRpZmljYXRlMBEGCWCGSAGG+EIBAQQEAwIGQDANLF
BgkqhkiG9w0BAQsFAAOBgQBkM4OqcL/rD93dCDPJMTtO90w7FtvpWXrgqYdHupNdLF
FDEcmqgLnoIey4seFs2pMnwO8LLFDecvIc+jeym9Mt9I3wWE5eAQkKSX6k/eVn20LF
1WIaMw/R1CRcNAppzc2wIJy9o+W9PYu2FJY/nbl5k/3Yqb6Zu3FGuKEffgJ6/kSeLF
Aw==LF
-----END CERTIFICATE-----
```

Figure 9. Root CA example

6.6.3.2 Client Cert


```
-----BEGIN CERTIFICATE-----
MIIDTTCCAragAwIBAgIJAO9DpxXUbkL7MA0GCSqGSIb3DQEBCwUAMIGnMQswCQYD
VQQGEwJUVzEPMA0GA1UECAwGVGFpd2FuMQ8wDQYDVQQHDAZUYWlwZWkxFjAUBgNV
BAoMDVN5bm9sb2d5IEluYy4xHjAcBgNVBAsMFUNlcnRpZmljYXRlIEF1dGhvcml0
eTEZMBcGA1UEAwwQU3lub2xvZ3kgSW5jLiBDQTEjMCEGCSqGSIb3DQEJARYUcHJv
ZHVjdEBzeW5vbG9neS5jb20wHhcNMTkwMTAxMDAwMjMzWhcNMzgwOTE4MDAwMjMz
WjCBpzELMAkGA1UEBhMCVFcxDzANBgNVBAgMBlRhaXdhbjEPMA0GA1UEBwwGVGFp
cGVpMRYwFAYDVQQKDA1TeW5vbG9neSBJbmMuMR4wHAYDVQQLDBVDZXJ0aWZpY2F0
ZSBBdXRob3JpdHkxGTAXBgNVBAMMEFN5bm9sb2d5IEluYy4gQ0ExIzAhBgkqhkiG
9w0BCQEWFHByb2R1Y3RAc3lub2xvZ3kuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GN
ADCBiQKBgQC5rEQPDlatQu4ppb7mbEJuSyq1kOY2/5WUa0SVmAiRqh47pDa3Dxc6
lVrUqxVScg3fcXiqcaoV41TBi0dLnuw3YHPQHd6w7KGX1eSSxEDAEljUna05bRBD
fBXAX+BHBx/avYpszZT3GLtLREc6Gf62q+yS7/f5S0qA0h81R0vPdQIDAQABo38w
fTAfBgNVHREEGDAWgRRwcm9kdWN0QHN5bm9sb2d5LmNvbTAPBgNVHRMECDAGAQH/
AgEAMDYGCWCGSAGG+EIBDQQpFidtb2Rfc3NsIGdlbmVyYXRlZCBjdXN0b20gQ0Eg
Y2VydGlmaWNhdGUwEQYJYIZIAYb4QgEBBAQDAgIEMA0GCSqGSIb3DQEBCwUAA4GB
AFpG2Du8sWPMcunulbuajwJ3W5bNeJI5o3TNVqOlKbhLMvwBdAymd3RJhouONuQt
6NEbAZi5V2pomIy+04tMhTlc2HktRMHK9T1q903bjDQe/kEQ1KQCdv6Qvj3WOVUC
KBBEJGDZGJMZ3sUmi0L/YAN+M3Jl4I999kt/4PvobuhP
-----END CERTIFICATE-----
```


Figure 10. Client certificate example

6.6.3.3 Private Key


```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQC5rEQPDlatQu4ppb7mbEJuSyq1kOY2/5WUa0SVmAiRqh47pDa3
Dxc6lVrUqxVScg3fcXiqcaoV41TBi0dLnuw3YHPQHd6w7KGX1eSSxEDAEljUna05
bRBDfBXAX+BHBx/avYpszZT3GLtLREc6Gf62q+yS7/f5S0qA0h81R0vPdQIDAQAB
AoGAXxiJ6yQu5KKpFI+djKraAl9gSooiPvz8gX3HLM8nZGO7BFUA6RVDkm+4i7/s
NcPQsyaM5M8+i+NZH5/ULjkBRiMA31kFWxoHbe/uF0QNoYfBUgHaPTk/bWZS1/Fe
DGoNWBCP/Karm9J9iAgkOqHhKcvMY1M4Z3btN/pr/X0GooECQQDyPPGMg2PLG1a9
P7ROQIj8rJycGfwoyU2sMFmF4MsMvj5LfTIMQzt4MD7hf7FxIfRZzmQpGhSN4E+V
Cice5X4FAkEAxDimwYWdGDshgtbBEPhJWQqElc6c9L2FXW6Hao5GNWYn4PWvHov/
ZdZV+KyDkGmY7jyDb9jyNsBeLaFzEUJWsQJACxavF/eLwehmeBNKdaqOmsJsCNlH
qbZNT+yGrJIluUecImiAuf754e6Tck8eGEetn87K1vbSqgvymKf2Unu+4QJALu3F
pB8K1VL/DsXB4wces7b6QsHpc/cnwFLA/FEOpZLLEthvi4S8DmUWpqTLymwW8VNz
s/pU45dabttgI86T4QJBAJCyrQF3d2F6ybrBggi4hiF4DyQDkAWlsnvQrFM8Q241
eNcPQkgdXW+4lbfgk6MRDQ5EOKPnPtmJh8yh51DqtuM=
-----END RSA PRIVATE KEY-----
```


Figure 11. Private key example

7. Hardware Accelerators

7.1 Set SRAM to Zero

7.1.1 API

Table 7. Hardware accelerator API

Item		Description
void da16x_memset32(UINT32 *data, UINT32 seed, UINT32 length)		
Parameter	data	Buffer pointer to set
	seed	Value to fill
	length	Length
Return		None
Description		Fill up memory with a certain value through hardware acceleration

7.1.2 Sample Code

```
#include <hal.h>

/* fill up a 1024 bytes buffer memory with 0 */
UINT32 buffer[1024];
da16x_memset32(buffer, 0, 1024);
```

7.2 CRC Calculation

7.2.1 API

Table 8. CRC API

Item		Description
UINT32 da16x_hwcrc32(UINT32 dwidth, UINT8 *data, UINT32 length, UINT32 seed)		
Parameter	dwidth	Data width to calculate CRC
	data	Data pointer
	length	Length
	seed	CRC32 seed value (default value is 0xFFFFFFFF).
Return		Calculated CRC32 value.
Description		Calculate CRC through hardware accelerator

7.2.2 Sample Code

```
#include <hal.h>

/* calculate a CRC value of data buffer */
UINT8 data[64], i;
For (i=0; I < 64; i++)
           data[i] = I;
UINT32 crc_value = da16x_hwcrc32(sizeof(UINT32), (void *)data, sizeof(data), (~0));
```

7.3 Pseudo Random Number Generator (PRNG)

7.3.1 API

Table 9. PRNG API

Item		Description
UINT32 da16x_random(void)		
Parameter	None	
Return		32 bits random value
Description		Generate 32 bits random value with hardware accelerator

7.3.2 Sample Code

```
#include <hal.h>

UINT32 random = da16x_random();
```

7.4 Memory Copy Using DMA

7.4.1 API

Table 10. Hardware DMA API

Item		Description
int memcpy_dma (void *dest, void *src, unsigned int len, unsigned int wait_time)		
Parameter	dest	A pointer to the location where the function copies the data (4 B aligned).
	src	A pointer to the buffer where to copy data from (4 B aligned).
	len	The number of bytes to copy.
	wait_time	0: After starting DMA operation, return from function. N: Wait until the memory copy is finished. If DMA operation time is greater than N milliseconds, the function returns after N milliseconds. N must have a value of at least 10 ms.
Return		Always 0.
Description		Copy bytes from one buffer to another using DMA.

7.4.2 Sample Code

```
#include <sys_dma.h>

char dest[100], src[100]

memcpy_dma(dest, src, 100, 0);
```

8. Watchdog Service

8.1 Overview

The system watchdog service (da16x_sys_watchdog) is designed to monitor system tasks and avoid system freezes. Figure 12 shows how to interact with the system.

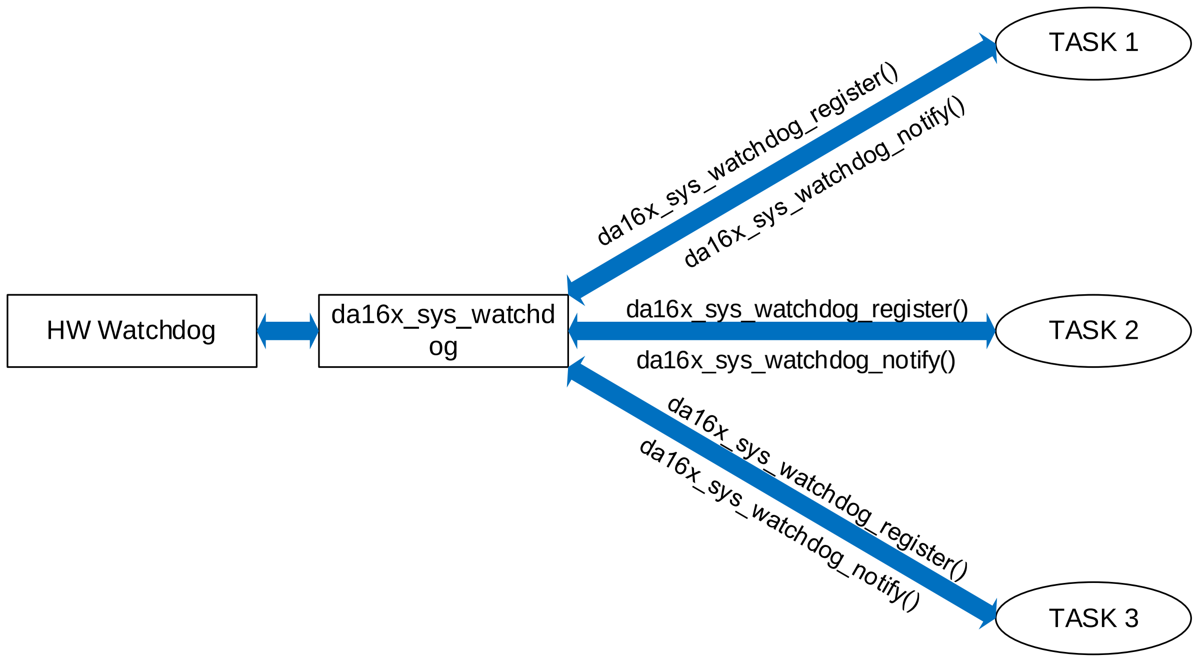


Figure 12. Watchdog overview

The `da16x_sys_watchdog` is a layer located on top of the watchdog low-level driver that allows multiple tasks to share the underlying hardware watchdog timer. The watchdog service can be used to trigger a full system reset. This allows the system to recover from a catastrophic failure in one or more tasks.

8.2 Concept

To monitor a task, register the task with `da16x_sys_watchdog` to receive a unique handle (id). Then, it periodically notifies `da16x_sys_watchdog` using the id to signal that the task is working properly. When an error occurs during the registration process, it returns -1.

The DA16200 Watchdog Timer is essentially a simple countdown timer (based on CMSDK Watchdog Timer) that triggers a full system reset if it expires. The watchdog timer interrupt is Non-Maskable Interrupt (NMI). That is, the interrupt cannot be disabled and should be controlled by Lock/Unlock process. To prevent this, the watchdog timer must be reset to its starting value before it expires. This starting value can be configured through the numerical macro `DA16X_SYS_WDOG_DEF_RESCALE_TIME` or `da16x_sys_watchdog_set_rescale_time()` in `da16x_sys_watchdog.h` file. The default value is 5 seconds. `DA16X_SYS_WDOG_MAX_TASKS_CNT` defines the maximum number of tasks that can be monitored.

If all monitored tasks during one watchdog period notify `da16x_sys_watchdog`, the hardware watchdog is updated. In this case, no platform reset is triggered for this watchdog period. However, a platform reset is triggered if at least one task does not notify `da16x_sys_watchdog` in time. There are two ways for a task to notify `da16x_sys_watchdog`.

Each task is responsible for periodically notifying `da16x_sys_watchdog` that it is still running using `da16x_sys_watchdog_notify()`. This must be done before the watchdog timer expires. Occasionally, a registered task may want to temporarily exclude itself from being monitored if it expects to be blocked for a long time waiting for an event. This is done using `the da16x_sys_watchdog_suspend()`. This function suspends monitoring of specific tasks in `da16x_sys_watchdog`, as there is no need to monitor a task that is blocked waiting for an event that might take too long to occur (for example, it leads to the task failed to notify the watchdog service, thus resulting in a system reset). When the task is unblocked, the `da16x_sys_watchdog_resume()`

should be called to restore task monitoring by the watchdog service. From that moment, the task should notify the watchdog service as usual.

Finally, the intension of `da16x_sys_watchdog_set_latency()` is to be used in cases where a task requires a watchdog period greater than the configured watchdog timer reset value. Using this API allows a task to delay notification of `da16x_sys_watchdog` for a given number of watchdog periods, without triggering a system reset. The effect of calling the API is one-off and therefore, it must be set every time increased latency is required.

8.3 API

Table 11. APIs of watchdog service

Item		Description
int da16x_sys_watchdog_init(void)		
Return		If it succeeds, return 0. If it fails, return an error code.
Description		Initialize da16x_sys_watchdog module.
int da16x_sys_watchdog_register(unsigned int notify_trigger)		
Parameter	notify_trigger	True if task notification should be triggered periodically. It is not supported yet.
Return		Identifier on success, -1 on failure.
Description		Register current task in da16x_sys_watchdog module.
int da16x_sys_watchdog_unregister(int id)		
Parameter	id	Identifier
Return		If it succeeds, return 0. If it fails, return an error code.
Description		Unregister task from da16x_sys_watchdog module.
void da16x_sys_watchdog_configure_idle_id(int id)		
Parameter	id	Identifier
Return		None
Description		Inform the da16x_sys_watchdog module of the watchdog ID for the IDLE task.
int da16x_sys_watchdog_suspend(int id)		
Parameter	id	Identifier
Return		0 on success.
Description		Suspend task monitoring in `da16x_sys_watchdog` module.
int da16x_sys_watchdog_resume(int id)		
Parameter	id	Identifier
Return		0 on success.
Description		Resume task monitoring in da16x_sys_watchdog module. This function does not notify the watchdog service for the task. It is possible that monitor resuming occurs too close to the time that the watchdog expires, before the task has a chance to explicitly send a notification. This can lead to an unwanted reset. Therefore, either call `da16x_sys_watchdog_notify()` before calling `da16x_sys_watchdog_resume()` or use `da16x_sys_watchdog_notify_and_resume()` instead.
int da16x_sys_watchdog_notify(int id)		
Parameter	id	Identifier
Return		0 on success.

Item		Description
Description		Notify `da16x_sys_watchdog` module for task. Registered task shall use this periodically to notify `sys_watchdog` module that it is alive. This should be done frequently enough to fit into `hw_watchdog` interval.
int da16x_sys_watchdog_notify_and_resume(int id)		
Parameter	id	Identifier
Return		0 on success.
Description		Notify `da16x_sys_watchdog` module for task with handle \p id and resume its monitoring. This function combines the functionality of `da16x_sys_watchdog_notify()` and `da16x_sys_watchdog_resume()`.
int da16x_sys_watchdog_set_latency(int id, unsigned char latency)		
Parameter	id	Identifier
	latency	Latency
Return		0 on success.
Description		Set watchdog latency for task. This allows a task to miss given number of notifications to `da16x_sys_watchdog` without triggering platform reset. When set, it is allowed that task does not notify `sys_watchdog` for latency consecutive `hw_watchdog` intervals which can be used to allow for parts of code which are known to block for long period of time (for example, computation). This value is set once and does not reload automatically, thus it shall be set every time increased latency is required.
int da16x_sys_watchdog_set_rescale_time(unsigned int rescale_time)		
Parameter	rescale_time	Rescale time (unit of times: 10 milliseconds).
Return		0 on success.
Description		Set watchdog rescale time.
int da16x_sys_watchdog_get_rescale_time(unsigned int rescale_time)		
Parameter		None
Return		Rescale time (unit of times: 10 milliseconds).
Description		Get watchdog rescale time.

8.4 Sample Code

To register the task with `da16x_sys_watchdog`, use the following code snippet:

```
#include <da16x_sys_watchdog.h>

/* Registration a task to be monitored by watchdog */
wdog_id = sys_watchdog_register(false);
```

To notify `da16x_sys_watchdog`, use `da16x_watchdog_notify()`. If the task is going to suspend for an event, then temporarily exclude the current task from being monitored using `da16x_sys_watchdog_suspend()`. When the task has received an event, it can resume its watchdog operation with `da16x_sys_watchdog_resume()`. See the following flow:

```
/* Notify watchdog on each loop since there is no other trigger for this –
 * monitoring will be suspended while blocking on xTaskNotifyWait()
 */
da16x_sys_watchdog_notify(wdog_id);
/*
 * Wait on any of the event group bits, then clear them all
 */
da16x_sys_watchdog_suspend(wdog_id);
```

```
ret = xTaskNotifyWait(0, 0xFFFFFFFF, &notif, portMAX_DELAY);
/* Blocks forever waiting for the task notification.
 * Therefore, the return value must always be pdPASS.
 */
da16x_sys_watchdog_resume(wdog_id);
```

9. Wi-Fi Interface Configuration

The DA16200/DA16600 SDK defines various parameters for Wi-Fi interface configuration, and they are saved as profiles in the NVRAM. After system reset, the DA16200/DA16600 reads an existing profile and sets the Wi-Fi interface based on that profile. Wi-Fi interface can be configured through API, Soft AP configuration, and Soft AP provisioning.

9.1 API

The DA16200/DA16600 SDK provides various functions to get or set system profiles:

- Simple functions to get or set a value (integer type or string type) of the name parameter (NVRAM item index).
- Error code to verify the result.

Table 12. APIs for Wi-Fi configuration

Item		Description
int da16x_set_config_int(int name, int value)		
Parameter	name	Parameter index to set.
	value	Integer value to set.
Return		If it succeeds, return 0 (CC_SUCCESS). If it fails, return an error code.
Description		Set a specific parameter with an integer value. For example: ret = da16x_set_config_int (Da16x_CONF_INT_CHANNEL, 11) ▪ Set the operating channel of the AP interface to 11.
int da16x_set_config_str (int name, char *value)		
Parameter	name	Parameter index to set.
	value	Pointer to the string value to set.
Return		If it succeeds, return 0 (CC_SUCCESS). If it fails, return an error code.
Description		Set a specific parameter with a string value. For example: ret = da16x_set_config_str(Da16x_CONF_STR_IP_0, “10.0.0.1”) ▪ Set the IP address of the STA interface to 10.0.0.1.
int da16x_get_config_int (int name, int *value)		
Parameter	name	Parameter index to get.
	value	Pointer to the integer variable to get the parameter value.
Return		If it succeeds, return 0 (CC_SUCCESS). If it fails, return an error code.
Description		Get an integer value of a specific parameter. For example: ret = da16x_get_config_int(Da16x_CONF_INT_CHANNEL, &channel) ▪ Get the operating channel of the AP interface.
int da16x_get_config_str (int name, char *value)		
Parameter	name	Parameter index to get.
	value	Pointer to the string buffer to get the parameter value.
Return		If it succeeds, return 0 (CC_SUCCESS). If it fails, return an error code.
Description		Get a string value of a specific parameter. For example: ret = da16x_get_config_str(Da16x_CONF_STR_IP_0, ip_addr) ▪ Get the IP address of the STA interface.
int da16x_set_nvcache_str(int name, char *value)		
Parameter	name	Parameter name to set.
	value	Points to the value (str) to set.

Item		Description
Return		If it succeeds, return 0 (CC_SUCCESS). If it fails, return an error code.
Description		Set name/value pair to NVRAM cache area (not in SFlash). To make it permanent, invoke da16x_nvcache2flash(). For example: ret = da16x_set_nvcache_str(Da16x_CONF_STR_IP_0, ip_addr) ▪ Set IP address of the STA interface.
int da16x_set_nvcache_int(int name, int value)		
Parameter	name	Parameter name to set.
	value	Points to the value (int) to set.
Return		If it succeeds, return 0 (CC_SUCCESS). If it fails, return an error code.
Description		Set name/value pair to NVRAM cache area (not in SFlash). To make it permanent, invoke da16x_nvcache2flash (). For example: ret = da16x_set_nvcache_int(Da16x_CONF_INT_CHANNEL, 11) Set the operating channel of the AP interface to 11.
void da16x_nvcache2flash(void)		
Parameter	None	
Return		None
Description		Commit parameters (set by da16x_set_nvcache_int/str) in NVRAM cache to flash.

9.1.1 Integer Type Parameters

Table 13. NVRAM integer type

Name	Description
DA16X_CONF_INT_MODE	Wi-Fi operation mode: 0: STA 1: Soft AP 2. Soft AP + STA (Concurrent mode)
DA16X_CONF_INT_AUTH_MODE_0	Wi-Fi authentication mode for STA interface: ▪ CC_VAL_AUTH_OPEN ▪ CC_VAL_AUTH_WEP ▪ CC_VAL_AUTH_WPA ▪ CC_VAL_AUTH_WPA2 ▪ CC_VAL_AUTH_WPA_AUTO (WPA & WPA2) ▪ CC_VAL_AUTH_WPA_EAP ▪ CC_VAL_AUTH_WPA2_EAP ▪ CC_VAL_AUTH_WPA_AUTO_EAP
DA16X_CONF_INT_AUTH_MODE_1	Wi-Fi authentication mode for Soft AP interface: ▪ CC_VAL_AUTH_OPEN ▪ CC_VAL_AUTH_WPA ▪ CC_VAL_AUTH_WPA2 ▪ CC_VAL_AUTH_WPA_AUTO (WPA & WPA2) (WEP is unsupported on the DA16200/DA16600 AP mode)
DA16X_CONF_INT_WEP_KEY_INDEX	Wi-Fi WEP key index number (0~3)
DA16X_CONF_INT_ENCRYPTION_0	Wi-Fi data encryption mode for STA interface: ▪ CC_VAL_ENC_TKIP ▪ CC_VAL_ENC_CCMP ▪ CC_VAL_ENC_AUTO (TKIP & CCMP)

Name	Description
DA16X_CONF_INT_ENCRYPTION_1	Wi-Fi data encryption mode for Soft AP interface: ▪ CC_VAL_ENC_TKIP ▪ CC_VAL_ENC_CCMP ▪ CC_VAL_ENC_AUTO (TKIP & CCMP)
DA16X_CONF_INT_WIFI_MODE_0	Wi-Fi mode based on IEEE 802.11 standard for STA interface: ▪ CC_VAL_WFMODE_BGN ▪ CC_VAL_WFMODE_GN ▪ CC_VAL_WFMODE_BG ▪ CC_VAL_WFMODE_N ▪ CC_VAL_WFMODE_G ▪ CC_VAL_WFMODE_B
DA16X_CONF_INT_WIFI_MODE_1	Wi-Fi mode based on IEEE 802.11 standard for Soft AP interface: ▪ CC_VAL_WFMODE_BGN ▪ CC_VAL_WFMODE_GN ▪ CC_VAL_WFMODE_BG ▪ CC_VAL_WFMODE_N ▪ CC_VAL_WFMODE_G ▪ CC_VAL_WFMODE_B
DA16X_CONF_INT_CHANNEL	Soft AP operation channel setting by channel number: 1~11: for US 0: Auto
DA16X_CONF_INT_FREQUENCY	Soft AP operation channel setting by frequency value (MHz).
DA16X_CONF_INT_ROAM	Operating roaming function for STA interface: 0: Stop 1: Run
DA16X_CONF_INT_ROAM_THRESHOLD	Roaming threshold for STA interface (-95 ~ 0 dBm).
DA16X_CONF_INT_BEACON_INTERVAL	IEEE 802.11 beacon interval (msec.).
DA16X_CONF_INT_INACTIVITY	Inactive STA disconnecting time (sec.).
DA16X_CONF_INT_RTS_THRESHOLD	IEEE 802.11 RTS threshold (byte).
DA16X_CONF_INT_WMM	WMM On/Off setting: 0: Off 1: On
DA16X_CONF_INT_WMM_PS	WMM-PS On/Off setting: 0: Off 1: On
DA16X_CONF_INT_DHCP_CLIENT	DHCP client On/Off for STA interface: 0: Off 1: On
DA16X_CONF_INT_DHCP_SERVER	DHCP server On/Off for Soft AP interface: 0: Off 1: On
DA16X_CONF_INT_DHCP_LEASE_TIME	DHCP server lease time (sec.)
DA16X_CONF_INT_HIDDEN_0	Flag to connect AP Hidden SSID: 0: Off 1: On

Name	Description
DA16X_CONF_INT_EAP_PHASE1_0	Phase#1 EAP type for WPA Enterprise: ▪ CC_VAL_EAP_DEFAULT ▪ CC_VAL_EAP_PEAP0 ▪ CC_VAL_EAP_PEAP1 ▪ CC_VAL_EAP_FAST ▪ CC_VAL_EAP_TTLS ▪ CC_VAL_EAP_TLS
DA16X_CONF_INT_EAP_PHASE2_0	Phase#2 EAP type for WPA Enterprise: ▪ CC_VAL_EAP_PHASE2_MIX ▪ CC_VAL_EAP_MSCHAPV2 ▪ CC_VAL_EAP_GTC

9.1.2 String Type Parameters

Table 14. NVRAM string type

Name	Description
DA16X_CONF_STR_SSID_0	AP SSID to connect (~ 32 letters).
DA16X_CONF_STR_SSID_1	Soft AP SSID to operate (~ 32 letters).
DA16X_CONF_STR_WEP_KEY0 DA16X_CONF_STR_WEP_KEY1 DA16X_CONF_STR_WEP_KEY2 DA16X_CONF_STR_WEP_KEY3	WEP keys of the AP to connect (5 or 13 letters with ASCII/10 or 26 letters with hexadecimal).
DA16X_CONF_STR_PSK_0	PSK of the AP to connect (~ 63 letters).
DA16X_CONF_STR_PSK_1	Soft AP PSK to operate (~ 63 letters).
DA16X_CONF_STR_COUNTRY	Country code (2 or 3 letters, for example, KR, US, JP, CH) defined by ISO 3166-1 alpha-2 standard.
DA16X_CONF_STR_DEVICE_NAME	DA16200/DA16600 device name (for WPS or Wi-Fi Direct).
DA16X_CONF_STR_IP_0	STA interface IP address.
DA16X_CONF_STR_NETMASK_0	STA interface netmask.
DA16X_CONF_STR_GATEWAY_0	STA interface gateway address.
DA16X_CONF_STR_IP_1	Soft AP interface IP address.
DA16X_CONF_STR_NETMASK_1	Soft AP interface netmask.
DA16X_CONF_STR_GATEWAY_1	Soft AP interface gateway address.
DA16X_CONF_STR_DNS_0	STA interface DNS address.
DA16X_CONF_STR_DHCP_START_IP DA16X_CONF_STR_DHCP_END_IP	DHCP server IP range assigned.
DA16X_CONF_STR_DHCP_DNS	DHCP server DNS IP address assigned.
DA16X_CONF_STR_EAP_IDENTITY	User-ID for WPA Enterprise (~ 64 letters).
DA16X_CONF_STR_EAP_PASSWORD	Password for WPA Enterprise (~ 64 letters).

9.1.3 Sample Code

When setting multiple names at the same time, use `da16x_set_nvcache_int/str()` and `da16x_nvcache2flash()`. Using `da16x_set_config_str/int()` is good for setting one or two values, but if it needs to set multiple NVRAM parameters (that is, Soft AP/STA setup), then always use cache function `da16x_set_nvcache_int/str` followed by `da16x_nvcache2flash()`, which gives much better performance to the application.

The following example explains how to set STA mode.

```
/* Wi-Fi Configuration */
clear_tmp_nvram_env(); // Clear Cache

// start setting names/values of NVRAM parameters to NVRAM Cache (no delay)
da16x_set_nvcache_int(DA16X_CONF_INT_MODE, 0);
da16x_set_nvcache_str(DA16X_CONF_STR_SSID_0, ssid);
da16x_set_nvcache_int(DA16X_CONF_INT_AUTH_MODE_0, auth_type);
if (auth_type == CC_VAL_AUTH_WEP) {
              da16x_set_nvcache_str(DA16X_CONF_STR_WEP_KEY0, wep_key[0]);
              da16x_set_nvcache_str(DA16X_CONF_STR_WEP_KEY1, wep_key[1]);
              da16x_set_nvcache_str(DA16X_CONF_STR_WEP_KEY2, wep_key[2]);
              da16x_set_nvcache_str(DA16X_CONF_STR_WEP_KEY3, wep_key[3]);
              da16x_set_nvcache_str(DA16X_CONF_INT_WEP_KEY_INDEX, wep_key_index);
} else if (auth_type > CC_VAL_AUTH_WEP) {
       da16x_set_nvcache_str(DA16X_CONF_STR_PSK_0, psk);
       da16x_set_nvcache_int(DA16X_CONF_INT_ENCRYPTION_0, encryption);
}
da16x_set_nvcache_int(DA16X_CONF_INT_WIFI_MODE_0, wifi_mode);

/* IP & DHCP Client Setting */
da16x_set_nvcache_int(DA16X_CONF_INT_DHCP_CLIENT, dhcp_client);
if (!dhcp_client) {
       da16x_set_nvcache_str(DA16X_CONF_STR_IP_0, ip);
       da16x_set_nvcache_str(DA16X_CONF_STR_NETMASK_0, subnet);
       da16x_set_nvcache_str(DA16X_CONF_STR_GATEWAY_0, gateway);
       da16x_set_nvcache_str(DA16X_CONF_STR_DNS_0, dns);
}
da16x_nvcache2flash(); // commit names/values parameters in Cache to flash memory

reboot_func(SYS_REBOOT);
```

The following example explains how to set STA mode for WPA Enterprise. Depending on the wireless environment, the certificate may be required when connecting to WPA Enterprise network. In this case, the Certificate API might be helpful to write to SFlash memory.

```
/* Certificate */
da16x_cert_write(DA16X_CERT_MODULE_WPA_ENTERPRISE, DA16X_CERT_TYPE_CA_CERT, ca_cert,
ca_cert_len); // Write CA Certificate
da16x_cert_write(DA16X_CERT_MODULE_WPA_ENTERPRISE, DA16X_CERT_TYPE_CERT, cert, cert_len); //
Write Certificate
da16x_cert_write(DA16X_CERT_MODULE_WPA_ENTERPRISE, DA16X_CERT_TYPE_PRIVATE_KEY, priv_key,
priv_key_len); // Write Private key
da16x_cert_write(DA16X_CERT_MODULE_WPA_ENTERPRISE, DA16X_CERT_TYPE_DH_PARAMS, dh_param,
dh_param_len); // Write Private key

/* Wi-Fi Configuration */
clear_tmp_nvram_env(); // Clear Cache

// start setting names/values of NVRAM parameters to NVRAM Cache (no delay)
da16x_set_nvcache_str(DA16X_CONF_STR_SSID_0, ssid);          //Set SSID
da16x_set_nvcache_int(DA16X_CONF_INT_AUTH_MODE_0, auth_mode);// Set Auth mode
da16x_set_nvcache_int(DA16X_CONF_INT_ENCRYPTION_0, enc_type);// Set Encryption type
da16x_set_nvcache_int(DA16X_CONF_INT_EAP_PHASE1, eap_phase1);// Set EAP Phase#1 type
da16x_set_nvcache_int(DA16X_CONF_INT_EAP_PHASE2, eap_phase2);// Set EAP Phase#2 type
da16x_set_nvcache_str(DA16X_CONF_STR_EAP_IDENTITY, user_id); // Set User-ID
da16x_set_nvcache_str(DA16X_CONF_STR_EAP_PASSWORD, password);// Set Password

da16x_nvcache2flash(); // commit names/values parameters in Cache to flash memory
```

```
reboot_func(SYS_REBOOT);
```

The following example explains how to set Soft AP mode.

```
/* Soft AP Configuration */
clear_tmp_nvram_env(); // Clear Cache
...
// start setting name/value NVRAM parameters to NVRAM Cache (no delay)
da16x_set_nvcache_int(DA16X_CONF_INT_MODE, 1);
da16x_set_nvcache_str(DA16X_CONF_STR_SSID_1, ssid);
da16x_set_nvcache_int(DA16X_CONF_INT_AUTH_MODE_1, auth_type);
if (auth_type > CC_VAL_AUTH_WEP) {
       da16x_set_nvcache_str (DA16X_CONF_STR_PSK_1, psk);
       da16x_set_nvcache_int(DA16X_CONF_INT_ENCRYPTION_1, encryption);
}
da16x_set_nvcache_int(DA16X_CONF_INT_CHANNEL, channel);
da16x_set_nvcache_int(DA16X_CONF_STR_COUNTRY, country_code);
da16x_set_nvcache_int(DA16X_CONF_INT_WIFI_MODE_1, wifi_mode);
da16x_set_nvcache_int(DA16X_CONF_INT_WMM, wmm);
da16x_set_nvcache_int(DA16X_CONF_INT_WMM_PS, wmm_ps);

/* IP Setting */
da16x_set_nvcache_str(DA16X_CONF_STR_IP_1, ip);
da16x_set_nvcache_str(DA16X_CONF_STR_NETMASK_1, subnet);
da16x_set_nvcache_str(DA16X_CONF_STR_GATEWAY_1, gateway);

/* DHCP Server Setting */
if (dhcp_server) {
       da16x_set_nvcache_str(DA16X_CONF_STR_DHCP_START_IP, start_ip);
       da16x_set_nvcache_str(DA16X_CONF_STR_DHCP_END_IP, end_ip);
      da16x_set_nvcache_str(DA16X_CONF_STR_DHCP_DNS, dhcp_dns);
       da16x_set_nvcache_str(DA16X_CONF_INT_DHCP_LEASE_TIME, dhcp_lease_time);
}
da16x_set_nvcache_int(DA16X_CONF_INT_DHCP_SERVER, dhcp_server);

da16x_nvcache2flash(); // commit names/values parameters in Cache to flash memory

reboot_func(SYS_REBOOT);
```

9.2 Soft AP Configuration by Factory Reset

Many IoT devices start as Soft AP device to operate AP provisioning. The DA16200/DA16600 has a Factory Reset function to start with Soft AP mode after pressing the Factory Reset button on the evaluation board. The details of Factory Reset button can be found in DA16200 and DA16600 EVBs (look for S2- Factory Reset Button) in Ref. [3], and it is connected to GPIO 7 on the DA16200/DA16600 EVB.

The DA16200/DA16600 SDK offers a simple method for users to configure the Soft AP interface with their own values. This section describes how to configure the default values in the DA16200/DA16600 SDK.

9.2.1 Configure Data Structure

The DA16200/DA16600 SDK has the structure to configure Soft AP interface. The details can be found in the following example:

```
[ ~/FreeRTOS_SDK/core/system/include/common/da16x_network_common.h ]

/* For Customer's Soft AP configuration */
#define       MAX_SSID_LEN         32
#define       MAX_PASSKEY_LEN      64
#define       MAX_IP_ADDR_LEN      16
```

```
#define       AP_OPEN_MODE        0
#define       AP_SECURITY_MODE    1

#define       IPADDR_DEFAULT      0
#define       IPADDR_CUSTOMER     1

#define       DHCPD_DEFAULT       0
#define       DHCPD_CUSTOMER      1

typedef struct _Soft AP_config {
     int      customer_cfg_flag;   // MODE_ENABLE, MODE_DISABLE

     char     ssid_name[MAX_SSID_LEN+1];
     char     psk[MAX_PASSKEY_LEN+1];
     char     auth_type;           // AP_OPEN_MODE, AP_SECURITY_MODE
     char     country_code[4];

     int      customer_ip_address; // IPADDR_DEFAULT, IPADDR_CUSTOMER
     char     ip_addr[MAX_IP_ADDR_LEN];
     char     subnet_mask[MAX_IP_ADDR_LEN];
     char     default_gw[MAX_IP_ADDR_LEN];
     char     dns_ip_addr[MAX_IP_ADDR_LEN];

     int      customer_dhcpd_flag; // DHCPD_DEFAULT, DHCPD_CUSTOMER
     //int    dhcpd_ip_cnt;
     int      dhcpd_lease_time;
     char     dhcpd_start_ip[MAX_IP_ADDR_LEN];
     char     dhcpd_end_ip[MAX_IP_ADDR_LEN];
     char     dhcpd_dns_ip_addr[MAX_IP_ADDR_LEN];
} Soft AP_config_t;
```

- int customer_cfg_flag: Flag for Soft AP configuration
  - MODE_DISABLE (0) : Do not use Soft AP configuration
  - MODE_ENABLE (1) : Use Soft AP configuration
- char ssid_name[MAX_SSID_LEN+1]: SSID of Soft AP. Max length is 32 bytes
- char psk[MAX_PASSKEY_LEN]: Pairwise key. Max length is 64 bytes
- char auth_type: Authentication type
  - OPEN_MODE (0)
  - AP_SECURITY_MODE (1)
- char country_code [4]: Country code

  See Appendix D.
- int customer_ip_address: IP address type
  - IPADDR_DEFAULT (0) : IP class is 10.0.0.1
  - IPADDR_CUSTOMER (1) : User defined IP address

    The following parameters should be defined:

    char ip_addr[MAX_IP_ADDR_LEN]

    char subnet_mask[MAX_IP_ADDR_LEN]

    char default_gw[MAX_IP_ADDR_LEN]

    char dns_ip_addr[MAX_IP_ADDR_LEN]
- int customer_dhcpd_flag: DHCP server IP address range
  - DHCPD_DEFAULT (0) : 10.0.0.2 ~ 10.0.0.11 (10 clients)
  - DHCPD_CUSTOMER (1) : User defined range

    Need to define the following parameters:

```
int     dhcpd_lease_time
char    dhcpd_start_ip[MAX_IP_ADDR_LEN]
char    dhcpd_end_ip[MAX_IP_ADDR_LEN]
char    dhcpd_dns_ip_addr[MAX_IP_ADDR_LEN]
```

9.2.2 Configure Soft AP Interface

The DA16200/DA16600 SDK has the function of configuring the Soft AP interface. This function is invoked when a factory reset is done. Users can write their own values, and the details can be found in the following example:

```
[ ~/FreeRTOS_SDK/customer/user_main/src/system_start.c ]
void set_customer_Soft AP_config(void)
{
#ifdef __SUPPORT_FACTORY_RST_APMODE__
      /* Set to user costomer's configuration */
      ap_config_param->customer_cfg_flag = MODE_DISABLE;
// MODE_ENABLE, MODE_DISABLE
      /*
       * Wi-Fi configuration
       */
/* SSID prefix */
      sprintf(ap_config_param->ssid_name, "%s", "DA16200");

      /* Default open mode: AP_OPEN_MODE, AP_SECURITY_MODE */
      ap_config_param->auth_type = AP_OPEN_MODE;
      if (ap_config_param->auth_type == AP_SECURITY_MODE);
        sprintf(ap_config_param->psk, "%s", "12345678");

      /* Country Code: Default country US */
      sprintf(ap_config_param->country_code, "%s", DFLT_AP_COUNTRY_CODE);

      /*
       * Network IP address configuration
       */
      ap_config_param->customer_ip_address = IPADDR_DEFAULT;
      if (ap_config_param->customer_ip_address == IPADDR_CUSTOMER) {
        sprintf(ap_config_param->ip_addr, "%s", "192.168.1.1");
        sprintf(ap_config_param->subnet_mask,    "%s", "255.255.255.0");
        sprintf(ap_config_param->default_gw,     "%s", "192.168.1.1");
        sprintf(ap_config_param->dns_ip_addr,    "%s", "8.8.8.8");
      }

      /*
       * DHCP Server configuration
       */
      ap_config_param->customer_dhcpd_flag = DHCPD_DEFAULT;
      if (ap_config_param->customer_dhcpd_flag == DHCPD_CUSTOMER) {
        ap_config_param->dhcpd_lease_time = 3600;

        sprintf(ap_config_param->dhcpd_start_ip, "%s", "192.168.1.101");
        sprintf(ap_config_param->dhcpd_end_ip, "%s", "192.168.1.108");
        sprintf(ap_config_param->dhcpd_dns_ip_addr, "%s", "8.8.8.8");
      }
#endif /* __SUPPORT_FACTORY_RST_APMODE__ */
}
```

9.3 Soft AP Provisioning Protocol

The DA16200/DA16600 supports the Soft AP mode for a Wi-Fi interface setup. The provisioning thread automatically runs when the DA16200/DA16600 starts in the Soft AP mode. See Ref. [4] for further details.

10. Wi-Fi Functionality

This section describes the Wi-Fi functionality that DA16200/DA16600 SDK provides.

10.1 Simple Roaming

Wi-Fi roaming allows the station (STA) automatically to change the connection to another AP within the coverage areas of APs or routers belonging to the same extended service set (ESS) which has the same SSID and credentials. DA16200/DA16600 SDK supports simplified and modified version of Wi-Fi roaming named Simple Roaming. If RSSI is lower than a predefined threshold, DA16200/DA16600 automatically scans APs, selects another AP with best RSSI, and connects the AP.

NOTE
The simple roaming is switched off automatically when DPM mode is enabled. Default threshold is -65 dBm and valid range is 0 ~ -95 dBm.

10.1.1 Using Simple Roaming

The feature is disabled in SDK by default. To configure threshold and enable the feature, use `da16x_set_config_int()` APIs as shown in the following sample codes. In addition, DA16200/600 SDK offers AT commands to enable and use the feature. See Ref. [6] for more about AT+WFROAP and AT+WFROTH commands.

```
// To configure roaming threshold
int threshold = -55;

if (da16x_set_config_int(DA16X_CONF_INT_ROAM_THRESHOLD, threshold)) {
    PRINTF("Failed to configure roaming threshold\n");
}

// To run the simple roaming
if (da16x_set_config_int(DA16X_CONF_INT_ROAM, 1)) {
    PRINTF("Failed to run simple roaming function\n");
}

// To stop the simple roaming
if (da16x_set_config_int(DA16X_CONF_INT_ROAM, 0)) {
    PRINTF("Failed to stop simple roaming function\n");
}
```

In addition, WPA CLI commands are available for configuring roaming threshold and running/stopping simple roaming.

```
// To configure roaming threshold
[/DA16200] # net.cli roam_threshold -35

// To run simple roaming
[/DA16200] # net.cli roam run

// To store configuration to NVRAM
[/DA16200] # net.cli save_config

// To stop simple roaming
/DA16200] # net.cli roam stop
```

When simple roaming operates, debug messages are displayed.

```
>>> [Roaming] Start - Current signal level is lower then the threshold.

>>> [Roaming] New  - BSSID[88:36:6c:20:a0:76], Level[-26]
>>> Roam Scan[0/39] BSS 88:36:6c:20:a0:76 ssid='RENESAS_TESTAP' (-26)
```

```
>>> [Roaming]   2  - BSSID[88:36:6c:20:a0:76], Level[-26]
>>> Roam Scan[2/39] BSS 88:36:6c:20:a0:76 ssid='RENESAS_TESTAP' (-26)
>>> [Roaming]   1  - BSSID[88:36:6c:20:a0:76], Level[-26]
>>> Roam Scan[0/39] BSS 88:36:6c:20:a0:76 ssid='RENESAS_TESTAP' (-26)

>>> Network Interface (wlan0) : DOWN
-- DHCP Client WLAN0: STOP(0)

>>> Network Interface (wlan0) : UP
>>> Associated with 88:36:6c:20:a0:76

Connection COMPLETE to 88:36:6c:20:a0:76

-- DHCP Client WLAN0: SEL(6)
-- DHCP Client WLAN0: REQ(1)

### User Call-back : Success to connect Wi-Fi ...
-- DHCP Client WLAN0: CHK(8)
-- DHCP Client WLAN0: BOUND(10)
         Assigned addr   : 192.168.2.6
               netmask   : 255.255.255.0
               gateway   : 192.168.2.1
               DNS addr  : 168.126.63.1

         DHCP Server IP  : 192.168.2.1
         Lease Time      : 02h 00m 00s
         Renewal Time    : 01h 00m 00s
```

10.2 Scanning and Example

Scanning is the process of finding the desired AP through the station that the user wants to connect to. Two types of scanning are available: active and passive scanning.

10.2.1 Active Scanning

In active scanning, the station device sends a frame which is called probe request frame to AP. Probe request can be unicast or broadcast. In response to the probe request, the AP sends a probe response, which is used by a station to take connection related decisions. For DA16200 DA16600 SDK, it broadcasts probe request frames on each channel and waits for probe response frames for a certain amount of time. As scanning results, BSSID, frequency, RSSI, security, and SSID information are included.

For active scanning in DA16200/DA16600 SDK, the `get_scan_result()` API is provided. Scanned APs are listed and sorted by signal strength. Alternatively, the `da16x_cli_reply()` API can be used directly as the `get_scan_result()` API is implemented in the API.

Here are overall descriptions of active scanning on DA16200/DA16600 SDK.

- The DA16200/DA16600 scans each channel based on a country code and `cc_power_level[]` and `cc_power_level_dsss[]` tables.
- Channel 14 has additional restrictions or cannot be used in all regulatory areas.
- If the transmission power grade of a channel is set to `0xF` in the tables, scanning this channel should be skipped.
- The DA16200/DA16600 SDK allows to scan full channels, not single channels.
- Active scan time:
  - Time to scan for 1 channel: about 30 ms.
    - Transmitting probe request frame and receiving probe response frame.
  - Time to switch channel: about 29 ms.
    - It may vary depending on interference and circumstances.
  - Total active scan time for full channels:

  - Time to scan for 1 channel x Numbers of channels + Channel switch time x (number of channels – 1).

Here is an example of using the `da16x_cli_reply()` API for active scan.

```
#define    SCAN_RSP_BUF_SIZE    (4 * 1024 )

char scan_result[SCAN_RSP_BUF_SIZE] = { 0, };

memset(scan_result, 0, SCAN_RSP_BUF_SIZE);
da16x_cli_reply("scan", NULL, scan_result);
// Scan failed
if (strlen(scan_result) < 30) {
      PRINTF("Scan: %s\n", scan_result);
}
```

Also, the AT command (AT+WFSCAN) is available for active scanning (see Ref. [7] for details).

10.2.2 Passive-Scanning

In passive scanning, the station device waits for a special frame, beacon frames that AP broadcasts periodically, and the beacon frames are buffered and used to decode and extract information about BSSs. As scanning results, BSSID, frequency, RSSI, security, and SSID information are included.

The DA16200 DA16600 SDK only supports passive scanning using AT commands because scanning results are transmitted over UART interface. See Ref. [7] about AT commands.

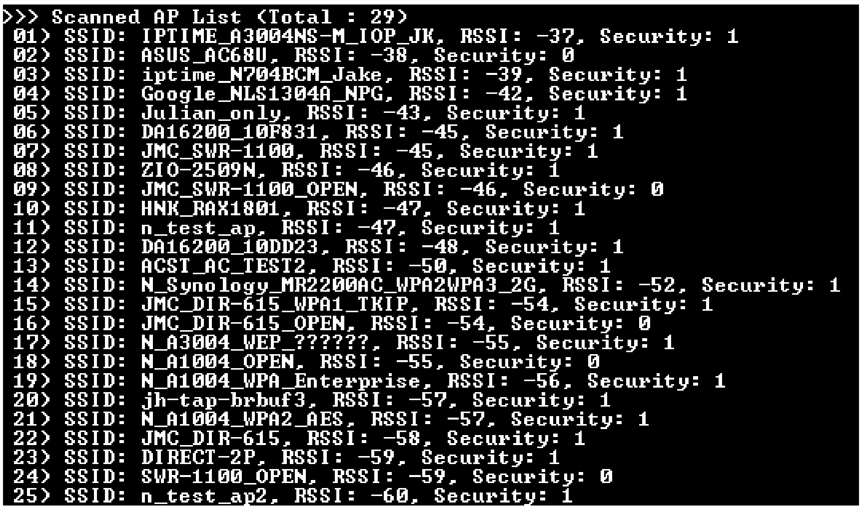
10.2.3 Get Scan Result Example

An example of active scan is available in SDK. To run the example, complete the following steps.

In the e^2 studio, import a project for the Scan result example application.

`~/SDK/apps/common/examples/ETC/Get_Scan_Result/projects/da16200`

1. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
2. After the boot is complete, the `get_scan_result` sample starts automatically.



```
>>> Scanned AP List (Total : 29)
 01) SSID: IPTIME_A3004NS-M_IOP_JK, RSSI: -37, Security: 1
 02) SSID: ASUS_AC68U, RSSI: -38, Security: 0
 03) SSID: iptime_N704BCM_Jake, RSSI: -39, Security: 1
 04) SSID: Google_NLS1304A_NPG, RSSI: -42, Security: 1
 05) SSID: Julian_only, RSSI: -43, Security: 1
 06) SSID: DA16200_10F831, RSSI: -45, Security: 1
 07) SSID: JMC_SWR-1100, RSSI: -45, Security: 1
 08) SSID: ZIO-2509N, RSSI: -46, Security: 1
 09) SSID: JMC_SWR-1100_OPEN, RSSI: -46, Security: 0
 10) SSID: HNK_RAX1801, RSSI: -47, Security: 1
 11) SSID: n_test_ap, RSSI: -47, Security: 1
 12) SSID: DA16200_10DD23, RSSI: -48, Security: 1
 13) SSID: ACST_AC_TEST2, RSSI: -50, Security: 1
 14) SSID: N_Synology_MR2200AC_WPA2WPA3_2G, RSSI: -52, Security: 1
 15) SSID: JMC_DIR-615_WPA1_TKIP, RSSI: -54, Security: 1
 16) SSID: JMC_DIR-615_OPEN, RSSI: -54, Security: 0
 17) SSID: N_A3004_WEP_??????, RSSI: -55, Security: 1
 18) SSID: N_A1004_OPEN, RSSI: -55, Security: 0
 19) SSID: N_A1004_WPA_Enterprise, RSSI: -56, Security: 1
 20) SSID: jh-tap-brbuf3, RSSI: -57, Security: 1
 21) SSID: N_A1004_WPA2_AES, RSSI: -57, Security: 1
 22) SSID: JMC_DIR-615, RSSI: -58, Security: 1
 23) SSID: DIRECT-2P, RSSI: -59, Security: 1
 24) SSID: SWR-1100_OPEN, RSSI: -59, Security: 0
 25) SSID: n_test_ap2, RSSI: -60, Security: 1
```

Figure 13. Get_Scan_Result AP list

This example shows how to use the void `get_scan_result(void *user_buf_ptr)` API and to get the Scan result on STA mode and Soft AP mode.

The `get_scan_result_sample` function is executed after the basic FreeRTOS initialization is complete. This example simply calls the user API void `get_scan_result()`.

```
void get_scan_result_sample(void * param)
{
    char        *user_buf = NULL;
    scan_result_t    *scan_result;
```

```
    int     i;

    /* Allocate buffer to get scan result */
    user_buf = pvPortMalloc(SCAN_RSP_BUF_SIZE);

    /* Get scan result */
    get_scan_result((void *) user_buf);

    … …
}
```

After running the `get_scan_result()` API, the user can use the received data. This example code shows how to display the scan list in the console.

```
/* Display result on console */
scan_result = (scan_result_t *)user_buf;

PRINTF("\n>>> Scanned AP List (Total : %d) \n", scan_result->ssid_cnt);

for (i = 0; i < scan_result->ssid_cnt; i++) {
    PRINTF(" %02d) SSID: %s, RSSI: %d, Security: %d\n",
                 i + 1,
                 scan_result->scanned_ap_info[i].ssid,
                 scan_result->scanned_ap_info[i].rssi,
                 scan_result->scanned_ap_info[i].auth_mode) ;
}

/* Buffer free */
vPortFree(user_buf);
```

The Scan results are stored in the following data structure format:

```
typedef struct scanned_ap_info {
    int       auth_mode;
    int       rssi;
    char      ssid[128];
} scanned_ap_info_t;

typedef struct scan_result_to_app {
    int       ssid_cnt;
    scanned_ap_info_t      scanned_ap_info[MAX_SCAN_AP_CNT];
} scan_result_t;
```

11. Network Examples: Socket Communication

This section describes how to develop Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) socket applications using the lwIP (Lightweight IP) APIs in the DA16200/DA1600 SDK. As a companion document, see Ref. [1] for details on all functions. To understand and implement applications using the DPM API, both non-DPM and DPM examples are provided. Before testing these examples, a test environment as shown in Figure 14 is required.

11.1 Test Environment

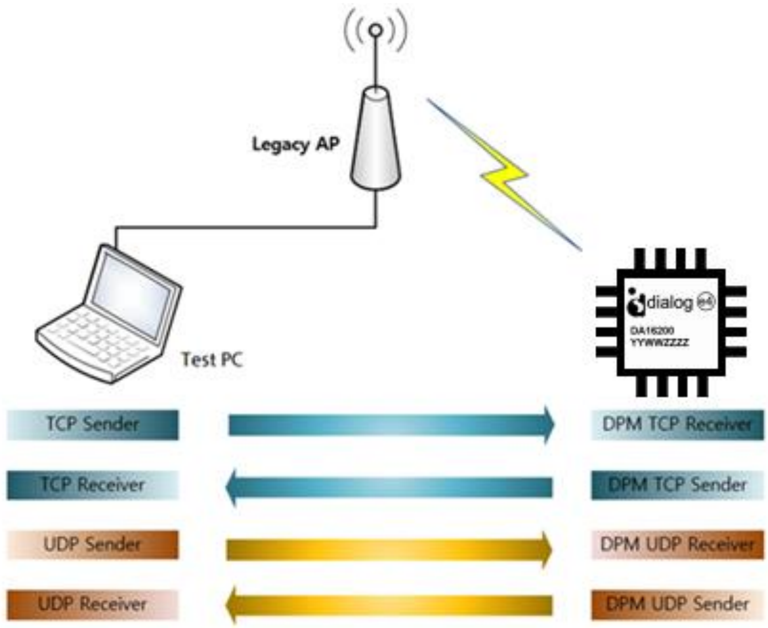


Figure 14. Overall test setup

11.1.1 DA16200

The files of example sources are included in the DA16200 SDK. The examples in this section require the DA16200 to be configured as a Wi-Fi station (STA Mode). See the Station Mode Setup section of Ref. [3] on how to set up Wi-Fi station mode. Also, after completing the STA mode setup, copy the IP address of the DA16200 EVB for later use. The IP address is printed after connecting to an Access Point (AP), and then TCP/UDP example application runs. See Figure 15.

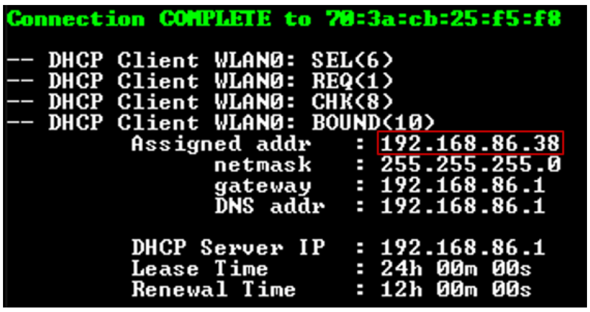


Figure 15. DA16200 EVB – AP connection complete

11.1.2 Peer Application

The examples in this section require a peer device (workstation or laptop) connected to the same AP running a TCP/UDP test application such as IO Ninja.

NOTE
For the Windows OS system, the user needs to install a proper application such as Packet Sender, Hercules, and IO Ninja. For a Linux system, proper test utilities or a test sample application are needed.

11.1.2.1 Example of Peer Application

This section describes how to run the peer application on the Windows operating system.

1. Start the IO Ninja utility on the test laptop or desktop computer.
   If it is not installed, download it from http://ioninja.com.
2. Select **File** > **New Session** for the test.

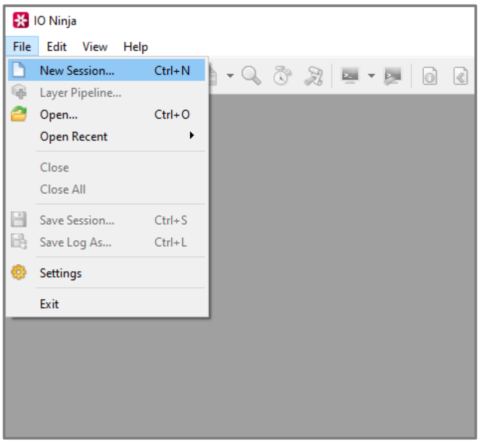


Figure 16. Start IO Ninja utility

3. To test the TCP Client, start the TCP Server.

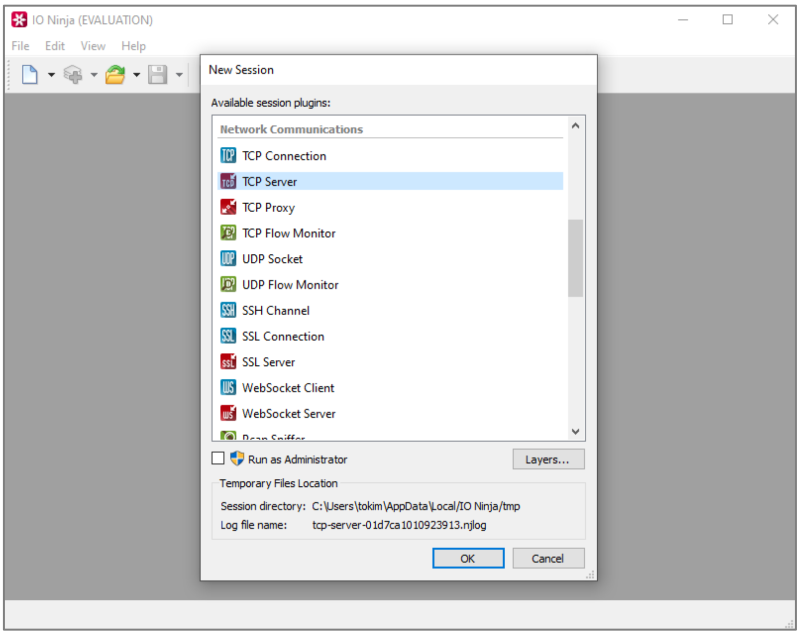


Figure 17. Select TCP server session

4. If **TCP Listener Socket** is selected, IO Ninja utility shows the TCP server test window.

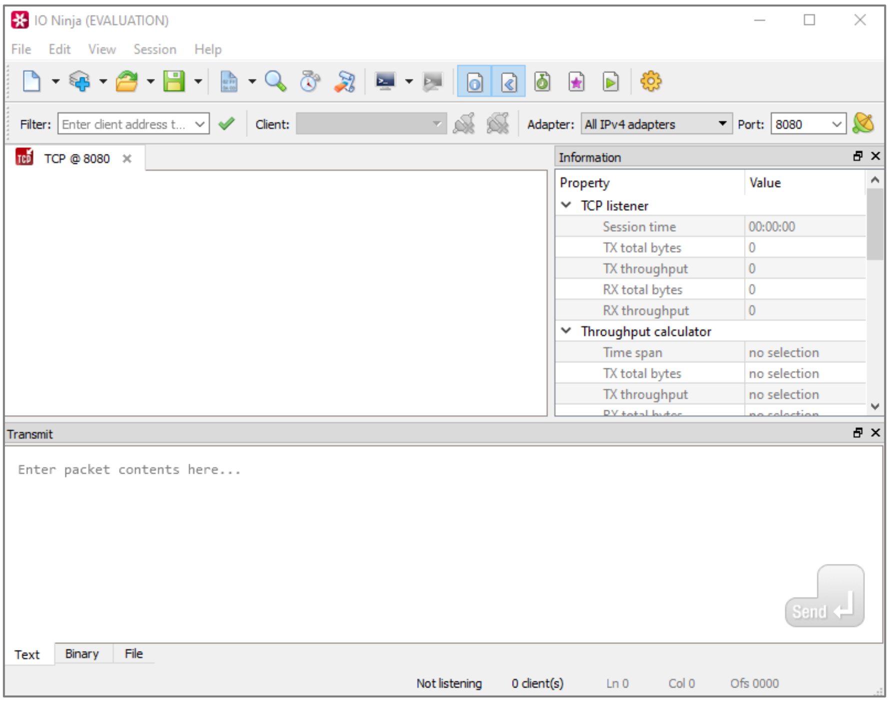


Figure 18. TCP server session windows

5. Start the TCP Server session (for example, TCP Client test).

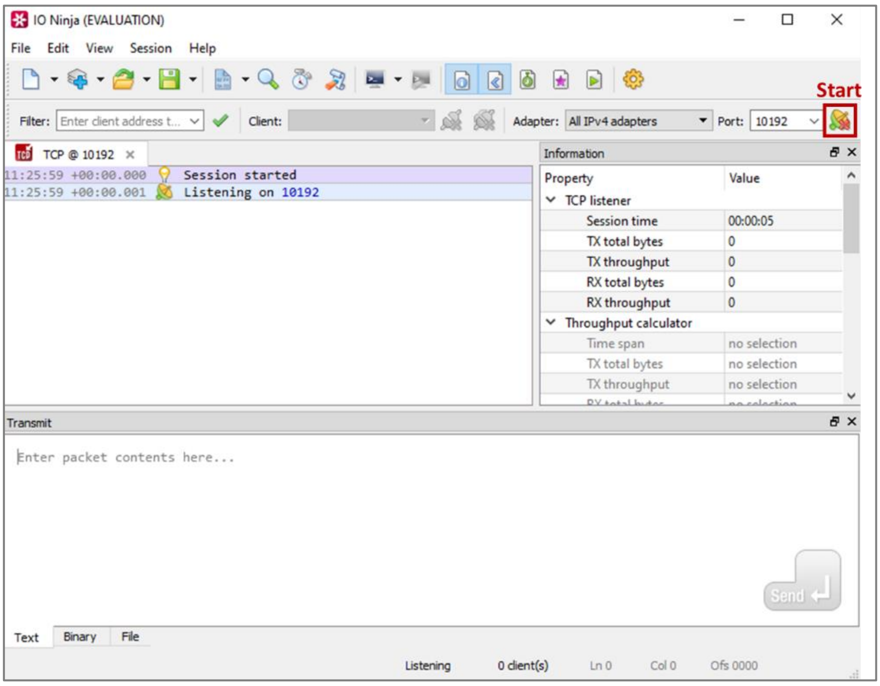


Figure 19. Start TCP server session

6. Connect to the TCP Client.

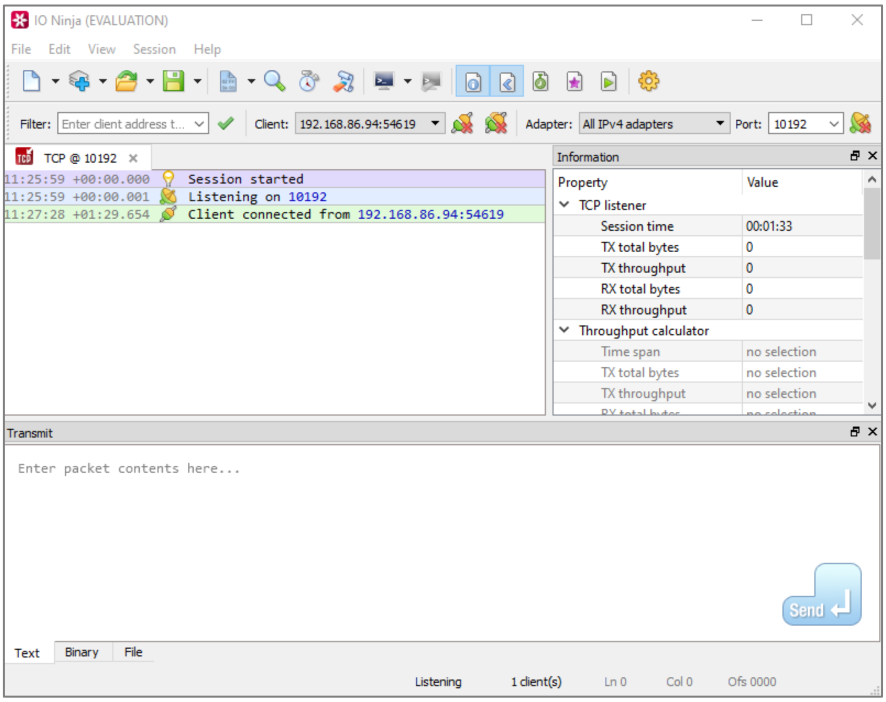


Figure 20. TCP connection with TCP client

7. Run data communication.

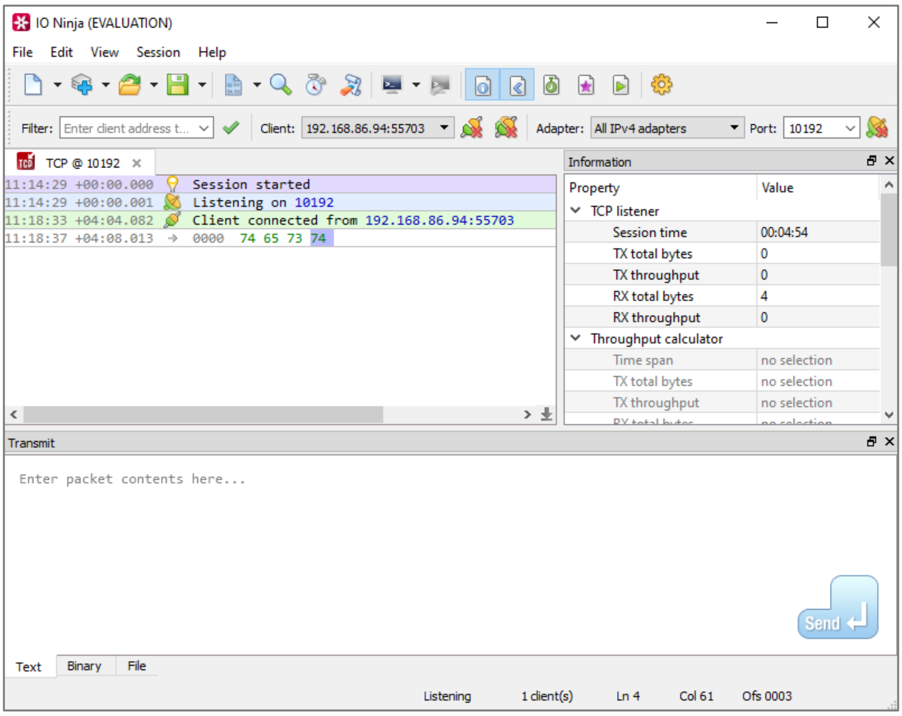


Figure 21. TCP data communication with TCP client

11.2 TCP Client

This section describes how the TCP client sample application is built and operated. The TCP client sample is an example of the simplest TCP echo client application. TCP is one of the main protocols of the Internet protocol suite. TCP provides a reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications that run on hosts that communicate through an IP network. The DA16200 SDK provides a lwIP's TCP protocol. lwIP is an open-source TCP/IP stack designed for embedded systems.

11.2.1 How to Run

1. Run a socket application on the peer computer (see Section 11.1.2) and open a TCP server socket with port number 10192 (default TCP Client test port).
2. In the e[2] studio, import a project for TCP Client sample application.
   ~/SDK/apps/common/examples/Network/TCP_Client/projects/da16200
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. Use the console command to set up the Wi-Fi station interface.
5. To set the IP address and port for the peer application (TCP Server) in the TCP Client Sample, edit the source code:

```
~/SDK/apps/common/examples/Network/TCP_Client/src/tcp_client_sample.c

#define TCP_CLIENT_SAMPLE_DEF_SERVER_IP_ADDR            "192.168.0.11"
#define TCP_CLIENT_SAMPLE_DEF_SERVER_PORT                   TCP_CLI_TEST_PORT
```

The example connects to the peer application (TCP Server) after a connection is made to the Wi-Fi AP.

11.2.2 How It Works

The DA16200 TCP Client sample application is a simple echo message. When the TCP server sends a message, the DA16200 TCP client echoes that message to the TCP server.

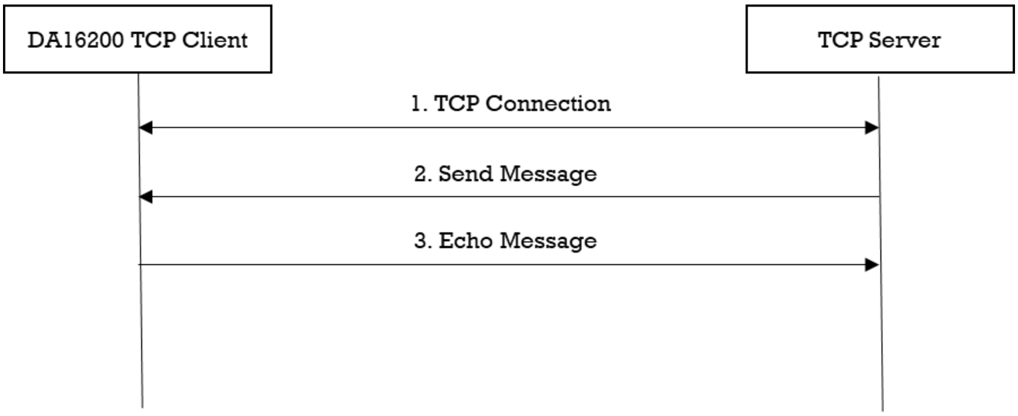


Figure 22. Workflow of TCP client

11.2.3 Sample Code

The DA16200 SDK provides the lwIP's TCP protocol. This sample application describes how a TCP socket is created, deleted, and configured.

11.2.3.1 Registration

The client side of the TCP connection initiates a connection request to a TCP server. The client TCP socket should be created with the socket() service and bound to a port via the bind() service. After the client socket is bound, the connect() service is used to establish a connection with a TCP server.

```
void tcp_client_sample(void *param)
{
      int ret = 0;
      int socket_fd;
      struct sockaddr_in local_addr;
      struct sockaddr_in srv_addr;

      memset(&local_addr, 0x00, sizeof(struct sockaddr_in));
      memset(&srv_addr, 0x00, sizeof(struct sockaddr_in));
      // Create TCP socket
      socket_fd = socket(PF_INET, SOCK_STREAM, 0);

      local_addr.sin_family = AF_INET;
```

```
      local_addr.sin_addr.s_addr = htonl(INADDR_ANY);
      local_addr.sin_port = htons(TCP_CLIENT_SAMPLE_DEF_PORT);

      // Bind TCP socket
      ret = bind(socket_fd, (struct sockaddr *)&local_addr,
                 sizeof(struct sockaddr_in));

      srv_addr.sin_family = AF_INET;
      srv_addr.sin_addr.s_addr = inet_addr(TCP_CLIENT_SAMPLE_DEF_SERVER_IP_ADDR);
      srv_addr.sin_port = htons(TCP_CLIENT_SAMPLE_DEF_SERVER_PORT);

      // Connect TCP socket
      ret = connect(socket_fd, (struct sockaddr_in *)&srv_addr,
                    sizeof(struct sockaddr_in));
         ....
}
```

11.2.3.2 Data Transmission

TCP data is received when function `recv()` is called. TCP incoming packet handles various connections and disconnections and is responsible for acknowledging transmissions.

TCP data is sent when function send() is called. This service first builds a TCP header in the front part of the packet (including the checksum calculation). If the receiver's window size is larger than the data in this packet, the packet is sent to the internet with the internal IP send routine. Otherwise, the caller may be suspended and wait for the receiver's window size to increase enough for this packet to be sent. At any given time, only one sender may suspend while trying to send TCP data.

```
void tcp_client_sample()
{
    ...
    while (1) {
        memset(data_buffer, 0x00, sizeof(data_buffer));

        PRINTF("< Read from server: ");
        len = recv(socket_fd, data_buffer, sizeof(data_buffer), 0);
        data_buffer[len] = '\0';
        PRINTF("%d bytes read\r\n", len);

        PRINTF("> Write to server: ");
        len = send(socket_fd, data_buffer, len, 0);
        PRINTF("%d bytes written\r\n", len
      }
    ...
}
```

11.2.3.3 Disconnection

The connection is closed when function close() is called. This function handles sockets to be closed and deleted internally. The socket must be in a CLOSED state or in the process of disconnecting before the port is released. Otherwise, an error is returned. Finally, if the application no longer needs the client socket, the vTaskDelete() function is called to delete the socket.

```
void tcp_client_sample()
{
    ...
    close(socket_fd);

end_of_task:

      PRINTF("[%s] End of TCP Client sample\r\n", __func__);
      vTaskDelete(NULL);
```

```
    return ;
}
```

11.3 TCP Client in DPM

This section describes how the TCP client in the DPM sample application is built and works. The TCP client in the Dynamic Power Management (DPM) sample application is an example of the simplest TCP echo client application in DPM mode. The DA16200 SDK can work in DPM mode. The user application requires an additional operation to work in DPM mode. The DA16200 SDK provides the DPM manager for the user network application. The DPM manager supports users to develop and manage a network application in Non-DPM and DPM modes.

11.3.1 How to Run

1. Run a socket application on the peer laptop (see Section 11.1.2) and open a TCP server socket with port number 10192.
2. In the e[2] studio, import a project for the TCP client in the DPM sample application.
   `~/SDK/apps/common/examples/Network/TCP_Client_DPM/projects/da16200`
   - Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. Use the console command to set up the Wi-Fi station interface.
4. To set the IP address and the port for the peer application (TCP Server) in the TCP Client Sample, do one of the following:
   - Edit the source code:

```
~/SDK/apps/common/examples/Network/TCP_Client_DPM/src/tcp_client_dpm_sample.c

#define define TCP_CLIENT_DPM_SAMPLE_DEF_SERVER_IP      "192.168.0.11"
#define TCP_CLIENT_DPM_SAMPLE_DEF_SERVER_PORT        TCP_CLI_TEST_PORT
```

   - Use the DA16200 console to save the values in NVRAM:

```
[/DA16200] # nvram.setenv TCPC_SERVER_IP 192.168.0.11
[/DA16200] # nvram.setenv TCPC_SERVER_PORT 10192
[/DA16200] # reboot
```

After a connection is made to a Wi-Fi AP, the example of connecting to the peer application (TCP Server).

11.3.2 How It Works

The DA16200 TCP Client in the DPM sample application is a simple echo message. When the TCP server sends a message, then the DA16200 TCP client echoes that message to the TCP server.

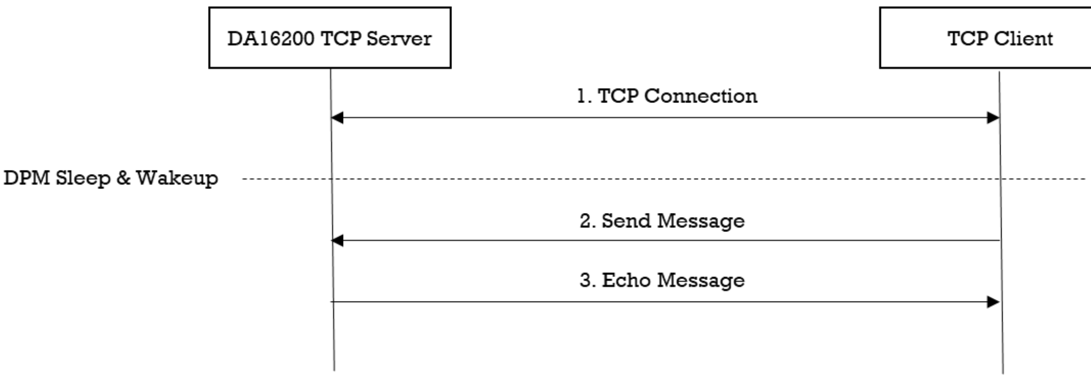


Figure 23. Workflow of TCP client in DPM

11.3.3 Sample Code

11.3.3.1 Registration

The TCP client in the DPM sample application works in DPM mode. The basic code is similar to the TCP client sample application. There are two differences from the TCP client sample application:

- An initial callback function is added, named tc9p_client_dpm_sample_wakeup_callback() in the code. The callback is called when the DPM state changes from sleep to wake-up
- Additional user configuration can be stored in RTM

In this sample, the TCP server information is stored.

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

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

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

        //Set External wake-up 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].sessionType = REG_TYPE_TCP_CLIENT;

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

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

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

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

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

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

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

        //Set auto reconnection flag
        user_config->sessionConfig[session_idx].sessionAutoReconn = TRUE;

        //Set user configuration
```

```
        user_config->ptrDataFromRetentionMemory = (UCHAR *)&srv_info;
        user_config->sizeOfRetentionMemory =
        sizeof(tcp_client_dpm_sample_svr_info_t);

        return ;
}
```

11.3.3.2 Data Transmission

The callback function is called when a TCP packet is received from a TCP server. In this sample, the received data is printed out and an echo message is sent to the TCP server.

```
void tcp_client_dpm_sample_recv_callback(void *sock, UCHAR *rx_buf, UINT rx_len,
                                  ULONG rx_ip, ULONG rx_port)
{
        unsigned char status = pdPASS;

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

        //Echo message
        status = dpm_mng_send_to_session(SESSION1, rx_ip, rx_port,
                                        (char *)rx_buf, rx_len);
        else
        {
             //Display sent packet
             PRINTF(" <== Sent Packet(%ld) \n", rx_len);
        }

        dpm_mng_job_done(); //Done opertaion
}
```

11.4 TCP Server

This section describes how the TCP server sample application is built and works. The TCP server sample application is an example of the simplest TCP echo server application. TCP is one of the main protocols of the Internet protocol suite. It provides a reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts that communicate through an IP network. The DA16200 SDK provides a lwIP's TCP protocol. lwIP is an open-source TCP/IP stack designed for embedded systems.

11.4.1 How to Run

1. In the e^2 studio, import a project for the TCP Server sample application.
   `~/SDK/apps/common/examples/Network/TCP_Server/projects/da16200`
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. To set the port of the TCP Server Sample, do one of the following:
   - Edit the source code:

```
~/SDK/apps/common/examples/Network/TCP_Server/src/tcp_server_sample.c

#define    TCP_SERVER_SAMPLE_DEF_SERVER_PORT      TCP_SVR_TEST_PORT
```

   - Use the DA16200 console to save the values in NVRAM:

```
[/DA16200] # nvram.setenv TCP_SVR_PORT 10190
[/DA16200] # reboot
```

4. Set up the Wi-Fi station interface using console commands.
5. When connected to the AP, the sample application creates a TCP server socket with port number 10190 and waits for a client connection.

6. Run a socket application on the peer computer (See Section 11.1.2).
7. Open a TCP client socket.

11.4.2 How It Works

The DA16200 TCP server sample application is a simple echo server. When a TCP client sends a message, the DA16200 TCP server echoes that message to the TCP client.

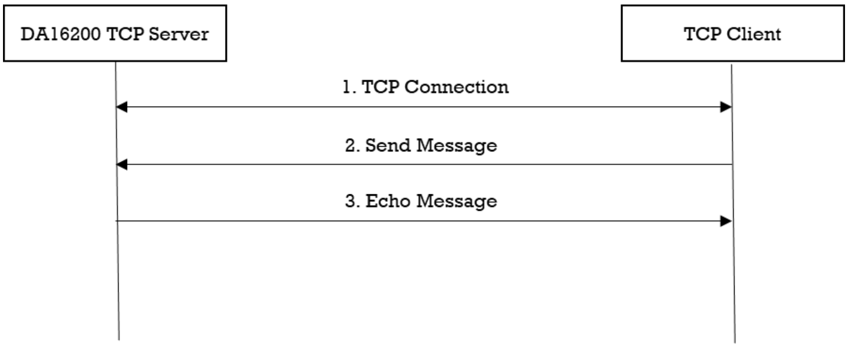


Figure 24. Workflow of TCP server

11.4.3 Sample Code

The DA16200 SDK provides the lwIP's TCP protocol. This sample application describes how a TCP socket is created, deleted, and configured.

11.4.3.1 Connection

The server waits for a client connection request. Next, the application must create a TCP socket with the socket() service. The server socket must also be set up to listen to connection requests with the listen() service. This service puts the server socket in the LISTEN state and binds the specified server port to the server socket. If the socket connection has already been established, the function simply returns a successful status.

```
void tcp_server_sample()
{
    int ret = 0;
    int listen_sock = -1;
    int client_sock = -1;

    struct sockaddr_in server_addr;
    struct sockaddr_in client_addr;

    memset(&server_addr, 0x00, sizeof(struct sockaddr_in));
    memset(&client_addr, 0x00, sizeof(struct sockaddr_in));

    // Create TCP socket
    listen_sock = socket(PF_INET, SOCK_STREAM, 0);
    if (listen_sock < 0) {
        PRINTF("[%s] Failed to create listen socket\r\n", __func__);
        goto end_of_task;
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    server_addr.sin_port = htons(TCP_SERVER_SAMPLE_DEF_PORT);

    // Bind TCP socket
    ret = bind(listen_sock, (struct sockaddr *)&server_addr,
               sizeof(struct sockaddr_in));

    // Listen TCP socket
    ret = listen(listen_sock, TCP_SERVER_SAMPLE_BACKLOG);
```

```
    while (1) {
        client_sock = -1;
        memset(&client_addr, 0x00, sizeof(struct sockaddr_in));
        client_addrlen = sizeof(struct sockaddr_in);

        // Accept TCP socket
        client_sock = accept(listen_sock, (struct sockaddr *)&client_addr,
                                          (socklen_t *)&client_addrlen);
        While (1){
        ...
        }
    }
}
```

11.4.3.2 Data Transmission

TCP data is received when function `recv()` is called. The TCP receive packet process is responsible for handling the various connections and disconnections as well as transmission acknowledgment process.

TCP data is sent when function send() is called. This service first builds a TCP header in the front part of the packet (including the checksum calculation). If the receiver's window size is larger than the data in this packet, the packet is sent on the Internet with the internal IP send routine. Otherwise, the caller may suspend and wait for the receiver`s window size to increase enough for this packet to be sent. At any given time, only one sender may suspend while trying to send TCP data.

```
void tcp_server_sample_run()
{
    ...
    while (NX_TRUE)
    {
        memset(data_buffer, 0x00, sizeof(data_buffer));

        PRINTF("< Read from client: ");
        len = recv(client_sock, data_buffer, sizeof(data_buffer), 0);
        data_buffer[len] = '\0';
        PRINTF("%d bytes read\r\n", len);

        PRINTF("> Write to client: ");
        len = send(client_sock, data_buffer, len, 0);
        PRINTF("%d bytes written\r\n", len);
    }
    ...
}
```

11.4.3.3 Disconnection

The connection is closed when function close() is called. This function handles sockets to be closed and deleted internally.

```
void tcp_server_sample()
{
    ...
    While (1) {
        Close(client_socket)
        ...
    }

end_of_task:
```

```
    PRINTF("[%s] End of TCP Server sample\r\n", __func__);
    close(listen_sock);
    close(client_sock);
    vTaskDelete(NULL);
    return ;
}
```

11.5 TCP Server in DPM

This section describes how the TCP server is built and works in the DPM sample application. The TCP server in the DPM sample application is an example of the simplest TCP echo server application. The DA16200 SDK can work in DPM mode. The user application is required to work in DPM mode. The DA16200 SDK provides the DPM manager for the user network application. The DPM manager supports the user to develop and manage a network application in Non-DPM and DPM modes. The codes are almost the same as for the TCP server example.

11.5.1 How to Run

1. Open the workspace for the TCP Server in DPM sample application.

   ~/SDK/apps/common/examples/Network/TCP_Server_DPM/projects/da16200

2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. To set the port of the TCP Server Sample, do one of the following:
   - Edit the source code:

     ~/SDK/apps/common/examples/Network/TCP_Server_DPM/src/tcp_server_dpm_sample.c

     #define TCP_SERVER_DPM_SAMPLE_DEF_SERVER_PORT TCP_SVR_TEST_PORT
   - Use the DA16200 console to save the values in NVRAM:

```
[/DA16200] # nvram.setenv TCP_SVR_PORT 10190
[/DA16200] # reboot
```

4. Use the console command to set up the Wi-Fi station interface.
5. When connected to the AP, the sample application creates a TCP server socket with port number 10190 (Default test port number) and waits for client connection.
6. Run a socket application on the peer computer (See Section 11.1.2).
7. Open a TCP client socket.

11.5.2 How It Works

The DA16200 TCP server in the DPM sample application is a simple echo server. When a TCP client sends a message, then the DA16200 TCP server echoes that message to the TCP client.

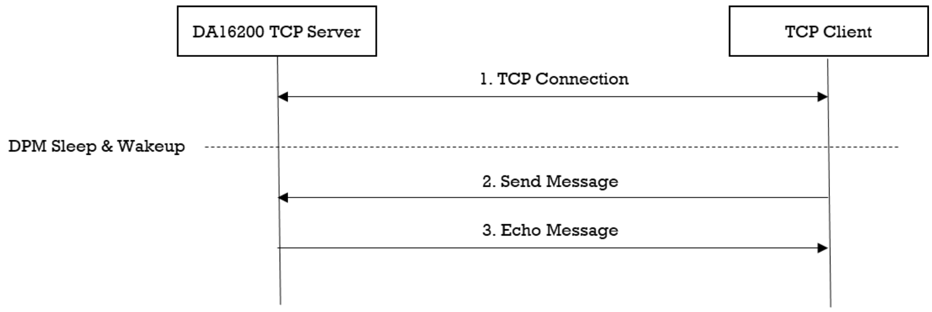


Figure 25. Workflow of TCP server in DPM

11.5.3 Sample Code

11.5.3.1 Registration

The TCP server in the DPM sample application works in DPM mode. The basic code is similar to the TCP server sample application. There are two differences from the TCP Server sample application:

- An initial callback function is added, named `tcp_server_dpm_sample_wakeup_callback()` in the code. The callback is called when the DPM state changes from sleep to wake-up.
- Additional user configuration can be stored in RTM.

In this sample, the TCP server information is stored.

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

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

      //Set DPM wakkup init callback
      user_config->wakeupInitCallback = tcp_server_dpm_sample_wakeup_callback;

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

      //Set session type(TCP Server)
      user_config->sessionConfig[session_idx].sessionType = REG_TYPE_TCP_SERVER;

      //Set local port
      user_config->sessionConfig[session_idx].sessionMyPort =
       TCP_SERVER_DPM_SAMPLE_DEF_SERVER_PORT;

      //Set Connection callback
      user_config->sessionConfig[session_idx].sessionConnectCallback =
       tcp_server_dpm_sample_connect_callback;

      //Set Recv callback
      user_config->sessionConfig[session_idx].sessionRecvCallback =
       tcp_server_dpm_sample_recv_callback;

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

      return ;
}
```

11.5.3.2 Data Transmission

The callback function is called when a TCP packet is received from a TCP client. In this sample, the received data is printed out and an echo message is sent to the TCP client.

```
void tcp_server_dpm_sample_recv_callback(void *sock, UCHAR *rx_buf, UINT rx_len,
                                          ULONG rx_ip, ULONG rx_port)
{
      //Display received packet
      PRINTF(" ==> Received Packet(%ld) \n", rx_len);

      //Echo message
      status = dpm_mng_send_to_session(SESSION1, rx_ip, rx_port,
                                       (char *)rx_buf, rx_len);

```

```
    //Display sent packet
    PRINTF(" <== Sent Packet(%ld) \n", rx_len);

    dpm_mng_job_done(); //Done opertaion
}
```

11.6 TCP Client with KeepAlive in DPM

This section describes how the TCP client with KeepAlive in the DPM sample application is built and works. The TCP client with KeepAlive in the DPM sample application is an example of the simplest TCP echo client application in DPM mode. The DA16200 SDK can work in DPM mode. The user application is required to work in DPM mode. The DA16200 SDK provides the DPM manager for the user network application. The DPM manager helps users to develop and manage a network application in both Non-DPM and DPM modes.

11.6.1 How to Run

1. Run a socket application on the peer computer (see Section 11.1.2) and open a TCP server socket with port number 10193 (Default TCP Client test port).
2. In the e2 studio, import a project for the TCP Client sample application.
   `~/SDK/apps/common/examples/Network/TCP_Client_KeepAlive_DPM/projects/da16200`
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. Use the console command to set up the Wi-Fi station interface.
5. To set the IP address and the port for the peer application (TCP Server) in the TCP Client KA DPM Sample, do one of the following:
   - Edit the source code:

```
~/SDK/apps/common/examples/Network/TCP_Client_KeepAlive_DPM/src/tcp_client_ka_dpm_sample.c
//Default TCP Server configuration
#define TCP_CLIENT_KA_DPM_SAMPLE_DEF_SERVER_IP          "192.168.0.11"
#define TCP_CLIENT_KA_DPM_SAMPLE_DEF_SERVER_PORT        TCP_CLI_KA_TEST_PORT
```

   - Use the DA16200 console to save the values in NVRAM:

```
[/DA16200] # nvram.setenv TCPC_SERVER_IP 192.168.0.11
[/DA16200] # nvram.setenv TCPC_SERVER_PORT 10192
[/DA16200] # reboot
```

After a connection is made to a Wi-Fi AP, the example connects to the peer application (TCP Server).

11.6.2 Sample Code

11.6.2.1 Registration

The TCP client with KeepAlive in the DPM sample application works in DPM mode. The basic code is similar to the TCP client with the KeepAlive sample application. The time period is 55 seconds to send a TCP KeepAlive message to the TCP server. Compared to the TCP client in the DPM sample application, there are two differences from the TCP client sample application:

- An initial callback function is added, named tcp_client_ka_dpm_sample_wakeup_callback() in the code. The callback function is called when the DPM state changes from sleep to wake-up.
- Additional user configuration can be stored in RTM.

In this example, TCP server information is stored.

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

    //Set Boot init callback
    user_config->bootInitCallback = tcp_client_ka_dpm_sample_init_callback;
```

```
    //Set DPM wake-up init callback
    user_config->wakeupInitCallback = tcp_client_ka_dpm_sample_wakeup_callback;

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

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

    //Set local port
    user_config->sessionConfig[session_idx].sessionMyPort =
     TCP_CLIENT_KA_DPM_SAMPLE_DEF_CLIENT_PORT;

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

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

    //Set Connection callback
    user_config->sessionConfig[session_idx].sessionConnectCallback =
     tcp_client_ka_dpm_sample_connect_callback;

    //Set Recv callback
    user_config->sessionConfig[session_idx].sessionRecvCallback =
     tcp_client_ka_dpm_sample_recv_callback;

    //Set connection retry count
    user_config->sessionConfig[session_idx].sessionConnRetryCnt =
     TCP_CLIENT_KA_DPM_SAMPLE_DEF_MAX_CONNECTION_RETRY;

    //Set connection timeout
    user_config->sessionConfig[session_idx].sessionConnWaitTime =
     TCP_CLIENT_KA_DPM_SAMPLE_DEF_MAX_CONNECTION_TIMEOUT;

    //Set auto reconnection flag
    user_config->sessionConfig[session_idx].sessionAutoReconn = pdTRUE;

    //Set KeepAlive timeout
    user_config->sessionConfig[session_idx].sessionKaInterval =
     TCP_CLIENT_KA_DPM_SAMPLE_DEF_KEEPALIVE_TIMEOUT;

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

    return ;
}
```

11.6.2.2 Data Transmission

The callback function is called when a TCP packet is received from the TCP server. In this example, the received data is printed out and an echo message is sent to the TCP server.

```
void tcp_client_ka_dpm_sample_recv_callback(void *sock, UCHAR *rx_buf, UINT rx_len,
                                            ULONG rx_ip, ULONG rx_port)
{
    //Display received packet
    PRINTF(" ==> Received Packet(%ld) \n", rx_len);
```

```
        //Echo message
        status = dpm_mng_send_to_session(SESSION1, rx_ip, rx_port, (char *)rx_buf, rx_len);
        else
        {
                //Display sent packet
                PRINTF(" <== Sent Packet(%ld) \n", rx_len);
        }

        dpm_mng_job_done(); //Done opertaion}
}
```

11.6.3 How It Works

The DA16200 TCP Client with KeepAlive in the DPM sample application is a simple echo message. When the TCP server sends a message, then the DA16200 TCP client echoes that message to the TCP server. A periodic TCP KeepAlive message is sent to the TCP server every 55 seconds.

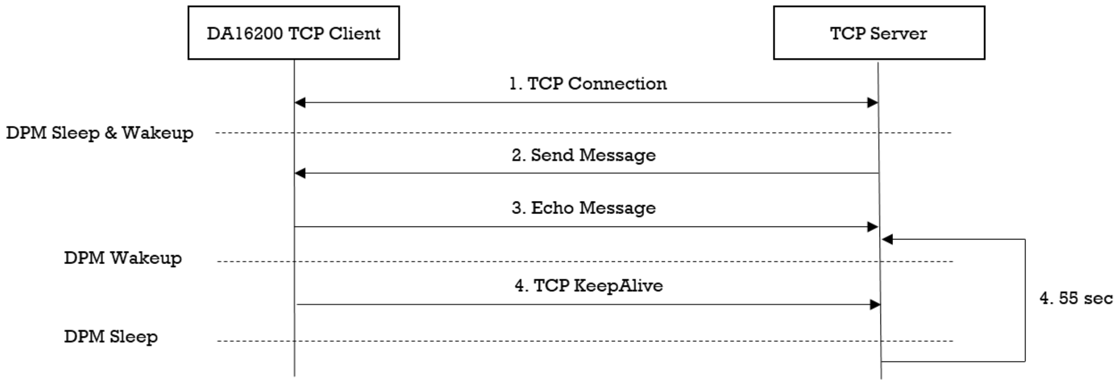


Figure 26. Workflow of TCP client with KeepAlive in DPM

11.7 UDP Socket

This section describes how the UDP socket sample application is built and works. The UDP socket sample application is an example of the simplest UDP echo application. UDP is one of the core members of the Internet protocol suite. It uses a simple connectionless communication model with minimum protocol mechanisms. UDP provides checksums for data integrity and port numbers to address different functions at the source and destination of the datagram. Since there is no handshaking, it exposes the user program to all the instability of the underlying network; there is no guarantee of delivery, order, or duplicate protection. The DA16200 SDK provides a lwIP's TCP protocol. lwIP is an open-source TCP/IP stack designed for embedded systems.

11.7.1 How to Run

1. Run a socket application on the peer computer (see Section 11.1.2) and open a UDP socket with port number 10195 (default UDP test port).
2. In the e² studio, import a project for the UDP socket sample application.
   `~/SDK/apps/common/examples/Network/UDP_Socket/projects/da16200`
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. Use the console command to set up the Wi-Fi station interface.
5. To set the port number for the peer application (UDP Socket) of the UDP Socket Sample, edit the source code:
   `~/SDK/apps/common/examples/Network/UDP_Socket/src/udp_socket_sample.c`

   #define UDP_SOCKET_SAMPLE_DEF_LOCAL_PORT UDP_CLI_TEST_PORT

After a connection is made to a Wi-Fi AP, the example connects to the peer application (UDP Socket).

11.7.2 How It Works

The DA16200 UDP socket sample application is a simple echo server. When a UDP peer sends a message, the DA16200 UDP socket sample application echoes that message to the UDP peer.

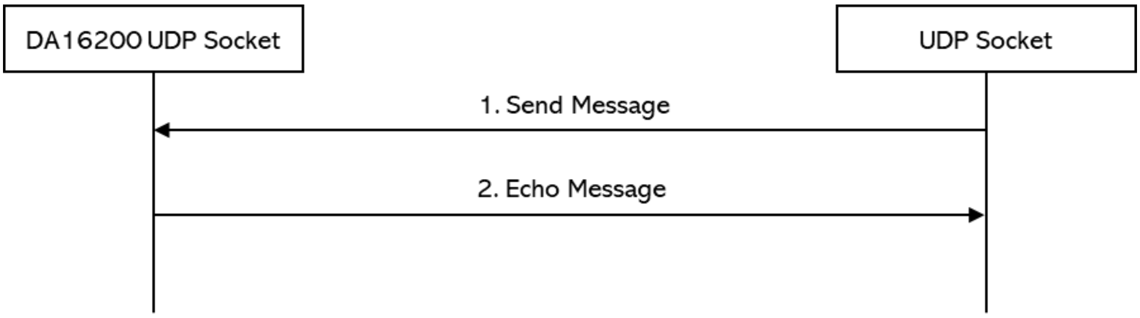


Figure 27. Workflow of UDP socket

11.7.3 Sample Code

The DA16200 SDK provides the lwIP's UDP protocol. This sample application describes how the UDP socket is created, deleted, and configured.

11.7.3.1 Initialization

A UDP port is a logical end point in the UDP protocol. There are 65,535 valid ports in the UDP component of lwIP, ranging from 1 through `0xFFFF`. To send or receive UDP data, the application should first create a UDP socket with function socket(), then bind the UDP socket to the desired port. Next, the application may send and receive data on that socket. The details are as follows:

```
void udp_socket_sample_run()
{
     int sock;

      struct sockaddr_in local_addr;
      struct sockaddr_in peer_addr;

      memset(&local_addr, 0x00, sizeof(local_addr));
      memset(&peer_addr, 0x00, sizeof(peer_addr));

      sock = socket(AF_INET, SOCK_DGRAM, 0);
      setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, (const void *)&optval,sizeof(int));

      local_addr.sin_family = AF_INET;
      local_addr.sin_addr.s_addr = htonl(INADDR_ANY);
      local_addr.sin_port = htons(UDP_SOCKET_SAMPLE_PEER_PORT);

      ret = bind(sock, (struct sockaddr *)&local_addr, sizeof(struct sockaddr_in));

    ...
}
```

11.7.3.2 Data Transmission

To receive a UDP packet, the function `recvfrom()` is called. The socket receive function delivers the oldest packet on the socket's receive queue. To send UDP data, the function `sendto()` is called. This service puts a UDP header in the front part of the packet and sends the packet on the Internet with the internal IP send routine.

```
void udp_socket_sample_run()
{
    ...
      while (1) {
          memset(&peer_addr, 0x00, sizeof(struct sockaddr_in));
```

```
            memset(data_buffer, 0x00, sizeof(data_buffer));

            PRINTF("< Read from peer: ");

            ret = recvfrom(sock, data_buffer, sizeof(data_buffer), 0,
                                 (struct sockaddr *)&peer_addr, (socklen_t *)&addr_len);
            if (ret > 0) {
                len = ret;
                PRINTF("%d bytes read(%d.%d.%d.%d:%d)\r\n", len,
                           (ntohl(peer_addr.sin_addr.s_addr) >> 24) & 0xff,
                           (ntohl(peer_addr.sin_addr.s_addr) >> 16) & 0xff,
                           (ntohl(peer_addr.sin_addr.s_addr) >>  8) & 0xff,
                           (ntohl(peer_addr.sin_addr.s_addr)      ) & 0xff,
                           (ntohs(peer_addr.sin_port)));

                PRINTF("> Write to peer: ");
                ret = sendto(sock, data_buffer, len, 0,
                                (struct sockaddr *)&peer_addr, addr_len);
                PRINTF("%d bytes written(%d.%d.%d.%d:%d)\r\n", len,
                           (ntohl(peer_addr.sin_addr.s_addr) >> 24) & 0xff,
                           (ntohl(peer_addr.sin_addr.s_addr) >> 16) & 0xff,
                           (ntohl(peer_addr.sin_addr.s_addr) >>  8) & 0xff,
                           (ntohl(peer_addr.sin_addr.s_addr)      ) & 0xff,
                           (ntohs(peer_addr.sin_port)));
            }
        }
}
```

11.8 UDP Server in DPM

This section describes how the UDP server in the DPM sample application is built and works. The UDP server in the DPM sample application is an example of the simplest UDP echo application in DPM mode. The DA16200 SDK can work in DPM mode. The DPM manager of the DA16200 SDK is helpful for the user to develop and manage a UDP server socket application in Non-DPM and DPM modes.

11.8.1 How to Run

1. Run a socket application on the peer computer (see Section 11.1.2) and open a UDP socket with port number 10194 (Default UDP test port).
2. In the e² studio, import a project for the UDP Server DPM sample application.
   `~/SDK/apps/common/examples/Network/UDP_Server_DPM/projects/da16200`
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. Use the console command to set up the Wi-Fi station interface.
5. To set the port number for the peer application (UDP Client) of the UDP Server DPM Sample, edit the source code:
   `~/SDK/apps/common/examples/Network/UDP_Server_DPM/src/udp_server_dpm_sample.c`

   #define UDP_SERVER_DPM_SAMPLE_DEF_SERVER_PORT   UDP_SVR_TEST_PORT

After a connection is made to a Wi-Fi AP, the example connects to the peer application (UDP Client).

11.8.2 How It Works

The DA16200 UDP Server in the DPM sample application is a simple echo server. When the peer's UDP application sends a message, the DA16200 UDP server echoes that message to the peer.

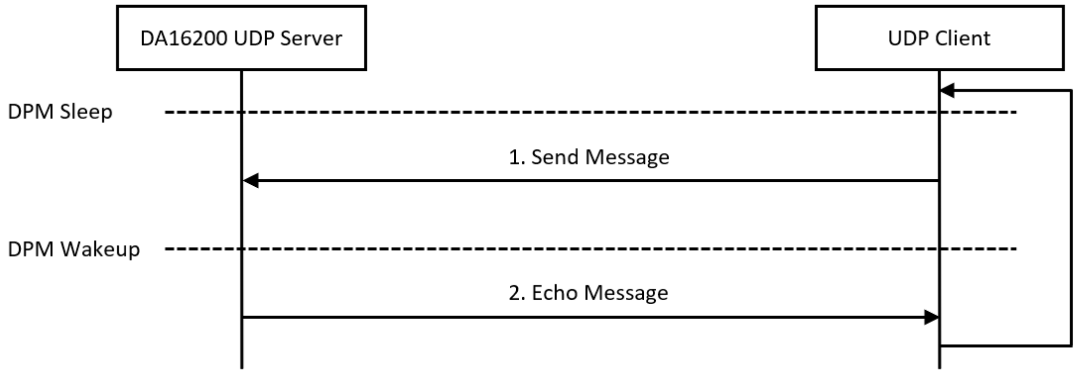


Figure 28. Workflow of UDP server in DPM

11.8.3 Sample Code

11.8.3.1 Registration

The UDP server in the DPM sample application works in DPM mode. The basic code is similar to the UDP server sample application. The only differences are as below:

- An initial callback function is added, named udp_server_dpm_sample_wakeup_callback() in the code. The callback function is called when the DPM state changes from sleep to wake-up.
- Additional user configuration can be stored in RTM.

In this sample, the peer's UDP socket port number is stored.

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

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

      //Set DPM wakkup init callback
      user_config->wakeupInitCallback = udp_server_dpm_sample_wakeup_callback;

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

      //Set session type(UDP Server)
      user_config->sessionConfig[session_idx].sessionType = REG_TYPE_UDP_SERVER;

      //Set local port
      user_config->sessionConfig[session_idx].sessionMyPort =
        UDP_SERVER_DPM_SAMPLE_DEF_SERVER_PORT;

      //Set Connection callback
      user_config->sessionConfig[session_idx].sessionConnectCallback =
        udp_server_dpm_sample_connect_callback;

      //Set Recv callback
      user_config->sessionConfig[session_idx].sessionRecvCallback =
        udp_server_dpm_sample_recv_callback;

      //Set secure mode
      user_config->sessionConfig[session_idx].supportSecure = pdFALSE;

      //Set user configuration
```

```
    user_config->ptrDataFromRetentionMemory = (UCHAR *)&srv_info;
    user_config->sizeOfRetentionMemory = sizeof(udp_server_dpm_sample_svr_info_t);

    return ;
}
```

11.8.3.2 Data Transmission

The callback function is called when a UDP packet is received from the peer's UDP socket application. In this example, the received data is printed out and an echo message is sent to the peer's UDP socket application.

```
void udp_server_dpm_sample_recv_callback(void *sock, UCHAR *rx_buf, UINT rx_len,
                                          ULONG rx_ip, ULONG rx_port)
{
    //Display received packet
    PRINTF(" =====> Received Packet(%ld) \n", 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);
    }
    dpm_mng_job_done(); //Done opertaion }
```

11.9 UDP Client in DPM

This section describes how the UDP client in the DPM sample application is built and works. The UDP client in the DPM sample application is an example of the simplest UDP echo application in DPM mode. The DA16200 SDK can work in DPM mode. The user application requires an additional operation to work in DPM mode. The DPM manager of the DA16200 SDK is helpful for the user to develop and manage a UDP client socket application in both Non-DPM and DPM modes.

11.9.1 How to Run

1. Run a socket application on the peer computer (see Section 11.1.2) and open a UDP socket with port number 10195 (Default UDP test port).
2. In the e[2] studio, import a project for the UDP Client DPM sample application.
   `~/SDK/apps/common/examples/Network/UDP_Client_DPM/projects/da16200`
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. Use the console command to set up the Wi-Fi station interface.
5. To set the port number for the peer application (UDP Server) of the UDP Client DPM Sample, edit the source code:
   `~/SDK/apps/common/examples/Network/ UDP_Client_DPM/src/udp_client_dpm_sample.c`

   #define UDP_CLIENT_DPM_SAMPLE_DEF_SERVER_PORT UDP_CLI_TEST_PORT

After a connection is made to a Wi-Fi AP, the example connects to the peer application (UDP Server).

11.9.2 How It Works

The DA16200 UDP Client in the DPM sample application is a simple echo message. When a peer’s UDP application sends a message, then the DA16200 UDP client echoes that message to the peer.

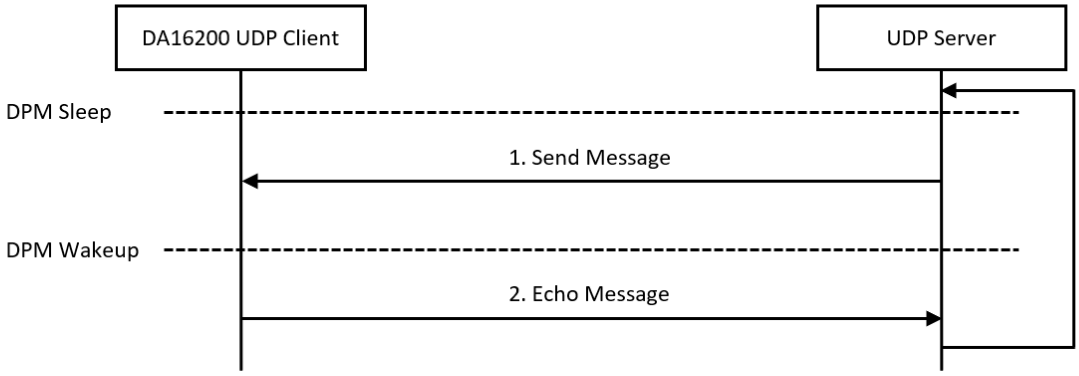


Figure 29. Workflow of UDP client in DPM

11.9.3 Sample Code

11.9.3.1 Registration

The UDP client in the DPM sample application works in DPM mode. The basic code is similar to the UDP client sample application. There are two differences from the UDP client sample application:

- An initial callback function is added, named udp_client_dpm_sample_wakeup_callback() in the code. The function is called when the DPM state changes from sleep to wake-up.
- Additional user configuration can be stored in RTM.

In this example, the peer's UDP IP address and port number are stored.

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

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

      //Set DPM wake up init callback
      user_config->wakeupInitCallback = udp_client_dpm_sample_wakeup_callback;

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

      //Set session type(UDP Client)
      user_config->sessionConfig[session_idx].sessionType = REG_TYPE_UDP_CLIENT;

      //Set local port
      user_config->sessionConfig[session_idx].sessionMyPort =
       UDP_CLIENT_DPM_SAMPLE_DEF_CLIENT_PORT;

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

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

      //Set Connection callback
      user_config->sessionConfig[session_idx].sessionConnectCallback =
       udp_client_dpm_sample_connect_callback;

      //Set Recv callback
      user_config->sessionConfig[session_idx].sessionRecvCallback =
```

```
        udp_client_dpm_sample_recv_callback;

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

      return ;
}
```

11.9.3.2 Data Transmission

The callback function is called when a UDP packet is received from the peer's UDP socket application. In this example, the received data is printed out and an echo message is sent to the peer's UDP socket application.

```
void udp_client_dpm_sample_recv_callback(void *sock, UCHAR *rx_buf, UINT rx_len,
                                         ULONG rx_ip, ULONG rx_port)
{
      //Display received packet
      PRINTF(" ==> Received Packet(%ld) \n", rx_len);

      status = dpm_mng_send_to_session(SESSION1, 0, 0, (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);
      }

      dpm_mng_job_done(); //Done opertaion
 }
```

12. Network Examples: Security

12.1 Peer Application

The examples in this section require a peer device (Laptop or desktop) connected to the same AP running a (D)TLS test application.

12.1.1 Peer Application Examples

There are many (D)TLS counter applications available. In this section, we use a self-implemented (D)TLS counter application to demonstrate these sample applications. It is based on the cryptography APIs of the Bouncy Castle (https://www.bouncycastle.org/java.html). These examples were written and tested on Windows and might be different than the local environment. Use them as references for testing TLS/DTLS servers or clients.

12.1.1.1 TLS Server

The TLS server application is for the DA16200 TLS client sample application. It runs with a default port number (10196) and waits for a TLS client to connect, as shown in Figure 30. One TLS client connection is allowed, and no client certificate is required during the TLS handshake. If the TLS session is established successfully, the TLS server application sends a message per five seconds periodically.

```
C:\Samples>tls_server.exe
**************************************************************
* TLS Server
* ver. 1.0
* Usage: tls_server.exe [Port]
**************************************************************

Bind on tcp/192.168.0.11:10196
```

Figure 30. Start TLS server

12.1.1.2 TLS Client

The TLS client application is for the DA16200 TLS server sample application. It runs with default TLS server information. The IP address is 192.168.0.2 and the port number is 10197. Figure 31 shows the TLS client tries to connect to the DA16200 TLS server sample application. If a TLS session is established successfully, the TLS client application sends a message per 5 seconds periodically.

Usage: tls_client.exe [TLS server IP address] [Port number]

```
C:\Samples>tls_client.exe
**************************************************************
* TLS Client
* ver. 1.0
* Usage: tls_client.exe [TLS Server IP Address] [Port]
**************************************************************

Server IP Address: 192.168.0.2
Server Port: 10197
```

Figure 31. Start TLS client

If the TLS client application cannot find a DA16200 TLS server, an exception occurs with a timeout message as shown in Figure 32.

```
C:\Samples>tls_client.exe
************************************************************
* TLS Client
* ver. 1.0
* Usage: tls_client.exe [TLS Server IP Address] [Port]
************************************************************

Server IP Address: 192.168.0.2
Server Port: 10197
Exception in thread "main" java.net.ConnectException: Connection timed out: connect
        at java.net.DualStackPlainSocketImpl.connect0(Native Method)
        at java.net.DualStackPlainSocketImpl.socketConnect(DualStackPlainSocketImpl.java:79)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:206)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:188)
        at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:172)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
        at java.net.Socket.connect(Socket.java:589)
        at java.net.Socket.connect(Socket.java:538)
        at java.net.Socket.<init>(Socket.java:434)
        at java.net.Socket.<init>(Socket.java:244)
        at da16200_tls_client_sample.TLSEchoClient.openTlsConnection(TLSEchoClient.java:83)
        at da16200_tls_client_sample.TLSEchoClient.main(TLSEchoClient.java:57)
```

Figure 32. TLS client timeout

12.1.1.3 DTLS Server

The DTLS server application is for the DA16200 DTLS client sample application. It runs with a default port number (10199) and waits for the DTLS client to connect, as shown in Figure 33. A client certificate is not required during the DTLS handshake. If a DTLS session is established successfully, the DTLS server application sends a message per five seconds periodically.

```
C:\Samples>dtls_server.exe
************************************************************
* DTLS Server
* ver. 1.0
* Usage: dtls_server.exe [Port]
************************************************************

Bind on udp/192.168.0.11:10199
```

Figure 33. Start DTLS server

12.1.1.4 DTLS Client

The Datagram Transport Layer Security (DTLS) client application is for the DA16200 DTLS server sample application. It runs with default DTLS server information. The IP address is 192.168.0.2 and the port number is 10199. The DTLS client tries to connect to the DA16200 DTLS server sample application as shown in Figure 34. If a DTLS session is established successfully, the DTLS client application sends a message per five seconds periodically.

Usage: dtls_client.exe [DTLS server IP address] [Port number]

```
C:\Samples>dtls_client.exe
************************************************************
* DTLS Client
* ver. 1.0
* Usage: dtls_client.exe [DTLS Server IP Address] [Port]
************************************************************

Server IP Address: 192.168.0.2
Server Port: 10199
```

Figure 34. Start DTLS client

12.2 TLS Server

This section describes how the TLS server sample application is built and works. The TLS server sample application is an example of the simplest TLS echo server application. Transport Layer Security (TLS) is a cryptographic protocol designed to provide communication security over a computer network. The DA16200 SDK provides an SSL library, called `mbedTLS`, on the secure hardware engine to support the TLS protocol. `MbedTLS` is one of the popular SSL libraries. It is helpful to easily develop a network application with a TLS protocol.

12.2.1 How to Run

1. In the e[2] studio, import a project for the TLS Server sample application.
   `~/SDK/apps/common/examples/Network/TLS_Server/projects/da16200`
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. Use the console command to set up the Wi-Fi station interface.
4. After a connection is made to an AP, the sample application creates a TLS server socket with port number 10197 and waits for a client connection.
5. Run a TLS client application on the peer computer.

12.2.2 How It Works

The DA16200 TLS Server sample is a simple echo server. When a TLS client sends a message, the DA16200 TLS server echoes that message to the TLS client.

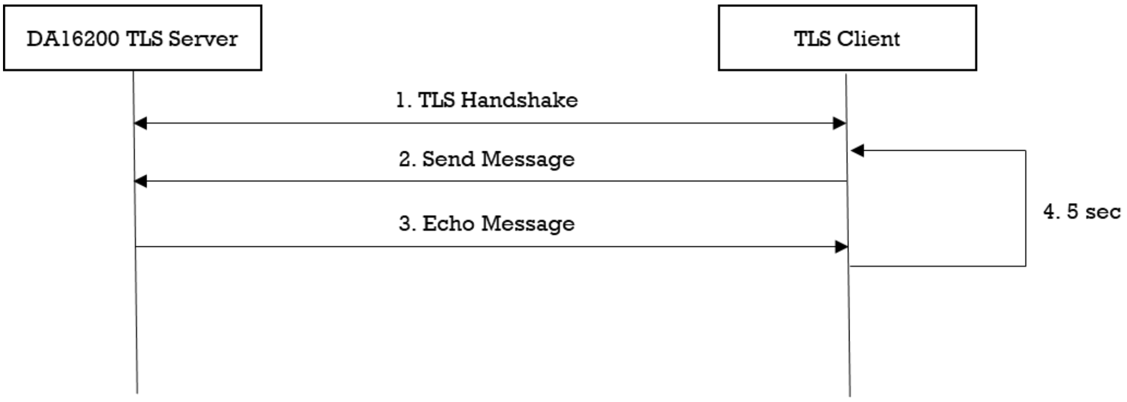


Figure 35. Workflow of TLS server

12.2.3 Sample Code

The DA16200 SDK provides the `mbedTLS` library. This section describes how the TLS server is implemented with an `mbedTLS` library and a socket library.

12.2.3.1 Initialization

The DA16200 secure hardware engine must be initialized with `da16x_secure_module_init()` before the TLS context is initialized. To set up a TLS session, initialization functions are called as follows:

```
void tls_server_sample(void *param)
{
     ...

     //Init session
     mbedtls_net_init(&listen_ctx);
     mbedtls_net_init(&client_ctx);

     //Init SSL context
     mbedtls_ssl_init(&ssl_ctx);

     //Init SSL config
     mbedtls_ssl_config_init(&ssl_conf);

     //Init CTR-DRBG context
     mbedtls_ctr_drbg_init(&ctr_drbg);

     //Init Entropy context
     mbedtls_entropy_init(&entropy);

     //Init Certificate context
```

```
    mbedtls_x509_crt_init(&cert);

    //Init Private key context
    mbedtls_pk_init(&pkey);

    //Init Private key context for ALT
    mbedtls_pk_init(&pkey_alt);

    //Parse certificate
    ret = mbedtls_x509_crt_parse(&cert, tls_server_sample_cert,
                                             tls_server_sample_cert_len);
    //Parse private key
    ret = mbedtls_pk_parse_key(&pkey, tls_server_sample_key,
                                        tls_server_sample_key_len, NULL, 0);

    snprintf(str_port, sizeof(str_port), "%d", TLS_SERVER_SAMPLE_DEF_PORT);
    ret = mbedtls_net_bind(&listen_ctx, NULL, str_port, MBEDTLS_NET_PROTO_TCP);

    ret = mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func, &entropy,
                                 (const unsigned char *)pers, strlen(pers));

    //Set default configuration
    ret = mbedtls_ssl_config_defaults(&ssl_conf, MBEDTLS_SSL_IS_SERVER,
                  MBEDTLS_SSL_TRANSPORT_STREAM, MBEDTLS_SSL_PRESET_DEFAULT);

    mbedtls_ssl_conf_rng(&ssl_conf, mbedtls_ctr_drbg_random, &ctr_drbg);

    //Import certificate & private key
    if (mbedtls_pk_get_type(&pkey) == MBEDTLS_PK_RSA) {
        ret = mbedtls_pk_setup_rsa_alt(&pkey_alt,
                                     (void *)mbedtls_pk_rsa(pkey),
                                      tls_server_sample_rsa_decrypt_func,
                                      tls_server_sample_rsa_sign_func,
                                      tls_server_sample_rsa_key_len_func);

        ret = mbedtls_ssl_conf_own_cert(&ssl_conf, &cert, &pkey_alt);
        if (ret) {
            PRINTF("\r\n[%s] Failed to set certificate(0x%x)\r\n", __func__, -ret);
            goto end_of_task;
        }
    } else {
        ret = mbedtls_ssl_conf_own_cert(&ssl_conf, &cert, &pkey);
        if (ret) {
            PRINTF("\r\n[%s] Failed to set certificate(0x%x)\r\n", __func__, -ret);
            goto end_of_task;
        }
    }
    //Don't care verificate of peer certificate
    mbedtls_ssl_conf_authmode(&ssl_conf, MBEDTLS_SSL_VERIFY_NONE);

    //Set up an SSL context for use.
    ret = mbedtls_ssl_setup(&ssl_ctx, &ssl_conf);

reset:
    ...
    mbedtls_ssl_set_bio(&ssl_ctx, &client_ctx, mbedtls_net_send, mbedtls_net_recv, NULL);
    ...
}
```

12.2.3.2 TLS Handshake

TLS is an encryption protocol designed to secure network communication. A TLS handshake is the process of initiating a communication session that uses TLS encryption. To do a TLS handshake, the function `mbedtls_ssl_handshake()` is called. If an error occurred during the TLS handshake, the API returns a specific error code. If a TLS session is established successfully, the API returns 0. The details are as follows:

```
void tls_server_sample(void *param)
{
    ...
reset:
    ...
   while ((ret = mbedtls_ssl_handshake(&ssl_ctx)) != 0) {
     if ((ret != MBEDTLS_ERR_SSL_WANT_READ) && (ret != MBEDTLS_ERR_SSL_WANT_WRITE)){
         PRINTF("\r\n[%s] Failed to do handshake(0x%x)\r\n", __func__, -ret);
         goto reset;
      }
   }
      ...
 }
```

12.2.3.3 Data Transmission

Encryption scrambles data so that only authorized parties can understand the information. While a TLS session is established, all application data must be encrypted to transfer application data. `MbedTLS` provides specific APIs to help encrypt and decrypt data. To write application data, the function `mbedtls_ssl_write()` of the `mbedTLS` library is called. The details are as follows:

```
void tls_server_sample(void *param)
{
    ...
reset:
    ...
    do {
        ...
        while ((ret = mbedtls_ssl_write(&ssl_ctx, data_buffer, len)) <= 0) {
            switch (ret) {
                case MBEDTLS_ERR_SSL_WANT_READ:
                case MBEDTLS_ERR_SSL_WANT_WRITE:
                    PRINTF("\r\nNeed more data - mbedtls_ssl_write(0x%x)\r\n", -ret);
                    continue;
                case MBEDTLS_ERR_SSL_PEER_CLOSE_NOTIFY:
                    PRINTF("\r\nConnection was closed gracefully\r\n");
                    break;
                case MBEDTLS_ERR_NET_CONN_RESET:
                    PRINTF("\r\nConnection was reset by peer\r\n");
                    break;
                default:
                    PRINTF("Failed to write data(0x%x)\r\n", -ret);
                    break;
            }
            break;
        }
    }
    ...
}
```

To read application data, the function `mbedtls_ssl_read()` of the `mbedTLS` library is called. In this sample, this function is called in `tls_server_sample()`. The details are as follows:

```
void tls_server_sample(void *param)
```

```
{
    ...
reset:
    ...
    do {
        len = sizeof(data_buffer) - 1;
        memset(data_buffer, 0x00, sizeof(data_buffer));

        PRINTF("< Read from client: ");

        ret = mbedtls_ssl_read(&ssl_ctx, data_buffer, len);
        if (ret <= 0) {
            switch (ret) {
                case MBEDTLS_ERR_SSL_WANT_READ:
                case MBEDTLS_ERR_SSL_WANT_WRITE:
                    PRINTF("\r\nNeed more data - mbedtls_ssl_write(0x%x)\r\n", -ret);
                    continue;
                case MBEDTLS_ERR_SSL_PEER_CLOSE_NOTIFY:
                    PRINTF("\r\nConnection was closed gracefully\r\n");
                    break;
                case MBEDTLS_ERR_NET_CONN_RESET:
                    PRINTF("\r\nConnection was reset by peer\r\n");
                    break;
                default:
                    PRINTF("\r\nFailed to read data(0x%x)\r\n", -ret);
                    break;
            }
            break;
        }

        len = ret;
        PRINTF("%d bytes read\r\n", len);

        while ((ret = mbedtls_ssl_write(&ssl_ctx, data_buffer, len)) <= 0) {
            ...
        }
    }
    ...
}
```

12.3 TLS Server in DPM

This section describes how the TLS server in the DPM sample application is built and works. The TLS server in the DPM sample application is an example of the simplest TLS echo server application. TLS is a set of cryptographic protocols designed to provide secure communication over a computer network. The DA16200 SDK can work in DPM mode. The user application requires an additional operation to work in DPM mode. The DA16200 SDK provides a DPM manager for the user network application. The DPM manager supports users to develop and manage a TLS network application in Non-DPM and DPM modes.

12.3.1 How to Run

1. In the e² studio, import a project for the TLS Server in the DPM sample application.
   `~/SDK/apps/common/examples/Network/TLS_Server_DPM/projects/da16200`
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. Use the console command to set up the Wi-Fi station interface.
4. After a connection is made to an AP, the example application creates a TLS server socket with port number 10197 and waits for a client connection.
5. Run a TLS client application on the peer computer.

12.3.2 How It Works

The DA16200 TLS Server in the DPM sample is a simple echo server. When a TLS client sends a message, then the DA16200 TLS server echoes that message to the TLS client. The DA16200 TLS server takes time to wait to establish a TLS session.

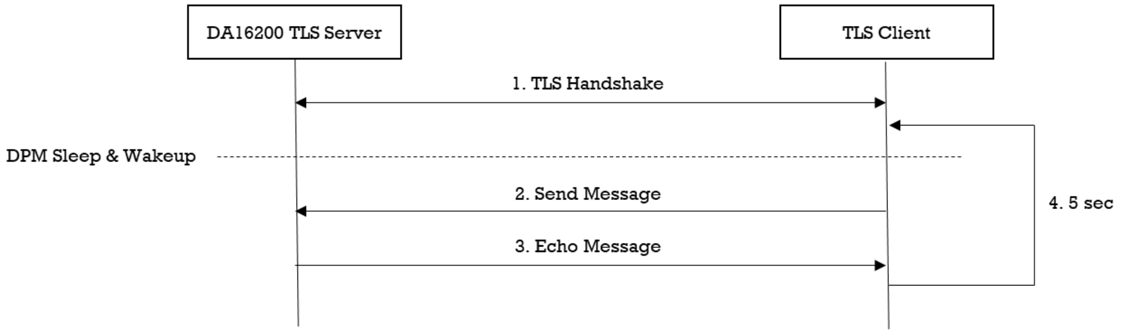


Figure 36. Workflow of TLS server in DPM

12.3.3 Sample Code

12.3.3.1 Registration

The TLS server in the DPM sample application works in DPM mode. The basic code is similar to the TLS server sample application. There are two differences with the TLS Server sample application:

- An initial callback function is added, named `tls_server_dpm_sample_wakeup_callback()` in the code. The function is called when the DPM state changes from sleep to wake-up.
- Additional user configuration can be stored in RTM.

In this sample, the TLS server information is stored.

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

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

      //Set DPM wakkup init callback
      user_config->wakeupInitCallback = tls_server_dpm_sample_wakeup_callback;

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

      //Set session type(TCP Server)
      user_config->sessionConfig[session_idx].sessionType = REG_TYPE_TCP_SERVER;

      //Set local port
      user_config->sessionConfig[session_idx].sessionMyPort =
        TLS_SERVER_DPM_SAMPLE_DEF_SERVER_PORT;

      //Set Connection callback
      user_config->sessionConfig[session_idx].sessionConnectCallback =
        tls_server_dpm_sample_connect_callback;

      //Set Recv callback
      user_config->sessionConfig[session_idx].sessionRecvCallback =
        tls_server_dpm_sample_recv_callback;

      //Set secure mode
      user_config->sessionConfig[session_idx].supportSecure = pdTRUE;
```

```
    //Set secure setup callback
    user_config->sessionConfig[session_idx].sessionSetupSecureCallback =
     tls_server_dpm_sample_secure_callback;

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

    return ;
}
```

12.3.3.2 TLS Setup

To establish a TLS session, TLS should be set up. DA16200 includes the `mbedTLS` library to provide the TLS protocol. Most APIs that are related to the TLS protocol are based on the `mbedTLS` library. TLS is set up by `sessionSetupSecureCallback` function. The details are as follows.

```
void tls_server_dpm_sample_secure_callback(void *config)
{
    const char *pers = "tls_server_dpm_sample";
    SECURE_INFO_T *secure_config = (SECURE_INFO_T *)config;

    ret = mbedtls_ssl_config_defaults(secure_config->ssl_conf,
                                      MBEDTLS_SSL_IS_SERVER,
                                      MBEDTLS_SSL_TRANSPORT_STREAM,
                                      MBEDTLS_SSL_PRESET_DEFAULT);

    //import test certificate
    ret = mbedtls_x509_crt_parse(secure_config->cert_crt,
                                 tls_server_dpm_sample_cert,
                                 tls_server_dpm_sample_cert_len);

    ret = mbedtls_pk_parse_key(secure_config->pkey_ctx,
                               tls_server_dpm_sample_key,
                               tls_server_dpm_sample_key_len,
                               NULL, 0);

    if (mbedtls_pk_get_type(secure_config->pkey_ctx) == MBEDTLS_PK_RSA) {
        ret = mbedtls_pk_setup_rsa_alt(secure_config->pkey_alt_ctx,
                                (void *)mbedtls_pk_rsa(*secure_config->pkey_ctx),
                                 tls_server_dpm_sample_rsa_decrypt_func,
                                 tls_server_dpm_sample_rsa_sign_func,
                                 tls_server_dpm_sample_rsa_key_len_func);

        ret = mbedtls_ssl_conf_own_cert(secure_config->ssl_conf,
                                        secure_config->cert_crt,
                                        secure_config->pkey_alt_ctx);
    } else {
        ret = mbedtls_ssl_conf_own_cert(secure_config->ssl_conf,
                                        secure_config->cert_crt,
                                        secure_config->pkey_ctx);
    }

    ret = dpm_mng_setup_rng(secure_config->ssl_conf);

    //Don't care verification in this sample.
    mbedtls_ssl_conf_authmode(secure_config->ssl_conf, MBEDTLS_SSL_VERIFY_NONE);

    ret = mbedtls_ssl_setup(secure_config->ssl_ctx, secure_config->ssl_conf);

    dpm_mng_job_done(); //Done opertaion
```

```
    return ;
}
```

12.3.3.3 Data Transmission

The callback function is called when a TLS packet is received from a TLS client. In this sample, the received data is printed out and an echo message is sent to the TLS server. Data is encrypted and decrypted in the callback function.

```
void tls_server_dpm_sample_recv_callback(void *sock, UCHAR *rx_buf, UINT rx_len,
                                         ULONG rx_ip, ULONG rx_port)
{
    //Display received packet
    PRINTF(" =====> Received Packet(%ld) \n", 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);
    }

    dpm_mng_job_done(); //Done opertaion
}
```

12.4 TLS Client

This section describes how the TLS client sample application is built and works. The TLS client sample application is an example of the simplest TLS echo client application. TLS is a cryptographic protocol designed to provide secure communication over a computer network. The DA16200 SDK provides a DPM manager for the user network application. The DA16200 SDK provides an SSL library called `mbedTLS` on a secure hardware engine to support the TLS protocol. `MbedTLS` is one of the popular SSL libraries and helps to easily develop a network application with a TLS protocol.

12.4.1 How to Run

1. Run a TLS server application on the peer computer and open a TLS server socket with port number 10196.
2. In the e[2] studio, import a project for the TLS Client sample application.
   `~/SDK/apps/common/examples/Network/TLS_Client/projects/da16200`
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. Use the console command to set up the Wi-Fi station interface.
5. After a connection is made to an AP, the example application connects to the peer.

12.4.2 How It Works

The DA16200 TLS Client sample is a simple echo message. When the TLS server sends a message, then the DA16200 TLS client echoes that message to the TLS server.

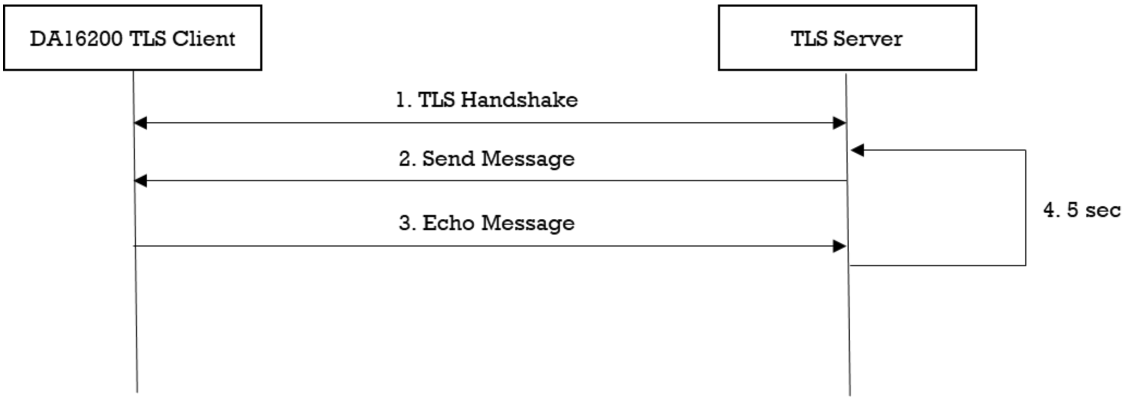


Figure 37. Workflow of TLS client

12.4.3 Sample Code

DA16200 SDK provides the `mbedTLS` library. This section describes how the TLS client is implemented with the `mbedTLS` library and socket library.

12.4.3.1 Registration

The DA16200 secure hardware engine must be initialized with da16x_secure_module_init() before the TLS context is initialized. To set up a TLS session, initialization functions are called as follows:

```
void tls_client_sample(void *param)
{
     ...

     //Init session
     mbedtls_net_init(&server_ctx);

     //Init SSL context
     mbedtls_ssl_init(&ssl_ctx);

     //Init SSL config
     mbedtls_ssl_config_init(&ssl_conf);

     //Init CTR-DRBG context
     mbedtls_ctr_drbg_init(&ctr_drbg);

     //Init Entropy context
     mbedtls_entropy_init(&entropy);

     snprintf(str_port, sizeof(str_port), "%d", TLS_CLIENT_SAMPLE_DEF_SERVER_PORT);
     ret = mbedtls_net_connect(&server_ctx,
                               TLS_CLIENT_SAMPLE_DEF_SERVER_IP_ADDR, str_port,
                               MBEDTLS_NET_PROTO_TCP);

     //Set default configuration
     ret = mbedtls_ssl_config_defaults(&ssl_conf,
                                       MBEDTLS_SSL_IS_CLIENT,
                                       MBEDTLS_SSL_TRANSPORT_STREAM,
                                       MBEDTLS_SSL_PRESET_DEFAULT);

     ret = mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func, &entropy,
                                 (const unsigned char *)pers, strlen(pers));

     mbedtls_ssl_conf_rng(&ssl_conf, mbedtls_ctr_drbg_random, &ctr_drbg);

     //Don't care verification in this sample.
     mbedtls_ssl_conf_authmode(&ssl_conf, MBEDTLS_SSL_VERIFY_NONE);
```

```
    //Setup an SSL context for use.
    ret = mbedtls_ssl_setup(&ssl_ctx, &ssl_conf);

    mbedtls_ssl_set_bio(&ssl_ctx, &server_ctx,
                              mbedtls_net_send, mbedtls_net_recv, NULL);

    ...
}
```

12.4.3.2 TLS Handshake

TLS is an encryption protocol designed to secure network communication. A TLS handshake is the process that starts a communication session that uses TLS encryption. To do a TLS handshake, the function `mbedtls_ssl_handshake()` is called. If an error occurred during the TLS handshake, the API returns a specific error code. If a TLS session is established successfully, the API returns 0. The details are as follows:

```
void tls_client_sample(void *param)
{
    ...
reset:
    ...
    while ((ret = mbedtls_ssl_handshake(&ssl_ctx)) != 0) {
       if (ret == MBEDTLS_ERR_NET_CONN_RESET) {
            PRINTF("\r\n[%s] Peer closed the connection(0x%x)\r\n", __func__, -ret);
            goto end_of_task;
       }

       if ((ret != MBEDTLS_ERR_SSL_WANT_READ) &&
                                     (ret != MBEDTLS_ERR_SSL_WANT_WRITE)) {
            PRINTF("\r\n[%s] Failed to do tls handshake(0x%x)\r\n", __func__, -ret);
            goto end_of_task;
       }
    }
    ...
 }
```

12.4.3.3 Data Transmission

Encryption scrambles data so that only authorized parties can understand the information. While a TLS session is established, all data must be encrypted to transfer application data. `MbedTLS` provides specific APIs to help encrypt and decrypt data. To write application data, the function `mbedtls_ssl_write()` of the `mbedTLS` library is called. The details are as follows:

```
void tls_client_sample(void *param)
{
    ...
    do {
        ...
        while ((ret = mbedtls_ssl_write(&ssl_ctx, data_buffer, len)) <= 0) {
            switch (ret) {
                case MBEDTLS_ERR_SSL_WANT_READ:
                case MBEDTLS_ERR_SSL_WANT_WRITE:
                    PRINTF("\r\nNeed more data - mbedtls_ssl_write(0x%x)\r\n", -ret);
                    continue;
                case MBEDTLS_ERR_SSL_PEER_CLOSE_NOTIFY:
                    PRINTF("\r\nConnection was closed gracefully\r\n");
                    break;
                case MBEDTLS_ERR_NET_CONN_RESET:
                    PRINTF("\r\nConnection was reset by peer\r\n");
```

```
                break;
            default:
                PRINTF("\r\nFailed to write data(0x%x)\r\n", -ret);
                break;
            }
            goto end_of_task;
        }
    }
    ...
}
```

To read application data, the function `mbedtls_ssl_read()` of the `mbedTLS` library is called. In this sample, this function is called in `tls_client_sample()`. The details are as follows:

```
void tls_client_sample(void *param)
{
    ...
    do {
        len = sizeof(data_buffer) - 1;
        memset(data_buffer, 0x00, sizeof(data_buffer));

        PRINTF("< Read from server: ");

        //Read at most 'len' application data bytes.
        ret = mbedtls_ssl_read(&ssl_ctx, data_buffer, len);
        if (ret <= 0) {
            switch (ret) {
            case MBEDTLS_ERR_SSL_WANT_READ:
            case MBEDTLS_ERR_SSL_WANT_WRITE:
                PRINTF("\r\nNeed more data - mbedtls_ssl_read(0x%x)\r\n", -ret);
                continue;
            case MBEDTLS_ERR_SSL_PEER_CLOSE_NOTIFY:
                PRINTF("\r\nConnection was closed gracefully\r\n");
                goto end_of_task;
            case MBEDTLS_ERR_NET_CONN_RESET:
                PRINTF("\r\nConnection was reset by peer\r\n");
                goto end_of_task;
            default:
                PRINTF("\r\nFailed to read data(0x%x)\r\n", -ret);
                break;
            }
            goto end_of_task;
        }

       len = ret;
       PRINTF("%d bytes read\r\n", len);

        while ((ret = mbedtls_ssl_write(&ssl_ctx, data_buffer, len)) <= 0) {
            ...
        }
    }
    ...
}
```

12.5 TLS Client in DPM

This section describes how the TLS client in the DPM sample application is built and works. The TLS client in the DPM sample application is an example of the simplest TLS echo client application in DPM mode. TLS is a set of cryptographic protocols designed to provide secure communication over a computer network. The DA16200

SDK can work in DPM mode. The user application requires an additional operation to work in DPM mode. The DA16200 SDK provides a DPM manager for the user network application. The DPM manager supports the user to develop and manage the TLS network application in Non-DPM and DPM modes.

12.5.1 How to Run

1. Run a TLS server application on the peer computer and open a TLS server socket with port number 10196.
2. In the e[2] studio, import a project for a TCP Client in the DPM sample application.
   `~/SDK/apps/common/examples/Network/TLS_Client_DPM/projects/da16200`
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. Use the console command to set up the Wi-Fi station interface.
5. Set the TLS server IP address and the port number as created the socket on the peer computer with the following console command and then reboot. These parameters can also be defined in the source code.

```
[/DA16200] # nvram.setenv TLSC_SERVER_IP 192.168.0.11
[/DA16200] # nvram.setenv TLSC_SERVER_PORT 10196
[/DA16200] # reboot
```

After connecting to the AP, the example application connects to the peer computer.

12.5.2 How It Works

The DA16200 TLS Client in the DPM sample is a simple echo message. When a TLS server sends a message, then the DA16200 TLS client echoes that message to the TLS server.

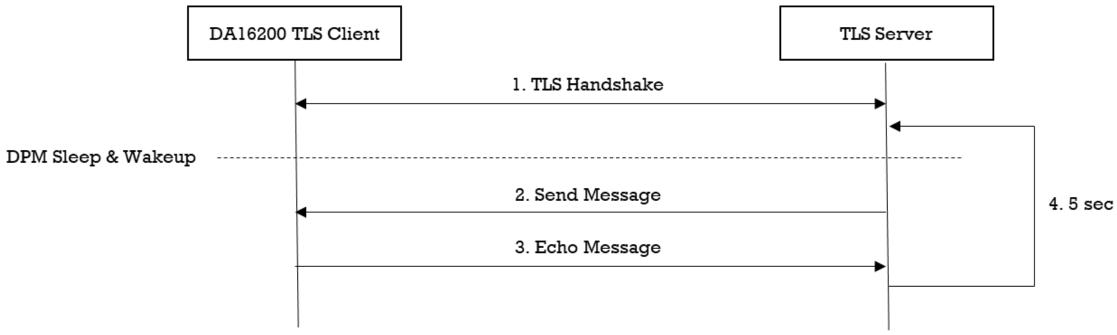


Figure 38. Workflow of TLS client in DPM

12.5.3 Sample Code

12.5.3.1 Registration

The TLS client in the DPM sample application works in DPM mode. The basic code is similar to the TLS client sample application. There are two differences with the TLS client sample application:

- An initial callback function is added, named `tls_client_dpm_sample_wakeup_callback()` in the code. It is called when the DPM state changes from sleep to wake-up.
- Additional user configuration that can be stored in RTM.

In this example, TLS server information is stored.

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

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

      //Set DPM wake-up init callback
      user_config->wakeupInitCallback = tls_client_dpm_sample_wakeup_callback;

      //Set External wake-up callback
```

```
    user_config->externWakeupCallback = tls_client_dpm_sample_external_callback;

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

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

    //Set local port
    user_config->sessionConfig[session_idx].sessionMyPort =
     TLS_CLIENT_DPM_SAMPLE_DEF_CLIENT_PORT;

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

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

    //Set Connection callback
    user_config->sessionConfig[session_idx].sessionConnectCallback =
     tls_client_dpm_sample_connect_callback;

    //Set Recv callback
    user_config->sessionConfig[session_idx].sessionRecvCallback =
     tls_client_dpm_sample_recv_callback;

    //Set connection retry count
    user_config->sessionConfig[session_idx].sessionConnRetryCnt =
     TLS_CLIENT_DPM_SAMPLE_DEF_MAX_CONNECTION_RETRY;

    //Set connection timeout
    user_config->sessionConfig[session_idx].sessionConnWaitTime =
     TLS_CLIENT_DPM_SAMPLE_DEF_MAX_CONNECTION_TIMEOUT;

    //Set auto reconnection flag
    user_config->sessionConfig[session_idx].sessionAutoReconn = pdTRUE;

    //Set secure mode
    user_config->sessionConfig[session_idx].supportSecure = pdTRUE;

    //Set secure setup callback
    user_config->sessionConfig[session_idx].sessionSetupSecureCallback =
     tls_client_dpm_sample_secure_callback;

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

    return ;
}
```

12.5.3.2 TLS Setup

To establish a TLS session, TLS should be set up. DA16200 includes the `mbedTLS` library to provide the TLS protocol. Most APIs that are related to the TLS protocol are based on an `mbedTLS` library. TLS is set up by `sessionSetupSecureCallback` function. The details are as shown below. This sample application does not include certificates.

```
void tls_client_dpm_sample_secure_callback(void *config)
{
```

```
    const char *pers = "tls_client_sample";
    SECURE_INFO_T *secure_config = (SECURE_INFO_T *)config;

    ret = mbedtls_ssl_config_defaults(secure_config->ssl_conf,
                                      MBEDTLS_SSL_IS_CLIENT,
                                      MBEDTLS_SSL_TRANSPORT_STREAM,
                                      MBEDTLS_SSL_PRESET_DEFAULT);

    ret = dpm_mng_setup_rng(secure_config->ssl_conf);

    //Don't care verification in this sample.
    mbedtls_ssl_conf_authmode(secure_config->ssl_conf, MBEDTLS_SSL_VERIFY_NONE);

    ret = mbedtls_ssl_setup(secure_config->ssl_ctx, secure_config->ssl_conf);

    dpm_mng_job_done(); //Done opertaion
    return ;
}
```

12.5.3.3 Data Transmission

The callback function is called when the TLS packet is received from the TLS server. In this sample, the received data is printed out and an echo message is sent to the TLS server. Data is encrypted and decrypted in the callback function.

```
void tls_client_dpm_sample_recv_callback(void *sock, UCHAR *rx_buf, UINT rx_len,
                                         ULONG rx_ip, ULONG rx_port)
{
    //Display received packet
    PRINTF(" =====> Received Packet(%ld) \n", rx_len);

    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);
    }

    dpm_mng_job_done(); //Done opertaion}
```

12.6 DTLS Server

This section describes how the Datagram Transport Layer Security (DTLS) server sample application is built and works. The DTLS server sample application is an example of the simplest DTLS echo server application. DTLS is a cryptographic protocol that provides security for datagram-based applications by allowing them to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. The DA16200 SDK provides an SSL library called `mbedTLS` on a secure hardware engine to support the DTLS protocol. `mbedTLS` is one of the popular SSL libraries. `mbedTLS` is helpful to develop a network application with a DTLS protocol.

12.6.1 How to Run

1. In the e² studio, import a project for the DTLS Server sample application.
   `~/SDK/apps/common/examples/Network/DTLS_Server/projects/da16200`

2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. Use the console command to set up the Wi-Fi station interface.
4. After a connection is made to an AP, the example application creates a DTLS server socket with port number 10199 and waits for a client connection.
5. Run a DTLS client application on the peer computer.

12.6.2 How It Works

The DA16200 DTLS Server sample is a simple echo server. When the DTLS client sends a message, then the DA16200 DTLS server echoes that message to the DTLS client.

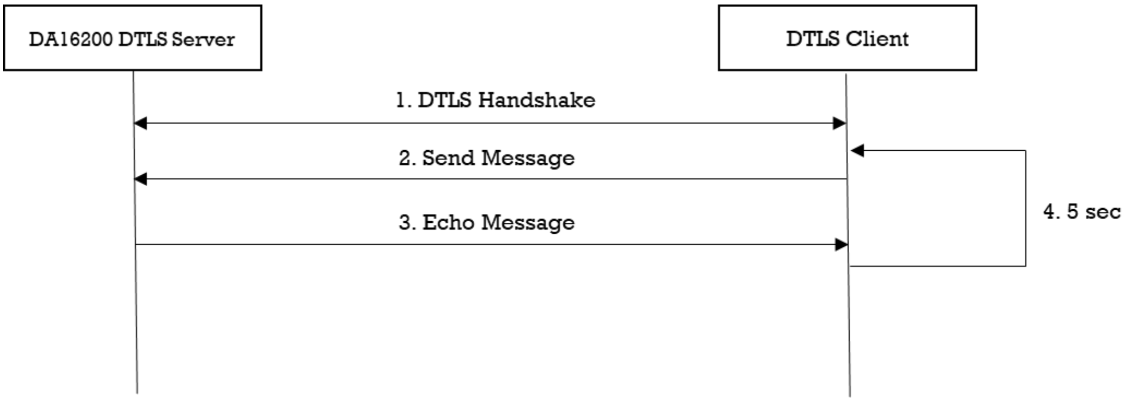


Figure 39. Workflow of DTLS server

12.6.3 Sample Code

The DA16200 SDK provides the `mbedTLS` library. This sample application describes how the `mbedTLS` library is called and applied for the socket library.

12.6.3.1 Initialization

The DA16200 secure hardware engine must be initialized with `da16x_secure_module_init()` before the TLS context is initialized. To set up a DTLS session, initialization functions are called as shown in the example code below. The DTLS session is established over a UDP protocol. In case a packet is lost, retransmission is required. So, the timer is registered to retransmit packet by function `mbedtls_ssl_set_timer_cb()`.

```
void dtls_server_sample(void *param)
{
     ...

     //Init session
     mbedtls_net_init(&listen_ctx);
     mbedtls_net_init(&client_ctx);

     //Init SSL context
     mbedtls_ssl_init(&ssl_ctx);

     //Init SSL config
     mbedtls_ssl_config_init(&ssl_conf);

     //Init CTR-DRBG context
     mbedtls_ctr_drbg_init(&ctr_drbg);

     //Init Entropy context
     mbedtls_entropy_init(&entropy);

     //Init Certificate context
     mbedtls_x509_crt_init(&cert);
```

```
    //Init Private key context
    mbedtls_pk_init(&pkey);

    //Init Private key context for ALT
    mbedtls_pk_init(&pkey_alt);

    //Init Cookies
    mbedtls_ssl_cookie_init(&cookies);
    memset(&timer, 0x00, sizeof(dtls_server_sample_timer_t));

    //Parse certificate
    ret = mbedtls_x509_crt_parse(&cert, dtls_server_sample_cert,
                                 dtls_server_sample_cert_len);
    //Parse private key
    ret = mbedtls_pk_parse_key(&pkey, dtls_server_sample_key,
                               dtls_server_sample_key_len, NULL, 0);

    snprintf(str_port, sizeof(str_port), "%d", DTLS_SERVER_SAMPLE_DEF_PORT);

    ret = mbedtls_net_bind(&listen_ctx, NULL, str_port, MBEDTLS_NET_PROTO_UDP);
    ret = mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func, &entropy,
                                (const unsigned char *)pers, strlen(pers));

    //Set default configuration
    ret = mbedtls_ssl_config_defaults(&ssl_conf,
                                      MBEDTLS_SSL_IS_SERVER,
                                      MBEDTLS_SSL_TRANSPORT_DATAGRAM,
                                      MBEDTLS_SSL_PRESET_DEFAULT);

    mbedtls_ssl_conf_rng(&ssl_conf, mbedtls_ctr_drbg_random, &ctr_drbg);

    //Import certificate & private key
    if (mbedtls_pk_get_type(&pkey) == MBEDTLS_PK_RSA) {
        ret = mbedtls_pk_setup_rsa_alt(&pkey_alt,
                                       (void *)mbedtls_pk_rsa(pkey),
                                       dtls_server_sample_rsa_decrypt_func,
                                       dtls_server_sample_rsa_sign_func,
                                       dtls_server_sample_rsa_key_len_func);

        ret = mbedtls_ssl_conf_own_cert(&ssl_conf, &cert, &pkey_alt);
    } else {
        ret = mbedtls_ssl_conf_own_cert(&ssl_conf, &cert, &pkey);
    }

    //Setup cookies
    ret = mbedtls_ssl_cookie_setup(&cookies, mbedtls_ctr_drbg_random, &ctr_drbg);

    //Register callbacks for DTLS cookies.
    mbedtls_ssl_conf_dtls_cookies(&ssl_conf,
                                  mbedtls_ssl_cookie_write,
                                  mbedtls_ssl_cookie_check,
                                  &cookies);

    //Don't care verificate of peer certificate
    mbedtls_ssl_conf_authmode(&ssl_conf, MBEDTLS_SSL_VERIFY_NONE);

    //Enable or disable anti-replay protection for DTLS.
    mbedtls_ssl_conf_dtls_anti_replay(&ssl_conf, MBEDTLS_SSL_ANTI_REPLAY_ENABLED);
    mbedtls_ssl_conf_read_timeout(&ssl_conf, DTLS_SERVER_SAMPLE_DEF_TIMEOUT);

    //Set retransmit timeout values for the DTLS handshake.
```

```
      mbedtls_ssl_conf_handshake_timeout(&ssl_conf,
                                         DTLS_SERVER_SAMPLE_DEF_HANDSHAKE_MIN_TIMEOUT,
                                         DTLS_SERVER_SAMPLE_DEF_HANDSHAKE_MAX_TIMEOUT);

      //Set up an SSL context for use.
      ret = mbedtls_ssl_setup(&ssl_ctx, &ssl_conf);
      mbedtls_ssl_set_timer_cb(&ssl_ctx, &timer, dtls_server_sample_timer_start,
                               dtls_server_sample_timer_get_state);

reset:
      ...
      ret = mbedtls_ssl_set_client_transport_id(&ssl_ctx, client_ip, client_ip_len);
      mbedtls_ssl_set_bio(&ssl_ctx, &client_ctx, mbedtls_net_send, NULL,
                          mbedtls_net_recv_timeout);
      ...
}
```

12.6.3.2 DTLS Handshake

DTLS is an encryption protocol designed to secure network communication. A DTLS handshake is the process that starts a communication session with DTLS encryption. To make a DTLS handshake, the application calls function `mbedtls_ssl_handshake()`. The DTLS server must verify cookies for the DTLS client. The DTLS client's transport-level identification information must be set up (generally an IP Address). After a ClientHello message is received, the DTLS server must set up its IP address. Then, a DTLS handshake should be retried as follows:

```
void dtls_server_sample(void *param)
{
    ...
reset:
    ...
    while ((ret = mbedtls_ssl_handshake(&ssl_ctx)) != 0) {
        if ((ret == MBEDTLS_ERR_SSL_WANT_READ) ||
                                              (ret == MBEDTLS_ERR_SSL_WANT_WRITE)) {
              continue;
        }
        if (ret == MBEDTLS_ERR_SSL_HELLO_VERIFY_REQUIRED) {
              PRINTF("hello verification requested\r\n");
              ret = 0;
              goto reset;
        } else {
              PRINTF("\r\n[%s] Failed to do handshake(0x%x)\r\n", __func__, -ret);
              goto reset;
        }
    }
    ...
 }
```

12.6.3.3 Data Transmission

Encryption scrambles data so that only authorized parties can understand the information. While a DTLS session is established, all data must be encrypted for transfer. `mbedTLS` provides specific APIs to help encrypt and decrypt data. To write application data, the function `mbedtls_ssl_write()` of the `mbedTLS` library is called. The details are as follows:

```
void dtls_server_sample(void *param){
    ...
    do {
        ...
        while ((ret = mbedtls_ssl_write(&ssl_ctx, data_buffer, len)) <= 0) {
            switch (ret) {
                case MBEDTLS_ERR_SSL_WANT_READ:
```

```
                case MBEDTLS_ERR_SSL_WANT_WRITE:
                    PRINTF("\r\nNeed more data - mbedtls_ssl_write(0x%x)\r\n", -ret);
                    continue;
            }
            PRINTF("\r\n[%s] Failed to write data(0x%x)\r\n", __func__, -ret);
            goto end_of_task;
        }
        PRINTF("%d bytes written\r\n", len);
    }
    ...
}
```

To read application data, the function `mbedtls_ssl_read()` of the `mbedTLS` library is called. In this sample, this function is called in `dtls_server_sample()`. The details are as follows:

```
void dtls_server_sample(void *param){
    ...
    do {
        len = sizeof(data_buffer) - 1;
        memset(data_buffer, 0x00, sizeof(data_buffer));

        PRINTF("< Read from server: ");

        //Read at most 'len' application data bytes.
        ret = mbedtls_ssl_read(&ssl_ctx, data_buffer, len);
        if (ret <= 0) {
            switch (ret) {
                case MBEDTLS_ERR_SSL_WANT_READ:
                case MBEDTLS_ERR_SSL_WANT_WRITE:
                    PRINTF("\r\nNeed more data - mbedtls_ssl_write(0x%x)\r\n", -ret);
                    continue;
                case MBEDTLS_ERR_SSL_PEER_CLOSE_NOTIFY:
                    PRINTF("\r\nConnection was closed gracefully\r\n");
                    ret = 0;
                    goto close_notify;
                case MBEDTLS_ERR_SSL_TIMEOUT:
                    PRINTF("\r\nTimeout\r\n");
                    goto reset;
                default:
                    PRINTF("\r\nFailed to read data(0x%x)\r\n", -ret);
                    break;
            }
            goto reset;
        }

        len = ret;
        PRINTF("%d bytes read\r\n", len);

        PRINTF("> Write to client: ");
        while ((ret = mbedtls_ssl_write(&ssl_ctx, data_buffer, len)) <= 0) {      ...
        }
    }
    ...
}
```

12.7 DTLS Server in DPM

This section describes how the DTLS server in the DPM sample application is built and works. The DTLS server in the DPM sample application is an example of the simplest DTLS echo server application. DTLS is a cryptographic protocol that provides security for datagram-based applications by allowing them to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. The DA16200 SDK can work in DPM mode. The user application requires an additional operation to work in DPM mode. The DA16200 SDK provides a DPM manager for the user network application. The DPM manager supports the user to develop and manage a DTLS network application in Non-DPM and DPM modes.

12.7.1 How to Run

1. In the e² studio, import a project for the DTLS Server in the DPM sample application.
   `~/SDK/apps/common/examples/Network/DTLS_Server_DPM/projects/da16200`
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. Use the console command to set up the Wi-Fi station interface.
4. After a connection is made to an AP, the example application creates a DTLS server socket with port number 10199 and waits for a client connection.
5. Run a DTLS client application on the peer computer.

12.7.2 How It Works

The DA16200 DTLS Server in the DPM sample is a simple echo server. When a DTLS client sends a message, then the DA16200 DTLS server echoes that message to the DTLS client.

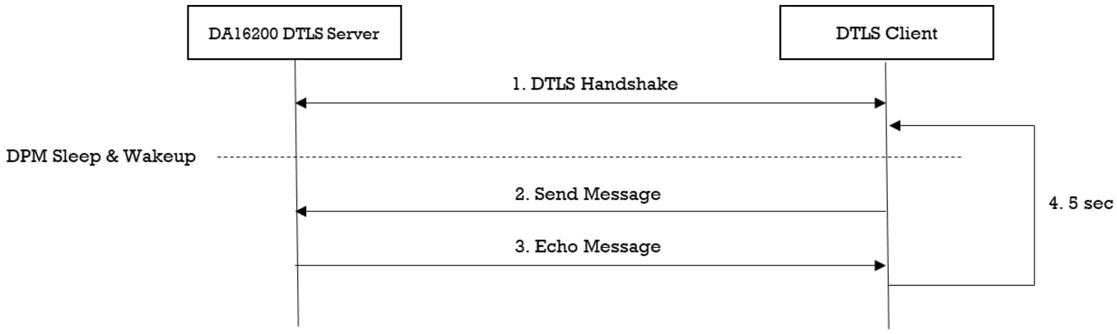


Figure 40. Workflow of DTLS server in DPM

12.7.3 Sample Code

12.7.3.1 Registration

The DTLS server in the DPM sample application works in DPM mode. The basic code is similar to the DTLS server sample application. There are two differences with the DTLS server sample application:

- An initial callback function is added, named dtls_server_dpm_sample_wakeup_callback() in the code. It is called when the DPM state changes from sleep to wake-up.
- Additional user configuration can be stored in RTM.

In this sample, DTLS server information is stored.

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

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

	//Set DPM wakkup init callback
	user_config->wakeupInitCallback = dtls_server_dpm_sample_wakeup_callback;

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

	//Set session type(UDP Server)
	user_config->sessionConfig[session_idx].sessionType = REG_TYPE_UDP_SERVER;

	//Set local port
```

```
      user_config->sessionConfig[session_idx].sessionMyPort =
       DTLS_SERVER_DPM_SAMPLE_DEF_SERVER_PORT;

      //Set Connection callback
      user_config->sessionConfig[session_idx].sessionConnectCallback =
       dtls_server_dpm_sample_connect_callback;

      //Set Recv callback
      user_config->sessionConfig[session_idx].sessionRecvCallback =
       dtls_server_dpm_sample_recv_callback;

      //Set secure mode
      user_config->sessionConfig[session_idx].supportSecure = pdTRUE;

      //Set secure setup callback
      user_config->sessionConfig[session_idx].sessionSetupSecureCallback =
       dtls_server_dpm_sample_secure_callback;

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

      return ; }
```

12.7.3.2 DTLS Setup

To establish a DTLS session, DTLS should be set up. The DA16200 includes an `mbedTLS` library to provide the DTLS protocol. Most APIs that are related to the DTLS protocol are based on an `mbedTLS` library. DTLS is set up by `sessionSetupSecureCallback` function. The details are as follows.

```
void dtls_server_dpm_sample_secure_callback(void *config)
{
      const char *pers = "dtls_server_dpm_sample";
      SECURE_INFO_T *secure_config = (SECURE_INFO_T *)config;

      ret = mbedtls_ssl_config_defaults(secure_config->ssl_conf,
                                        MBEDTLS_SSL_IS_SERVER,
                                        MBEDTLS_SSL_TRANSPORT_DATAGRAM,
                                        MBEDTLS_SSL_PRESET_DEFAULT);

      //import test certificate
      ret = mbedtls_x509_crt_parse(secure_config->cert_crt,
                                   dtls_server_dpm_sample_cert,
                                   dtls_server_dpm_sample_cert_len);

      ret = mbedtls_pk_parse_key(secure_config->pkey_ctx,
                                 dtls_server_dpm_sample_key,
                                 dtls_server_dpm_sample_key_len,
                                 NULL, 0);

      if (mbedtls_pk_get_type(secure_config->pkey_ctx) == MBEDTLS_PK_RSA) {
           ret = mbedtls_pk_setup_rsa_alt(secure_config->pkey_alt_ctx,
                                   (void *)mbedtls_pk_rsa(*secure_config->pkey_ctx),
                                   dtls_server_dpm_sample_rsa_decrypt_func,
                                   dtls_server_dpm_sample_rsa_sign_func,
                                   dtls_server_dpm_sample_rsa_key_len_func);


           ret = mbedtls_ssl_conf_own_cert(secure_config->ssl_conf,
                                           secure_config->cert_crt,
```

```
                                              secure_config->pkey_alt_ctx);
      } else {
          ret = mbedtls_ssl_conf_own_cert(secure_config->ssl_conf,
                                          secure_config->cert_crt,
                                          secure_config->pkey_ctx);
      }

      ret = dpm_mng_setup_rng(secure_config->ssl_conf);
      ret = dpm_mng_cookie_setup_rng(secure_config->cookie_ctx);
      mbedtls_ssl_conf_dtls_cookies(secure_config->ssl_conf,
                                    mbedtls_ssl_cookie_write,
                                    mbedtls_ssl_cookie_check,
                                    secure_config->cookie_ctx);

      //Don't care verification in this sample.
      mbedtls_ssl_conf_authmode(secure_config->ssl_conf, MBEDTLS_SSL_VERIFY_NONE);

      //use default value
      mbedtls_ssl_conf_max_frag_len(secure_config->ssl_conf, 0);
      mbedtls_ssl_conf_dtls_anti_replay(secure_config->ssl_conf,
                                        MBEDTLS_SSL_ANTI_REPLAY_ENABLED);

      mbedtls_ssl_conf_read_timeout(secure_config->ssl_conf,
                                    DTLS_SERVER_DPM_SAMPLE_RECEIVE_TIMEOUT * 10);

      mbedtls_ssl_conf_handshake_timeout(secure_config->ssl_conf,
                              DTLS_SERVER_DPM_SAMPLE_HANDSAHKE_MIN_TIMEOUT * 10,
                              DTLS_SERVER_DPM_SAMPLE_HANDSAHKE_MAX_TIMEOUT * 10);

      ret = mbedtls_ssl_setup(secure_config->ssl_ctx, secure_config->ssl_conf);

      dpm_mng_job_done(); //Done opertaion
      return ;
}
```

12.7.3.3 Data Transmission

The callback function is called when a DTLS packet is received from the DTLS client. In this example, the received data is printed out and an echo message is sent to the DTLS server. Data is encrypted and decrypted in the callback function.

```
void dtls_server_dpm_sample_recv_callback(void *sock, UCHAR *rx_buf, UINT rx_len,
                                          ULONG rx_ip, ULONG rx_port)
{
      //Display received packet
      PRINTF(" ==> Received Packet(%ld) \n", 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);
```

```
    }

    dpm_mng_job_done(); //Done opertaion}
```

12.8 DTLS Client

This section describes how the DTLS client sample application is built and works. The DTLS client sample application is an example of the simplest DTLS echo client application. DTLS is a cryptographic protocol that provides security for datagram-based applications by allowing them to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. The DA16200 SDK provides an SSL library called `mbedTLS` on a secure hardware engine to support the DTLS protocol. `mbedTLS` is one of the popular SSL libraries. `mbedTLS` is helpful to easily develop a network application with the DTLS protocol.

12.8.1 How to Run

1. Run a DTLS server application on the peer computer and open a DTLS server socket with port number 10199.
2. In the e² studio, import a project for the DTLS client sample application.
   `~/SDK/apps/common/examples/Network/DTLS_Client/projects/da16200`
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. Use the console command to set up the Wi-Fi station interface.

After a connection is made to an AP, the sample application connects to the peer computer.

12.8.2 How It Works

The DA16200 DTLS Client sample is a simple echo message. When the DTLS server sends a message, then the DA16200 DTLS client echoes that message to the DTLS server.

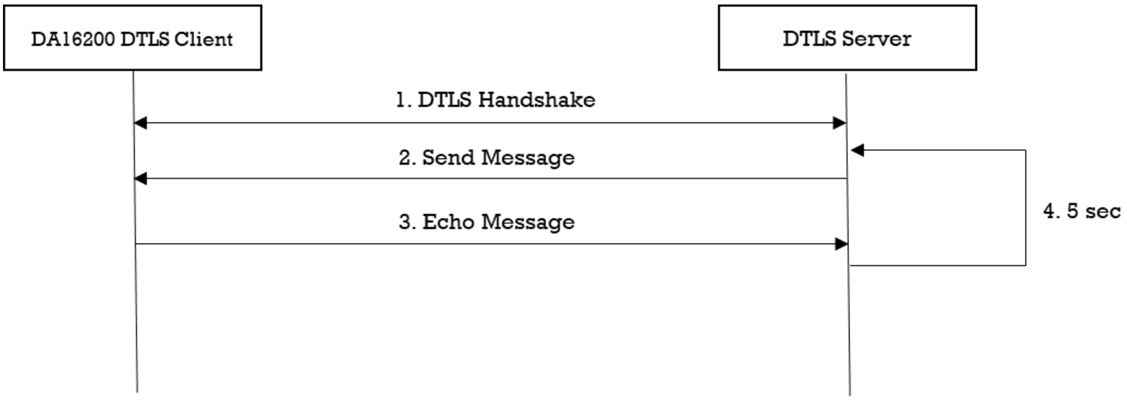


Figure 41. Workflow of DTLS client

12.8.3 Sample Code

The DA16200 SDK provides an `mbedTLS` library. This sample application describes how an `mbedTLS` library is called and applied for the socket library.

12.8.3.1 Initialization

The DA16200 secure hardware engine must be initialized with `da16x_secure_module_init()` before the DTLS context is initialized. To set up a DTLS session, initialization functions are called as shown in the example code below. A DTLS session is established over the UDP protocol. If a packet is lost, then retransmission is required. So, the timer is registered to retransmit the packet by function `mbedtls_ssl_set_timer_cb()`.

```
void dtls_client_sample(void *param)
{
    ...

    //Init session
```

```
    mbedtls_net_init(&server_ctx);

    //Init SSL context
    mbedtls_ssl_init(&ssl_ctx);

    //Init SSL config
    mbedtls_ssl_config_init(&ssl_conf);

    //Init CTR-DRBG context
    mbedtls_ctr_drbg_init(&ctr_drbg);

    //Init Entropy context
    mbedtls_entropy_init(&entropy);

    memset(&timer, 0x00, sizeof(dtls_client_sample_timer_t));

    PRINTF("\r\nConnecting to udp/%s:%d...",
           DTLS_CLIENT_SAMPLE_DEF_SERVER_IP_ADDR,
           DTLS_CLIENT_SAMPLE_DEF_SERVER_PORT);

    snprintf(str_port, sizeof(str_port),"%d", DTLS_CLIENT_SAMPLE_DEF_SERVER_PORT);

    ret = mbedtls_net_connect(&server_ctx,
                              DTLS_CLIENT_SAMPLE_DEF_SERVER_IP_ADDR, str_port,
                              MBEDTLS_NET_PROTO_UDP);

    ret = mbedtls_ssl_config_defaults(&ssl_conf,
                                      MBEDTLS_SSL_IS_CLIENT,
                                      MBEDTLS_SSL_TRANSPORT_DATAGRAM,
                                      MBEDTLS_SSL_PRESET_DEFAULT);

    ret = mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func, &entropy,
                                (const unsigned char *)pers, strlen(pers));

    mbedtls_ssl_conf_rng(&ssl_conf, mbedtls_ctr_drbg_random, &ctr_drbg);
    mbedtls_ssl_conf_authmode(&ssl_conf, MBEDTLS_SSL_VERIFY_NONE);
    mbedtls_ssl_conf_dtls_anti_replay(&ssl_conf, MBEDTLS_SSL_ANTI_REPLAY_ENABLED);
    mbedtls_ssl_conf_read_timeout(&ssl_conf, DTLS_CLIENT_SAMPLE_DEF_TIMEOUT);
    mbedtls_ssl_conf_handshake_timeout(&ssl_conf,
                                       DTLS_CLIENT_SAMPLE_DEF_HANDSHAKE_MIN_TIMEOUT,
                                       DTLS_CLIENT_SAMPLE_DEF_HANDSHAKE_MAX_TIMEOUT);

    ret = mbedtls_ssl_setup(&ssl_ctx, &ssl_conf);

    mbedtls_ssl_set_bio(&ssl_ctx, &server_ctx,
                        mbedtls_net_send, NULL, mbedtls_net_recv_timeout);

    mbedtls_ssl_set_timer_cb(&ssl_ctx, &timer,
                             dtls_client_sample_timer_start,
                             dtls_client_sample_timer_get_state);

    ...
}
```

12.8.3.2 DTLS Handshake

DTLS is an encryption protocol designed to secure network communication. A DTLS handshake is the process of initiating a communication session that uses DTLS encryption. To do a DTLS handshake, the function `mbedtls_ssl_handshake()` is called. If an error occurs during a DTLS handshake, the API returns the specific error code. If a DTLS session is established successfully, the API returns 0. The details are as follows:

```
void dtls_client_sample(void *param)
{
    ...
    while ((ret = mbedtls_ssl_handshake(&ssl_ctx)) != 0) {
        if (ret == MBEDTLS_ERR_NET_CONN_RESET) {
            PRINTF("\r\n[%s] Peer closed the connection(0x%x)\r\n", __func__, -ret);
            goto end_of_task;
        }

       if ((ret != MBEDTLS_ERR_SSL_WANT_READ) && (ret != MBEDTLS_ERR_SSL_WANT_WRITE))
       {
            PRINTF("\r\n[%s] Failed to do dtls handshake(0x%x)\r\n", __func__, -ret);
            goto end_of_task;
       }
    }
    ...
 }
```

12.8.3.3 Data Transmission

Encryption scrambles data so that only authorized parties can understand the information. While a DTLS session is established, all data must be encrypted to transfer application data. `mbedTLS` provides specific APIs to help encrypt and decrypt data. To write application data, call function `mbedtls_ssl_write()` of the `mbedTLS` library. The details are as follows:

```
void dtls_client_sample(void *param)
{
    ...
    do {
        ...
        while ((ret = mbedtls_ssl_write(&ssl_ctx, data_buffer, len)) <= 0) {
            switch (ret) {
                case MBEDTLS_ERR_SSL_WANT_READ:
                case MBEDTLS_ERR_SSL_WANT_WRITE:
                   PRINTF("\r\nNeed more data - mbedtls_ssl_write(0x%x)\r\n", -ret);
                   continue;
                case MBEDTLS_ERR_SSL_PEER_CLOSE_NOTIFY:
                   PRINTF("\r\nConnection was closed gracefully\r\n");
                   goto end_of_task;
                case MBEDTLS_ERR_NET_CONN_RESET:
                   PRINTF("\r\nConnection was reset by peer\r\n");
                   goto end_of_task;
                default:
                   PRINTF("\r\nFailed to write data(0x%x)\r\n", -ret);
                   break;
            }
            goto end_of_task;
        }
        PRINTF("%d bytes written\r\n", len);
    }
    ...
}
```

To read application data, the function `mbedtls_ssl_read()` of the `mbedTLS` library is called. In this sample, this function is called in `dtls_client_sample()`. The details are as follows:

```
void dtls_server_sample(void *param)
{
    ...
    do {
        len = sizeof(data_buffer) - 1;
        memset(data_buffer, 0x00, sizeof(data_buffer));
```

```
        PRINTF("< Read from server: ");

        ret = mbedtls_ssl_read(&ssl_ctx, data_buffer, len);
        if (ret <= 0) {
            switch (ret) {
                case MBEDTLS_ERR_SSL_WANT_READ:
                case MBEDTLS_ERR_SSL_WANT_WRITE:
                    PRINTF("\r\nNeed more data - mbedtls_ssl_read(0x%x)\r\n", -ret);
                    continue;
                case MBEDTLS_ERR_SSL_PEER_CLOSE_NOTIFY:
                    PRINTF("\r\nConnection was closed gracefully\r\n");
                    goto end_of_task;
                case MBEDTLS_ERR_NET_CONN_RESET:
                    PRINTF("\r\nConnection was reset by peer\r\n");
                    goto end_of_task;
                default:
                    PRINTF("\r\nFailed to read data(0x%x)\r\n", -ret);
                    break;
            }
            goto end_of_task;
        }

        len = ret;
        PRINTF("%d bytes read\r\n", len);

        PRINTF("> Write to server: ");
        while ((ret = mbedtls_ssl_write(&ssl_ctx, data_buffer, len)) <= 0) {
             ...
        }
    }
    ...
}
```

12.9 DTLS Client in DPM

This section describes how the DTLS client in the DPM sample application is built and works. The DTLS client in the DPM sample application is an example of the simplest DTLS echo client application in DPM mode. DTLS is a cryptographic protocol that provides security for datagram-based applications by allowing them to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. The DA16200 SDK can work in DPM mode. The user application requires an additional operation to work in DPM mode. The DA16200 SDK provides the DPM manager for the user network application. The DPM manager supports the user to develop and manage a DTLS network application in Non-DPM and DPM modes.

12.9.1 How to Run

1. Run a DTLS server application on the peer computer and open a DTLS server socket with port number 10199.
2. In the e^2 studio, import a project for the DTLS Client in the DPM sample application.
   `~/SDK/apps/common/examples/Network/DTLS_Client_DPM/projects/da16200`
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. Use the console command to set up the Wi-Fi station interface.
5. Set the DTLS server IP address and the port number as created the socket on the peer computer with the following console command and then reboot. These parameters can also be defined in the source code.

```
[/DA16200] # nvram.setenv DTLSC_SERVER_IP 192.168.0.11
[/DA16200] # nvram.setenv DTLSC_SERVER_PORT 10199
[/DA16200] # reboot
```

After a connection is made to an AP, the sample application connects to the peer computer.

12.9.2 How It Works

The DA16200 DTLS Client in the DPM sample is a simple echo message. When the DTLS server sends a message, then the DA16200 DTLS client echoes that message to the DTLS server.

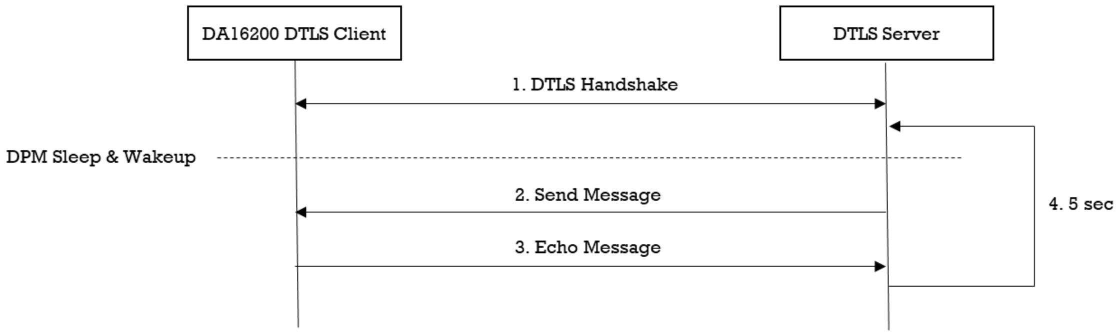


Figure 42. Workflow of DTLS client in DPM

12.9.3 Sample Code

12.9.3.1 Registration

The DTLS client in the DPM sample application works in DPM mode. The basic code is similar to the DTLS client sample application. There are two differences with the DTLS client sample application:

- An initial callback function is added, named dtls_client_dpm_sample_wakeup_callback() in the code. It is called when the DPM state changes from sleep to wake-up.
- Additional user configuration can be stored in RTM.

In this sample, DTLS server information is stored.

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

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

      //Set DPM wake up init callback
      user_config->wakeupInitCallback = dtls_client_dpm_sample_wakeup_callback;

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

      //Set session type(UDP Client)
      user_config->sessionConfig[session_idx].sessionType = REG_TYPE_UDP_CLIENT;

      //Set local port
      user_config->sessionConfig[session_idx].sessionMyPort =
       DTLS_CLIENT_DPM_SAMPLE_DEF_CLIENT_PORT;

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

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

      //Set Connection callback
      user_config->sessionConfig[session_idx].sessionConnectCallback =
       dtls_client_dpm_sample_connect_callback;
```

```
    //Set Recv callback
    user_config->sessionConfig[session_idx].sessionRecvCallback =
     dtls_client_dpm_sample_recv_callback;

    //Set secure mode
    user_config->sessionConfig[session_idx].supportSecure = pdTRUE;

    //Set secure setup callback
    user_config->sessionConfig[session_idx].sessionSetupSecureCallback =
     dtls_client_dpm_sample_secure_callback;

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

    return ;
}
```

12.9.3.2 DTLS Setup

To establish a DTLS session, DTLS should be set up. The DA16200 includes an `mbedTLS` library to provide the DTLS protocol. Most APIs that are related to the DTLS protocol are based on an `mbedTLS` library. DTLS is set up by function `session_setupSecureCallback()`. The details are as shown below. This sample application does not include certificates.

```
void dtls_client_dpm_sample_secure_callback(void *config)
{
    const char *pers = "dtls_client_dpm_sample";
    SECURE_INFO_T *secure_config = (SECURE_INFO_T *)config;

    ret = mbedtls_ssl_config_defaults(secure_config->ssl_conf,
                                      MBEDTLS_SSL_IS_CLIENT,
                                      MBEDTLS_SSL_TRANSPORT_DATAGRAM,
                                      MBEDTLS_SSL_PRESET_DEFAULT);

    ret = dpm_mng_setup_rng(secure_config->ssl_conf);

    //don't care verification in this sample.
    mbedtls_ssl_conf_authmode(secure_config->ssl_conf, MBEDTLS_SSL_VERIFY_NONE);

    //use default value
    mbedtls_ssl_conf_max_frag_len(secure_config->ssl_conf, 0);
    mbedtls_ssl_conf_dtls_anti_replay(secure_config->ssl_conf,
                                      MBEDTLS_SSL_ANTI_REPLAY_ENABLED);
    mbedtls_ssl_conf_read_timeout(secure_config->ssl_conf,
                                  DTLS_CLIENT_DPM_SAMPLE_RECEIVE_TIMEOUT * 10);

    mbedtls_ssl_conf_handshake_timeout(secure_config->ssl_conf,
                           DTLS_CLIENT_DPM_SAMPLE_HANDSAHKE_MIN_TIMEOUT * 10,
                           DTLS_CLIENT_DPM_SAMPLE_HANDSAHKE_MAX_TIMEOUT * 10);

    ret = mbedtls_ssl_setup(secure_config->ssl_ctx, secure_config->ssl_conf);

    dpm_mng_job_done(); //Done opertaion

    return ;
}
```

12.9.3.3 Data Transmission

The callback function is called when a DTLS packet is received from the DTLS server. In this sample, the received data is printed out and an echo message is sent to the DTLS server. Data is encrypted and decrypted in the callback function.

```
void dtls_client_dpm_sample_recv_callback(void *sock, UCHAR *rx_buf, UINT rx_len,
                                          ULONG rx_ip, ULONG rx_port)
{
      //Display received packet
      PRINTF(" ==> Received Packet(%ld) \n", rx_len);

      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);
      }

      dpm_mng_job_done(); //Done opertaion
}
```

13. Network Examples: MQTT

13.1 Overview

MQTT (Message Queue Telemetry Transport) is an ISO standard (ISO/IEC PRF 20922) publish-subscribe based messaging protocol. It works on top of the TCP/IP protocol. The publisher sends (PUBLISH) messages to the subscriber through the broker. The subscriber needs to keep the connection with the broker by TCP session while the publisher can disconnect the session with the broker after sending a message.

As shown in Figure 43, when the broker receives a message with a specific topic the message is sent to subscribers that already registered with the topic. A subscriber can register with more than one topic. There can be many or no subscribers which register with a specific topic.

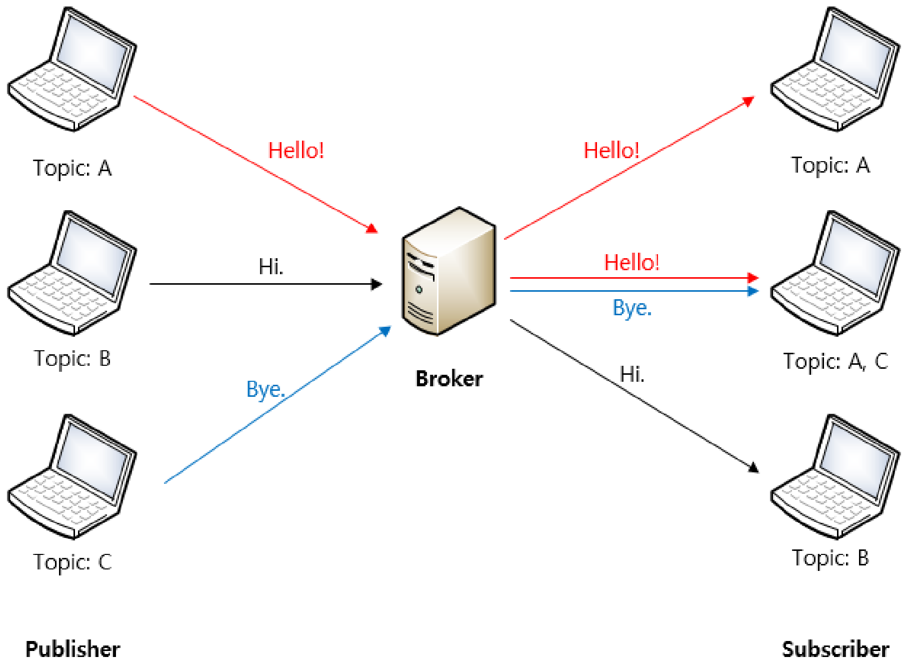


Figure 43. MQTT messaging concept

The exchange of MQTT messages supports QoS (Quality of Service). QoS has three levels (0, 1, and 2) and the process of each QoS level is described in the following sections.

The DA16200/DA16600 supports both publisher and subscriber functions and allows simultaneous use. The subscriber function supports DPM mode. TLS is available for message encryption.

13.1.1 SDK Build

Source files should be modified in the DA16200/DA16600 SDK to use the MQTT function. To enable the MQTT, modify it as shown in the following:

```
config_generic_sdk.h
...
#define        __SUPPORT_MQTT__           // Support MQTT
```

13.2 API

13.2.1 APIs for Operating MQTT

The APIs listed in Table 15 are used to create or terminate the MQTT thread, to check the status, and to publish a message. The configuration to execute MQTT protocols is explained in the next section.

Table 15. APIs for operating MQTT

Item		Description
int mqtt_client_start(void)		
Return		If it succeeds, return 0. If it fails, return an error code.
Description		Create the MQTT client thread.
int mqtt_client_stop(void)		
Return		If it succeeds, return 0. If there is no thread to terminate, return -1.
Description		Terminate the MQTT client thread.
int mqtt_client_check_conn(void)		
Return		1 (true): Connected to a broker. 0 (false): Not connected.
Description		Check whether the MQTT session is connected.
int mqtt_client_send_message(char *top, char *publish)		
Return		0: Succeeded in publishing. -1: Failed because MQTT is not connected. -2: Failed because the previous message sent is in progress. -3: Failed because MQTT topic is missing. Other: Failed due to other causes. See enum "mqtt_client_error_code" to identify the cause.
Parameter	top	Topic (if NULL, the MQTT publisher sends a PUBLISH message with the topic stored in NVRAM.)
	publish	Message to be published.
Description		Publisher sends an MQTT message (PUBLISH).
int mqtt_client_send_message_with_qos(char *top, char *publish, timeout)		
Return		0: Succeeded in publishing. -1: Failed to publish because Publisher is not ready to send. -2: Failed to publish because the timeout expired.
Parameter	top	Topic (if NULL, the MQTT module sends a PUBLISH message with the topic stored in NVRAM.)
	publish	Message to be published.
	timeout	Timeout to wait for a previous QoS=1/2 Message to process completely (unit: 10 ms).
Description		Publisher sends an MQTT message (PUBLISH) with a timeout check.
int mqtt_client_unsub_topic(char *topic)		
Parameter	topic	Topic to unsubscribe
Return		0: Succeeded in unsubscribing. 4: Failed because MQTT is not connected. 3: Failed because the topic is NULL 1: Failed because of memory allocation failure Other: Failed due to other causes. See enum "mqtt_client_error_code" to identify the cause.
Description		Unsubscribe from the specified topic. Invoke this function only when MQTT client is in a connected state with Broker.

13.2.2 APIs for Configure MQTT Messaging

With NVRAM items, you can configure MQTT messaging. This allows configuring the publisher and the subscriber.

Table 16. APIs for configuring MQTT message

Item		Description
int mqtt_client_config_initialize(void)		
Return		If it succeeds, return 0 (MOSQ_ERR_SUCCESS). If it fails, return an error code.
Description		Reset all MQTT configurations.
void mqtt_sub_callback_set(void (*user_cb)(void))		
Parameter	user_cb	User callback function to set.
Return		None
Description		Register a callback function that is invoked when a MQTT Subscriber is connected with a Broker.
void mqtt_pub_callback_set(void (*user_cb)(void))		
Parameter	user_cb	User callback function to set.
Return		None
Description		Register a callback function that is invoked when publishing a message is done.
void mqtt_msg_callback_set(void (*user_cb)(const char *buf, int len, const char *topic))		
Parameter	user_cb	User callback function to set. buf: PUBLISH message received len: PUBLISH message length topic: the topic of the PUBLISH message received
Return		None
Description		Register a callback function that is invoked when a PUBLISH message arrives.
void mqtt_sub_disconn_cb_set(void (*user_cb)(void));		
Parameter	user_cb	User callback function to set.
Return		None
Description		Register a callback function that is invoked when the MQTT client is disconnected.
void mqtt_sub_disconn2_cb_set(void (*user_cb)(void));		
Parameter	user_cb	User callback function to set.
Return		None
Description		▪ Register callback function called when MQTT Subscriber is disconnected by receiving a message with invalid unsupported length if the connection is clean_session=0 and qos > 0. ▪ On receipt of this callback, the application needs to clear the message in Broker by connecting with clean_sesion=1. ▪ To use this API, __MQTT_CLEAN_SESSION_MODE_SUPPORT__ should be enabled.
void mqtt_subscribe_callback_set(void (*user_cb)(void));		
Parameter	user_cb	User callback function to set.
Return		None
Description		Register a callback function that is invoked when a SUBSCRIBE request to a topic is finished.

Item		Description
void mqtt_unsubscribe_callback_set(void (*user_cb)(void));		
Parameter	user_cb	User callback function to set.
Return		None
Description		Register a callback function that is invoked when an UNSUBSCRIBE request is finished.

Table 17. MQTT messaging configuration (String type)

Name	Description	Example
DA16X_CONF_STR_MQTT_BROKER_IP	Broker IP address (or URI)	da16x_set_config_str(DA16X_CONF_STR_MQTT_BROKER_IP, "192.168.0.1");
DA16X_CONF_STR_MQTT_SUB_TOPIC	Subscriber topic (previous topics are removed) (Note 1)	da16x_set_config_str(DA16X_CONF_STR_MQTT_SUB_TOPIC, topic);
DA16X_CONF_STR_MQTT_SUB_TOPIC_ADD	Subscriber topic to add (up to four) (Note 1)	da16x_set_config_str(DA16X_CONF_STR_MQTT_SUB_TOPIC_ADD, topic);
DA16X_CONF_STR_MQTT_SUB_TOPIC_DEL	Subscriber topic to remove (Note 1)	da16x_set_config_str(DA16X_CONF_STR_MQTT_SUB_TOPIC_DEL, topic);
DA16X_CONF_STR_MQTT_PUB_TOPIC	Topic to publish	da16x_set_config_str(DA16X_CONF_STR_MQTT_PUB_PUB_TOPIC, "pub_topic");
DA16X_CONF_STR_MQTT_USERNAME	Username to log in to a broker	da16x_set_config_str(DA16X_CONF_STR_MQTT_USERNAME, "mqtt_id");
DA16X_CONF_STR_MQTT_PASSWORD	Password to login to a broker	da16x_set_config_str(DA16X_CONF_STR_MQTT_PASSWORD, "mqtt_password");
DA16X_CONF_STR_MQTT_WILL_TOPIC	Will Topic	da16x_set_config_str(DA16X_CONF_STR_MQTT_WILL_TOPIC, "will_topic");
DA16X_CONF_STR_MQTT_WILL_MSG	Will Message	da16x_set_config_str(DA16X_CONF_STR_MQTT_WILL_MSG, "will_msg");
DA16X_CONF_STR_MQTT_SUB_CLIENT_ID	MQTT client ID	da16x_set_config_str(DA16X_CONF_STR_MQTT_SUB_CLIENT_ID, "sub_id");
DA16X_CONF_STR_MQTT_TLS_SNI	MQTT TLS SNI (Server Name Indication)	da16x_set_config_str(DA16X_CONF_STR_MQTT_TLS_SNI, "sni_str")

Note 1 Up to four subscriber topics can be registered, and only one publisher topic can be registered.

Table 18. MQTT messaging configuration (Integer type)

Name	Description	Example
DA16X_CONF_INT_MQTT_SUB	MQTT operation (0: stop, 1: start)	da16x_set_config_int(DA16X_CONF_INT_MQTT_SUB, 1);
DA16X_CONF_INT_MQTT_AUTO	MQTT Auto-start at booting system (0: disable, 1: enable)	da16x_set_config_int(DA16X_CONF_INT_MQTT_AUTO, 1);
DA16X_CONF_INT_MQTT_PORT	Broker port number	da16x_set_config_int(DA16X_CONF_INT_MQTT_PORT, 8883);
DA16X_CONF_INT_MQTT_QOS	QoS level (0~2)	da16x_set_config_int(DA16X_CONF_INT_MQTT_QOS, 2);
DA16X_CONF_INT_MQTT_TLS	TLS (0: disable, 1: enable)	da16x_set_config_int(DA16X_CONF_INT_MQTT_TLS, 1);
DA16X_CONF_INT_MQTT_WILL_QOS	QoS level of Will messages (0~2)	da16x_set_config_int(DA16X_CONF_INT_MQTT_WILL_QOS, 1);
DA16X_CONF_INT_MQTT_PING_PERIOD	MQTT ping period (secs)	da16x_set_config_int(DA16X_CONF_INT_MQTT_PING_PERIOD, 86400);

Name	Description	Example
DA16X_CONF_INT_MQTT_VER311	MQTT protocol: 1 (v3.1.1)/0 (v3.1)	da16x_set_config_int(DA16X_CONF_INT_MQTT_VER311, 1)
DA16X_CONF_INT_MQTT_TLS_INCOMING	TLS incoming buffer size: default (1024*4), min (1024*2), max (1024*8)	da16x_set_config_int(DA16X_CONF_INT_MQTT_TLS_INCOMING, 1024*4)
DA16X_CONF_INT_MQTT_TLS_OUTGOING	TLS outgoing buffer size: default (1024*4), min (1024*2), max (1024*8)	da16x_set_config_int(DA16X_CONF_INT_MQTT_TLS_OUTGOING, 1024*4)
DA16X_CONF_INT_MQTT_TLS_AUTHMODE	TLS peer certificate verification mode: 0 (not verify), 1 (optional), 2 (required), default is 1	da16x_set_config_int(DA16X_CONF_INT_MQTT_TLS_AUTHMODE, 1)
DA16X_CONF_INT_MQTT_CLEAN_SESSION	MQTT Clean Session mode (1: clean the previous session, 0: do not clean the previous session)	da16x_set_config_int(DA16X_CONF_INT_MQTT_CLEAN_SESSION, 1);

13.3 MQTT Connection and Flow Chart

Table 19 shows that MQTT client is started/not started depending on the configuration and use cases.

Table 19. MQTT client start conditions

Configuration	Use case	Result
Automatic connection enabled. (`DA16X_CONF_INT_MQTT_AUTO: 1`)	The DA16200 boots from POR or software reset (Non-wake-up case)	MQTT client is started.
	MQTT connection is established, and then the DA16200 wakes up from DPM LPM.	MQTT client is started.
	MQTT client is stopped, and then the DA16200 wakes up from DPM LPM.	MQTT client is not started.
Automatic connection disabled. (`DA16X_CONF_INT_MQTT_AUTO: 0`)	The DA16200 boots from POR or software reset.	MQTT client is not started.
	MQTT connection is established, and then the DA16200 wakes up from DPM LPM.	MQTT client is started.
	MQTT client is stopped, and then the DA16200 wakes up from DPM LPM.	MQTT client is not started.

Figure 44 describes how the MQTT client in the DA16200/DA16600 tries to make MQTT connection. For example, MQTT client is started if `DA16X_CONF_INT_MQTT_AUTO` is enabled and the DA16200 boots from POR or software reset. If connection is established, then the DA16200 enters DPM LPM or stay awake depending on DPM configuration. If MQTT connection is failed, it retries the connection up to the values defined in the `MQTT_RESTART_MAX_RETRY` and `MQTT_CONN_MAX_RETRY`.

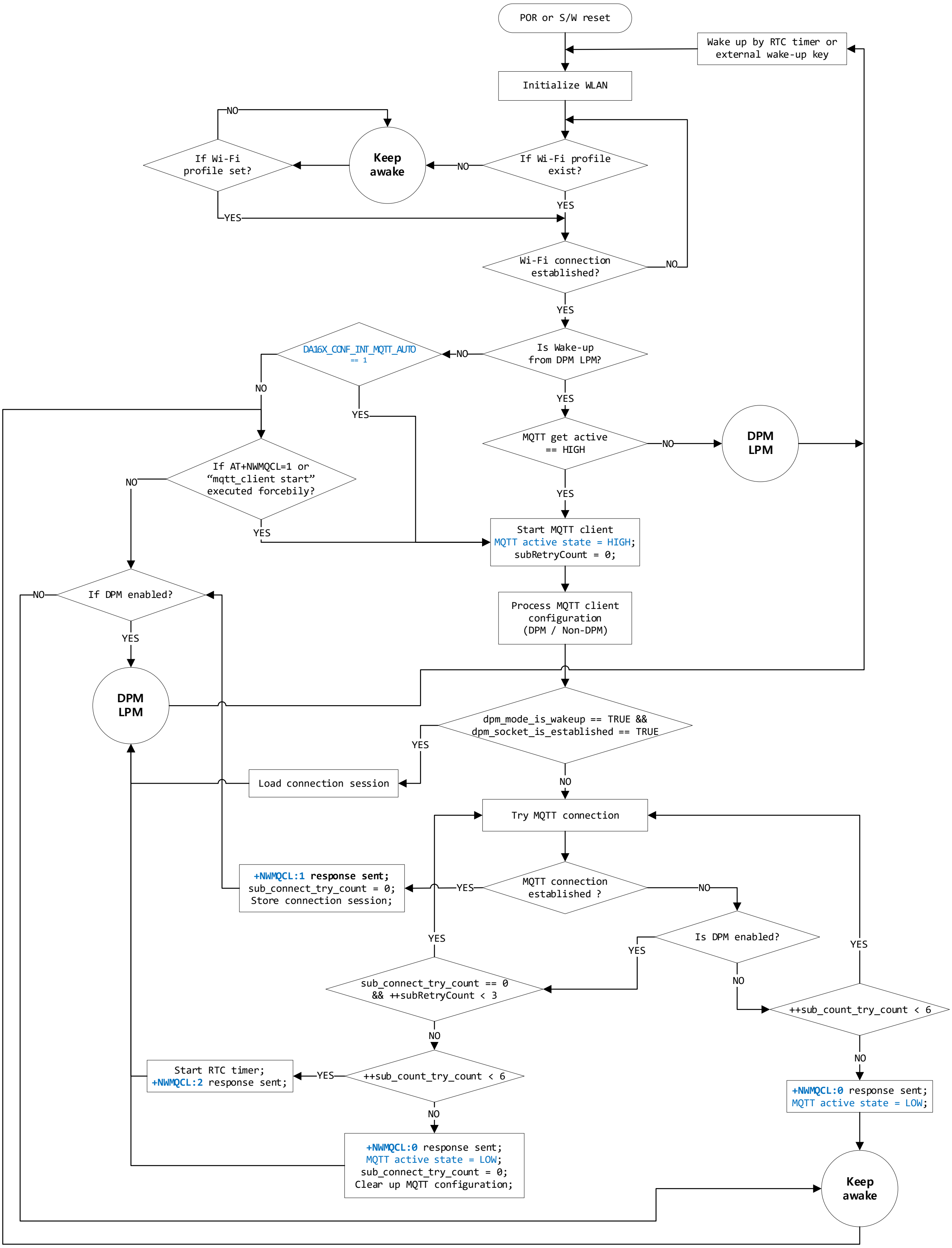


Figure 44. MQTT client flow chart

13.4 Test

This section explains how to test the MQTT function on the DA1620/DA16600 debug console window.

13.4.1 Test Environment

For this test the Mosquitto MQTT broker is used, which can be downloaded from the following URL: https://mosquitto.org/download/. If the broker cannot be installed, use the one provided by Renesas Electronics. Extract and run it on local Windows computer.

13.4.2 Setup

Open a command window and go to the Mosquitto folder.

1. Run a broker.

```
mosquitto -v -p <Port Number>
```

```
C:₩mosquitto>mosquitto -v -p 1884
1582173416: mosquitto version 1.4.14 (build date 11/07/2017  0:03:18.53) starting
1582173416: Using default config.
1582173416: Opening ipv6 listen socket on port 1884.
1582173416: Opening ipv4 listen socket on port 1884.
```

2. Open a new command window and run a subscriber.

```
mosquitto_sub -h <Broker IP> -p <Port Number> -t <Topic>
```

```
C:₩mosquitto>mosquitto_sub -h 172.16.30.163 -p 1884 -t da16k
```

The following message is shown in the broker window.

```
C:₩mosquitto>mosquitto -v -p 1884
1582173276: mosquitto version 1.4.14 (build date 11/07/2017  0:03:18.53) starting
1582173276: Using default config.
1582173276: Opening ipv6 listen socket on port 1884.
1582173276: Opening ipv4 listen socket on port 1884.
1582173309: New connection from 172.16.30.163 on port 1884.
1582173309: New client connected from 172.16.30.163 as mosqsub|13800-KR-ENG-LT (c1, k60).
1582173309: Sending CONNACK to mosqsub|13800-KR-ENG-LT (0, 0)
1582173309: Received SUBSCRIBE from mosqsub|13800-KR-ENG-LT
1582173309:     da16k (QoS 0)
1582173309: mosqsub|13800-KR-ENG-LT 0 da16k
1582173309: Sending SUBACK to mosqsub|13800-KR-ENG-LT
```

3. Open a new command window and publish a message.

```
mosquitto_pub -h <Broker IP> -p <Port Number> -t <Topic> -m "<Message>"
```

```
C:₩mosquitto>mosquitto_pub -h 172.16.30.163 -p 1884 -t da16k -m "Hello World!"
```

The following message is shown in the broker window.

```
582173567: New connection from 172.16.30.163 on port 1884.
582173567: New client connected from 172.16.30.163 as mosqpub|3508-KR-ENG-LT- (c1, k60).
582173567: Sending CONNACK to mosqpub|3508-KR-ENG-LT- (0, 0)
582173567: Received PUBLISH from mosqpub|3508-KR-ENG-LT- (d0, q0, r0, m0, 'da16k', ... (12 bytes))
582173567: Sending PUBLISH to mosqsub|17076-KR-ENG-LT (d0, q0, r0, m0, 'da16k', ... (12 bytes))
582173567: Received DISCONNECT from mosqpub|3508-KR-ENG-LT-
582173567: Client mosqpub|3508-KR-ENG-LT- disconnected.
```

The subscriber receives the message as shown below.

```
C:₩mosquitto>mosquitto_sub -h 172.16.30.163 -p 1884 -t da16k
Hello World!
```

13.4.3 Certificate

See Section 6 for MQTT client certificate.

13.4.4 Publisher

13.4.4.1 QoS=0 Message

This section gives an example of publishing a QoS=0 message.

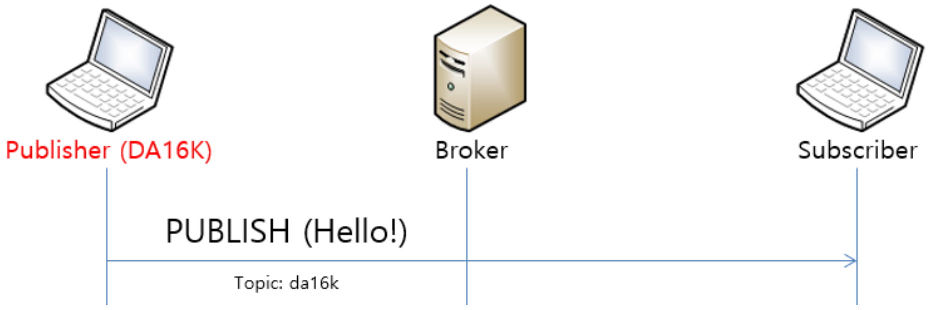


Figure 45. Publish QoS=0 message

1. After the DA16200/DA16600 EVB is connected to an AP, configure the parameters, and publish a message.

```
[/DA16200]# net
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config pub_topic <Topic>
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)

[/DA16200/NET]# mqtt_client -m <Message>
…
[/DA16200/NET]# mqtt_client stop
```

Optionally, "client_id" can also be set with the following command:

`[/DA16200/NET]# mqtt_config client_id <client_id_string>`

For example, mqtt_config client_id abcd1111

`client_id` should be unique per each device. By default, `client_id` is generated internally as "da16x_<the last 2 bytes of mac address>". For example, da16x_FCFA.

2. When message transmission -m "Hello!" is successful, the following messages are displayed:

`Hello! (Send, Len: 6, Topic: da16k, Message ID: 1)`

The following syntax allows you to send a message with a new topic:

```
[/DA16200/NET] mqtt_client -m <Message> <NewTopic>
```

If the previous parameters for broker, port, and topics are not changed, then no action is required to set the parameters for the publication of every message.

The max length of the console command is 158. To send a longer PUBLISH, write the following command:

```
[/DA16200/NET] mqtt_client -l
Typing data: (MQTT Publisher message)
        Cancel - CTRL+D, End of Input - CTRL+C or CTRL+Z
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
3456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234
56789012345678901234567890123456789012345678901234567890 …
```

Use the keyboard combinations Ctrl+C or Ctrl+Z to send the message.

Optionally, "retain" can also be set with the following command:

```
[/DA16200/NET]# mqtt_config pub_topic <Topic> -r
```

For example, `mqtt_config pub_topic abdc111 -r`

MQTT client is a feature used to ensure that the last published message on a specific topic is stored by the broker. When a client subscribes to that topic, it immediately receives the retained message, without waiting for a new one to be published.

13.4.4.2 QoS=1/2 Message

This section gives an example of publishing a QoS=1/2 Message.

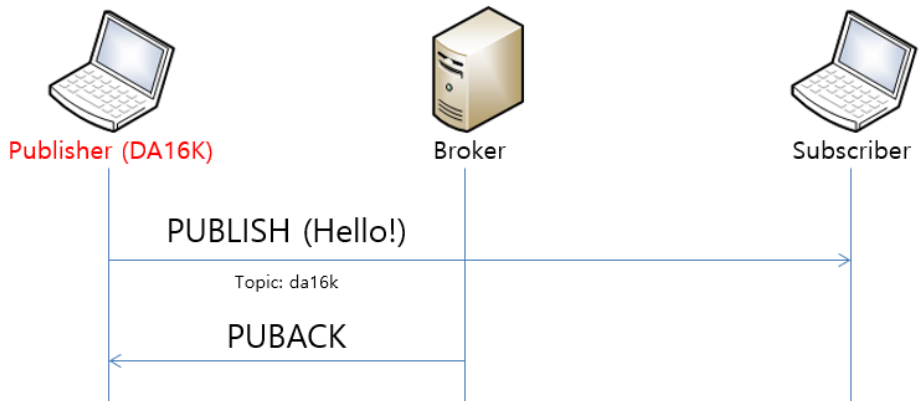


Figure 46. Publish QoS 1 message

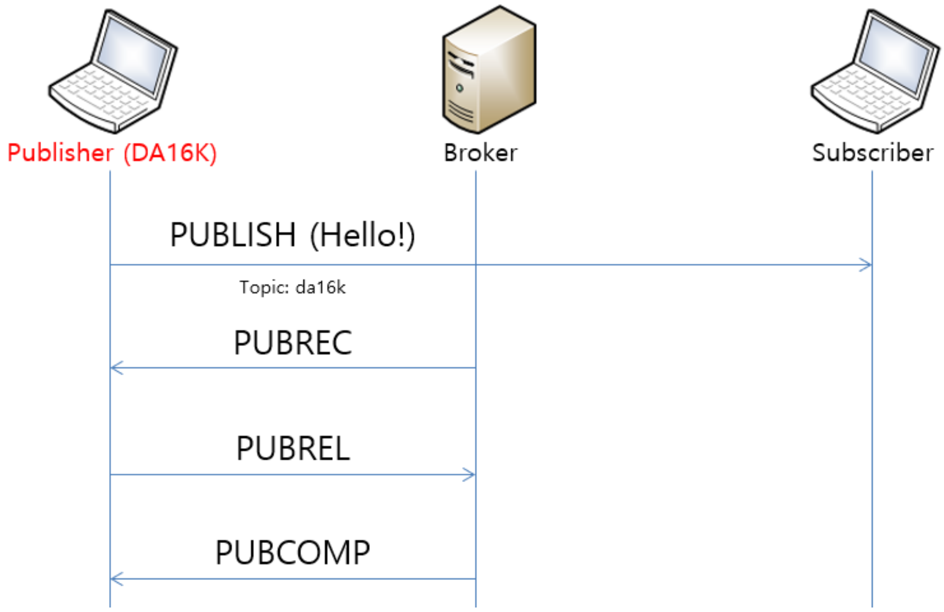


Figure 47. Publish QoS 2 message

- Configure the parameters and publish a message.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config pub_topic <Topic>
[/DA16200/NET]# mqtt_config qos <QoS Level>
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)

[/DA16200/NET]# mqtt_client -m <Message>
```

```
1582175804: Received PUBLISH from PUB-da16x_FD26 (d0, q1, r0, m1, 'da16k', ... (6 bytes))
1582175804: Sending PUBACK to PUB-da16x_FD26 (Mid: 1)
```

```
1582175859: Received PUBLISH from PUB-da16x_FD26 (d0, q2, r0, m1, 'da16k', ... (6 bytes))
1582175859: Sending PUBREC to PUB-da16x_FD26 (Mid: 1)
1582175859: Received PUBREL from PUB-da16x_FD26 (Mid: 1)
1582175859: Sending PUBCOMP to PUB-da16x_FD26 (Mid: 1)
```

Figure 48. Configure parameters and publish message

13.4.4.3 MQTT over TLS

The DA16200/DA16600 SDK provides a TLS encrypted session for secure MQTT messages.

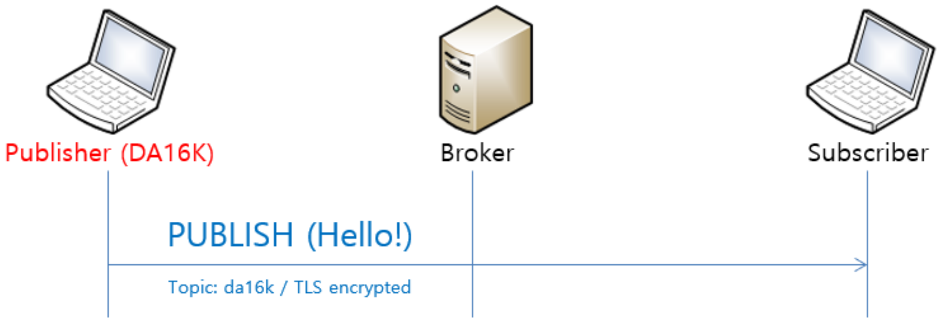


Figure 49. Publish secure message

NOTE
It is required to store certificates in the DA1620/DA16600 to use TLS encryption. This procedure is explained in Section 13.4.3.

1. Run a broker with a secure port.

```
mosquitto -c mosquitto.conf -p <Port Number> -v
```

```
C:₩mosquitto>mosquitto -c mosquitto.conf -p 8883 -v
1582174980: mosquitto version 1.4.14 (build date 11/07/2017  0:03:18.53) starting
1582174980: Config loaded from mosquitto.conf.
1582174980: Opening ipv6 listen socket on port 8883.
1582174980: Opening ipv4 listen socket on port 8883.
```

2. Run a subscriber.

```
mosquitto_sub -h <Broker IP> -p <Port> --cafile <CA Certificate> --cert <Client Certificate> --key
<Client Private Key> --tls-version <TLS Protocol Version> --insecure -t <Topic>
```

```
C:₩mosquitto>mosquitto_sub -h 172.16.30.163 -p 8883 --cafile cas.pem --cert wifiuser.pem --key wifiuser.key --tls-ver
sion tlsv1 --insecure -t da16k
```

3. Set the current time in the DA16200/DA16600 EVB to check if the certificate is valid.
(If SNTP for time sync is needed, input the command `net.sntp enable` to get the current time.)

```
[/DA16200]# time set <yyyy-mm-dd> <hh:mm:ss>
```

4. Store three Certificates (see Section 6.6) in the DUT and then complete the following steps.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config pub_topic <Topic>
[/DA16200/NET]# mqtt_config tls 1
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)

[/DA16200/NET]# mqtt_client -m <Message>
```

13.4.4.4 Username and Password

1. Set up a username and password to authenticate users.

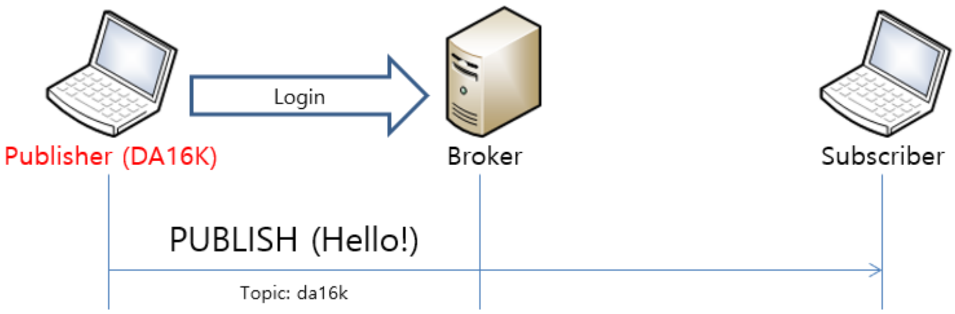


Figure 50. User login

2. Run a broker with a secure port. It needs to be prepared for the configuration file.

```
mosquitto -c <Config File> -p <Port> -v
```

In the Mosquitto package provided by Renesas Electronics, file `mosq_idpw.conf` is used for the <Config File> parameter, and user accounts are registered in file p1.txt.

3. Add a new account in this file with the following command:

```
mosquitto_passwd.exe -b p1.txt <username> <password>
```

4. At the Mosquitto command prompt, run the `mosquito_sub` command to log in successfully to the broker.

```
mosquitto_sub -h <broker_ip> -p <port> -t <topic> -u <id> -P <pass>
```

5. On `mqtt_client` (DUT), set the username and password, and start `mqtt_client`.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config pub_topic <Topic>
[/DA16200/NET]# mqtt_config tls 0
[/DA16200/NET]# mqtt_config username <Username>
[/DA16200/NET]# mqtt_config password <Password>
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)

[/DA16200/NET]# mqtt_client -m <Message>
```

NOTE
▪ The max length of the console command is 158 so to type in a password exceeding the limit of the console, use the command `mqtt_config long_password`. ▪ The max length of the buffer is currently 160 for a password, 64 for a username. If it needs to change max length, modify `MQTT_USERNAME_MAX_LEN` or `MQTT_PASSWORD_MAX_LEN`.

13.4.5 Subscriber

13.4.5.1 Setup

1. Configure the parameters and start the subscriber.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
```

```
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config sub_topic 1 <Topic>
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)
…
[/DA16200/NET]# mqtt_client stop
```

2. Multiple topics can be registered. Add the parameter for the number of topics in the command (up to four).

```
[/DA16200/NET]# mqtt_client stop
[/DA16200/NET]# mqtt_config sub_topic <Topic count> <Topic#1> <Topic#2> …
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)
…
[/DA16200/NET]# mqtt_config sub_topic_add <New topic>
[/DA16200/NET]# mqtt_config sub_topic_del <Topic to remove>
```

13.4.5.2 MQTT over TLS

Set the current time in the DA16200/DA16600 EVB to check if the certificate is valid. (If SNTP is auto-started during boot, skip this step.)

```
[/DA16200]# time set <yyyy-mm-dd> <hh:mm:ss>
```

1. Run the broker as below.

```
mosquitto -c mosquitto.conf -p <Port Number> -v
```

2. Add three Certificates (see Section 6.6) for the DUT and then complete the following steps.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config sub_topic 1 <Topic>
[/DA16200/NET]# mqtt_config tls 1
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)
```

3. Run a publisher on your computer.

```
mosquitto_pub -h <Broker IP> -p <Port> --cafile <CA Certificate> --cert <Client Certificate> --key
<Client Private Key> --tls-version <TLS Protocol Version> -t <Topic> --insecure -m <message>

Example: mosquitto_pub -h 192.168.0.101 -p 1884 --cafile cas.pem --cert wifiuser.pem --key
wifiuser.key --tls-version tlsv1 -t da16k --insecure -m "hello"
```

13.4.5.3 Username and Password

1. DUT: Set username and password.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config sub_topic 1 <Topic>
[/DA16200/NET]# mqtt_config tls 0
[/DA16200/NET]# mqtt_config username <Username>
[/DA16200/NET]# mqtt_config password <Password>
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)
```

2. In the Mosquitto package provided by Renesas Electronics, file `mosq_idpw.conf` is used for the <Config File> parameter and user accounts are registered in file p1.txt. Add a new account in this file with the following command.

```
mosquitto_pub -h [Broker IP] -p [port] -t [topic] -m <message> -u [id] -P [password]
Example:
mosquitto_pub -h 192.168.0.101 -p 1884 -t da16k -u mike -P 1234 -m hello
```

13.4.5.4 WILL

1. Sub#1 (DUT): Set the Will message.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config sub_topic 1 <Topic>
[/DA16200/NET]# mqtt_config will_topic <Topic>
[/DA16200/NET]# mqtt_config will_message <Message>
[/DA16200/NET]# mqtt_config will_qos <QoS Level>
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)
```

2. Broker: Write the following command.

```
>mosquitto -v -p 1884
```

3. Sub#2 (PC): Write the following command.

```
>mosquitto_sub -h 192.168.0.101 -t da16k -p 1884 -q 0
```

4. Sub#1 (DUT): Try an unexpected disconnection.

```
[/DA16200/NET]# reset

>>> Network Interface (wlan0): DOWN
 [mqtt_subscriber_main] Request mqtt_restart
[wpa_supplicant_event_disassoc] CTRL-EVENT-DISCONNECTED bssid=ec:08:6b:d6:53:62 reason=3
locally_generated=1

DA16200 ROM-Boot [ffffc000]
[MROM]
```

5. Sub#2 (PC): Wait until the following message is printed.

```
>mosquitto_sub -h 192.168.0.101 -t da16k -p 1884 -q 2
imwill
```

13.4.6 MQTT Pub/Sub Test with DPM and TLS

In this test, the Pub and Sub are run with the DPM mode enabled.

1. Broker: Run with TLS enabled.

```
>mosquitto -c mosquitto.conf -p 8883 -v
```

2. Sub#2 (PC): Write the following command.

```
>mosquitto_sub -h 192.168.0.101 -p 8883 --cafile cas.pem --cert wifiuser.pem --key wifiuser.key --tls-
version tlsv1 -t da16k --insecure
```

3. Sub-Pub#1 (DUT): Write the following command.

```
[/DA16200/NET]# mqtt_config auto 1
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config sub_topic 1 <Topic>
[/DA16200/NET]# mqtt_config pub_topic <Topic>
[/DA16200/NET]# mqtt_config tls 1
[/DA16200/NET]# sntp enable
[/DA16200/NET]# nvram.setenv dpm_mode 1
[/DA16200/NET]# reboot
```

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

-- DHCP Client WLAN0: SEL(3)
-- DHCP Client WLAN0: REQ(4)
-- DHCP Client WLAN0: BOUND(5)
        Assigned addr   : 192.168.0.20
              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      : 18h 00m 00s
        Renewal Time    : 15h 00m 00s


>>> SNTP Server: pool.ntp.org (106.247.248.106)
>>> SNTP Time sync : 2021.03.11 - 01:29:17
>>> MQTT Client connection OK (da16x_DD12)

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

Figure 51. DPM sleep after MQTT connection

```
#Pub (PC): Send the Pub message as below.
>mosquitto_pub -h 192.168.0.101 -p 1884 --cafile cas.pem --cert wifiuser.pem --key wifiuser.key --tls-version tlsv1 -t da16k --insecure -m "Hello World!!"
When the message is received, DA16200/DA16600 wakes up from DPM Sleep and prints the message.
```

```
Wakeup source is 0x82

>>> TIM STATUS: 0x00000001
>>> TIM : UC
Waking up MCU ...
(Rx: Len=8,Topic=SUB_TOPIC,Msg_ID=0)

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

Figure 52. MQTT UC wake-up

```
If the code examples are applied, the MQTT publisher starts to post a periodic message every 30
seconds and the MQTT subscriber processes the received PUBLISH messages.
```

```
Wakeup source is 0x82


(5)rtc_timeout
>>> TIM STATUS: 0x00000010
>>> TIM : FAST
(Tx: Len=26,Topic=PUB_TOPIC,Msg_ID=6)
<< Mqtt Pub EnQ : SUCCESS >>
[PUBREC] (Rx: Msg_ID=6)
[PUBREL] (Tx: Msg_ID=6)
[PUBCOMP] (Rx, Msg_ID=6)

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

Figure 53. MQTT wake-up for sending message

13.4.6.1 MQTT Reconnection Scheme

When the broker is disconnected, MQTT Client tries to reconnect to the broker based on the following scheme.

13.4.6.1.1 Non-DPM Mode

MQTT Client tries to reconnect six times (MQTT_CONN_MAX_RETRY) and the attempt to retry is terminated after the max number of trials is reached.

13.4.6.1.2 DPM Mode

1. After the disconnection from the broker is recognized, the system wakes up from DPM Sleep, and MQTT Client tries to reconnect three times (MQTT_RESTART_MAX_RETRY), and the system enters DPM Sleep when the trials fail.

2. In five seconds, the system wakes up and MQTT Client tries reconnection with the broker. If it fails in connecting to the broker, the system enters DPM Sleep.
3. **Step 2** is repeated six times (MQTT_CONN_MAX_RETRY) and MQTT Client is terminated after the max number of trials (MQTT_CONN_MAX_RETRY) is reached. The system then enters DPM Sleep.
4. In case other DPM wake-up (User Wake-up, user RTC Wake-up, UC...) happens after Step 3, **Step 2** is repeated six times.

13.4.6.2 DPM Power Profile

With Keysight, a current consumption measuring tester, check how DPM works in MQTT communication. DPM allows the system to stay in Sleep mode most of the time and only wake up (and stay active for only a small amount of time to get the job done) when needed.

In the Keysight snapshot below, the DA16200/DA16600 was in Sleep mode until it woke up to post a periodic status message to the broker. When the DA16200/DA16600 receives the response, it enters and stays in Sleep mode until the next Status Message TX time (the interval depends on application).

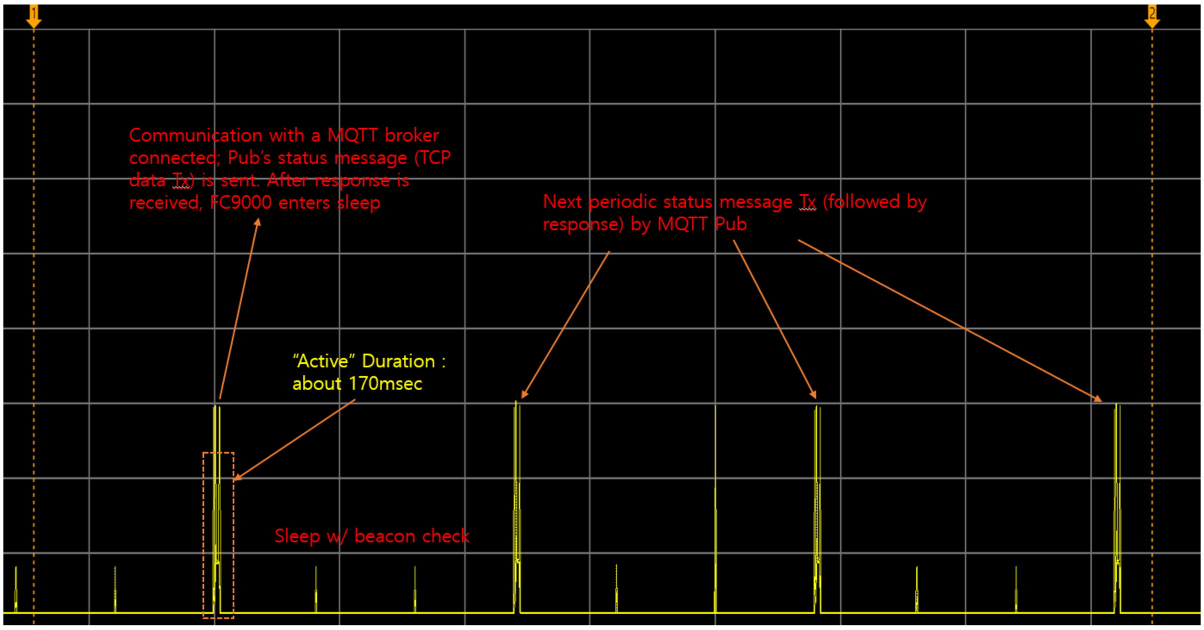


Figure 54. MQTT communication

13.4.7 MQTT CleanSession=0 Test Guide

13.4.7.1 CleanSession=0 Mode

When an MQTT Client (hereinafter referred to as MQTTC) establishes a connection with an MQTT Broker (Broker onward), there are two types of session: CleanSession=1 and CleanSession=0.

CleanSession=1: default session type. When the Broker receives a connect request from an MQTTC that tries to connect with an option "CleanSession=1" (which is default config on DA16x), Broker treats the connection as a "new" session. If an existing session associated with the same client_id is found, the Broker clears that previous session and creates a new one with the client_id.

CleanSession=0: When the Broker receives a connect request from an MQTTC that tries to connect with an option "CleanSession=0", the Broker first tries to find a session (session data) with the same client_id. If it finds one, it keeps using that session for the new MQTTC.

While MQTTC is connected to the Broker, there may be times when the TCP connection becomes unstable and disconnected (for example, mqtt ping failed). This may cause some messages that had been published to the Broker during that time to not be delivered to a subscriber. If new messages (with QoS > 0) are published to the Broker and for sessions that have been configured in "CleanSession=0", the Broker retains and re-sends them when the MQTTC is re-connected. MQTTC (if CleanSession=0 is enabled) also should retain the state of the unfinished/unacked messages until reconnection.

```
1647307743: New connection from 192.168.0.2 on port 8883.
1647307743: New client connected from 192.168.0.2 as da16x_D9CC (c1, k60).
1647307743: Sending CONNACK to da16x_D9CC (0, 0)
1647307743: Received SUBSCRIBE from da16x_D9CC
1647307743:     SUB_TOPIC (QoS 2)
1647307743: da16x_D9CC 2 SUB_TOPIC
1647307743: Sending SUBACK to da16x_D9CC
```

Figure 55. Broker console - CleanSession=1 connection

```
1647307894: Client da16x_D9CC disconnected.
1647307898: New connection from 192.168.0.2 on port 8883.
1647307898: New client connected from 192.168.0.2 as da16x_D9CC (c0, k60).
1647307898: Sending CONNACK to da16x_D9CC (0, 0)
1647307898: Received SUBSCRIBE from da16x_D9CC
1647307898:     SUB_TOPIC (QoS 2)
1647307898: da16x_D9CC 2 SUB_TOPIC
1647307898: Sending SUBACK to da16x_D9CC
```

Figure 56. Broker console - CleanSession=0 connection

Even with CleanSession=0 connection, the Broker does not maintain session data if MQTTC is disconnected in the following cases.

- If a new message is published with QoS 0 after MQTTC is disconnected.
- If MQTTC's connection QoS is 0.

The DA16200 and DA16600 support CleanSession=0 mode in the following method.

CleanSession=0 feature is enabled by default in SDK v3.2.3.0 or higher (__MQTT_CLEAN_SESSION_MODE_SUPPORT__)

If an application uses **QoS 1 or Qos 2 and CleanSession=0**, the message (payload) size (both Tx and Rx) should be pre-decided (because there is limitation in the dpm user pool size). By default, 100 bytes are defined.

`#define MQTT_MSG_TBL_PRESVD_MAX_PLAYLOAD_LEN 100`

- Depending on the application's expected maximum payload size, a different value can be defined.
- The DPM User Pool has a limited size (approximately 8K in total) in the system.
- First check the available **free** DPM User Pool size using the console command `dpm user_pool` and then calculate the max payload length and message number for the application if needed.
- The default configuration (payload_len: 100, max_count: 10) allocates approximately 1.9 kB of DPM user pool (Check mq_msg_tbl_presvd_t for detail).
- Search for the following compiler options in config_generic_sdk.h.

  //max payload length of a preserved message

  #define MQTT_MSG_TBL_PRESVD_MAX_PLAYLOAD_LEN 100

  // max number of preserved messages

  #define MQTT_MSG_TBL_PRESVD_MAX_MSG_CNT 10
- The following console command is provided to configure CleanSession mode:

  mqtt_client clean_session <1|0>

13.4.7.1.1 CleanSession and QoS Matrix Table for PUBLISH Rx

Table 20. CleanSession and QoS matrix in message Rx

Subscriber			Unacked message delivery (After MQTT reconnection)	QoS (Effective actual)	Publisher Message's QoS
Case	Clean session	QoS			
1	1	0	X	0	0

Subscriber			Unacked message delivery (After MQTT reconnection)	QoS (Effective actual)	Publisher Message's QoS
Case	Clean session	QoS			
2	1	1	X	0	0
3	1	2	X	0	0
4	1	0	X	0	1
5	1	1	X	1	1
6	1	2	X	1	1
7	1	0	X	0	2
8	1	1	X	1	2
9	1	2	X	2	2
10	0	0	X	0	0
11	0	1	X	0	0
12	0	2	X	0	0
13	0	0	X	0	1
14	0	1	O	1	1
15	0	2	O	1	1
16	0	0	X	0	2
17	0	1	O	1	2
18	0	2	O	2	2

Basically, with CleanSession=1, no unacked message delivery happens when a MQTT reconnect happens (marked as x).

With CleanSession=0, only case 14, 15, 17, and 18 makes message redelivery happen for messages that had been delivered to the Broker while the MQTT was offline (marked as O).

13.4.7.1.2 CleanSession and QoS Matrix Table for PUBLISH Tx

Expectation 1	Application assumes "sending message" fails and waits until MQTT gets re-connected before retrying.
Behavior 1	Application does message send retry.
Expectation 2	Application assumes "sending message" resumes when MQTT gets re-connected.
Behavior 2	Application waits until the message send retry by MQTT is complete.

Table 21. CleanSession and QoS matrix in message TX

Publisher			Expectation if MQTT gets disconnected (while QoS 1/2 message is not fully acked or QoS 0 send is being sent)	Behavior expected when MQTT client re-connected
Case	Clean session	QoS		
1	1	0	Expectation 1	Behavior 1
2	1	1	Expectation 1	Behavior 1
3	1	2	Expectation 1	Behavior 1
4	0	0	Expectation 1	Behavior 1
5	0	1	Expectation 2	Behavior 2
6	0	2	Expectation 2	Behavior 2

When publishing a message from DA16x, the application’s expectation and action/behavior may be different if CleanSession=0 and QoS 1 or 2 are used in some specific cases.

In normal network conditions, there is no difference in message send behavior between CleanSession=0 and CleanSession=1.

In some abnormal cases where QoS 1/2's ACK message (PUBACK, PUBREC, PUBREL, or PUBCOMP) gets lost due to bad network conditions (which can cause a MQTTC re-connection), CleanSession=0 can recover the previous message state and resume communication with the Broker.

However, if CleanSession=1 is used, when MQTTC is disconnected, it can safely re-transmit the message when MQTTC is re-connected. Depending on the use case, either approach (CleanSession=0 or CleanSession=1) can be utilized.

13.4.7.2 Test Steps

13.4.7.2.1 How to connect with CleanSession=0

```
[/DA16200/NET] # mqtt_client stop

[/DA16200/NET] # mqtt_config clean_session 0

[/DA16200/NET] # mqtt_config qos 2

[/DA16200/NET] # mqtt_config status

MQTT Client Information:
  - MQTT Status        : Not Running
  - Broker IP          : 192.168.0.230
  - Port               : 8883
  - Pub. Topic         : PUB_TOPIC
  - Sub. Topic         : SUB_TOPIC
  - QoS Level          : 2
  - TLS                : Enable
  - Clean Session      : No
  - TLS ALPN           : (None)
  - TLS SNI            : (None)
  - TLS CIPHER SUIT    : (None)
  - Ping Period        : 60
  - TLS Incoming buf   : 4096(bytes)
  - TLS Outgoing buf   : 4096(bytes)
  - TLS Auth mode      : 1
  - User name          : (None)
  - Password           : (None)
  - Client ID          : (default: da16x_D9CC)
  - MQTT VER           : 3.1
[/DA16200/NET] # mqtt_client start

MQTT CleanSession=0 Support Mode enabled.
[/DA16200/NET] # >>> MQTT Client connection OK (da16x_D9CC)
```

To activate "**CleanSession=0 support mode**" in DA16x, QoS should be 1 or 2 and CleanSession option should be set to 0.

If either option (CleanSession and QoS) is not set as above, CleanSession=0 support mode is disabled.

13.4.7.2.2 How to restart CleanSession=0 test

If re-testing (fresh new test) with CleanSession=0 mode, the Broker may be needed to "clear the previous session" depending on the previous session type.

The reason is that since an MQTTC connects with CleanSession=0, the Broker does not delete the session data until the MQTTC re-connects with CleanSession=1.

Case 1: Previous session is CleanSession=1 and restart a new CleanSession=0 test.

```
[/DA16200/NET] # mqtt_client stop
```

```
 [mqtt_subscriber_main] mosquitto_loop_forever exited (rc=17, sock=0, errno=0,
runContinueSub=0)
[mqtt_client] terminated
[/DA16200/NET] # mqtt_config clean_session 0

[/DA16200/NET] # mqtt_client start

MQTT CleanSession=0 Support Mode enabled.
[/DA16200/NET] # >>> MQTT Client connection OK (da16x_D9CC)
```

Case 2: Previous session is CleanSession=0 and re-test of CleanSession=0.

```
[/DA16200/NET] # mqtt_client stop

 [mqtt_subscriber_main] mosquitto_loop_forever exited (rc=17, sock=0, errno=0,
runContinueSub=0)
[mqtt_client] terminated
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] # mqtt_config clean_session 1

[/DA16200/NET] # mqtt_client start

[/DA16200/NET] # >>> MQTT Client connection OK (da16x_D9CC)

[/DA16200/NET] #
[/DA16200/NET] # mqtt_client stop

 [mqtt_subscriber_main] mosquitto_loop_forever exited (rc=17, sock=0, errno=0,
runContinueSub=0)
[mqtt_client] terminated
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] # mqtt_config clean_session 0

[/DA16200/NET] # mqtt_client start

MQTT CleanSession=0 Support Mode enabled.
[/DA16200/NET] # >>> MQTT Client connection OK (da16x_D9CC)
```

13.4.7.2.3 PUBLISH RX Test

1) Test Steps

Test steps are as follows under non-DPM and DPM modes.

In non-DPM mode:

- DA16x: connect to Broker
- Publisher: send one or two messages
- DA16x: check if the messages are received.
- DA16x: disconnect from Broker
- Publisher: send one or two messages (let say msg_A)
- DA16x: reconnect to Broker
- DA16x: check if msg_A (sent while DA16x is offline) is received.

DPM mode:

- DA16x: connect to Broker. Enter DPM sleep
- Publisher: send one or two messages

- DA16x: check if the messages are received
- DA16x: turn off AP. Wait for the MQTT keep alive period to finish (to make sure Broker recognizes the MQTTC disconnection)
- Publisher: send one or two messages (let say msg_A)
- DA16x: turn on AP. Wait until DA16x is connected to AP
- DA16x: reconnected to AP and check if msg_A (sent while DA16x is offline) is received.

NOTE
▪ Mosquitto broker (Broker), Mosquitto publisher (Publisher), and DA16x (Subscriber) are used for the test. ▪ Message length from publisher should be less than or equal to 100. If longer messages are sent, they may not be restored properly when MQTT is reconnected.

2) Test Steps - Example 1 (non-DPM)

The following are the test steps for case 15 (non-DPM mode).

[DA16x] Connect MQTTC with CleanSession=0 and QoS 2

```
[/DA16200/NET] # mqtt_client stop

 [mqtt_subscriber_main] mosquitto_loop_forever exited (rc=17, sock=0, errno=0, 
runContinueSub=0)
[mqtt_client] terminated
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] # mqtt_config qos 2

[/DA16200/NET] # mqtt_config clean_session 0

[/DA16200/NET] # mqtt_client start

MQTT CleanSession=0 Support Mode enabled.
[/DA16200/NET] # >>> MQTT Client connection OK (da16x_D9CC)
```

[Other Publisher] Publish messages

```
C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert wifiuser.pem --
key wifiuser.key --tls-version tlsv1 --insecure -q 1 -t SUB_TOPIC -m "hello_qos_0"

C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert wifiuser.pem --
key wifiuser.key --tls-version tlsv1 --insecure -q 1 -t SUB_TOPIC -m "hello_qos_1"
```

[DA16x] Check the messages are successfully received

```
[/DA16200/NET] #
[/DA16200/NET] # (Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=1)
[PUBACK] (Tx: Msg_ID=1)
(Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=2)
[PUBACK] (Tx: Msg_ID=2)
```

[DA16x] Disconnect from Broker

```
[/DA16200/NET] #
[/DA16200/NET] # mqtt_client stop

 [mqtt_subscriber_main] mosquitto_loop_forever exited (rc=17, sock=0, errno=0, 
runContinueSub=0)
[mqtt_client] terminated
```

[Other Publisher] Publish two messages (while DA16x is in disconnected state)

```
C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert wifiuser.pem --key wifiuser.key --tls-version tlsv1 --insecure -q 1 -t SUB_TOPIC -m "hello_qos_2"

C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert wifiuser.pem --key wifiuser.key --tls-version tlsv1 --insecure -q 1 -t SUB_TOPIC -m "hello_qos_3"
```

[DA16x] Re-connect to the Broker and check if the two messages that had been published while DA16x was in a disconnected state are received successfully.

```
[/DA16200/NET] #
[/DA16200/NET] # mqtt_client start

MQTT CleanSession=0 Support Mode enabled.
[/DA16200/NET] # (Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=3)
[PUBACK] (Tx: Msg_ID=3)
(Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=4)
[PUBACK] (Tx: Msg_ID=4)
>>> MQTT Client connection OK (da16x_D9CC)
```

3) Test Steps - Example 2 (DPM)

The following are the test steps for case 18 (DPM mode). Mosquitto broker and Mosquitto publisher are used for the test.

[DA16x] Connect with CleanSession=0 and QoS 2

```
[/DA16200/NET] # mqtt_config qos 2

[/DA16200/NET] # mqtt_config clean_session 0

[/DA16200/NET] # mqtt_config status

MQTT Client Information:
  - MQTT Status  : Not Running
  - Broker IP          : 192.168.0.230
  - Port               : 8883
  - Pub. Topic         : PUB_TOPIC
  - Sub. Topic         : SUB_TOPIC
  - QoS Level          : 2
  - TLS                : Enable
  - Clean Session      : No
  - TLS ALPN           : (None)
  - TLS SNI            : (None)
  - TLS CIPHER SUIT    : (None)
  - Ping Period        : 60
  - TLS Incoming buf   : 4096(bytes)
  - TLS Outgoing buf   : 4096(bytes)
  - TLS Auth mode      : 1
  - User name          : (None)
  - Password           : (None)
  - Client ID          : (default: da16x_D9CC)
  - MQTT VER           : 3.1
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] # dpm on
[DP
Wake-up source is 0x0
[dpm_init_retmemory] DPM INIT CONFIGURATION(1)

...

System Mode : Station Only (0)
>>> Start DA16X Supplicant ...
```

```
>>> DA16x Supp Ver2.7 - 2020_07
>>> Wi-Fi mode : b/g/n -> b/g (for DPM)
>>> MAC address (sta0) : d4:3d:39:10:d9:cc
...
Connection COMPLETE to 00:11:32:ce:8e:6f

-- DHCP Client WLAN0: SEL(6)
-- DHCP Client WLAN0: REQ(1)
-- DHCP Client WLAN0: CHK(8)
-- DHCP Client WLAN0: BOUND(10)
         Assigned addr   : 192.168.1.195
               netmask   : 255.255.255.0
               gateway   : 192.168.1.1
               DNS addr  : 192.168.1.1

         DHCP Server IP  : 192.168.1.1
         Lease Time      : 24h 00m 00s
         Renewal Time    : 20h 00m 00s

MQTT CleanSession=0 Support Mode enabled.
>>> Hello World #2 ( network dependent application ) !!!
>>> MQTT Client connection OK (da16x_D9CC)
>>> Start DPM Power-Down !!!
```

[Other Publisher] Publish messages

```
C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert wifiuser.pem --
key wifiuser.key --tls-version tlsv1 --insecure -q 2 -t SUB_TOPIC -m "hello_qos_1"

C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert wifiuser.pem --
key wifiuser.key --tls-version tlsv1 --insecure -q 2 -t SUB_TOPIC -m "hello_qos_2"
```

[DA16x] Check the messages are successfully received

```
Wake-up source is 0x82

>>> Start DA16X Supplicant ...
>>> TIM STATUS: 0x00000001
>>> TIM : UC
>>> Hello World #1 ( Non network dependent application ) !!!
  MQTT CleanSession=0 Support Mode enabled.
>>> Hello World #2 ( network dependent application ) !!!
(Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=1)
[PUBREC] (Tx: Msg_ID=1)
[PUBREL] (Rx: Msg_ID=1)
[PUBCOMP] (Tx: Msg_ID=1)
>>> Start DPM Power-Down !!!
[i3ed11_dpm_tcp_ack_proc] TCP Update SEQ Ný
Wake-up source is 0x82

>>> Start DA16X Supplicant ...
>>> TIM STATUS: 0x00000001
>>> TIM : UC
>>> Hello World #1 ( Non network dependent application ) !!!
  MQTT CleanSession=0 Support Mode enabled.
>>> Hello World #2 ( network dependent application ) !!!
(Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=2)
[PUBREC] (Tx: Msg_ID=2)
[PUBREL] (Rx: Msg_ID=2)
[PUBCOMP] (Tx: Msg_ID=2)
>>> Start DPM Power-Down !!!
[i3ed11_dpm_tcp_ack_proc] TCP Update SEQ Num(20d7)
```

```
PS TIME 130369 us
```

[DA16x] Turn off AP

```
Wake-up source is 0x82

>>> Start DA16X Supplicant ...
>>> TIM STATUS: 0x00000008
>>> TIM : No BCN

>>> Network Interface (wlan0) : DOWN
[wpa_supplicant_event_disassoc] CTRL-EVENT-DISCONNECTED bssid=00:11:32:ce:8e:6f reason=4
locally_generated=1
Fast scan, freq=2432, num_ssids=1
!!! No selected network !!!
Fast scan, freq=2432, num_ssids=1
>>> Hello World #1 ( Non network dependent application ) !!!
!!! No selected network !!!

### User Call-back : Wi-Fi disconnected ( reason_code = 4 ) ...
Fast scan, freq=2432, num_ssids=1
!!! No selected network !!!
!!! No selected network !!!
!!! No selected network !!!
!!! No selected network !!!
!!! No selected network !!!

rtc_timeout (tid:14)
!!! No selected network !!!
[dpm_timer_process] 'mqtt_sub' is not ready. Callback can't be called. (/14)
>> Abnormal DPM(1) operation after 1 second
...
```

[Broker] Make sure MQTTC is disconnected

```
...
1647318510: Socket error on client da16x_D9CC, disconnecting.
...
```

[Other Publisher] Publish two messages (while DA16x is in a disconnected state)

```
C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert wifiuser.pem --
key wifiuser.key --tls-version tlsv1 --insecure -q 2 -t SUB_TOPIC -m "hello_qos_3"

C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert wifiuser.pem --
key wifiuser.key --tls-version tlsv1 --insecure -q 2 -t SUB_TOPIC -m "hello_qos_4"
```

[DA16x] Turn ON AP

[DA16x] Wait until AP is connected and see whether `hello_qos_3` and `hello_qos_4` are received

```
...
Wake-up source is 0x82


System Mode : Station Only (0)
>>> Start DA16X Supplicant ...
>>> DA16x Supp Ver2.7 - 2020_07
>>> Wi-Fi mode : b/g/n -> b/g (for DPM)
>>> MAC address (sta0) : d4:3d:39:10:d9:cc
>>> sta0 interface add OK
>>> Start STA mode...
>>> Hello World #1 ( Non network dependent application ) !!!
```

```
>>> Network Interface (wlan0) : UP
>>> Associated with 00:11:32:ce:8e:6f

Connection COMPLETE to 00:11:32:ce:8e:6f

-- DHCP Client WLAN0: SEL(6)
-- DHCP Client WLAN0: REQ(1)
-- DHCP Client WLAN0: CHK(8)
-- DHCP Client WLAN0: BOUND(10)
         Assigned addr   : 192.168.1.195
               netmask   : 255.255.255.0
               gateway   : 192.168.1.1
               DNS addr  : 192.168.1.1

         DHCP Server IP  : 192.168.1.1
         Lease Time      : 24h 00m 00s
         Renewal Time    : 20h 00m 00s

  MQTT CleanSession=0 Support Mode enabled.
>>> Hello World #2 ( network dependent application ) !!!
(Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=3)
[PUBREC] (Tx: Msg_ID=3)
(Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=4)
[PUBREC] (Tx: Msg_ID=4)
>>> MQTT Client connection OK (da16x_D9CC)
[PUBREL] (Rx: Msg_ID=3)
[PUBCOMP] (Tx: Msg_ID=3)
[PUBREL] (Rx: Msg_ID=4)
[PUBCOMP] (Tx: Msg_ID=4)
>>> Start DPM Power-Down !!!
```

13.4.7.2.4 PUBLISH Tx Test

1) Test Steps

Test steps are as follows.

- DA16x: connect to Broker
- DA16x: send messages
- DA16x: check if the message sent is successful.

NOTE
Message length from DA16x should be less than or equal to 100 for case 5 and 6 configuration. Sending longer messages returns failure. For cases other than case 5 or 6, message length limit is 3K.

2) Test Steps – Example

The following are the test steps for case 5 (non-DPM mode).

```
[/DA16200/NET] # mqtt_config qos 1

[/DA16200/NET] # mqtt_config clean_session 0

[/DA16200/NET] # mqtt_client start

  MQTT CleanSession=0 Support Mode enabled.
[/DA16200/NET] # user cb: on_connect
user cb: on_subscribe
>>> MQTT Client connection OK (da16x_D9CC)

[/DA16200/NET] #
[/DA16200/NET] # mqtt_client -m hello_q1
```

```
[/DA16200/NET] # (Tx: Len=8,Topic=PUB_TOPIC,Msg_ID=2)
<< Mqtt Pub EnQ : SUCCESS >>
[PUBACK] (Rx, Msg_ID=2)
user cb: on_publish(mid=2)
```

13.4.8 Reset

The following command clears all MQTT configurations:

```
[/DA16200/NET]# mqtt_config reset
```

13.5 Sample Code

This section explains how to test the MQTT client sample application on the DA16200/DA16600 EVB. This section describes how to configure and run MQTT client, and how to send or receive a message using DA16x MQTT APIs.

NOTE
This sample version is available in DA16200/DA16600 SDK v3.2.5.0 or higher.

13.5.1 Test Environment

Users can use an MQTT broker that is compliant to MQTT Spec 3.1 or 3.1.1 but, for this test, Mosquitto broker, Mosquitto subscriber, and Mosquitto publisher are used, which can be download from the following URL: https://mosquitto.org/files.

The DA16200/DA16600 contains the MQTT client module (hereinafter referred to as **mqtt_client**) that can work with an MQTT broker.

13.5.2 Setup

MQTT Broker (hereinafter referred to as **mqtt_broker**)

- Open a command prompt and go to the Mosquitto folder
- Run the Mosquitto broker (mqtt_broker) with TLS configured
- Any MQTT broker can be used but, the Mosquitto broker is used for this test
  - The following config options are used in the .conf file for the sample (depending on the local environment, other options can be modified. For detail explanation for each option, check default .conf file included in the Mosquitto package:

    >> bind_address, cafile <ca_file>, certfile <cert_file>, keyfile <key_file>, require_certificate yes

    >> If TLS is not used, comment out the following options: cafile, certfile, keyfile, and require_certificate as default

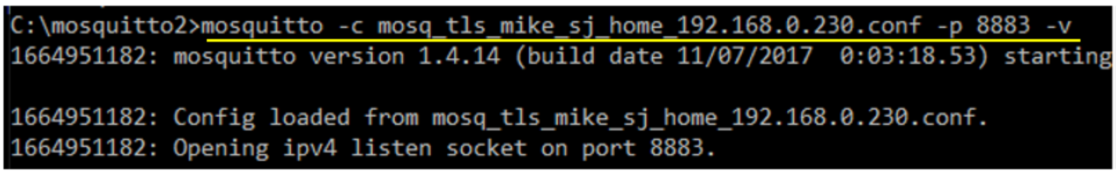


Figure 57. Mosquitto MQTT broker

MQTT subscriber (hereinafter referred to as **mqtt_sub**)

- Open a new command prompt and run the Mosquitto subscriber (mqtt_sub) with topic **_da16k.**
- mqtt_sub should be connected to mqtt_broker.

```
C:\mosquitto2>mosquitto_sub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert wifiu
ser.pem --key wifiuser.key --tls-version tlsv1 --insecure -t _da16k -q 0
```

Figure 58. Mosquitto MQTT subscriber

MQTT publisher (hereinafter referred to as **mqtt_pub**)

- Open a new command prompt and run the Mosquitto publisher (mqtt_pub) with topic **_da16k.**
- Make sure that the mqtt_sub receives the published message sent by mqtt_pub.

```
C:\mosquitto2>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert wifiuser.pem
--key wifiuser.key --tls-version tlsv1 --insecure -q 0 -t da16k2 -m "hello2"
```

Figure 59. Mosquitto MQTT publisher

MQTT Client sample application **(**mqtt_app).

The mqtt_app acts as MQTT publisher and MQTT subscriber.

13.5.3 How to Test

1. In the e^2 studio environment, import a project for the MQTT client sample application as follows:

```
~/SDK/apps/common/examples/Network/MQTT_Client/projects/da16200
~/SDK/apps/common/examples/Network/MQTT_Client/projects/da16600
```

2. Modify the MQTT broker address, port number, TLS, and cert info for the local environment in the source.

```
#define MQTT_SAMPLE_BROKER_IP          "192.168.0.230"
#define MQTT_SAMPLE_BROKER_PORT        8883
#define MQTT_SAMPLE_TLS                1
...
```

   If TLS is enabled, generate certificate sets. Same certificate sets should be set on the broker side as well to get TLS communication working.

```
static const char *cert_buffer0 = ...
static const char *cert_buffer1 = ...
static const char *cert_buffer2 = ...
...
```

3. (Optional) There are two APIs for sending a message; mqtt_client_send_message() and mqtt_client_send_message_with_qos(). By default, mqtt_client_send_message() is used. If the other APIs should be used, enable USE_MQTT_SEND_WITH_QOS_API in mqtt_client_sample.c.
4. Build the DA16200 SDK (do not download it to the DA16200 EVB yet).
5. DA16200 EVB:
   a. At the [/DA16200] prompt, type factory to do a factory reset (see Ref. [3])
   b. Reboot DA16200 EVB.
6. Download the build image to DA16200 EVB and reboot.
7. DA16200 EVB:
   a. Run setup (to connect to a Wi-Fi router) : see the Station Mode Setup section of Ref. [3]
      i. Setup as STA.
      ii. SNTP Client enable?: select Yes using the default setting (the Internet should be accessible through a Wi-Fi router).
      iii. Dialog DPM (Dynamic Power Management)?: select No.
8. Reboot DA16200 EVB and connect to a Wi-Fi router, and the **mqtt_app** starts running.

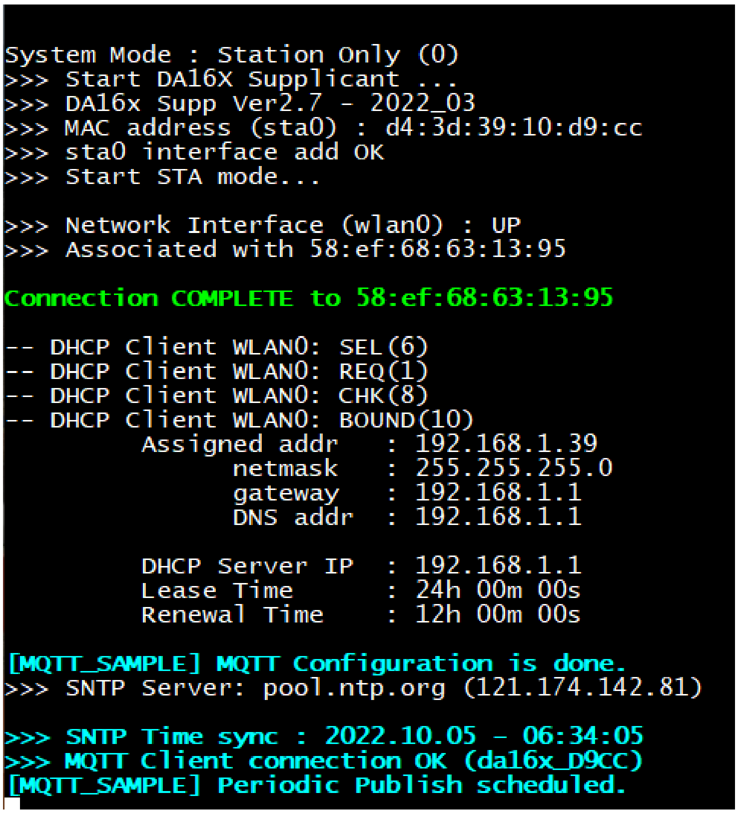


Figure 60. MQTT client is ready

13.5.3.1 Test with Non-DPM Mode

1. When the mqtt_app starts for the first time, it configures MQTT client. When it is configured, find MQTT configuration parameters in NVRAM. See my_app_mqtt_user_config() for more information on configuration.
2. Next the mqtt_app waits for the system to sync the system time with SNTP server which is required for successful TLS session.
3. Then, the mqtt_app starts the mqtt_client.
4. After checking the successful connection of mqtt_client with mqtt_broker, the mqtt_app initializes the app resource within my_app_init() and enters the main loop to handle various events. See EVT_ANY.

13.5.3.1.1 MQTT Publish

1. The mqtt_app starts 30-second timer in my_app_init(). Every 30 seconds, the mqtt_app tries to send a periodic message. _my_app_mqtt_pub_send_periodic() is the timer callback that triggers the MQTT publish.
2. The mqtt_sub displays the published message from the mqtt_app.

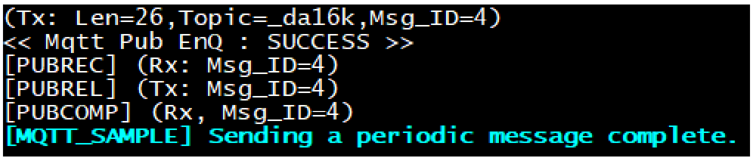


Figure 61. MQTT publish

13.5.3.1.2 Receive MQTT Message

1. mqtt_pub: publish a message "hello" to the topic **da16k1** and try to publish it to other topics as well. The mqtt_app has subscribed to 3 topics: da16k1, da16k2, and da16k3.
2. mqtt_app: receive the message "hello". See the message callback "my_app_mqtt_msg_cb()"

Figure 62. Receive MQTT message

13.5.3.1.3 Receive and Reply MQTT Message

1. mqtt_pub: publish a message **reply_needed** to the topic **da16k1**.
2. mqtt_app: receive the message and try to publish a message **DA16K status: Not bad ( )** to the topic **_da16k**. The message callback "my_app_mqtt_msg_cb()" upon receipt of **reply_needed**, tries to publish a message to the topic **_da16k**.
3. mqtt_sub (subscribed to _da16k): display the message received.

```
(Rx: Len=12,Topic=da16k1,Msg_ID=0)
[MQTT_SAMPLE] Msg Recv: Topic=da16k1, Msg=reply_needed
(Tx: Len=26,Topic=_da16k,Msg_ID=10)
<< Mqtt Pub EnQ : SUCCESS >>
[PUBREC] (Rx: Msg_ID=10)
[PUBREL] (Tx: Msg_ID=10)
[PUBCOMP] (Rx, Msg_ID=10)
[MQTT_SAMPLE] Sending a reply message complete.
```

Figure 63. Receive and reply MQTT message

13.5.3.1.4 MQTT Unsubscribe

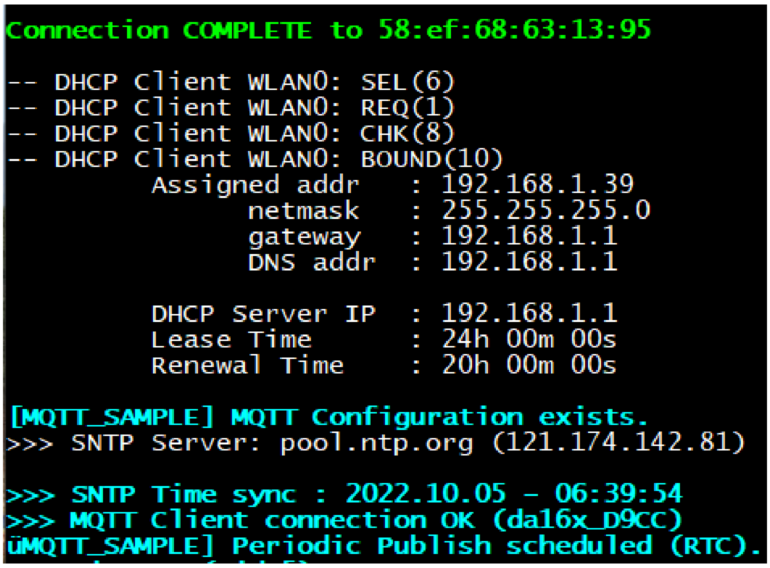
1. mqtt_pub: publish a message “unsub:da16k2” to the topic **da16k1**.
2. mqtt_app: receive the message and try to unsubscribe one of subscribed topics. The message callback “my_app_mqtt_msg_cb()” upon receipt of “unsub:da16k2,” tries to unsubscribe da16k2.
3. Mqtt_pub: try publishing a message to the **da16k2**. Make sure that the mqtt_app does not receive the message.

```
(Rx: Len=12,Topic=da16k1,Msg_ID=0)
[MQTT_SAMPLE] Msg Recv: Topic=da16k1, Msg=unsub:da16k2
[MQTT_SAMPLE] Topic to unsub = da16k2
[MQTT_SAMPLE] Unsubscribe complete.
```

Figure 64. MQTT unsubscribe

13.5.3.2 Test with DPM Mode

1. DA16200 EVB: enter the command **dpm on** in the command prompt.
2. DA16200 EVB: reboot automatically with the DPM mode (by **dpm on**).
3. When the `mqtt_app` starts for the first time, it configures MQTT client. When it is configured, find MQTT configuration parameters in NVRAM. See `my_app_mqtt_user_config()` for more information on configuration.
4. Next the `mqtt_app` waits for the system to sync the system time with SNTP server which is required for successful TLS session.
5. Then, the `mqtt_app` starts `mqtt_client`.
6. After checking the successful connection of `mqtt_client` with `mqtt_broker`, the `mqtt_app` initializes the app resources in `my_app_init()`, and enter DPM sleep.

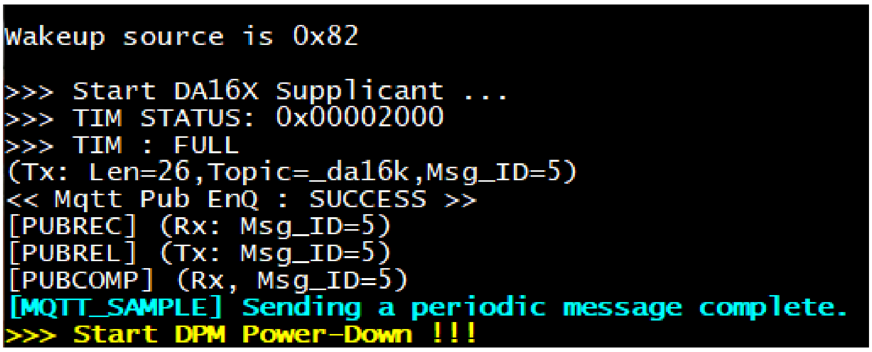


```
Connection COMPLETE to 58:ef:68:63:13:95

-- DHCP Client WLAN0: SEL(6)
-- DHCP Client WLAN0: REQ(1)
-- DHCP Client WLAN0: CHK(8)
-- DHCP Client WLAN0: BOUND(10)
         Assigned addr   : 192.168.1.39
               netmask   : 255.255.255.0
               gateway   : 192.168.1.1
               DNS addr  : 192.168.1.1

         DHCP Server IP  : 192.168.1.1
         Lease Time      : 24h 00m 00s
         Renewal Time    : 20h 00m 00s

[MQTT_SAMPLE] MQTT Configuration exists.
>>> SNTP Server: pool.ntp.org (121.174.142.81)

>>> SNTP Time sync : 2022.10.05 - 06:39:54
>>> MQTT Client connection OK (da16x_D9CC)
üMQTT_SAMPLE] Periodic Publish scheduled (RTC).
```

Figure 65. MQTT client sample start-up (in DPM mode)

13.5.3.2.1 MQTT Publish

1. mqtt_app: register 30-second RTC timer in `my_app_init()` when the system is in the DPM mode. Every 30 seconds, the DA16200 EVB wakes up, and the `mqtt_app` tries to send a periodic message. `my_app_mqtt_pub_send_periodic()` is the RTC timer callback that triggers MQTT publish.
2. mqtt_sub: display the published message from the `mqtt_app`.
3. mqtt_app: after publishing the periodic message, enter DPM sleep.

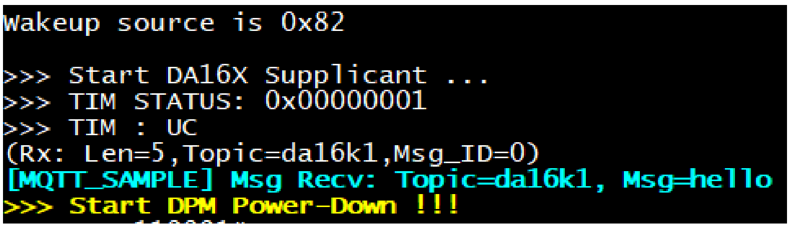


```
Wakeup source is 0x82

>>> Start DA16X Supplicant ...
>>> TIM STATUS: 0x00002000
>>> TIM : FULL
(Tx: Len=26,Topic=_da16k,Msg_ID=5)
<< Mqtt Pub EnQ : SUCCESS >>
[PUBREC] (Rx: Msg_ID=5)
[PUBREL] (Tx: Msg_ID=5)
[PUBCOMP] (Rx, Msg_ID=5)
[MQTT_SAMPLE] Sending a periodic message complete.
>>> Start DPM Power-Down !!!
```

Figure 66. Periodic MQTT publish (in DPM mode)

13.5.3.2.2 Receive MQTT Message

1. mqtt_pub: publish a message "hello" to the topic **da16k1** and try to publish it to other topics as well. The `mqtt_app` has subscribed to 3 topics: da16k1, da16k2, and da16k3.
2. The DA16200 EVB wakes up from DPM Sleep and the `mqtt_app` receives and display the message "hello" in the console. See the message callback `my_app_mqtt_msg_cb()`.
3. After displaying the message, the DA16200 EVB enters DPM Sleep.

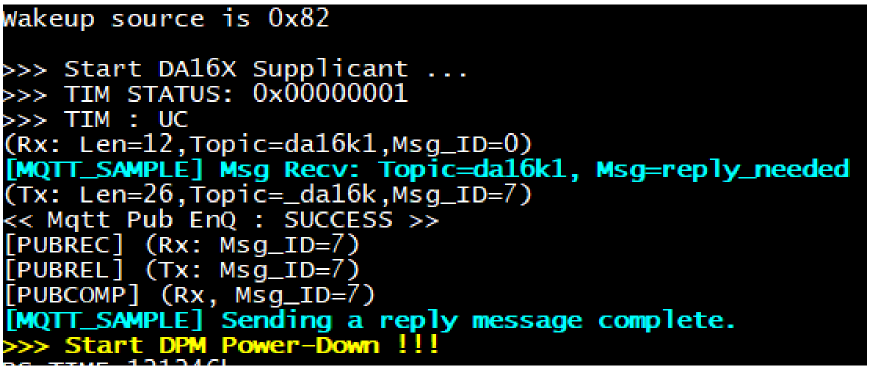


```
Wakeup source is 0x82

>>> Start DA16X Supplicant ...
>>> TIM STATUS: 0x00000001
>>> TIM : UC
(Rx: Len=5,Topic=da16k1,Msg_ID=0)
[MQTT_SAMPLE] Msg Recv: Topic=da16k1, Msg=hello
>>> Start DPM Power-Down !!!
```

Figure 67. Receive MQTT message (in DPM mode)

13.5.3.2.3 Receive and Reply MQTT Message

1. mqtt_pub: publish a message `reply_needed` to the topic **da16k1**.
2. The DA16200 EVB wakes up from DPM Sleep and the `mqtt_app` receives the message and try to publish a message "DA16K status: Not bad ( )" to topic **_da16k**. The message callback `my_app_mqtt_msg_cb()` upon receipt of "reply_needed", tries to publish a message to topic **_da16k**.
3. mqtt_sub (subscribed to _da16k): display the message received.
4. After displaying the message, the DA16200 EVB enters DPM sleep.

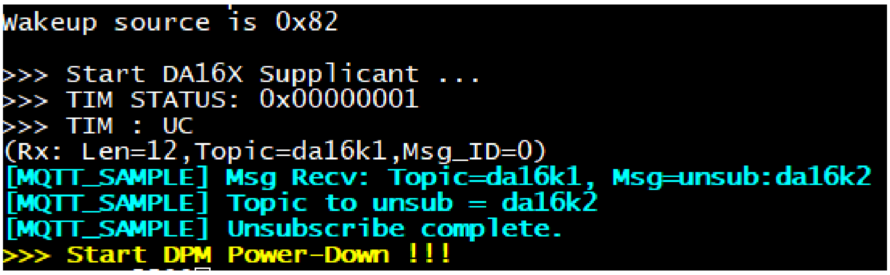


```
Wakeup source is 0x82

>>> Start DA16X Supplicant ...
>>> TIM STATUS: 0x00000001
>>> TIM : UC
(Rx: Len=12,Topic=da16k1,Msg_ID=0)
[MQTT_SAMPLE] Msg Recv: Topic=da16k1, Msg=reply_needed
(Tx: Len=26,Topic=_da16k,Msg_ID=7)
<< Mqtt Pub EnQ : SUCCESS >>
[PUBREC] (Rx: Msg_ID=7)
[PUBREL] (Tx: Msg_ID=7)
[PUBCOMP] (Rx, Msg_ID=7)
[MQTT_SAMPLE] Sending a reply message complete.
>>> Start DPM Power-Down !!!
```

Figure 68. MQTT message receive and reply (in DPM mode)

13.5.3.2.4 MQTT Unsubscribe

1. mqtt_pub: publish a message "unsub:da16k2" to the topic **da16k1**.
2. The DA16200 EVB wakes up from DPM Sleep and the `mqtt_app` receives the message and try to unsubscribe one of subscribed topics. The message callback `my_app_mqtt_msg_cb()`, upon receipt of "unsub:da16k2," tries to unsubscribe da16k2.
3. mqtt_pub: try publishing a message to **da16k2**. Make sure that the mqtt_app does not receive the message.
4. After displaying the message, the DA16200 EVB enters DPM sleep.



```
Wakeup source is 0x82

>>> Start DA16X Supplicant ...
>>> TIM STATUS: 0x00000001
>>> TIM : UC
(Rx: Len=12,Topic=da16k1,Msg_ID=0)
[MQTT_SAMPLE] Msg Recv: Topic=da16k1, Msg=unsub:da16k2
[MQTT_SAMPLE] Topic to unsub = da16k2
[MQTT_SAMPLE] Unsubscribe complete.
>>> Start DPM Power-Down !!!
```

Figure 69. MQTT unsubscribe action (in DPM mode)

13.5.4 Code Walkthrough

The MQTT client sample consists of two threads. The main thread is `mqtt_client_sample()`. The job handling thread is `my_app_q_handler()`.

Each job for sample application is triggered by callbacks.

```
void mqtt_client_sample(void * param)
{
…
      // Register callbacks to mqtt_client
      mqtt_client_set_msg_cb(my_app_mqtt_msg_cb);
      mqtt_client_set_pub_cb(my_app_mqtt_pub_cb);
      mqtt_client_set_conn_cb(my_app_mqtt_conn_cb);
      mqtt_client_set_subscribe_cb(my_app_mqtt_sub_cb);
…
```

```
        // mqtt user config in the 1st run
        my_app_mqtt_user_config();


…
        // Wait for SNTP sync
        ret = sntp_wait_sync(10);
…
        // Start mqtt_client
        mqtt_client_start();
…
        // Wait until mqtt_client is connected to mqtt_broker
        my_app_mqtt_chk_connection(10)
…
        // Application init
        ret = my_app_init();
…
        // Main event loop
        while (1) {
          …
          events = xEventGroupWaitBits(my_app_event_group,…
          …
          if (events & EVT_PUB_COMPLETE) {
          …
                  PRINTF(CYAN_COLOR "[MQTT_SAMPLE] Sending a periodic message complete. \n" CLEAR_COLOR);
          …
          } else if (events & EVT_PUB_ERROR) {
          …
          } else if (events & EVT_UNSUB_DONE) {
                  PRINTF(CYAN_COLOR "[MQTT_SAMPLE] Unsubscribe complete. \n" CLEAR_COLOR);
        …
          } else if (events & EVT_UNSUB_ERR) {
        …
        }
}
```

Periodic MQTT publish is triggered by `my_app_mqtt_pub_send_periodic()` or `_my_app_mqtt_pub_send_periodic()` in the DPM mode.

```
static void _my_app_mqtt_pub_send_periodic(TimerHandle_t xTimer)
{
        DA16X_UNUSED_ARG(xTimer);

        BaseType_t ret;

        if (!mqtt_client_is_running() && !is_mqtt_client_thd_alive()) {
          PRINTF(CYAN_COLOR "[MQTT_SAMPLE] Mqtt_client is in terminated state, terminating my app ... \n"
CLEAR_COLOR);

          if ((ret = my_app_send_to_q(NAME_JOB_MY_APP_TERM, NULL, APP_MSG_TERMINATE, NULL)) != pdPASS ) {
                  PRINTF(RED_COLOR "[%s] Failed to add a message to Q (%d)\r\n" CLEAR_COLOR, __func__, ret);
          }

          return;
        } else if (!mqtt_client_is_running() && is_mqtt_client_thd_alive()) {
          PRINTF(CYAN_COLOR "[MQTT_SAMPLE] Mqtt_client may be trying to reconnect ... canceling the job this
time \n" CLEAR_COLOR);
          return;
        }

        if ((ret = my_app_send_to_q(NAME_JOB_MQTT_TX_PERIODIC, &tx_periodic, APP_MSG_PUBLISH, NULL)) !=
pdPASS ) {
          PRINTF(RED_COLOR "[%s] Failed to add a message to Q (%d)\r\n" CLEAR_COLOR, __func__, ret);
```

```
        }

        return;
}
```

The `mqtt_app` receives the message through the message callback `my_app_mqtt_msg_cb()`, which triggers the MQTT publish or MQTT unsubscribe action depending on message contents or types.

```
void my_app_mqtt_msg_cb(const char *buf, int len, const char *topic)
{
        DA16X_UNUSED_ARG(len);

        BaseType_t ret;

        PRINTF(CYAN_COLOR "[MQTT_SAMPLE] Msg Recv: Topic=%s, Msg=%s \n" CLEAR_COLOR, topic, buf);

        if (strcmp(buf, "reply_needed") == 0) {
          if ((ret = my_app_send_to_q(NAME_JOB_MQTT_TX_REPLY, &tx_reply, APP_MSG_PUBLISH, NULL)) != pdPASS )
{
                    PRINTF(RED_COLOR "[%s] Failed to add a message to Q (%d)\r\n" CLEAR_COLOR, __func__, ret);
          }
        } else if (strncmp(buf, APP_UNSUB_HDR, 6) == 0) {
          if ((ret = my_app_send_to_q(NAME_JOB_MQTT_UNSUB, NULL, APP_MSG_UNSUB, buf)) != pdPASS ) {
                    PRINTF(RED_COLOR "[%s] Failed to add a message to Q (%d)\r\n" CLEAR_COLOR, __func__, ret);
          }
        } else {
          return;
        }
}
```

For each action execution, the thread `my_app_q_handler()` handles the job when one is submitted to the message queue `my_app_q`.

```
void my_app_q_handler(void* arg)
{
…
        while (1) {
          …
          xStatus = xQueueReceive(my_app_q, &RecvVal, portMAX_DELAY);
          …
          if (RecvVal == APP_MSG_PUBLISH) {
                  …
                  my_app_mqtt_pub_msg( ) // invoke mqtt_client_send_message( )
          } else if (RecvVal == APP_MSG_UNSUB) {
                  mqtt_client_unsub_topic( )
                  …
        }
}
```

- Cert 2: Client Key

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEA3KO5EOFtm/3wcNmYEgF1VgQpiVtMmsfCuvNpEYh5QdWieSJv
K0xJLWZTw0FYaDt1K/iI/WPLpA9x6gjGveU9Wty8vZYQyDBP1UakYGURmvxQv45I
ivbvUoCFz2aiZNbPyVRu2u3XgvAbyoqiBYV6B5dDeJyccFQPJGoOPHV2608azh9u
gvasFPOYkv3NaMxyTJqtOdlj0kGSCEqvPlZsZQm218UO5FNqGZMQ6lt4TCNzj0vN
LPKuLTM7orb8xTtCbwB4IeCBchO8oJyBO/pTPX9xMMxAsPZxAXS+wL352C4ZSBCP
EvMGU1KZ3fFwOULO0GuKyzbqiNu92SFiS4fb/wIDAQABAoIBAQDcnbCc2mt5AM98
Z3aQ+nhSy9Kkj2/njDqAKIc0ituEIpNUwEOcbaj2Bk1W/W3iuyEMGHURuMmUgAUN
WD0w/5j705+9ieG56eTJgts1r5mM+SHch+6tVQAz5GLn4N4cKlaWHyDBM/S77k47
lacwEijUkkFaxm3+O27woEMf3OxNl24KmRenMYBhqcsoT4BYBw3Bh8xe+XN95rXj
2BdIbr5+RWGc9Zsz4o5Wmd4mL/JvbKeohrsecien4TZRzWFku93XV5kie1c1aJy1
nJ85bGJk4focmP/2ToxQysTbPYCxHVTIHuADK/qf9SGHJ9F7EBHE7+0isuwBbqOD
```

```
OzS8rHdRAoGBAPCXlaHumEkLIRv3enhpHPBYxnDndNCtT1T6+Cuit/vfo6K6oA7p
iUaej/GPZsDKXhayeTiEaq7QMinUtGkiCgGlVtXghXuCZz6KrH19W6wzC6Pbokmq
BZak4LQcvGavt3VzjliAKLcdn6nQt/+bp/jKDJOKVbvb30sjS035Ah4zAoGBAOrF
BgE9UTEnfQHIh7pyiM1DAomBbdrlRos8maQl26cHqUHN3+wy1bGHLzOjYFFoAasx
eizw7Gudgbae28WIP1yLGrpt15cqVAvlCYmBtZ3C98FuT3FYgEEZpWNmE8Om+5UM
td+mtMjonWAPkCYC+alqUZzeIs+CZs5CHKYCDqcFAoGBAOfkQv38GV2102jARJPQ
RGtINaRXApmrod43s4Fjac/kAzVyiZk18PFXHUhnvlMt+jgIN5yIzMbHtsHo2SbH
/zsM4MBuklm0G80FHjIp5HT6EksSA77amF5VdptDYzfaP4p+IYIdrKCqddzYZrCA
mArMvAhs+iuCRhuG3is+SZNPAoGAHs6r8w2w0dp0tP8zkGvnN8hLVO//EnJzx2G0
Z63wHQMMWu5BLCWflSRANW6C/SvAzE450hvralPI6cX+4PT4G5TFdSFk4RlU3hq4
Has/wewLxv5Kvnz2l5Rd96U1gr8u1GhOlYKyxop/3FMuf050pJ6nBwa/WquqAfb6
+23ZrmECgYEA6l0GFHwMFBNnpPuxHgYgS5+4g3+8DhZZIDc7IflBCBWF/ZwbM+nH
+JSxiYYjvD7zIBhndqERcZ+fvbZTQ8oymr3j5AESM0ZfAHbft6IFQWjDUC3IDUF/
4F0cUidFC8smu6Wa2tjvSIz7DfvmDsn1l+7s9qQvDxdyPas0IkL/v8w=
-----END RSA PRIVATE KEY-----
```

14. Network Examples: Protocols/Applications

14.1 CoAP Client

14.1.1 Peer Application

The example in this section requires a peer device (Laptop or desktop) running a CoAP test server application to demonstrate the DA16200 CoAP client sample application. The sample application is based on Eclipse Californium™ (https://www.eclipse.org/californium/) and runs on a Windows OS as shown in Figure 70.

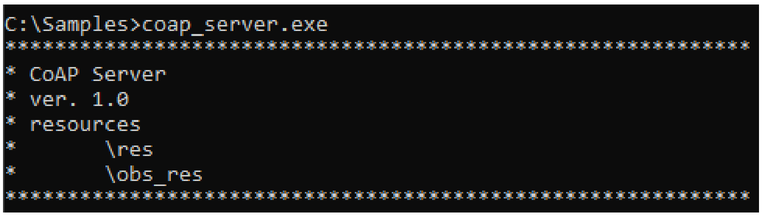


Figure 70. Start of CoAP server application

The CoAP server application is a simple CoAP server. It has two resources, called \res and \obs_res. The **res** resource allows GET, POST, PUT, DELETE, and PING methods. The **obs_res** resource allows OBSERVE request to send an observe notification every ten seconds.

14.1.2 How to Run

1. Run a CoAP server application on the peer computer.
2. In the e² studio, import a project for the CoAP Client application.
   `~/SDK/apps/common/examples/Network/CoAP_Client/projects/da16200`
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. Use the console command to set up the Wi-Fi station interface.

After a connection is made to an AP, the example application initializes a CoAP client to start the service.

14.1.3 CoAP Client Initialization

This section explains how to initialize and construct a CoAP client.

```
int coap_client_sample_init_config(coap_client_sample_conf_t *config)
{
      int ret = DA_APP_SUCCESS;

      coap_client_t *coap_client_ptr = &config->coap_client;

      config->state = COAP_CLIENT_SAMPLE_STATE_SUSPEND;

      //Init coap client
      ret = coap_client_init(coap_client_ptr, COAP_CLIENT_SAMPLE_DEF_NAME);
      if (ret != DA_APP_SUCCESS) {
          PRINTF("[%s]Failed to init coap client(0x%x)\r\n", __func__, -ret);
          goto end;
      }

      coaps_client_set_authmode(coap_client_ptr, 0);

      config->req_port = COAP_CLIENT_SAMPLE_REQUEST_PORT;
      config->obs_port = COAP_CLIENT_SAMPLE_OBSERVE_PORT;

end:

      return ret;
}
```

The `coap_client_sample_init_config` function guides how the CoAP client is initialized. The `coap_client_init` function initializes the CoAP Client instance. If a CoAP observe relationship is already established in DPM wake-up, it is recovered. The API's details are as follows:

Table 22. APIs for initializing CoAP client

Item	Description
int coap_client_init(coap_client_t *client_ptr, char *name_ptr)	
Prototype	int coap_client_init(coap_client_t *client_ptr, char *name_ptr)
Parameter	client_ptr: CoAP Client instance pointer name_ptr: Name of CoAP Client
Return	0 (DA_APP_SUCCESS) on success.
Description	Initialize CoAP Client.
int coaps_client_set_authmode(coap_client_t *client_ptr, unsigned int mode)	
Prototype	int coaps_client_set_authmode(coap_client_t *client_ptr, unsigned int mode)
Parameter	client_ptr: CoAP Client instance pointer mode: DTLS's auth mode
Return	0 (DA_APP_SUCCESS) on success.
Description	If true, DTLS server's certificate validity is checked during DTLS handshake. Default is false.

14.1.4 CoAP Client Deinitialization

This section explains how to release the CoAP client.

```
int coap_client_sample_deinit_config(coap_client_sample_conf_t *config)
{
      int ret = DA_APP_SUCCESS;
      coap_client_t *coap_client_ptr = &config->coap_client;

      //Deinit coap client
      ret = coap_client_deinit(coap_client_ptr);
      if (ret != DA_APP_SUCCESS) {
          PRINTF("[%s]Failed to deinit coap client(0x%x)\r\n", __func__, -ret);
      }

      return ret;
}
```

The coap_client_deinit function releases the CoAP client. The API details are as follows.

Table 23. API for deinitializing CoAP client

Item	Description
int coap_client_deinit(coap_client_t *client_ptr)	
Prototype	int coap_client_deinit(coap_client_t *client_ptr)
Parameter	client_ptr: CoAP Client instance pointer
Return	0 (DA_APP_SUCCESS) on success.
Description	Deinitialize CoAP client.

14.1.5 CoAP Client Request and Response

The DA16200 provides a CoAP client request (GET/POST/PUT/DELETE/PING) and response. In this section, we describe how the DA16200 sends the CoAP request to the CoAP server and receives the CoAP response.

14.1.5.1 CoAP URI and Proxy URI

To transmit a CoAP request and response, a URI must be set up. DA16200 provides APIs as shown below.

Table 24. APIs for setting up CoAP URI and proxy URI

Item	Description
int coap_client_set_uri(coap_client_t *client_ptr, unsigned char *uri, size_t urilen)	
Prototype	int coap_client_set_uri(coap_client_t *client_ptr, unsigned char *uri,size_t urilen)
Parameter	client_ptr: CoAP Client instance pointer uri: URI of CoAP request urilen: Length of URI
Return	0 (DA_APP_SUCCESS) on success.
Description	Setup URI.
int coap_client_set_proxy_uri(coap_client_t *client_ptr, unsigned char *uri, size_t urilen)	
Prototype	int coap_client_set_proxy_uri(coap_client_t *client_ptr, unsigned char *uri, size_t urilen)
Parameter	client_ptr: CoAP Client instance pointer uri: Proxy URI of CoAP request urilen: Length of URI
Return	0 (DA_APP_SUCCESS) on success.
Description	Setup Proxy URI. If URI is NULL, previous Proxy URI is removed.

14.1.5.2 GET Method

The DA16200 provides an API to send a GET request as shown in the example code.

```
int coap_client_sample_request_get(coap_client_sample_conf_t *config,
                                    coap_client_sample_request_t *request)

{
    int ret = DA_APP_SUCCESS;
    coap_client_t *coap_client_ptr = &config->coap_client;
    coap_rw_packet_t resp_packet;
    memset(&resp_packet, 0x00, sizeof(coap_rw_packet_t));

    //set URI.
    ret = coap_client_set_uri(coap_client_ptr,
                              (unsigned char *)request->uri,
                               request->urilen);

    //set Proxy URI. If null, previous proxy uri will be removed.
    ret = coap_client_set_proxy_uri(coap_client_ptr,
                                    (unsigned char *)request->proxy_uri,
                                     request->proxy_urilen);

    //send coap request
    ret = coap_client_request_get_with_port(coap_client_ptr, config->req_port);

    //receive coap response
    ret = coap_client_recv_response(coap_client_ptr, &resp_packet);

    //display output
    if (resp_packet.payload.len) {
      coap_client_sample_hexdump("GET Request",
                                 resp_packet.payload.p,
```

```
                                           resp_packet.payload.len);
      }

end:
      //release coap response
      coap_clear_rw_packet(&resp_packet);

      return ret;
}
```

The CoAP GET request is generated and sent in function `coap_client_request_get_with_port()`. A CoAP response is received in function `coap_client_recv_response()`. The API details are as follows:

Table 25. GET API for CoAP client

Item	Description
int coap_client_request_get_with_port(coap_client_t *client_ptr, unsigned int port)	
Prototype	int coap_client_request_get_with_port(coap_client_t *client_ptr, unsigned int port)
Parameter	client_ptr: CoAP Client instance pointer port: UDP socket's local port number
Return	0 (DA_APP_SUCCESS) on success.
Description	CoAP client sends GET request.

The DA16200 CoAP client sample application provides a command to send a GET request to the CoAP server. Figure 71, Figure 72, and Figure 73 show the interaction of two DA16200 CoAP clients with the CoAP server for a GET request.

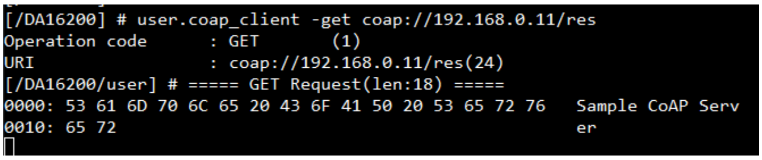


Figure 71. GET method of CoAP client #1

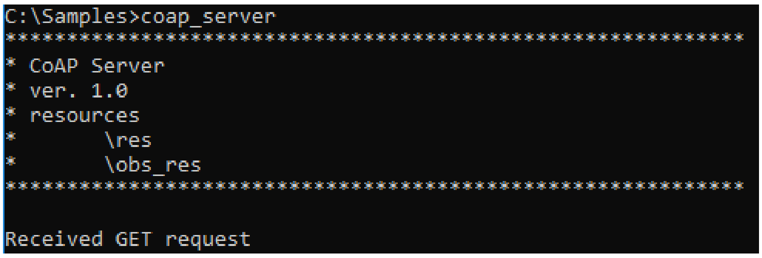


Figure 72. GET method of CoAP client #2

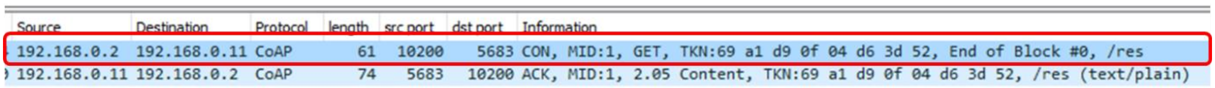


Figure 73. GET method of CoAP client #3

14.1.5.3 POST Method

The DA16200 provides an API to send a POST request as shown in the example code.

```
int coap_client_sample_request_post(coap_client_sample_conf_t *config,
                                    coap_client_sample_request_t *request)
{
      int ret = DA_APP_SUCCESS;
      coap_client_t *coap_client_ptr = &config->coap_client;
      coap_rw_packet_t resp_packet;
```

```
        memset(&resp_packet, 0x00, sizeof(coap_rw_packet_t));

        //set URI
        ret = coap_client_set_uri(coap_client_ptr,
                                  (unsigned char *)request->uri,
                                   request->urilen);

        //set Proxy URI. If null, previous proxy uri will be removed.
        ret = coap_client_set_proxy_uri(coap_client_ptr,
                                        (unsigned char *)request->proxy_uri,
                                         request->proxy_urilen);

        //send coap request
        ret = coap_client_request_post_with_port(coap_client_ptr, config->req_port,
                                                 request->data, request->datalen);

        //receive coap response
        ret = coap_client_recv_response(coap_client_ptr, &resp_packet);

        //display output
        if (resp_packet.payload.len) {
            coap_client_sample_hexdump("POST Request",
                                        resp_packet.payload.p,
                                        resp_packet.payload.len);
        }

end:
        //release coap response
        coap_clear_rw_packet(&resp_packet);
        return ret;
}
```

A CoAP POST request is generated and sent in function `coap_client_request_post_with_port()`. A CoAP response is received in function `coap_client_recv_response()`. The API details are as follows.

Table 26. POST API for CoAP client

Item	Description
UINT coap_client_request_post_with_port(coap_client_t *client_ptr, UINT port, unsigned char * payload, unsigned int payload_len)	
Prototype	UINT coap_client_request_post_with_port(coap_client_t *client_ptr, UINT port, unsigned char *payload, unsigned int payload_len)
Parameter	client_ptr: CoAP Client instance pointer port: UDP socket's local port number payload: Payload pointer payload_len: Length of payload
Return	0 (DA_APP_SUCCESS) on success.
Description	CoAP client sends POST request.

The DA16200 CoAP client sample application has a command to send a POST request to a CoAP server. Figure 74, Figure 75, and Figure 76 show the interaction of two DA16200 CoAP clients with the CoAP server for a POST request.

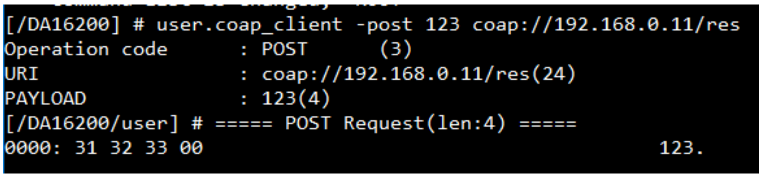


Figure 74. POST method of CoAP client #1

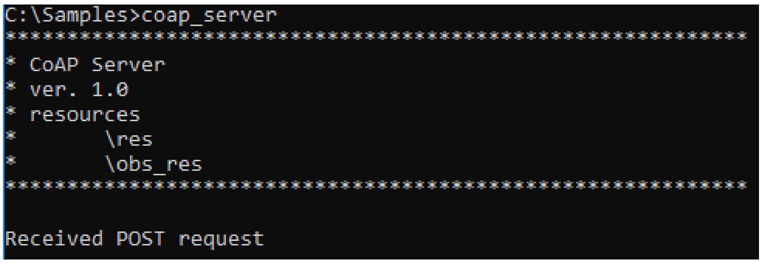


Figure 75. POST method of CoAP client #2

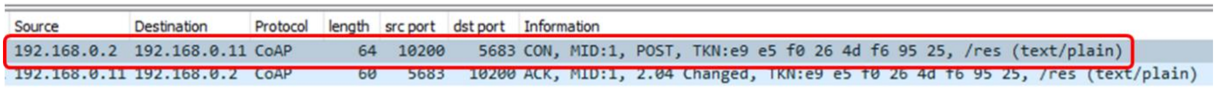


Figure 76. POST method of CoAP client #3

14.1.5.4 PUT Method

The DA16200 provides an API to send a PUT request as shown in the example code.

```
int coap_client_sample_request_put(coap_client_sample_conf_t *config,
                                   coap_client_sample_request_t *request)
{
      int ret = DA_APP_SUCCESS;
      coap_client_t *coap_client_ptr = &config->coap_client;
      coap_rw_packet_t resp_packet;

      memset(&resp_packet, 0x00, sizeof(coap_rw_packet_t));

      //set URI
      ret = coap_client_set_uri(coap_client_ptr,
                                (unsigned char *)request->uri,
                                 request->urilen);

      //set Proxy URI. If null, previous proxy uri will be removed.
      ret = coap_client_set_proxy_uri(coap_client_ptr,
                                      (unsigned char *)request->proxy_uri,
                                       request->proxy_urilen);


      //send coap request
      ret = coap_client_request_put_with_port(coap_client_ptr, config->req_port,
                                              request->data,
                                              request->datalen);

      //receive coap response
      ret = coap_client_recv_response(coap_client_ptr, &resp_packet);

      //display output
      if (resp_packet.payload.len) {
        coap_client_sample_hexdump("PUT Request",
                                   resp_packet.payload.p,
                                   resp_packet.payload.len);
      }
```

```
end:
      //release coap response
      coap_clear_rw_packet(&resp_packet);

      return ret;
}
```

The CoAP PUT request is generated and sent in function `coap_client_request_put_with_port()`. A CoAP response is received in function `coap_client_recv_response()`. The API details are as follows.

Table 27. PUT API for CoAP client

Item	Description
int coap_client_request_put_with_port(coap_client_t *client_ptr, unsigned int port, unsigned char *payload, unsigned int payload_len)	
Prototype	int coap_client_request_put_with_port(coap_client_t *client_ptr, unsigned int port, unsigned char *payload, unsigned int payload_len)
Parameter	client_ptr: CoAP Client instance pointer port: UDP socket's local port number payload: Payload pointer payload_len: Length of payload
Return	0 (DA_APP_SUCCESS) on success.
Description	CoAP Client sends PUT request.

The DA16200 CoAP client sample application provides a command to send a PUT request to the CoAP server. Figure 77, Figure 78, and Figure 79 show the interaction of two DA16200 CoAP clients and the CoAP server for PUT requests.

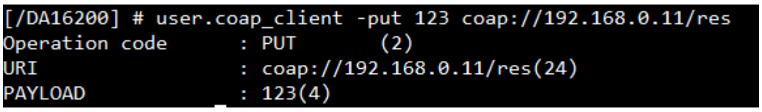


Figure 77. PUT method of CoAP client #1

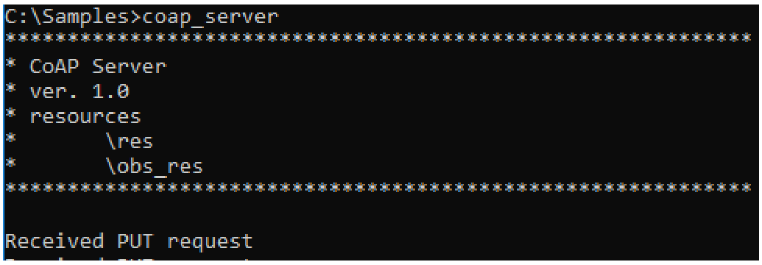


Figure 78. PUT method of CoAP client #2

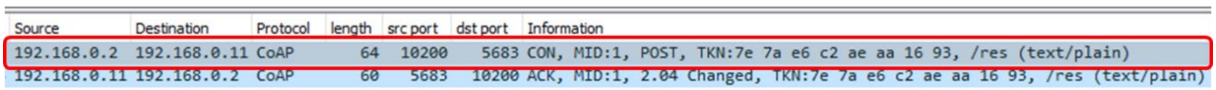


Figure 79. PUT method of CoAP client #3

14.1.5.5 DELETE Method

The DA16200 provides an API to send a DELETE request as shown in the example code.

```
int coap_client_sample_request_delete(coap_client_sample_conf_t *config,
                                      coap_client_sample_request_t *request)
{
      int ret = DA_APP_SUCCESS;
      coap_client_t *coap_client_ptr = &config->coap_client;
      coap_rw_packet_t resp_packet;
```

```
	memset(&resp_packet, 0x00, sizeof(coap_rw_packet_t));

	//set URI
	ret = coap_client_set_uri(coap_client_ptr,
	                          (unsigned char *)request->uri,
	                           request->urilen);

	//set Proxy URI. If null, previous proxy uri will be removed.
	ret = coap_client_set_proxy_uri(coap_client_ptr,
	                                (unsigned char *)request->proxy_uri,
	                                 request->proxy_urilen);

	//send coap request
	ret = coap_client_request_delete_with_port(coap_client_ptr, config->req_port);

	//receive coap response
	ret = coap_client_recv_response(coap_client_ptr, &resp_packet);

	//display output
	if (resp_packet.payload.len) {
	  coap_client_sample_hexdump("DELETE Request",
	                             resp_packet.payload.p,
	                             resp_packet.payload.len);
	}

end:
	//release coap response
	coap_clear_rw_packet(&resp_packet);

	return ret;
}
```

A CoAP DELETE request is generated and sent in function `coap_client_request_delete_with_port()`. A CoAP response is received in function `coap_client_recv_response()`. The API details are as follows.

Table 28. DELETE API for CoAP client

Item	**Description**
int coap_client_request_delete_with_port(coap_client_t *client_ptr, unsigned int port)	
Prototype	int coap_client_request_delete_with_port(coap_client_t *client_ptr, unsigned int port)
Parameter	client_ptr: CoAP Client instance pointer port: UDP socket's local port number
Return	0(DA_APP_SUCCESS) on success
Description	CoAP Client sends DELETE request to the URI

The DA16200 CoAP client sample application provides a command to send a DELETE request to the CoAP server. Figure 80, Figure 81, and Figure 82 show the interaction of a DA16200 CoAP client and the CoAP server for a DELETE request.

```
[/DA16200] # user.coap_client -delete coap://192.168.0.11/res
Operation code       : DELETE    (4)
URI                  : coap://192.168.0.11/res(24)
```

Figure 80. DELETE method of CoAP client #1

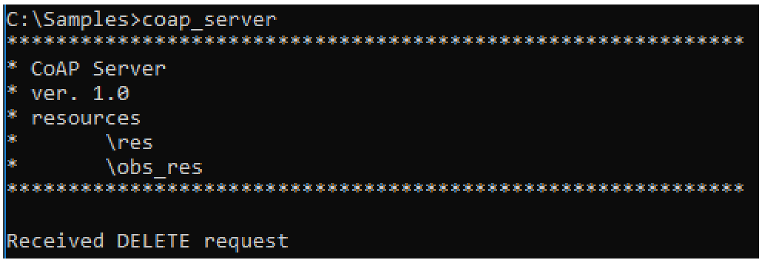


Figure 81. DELETE method of CoAP client #2

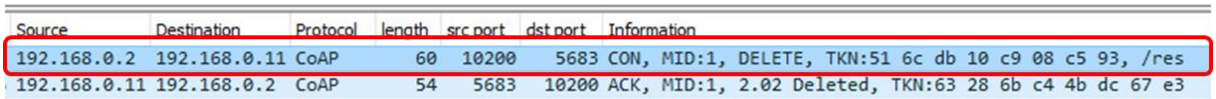


Figure 82. DELETE method of CoAP client #3

14.1.5.6 CoAP Ping

The DA16200 provides an API to send a PING request as shown in the example code.

```
int coap_client_sample_request_ping(coap_client_sample_conf_t *config,
                                    coap_client_sample_request_t *request)
{
      int ret = DA_APP_SUCCESS;
      coap_client_t *coap_client_ptr = &config->coap_client;

      //set URI
      ret = coap_client_set_uri(coap_client_ptr,
                                (unsigned char *)request->uri,
                                 request->urilen);

      //set Proxy URI. If null, previous proxy uri will be removed.
      ret = coap_client_set_proxy_uri(coap_client_ptr,
                                      (unsigned char *)request->proxy_uri,
                                       request->proxy_urilen);

      //progress ping request
      ret = coap_client_ping_with_port(coap_client_ptr, config->req_port);

end:

      return ret;
}
```

A CoAP PING request is processed in function `coap_client_ping_with_port()`. The API details are as follows.

Table 29. PING API for CoAP client

Item	Description
int coap_client_ping_with_port(coap_client_t *client_ptr, unsigned int port)	
Prototype	int coap_client_ping_with_port(coap_client_t *client_ptr, unsigned int port)
Parameter	client_ptr: CoAP Client instance pointer port: UDP socket's local port number
Return	0 (DA_APP_SUCCESS) on success.
Description	CoAP client sends PING request.

The DA16200 CoAP client sample application has a command to send a PING method to the CoAP server. Figure 83 and Figure 84 show the interaction of the DA16200 CoAP client and the CoAP server for a PING request.

```
[/DA16200] # user.coap_client -ping coap://192.168.0.11/res
Operation code      : PING      (7)
URI                 : coap://192.168.0.11/res(24)
```

Figure 83. PING method of CoAP client #1

Source	Destination	Protocol	length	src port	dst port	Information
192.168.0.2	192.168.0.11	CoAP	60	10200	5683	CON, MID:3, Empty Message, TKN:7e 7a e6 c2 ae aa 16 95, /res
192.168.0.11	192.168.0.2	CoAP	46	5683	10200	RST, MID:3, Empty Message

Figure 84. PING method of CoAP client #2

14.1.5.7 CoAP Response

The DA16200 constructs a CoAP response in `coap_rw_packet_t` structure. In this section, details are given of how a CoAP response is constructed.

```
~/SDK/core/coap/coap_common.h

typedef struct {
    /// Version number
    uint8_t version;
    /// Message type
    uint8_t type;
    /// Token length
    uint8_t token_len;
    /// Status code
    uint8_t code;
    /// Message-ID
    uint8_t msg_id[2];
} coap_header_t;

typedef struct {
    /// Option number
    uint8_t num;
    /// Option value
    coap_rw_buffer_t buf;
} coap_rw_option_t;

typedef struct {
    /// Header of the packet
    coap_header_t header;
    /// Token value, size as specified by header.token_len
    coap_rw_buffer_t token;
    /// Number of options
    uint8_t numopts;
    /// Options of the packet
    coap_rw_option_t opts[MAXOPT];
    /// Payload carried by the packet
    coap_rw_buffer_t payload;
} coap_rw_packet_t;
```

The `coap_rw_packet_t` structure includes the CoAP response information. After CoAP response is received, the DA16200 parses and constructs it. To receive a CoAP response, DA16200 provides an API. See Table 30. The API must be called after a CoAP requests to send a response.

Table 30. Response APIs for CoAP client

Item	Description
void coap_clear_rw_packet(coap_rw_packet_t *packet)	
Prototype	void coap_clear_rw_packet(coap_rw_packet_t *packet)

Item	Description
Parameter	client_ptr: CoAP Client instance pointer resp_ptr: CoAP response
Return	0 (DA_APP_SUCCESS) on success.
Description	Release coap_rw_packet structure.
int coap_client_recv_response(coap_client_t *client_ptr, coap_rw_packet_t *resp_ptr)	
Prototype	int coap_client_recv_response(coap_client_t *client_ptr, coap_rw_packet_t *resp_ptr)
Parameter	client_ptr: CoAP Client instance pointer resp_ptr: CoAP response
Return	0 (DA_APP_SUCCESS) on success.
Description	Receive CoAP response for specific CoAP request.

14.1.6 CoAP Observe

This section describes how CoAP observe is registered and deregistered from the CoAP server. DA16200 provides CoAP observe functionality. After registration at a CoAP server, DA16200 (CoAP client) is ready to receive an observe notification.

14.1.6.1 Registration

The DA16200 provides an API to register a CoAP observe as shown in the example code.

```
int coap_client_sample_register_observe(coap_client_sample_conf_t *config,
                                        coap_client_sample_request_t *request)
{
      int ret = DA_APP_SUCCESS;
      coap_client_t *coap_client_ptr = &config->coap_client;

      //set URI
      ret = coap_client_set_uri(coap_client_ptr,
                                (unsigned char *)request->uri,
                                 request->urilen);

      //set Proxy URI. If null, previous proxy uri will be removed.
      ret = coap_client_set_proxy_uri(coap_client_ptr,
                                      (unsigned char *)request->proxy_uri,
                                       request->proxy_urilen);

      //register coap observe
      ret = coap_client_set_observe_notify_with_port(coap_client_ptr,
                                          config->obs_port,
                                          coap_client_sample_observe_notify,
                                          coap_client_sample_observe_close_notify);

end:

      return ret;
}
```

The DA16200 CoAP observe allows only one connection. After successful registration, the DA16200 CoAP client allows receiving an observe notification. When the observe notification is received, the callback function (observe_notify) is called. If there is no observe notification during the max-age, the close callback function (observe_close_notify) is called. The API details are as follows.

Table 31. Observe registration API for CoAP client

Item	Description
int coap_client_set_observe_notify_with_port(coap_client_t *client_ptr, unsigned int port, int (*observe_notify)(void *client_ptr, coap_rw_packet_t *resp_ptr), void (*observe_close_notify)(char *timer_name))	
Prototype	int coap_client_set_observe_notify_with_port(coap_client_t *client_ptr, unsigned int port, int (*observe_notify)(void *client_ptr, coap_rw_packet_t *resp_ptr), void (*observe_close_notify) (char *timer_name))
Parameter	client_ptr: CoAP Client instance pointer port: UDP socket's local port number observe_notify: Callback function for CoAP observe notification observe_close_notify: Callback function for CoAP observe closing
Return	0 (DA_APP_SUCCESS) on success.
Description	Register CoAP observe. The callback function, observe_notify, is called when CoAP observe notification is received.

The DA16200 CoAP client sample application has a command for CoAP observe. Figure 85, Figure 86, and Figure 87 show the interaction of the DA16200 CoAP client and the CoAP server for CoAP observe. The CoAP server sends an observe notification every five seconds before deregistration.

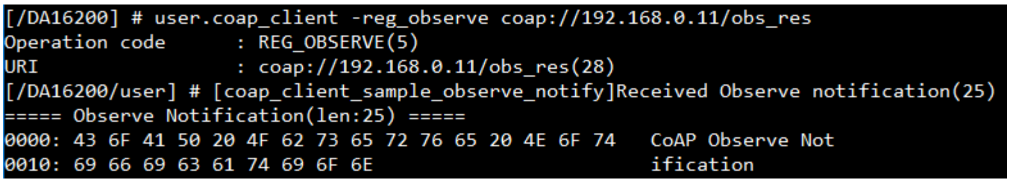

```
[/DA16200] # user.coap_client -reg_observe coap://192.168.0.11/obs_res
Operation code       : REG_OBSERVE(5)
URI                  : coap://192.168.0.11/obs_res(28)
[/DA16200/user] # [coap_client_sample_observe_notify]Received Observe notification(25)
===== Observe Notification(len:25) =====
0000: 43 6F 41 50 20 4F 62 73 65 72 76 65 20 4E 6F 74    CoAP Observe Not
0010: 69 66 69 63 61 74 69 6F 6E                         ification
```

Figure 85. CoAP observe of CoAP client #1

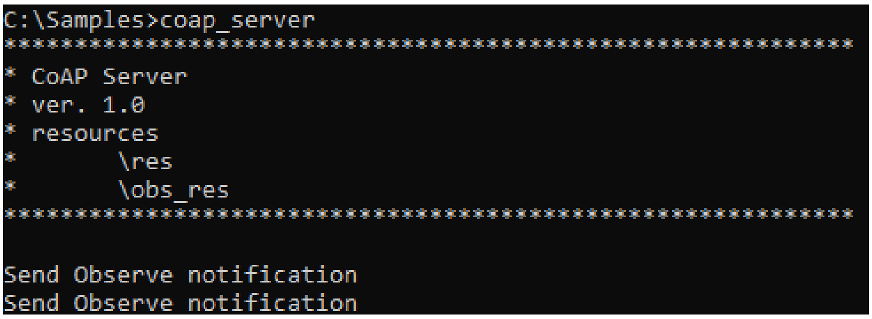
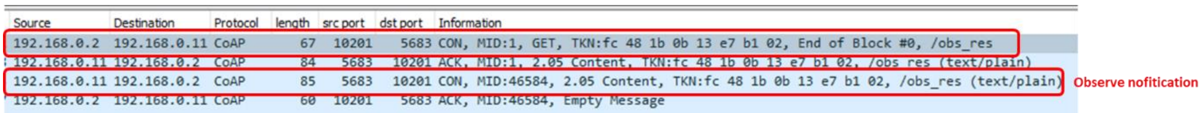

```
C:\Samples>coap_server
************************************************************
* CoAP Server
* ver. 1.0
* resources
*       \res
*       \obs_res
************************************************************

Send Observe notification
Send Observe notification
```

Figure 86. CoAP observe of CoAP client #2



Source	Destination	Protocol	length	src port	dst port	Information
192.168.0.2	192.168.0.11	CoAP	67	10201	5683	CON, MID:1, GET, TKN:fc 48 1b 0b 13 e7 b1 02, End of Block #0, /obs_res
192.168.0.11	192.168.0.2	CoAP	84	5683	10201	ACK, MID:1, 2.05 Content, TKN:fc 48 1b 0b 13 e7 b1 02, /obs_res (text/plain)
192.168.0.11	192.168.0.2	CoAP	85	5683	10201	CON, MID:46584, 2.05 Content, TKN:fc 48 1b 0b 13 e7 b1 02, /obs_res (text/plain)
192.168.0.2	192.168.0.11	CoAP	60	10201	5683	ACK, MID:46584, Empty Message

Figure 87. CoAP observe of CoAP client #3

14.1.6.2 Deregistration

The DA16200 provides an API to deregister a CoAP observe as shown in the example code.

```
int coap_client_sample_unregister_observe(coap_client_sample_conf_t *config)
{
      int ret = DA_APP_SUCCESS;
      coap_client_t *coap_client_ptr = &config->coap_client;
      coap_client_clear_observe(coap_client_ptr);

      return ret;
```

```
}
```

The API details are as follows.

Table 32. Observe deregistration API for CoAP client

Item	Description
VOID coap_client_clear_observe(coap_client_t *coap_client)	
Prototype	VOID coap_client_clear_observe(coap_client_t *coap_client)
Parameter	coap_client: CoAP Client instance pointer
Return	
Description	Deregister CoAP observe relation.

14.2 DNS Query

14.2.1 How to Run

This section shows how to get the IPv4 address from a domain name URL. Two types of API functions are supported to get the IP address:

- Get a single IPv4 address:
  - char *dns_A_Query(char *domain_name, unsigned long wait_option)
- Get multiple IPv4 addresses:
  - unsigned int dns_ALL_Query(unsigned char *domain_name,
    unsigned char *record_buffer,
    unsigned int record_buffer_size,
    unsigned int *record_count,
    unsigned long wait_option)

1. In the e[2] studio, import a project for the DNS Query sample application.
   `~/SDK/apps/common/examples/Network/DNS_Query/projects/da16200`
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. Use the console to set up the Wi-Fi station interface.
4. After a connection is made to an AP, the example application starts a DNS query operation with a test URL.

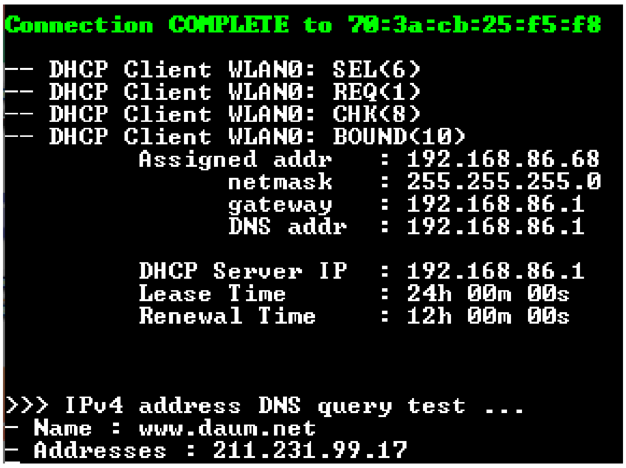


Figure 88. DNS query result

14.2.2 DNS Query Initialization

This example creates entry function which is `dns_query_sample()`.

```
void dns_query_sample(void * param)
{
```

```
    char        *test_url = NULL;
    if (netmode[WLAN0_IFACE] == DHCPCLIENT) {
      // wait until dhcp is done
      while (da16x_network_main_check_dhcp_state(WLAN0_IFACE) != DHCP_STATE_BOUND) {
          vTaskDelay(100);
      }
    }

    vTaskDelay(500);

    /* Check test url */
    test_url = read_nvram_string("TEST_DOMAIN_URL");
    if (test_url == NULL) {
        test_url = TEST_URL;
    }

    PRINTF("\n\n");
    dns_A_query_sample(test_url);
    vTaskDelete(NULL);
}
```

14.2.3 Get Single IPv4 Address

This example shows the use of the API function `char *dns_A_Query(char *domain_name, unsigned long wait_option)` to get the IPv4 address string with a domain name URL.

```
void dns_A_query_sample(char *test_url_str)
{
      char     *ipaddr_str = NULL;

      PRINTF(">>> IPv4 address DNS query test ...\n");

      /* DNS query with test url string */
      ipaddr_str = dns_A_Query(test_url_str, MAX_DNS_QUERY_TIMEOUT);

      /* Fail checking ... */
      if (ipaddr_str == NULL) {
          PRINTF("\nFailed to dns-query with %s\n", test_url_str);
      } else {
          PRINTF("- Name : %s\n", test_url_str);
          PRINTF("- Addresses : %s\n", ipaddr_str);
      }
}
```

14.3 SNTP and Get Current Time

Wi-Fi devices may need to synchronize the device clock on the internet with the use of TLS or communication with the server. DA16200 provides SNTP for this operation and users can use this function to get the current time.

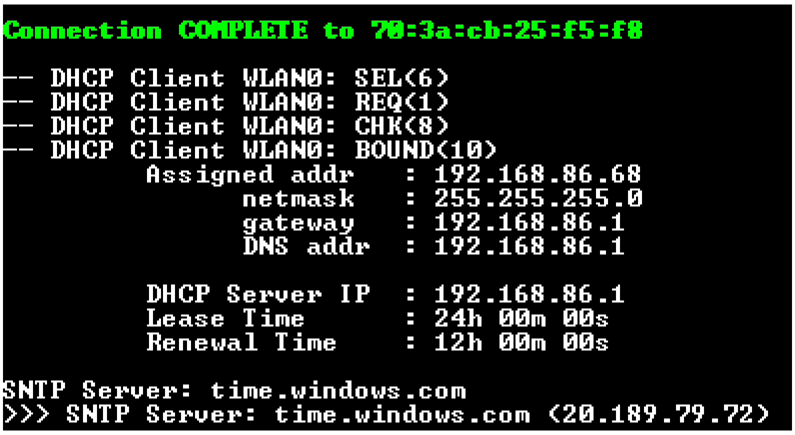
14.3.1 How to Run

1. In the e^2 studio, import a project for the SNTP and current time sample application.
   `~/SDK/apps/common/examples/ETC/Cur_Time/projects/da16200`
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. Use the console to set up the Wi-Fi station interface.
4. After a connection is made to an AP, the example application starts an SNTP client with test values.

```
 ~/SDK/apps/common/examples/ETC/Cur_Time/src/cur_time_sample.c
#define       TEST_SNTP_SERVER            "time.windows.com"
#define       TEST_SNTP_RENEW_PERIOD      600
#define       TEST_TIME_ZONE             (9 * 3600)     // seconds
#define       SNTP_ENABLE                 1
#define       ONE_SECONDS                 100
#define       CUR_TIME_LOOP_DELAY         10             // seconds
```

The legacy AP must be connected to the internet.

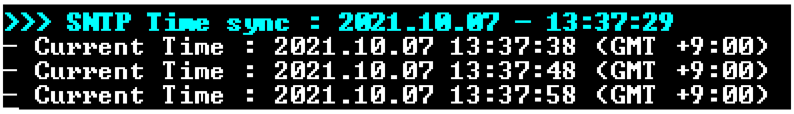
5. After a connection is made to the SNTP server, DA16200 shows the connection result on the debug console.



```
Connection COMPLETE to 70:3a:cb:25:f5:f8

-- DHCP Client WLAN0: SEL(6)
-- DHCP Client WLAN0: REQ(1)
-- DHCP Client WLAN0: CHK(8)
-- DHCP Client WLAN0: BOUND(10)
         Assigned addr   : 192.168.86.68
               netmask   : 255.255.255.0
               gateway   : 192.168.86.1
               DNS addr  : 192.168.86.1

         DHCP Server IP  : 192.168.86.1
         Lease Time      : 24h 00m 00s
         Renewal Time    : 12h 00m 00s

SNTP Server: time.windows.com
>>> SNTP Server: time.windows.com (20.189.79.72)
```

Figure 89. Result of DA16200 SNTP #1

The DA16200 periodically gets the current time (the test period: 10 seconds).



```
>>> SNTP Time sync : 2021.10.07 - 13:37:29
- Current Time : 2021.10.07 13:37:38 (GMT +9:00)
- Current Time : 2021.10.07 13:37:48 (GMT +9:00)
- Current Time : 2021.10.07 13:37:58 (GMT +9:00)
```

Figure 90. Result of DA16200 SNTP #2

14.3.2 Sample Code

1. Configure SNTP client information.

```
 ~/SDK/apps/common/examples/ETC/Cur_Time/src/cur_time_sample.c
void cur_time_sample(void * param)
{
      unsigned char  status;
      __time64_t     now;
      struct   tm *ts;
      char     time_buf[80];

      /* Config SNTP client */
      status = set_n_start_SNTP();
      if (status == pdFAIL) {
        PRINTF("[%s] Faile to start SNTP client ...\n", __func__);
        vTaskDelete(NULL);
        return;
      }
}
```

2. If the SNTP client has already been started with predefined values, then skip this configuration. Set the SNTP server address, time update period, and time zone and finally enable the function.

```
 ~/SDK/apps/common/examples/ETC/Cur_Time/src/cur_time_sample.c
static UCHAR set_n_start_SNTP(void)
{
```

```
      unsigned int   status = TX_SUCCESS;

      /* Check current SNTP running status */
      status = getSNTPuse();
      if (status == TX_TRUE) {
       /* Already SNTP module running ... */
       return TX_SUCCESS;
      }

      /* Config and save SNTP server domain */
      status = (unsigned int)setSNTPsrv(TEST_SNTP_SERVER, 0);

      if (status != TX_SUCCESS) {
       PRINTF("[%s] Failed to write nvram operation (SNTP server
                  domain)...\n", __func__);
       status = TX_START_ERROR;
       goto _exit;
      }

      /* Config and save SNTP periodic renew time : seconds */
      status = (unsigned int)setSNTPperiod(TEST_SNTP_RENEW_PERIOD);

      if (status != TX_SUCCESS) {
       PRINTF("[%s] Failed to write nvram operation (SNTP renew
                  period)...\n", __func__);
       status = TX_START_ERROR;
       goto _exit;
      }

      /* Config and save SNTP time zone */
      status = (unsigned int)setTimezone(TEST_TIME_ZONE);

      if (status != TX_SUCCESS) {
       PRINTF("[%s] Failed to write nvram operation (SNTP renew
                  period)...\n", __func__);
       status = TX_START_ERROR;
       goto _exit;
      }

      da16x_SetTzoff(TEST_TIME_ZONE);

      /* Config and save SNTP client mode : enable */
      status = setSNTPuse(SNTP_ENABLE);

      if (status != TX_SUCCESS) {
       PRINTF("[%s] Failed to write nvram operation (SNTP mode)...\n",
                   __func__);
       status = TX_START_ERROR;
       goto _exit;
      }

_exit :
      return status;
}
```

3. After a connection is made to the SNTP server, the DA16200 periodically gets the current time.

```
  ~/SDK/apps/common/examples/ETC/Cur_Time/src/cur_time_sample.c
void cur_time_sample(void * param)
{
      ...
      while (1) {
```

```
        /* delay */
        vTaskDelay(CUR_TIME_LOOP_DELAY * ONE_SECONDS);

        /* get current time */
        da16x_time64(NULL, &now);
        ts = (struct tm *)da16x_localtime64(&now);

        /* make time string */
        da16x_strftime(time_buf, sizeof(time_buf), "%Y.%m.%d %H:%M:%S", ts);

        /* display current time string */
        PRINTF("- Current Time : %s (GMT %+02ld:%02ld)\n",
                  time_buf,
                  da16x_Tzoff() / 3600,
                  da16x_Tzoff() % 3600);
      }
}
```

14.4 SNTP and Get Current Time in DPM

This example application applies to the DPM function. Most parts of the example came from as Section 14.3 and the only different part is to apply the example to the DPM mode.

14.4.1 How to Run

1. In the e[2] studio, import a project for the SNTP and the current time in the DPM sample application.
   `~/SDK/apps/common/examples/ETC/Cur_Time_DPM/projects/da16200`
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. Use the console to set up the Wi-Fi station interface.
4. After a connection is made to an AP, the example application starts an SNTP client with test values.

```
~/SDK/apps/common/examples/ETC/Cur_Time_DPM/src/cur_time_dpm_sample.c

#define       TEST_SNTP_SERVER            "time.windows.com"
#define       TEST_SNTP_RENEW_PERIOD      600
#define       TEST_TIME_ZONE             (9 * 3600)     // seconds
#define       SNTP_ENABLE                 1
#define       ONE_SECONDS                 100
#define       CUR_TIME_LOOP_DELAY         10             // seconds
```

NOTE
▪ If the SNTP client is started with pre-defined values, this configuration is ignored. ▪ The legacy AP must be connected to the internet.

5. After a connection is made to the SNTP server, DA16200 shows the connection result on the debug console and goes to DPM Sleep mode.

```
Connection COMPLETE to 70:3a:cb:25:f5:f8

-- DHCP Client WLAN0: SEL(6)
-- DHCP Client WLAN0: REQ(1)
-- DHCP Client WLAN0: CHK(8)
-- DHCP Client WLAN0: BOUND(10)
         Assigned addr   : 192.168.86.68
               netmask   : 255.255.255.0
               gateway   : 192.168.86.1
               DNS addr  : 192.168.86.1

         DHCP Server IP  : 192.168.86.1
         Lease Time      : 24h 00m 00s
         Renewal Time    : 20h 00m 00s

>>> SNTP Server: time.windows.com (20.189.79.72)

>>> SNTP Time sync : 2021.10.07 - 13:48:05
>>> Start DPM Power-Down !!!
```

Figure 91. Result of DA16200 SNTP DPM #1

The DA16200 periodically gets the current time (the test period is 10 seconds).

```
rtc_timeout (tid:5)

Wakeup source is 0x82

>>> TIM STATUS: 0x00000010
>>> TIM : FAST
- Current Time : 2021.10.07 13:51:55 (GMT +9:00)
>>> Start DPM Power-Down !!!
PS TIME 109826 us

rtc_timeout (tid:5)

Wakeup source is 0x82

>>> TIM STATUS: 0x00000010
>>> TIM : FAST
- Current Time : 2021.10.07 13:52:05 (GMT +9:00)
>>> Start DPM Power-Down !!!
PS TIME 109892 us
```

Figure 92. Result of DA16200 SNTP DPM #2

14.4.2 Sample Code

The SNTP configuration interface is the same as the non-DPM SNTP example. When the DA16200 wakes up from DPM Sleep mode, use the RTM API to get the current SNTP status, or save the SNTP status into the RTM.

```
~/SDK/apps/common/examples/ETC/Cur_Time_DPM/src/cur_time_dpm_sample.c
static unsigned char set_n_start_SNTP(void)
{
      unsigned char  status = pdPASS;

      /* Check current SNTP running status */
      if (dpm_mode_is_wakeup() == DPM_WAKEUP) {
          status = get_sntp_use_from_rtm();
      } else {
          status = get_sntp_use();
      }

      if (status == pdPASS) {
          long       time_zone;

          /* Already SNTP module running, set again time-zone ... */
          time_zone = get_timezone_from_rtm();
          da16x_SetTzoff(time_zone);

          return pdPASS;
```

```
        }

        if (dpm_mode_is_wakeup() == NORMAL_BOOT) {
            /* Config and save SNTP server URI */
            status = set_sntp_server(TEST_SNTP_SERVER, 0);

            if (status != pdPASS) {
                 PRINTF("Failed to write nvram operation (SNTP server domain)...\n");

                 status = pdFAIL;
                 goto _exit;
            }

            /* Config and save SNTP periodic renew time : seconds */
            status = set_sntp_period(TEST_SNTP_RENEW_PERIOD);
            if (status != pdPASS) {
                 PRINTF("Failed to write nvram operation (SNTP renew period)...\n");
                 status = pdFAIL;
                 goto _exit;
            }

            /* Config and save SNTP time zone */
            set_time_zone(TEST_TIME_ZONE);
            set_timezone_to_rtm(TEST_TIME_ZONE);
            da16x_SetTzoff(TEST_TIME_ZONE);
            set_time_zone(TEST_TIME_ZONE);

            /* Config, save, and run SNTP client */
            if (set_sntp_use(SNTP_ENABLE) != 0) {
                PRINTF("[%s] Failed to run SNTP...\n", __func__);
                status = pdFAIL;
                goto _exit;
            }

            /* Save config and start SNTP client */
            set_sntp_use_to_rtm(status);
        }

_exit :
        return status;
}
```

When connected to the SNTP server, the DA16200 starts an RTC timer to periodically get the current time.

```
~/SDK/apps/common/examples/ETC/Cur_Time_DPM/src/cur_time_dpm_sample.c
void cur_time_dpm_sample(void * param)
{
        ...
        /* Register periodic RTC Timer : Get current time */
        if (dpm_mode_is_wakeup() == NORMAL_BOOT){
            /* Time delay for stable running SNTP client */
            vTaskDelay(10);

            status = dpm_timer_create(SAMPLE_CUR_TIME_DPM,
                                      "timer1",
                                      display_cur_time,
                                      CUR_TIME_LOOP_DELAY,
                                      CUR_TIME_LOOP_DELAY);

           switch ((int)status) {
           case DPM_MODE_NOT_ENABLED    :
```

```
          case DPM_TIMER_SEC_OVERFLOW :
          case DPM_TIMER_ALREADY_EXIST:
          case DPM_TIMER_NAME_ERROR   :
          case DPM_UNSUPPORTED_RTM    :
          case DPM_TIMER_REGISTER_FAIL:
          case DPM_TIMER_MAX_ERR      :
             PRINTF(">>> Fail to create %s timer (err=%d)\n",
                   SAMPLE_CUR_TIME_DPM, (int)status);

             // Delay to display above message on console ...
             vTaskDelay(2);

             break;
          }

           /* Set flag to go to DPM Sleep mode 3 */
          dpm_app_sleep_ready_set(SAMPLE_CUR_TIME_DPM);
      } else {
           /* Notice initialize done to DPM module */
           dpm_app_wakeup_done(SAMPLE_CUR_TIME_DPM);
      }

      vTaskDelete(NULL);
}
```

The SNTP configuration interface is the same as for the non-DPM SNTP example.

```
~/SDK/apps/common/examples/ETC/Cur_Time_DPM/src/cur_time_dpm_sample.c
static void display_cur_time(char *timer_name)
{
      dpm_app_wakeup_done(SAMPLE_CUR_TIME_DPM);

      __time64_t     now;
      struct  tm *ts;
      char    time_buf[80];

      /* get current time */
      da16x_time64(NULL, &now);
      ts = (struct tm *)da16x_localtime64(&now);

      /* make time string */
      da16x_strftime(time_buf, sizeof(time_buf), "%Y.%m.%d %H:%M:%S", ts);

      /* display current time string */
      PRINTF("- Current Time : %s (GMT %+02ld:%02ld)\n",
          time_buf,
          da16x_Tzoff() / 3600,
          da16x_Tzoff() % 3600);

      vTaskDelay(1);

      /* Set flag to go to DPM Sleep mode 3 */
      dpm_app_sleep_ready_set(SAMPLE_CUR_TIME_DPM);
}
```

14.5 HTTP Client

The DA16200 SDK has a ported lwIP 2.1.2 stack. With this product, an application programmer can develop an HTTP client application that uses lwIP HTTP APIs.

14.5.1 How to Run

1. In the e² studio, import a project for the HTTP_Client sample application.
   `~/SDK/apps/common/examples/Network/Http_Client/projects/da16200`
5. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
2. Use the console to set up the Wi-Fi station interface and connect to the AP that is connected to the Internet.
3. Complete the setup and (re)start the sample.

14.5.2 Sample Code

The sample code shows the -get and -post methods. When the sample starts by default, it is executed as a -get method request. To request -post, define ENABLE_METHOD_POST_TEST at the top of the sample code.

To connect to HTTPS(TLS) server, enter "https://" instead of "http://" in the URL address. To set valid time information in the certificate before the HTTPS request, the system's current time must be set (SNTP service must be enabled). The URL and data of the sample code are examples, and it needs to modify them according to the user environment.

The sample code is executed as follows:

1. Using the `http_client_parse_uri()` API, set the port number for HTTP or HTTPS and parse the path and `host_name`.

```
unsigned char g_http_url[256] = {"http://httpbin.org/get"};
error = http_client_parse_uri(g_http_url, strlen((char *)g_http_url), &request);
if (error != ERR_OK) {
    PRINTF("Failed to set URI(error=%d) \r\n", error);
    goto finish;
}
```

2. Set a variable in the httpc_connection_t type and set the value to be passed to the API.
3. If the user registers the callback function in *headers_done_fn* and *result_fn*, the header response received from the server and the result value of the HTTP Client can be returned.

```
static httpc_connection_t g_conn_settings;
g_conn_settings.use_proxy = 0;
g_conn_settings.altcp_allocator = NULL;
g_conn_settings.headers_done_fn = httpc_cb_headers_done_fn;
g_conn_settings.result_fn = httpc_cb_result_fn;
```

4. When `ENABLE_METHOD_POST_TEST` is defined, users can insert the data they want to send to the server using the `httpc_insert_send_data()` API.

```
#if defined (ENABLE_METHOD_POST_TEST)
error = httpc_insert_send_data("POST", user_post_data, strlen(user_post_data));
if (error != ERR_OK) {
    PRINTF("Failed to insert data\n");
}
#endif
```

5. To perform TLS communication with the HTTP server that requires the HTTP client's certificate, define ENABLE_HTTPS_SERVER_VERIFY_REQUIRED. The certificate must have been previously stored in the TLS area of the DA16200 SFlash.

```
if (g_conn_settings.insecure) {
    memset(&g_conn_settings.tls_settings, 0x00,sizeof(httpc_secure_connection_t));
    g_conn_settings.tls_settings.incoming_len = HTTPC_MAX_INCOMING_LEN;
    g_conn_settings.tls_settings.outgoing_len = HTTPC_DEF_OUTGOING_LEN;

#ifdef ENABLE_HTTPS_SERVER_VERIFY_REQUIRED
    http_client_read_certs(&g_conn_settings.tls_settings);
    g_conn_settings.tls_settings.auth_mode = MBEDTLS_SSL_VERIFY_NONE;

/* SNI */
    sni_str = read_nvram_string(HTTPC_NVRAM_CONFIG_TLS_SNI);
    …
/* ALPN */
    if (read_nvram_int(HTTPC_NVRAM_CONFIG_TLS_ALPN_NUM, &alpn_cnt) == 0) {
    …

#endif //ENABLE_HTTPS_SERVER_VERIFY_REQUIRED
}
```

6. Call API to get request. User calls `httpc_get_file()` or `httpc_get_file_dns()` depending on whether hostname needs a DNS query. If the request is successful, the user can receive payload data through the registered `httpc_cb_recv_fn` callback function.

```
if (isvalidip((char *)request.hostname)) {
    ip4addr_aton(g_request.hostname, &g_server_addr);
    error = httpc_get_file(&g_server_addr,
                               g_request.port,
                               (char *)&g_request.path[0],
                               &g_conn_settings,
                               (altcp_recv_fn)httpc_cb_recv_fn,
                               NULL,
                               &g_connection);
} else {
    error = httpc_get_file_dns((char *)&g_request.hostname[0],
                                   g_request.port,
                                   (char *)&g_request.path[0],
                                   &g_conn_settings,
                                   (altcp_recv_fn)httpc_cb_recv_fn,
                                   NULL,
                                   &g_connection);
}
```

7. The `httpc_cb_recv_fn()` callback function receives a pbuf pointer. p->payload is the data received from the server.

```
static err_t httpc_cb_recv_fn(void *arg, struct tcp_pcb *tpcb,
                              struct pbuf *p, err_t err)
{
    if (p == NULL) {
       PRINTF("\n[%s:%d] Receive data is NULL !! \r\n", __func__, __LINE__);
       return ERR_BUF;
    } else {
       PRINTF("\n[%s:%d] Received length = %d \r\n", __func__, __LINE__, p->len);
       hexa_dump_print("Received data \r\n", p->payload,
                       p->len, 0, OUTPUT_HEXA_ASCII);
    }

    return ERR_OK;
}
```

14.6 HTTP Client in DPM

The DA16200 SDK has a ported lwIP 2.1.2 stack. With this product, an application programmer can develop an HTTP client application that uses lwIP HTTP APIs.

14.6.1 How to Run

1. In the e2 studio, import a project for the HTTP_Client sample application.
   `~/SDK/apps/common/examples/Network/Http_Client_DPM/projects/ da16200`
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. Use the console to set up the Wi-Fi station interface and connect to the AP that is connected to the Internet.
4. Complete the setup and (re)start the sample.

14.6.2 Sample Code

The sample code shows the -get and -post methods. When the sample starts by default, it is executed as a -get method request. To request -post, define ENABLE_METHOD_POST_TEST at the top of the sample code.

To connect to HTTPS(TLS) server, enter "https://" instead of "http://" in the URL address. To set valid time information in the certificate before the HTTPS request, the system's current time must be set (SNTP service must be enabled). The URL and data of the sample code are examples, and it needs to modify them according to the user environment.

The sample code is executed as follows:

1. If an application that uses the HTTP protocol is registered in DPM, a setting must be made not to enter DPM_SLEEP while HTTP transmission (request/response) is in progress. Set DPM_SLEEP after all transfers are complete.

```
void http_client_dpm_sample_entry(void * param)
{
    ...
    dpm_app_register(HTTP_CLIENT_SAMPLE_TASK_NAME, request.port);
    dpm_app_sleep_ready_clear(HTTP_CLIENT_SAMPLE_TASK_NAME);
    ...
}

static void httpc_cb_result_fn(void *arg, httpc_result_t httpc_result, u32_t
                                rx_content_len, u32_t srv_res, err_t err)
{
    PRINTF("\n httpc_result: %d, received: %d byte\r\n",
            httpc_result, rx_content_len);
    dpm_app_sleep_ready_set(HTTP_CLIENT_SAMPLE_TASK_NAME);
    return;
}
```

2. Using the `http_client_parse_uri()` API, set the port number for HTTP or HTTPS and parse the path and `host_name`.

```
unsigned char g_http_url[256] = {"http://httpbin.org/get"};

error = http_client_parse_uri(g_http_url, strlen((char *)g_http_url), &request);
if (error != ERR_OK) {
    PRINTF("Failed to set URI(error=%d) \r\n", error);
    goto finish;
}
```

3. Set a variable in the httpc_connection_t type and set the value to be passed to the API.
4. If the user registers the callback function in *headers_done_fn* and *result_fn*, the header response received from the server and the result value of the HTTP Client can be returned.

```
static httpc_connection_t conn_settings;
g_conn_settings.use_proxy = 0;
g_conn_settings.altcp_allocator = NULL;
g_conn_settings.headers_done_fn = httpc_cb_headers_done_fn;
g_conn_settings.result_fn = httpc_cb_result_fn;
```

5. When `ENABLE_METHOD_POST_TEST` is defined, users can insert the data they want to send to the server using the `httpc_insert_send_data()` API.

```
#if defined (ENABLE_METHOD_POST_TEST)
  error = httpc_insert_send_data("POST", user_post_data, strlen(user_post_data));
  if (error != ERR_OK) {
    PRINTF("Failed to insert data\n");
  }
#endif
```

6. To perform TLS communication with the HTTP Server that requires the HTTP Client's certificate, define `ENABLE_HTTPS_SERVER_VERIFY_REQUIRED`. The certificate must have been previously stored in the TLS area of the DA16200 SFlash.

```
if (g_conn_settings.insecure) {
    memset(&g_conn_settings.tls_settings, 0x00,sizeof(httpc_secure_connection_t));
    g_conn_settings.tls_settings.incoming_len = HTTPC_MAX_INCOMING_LEN;
    g_conn_settings.tls_settings.outgoing_len = HTTPC_DEF_OUTGOING_LEN;
#ifdef ENABLE_HTTPS_SERVER_VERIFY_REQUIRED
    http_client_read_certs(&g_conn_settings.tls_settings);
    g_conn_settings.tls_settings.auth_mode = MBEDTLS_SSL_VERIFY_NONE;

/* SNI */
    sni_str = read_nvram_string(HTTPC_NVRAM_CONFIG_TLS_SNI);
    …
/* ALPN */
    if (read_nvram_int(HTTPC_NVRAM_CONFIG_TLS_ALPN_NUM, &alpn_cnt) == 0) {
    …

#endif //ENABLE_HTTPS_SERVER_VERIFY_REQUIRED
}
```

7. Call API to get request. User calls `httpc_get_file()` or `httpc_get_file_dns()` depending on whether hostname needs a DNS query. If the request is successful, the user can receive payload data through the registered `httpc_cb_recv_fn` callback function.

```
if (isvalidip((char *)g_request.hostname)) {
    ip4addr_aton(g_request.hostname, & g_server_addr);
    error = httpc_get_file(&g_server_addr,
                               g_request.port,
                               char *)&g_request.path[0],
                               & g_conn_settings,
                               (altcp_recv_fn)httpc_cb_recv_fn,
                               NULL,
                               &connection);
} else {
    error = httpc_get_file_dns((char *)&g_request.hostname[0],
                                    g_request.port,
                                    (char *)&g_request.path[0],
                                    &g_conn_settings,
                                    altcp_recv_fn)httpc_cb_recv_fn,
                                    NULL,
                                    &connection);
}
```

8. The `httpc_cb_recv_fn()` callback function receives a pbuf pointer. p->payload is the data received from the server.

```
static err_t httpc_cb_recv_fn(void *arg, struct tcp_pcb *tpcb,
                              struct pbuf *p, err_t err)
{
    if (p == NULL) {
      PRINTF("\n[%s:%d] Receive data is NULL !! \r\n", __func__, __LINE__);
      return ERR_BUF;
    } else {
      PRINTF("\n[%s:%d] Received length = %d \r\n", __func__, __LINE__, p->len);
      hexa_dump_print("Received data \r\n", p->payload,
                      p->len, 0, OUTPUT_HEXA_ASCII);
    }
```

```
    return ERR_OK;
}
```

14.7 HTTP Server

The DA16200 SDK has a ported lwIP 2.1.2 stack. With this product, an application programmer can develop an HTTP server application that uses lwIP HTTP APIs.

14.7.1 How to Run

1. In the e^2 studio, import a project for the HTTP_Server sample application.
   `~/SDK/apps/common/examples/Network/Http_Server/projects/da16200`
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. Use the console to set up the Wi-Fi station interface and connect to the AP.
4. Complete the setup and (re)start the sample.

14.7.2 Sample Code

The sample code shows the -get methods.

1. The HTTP Server sample code supports both HTTP and HTTPS (Default is HTTP).

   To operate with HTTPS, define ENABLE_HTTPS_SERVER as shown below. Also, update the certificate embedded in the code (tls_srv_sample_cert, tls_srv_sample_key) as needed.

   ```
   /// HTTPS server
   #define ENABLE_HTTPS_SERVER
   ```

2. The HTTP server can be operated simply by calling the httpd_init() API.

   Also, user callback function can be registered as argument value. The registered callback function is called when data is received from the HTTP Client.

```
   /* Callback function*/
static err_t http_server_cb_recv_fn(struct pbuf *p, err_t err)
{
    err_t error = ERR_OK;
    extern void hex_dump(UCHAR * data, UINT length);

    PRINTF("[%s] err = %d, p->tot_len = %d, p->len = %d\n", __func__, err, p->tot_len, p->len);
    hex_dump(p->payload, p->len);

    return error;
}

/* Server task */
static void http_server_sample(void *params)
{
    httpd_init((altcp_user_recv_fn)http_server_cb_recv_fn);

    PRINTF("[%s] HTTP-Server Start!! \r\n", __func__);

    while (1){
        vTaskDelay(100);
    }
    return ;
}
```

3. The HTTPS server must set the key and certificate information required for TLS.

```
struct altcp_tls_config *tls_srv_sample_config = NULL;
...
tls_srv_sample_config =
```

```
  altcp_tls_create_config_server_privkey_cert(tls_srv_sample_key,
                                                    tls_srv_sample_key_len,
                                                    NULL,
                                                    0,
                                                    tls_srv_sample_cert,
                                                    tls_srv_sample_cert_len);
  if (!tls_srv_sample_config) {
      PRINTF("[%s] Failed to create tls config\r\n", __func__);
      goto end_of_task;
  }

  httpd_inits(tls_srv_sample_config, (altcp_user_recv_fn)http_server_cb_recv_fn);

  PRINTF("[%s] HTTPS-Server Start!! \r\n", __func__);

end_of_task:
  while (1) {
      vTaskDelay(100);
  }
```

4. If the HTTP Server works successfully, test the **-get** method as follows.

   Use the web browser of the test computer that is connected to the same network.

   - Access from a Web browser

     `http://[Server IP]/index.html`

   - The page displayed is located below

   `[ ~/sdk/libraries/3rdloty/lwip/src/src/apps/http/fs/index.html ]`

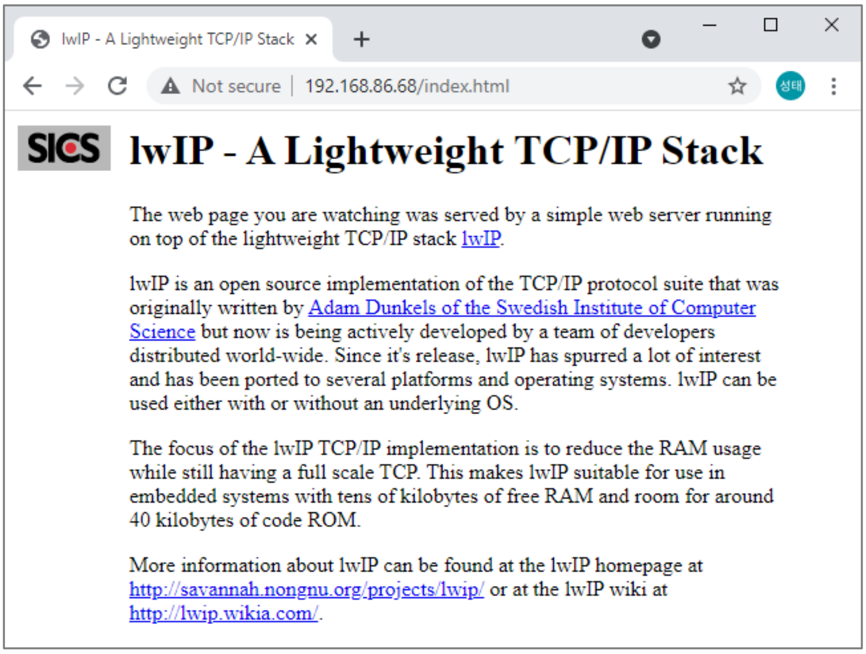


Figure 93. Result of DA16200 HTTP server

For POST, write "/post" at the end of the URL (http://[Server IP]/post).

NOTE
To modify the html page, see the readme.txt file in .\core\libraries\3rdparty\lwip\src\apps\http\makefsdata.
Complete the following steps to create new fsdata.c code suitable for httpd for given html pages.
1. Make sure to install Perl or else install Perl.
2. Run the Git bash or Unix based terminal, or windows command prompt if Cygwin is installed.
3. Navigate to the directory in SDK: .\core\libraries\3rdparty\lwip\src\apps\http\makefsdata

4. Copy makefsdata and rename it (to makefsdata_run for example) and rename the current fs_data.c as well like fs_data.c.ori for future reference/compare when needed.
5. Type this command "perl makefsdata_run", it overwrites the fsdata.c.
6. After the fsdata.c is created, go to the end of the file and add this code manually in all const struct fsdata_file file_*** variables.

   ",FS_FILE_FLAGS_HEADER_INCLUDED | FS_FILE_FLAGS_HEADER_PERSISTENT,"
7. The code before the update is as follows:

   const struct fsdata_file file_404_html[] = {{NULL, data_404_html, data_404_html + 10, sizeof(data_404_html) - 10}};

   const struct fsdata_file file_img_sics_gif[] = {{file_404_html, data_img_sics_gif, data_img_sics_gif + 14, sizeof(data_img_sics_gif) - 14}};

   const struct fsdata_file file_index_html[] = {{file_img_sics_gif, data_index_html, data_index_html + 12, sizeof(data_index_html) - 12}};
8. It must be:

   const struct fsdata_file file_404_html[] = {{NULL, data_404_html, data_404_html + 10, sizeof(data_404_html) - 10, FS_FILE_FLAGS_HEADER_INCLUDED | FS_FILE_FLAGS_HEADER_PERSISTENT,}};

   const struct fsdata_file file_img_sics_gif[] = {{file_404_html, data_img_sics_gif, data_img_sics_gif + 14, sizeof(data_img_sics_gif) - 14, FS_FILE_FLAGS_HEADER_INCLUDED | FS_FILE_FLAGS_HEADER_PERSISTENT,}};

   const struct fsdata_file file_index_html[] = {{file_img_sics_gif, data_index_html, data_index_html + 12, sizeof(data_index_html) - 12, FS_FILE_FLAGS_HEADER_INCLUDED | FS_FILE_FLAGS_HEADER_PERSISTENT,}};
9. Build the SDK and try again.

14.8 WebSocket Client

This section describes the behavior of the example WebSocket Client application and how to build it.

NOTE
WebSocket client does not support DPM mode.

14.8.1 How to Run

1. In the e[2] studio, import a project for the Websocket_Client application.

`~/SDK/apps/common/examples/Network/WebSocket_Client/projects/da16200`

2. To set the WebSocket Server URI in the WebSocket Client Sample, edit the source code:

   `~/SDK/apps/common/examples/Network/WebSocket_Client/src/websocket_client_sample.c`

   #define WEBSOCKET_SERVER_URI "ws(wss)://xxxx.xxxx.xxxx"
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. Use the console to set up the Wi-Fi station interface and connect to the AP that is connected to the Internet.
5. Complete the setup and (re)start the sample.

14.8.2 Sample Code

When the WebSocket client application starts, it tries to connect to a WebSocket Server and send a message 10 times. The URI and data of the example code are for demonstration purposes and can be modified as required to create a custom application. To use the WebSocket Secure connection, enter "wss://" instead of "ws://" in the URI.

The sample code is executed as follows:

1. Set `websocket_cfg.uri` for the WebSocket Server URI and WebSocket initializes with the WebSocket configurations.

```
websocket_cfg.uri = WEBSOCKET_SERVER_URI;
WS_LOGI(TAG, "Connecting to %s...\n", websocket_cfg.uri);
websocket_client_handle_t client = websocket_client_init(&websocket_cfg);
```

2. To receive event data, register *websocket_client_event_callback* function before starting the WebSocket Client.

   `static void ws_event_handler (websocket_client_event_id_t event_id, websocket_client_event_data_t *event_data)`

```
{
      websocket_client_event_data_t *data = (websocket_client_event_data_t *)event_data;

    switch (event_id) {
    case WEBSOCKET_CLIENT_EVENT_CONNECTED:
        WS_LOGW(TAG, "WEBSOCKET_CONNECTED\n");
        break;
    case WEBSOCKET_CLIENT_EVENT_DISCONNECTED:
        WS_LOGW(TAG, "WEBSOCKET_DISCONNECTED\n");
        break;
    case WEBSOCKET_CLIENT_EVENT_DATA:
               …
               …
        if(data->op_code == WS_TRANSPORT_OPCODES_CLOSE) {
               WS_LOGW(TAG, "Websocket Server Closed\n");
               websocket_client_abort_connection(data->client);
        }
        xTimerReset(shutdown_signal_timer, portMAX_DELAY);
        break;
    case WEBSOCKET_CLIENT_EVENT_ERROR:
        WS_LOGE(TAG, "WEBSOCKET_ERROR\n");
        break;
    }
}

websocket_client_start(client, ws_event_handler);
```

3. When the WebSocket Client connects to the server, it sends a message 10 times using `websocket_client_send_text()` API. If no event data is received for 5 seconds, `shutdown_signal_timer` disconnects the WebSocket connection.

```
while (i < 10) {
    if (websocket_client_is_connected(client)) {
        int len = sprintf(data, "hello %04d", i++);
        WS_LOGI(TAG, "Sending %s\n", data);
        websocket_client_send_text(client, data, len, portMAX_DELAY);
    }
    vTaskDelay(1000 / portTICK_PERIOD_MS);
}
```

4. `shutdown_signal_timer` disconnects the WebSocket connection using `the websocket_client_stop()` API.

```
if(websocket_client_stop(client)== WS_OK){
      WS_LOGI(TAG, "Websocket Stopped\n");
    }
```

15. Network Examples: OTA

15.1 Overview

The DA16200/DA16600 provides support for over the air (OTA) firmware update using the HTTP protocol. The chip operates as an HTTP client which can download and update new firmware from an HTTP server.

The DA16200 firmware image set consists of Bootloader (Second bootloader) and RTOS. The boot loader cannot be updated through OTA, but only RTOS. This product allows application programmers to develop an OTA firmware application that uses the OTA APIs. In addition, users can update certificates such as TLS Certificate Key #1 and TLS Certificate Key #2 and support a firmware update of MCU. Users can easily develop these functions using the API provided by the DA16200/DA16600 SDK.

NOTE
When DPM mode is enabled and an OTA (firmware) update is in progress, DPM Sleep mode is paused temporarily due to SFlash write operations. When the firmware update is complete, DPM Sleep mode returns to normal operation.

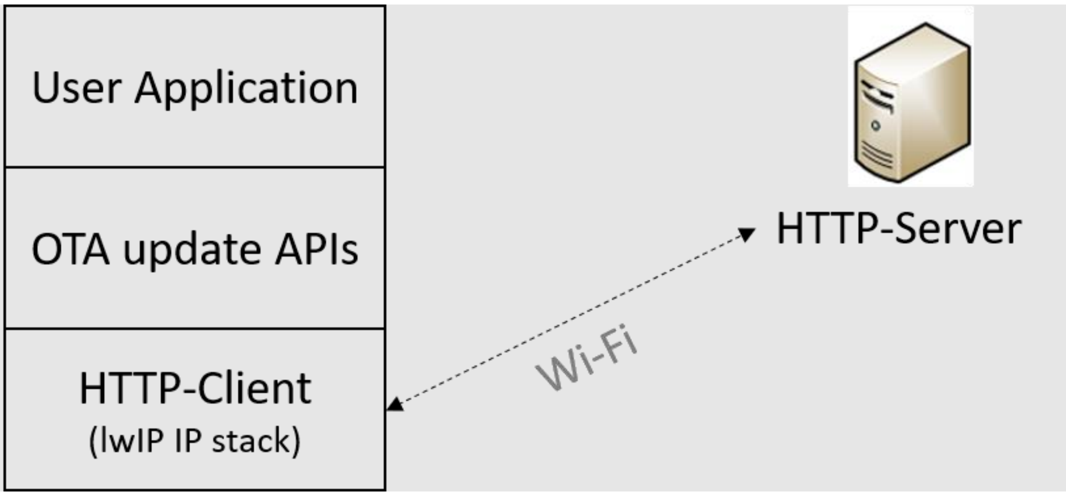


Figure 94. OTA update layer

15.2 SFLASH Memory Area

The DA16200/DA16600 does not support file systems, so the firmware should be stored in SFlash memory. The SFlash is divided into several areas as shown in the Table 33. Among them, the areas that users can directly access are as follows:

- User accessible SFlash areas:
  - RTOS #0
  - RTOS #1
  - TLS Certificate #1
  - TLS Certificate #2
  - User Area

NOTE
Incorrect access to other areas may cause serious failure in the system.

Table 33. 4 MB sflash memory map

Address	Name	Size (kB)
0x0000_0000	2nd Bootloader	136
0x0002_2000	Boot Index	4
0x0002_3000	RTOS #0	1788
0x001E_2000	RTOS #1	1788

<table>
<tr><th>Address</th><th colspan="2">Name</th><th>Size (kB)</th></tr>
<tr><td>0x003A_1000</td><td colspan="2">Reserved Area</td><td>4</td></tr>
<tr><td>0x003A_2000</td><td colspan="2">Debug/RMA Certificate</td><td>4</td></tr>
<tr><td>0x003A_3000</td><td rowspan="4">TLS Certificate #1 (MQTT)</td><td>CA</td><td>4</td></tr>
<tr><td>0x003A_4000</td><td>Cert</td><td>4</td></tr>
<tr><td>0x003A_5000</td><td>Private key</td><td>4</td></tr>
<tr><td>0x003A_6000</td><td>Diffie-Hellmann key</td><td>4</td></tr>
<tr><td>0x003A_7000</td><td rowspan="4">TLS Certificate #2 (HTTPs/OTA)</td><td>CA</td><td>4</td></tr>
<tr><td>0x003A_8000</td><td>Cert</td><td>4</td></tr>
<tr><td>0x003A_9000</td><td>Private key</td><td>4</td></tr>
<tr><td>0x003A_A000</td><td>Diffie-Hellmann key</td><td>4</td></tr>
<tr><td>0x003A_B000</td><td colspan="2">NVRAM #0</td><td>4</td></tr>
<tr><td>0x003A_C000</td><td colspan="2">NVRAM #1 (Backup)</td><td>4</td></tr>
<tr><td>0x003A_D000</td><td colspan="2">User Area (including DA14531 image) (Note 1)</td><td>256</td></tr>
<tr><td>0x003E_D000</td><td rowspan="4">TLS Certificate Key #3 (WPA Enterprise)</td><td>CA</td><td>4</td></tr>
<tr><td>0x003E_E000</td><td>Cert</td><td>4</td></tr>
<tr><td>0x003E_F000</td><td>Private</td><td>4</td></tr>
<tr><td>0x003F_0000</td><td>Diffie-Hellmann key</td><td>4</td></tr>
<tr><td>0x003F_1000</td><td rowspan="4">TLS Certificate Key #4 (Reserved)</td><td>CA</td><td>4</td></tr>
<tr><td>0x003F_2000</td><td>Certificate</td><td>4</td></tr>
<tr><td>0x003F_3000</td><td>Private Key</td><td>4</td></tr>
<tr><td>0x003F_4000</td><td>Diffie-Hellmann key</td><td>4</td></tr>
<tr><td>0x003F_5000</td><td colspan="2">NVRAM FOOTPRINT</td><td>4</td></tr>
<tr><td>0x003F_6000</td><td colspan="2">AT-CMD TLS Certificate Key #0 ~ #9</td><td>40</td></tr>
</table>

Note 1 For DA16600, the DA14531 image is stored in the User Area (0x003A_D000 ~ 0x003C_1FFF). See Ref. [3] for further details.

15.3 HTTP Protocol

The DA16200/DA16600 supports HTTP/HTTPS 1.1 and requests firmware download to the HTTP server by using the GET method of the HTTP client.

The OTA update application should know the URL of the HTTP server before requesting a download. How to obtain the URL depends on the user's preference. When using HTTPS, the DA16200/DA16600 should have at least 36 kB of heap memory for TLS encryption and decryption. The user can print the current memory usage from the terminal.

- CLI commands

```
[/DA16200] # sys.os.heap
[/DA16200] # sys.os.pool
```

- API

```
extern void memoryPoolInfo(void);
extern void   cmd_heapinfo_func(int argc, char *argv[]);
memoryPoolInfo();
cmd_heapinfo_func(0, NULL);
```

15.4 OTA Firmware Update

The OTA firmware update is divided into two stages: **DOWNLOAD** and **RENEW**.

DOWNLOAD refers to the process of downloading the new firmware from the OTA server. In this case, the firmware is not yet applied.

RENEW is the process of applying the downloaded firmware. When the firmware is successfully applied, the new firmware is executed after reboot.

15.4.1 Header

Figure 95 shows DA16200/DA16600 header information as an example. Header information is 96 bytes and is automatically inserted when the firmware is built. The red box in Figure 95 is the magic number and version information. The yellow box is information for checking firmware Cyclic Redundancy Check (CRC). Users only need to check the version information in the red box.

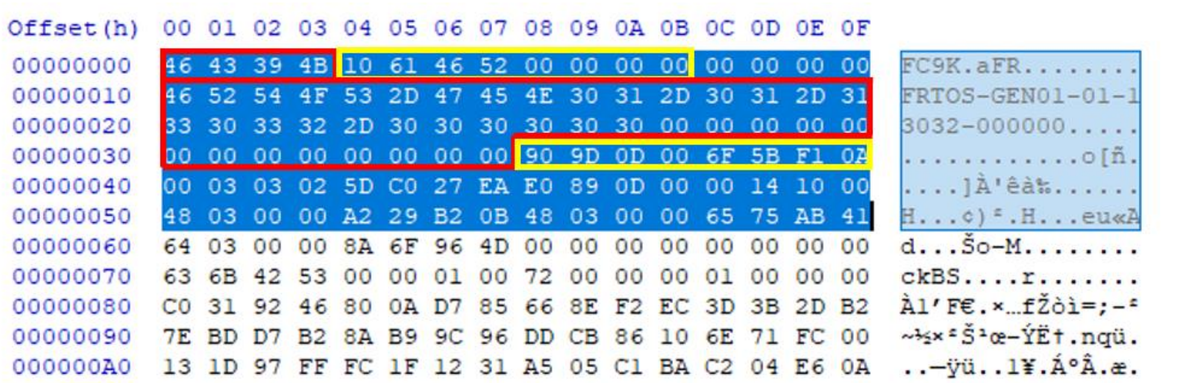


Figure 95. Firmware header information

15.4.2 Version

DA16200/DA16600's RTOS has unique version rules for system protection. The version name is inserted as a string of up to 39 bytes including the separator "-" in the header part of the firmware image at build time.

There are five elements in the version string, separated by "-": Type, Vendor, Major, Minor, and Customer. For example, FRTOS-GEN01-01-12345-000001.

The file name of the firmware does not have to be the same as the version. DA16200/DA16600 only refers to the version inserted in the firmware header.

Version String

Type-Vendor-Major-Minor-Customer

- Type (6 bytes): Identify the type of firmware
- Vendor (6 bytes): Vendor classification
- Major (3 bytes): Major number to check compatibility
- Minor (10 bytes): SDK patch number
- Customer (10 bytes): User configurable version

Type-Vendor-Major determines whether DOWNLOAD or RENEW is compared to the version of firmware currently in operation. **Minor-Customer** can be used by the user for firmware version management.

Users can change the customer version by editing ..\version\3rd_customer_build_num.h. If users change the customer version and build the SDK, the customer version is applied to the image.

15.4.3 Result Code

All APIs provided by OTA update return the result codes as shown in the Table 34. It is delivered through the callback function connected with DOWNLOAD and RENEW APIs.

Table 34. Result code

Result Code	Value	Description
OTA_SUCCESS	0x00	Return success
OTA_FAILED	0x01	Return failed

Result Code	Value	Description
OTA_ERROR_SFLASH_ADDR	0x02	SFlash address is wrong.
OTA_ERROR_TYPE	0x03	Firmware type is unknown.
OTA_ERROR_URL	0x04	Server URL is unknown.
OTA_ERROR_SIZE	0x05	The firmware size is too big.
OTA_ERROR_CRC	0x06	CRC is not correct.
OTA_VERSION_UNKNOWN	0x07	Firmware version is unknown.
OTA_VERSION_INCOMPATI	0x08	The firmware version is incompatible.
OTA_NOT_FOUND	0x09	Firmware was not found on the server.
OTA_NOT_CONNECTED	0x0A	Failed to connect to the server.
OTA_NOT_ALL_DOWNLOAD	0x0B	All new firmware has not been downloaded.
OTA_MEM_ALLOC_FAILED	0x0C	Failed to allocate memory.
OTA_BLE_VERSION_UNKNOWN	0xA1	The Bluetooth® LE firmware version is unknown.

15.4.4 DOWNLOAD

DOWNLOAD is the process of downloading firmware from the OTA server and saving it into the SFlash area.

The communication protocol with the OTA server uses HTTP and can be implemented using the HTTP API supported by lwIP. Therefore, the process of communicating with HTTP-server works the same as lwIP's HTTP Client.

NOTE
Firmware downloads may fail because of HTTP timeout caused by high network latency in low bandwidth environments. To avoid firmware download failure, the HTTP timeout can be set to a higher value than the default. However, the HTTP timeout affects power consumption as the device cannot enter Sleep mode while waiting for the HTTP response. Therefore, using a high HTTP timeout value may increase power consumption. For this reason, the minimum recommended bandwidth for firmware download should be higher than 1 Mbps, although this may vary as the environment and conditions may not be the same. Default HTTP timeout: 6 seconds (`.\core\libraries\3rdparty\lwip\src\apps\http\http_client.c`) `#define HTTPC_POLL_TIMEOUT      30    // 200 ms Interval x 30 times = 6 seconds timeout.`

The download sequence proceeds as follows, and both success and failure results can be delivered through the callback function (see Table 34 for results):

1. Request a query from the HTTP server.
2. Confirm that the response was successfully received from the HTTP server. If the server connection fails or receives a failure response, the download is terminated, and the result is transferred to the callback function. See Table 34 for result values.
3. Check the magic number and version name in the firmware header, and if they do not match, the download is terminated, and the result is transferred to the callback function.
4. If the magic number and version name are normal, the downloaded data is written to SFlash. The SFlash address where the data is written is automatically determined by the boot index (see Section 15.4.5.1). When the download is completed successfully, the entire firmware stored in SFlash has a CRC check.
5. When the CRC check is successfully completed, the result value of 0x00 is transferred to the callback function and the download is terminated.

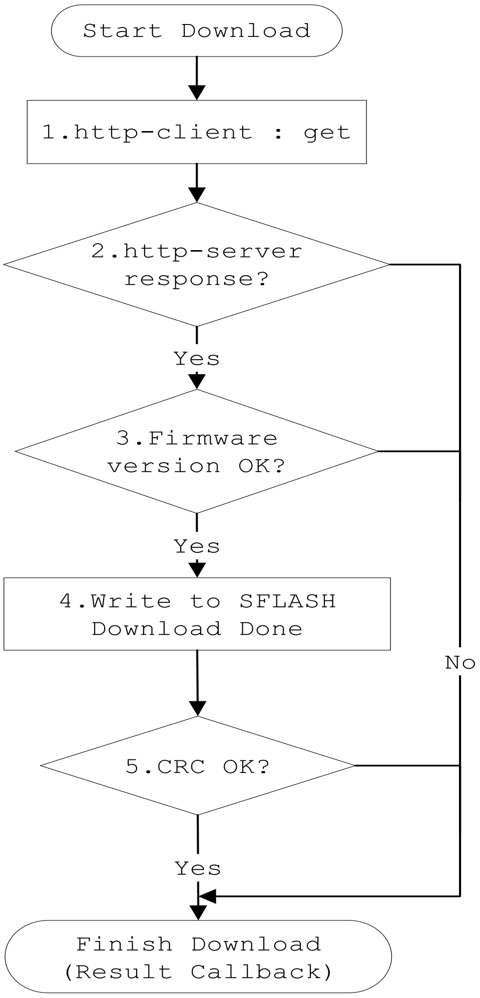


Figure 96. Firmware DOWNLOAD

15.4.5 RENEW

RENEW only operates when the firmware download is successful. DA16200/DA16600 should have the download history after power is on.

1. Check whether the download was successful. After turning on the power, check the download history.
2. Check the CRC of the firmware stored in SFlash. In case of failure, RENEW ends and the result is transmitted to the callback function.
3. Check the firmware version stored in the flash. In case of failure, RENEW ends and the result is transmitted to the callback function.
4. Determine if the new firmware is normal and change the boot index to the new firmware location.

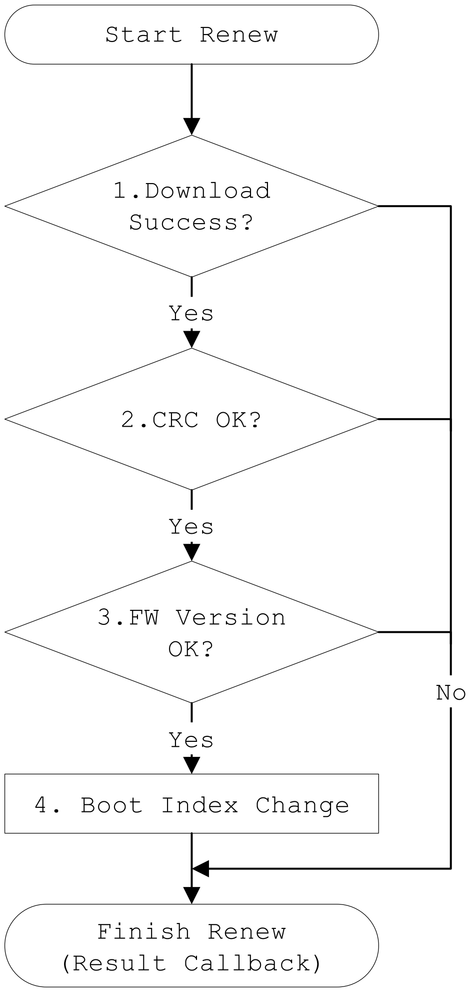


Figure 97. Firmware RENEW

15.4.5.1 Boot Index

The DA16200/DA16600 is divided into firmware download area and current area for OTA firmware update. The two areas are toggled on each other by the boot index. For example, if the boot index value is 0, it operates as the firmware stored in the SFlash RTOS #0 area upon booting, and the newly downloaded firmware is stored in RTOS #1. After that, if RENEW is operated successfully, the boot index value is changed to 1, rebooted, and the firmware stored in the SFlash RTOS#1 area is operated.

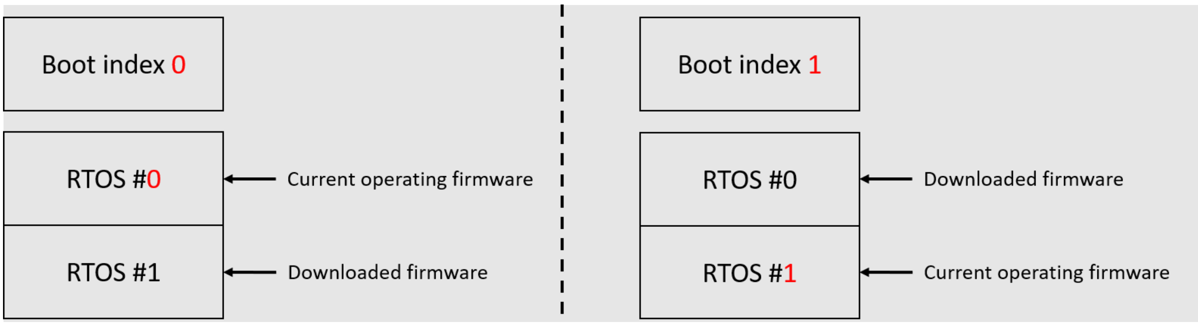


Figure 98. Boot index operation

15.5 API

This section describes the structures and application programming interface (API) required for the OTA firmware update application.

15.5.1 Type

OTA update task is operated based on the type defined in the OTA update type. The operation sequence is tailored to the specified type.

Table 35. OTA update type

Name	ota_update_type
Description	Identify and specify targets for OTA updates.
	/// Operation step of process typedef enum { OTA_TYPE_INIT, // Init value OTA_TYPE_RTOS, // RTOS OTA_TYPE_BLE_FW, // Bluetooth® firmware, for DA166x OTA_TYPE_BLE_COMBO, // RTOS and Bluetooth® firmware, for DA166x OTA_TYPE_MCU_FW, // MCU firmware, not DA16x OTA_TYPE_CERT_KEY, // Certificate or Key OTA_TYPE_UNKNOWN // Unknown value } ota_update_type;

15.5.2 Structure

OTA UPDATE CONFIG sets the necessary parameters when calling OTA firmware update API.

Table 36. OTA update configuration

Name	OTA_UPDATE_CONFIG
Description	Contain information to be passed as argument values to OTA update APIs.
	/// OTA update configuration structure typedef struct { /// Update type. ota_update_type update_type; /// Server address where firmware is located. char url[OTA_HTTP_URL_LEN]; /// Callback function pointer to check the download status. void (*download_notify)(ota_update_type update_type, UINT ret_status, UINT progress); /// Callback function pointer to check the renew state. Only for RTOS. void (*renew_notify)(UINT ret_status); /// If the value is true, if the new firmware download is successful, it reboots with the new firmware. Only for RTOS UINT auto_renew; /// Address of sflash where other_fw is stored. Only for MCU_FW and CERT_KEY UINT download_sflash_addr; #if defined (__BLE_COMBO_REF__) /// Server address where Bluetooth firmware is located. Char url_ble_fw[OTA_HTTP_URL_LEN]; #endif /* __BLE_COMBO_REF__ */ } OTA_UPDATE_CONFIG;

15.5.3 APIs

This section describes the APIs required for the OTA firmware update application.

Table 37. APIs for OTA firmware

Item		Description
UINT ota_update_start_download(OTA_UPDATE_CONFIG *ota_update_conf)		
Parameter	[in] ota_update_conf	The pointer of OTA_UPDATE_CONFIG structure. update_type: Update type. url: Server address where firmware is located. (*download_notify)(ota_update_type update_type, UINT ret_status, UINT progress): Callback function pointer to check the download status. (*renew_notify)(UINT ret_status): Callback function pointer to check the renew state. Only for RTOS. auto_renew: If the value is true, if the new firmware download is successful, it reboots with the new firmware. Only for RTOS. download_sflash_addr: This can set the SFlash address to download MCU_FW, BLE_FW and CERT_KEY excluding RTOS within the User Area range. The default value is 0x003A_D000. url_ble_fw: Server address where Bluetooth firmware is located when (__BLE_COMBO_REF__) is defined.
Return		Return 0x00 on success. See Table 34.
Description		HTTP Client task is created and send a query to the HTTP server. It checks the version compatibility of the firmware received from the server and writes it to the download area of SFlash.
UINT ota_update_stop_download(void)		
Parameter	None	
Return		Return 0x00 on success. See Table 34.
Description		A download can be stopped while downloading from the HTTP server.
UINT ota_update_get_download_progress(ota_update_type update_type)		
Parameter	[in] update_type	Specify the type to be updated.
Return		Return a value between 0 and 100. If the download was successful, it returns 100.
Description		Check the progress while downloading or after completion.
UINT ota_update_start_renew(OTA_UPDATE_CONFIG *ota_update_conf)		
Parameter	[in] ota_update_conf	The pointer of OTA_UPDATE_CONFIG structure.
Return		Return 0x00 on success. See Table 34.
Description		Check the version compatibility and CRC, changes the boot index to the new firmware location, and then reboots automatically.
UINT ota_update_get_new_sflash_addr(UINT update_type)		
Parameter	[in] update_type	Specify the type to be updated.
Return		Return the SFlash address.
Description		The user can know the address of SFlash where the new firmware(data) downloaded from the server is stored.
UINT ota_update_read_flash(UINT addr, VOID *buf, UINT len)		
Parameter	[in] addr [out] buf [in] len	SFlash address (hex). Buffer pointer to store read data. Length to read.
Return		Return 0x00 on success. See Table 34.

Item		Description
Description		Reads SFlash as much as the input address and length.
UINT ota_update_erase_flash(UINT addr, UINT len)		
Parameter	[in] addr [in] len	SFlash address (hex). Length to erase.
Return		Returns erased length.
Description		Erases SFlash as much as the input address and length.
UINT ota_update_copy_flash(UINT dest_addr, UINT src_addr, UINT len)		
Parameter	[in] dest_addr [in] src_addr [in] len	dest_addr Destination Sflash address (hex). src_addr Source Sflash address (hex). Length to copy.
Return		Return 0x00 on success. See Table 34.
Description		Copy as much as the length from SFlash address src_addr to dest_addr.
UINT ota_update_set_mcu_fw_name(char *name)		
Parameter	[in] name	Input the firmware name (version). Maximum 8 bytes.
Return		Return 0x00 on success. See Table 34.
Description		Set the name (version) of MCU firmware to be downloaded to SFlash. If not set, it is set as the default string. /* ota_update.h */ #define OTA_MCU_FW_NAME "MCU_FW"
UINT ota_update_get_mcu_fw_name(char *name)		
Parameter	[out] name	Pointer to get the name (version) of MCU firmware.
Return		Return 0x00 on success. See Table 34.
Description		Get name (version) of MCU firmware downloaded to SFlash.
UINT ota_update_get_mcu_fw_info(char *name, UINT *size, UINT *crc)		
Parameter	[out] name [out] size [out] crc	Pointer to get the name (version) of MCU firmware. Pointer to get the size of MCU firmware. Pointer to get the CRC32 value of MCU firmware.
Return		Return 0x00 on success. See Table 34.
Description		Get name (version), size, and CRC32 of MCU firmware downloaded to SFlash.
UINT ota_update_read_mcu_fw(UINT sflash_addr, UINT size)		
Parameter	[in] sflash_addr [int] size	sflash_addr Start address for reading. Read size.
Return		Return 0x00 on success. See Table 34.
Description		Start transmission of MCU firmware stored in flash through interface as much as the set size.
UINT ota_update_trans_mcu_fw(void)		
Parameter	void	None
Return		Return 0x00 on success. See Table 34.
Description		Start transmission of MCU firmware stored in flash through interface.
UINT ota_update_erase_mcu_fw(void)		
Parameter	void	None

Item		Description
Return		Return 0x00 on success. See Table 34.
Description		Delete MCU firmware saved in SFlash.
UINT ota_update_calcu_mcu_fw_crc(int sflash_addr, int size)		
Parameter	[in] sflash_addr [int] size	sflash_addr CRC calculation start address. CRC calculation size.
Return		Return 0x00 on success. See Table 34.
Description		Calculate CRC32 of MCU firmware stored in SFlash.
UINT ota_update_set_tls_auth_mode_nvram(int tls_auth_mode);		
Parameter	[in] tls_auth_mode	Set the certificate verification mode. #define MBEDTLS_SSL_VERIFY_NONE 0 #define MBEDTLS_SSL_VERIFY_OPTIONAL 1 #define MBEDTLS_SSL_VERIFY_REQUIRED 2
Return		Return 0x00 on success. See Table 34.
Description		Initialize interface to transfer firmware between the DA16200/DA16600 and MCU.

15.5.4 Example

This is an example of the DA16200 firmware update.

1. Make sure to set update type to OTA_TYPE_RTOS.

   /* Setting the type to be updated */

   g_ota_update_conf->update_type = OTA_TYPE_RTOS;

1. Set URL to suit the user environment.

   /* URL setting example - Change it to suit your environment. */

   memcpy(g_ota_update_conf->url, ota_server_url_rtos, strlen(ota_server_url_rtos));

2. If the download completes successfully, the user can set it to automatically activate RENEW.

   g_ota_update_conf->auto_renew = 1;

2. By registering a callback function in download_notify, the user can be notified whether the download succeeds or fails. Users can check whose notification is by update_type.

   g_ota_update_conf->download_notify = user_sample_da16_fw_download_notify;

3. Receive notification about the RENEW status by registering a callback function. If the notification status is successful, the DA16200 automatically reboots after 2-3 seconds.

   g_ota_update_conf->renew_notify = user_sample_da16_fw_renew_notify;

4. Finally, call the OTA update start API. When ota_update_start_download() is called, an OTA update task is created internally, and the creation status of the task is immediately returned. The process is not blocked.

   status = ota_update_start_download(g_ota_update_conf);

3. Reboot the DA16200 when the firmware is updated.

15.5.4.1 Test Command

The DA16200/DA16600 SDK includes sample code and CLI commands to make it easier for users to use the OTA update. It is possible to program directly by referring to the sample code, but the user can simply check the network status with the OTA server by using the CLI command before that.

- Download Example Using CLI Command

```
[/DA16200/NET] # ota_update rtos https://ota-server/NEW_RTOS.img
   > Server FW version: RTOS-GEN01-01-12345-000000
   >> HTTP(s) Client Downloading... 100 % (800000/800000 bytes)
```

```
- OTA Update: <RTOS> Download - Success
[/DA16200/NET] # ota_update renew
```

Table 38. OTA test command

Command		Option	Description
ota_update		[update_type] [url]	Start the firmware download. * update_type rtos: update_type of RTOS cert_key: update_type of cert or key. mcu_fw: update_type of MCU firmware. url: Server URL where firmware exists ex) ota_update rtos http://192.168.0.1/rtos.img
	stop	-	Stop the firmware download. For example, ota_update stop
	renew	-	Change the current firmware to new firmware. For example, ota_update renew
	info	-	Show the firmware information. For example, ota_update info
	crc	[addr]	Check CRC of firmware. For example, ota_update crc 0x1e2000
	read_sflash	[addr] [size]	Read SFlash data. For example, ota_update read_sflash 0x1e2000 128
	copy_sflash	[dst_addr] [src_addr] [size]	Copy from SFlash data src_add to dst_add. For example, ota_update copy_sflash 0x3ad000 0x1e2000 4096
	erase_sflash	[addr] [size]	Erase SFlash data. For example, ota_update erase_sflash 0x3ad000 4096
	set_name_mcu	-	Set the name (version) of MCU firmware to be downloaded to SFlash. For example, ota_update set_name_mcu MCU_FW
	get_name_mcu	-	Get the name (version) of MCU firmware downloaded to SFlash. For example, ota_update get_name_mcu
	read_mcu	-	Read the firmware as much as the size from the read_addr and transmit it. For example, ota_update read_sflash 0x3ad000 4096
	trans_mcu	-	Transmit the firmware to MCU through interface. For example, ota_update trans_mcu
	erase_mcu	-	Erase the firmware stored in a serial flash of the DA16200/DA16600. For example, ota_update erase_mcu
	get_boot_index	-	Get the current boot index information. For example, ota_update get_boot_index
	toggle_boot_index	-	Toggle boot index. For example, ota_update toggle_boot_index

15.5.4.2 Sample Code

The DA16200/DA16600 SDK provides sample code and user guide:

- Sample code

  The sample code includes not only the DA16200/DA16600 firmware update, but also a sample of the MCU firmware and certificate update.

  .\sample\Network\OTA_Update\src\ota_update_sample.c

15.6 OTA Firmware Update – Extensions

The OTA firmware update supports updating not only the DA16200/DA16600 firmware but also the firmware of the MCU chip or the certificate for TLS protocol.

15.6.1 Certificates

To update the SFlash `TLS Certificate #1 and TLS Certificate #2 areas`:

Download directly to SFlash `TLS Certificate #1, TLS Certificate #2`, or User Area and copy to SFlash `TLS Certificate #1 or TLS Certificate #2`.

1. Set URL to suit the user environment.

   /* URL setting example - Change it to suit your environment. */

   memcpy(g_ota_update_conf->url, ota_server_url_cert, strlen(ota_server_url_cert));
2. Make sure to set update_type to OTA_TYPE_CERT_KEY.

   g_ota_update_conf->update_type = OTA_TYPE_CERT_KEY;
3. Set the address of SFlash to be saved when downloading. If not set, the default is SFLASH_USER_AREA_0_START (See Table 33).

   g_ota_update_conf->download_sflash_addr = SFLASH_USER_AREA_0_START;
4. Register a callback to be notified of the download status.

   g_ota_update_conf->download_notify = user_sample_cert_key_download_notify;
5. Finally, call the OTA update and start API. When ota_update_start_download() is called, an OTA update task is created internally, and the creation status of the task is immediately returned. The process is not blocked.

   status = ota_update_start_download(g_ota_update_conf);
6. Copy them to the TLS Certificate Key #0 and TLS Certificate Key #1 areas when downloaded.

   status = ota_update_copy_flash(SFLASH_ROOT_CA_ADDR1, g_ota_update_conf->download_sflash_addr, 4096);

15.6.2 MCU Firmware

To update the firmware of the MCU connected to the DA16200/DA16600 interface,

1. Set URL to suit the user environment.

   /* URL setting example - Change it to suit your environment. */

   memcpy(g_ota_update_conf->url, ota_server_url_mcu, strlen(ota_server_url_mcu));
2. Set update_type to OTA_TYPE_MCU_FW.

   g_ota_update_conf->update_type = OTA_TYPE_MCU_FW;
3. Set the address of SFlash to save when downloading. If not set, the default is SFLASH_USER_AREA_0_START. (See Table 33)

   g_ota_update_conf->download_sflash_addr = SFLASH_USER_AREA_0_START;
4. Register a callback to notify the download status.

   g_ota_update_conf->download_notify = user_sample_mcu_fw_download_notify;
5. Call the OTA update start API. When ota_update_start_download() is called, an OTA update task is created internally, and the creation status of the task is immediately returned. The process is not blocked.

   status = ota_update_start_download(g_ota_update_conf);
6. Transmit the firmware to the MCU when downloaded.

When the transmission API is called, transmit <FW_NAME>, <FW_SIZE>, <FW_CRC> information to the MCU first. Next, transmit the divided buffer size of the entire firmware to the MCU.

That is, transmit "MCU_FW,4096,5aa8b6c4" to the MCU first. Then, transmit 2048 bytes, the divided buffer size of the firmware (4096 divided by 2), to the MCU in sequential order as shown in Figure 99.

NOTE
Buffer size is defined by OTA_MCU_BUF_SIZE. `ota_update_trans_mcu_fw();`

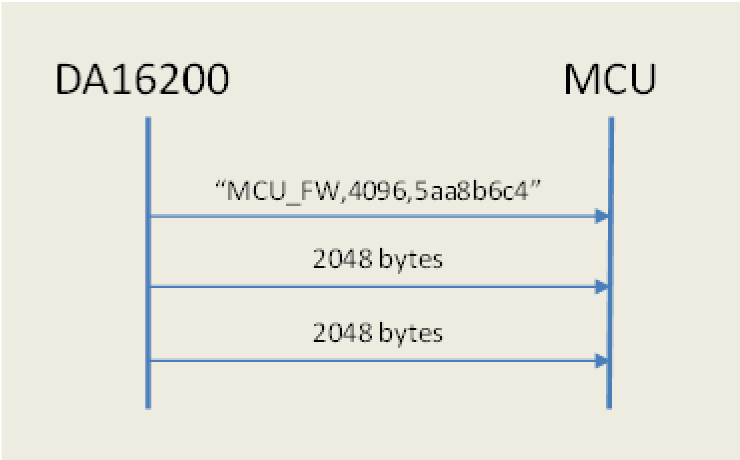


Figure 99. MCU firmware

15.6.2.1 CRC-32 Calculation

This is an example for calculating the CRC value required in the Transfer protocol.

```
static const unsigned int ota_crc_table[] =
{
  0x00000000L, 0x77073096L, 0xee0e612cL, 0x990951baL, 0x076dc419L,
  0x706af48fL, 0xe963a535L, 0x9e6495a3L, 0x0edb8832L, 0x79dcb8a4L,
  0xe0d5e91eL, 0x97d2d988L, 0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L,
  0x90bf1d91L, 0x1db71064L, 0x6ab020f2L, 0xf3b97148L, 0x84be41deL,
  0x1adad47dL, 0x6ddde4ebL, 0xf4d4b551L, 0x83d385c7L, 0x136c9856L,
  0x646ba8c0L, 0xfd62f97aL, 0x8a65c9ecL, 0x14015c4fL, 0x63066cd9L,
  0xfa0f3d63L, 0x8d080df5L, 0x3b6e20c8L, 0x4c69105eL, 0xd56041e4L,
  0xa2677172L, 0x3c03e4d1L, 0x4b04d447L, 0xd20d85fdL, 0xa50ab56bL,
  0x35b5a8faL, 0x42b2986cL, 0xdbbbc9d6L, 0xacbcf940L, 0x32d86ce3L,
  0x45df5c75L, 0xdcd60dcfL, 0xabd13d59L, 0x26d930acL, 0x51de003aL,
  0xc8d75180L, 0xbfd06116L, 0x21b4f4b5L, 0x56b3c423L, 0xcfba9599L,
  0xb8bda50fL, 0x2802b89eL, 0x5f058808L, 0xc60cd9b2L, 0xb10be924L,
  0x2f6f7c87L, 0x58684c11L, 0xc1611dabL, 0xb6662d3dL, 0x76dc4190L,
  0x01db7106L, 0x98d220bcL, 0xefd5102aL, 0x71b18589L, 0x06b6b51fL,
  0x9fbfe4a5L, 0xe8b8d433L, 0x7807c9a2L, 0x0f00f934L, 0x9609a88eL,
  0xe10e9818L, 0x7f6a0dbbL, 0x086d3d2dL, 0x91646c97L, 0xe6635c01L,
  0x6b6b51f4L, 0x1c6c6162L, 0x856530d8L, 0xf262004eL, 0x6c0695edL,
  0x1b01a57bL, 0x8208f4c1L, 0xf50fc457L, 0x65b0d9c6L, 0x12b7e950L,
  0x8bbeb8eaL, 0xfcb9887cL, 0x62dd1ddfL, 0x15da2d49L, 0x8cd37cf3L,
  0xfbd44c65L, 0x4db26158L, 0x3ab551ceL, 0xa3bc0074L, 0xd4bb30e2L,
  0x4adfa541L, 0x3dd895d7L, 0xa4d1c46dL, 0xd3d6f4fbL, 0x4369e96aL,
  0x346ed9fcL, 0xad678846L, 0xda60b8d0L, 0x44042d73L, 0x33031de5L,
  0xaa0a4c5fL, 0xdd0d7cc9L, 0x5005713cL, 0x270241aaL, 0xbe0b1010L,
  0xc90c2086L, 0x5768b525L, 0x206f85b3L, 0xb966d409L, 0xce61e49fL,
  0x5edef90eL, 0x29d9c998L, 0xb0d09822L, 0xc7d7a8b4L, 0x59b33d17L,
  0x2eb40d81L, 0xb7bd5c3bL, 0xc0ba6cadL, 0xedb88320L, 0x9abfb3b6L,
  0x03b6e20cL, 0x74b1d29aL, 0xead54739L, 0x9dd277afL, 0x04db2615L,
  0x73dc1683L, 0xe3630b12L, 0x94643b84L, 0x0d6d6a3eL, 0x7a6a5aa8L,
```

```
   0xe40ecf0bL, 0x9309ff9dL, 0x0a00ae27L, 0x7d079eb1L, 0xf00f9344L,
   0x8708a3d2L, 0x1e01f268L, 0x6906c2feL, 0xf762575dL, 0x806567cbL,
   0x196c3671L, 0x6e6b06e7L, 0xfed41b76L, 0x89d32be0L, 0x10da7a5aL,
   0x67dd4accL, 0xf9b9df6fL, 0x8ebeeff9L, 0x17b7be43L, 0x60b08ed5L,
   0xd6d6a3e8L, 0xa1d1937eL, 0x38d8c2c4L, 0x4fdff252L, 0xd1bb67f1L,
   0xa6bc5767L, 0x3fb506ddL, 0x48b2364bL, 0xd80d2bdaL, 0xaf0a1b4cL,
   0x36034af6L, 0x41047a60L, 0xdf60efc3L, 0xa867df55L, 0x316e8eefL,
   0x4669be79L, 0xcb61b38cL, 0xbc66831aL, 0x256fd2a0L, 0x5268e236L,
   0xcc0c7795L, 0xbb0b4703L, 0x220216b9L, 0x5505262fL, 0xc5ba3bbeL,
   0xb2bd0b28L, 0x2bb45a92L, 0x5cb36a04L, 0xc2d7ffa7L, 0xb5d0cf31L,
   0x2cd99e8bL, 0x5bdeae1dL, 0x9b64c2b0L, 0xec63f226L, 0x756aa39cL,
   0x026d930aL, 0x9c0906a9L, 0xeb0e363fL, 0x72076785L, 0x05005713L,
   0x95bf4a82L, 0xe2b87a14L, 0x7bb12baeL, 0x0cb61b38L, 0x92d28e9bL,
   0xe5d5be0dL, 0x7cdcefb7L, 0x0bdbdf21L, 0x86d3d2d4L, 0xf1d4e242L,
   0x68ddb3f8L, 0x1fda836eL, 0x81be16cdL, 0xf6b9265bL, 0x6fb077e1L,
   0x18b74777L, 0x88085ae6L, 0xff0f6a70L, 0x66063bcaL, 0x11010b5cL,
   0x8f659effL, 0xf862ae69L, 0x616bffd3L, 0x166ccf45L, 0xa00ae278L,
   0xd70dd2eeL, 0x4e048354L, 0x3903b3c2L, 0xa7672661L, 0xd06016f7L,
   0x4969474dL, 0x3e6e77dbL, 0xaed16a4aL, 0xd9d65adcL, 0x40df0b66L,
   0x37d83bf0L, 0xa9bcae53L, 0xdebb9ec5L, 0x47b2cf7fL, 0x30b5ffe9L,
   0xbdbdf21cL, 0xcabac28aL, 0x53b39330L, 0x24b4a3a6L, 0xbad03605L,
   0xcdd70693L, 0x54de5729L, 0x23d967bfL, 0xb3667a2eL, 0xc4614ab8L,
   0x5d681b02L, 0x2a6f2b94L, 0xb40bbe37L, 0xc30c8ea1L, 0x5a05df1bL,
   0x2d02ef8dL
};

/* update the CRC on the data block one byte at a time */
static unsigned int update_crc (unsigned int init, const unsigned char *buf, int len)
{
      unsigned int crc = init;
      while (len--)
       crc = ota_crc_table[(crc ^ *(buf++)) & 0xFF] ^ (crc >> 8);
      return ~crc;
}
```

15.7 Bluetooth® LE Firmware Update OTA

After building the code of the DA14531 SDK, the following images are available to update DA14531 firmware through OTA.

The DA14531 SDK:

[DA16600 SDK ROOT]\utility\combo\da14531_sdk_v_6.0.14.1114.zip

- The Bluetooth® OTA firmware images for the DA16600 examples (after code build):
  - IoT Sensor gateway example (central example)

    [DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_monitor_aux_ext_coex\Keil_5\out_img\px**m**_coex_ext_531_6_0_14_1_ota.img.
  - Rest of the DA16600 examples (peripheral examples)

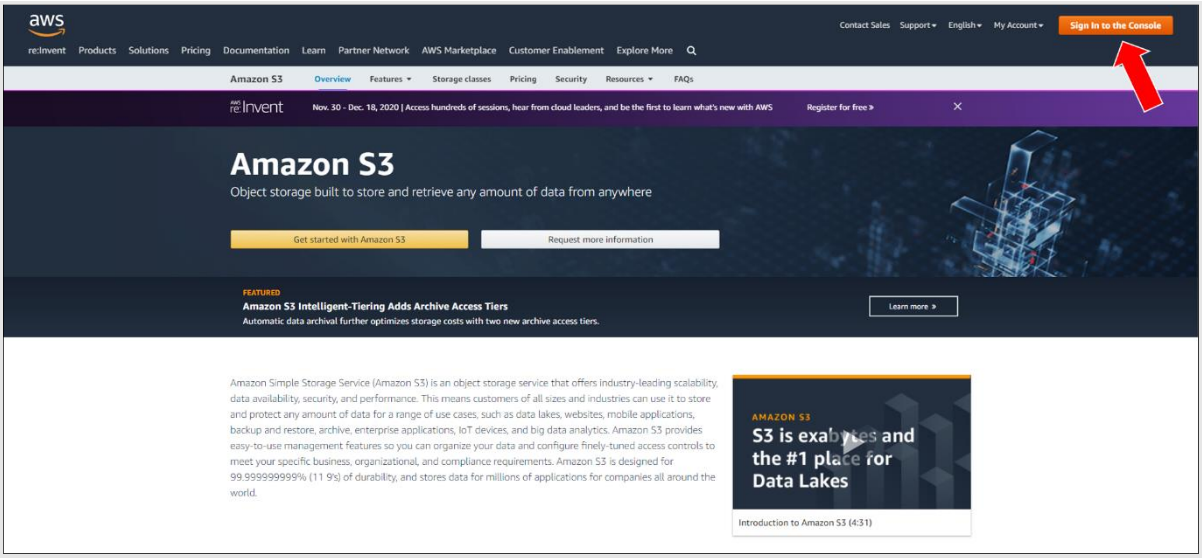
    [DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex\Keil_5\out_img\px**r**_sr_coex_ext_531_6_0_14_1114_1_**ota**.img.

15.8 OTA Test Server

OTA update complies with HTTP protocol to download firmware. Therefore, users can easily implement an OTA server using HTTP-server. This manual does not provide a guide on configuring OTA servers. However, it explains how to configure a simple test environment for functional testing in the application development stage on the cloud environment.

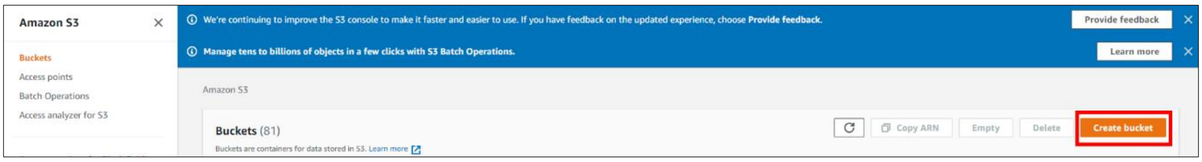
Amazon Simple Storage Service (Amazon S3) is recommended for the OTA test server.

1. Sign up for an AWS account and log in to the console.

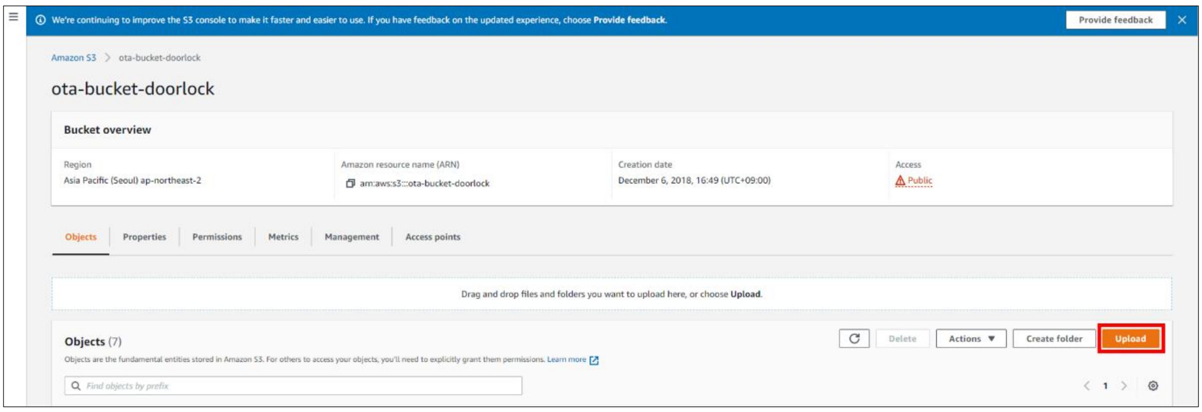


2. In the AWS console, go to **Storage** and choose **S3**.

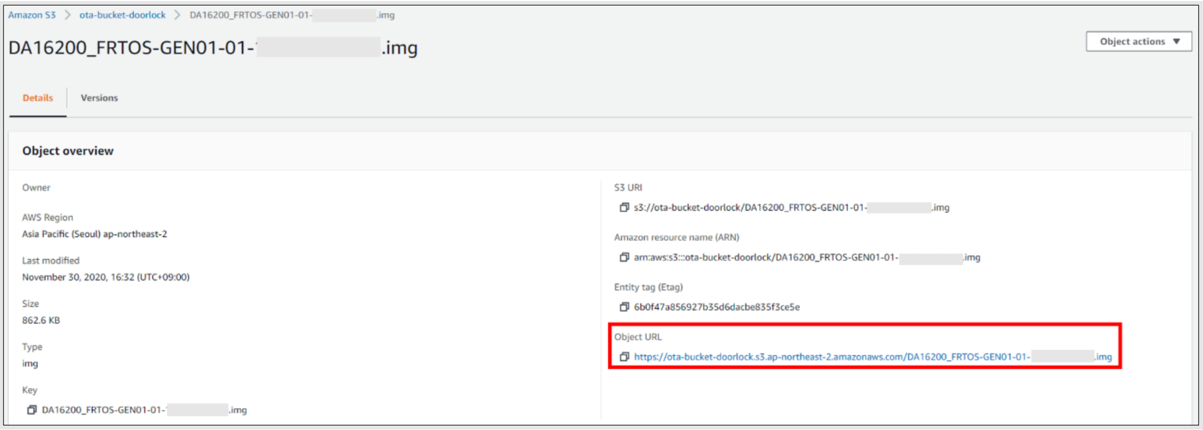
3. To create a bucket with default settings, click **Create Bucket**.



4. Upload the firmware to the created bucket.



5. Check the URL (https://) of the uploaded firmware.



6. Set the URL as the OTA update API parameter value and proceed with the test.

16. Crypto Examples

16.1 Crypto API

This section describes how it is built and works. The Crypto API sample application demonstrates common use cases of cryptographic algorithms such as AES, DES, and Hash. The DA16200 SDK includes an **mbedTLS** library which is an implementation of the TLS and SSL protocols and the respective cryptographic algorithms.

16.1.1 How to Run

1. In the e² studio, import a project for the Crypto API sample application.
   `~/SDK/apps/common/examples/Crypto/Crypto_API/projects/da16200`
2. Enable features of what cryptographic algorithms are required.
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.

16.1.2 How to Enable Cryptographic Algorithm

The Crypto API sample application includes 11 types of cryptographic algorithms. Each type can be enabled by feature definition in `crypto_sample.h` file as follows. By default, AES cryptographic algorithms are enabled.

- AES Algorithms
- Cipher API
- DES Algorithms
- Diffie-Hellman Key Exchange
- DRBG
- ECDH
- ECDSA
- HASH and HMAC Algorithms
- Key Derivation Function
- Public Key Abstraction Layer
- RSA PKCS#1.

```
// AES Algorithms
#define __CRYPTO_SAMPLE_AES__

// Cipher API
#undef  __CRYPTO_SAMPLE_CIPHER__

// DES Algorithms
#undef  __CRYPTO_SAMPLE_DES__

// Diffie-Hellman key exchange
#undef  __CRYPTO_SAMPLE_DHM__

// DRBG
#undef  __CRYPTO_SAMPLE_DRBG__

// ECDH
#undef  __CRYPTO_SAMPLE_ECDH__

// ECDSA
#undef  __CRYPTO_SAMPLE_ECDSA__

// Hash & HMAC Algorithms
#undef  __CRYPTO_SAMPLE_HASH__

// Key Derivation Function
#undef  __CRYPTO_SAMPLE_KDF__
```

```
// Public Key abstraction layer.
#undef  __CRYPTO_SAMPLE_PK__

// RSA PKCS#1
#undef  __CRYPTO_SAMPLE_RSA__
```

16.1.3 Cryptographic Algorithms – AES

The AES algorithm sample application demonstrates common use cases of AES ciphers such as CBC, CFB, and ECB. The sample application runs five types of cryptographic algorithms:

- AES-CBC-128, 192, and 256
- AES-CFB128-128, 192, and 256
- AES-ECB-128, 192, and 256
- AES-ECB-128, 192, and 256
- AES-CTR-128
- AES-CCM

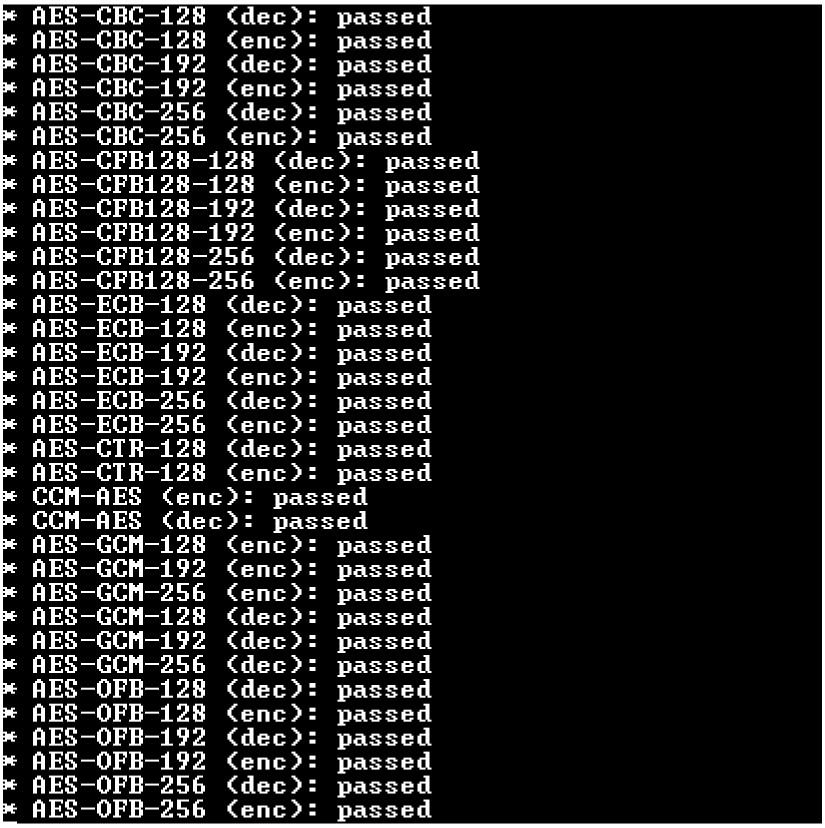


Figure 100. Results of crypto AES

16.1.3.1 Application Initialization

The following example describes how the user uses the AES algorithms of the **mbedTLS** library to encrypt and decrypt data.

```
void crypto_sample_aes(void *param)
{
#if defined(MBEDTLS_CIPHER_MODE_CBC)
    crypto_sample_aes_cbc();
#endif // (MBEDTLS_CIPHER_MODE_CBC)

#if defined(MBEDTLS_CIPHER_MODE_CFB)
```

```
    crypto_sample_aes_cfb();
#endif // (MBEDTLS_CIPHER_MODE_CFB)

    crypto_sample_aes_ecb();

#if defined(MBEDTLS_CIPHER_MODE_CTR)
    crypto_sample_aes_ctr();
#endif // (MBEDTLS_CIPHER_MODE_CTR)

    crypto_sample_aes_ccm();
    crypto_sample_aes_gcm();

#if defined(MBEDTLS_CIPHER_MODE_OFB)
    crypto_sample_aes_ofb();
#endif // (MBEDTLS_CIPHER_MODE_OFB)

    return ;
}
```

16.1.3.2 AES-CBC-128, 192, and 256

The DA16200 supports cryptographic algorithms for AES-CBC-128, 192, and 256. To explain how AES-CBC works, see the test vector in http://csrc.nist.gov/archive/aes/rijndael/rijndael-vals.zip.

```
int crypto_sample_aes_cbc()
{
    mbedtls_aes_context *ctx = NULL;

    // Initialize the AES context.
    mbedtls_aes_init(ctx);

    for (i = 0; i < 6; i++) {
        u = i >> 1;
        v = i  & 1;

        PRINTF("* AES-CBC-%3d (%s): ", 128 + u * 64,
               (v == MBEDTLS_AES_DECRYPT) ? "dec" : "enc");

        if (v == MBEDTLS_AES_DECRYPT) {
            // Set the decryption key.
            mbedtls_aes_setkey_dec(ctx, key, 128 + u * 64);

            // Performs an AES-CBC decryption operation on full blocks.
            for (j = 0; j < CRYPTO_SAMPLE_AES_LOOP_COUNT ; j++) {
                mbedtls_aes_crypt_cbc(ctx, v, 16, iv, buf, buf);
            }
        } else {
            // Set the encryption key.
            mbedtls_aes_setkey_enc(ctx, key, 128 + u * 64);

            // Performs an AES-CBC encryption operation on full blocks.
            for (j = 0 ; j < CRYPTO_SAMPLE_AES_LOOP_COUNT ; j++) {
                unsigned char tmp[16] = {0x00,};
                mbedtls_aes_crypt_cbc(ctx, v, 16, iv, buf, buf);

                memcpy(tmp, prv, 16);
                memcpy(prv, buf, 16);
                memcpy(buf, tmp, 16);
            }
        }
    }
```

```
    // Clear the AES context.
    mbedtls_aes_free(ctx);
}
```

The mbedtls_aes_context is the AES context-type definition to use the AES algorithm. It is initialized by function mbedtls_aes_init. Function mbedtls_aes_crypt_cbc does an AES-CBC encryption or decryption operation on full blocks. And it does the operation defined in the mode parameter (encrypt/decrypt), on the input data buffer defined in the input parameter. To do encryption or decryption, the function mbedtls_aes_setkey_enc or mbedtls_aes_setkey_dec should be called first. After the operation is complete, the function mbedtls_aes_free should be called to clear the AES context.

16.1.3.3 AES-CFB128-128, 192, and 256

The DA16200 supports a cryptographic algorithm for AES-CFB128-128, 192, and 256. To explain how AES-CFB128 works, see the test vector in http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

```
int crypto_sample_aes_cfb()
{
    mbedtls_aes_context *ctx = NULL;

    // Initialize the AES context.
    mbedtls_aes_init(ctx);

    for (i = 0; i < 6; i++) {
        u = i >> 1;
        v = i  & 1;

        PRINTF("* AES-CFB128-%3d (%s): ", 128 + u * 64,
                 (v == MBEDTLS_AES_DECRYPT) ? "dec" : "enc");

        // Set the key.
        mbedtls_aes_setkey_enc(ctx, key, 128 + u * 64);

        if (v == MBEDTLS_AES_DECRYPT) {
            // Perform an AES-CFB128 decryption operation.
            mbedtls_aes_crypt_cfb128(ctx, v, 64, &offset, iv, buf, buf);
        } else {
            // Perform an AES-CFB128 encryption operation.
            mbedtls_aes_crypt_cfb128(ctx, v, 64, &offset, iv, buf, buf);
        }
    }

    // Clear the AES context.
    mbedtls_aes_free(ctx);
}
```

The mbedtls_aes_context is the AES context-type definition to use the AES algorithm. It is initialized by function mbedtls_aes_init. Function mbedtls_aes_crypt_cfb128 does AES-CFB128 encryption or decryption. And it does the operation defined in the mode parameter (encrypt or decrypt) on the input data buffer defined in the input parameter. For CFB, the user should set up the context with function mbedtls_aes_setkey_enc, regardless of whether to encrypt or decrypt operations, that is, regardless of the mode parameter. This is because CFB mode uses the same key schedule for encryption and decryption. After the operation is complete, the function mbedtls_aes_free should be called to clear the AES context.

16.1.3.4 AES-ECB-128, 192, and 256

The DA16200 supports cryptographic algorithms for AES-ECB-128, 192, and 256. To explain how AES-ECB works, see the test vector in http://csrc.nist.gov/archive/aes/rijndael/rijndael-vals.zip.

```
int crypto_sample_aes_ecb()
{
    mbedtls_aes_context *ctx = NULL;
```

```
    // Initialize the AES context.
    mbedtls_aes_init(ctx);

    for (i = 0; i < 6; i++) {
        u = i >> 1;
        v = i  & 1;

        PRINTF("* AES-ECB-%3d (%s): ", 128 + u * 64,
                 (v == MBEDTLS_AES_DECRYPT) ? "dec" : "enc");

        if (v == MBEDTLS_AES_DECRYPT) {
            // Set the decryption key.
            mbedtls_aes_setkey_dec(ctx, key, 128 + u * 64);

            // Perform an AES single-block decryption operation.
            for (j = 0 ; j < CRYPTO_SAMPLE_AES_LOOP_COUNT ; j++) {
                mbedtls_aes_crypt_ecb(ctx, v, buf, buf);
            }
        } else {
            // Set the encryption key.
            mbedtls_aes_setkey_enc(ctx, key, 128 + u * 64);

            // Perform an AES single-block encryption operation.
            for (j = 0 ; j < CRYPTO_SAMPLE_AES_LOOP_COUNT ; j++) {
                mbedtls_aes_crypt_ecb(ctx, v, buf, buf);
            }
        }
    }

    // Clear the AES context.
    mbedtls_aes_free(ctx);
}
```

The mbedtls_aes_context is the AES context-type definition to use the AES algorithm. It is initialized by function mbedtls_aes_init. Function mbedtls_aes_crypt_ecb does an AES single-block encryption or decryption operation. And it does the operation defined in the mode parameter (encrypt or decrypt) on the input data buffer defined in the input parameter. Function mbedtls_aes_init and either function mbedtls_aes_setkey_enc function or function mbedtls_aes_setkey_dec should be called before the first call to this API with the same context. After the operation is complete, the function mbedtls_aes_free should be called to clear the AES context.

16.1.3.5 AES-CTR-128

The DA16200 supports cryptographic algorithms for AES-CTR-128. To explain how AES-CTR works, see the Test Vectors section in http://www.faqs.org/rfcs/rfc3686.html.

```
int crypto_sample_aes_ctr()
{
    mbedtls_aes_context *ctx = NULL;

    // Initialize the AES context.
    mbedtls_aes_init(ctx);

    for (i = 0; i < 2; i++) {
        v = i  & 1;

        PRINTF("* AES-CTR-128 (%s): ",
                    (v == MBEDTLS_AES_DECRYPT) ? "dec" : "enc");

        // Set the key.
        mbedtls_aes_setkey_enc(ctx, key, 128);
```

```
        if (v == MBEDTLS_AES_DECRYPT) {
            // Perform an AES-CTR decryption operation.
            mbedtls_aes_crypt_ctr(ctx, len, &offset,
                                  nonce_counter, stream_block, buf, buf);

        } else {
            // Perform an AES-CTR encryption operation.
            mbedtls_aes_crypt_ctr(ctx, len, &offset,
                                  nonce_counter, stream_block, buf, buf);
        }
    }

    // Clear the AES context.
    mbedtls_aes_free(ctx);
}
```

The mbedtls_aes_context is the AES context-type definition to use the AES algorithm. It is initialized by function mbedtls_aes_init. Function mbedtls_aes_crypto_ctr does an AES-CTR encryption or decryption operation. And it does the operation defined in the mode parameter (encrypt/decrypt) on the input data buffer, defined in the input parameter. Use the same key schedule for both encryption and decryption operations because of the nature of CTR. Therefore, use the context initialized with function mbedtls_aes_setkey_enc for both MBEDTLS_AES_ENCRYPT and MBEDTLS_AES_DECRYPT. After the operation is complete, call function mbedtls_aes_free to clear the AES context.

16.1.3.6 AES-CCM-128, 192, and 256

The DA16200 supports cryptographic algorithms for AES-CCM-128, 192, and 256. To explain how AES-CCM works, see the test vector in SP800-38C Appendix C #1.

```
int crypto_sample_aes_ccm()
{
    mbedtls_ccm_context *ctx = NULL;

    // Initialize the CCM context
    mbedtls_ccm_init(ctx);

    /* Initialize the CCM context set in the ctx parameter
    * and sets the encryption key.
    */

    ret = mbedtls_ccm_setkey(ctx, MBEDTLS_CIPHER_ID_AES,
                             crypto_sample_ccm_key,
                             8 * sizeof(crypto_sample_ccm_key));
    PRINTF("* CCM-AES (enc): ");

    // Encrypt a buffer using CCM.
    ret = mbedtls_ccm_encrypt_and_tag(ctx, crypto_sample_ccm_msg_len,
                                      crypto_sample_ccm_iv, crypto_sample_ccm_iv_len,
                                      crypto_sample_ccm_ad, crypto_sample_ccm_add_len,
                                      crypto_sample_ccm_msg, out,
                                      out + crypto_sample_ccm_msg_len,
                                      crypto_sample_ccm_tag_len);

    PRINTF("* CCM-AES (dec): ");

    // Perform a CCM* authenticated decryption of a buffer.
    ret = mbedtls_ccm_auth_decrypt(ctx, crypto_sample_ccm_msg_len,
                               crypto_sample_ccm_iv, crypto_sample_ccm_iv_len,
                               crypto_sample_ccm_ad, crypto_sample_ccm_add_len,
                               crypto_sample_ccm_res, out,
                               crypto_sample_ccm_res + crypto_sample_ccm_msg_len,
                               crypto_sample_ccm_tag_len);
```

```
    // Clear the CCM context.
    mbedtls_ccm_free(ctx);
}
```

The mbedtls_ccm_context is the CCM context-type definition for the CCM authenticated encryption mode for block ciphers. It is initialized by function mbedtls_ccm_init. Function mbedtls_ccm_setkey initializes the CCM context set in the ctx parameter and sets the encryption key. Function mbedtls_ccm_encrypt_and_tag encrypts a buffer with CCM. And function mbedtls_ccm_auth_decrypt does CCM-authenticated decryption of a buffer. After the operation is complete, call function mbed_ccm_free to release and clear the specified CCM context and underlying cipher subcontext.

16.1.3.7 AES-GCM-128, 192, and 256

The DA16200 supports cryptographic algorithms for AES-GCM-128, 192, and 256. To explain how AES-GCM works, see the test vector in the GCM test vectors of CSRC (http://csrc.nist.gov/groups/STM/cavp/documents/mac/gcmtestvectors.zip).

```
int crypto_sample_aes_gcm()
{
    //The GCM context structure.
    mbedtls_gcm_context *ctx = NULL;
    mbedtls_cipher_id_t cipher = MBEDTLS_CIPHER_ID_AES;

    // Initialize the specified GCM context.
    mbedtls_gcm_init(ctx);

    // AES-GCM Encryption Test
    for (j = 0; j < 3; j++) {
        int key_len = 128 + 64 * j;

        PRINTF("* AES-GCM-%3d (%s): ", key_len, "enc");

        // Associate a GCM context with a cipher algorithm and a key.
        mbedtls_gcm_setkey(ctx, cipher, crypto_sample_gcm_key, key_len);

        // Perform GCM encryption of a buffer.
        ret = mbedtls_gcm_crypt_and_tag(ctx, MBEDTLS_GCM_ENCRYPT,
                                   sizeof(crypto_sample_gcm_pt),
                                   crypto_sample_gcm_iv, sizeof(crypto_sample_gcm_iv),
                                   crypto_sample_gcm_additional,
                                   sizeof(crypto_sample_gcm_additional),
                                   crypto_sample_gcm_pt, buf,
                                   16, tag_buf);

        // Clear a GCM context and the underlying cipher sub-context.
        mbedtls_gcm_free(ctx);
    }

    //AES-GCM Decryption Test
    for (j = 0; j < 3; j++) {
        int key_len = 128 + 64 * j;

        PRINTF("* AES-GCM-%3d (%s): ", key_len, "dec");

        // Associate a GCM context with a cipher algorithm and a key.
        mbedtls_gcm_setkey(ctx, cipher, crypto_sample_gcm_key, key_len);

        // Perform GCM decryption of a buffer.
        ret = mbedtls_gcm_crypt_and_tag(ctx, MBEDTLS_GCM_DECRYPT,
                            sizeof(crypto_sample_gcm_pt),
                                   crypto_sample_gcm_iv, sizeof(crypto_sample_gcm_iv),
```

```
                                crypto_sample_gcm_additional,
                                sizeof(crypto_sample_gcm_additional),
                                crypto_sample_gcm_ct[j], buf,
                                16, tag_buf);

            // Clear a GCM context and the underlying cipher sub-context.
            mbedtls_gcm_free(ctx);
        }
}
```

The mbedtls_gcm_context is the GCM context-type definition. It is initialized by function mbedtls_gcm_init. Function mbedtls_gcm_setkey associates a GCM context with a cipher algorithm (AES) and a key. Function mbedtls_gcm_crypt_and_tag does GCM encryption or decryption of a buffer by the second parameter. After the operation is complete, the function mbed_gcm_free should be called to clear a GCM context and underlying cipher sub-context.

16.1.3.8 AES-OFB-128, 192, and 256

The DA16200 supports cryptographic algorithms for AES-OFB-128, 192, and 256. To explain how AES-OFB works, see the test vector in the OFB test vectors of CSRC (https://csrc.nist.gov/publications/detail/sp/800-38a/final).

```
int crypto_sample_aes_ofb()
{
    mbedtls_aes_context *ctx = NULL;

    // Initialize the AES context.
    mbedtls_aes_init(ctx);

    // Test OFB mode
    for (i = 0; i < 6; i++) {
        PRINTF("* AES-OFB-%3d (%s): ", keybits,
                 (v == MBEDTLS_AES_DECRYPT) ? "dec" : "enc");

        memcpy(iv, crypto_sample_aes_ofb_iv, 16);
        memcpy(key, crypto_sample_aes_ofb_key[u], keybits / 8);

        // Set the encryption key.
        ret = mbedtls_aes_setkey_enc(ctx, key, keybits);

        if (v == MBEDTLS_AES_DECRYPT) {
            memcpy(buf, crypto_sample_aes_ofb_ct[u], 64);
            expected_out = crypto_sample_aes_ofb_pt;
        } else {
            memcpy(buf, crypto_sample_aes_ofb_pt, 64);
            expected_out = crypto_sample_aes_ofb_ct[u];
        }

        // Perform an AES-OFB (Output Feedback Mode) encryption or decryption
        // operation.
        ret = mbedtls_aes_crypt_ofb(ctx, 64, &offset, iv, stream_block, buf, buf);
    }

    // Clear the AES context.
    mbedtls_aes_free(ctx);
}
```

The mbedtls_aes_context is the AES context-type definition to use the AES algorithm. It is initialized by mbedtls_aes_init. Function mbedtls_aes_crypt_ofb does an AES-OFB (Output Feedback Mode) encryption or decryption operation. For OFB, the user should set up the context with the function mbedtls_aes_setkey_enc, regardless of whether the user does an encryption or decryption operation. This is because OFB mode uses the same key schedule for encryption and decryption. The OFB operation is identical for encryption or decryption, therefore no operation mode needs to be specified. After the operation is complete, call function mbedtls_aes_free to clear the AES context.

16.1.4 Cryptographic Algorithms – DES

The DES algorithm sample application demonstrates common use cases of DES and Triple-DES ciphers. The sample application runs two types of cryptography algorithms:

- DES-CBC-56
- DES3-CBC-112 and 168.

Figure 101. Result of crypto DES

16.1.4.1 Application Initialization

The following example shows how to use DES algorithms of the **mbedTLS** library to encrypt and decrypt data.

```
void crypto_sample_des(void *param)
{
#if defined(MBEDTLS_CIPHER_MODE_CBC)
    crypto_sample_des_cbc();
#endif // (MBEDTLS_CIPHER_MODE_CBC)
    return ;
}
```

16.1.4.2 DES-CBC-56, DES3-CBC-112, and 168

The DA16200 supports cryptographic algorithms for DES-CBC-56, DES3-CBC-112, and 168.

```
int crypto_sample_des_cbc()
{
    mbedtls_des_context *ctx = NULL;
    mbedtls_des3_context *ctx3 = NULL;

    // Initialize the DES context.
    mbedtls_des_init(ctx);

    // Initialize the Triple-DES context.
    mbedtls_des3_init(ctx3);

    // Test CBC
    for (i = 0; i < 6; i++) {
        u = i >> 1;
        v = i  & 1;

        PRINTF("* DES%c-CBC-%3d (%s): ",
                 ( u == 0 ) ? ' ' : '3', 56 + u * 56,
                 ( v == MBEDTLS_DES_DECRYPT ) ? "dec" : "enc" );


        switch (i) {
            case 0: {
                // DES key schedule (56-bit, decryption).
                mbedtls_des_setkey_dec(ctx, crypto_sample_des3_keys);
            }
            break;
            case 1: {
```

```
                // DES key schedule (56-bit, encryption).
                mbedtls_des_setkey_enc(ctx, crypto_sample_des3_keys);
            }
            break;
            case 2: {
                // Triple-DES key schedule (112-bit, decryption).
                mbedtls_des3_set2key_dec(ctx3, crypto_sample_des3_keys);
            }
            break;
            case 3: {
                // Triple-DES key schedule (112-bit, encryption).
                mbedtls_des3_set2key_enc(ctx3, crypto_sample_des3_keys);
            }
            break;
            case 4: {
                // Triple-DES key schedule (168-bit, decryption).
                mbedtls_des3_set3key_dec(ctx3, crypto_sample_des3_keys);
            }
            break;
            case 5: {
                // Triple-DES key schedule (168-bit, encryption).
                mbedtls_des3_set3key_enc(ctx3, crypto_sample_des3_keys);
            }
            break;
        }

        if (v == MBEDTLS_DES_DECRYPT) {
            for (j = 0 ; j < CRYPTO_SAMPLE_DES_LOOP_COUNT ; j++) {
                if (u == 0) {
                    // DES-CBC buffer decryption.
                    mbedtls_des_crypt_cbc(ctx, v, 8, iv, buf, buf);
                } else {
                    // 3DES-CBC buffer decryption.
                    mbedtls_des3_crypt_cbc(ctx3, v, 8, iv, buf, buf);
                }
            }
        } else {
            for (j = 0; j < CRYPTO_SAMPLE_DES_LOOP_COUNT; j++) {
                if (u == 0) {
                    // DES-CBC buffer encryption.
                    mbedtls_des_crypt_cbc(ctx, v, 8, iv, buf, buf);
                } else {
                    // 3DES-CBC buffer encryption.
                    mbedtls_des3_crypt_cbc(ctx3, v, 8, iv, buf, buf);
                }
            }
        }
    }

    // Clear the DES context.
    mbedtls_des_free(ctx);

    // Clear the Triple-DES context.
    mbedtls_des3_free(ctx3);
}
```

The mbedtls_des_context is the DES context structure. It is initialized by function mbedtls_des_init. Function mbedtls_des_crypt_cbc does DES-CBC buffer encryption and decryption. Before that, the key should be set up by function mbedtls_des_setkey_enc. After the operation is complete, call function mbed_des_free to clear the DES context.

The mbedtls_des3_context is the Triple-DES context structure. It is initialized by function mbedtls_des3_init. There are two key-sizes supported: 112 bits and 168 bits. Based on the key-size, the key is set up through mbedtls_des3_set2key_enc (or mbedtls_des3_set2key_dec) or mbedtls_des3_set3key_enc(or mbedtls_des3_set3key_dec). After that, the function mbedtls_des3_crypt_cbc does Triple-DES CBC encryption and decryption. After the operation is complete, call function mbedtls_des3_free to clear the DES3 context.

16.1.5 Cryptographic Algorithms – HASH and HMAC

The HASH and HMAC algorithms sample application demonstrates common use cases of HASH and HMAC algorithms such as SHA-1, SHA-256, and SHA-512. The sample application runs six types of hash algorithms and HMAC algorithms:

- SHA1, SHA-224, SHA-256, SHA-384, SHA-512, and MD5
- HMAC.

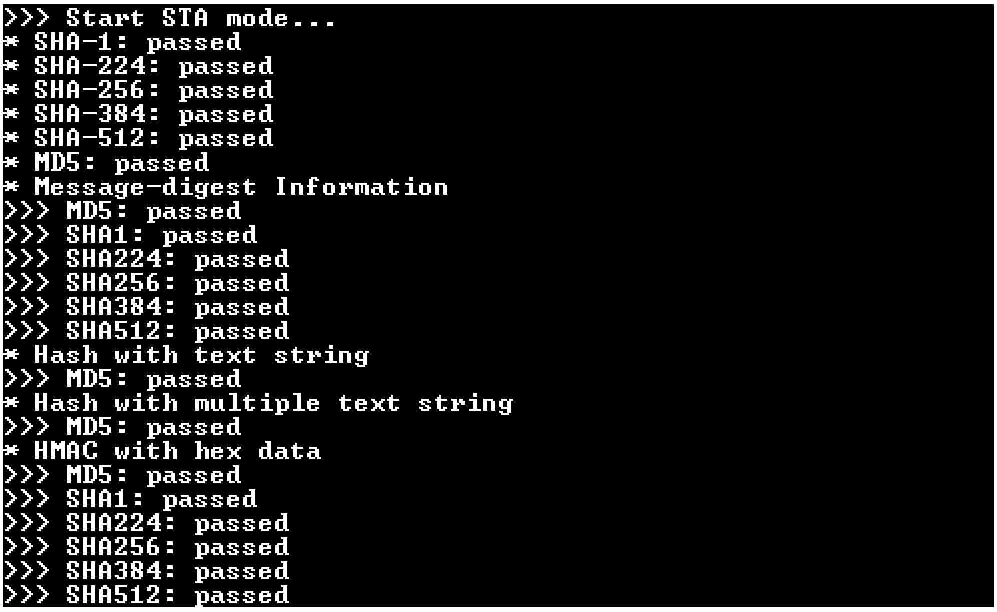


Figure 102. Result of crypto hash #1

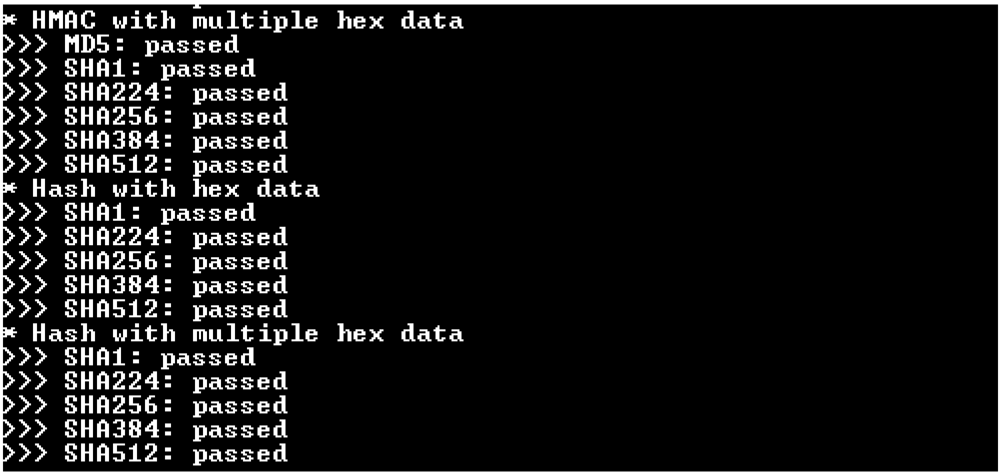


Figure 103. Result of crypto hash #2

16.1.5.1 Application Initialization

This example describes how the user can use hash and HMAC algorithms of the `mbedTLS` library.

```
void crypto_sample_hash(void *param)
{
      crypto_sample_hash_sha1();
```

```
      crypto_sample_hash_sha224();

      crypto_sample_hash_sha256();

      crypto_sample_hash_sha384();

      crypto_sample_hash_sha512();

#if defined(MBEDTLS_MD5_C)
      crypto_sample_hash_md5();
#endif // (MBEDTLS_MD5_C)

      crypto_sample_hash_md_wrapper();

      return ;
}
```

16.1.5.2 SHA-1 Hash

The DA16200 supports cryptographic algorithms for the SHA-1 hash. To explain how the SHA-1 hash works, see the test vector in FIPS-180-1.

```
int crypto_sample_hash_sha1()
{
    mbedtls_sha1_context *ctx = NULL;

    PRINTF("* SHA-1: ");

    // Initialize a SHA-1 context.
    mbedtls_sha1_init(ctx);

    // Start a SHA-1 checksum calculation.
    mbedtls_sha1_starts_ret(ctx);

    // Feed an input buffer into an ongoing SHA-1 checksum calculation.
    mbedtls_sha1_update_ret(ctx, crypto_sample_hash_sha1_buf,
                             crypto_sample_hash_sha1_buflen);

    // Finish the SHA-1 operation, and writes the result to the output buffer.
    mbedtls_sha1_finish(ctx, sha1sum);

    // Clear a SHA-1 context.
    mbedtls_sha1_free(ctx);
}
```

The mbedtls_sha1_context is the SHA-1 context structure. Function mbedtls_sha1_init is called to initialize the context. To calculate SHA-1 Hash, three functions should be called. The details can be found in Table 39.

Table 39. APIs for SHA-1 hash

Item	Description
int mbedtls_sha1_starts_ret(mbedtls_sha1_context *ctx)	
Prototype	int mbedtls_sha1_starts_ret(mbedtls_sha1_context *ctx)
Parameter	ctx: The SHA-1 context to initialize. This must be initialized.
Return	0 on success. A negative error code on failure.
Description	This function starts a SHA-1 checksum calculation.

Item	Description
int mbedtls_sha1_update_ret(mbedtls_sha1_context *ctx, const unsigned char *input, size_t ilen)	
Prototype	int mbedtls_sha1_update_ret(mbedtls_sha1_context *ctx, const unsigned char *input, size_t ilen)
Parameter	ctx: The SHA-1 context. This must be initialized and have a hash operation started. input: The buffer holding the input data. This must be a readable buffer of length ilen bytes. ilen: The length of the input data input in bytes.
Return	0 on success. A negative error code on failure.
Description	This function feeds an input buffer into an ongoing SHA-1 checksum calculation.
int mbedtls_sha1_finish_ret(mbedtls_sha1_context *ctx, unsigned char output[20])	
Prototype	int mbedtls_sha1_finish_ret(mbedtls_sha1_context *ctx, unsigned char output[20])
Parameter	ctx: The SHA-1 context to use. This must be initialized and have a hash operation started. output: The SHA-1 checksum result. This must be a writable buffer of length 20 bytes.
Return	0 on success. A negative error code on failure.
Description	This function finishes the SHA-1 operation and writes the result to the output buffer.

16.1.5.3 SHA-224 Hash

The DA16200 supports cryptographic algorithms for the SHA-224 hash. To explain how SHA-224 hash works, see the test vector in FIPS-180-2.

```
int crypto_sample_hash_sha224()
{
      mbedtls_sha256_context *ctx = NULL;

      PRINTF("* SHA-224: ");

      // Initialize the SHA-224 context.
      mbedtls_sha256_init(ctx);

      // Start a SHA-224 checksum calculation.
      mbedtls_sha256_starts_ret(ctx, 1);

      // Feeds an input buffer into an ongoing SHA-224 checksum calculation.
      mbedtls_sha256_update_ret(ctx, crypto_sample_hash_sha224_buf,
                                crypto_sample_hash_sha224_buflen);

      // Finishes the SHA-224 operation, and writes the result to the output buffer.
      mbedtls_sha256_finish_ret(ctx, sha224sum);

      //Clear s SHA-224 context.
      mbedtls_sha256_free(ctx);
}
```

The mbedtls_sha256_context is the SHA-256 context structure. The `mbedTLS` library supports SHA-224 and SHA-256 using the context. This sample describes SHA-224. Call function mbedtls_sha256_init to initialize the context. To calculate SHA-224 Hash, three functions should be called. The details can be found in Table 40.

Table 40. APIs for SHA-224 and SHA-256 hash

Item	Description
int mbedtls_sha256_starts_ret(mbedtls_sha256_context *ctx, int is224)	
Prototype	int mbedtls_sha256_starts_ret(mbedtls_sha256_context *ctx, int is224)

Item	Description
Parameter	ctx: The context to use. This must be initialized. is224: This determines which function to use. This must be either 0 for SHA-256, or 1 for SHA-224.
Return	0 on success. A negative error code on failure.
Description	This function starts a SHA-224 or SHA-256 checksum calculation.
int mbedtls_sha256_update_ret(mbedtls_sha256_context *ctx, const unsigned char *input, size_t ilen)	
Prototype	int mbedtls_sha256_update_ret(mbedtls_sha256_context *ctx, const unsigned char *input, size_t ilen)
Parameter	ctx: The SHA-256 context. This must be initialized and have a hash operation started. input: The buffer holding the input data. This must be a readable buffer of length ilen bytes. ilen: The length of the input data input in bytes.
Return	0 on success. A negative error code on failure.
Description	This function feeds an input buffer into an ongoing SHA-256 checksum calculation.
int mbedtls_sha256_finish_ret(mbedtls_sha256_context *ctx, unsigned char output[32])	
Prototype	int mbedtls_sha256_finish_ret(mbedtls_sha256_context *ctx, unsigned char output[32])
Parameter	ctx: The SHA-256 context to use. This must be initialized and have a hash operation started. output: The SHA-224 or SHA-256 checksum result. This must be a writable buffer of length 32 bytes.
Return	0 on success. A negative error code on failure.
Description	This function finishes the SHA-256 operation and writes the result to the output buffer.

16.1.5.4 SHA-256 Hash

The DA16200 supports cryptographic algorithms for the SHA-256 hash. To explain how the SHA-256 hash works, see the test vector in FIPS-180-2.

```
int crypto_sample_hash_sha256()
{
    mbedtls_sha256_context *ctx = NULL;

    PRINTF("* SHA-256: ");

    // Initialize the SHA-256 context.
    mbedtls_sha256_init(ctx);

    // Start a SHA-256 checksum calculation.
    mbedtls_sha256_starts_ret(ctx, 0);

    // Feeds an input buffer into an ongoing SHA-256 checksum calculation.
    mbedtls_sha256_update_ret(ctx, rypto_sample_hash_sha256_buf,
                              crypto_sample_hash_sha256_buflen);

    // Finishe the SHA-256 operation, and writes the result to the output buffer.
    mbedtls_sha256_finish_ret(ctx, sha256sum);

    //Clear s SHA-256 context.
    mbedtls_sha256_free(ctx);
}
```

This example is the same as the Cryptographic Algorithm for the SHA-224 code (see Section 16.1.5.3). When starting the SHA-256 checksum calculation, the second parameter should be set to 0 for SHA-256.

16.1.5.5 SHA-384 Hash

The DA16200 supports cryptographic algorithms for the SHA-384 hash. To explain how the SHA-384 hash works, see the test vector in FIPS-180-2.

```
int crypto_sample_hash_sha384()
{
    mbedtls_sha512_context *ctx = NULL;

    PRINTF("* SHA-384: ");

    // Initialize a SHA-384 context.
    mbedtls_sha512_init(ctx);

    // Start a SHA-384 checksum calculation.
    mbedtls_sha512_starts_ret(ctx, 1);

    // Feed an input buffer into an ongoing SHA-384 checksum calculation.
    mbedtls_sha512_update(ctx, crypto_sample_hash_sha384_buf,
                          crypto_sample_hash_sha384_buflen);

    // Finishe the SHA-384 operation, and writes the result to the output buffer.
    mbedtls_sha512_finish(ctx, sha384sum);

    // Clear a SHA-384 context.
    mbedtls_sha512_free(ctx);
}
```

The mbedtls_sha512_context is the SHA-512 context structure. `mbedTLS` library supports SHA-384 and SHA-512 using the context. This example describes SHA-384. Function mbedtls_sha512_init is called to initialize the context. To calculate SHA-384 Hash, three functions should be called. The details can be found in Table 41.

Table 41. APIs for SHA-384 and SHA-512 hash

Item	Description
int mbedtls_sha512_starts_ret(mbedtls_sha512_context *ctx, int is384)	
Prototype	int mbedtls_sha512_starts_ret(mbedtls_sha512_context *ctx, int is384)
Parameter	ctx: The context to use. This must be initialized. is384: This determines which function to use. This must be either 0 for SHA-512, or 1 for SHA-384.
Return	0 on success. A negative error code on failure.
Description	This function starts a SHA-384 or SHA-512 checksum calculation.
int mbedtls_sha512_update_ret(mbedtls_sha512_context *ctx, const unsigned char *input, size_t ilen)	
Prototype	int mbedtls_sha512_update_ret(mbedtls_sha512_context *ctx, const unsigned char *input, size_t ilen)
Parameter	ctx: The SHA-512 context. This must be initialized and have a hash operation started. input: The buffer holding the input data. This must be a readable buffer of length ilen bytes. ilen: The length of the input data input in bytes.
Return	0 on success. A negative error code on failure.
Description	This function feeds an input buffer into an ongoing SHA-512 checksum calculation.
int mbedtls_sha512_finish_ret(mbedtls_sha512_context *ctx, unsigned char output[64])	
Prototype	int mbedtls_sha512_finish_ret(mbedtls_sha512_context *ctx, unsigned char output[64])
Parameter	ctx: The SHA-512 context to use. This must be initialized and start a hash operation. output: The SHA-384 or SHA-512 checksum result. This must be a writable buffer of length 64 bytes.

Item	Description
Return	0 on success. A negative error code on failure.
Description	This function finishes the SHA-512 operation and writes the result to the output buffer.

16.1.5.6 SHA-512 Hash

The DA16200 supports cryptographic algorithms for the SHA-512 hash. To explain how the SHA-512 hash works, see the test vector in FIPS-180-2.

```
int crypto_sample_hash_sha512()
{
    mbedtls_sha512_context *ctx = NULL;

    PRINTF("* SHA-512: ");

    // Initialize a SHA-512 context.
    mbedtls_sha512_init(ctx);

    // Start a SHA-512 checksum calculation.
    mbedtls_sha512_starts_ret(ctx, 0);

    // Feed an input buffer into an ongoing SHA-512 checksum calculation.
    mbedtls_sha512_update_ret(ctx, crypto_sample_hash_sha512_buf,
                                                    crypto_sample_hash_sha512_buflen);

    // Finishe the SHA-512 operation, and writes the result to the output buffer.
    mbedtls_sha512_finish(ctx, sha512sum);

    // Clear a SHA-512 context.
    mbedtls_sha512_free(ctx);
}
```

This sample is the same as Cryptographic Algorithm for the SHA-384 code (see Section 16.1.5.5). When the SHA-512 checksum calculation is started, the second parameter should be set to 0 for SHA-512.

16.1.5.7 MD5 Hash

The DA16200 supports cryptographic algorithms for an MD5 hash. To explain how the MD5 hash works, see the test vector in RFC1321.

```
int crypto_sample_hash_md5()
{
    PRINTF("* MD5: ");

    // Output = MD5(input buffer)
    mbedtls_md5_ret(crypto_sample_hash_md5_buf,
                    crypto_sample_hash_md5_buflen, md5sum);
    return ret;
}
```

In this example, the MD5 hash function is calculated by function mbedtls_md5_ret. The details can be found in Table 42.

Table 42. APIs for MD5 hash

Item	Description
int mbedtls_md5_ret(const unsigned char *input, size_t ilen, unsigned char output[16])	
Prototype	int mbedtls_md5_ret(const unsigned char *input, size_t ilen, unsigned char output[16])
Parameter	Input: buffer holding the data Ilen: length of the input data Output: MD5 checksum result
Return	0 if successful.
Description	Output = MD5 (input buffer)

16.1.5.8 HASH and HMAC with Generic Message-Digest Wrapper

The `mbedTLS` library provides the generic message-digest wrapper to calculate HASH and HMAC. The APIs and sample codes show how HASH and HMAC are calculated with the generic message-digest wrapper functions. The API details are as follows:

Table 43. APIs for generic message digest wrapper

Item	Description
const mbedtls_md_info_t* mbedtls_md_info_from_type(mbedtls_md_type_t md_type)	
Prototype	const mbedtls_md_info_t* mbedtls_md_info_from_type(mbedtls_md_type_t md_type) c
Parameter	md_type: The type of digest to search for.
Return	The message-digest information associated with md_type. NULL if the associated message-digest information is not found.
Description	This function returns the message-digest information associated with the given digest type.
const mbedtls_md_info_t* mbedtls_md_info_from_string(const char* md_name)	
Prototype	const mbedtls_md_info_t* mbedtls_md_info_from_string(const char* md_name) (See Section 16.1.5.8.1)
Parameter	md_name: The name of the digest to search for.
Return	The message-digest information is associated with md_name. NULL if the associated message-digest information is not found.
Description	This function returns the message-digest information associated with the given digest name
mbedtls_md_type_t mbedtls_md_get_type(const mbedtls_md_info_t* md_info)	
Prototype	const mbedtls_md_info_t* mbedtls_md_info_from_string(const char* md_name) (See Section 16.1.5.8.1)
Parameter	md_info: The information structure of the message-digest algorithm to use.
Return	The type of the message digest.
Description	This function extracts the message-digest type from the message-digest information structure.
unsigned char mbedtls_md_get_size(const mbedtls_md_info_t* md_info)	
Prototype	unsigned char mbedtls_md_get_size(const mbedtls_md_info_t* md_info) (See Section 16.1.5.8.1)
Parameter	md_info: The information structure of the message-digest algorithm to use.
Return	The size of the message-digest output in bytes.
Description	This function extracts the message-digest size from the message-digest information structure.
const char* mbedtls_md_get_name(const mbedtls_md_info_t* md_info)	
Prototype	const char* mbedtls_md_get_name(const mbedtls_md_info_t* md_info) (See Section 16.1.5.8.1)

Item	Description
Parameter	md_info: The information structure of the message-digest algorithm to use.
Return	The name of the message digest.
Description	This function extracts the message-digest name from the message-digest information structure.
const int* mbedtls_md_list(void)	
Prototype	const int* mbedtls_md_list(void) (See Section 16.1.5.8.1)
Parameter	None
Return	A statically allocated array of digests. Each element in the returned list is an integer belonging to the message-digest enumeration mbedtls_md_type_t. The last entry is 0.
Description	This function returns the list of digests supported by the generic digest module.
int mbedtls_md(const mbedtls_md_info_t* md_info, const unsigned char* input, size_t ilen, unsigned char* output)	
Prototype	int mbedtls_md(const mbedtls_md_info_t* md_info, const unsigned char* input, size_t ilen, unsigned char* output) (See Section 16.1.5.8.2)
Parameter	md_info: The information structure of the message-digest algorithm to use. input: The buffer holding the data. ilen: The length of the input data. output: The generic message-digest checksum result.
Return	0 on success. MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.
Description	This function calculates the message-digest of a buffer, with respect to a configurable message-digest algorithm in a single call. The result is calculated as Output = message_digest (input buffer).
void mbedtls_md_init(mbedtls_md_context_t* ctx)	
Prototype	void mbedtls_md_init(mbedtls_md_context_t* ctx) (See Section 16.1.5.8.3)
Parameter	ctx: The context to initialize.
Return	None
Description	This function initializes a message-digest context without binding to a particular message-digest algorithm.
int mbedtls_md_setup(mbedtls_md_context_t* ctx, const mbedtls_md_info_t * md_info, int hmac)	
Prototype	int mbedtls_md_setup(mbedtls_md_context_t* ctx, const mbedtls_md_info_t * md_info, int hmac) (See Section 16.1.5.8.3)
Parameter	ctx: The context to set up. md_info: The information structure of the message-digest algorithm to use. hmac: Defines if HMAC is used. 0: HMAC is not used (saves some memory), or non-zero: HMAC is used with this context.
Return	0 on success. MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure. MBEDTLS_ERR_MD_ALLOC_FAILED on memory-allocation failure.
Description	This function selects the message digest algorithm to use and allocates internal structures.
int mbedtls_md_update(mbedtls_md_context_t* ctx, const unsigned char* input, size_t ilen)	
Prototype	int mbedtls_md_update(mbedtls_md_context_t* ctx, const unsigned char* input, size_t ilen) (See Section 16.1.5.8.3)
Parameter	ctx: The generic message-digest context. input: The buffer holding the input data. ilen: The length of the input data.

Item	Description
Return	0 on success. MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.
Description	This function feeds an input buffer into an ongoing message-digest computation.
int mbedtls_md_finish(mbedtls_md_context_t* ctx, unsigned char* output)	
Prototype	int mbedtls_md_finish(mbedtls_md_context_t* ctx, unsigned char* output) (See Section 16.1.5.8.3)
Parameter	ctx: The generic message-digest context. output: The buffer for the generic message-digest checksum result.
Return	0 on success. MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.
Description	This function finishes the digest operation and writes the result to the output buffer.
void mbedtls_md_free(mbedtls_md_context_t* ctx)	
Prototype	void mbedtls_md_free(mbedtls_md_context_t* ctx) (See Section 16.1.5.8.3)
Parameter	ctx: The generic message-digest context.
Return	None
Description	This function clears the internal structure of ctx and frees any embedded internal structure but does not free ctx itself.
int mbedtls_md_hmac(const mbedtls_md_info_t* md_info, const unsigned char* key, size_t keylen, const unsigned char* input, size_t ilen, unsigned char* output)	
Prototype	int mbedtls_md_hmac(const mbedtls_md_info_t* md_info, const unsigned char* key, size_t keylen, const unsigned char* input, size_t ilen, unsigned char* output) (See Section 16.1.5.8.4)
Parameter	md_info: The information structure of the message-digest algorithm to use. key: The HMAC secret key. keylen: The length of the HMAC secret key in bytes. input: The buffer holding the input data. ilen: The length of the input data. output: The generic HMAC result.
Return	0 on success. MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.
Description	This function calculates the full generic HMAC on the input buffer with the provided key. The function allocates the context, does the calculation, and frees the context. The HMAC result is calculated as output = generic HMAC (hmac key, input buffer).
int mbedtls_md_hmac_starts(mbedtls_md_context_t* ctx, const unsigned char* key, size_t keylen)	
Prototype	int mbedtls_md_hmac_starts(mbedtls_md_context_t* ctx, const unsigned char* key, size_t keylen) (See Section 16.1.5.8.4)
Parameter	ctx: The message digest context containing an embedded HMAC context. key: The HMAC secret key. keylen: The length of the HMAC key in bytes.
Return	0 on success. MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.
Description	This function sets the HMAC key and prepares to authenticate a new message
int mbedtls_md_hmac_update(mbedtls_md_context_t* ctx, const unsigned char* input, size_t ilen)	
Prototype	int mbedtls_md_hmac_update(mbedtls_md_context_t* ctx, const unsigned char* input, size_t ilen) (See Section 16.1.5.8.4)

Item	Description
Parameter	ctx: The message digest context containing an embedded HMAC context. input: The buffer holding the input data. ilen: The length of the input data.
Return	0 on success. MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.
Description	This function feeds an input buffer into an ongoing HMAC computation.
int mbedtls_md_hmac_finish(mbedtls_md_context_t* ctx, unsigned char* output)	
Prototype	int mbedtls_md_hmac_finish(mbedtls_md_context_t* ctx, unsigned char* output) (See Section 16.1.5.8.4)
Parameter	ctx: The message digest context containing an embedded HMAC context. output: The generic HMAC checksum result.
Return	0 on success. MBEDTLS_ERR_MD_BAD_INPUT_DATA on parameter-verification failure.
Description	This function finishes the HMAC operation and writes the result to the output buffer.

16.1.5.8.1 Supported Message-digest Functions

The user needs to check which message-digests are supported by the `mbedTLS` library. The sample code below shows how to get and check message-digest information.

```
int crypto_sample_hash_md_wrapper_info(char *md_name, mbedtls_md_type_t md_type, int md_size)
{
    const mbedtls_md_info_t *md_info = NULL;
    const int *md_type_ptr = NULL;

    // Get the message-digest information associated with the given digest type.
    md_info = mbedtls_md_info_from_type(md_type);
    if (!md_info) {
        PRINTF("[%s] Unknown Hash Type(%d)\r\n", __func__, md_type);
        goto cleanup;
    }

    // Get the message-digest information associated with the given digest name.
    if (md_info != mbedtls_md_info_from_string(md_name)) {
        PRINTF("[%s] Unknown Hash Name(%s)\r\n", md_name);
        goto cleanup;
    }

    // Extract the message-digest type from the message-digest information
    // structure.
    if (mbedtls_md_get_type(md_info) != (mbedtls_md_type_t)md_type) {
        PRINTF("[%s] Not matched Hash Type\r\n", __func__);
        goto cleanup;
    }

    // Extract the message-digest size from the message-digest information
    // structure.
    if (mbedtls_md_get_size(md_info) != (unsigned char)md_size) {
        PRINTF("[%s] Not matched Hash Size\r\n", __func__);
        goto cleanup;
    }

    // Extract the message-digest name from the message-digest information
    // structure.
    if (strcmp(mbedtls_md_get_name(md_info), md_name) != 0) {
```

```
        PRINTF("[%s] Not matched Hash Name\r\n", __func__);
        goto cleanup;
    }

    // Find the list of digests supported by the generic digest module.
    for (md_type_ptr = mbedtls_md_list() ; *md_type_ptr != 0 ; md_type_ptr++) {
        if (*md_type_ptr == md_type) {
            found = 1;
            break;
        }
    }

    return ret;
}
```

16.1.5.8.2 How to Calculate HASH Using Single Text String

The following sample code describes how a HASH function is calculated using the generic message-digest. In this sample, the text_src_string is used to calculate the message-digest algorithm, and the expected output is hex_hash_string.

```
int crypto_sample_hash_md_wrapper_text(char *md_name, char *text_src_string, char
*hex_hash_string)
{
    const mbedtls_md_info_t *md_info = NULL;

    // Get the message-digest information associated with the given digest name.
    md_info = mbedtls_md_info_from_string(md_name);

    /* Calculates the message-digest of a buffer,
     * with respect to a configurable message-digest algorithm in a single call.
     */
    ret = mbedtls_md(md_info,
                     (const unsigned char *)text_src_string,
                      strlen(text_src_string),
                      output);
}
```

16.1.5.8.3 How to Calculate HASH Using Multiple Text Strings

The sample code is similar to Section 16.1.5.8.2 and the only difference is that multiple text strings are used.

```
int crypto_sample_hash_md_wrapper_text_multi(char *md_name, char *text_src_string, char
*hex_hash_string)
{
    const mbedtls_md_info_t *md_info = NULL;
    mbedtls_md_context_t *ctx = NULL;       //The generic message-digest context.

    /* Initialize a message-digest context without binding it
     * to a particular message-digest algorithm.
     */
    mbedtls_md_init(ctx);

    // Get the message-digest information associated with the given digest name.
    md_info = mbedtls_md_info_from_string(md_name);

    // Select the message digest algorithm to use, and allocates internal
    // structures.
    ret = mbedtls_md_setup(ctx, md_info, 0);

    // Start a message-digest computation.
    ret = mbedtls_md_starts(ctx);
```

```
    // Feed an input buffer into an ongoing message-digest computation.
    ret = mbedtls_md_update(ctx, (const unsigned char *)text_src_string, halfway);

    // Feed an input buffer into an ongoing message-digest computation.
    ret = mbedtls_md_update(ctx,
                            (const unsigned char *)(text_src_string + halfway),
                            len - halfway);
    // Finish the digest operation, and writes the result to the output buffer.
    ret = mbedtls_md_finish(ctx, output);

    /* Clear the internal structure of ctx and free any embedded internal
       structure,
     * but does not free ctx itself.
     */
    mbedtls_md_free(ctx);
}
```

16.1.5.8.4 How to Calculate HMAC and HASH

- **How to calculate HMAC using single hex data**

The sample code below shows how the HMAC function is calculated using the generic message-digest wrapper. The hex_key_string is the HMAC secret key, the hex_src_string is input data, and the hex_hash_string is expected output. The mbedtls_md_hmac is for a single hex data.

```
int crypto_sample_hash_md_wrapper_hmac(char *md_name, int trunc_size, char *hex_key_string,
char *hex_src_string, char *hex_hash_string)
{
    const mbedtls_md_info_t *md_info = NULL;

    // Get the message-digest information associated with the given digest name.
    md_info = mbedtls_md_info_from_string(md_name);

    // Calculate the full generic HMAC on the input buffer with the provided key.
    ret = mbedtls_md_hmac(md_info, key_str, key_len, src_str, src_len, output);
}
```

- **How to calculate HMAC using multiple hex data**

The sample code is similar to **How to calculate HMAC using single hex data** and the only difference is that multiple hex data are used for input value.

```
int crypto_sample_hash_md_wrapper_hmac_multi(char *md_name, int trunc_size, char
*hex_key_string, char *hex_src_string, char *hex_hash_string)
{
    const mbedtls_md_info_t *md_info = NULL;
    mbedtls_md_context_t *ctx = NULL;

    /* Initialize a message-digest context without binding it
     * to a particular message-digest algorithm.
     */
    mbedtls_md_init(ctx);

    md_info = mbedtls_md_info_from_string(md_name);

    // Select the message digest algorithm to use, and allocate internal
    // structures.
    ret = mbedtls_md_setup(ctx, md_info, 1);

    // Start a message-digest computation.
    ret = mbedtls_md_hmac_starts(ctx, key_str, key_len);
```

```
    // Feed an input buffer into an ongoing message-digest computation.
    ret = mbedtls_md_hmac_update(ctx, src_str, halfway);

    // Feed an input buffer into an ongoing message-digest computation.
    ret = mbedtls_md_hmac_update(ctx, src_str + halfway, src_len - halfway);

    // Finish the digest operation, and writes the result to the output buffer.
    ret = mbedtls_md_hmac_finish(ctx, output);

    /* Clear the internal structure of ctx and free any embedded internal
       structure,
     * but does not free ctx itself.
     */
    mbedtls_md_free(ctx);
}
```

- **How to calculate HASH using a single hex data**

The sample code below describes how the HASH function is calculated with the generic message-digest function. The code is similar to Section 16.1.5.8.2 and the only difference is that a single hex data is used for input value.

```
int crypto_sample_hash_md_wrapper_hex(char *md_name, char *hex_src_string, char
*hex_hash_string)
{
    const mbedtls_md_info_t *md_info = NULL;

    // Get the message-digest information associated with the given digest name.
    md_info = mbedtls_md_info_from_string(md_name);

    /* Calculates the message-digest of a buffer,
     * with respect to a configurable message-digest algorithm in a single call.
     */
    ret = mbedtls_md(md_info, src_str, src_len, output);
}
```

- **How to calculate HASH using multiple hex data**

The sample code below describes how the HASH function is calculated with the generic message-digest function. The code is similar to Section 16.1.5.8.3 and the only difference is that multiple hex data are used for input value.

```
int crypto_sample_hash_md_wrapper_hex_multi(char *md_name, char *hex_src_string, char
*hex_hash_string)
{
    const mbedtls_md_info_t *md_info = NULL;
    mbedtls_md_context_t *ctx = NULL;

    /* Initialize a message-digest context without binding it
     * to a particular message-digest algorithm.
     */
    mbedtls_md_init(ctx);

    // Get the message-digest information associated with the given digest name.
    md_info = mbedtls_md_info_from_string(md_name);

    // Select the message digest algorithm to use, and allocate internal
    // structures.
    ret = mbedtls_md_setup(ctx, md_info, 0);

    // Start a message-digest computation.
    ret = mbedtls_md_starts(ctx);
```

```
    // Feed an input buffer into an ongoing message-digest computation.
    ret = mbedtls_md_update(ctx, src_str, halfway);

    // Feed an input buffer into an ongoing message-digest computation.
    ret = mbedtls_md_update(ctx, src_str + halfway, src_len - halfway);

    // Finish the digest operation, and writes the result to the output buffer.
    ret = mbedtls_md_finish(ctx, output);

    /* Clear the internal structure of ctx and free any embedded internal
       structure,
     * but does not free ctx itself.
     */

    mbedtls_md_free(ctx);
}
```

16.1.6 Cryptographic Algorithms – DRBG

The random generator sample application demonstrates common use cases of CTR-DRBG (Counter mode Deterministic Random Byte Generator) and HMAC-DRBG (HMAC Deterministic Random Byte Generator). The sample application explains how to use the DRBG function with CTR and HMAC.

- CTR_DRBG
- HMAC_DRBG.

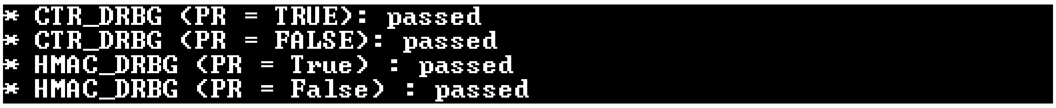


Figure 104. Result of crypto DRBG

16.1.6.1 Application Initialization

This example describes how the user uses CTR DRBG and HMAC DRBG of the `mbedTLS` library. CTR_DRBG is a standardized way of building a PRNG from a block-cipher in counter mode operation, as defined in NIST SP 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators. To implement `mbedTLS` of CTR_DRBG, use AES-256 (default) or AES-128 as the underlying block cipher. HMAC_DRBG is based on a Hash-based message authentication code.

```
void crypto_sample_drbg(void *param)
{
    crypto_sample_ctr_drbg_pr_on();

    crypto_sample_ctr_drbg_pr_off();

    crypto_sample_hmac_drbg_pr_on();

    crypto_sample_hmac_drbg_pr_off();

    return ;
}
```

16.1.6.2 CTR_DRBG with Prediction Resistance

This example describes how to use CTR_DRBG with prediction resistance.

```
int crypto_sample_ctr_drbg_pr_on()
{
    mbedtls_ctr_drbg_context *ctx = NULL;    //The CTR_DRBG context structure.

    // Based on a NIST CTR_DRBG test vector (PR = True)
```

```
    PRINTF("* CTR_DRBG (PR = TRUE): ");

    // Initialize the CTR_DRBG context.
    mbedtls_ctr_drbg_init( ctx );

    ret = mbedtls_ctr_drbg_seed_entropy_len(ctx, drbg_test_entropy,
                                      (void *)crypto_sample_ctr_drbg_entropy_src_pr,
                                       crypto_sample_ctr_brdg_nonce_pers_pr,
                                       16,
                                       32);

    // Turn prediction resistance on
    mbedtls_ctr_drbg_set_prediction_resistance(ctx, MBEDTLS_CTR_DRBG_PR_ON);

    // Generate random data using CTR_DRBG.
    ret = mbedtls_ctr_drbg_random(ctx, buf, MBEDTLS_CTR_DRBG_BLOCKSIZE);

    // Generate random data using CTR_DRBG.
    ret = mbedtls_ctr_drbg_random(ctx, buf, MBEDTLS_CTR_DRBG_BLOCKSIZE);

    // Clear CTR_CRBG context data.
    mbedtls_ctr_drbg_free(ctx);
}
```

The API details are as follows:

Table 44. APIs for CTR DRBG

Item	Description
void mbedtls_ctr_drbg_init(mbedtls_ctr_drbg_context* ctx)	
Prototype	int mbedtls_md_hmac_starts(mbedtls_md_context_t* ctx, const unsigned char* key, size_t keylen)
Parameter	ctx: The CTR_DRBG context to initialize.
Return	None
Description	This function initializes the CTR_DRBG context and prepares it for mbedtls_ctr_drbg_seed() or mbedtls_ctr_drbg_free().
void mbedtls_ctr_drbg_set_prediction_resistance(mbedtls_ctr_drbg_context* ctx, int resistance)	
Prototype	void mbedtls_ctr_drbg_set_prediction_resistance(mbedtls_ctr_drbg_ context* ctx, int resistance)
Parameter	resistance: MBEDTLS_CTR_DRBG_PR_ON or MBEDTLS_CTR_DRBG_PR_OFF
Return	None
Description	This function turns prediction resistance on or off. The default value is off.
int mbedtls_ctr_drbg_random(void* p_rng, unsigned char *output, size_t output_len)	
Prototype	int mbedtls_ctr_drbg_random(void* p_rng, unsigned char *output, size_t output_len)
Parameter	p_rng: The CTR_DRBG context. This must be a pointer to a mbedtls_ctr_drbg_context structure output: The buffer to fill. output_len: The length of the buffer.
Return	0 on success. MBEDTLS_ERR_CTR_DRBG_ENTROPY_SOURCE_FAILED or MBEDTLS_ERR_CTR_DRBG_REQUEST_TOO_BIG on failure.
Description	This function uses CTR_DRBG to generate random data.

Item	Description
void mbedtls_ctr_drbg_free(mbedtls_ctr_drbg_context* ctx)	
Prototype	void mbedtls_ctr_drbg_free(mbedtls_ctr_drbg_context* ctx)
Parameter	ctx: The CTR_DRBG context to clear.
Return	None
Description	This function clears CTR_CRBG context data.

16.1.6.3 CTR_DRBG Without Prediction Resistance

This example describes how to use CTR_DRBG without prediction resistance.

```
int crypto_sample_ctr_drbg_pr_off()
{
    mbedtls_ctr_drbg_context ctx;   //The CTR_DRBG context structure.

    // Based on a NIST CTR_DRBG test vector (PR = FALSE)
    PRINTF("* CTR_DRBG (PR = FALSE): ");

    // Initialize the CTR_DRBG context.
    mbedtls_ctr_drbg_init(&ctx);

    ret = mbedtls_ctr_drbg_seed_entropy_len(&ctx, drbg_test_entropy,
                                (void *) crypto_sample_ctr_drbg_entropy_src_nopr,
                                 crypto_sample_ctr_brdg_nonce_pers_nopr, 16, 32);

    // Generate random data using CTR_DRBG.
    ret = mbedtls_ctr_drbg_random(&ctx, buf, MBEDTLS_CTR_DRBG_BLOCKSIZE);

    // Reseed the CTR_DRBG context, that is extracts data from the entropy source.
    ret = mbedtls_ctr_drbg_reseed(&ctx, NULL, 0);

    // Generate random data using CTR_DRBG.
    ret = mbedtls_ctr_drbg_random(&ctx, buf, MBEDTLS_CTR_DRBG_BLOCKSIZE);

    // Clear CTR_CRBG context data.
    mbedtls_ctr_drbg_free(&ctx);
}
```

16.1.6.4 HMAC_DRBG with Prediction Resistance

This example describes how to use HMAC_DRBG with prediction resistance.

```
int crypto_sample_hmac_drbg_pr_on()
{
     mbedtls_hmac_drbg_context ctx;
     const mbedtls_md_info_t *md_info = mbedtls_md_info_from_type(MBEDTLS_MD_SHA1);

     PRINTF("* HMAC_DRBG (PR = True) : ");

     // Initialize HMAC DRBG context.
     mbedtls_hmac_drbg_init(&ctx);

     // HMAC_DRBG initial seeding Seed and setup entropy source for future reseeds.
     ret = mbedtls_hmac_drbg_seed(&ctx, md_info,
                               drbg_test_entropy,
                              (void *)crypto_sample_hmac_drbg_entropy_src_pr,
                               NULL, 0);

     // Enable prediction resistance.
```

```
    mbedtls_hmac_drbg_set_prediction_resistance(&ctx, MBEDTLS_HMAC_DRBG_PR_ON);

    // Generate random.
    ret = mbedtls_hmac_drbg_random(&ctx, buf, CRYPTO_SAMPLE_HMAC_DRBG_OUTPUT_LEN);

    // Generate random.
    ret = mbedtls_hmac_drbg_random(&ctx, buf, CRYPTO_SAMPLE_HMAC_DRBG_OUTPUT_LEN);

    // Free an HMAC_DRBG context.
    mbedtls_hmac_drbg_free(&ctx);
}
```

The API details are as follows:

Table 45. APIs for HMAC DRBG

Item	Description
void mbedtls_hmac_drbg_init (mbedtls_hmac_drbg_context* ctx)	
Prototype	void mbedtls_hmac_drbg_init(mbedtls_hmac_drbg_context *ctx)
Parameter	ctx: HMAC_DRBG context to be initialized.
Return	None
Description	HMAC_DRBG context initialization makes the context ready for mbedtls_hmac_drbg_seed(), mbedtls_hmac_drbg_seed_buf() or mbedtls_hmac_drbg_free().
int mbedtls_hmac_drbg_seed(mbedtls_hmac_drbg_context* ctx, const mbedtls_md_info_t * md_info, int (*f_entropy)(void*, unsigned char*, size_t), void* p_entropy, const unsigned char* custom, size_t len)	
Prototype	int mbedtls_hmac_drbg_seed(mbedtls_hmac_drbg_context* ctx, const mbedtls_md_info_t * md_info, int (*f_entropy)(void*, unsigned char*, size_t), void* p_entropy, const unsigned char* custom, size_t len)
Parameter	ctx: HMAC_DRBG context to be seeded. md_info: MD algorithm to use for HMAC_DRBG. f_entropy: Entropy callback (p_entropy, buffer to fill, buffer length). p_entropy: Entropy context. custom: Personalization data (Device specific identifiers) (Can be NULL). len: Length of personalization data.
Return	0 if successful, or MBEDTLS_ERR_MD_BAD_INPUT_DATA, or MBEDTLS_ERR_MD_ALLOC_FAILED, or MBEDTLS_ERR_HMAC_DRBG_ENTROPY_SOURCE_FAILED
Description	HMAC_DRBG initial seeding Seed and setup entropy source for future reseeds.
void mbedtls_hmac_drbg_set_prediction_resistance(mbedtls_hmac_drbg_context *ctx, int resistance)	
Prototype	void mbedtls_hmac_drbg_set_prediction_resistance(mbedtls_hmac_drbg_ context *ctx, int resistance)
Parameter	ctx: HMAC_DRBG context. resistance: MBEDTLS_HMAC_DRBG_PR_ON or MBEDTLS_HMAC_DRBG_PR_OFF.
Return	None
Description	Enable/disable prediction resistance (Default: Off).
int mbedtls_hmac_drbg_random(void *p_rng, unsigned char *output, size_t out_len)	
Prototype	int mbedtls_hmac_drbg_random(void *p_rng, unsigned char *output, size_t out_len)
Parameter	p_rng: HMAC_DRBG context. output: Buffer to fill. out_len: Length of the buffer.

Item	Description
Return	0 if successful, or MBEDTLS_ERR_HMAC_DRBG_ENTROPY_SOURCE_FAILED, or MBEDTLS_ERR_HMAC_DRBG_REQUEST_TOO_BIG.
Description	HMAC_DRBG generates random.
void mbedtls_hmac_drbg_free(mbedtls_hmac_drbg_context *ctx)	
Prototype	void mbedtls_hmac_drbg_free(mbedtls_hmac_drbg_context *ctx)
Parameter	ctx: HMAC_DRBG context to free.
Return	None
Description	Free an HMAC_DRBG context.
int mbedtls_hmac_drbg_reseed(mbedtls_hmac_drbg_context *ctx, const unsigned char *additional, size_t len)	
Prototype	int mbedtls_hmac_drbg_reseed(mbedtls_hmac_drbg_context *ctx, const unsigned char *additional, size_t len)
Parameter	ctx: HMAC_DRBG context. additional: Additional data to add to state (can be NULL). len: Length of additional data.
Return	0 if successful, or MBEDTLS_ERR_HMAC_DRBG_ENTROPY_SOURCE_FAILED.
Description	HMAC_DRBG reseeding (extracts data from entropy source).

16.1.6.5 HMAC_DRBG Without Prediction Resistance

This example describes how to use HMAC_DRBG without prediction resistance.

```
int crypto_sample_hmac_drbg_pr_off()
{
      mbedtls_hmac_drbg_context ctx;
      const mbedtls_md_info_t *md_info = mbedtls_md_info_from_type(MBEDTLS_MD_SHA1);

      PRINTF("* HMAC_DRBG (PR = False) : ");

      // Initialize HMAC DRBG context.
      mbedtls_hmac_drbg_init(&ctx);

      // HMAC_DRBG initial seeding Seed and setup entropy source for future reseeds.
      ret = mbedtls_hmac_drbg_seed(&ctx, md_info,
                                   drbg_test_entropy,
                                   (void *)crypto_sample_hmac_drbg_entropy_src_nopr,
                                   NULL, 0);

      // HMAC_DRBG reseeding (extracts data from entropy source)
      ret = mbedtls_hmac_drbg_reseed(&ctx, NULL, 0);

      // Generate random.
      ret = mbedtls_hmac_drbg_random(&ctx, buf, CRYPTO_SAMPLE_HMAC_DRBG_OUTPUT_LEN);

      // Generate random.
      ret = mbedtls_hmac_drbg_random(&ctx, buf, CRYPTO_SAMPLE_HMAC_DRBG_OUTPUT_LEN);

      // Free an HMAC_DRBG context.
      mbedtls_hmac_drbg_free(&ctx);
}
```

16.1.7 Cryptographic Algorithms – ECDSA

The Elliptic Curve Digital Signature Algorithm sample application demonstrates common use cases of the Elliptic Curve Digital Signature Algorithm. In cryptography, the Elliptic Curve Digital Signature Algorithm (ECDSA) offers a variant of the Digital Signature Algorithm (DSA) which uses elliptic curve cryptography.

```
* Seeding the random number generator: passed
* Generating key pair: passed - (key size: 192 bits)
* Computing message hash: passed
* Signing message hash: passed - (signature length = 56)
* Preparing verification context: passed
* Verifying signature: passed
```

Figure 105. Result of crypto ECDSA

16.1.7.1 Application Initialization

In cryptography, the ECDSA offers a variant of the DSA, which uses elliptic curve cryptography. This sample describes how the user uses the ECDSA of the `mbedTLS` library.

```
void crypto_sample_ecdsa(void *param)
{
    crypto_sample_ecdsa_test();

    return ;
}
```

16.1.7.2 Generate ECDSA Key Pair and Verifies ECDSA Signature

This example generates an ECDSA keypair and verifies the self-computed ECDSA signature.

```
int crypto_sample_ecdsa_test()
{
    int ret = -1;

    const char *pers = "crypto_sample_ecdsa";

    mbedtls_ecdsa_context ctx_sign;
    mbedtls_ecdsa_context ctx_verify;
    mbedtls_entropy_context entropy;
    mbedtls_ctr_drbg_context ctr_drbg;
    mbedtls_sha256_context sha256_ctx;

    // Initialize an ECDSA context.
    mbedtls_ecdsa_init(&ctx_sign);
    mbedtls_ecdsa_init(&ctx_verify);

    // Initialize the CTR_DRBG context.
    mbedtls_ctr_drbg_init(&ctr_drbg);

    // Initialize the SHA-256 context.
    mbedtls_sha256_init(&sha256_ctx);

    // Initialize the entropy context.
    mbedtls_entropy_init(&entropy);

    memset(sig, 0x00, MBEDTLS_ECDSA_MAX_LEN);
    memset(message, 0x25, 100);

    // Generate a key pair for signing
    PRINTF("* Seeding the random number generator: ");

    // Seed and sets up the CTR_DRBG entropy source for future reseeds.
    ret = mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func, &entropy,
```

```
                                (const unsigned char *)pers, strlen(pers));


    PRINTF("* Generating key pair: ");
    // Generate an ECDSA keypair on the given curve.
    ret = mbedtls_ecdsa_genkey(&ctx_sign, MBEDTLS_ECP_DP_SECP192R1,
                                mbedtls_ctr_drbg_random, &ctr_drbg);

    // Compute message hash
    PRINTF("* Computing message hash: ");

    // Start a SHA-256 checksum calculation.
    mbedtls_sha256_starts_ret(&sha256_ctx, 0);

    // Feeds an input buffer into an ongoing SHA-256 checksum calculation.
    mbedtls_sha256_update_ret(&sha256_ctx, message, 100);

    // Finishe the SHA-256 operation, and writes the result to the output buffer.
    mbedtls_sha256_finish(&sha256_ctx, hash);

    // Sign message hash
    PRINTF("* Signing message hash: ");

    // Compute the ECDSA signature and writes it to a buffer.
    ret = mbedtls_ecdsa_write_signature(&ctx_sign, MBEDTLS_MD_SHA256, hash, 32,
                           sig, &sig_len, mbedtls_ctr_drbg_random, &ctr_drbg);

    // Verify signature
    PRINTF("* Verifying signature: ");

    // Read and verify an ECDSA signature.
    ret = mbedtls_ecdsa_read_signature(&ctx_verify, hash, 32, sig, sig_len);

    // Free an ECDSA context.
    mbedtls_ecdsa_free(&ctx_verify);
    mbedtls_ecdsa_free(&ctx_sign);

    // Clear CTR_CRBG context data.
    mbedtls_ctr_drbg_free(&ctr_drbg);

    // Free the data in the context.
    mbedtls_entropy_free(&entropy);

    // Clear s SHA-256 context.
    mbedtls_sha256_free(&sha256_ctx);
}
```

The API details are as follows:

Table 46. APIs for ECDSA

Item	Description
void mbedtls_ecdsa_init(mbedtls_ecdsa_context *ctx)	
Prototype	void mbedtls_ecdsa_init(mbedtls_ecdsa_context *ctx)
Parameter	ctx: The ECDSA context to initialize. This must not be NULL.
Return	None
Description	This function initializes an ECDSA context.

Item	Description
int mbedtls_ecdsa_genkey(mbedtls_ecdsa_context *ctx, mbedtls_ecp_group_id gid, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)	
Prototype	int mbedtls_ecdsa_genkey(mbedtls_ecdsa_context *ctx, mbedtls_ecp_group_id gid, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)
Parameter	ctx: The ECDSA context to store the keypair in. This must be initialized. gid: The elliptic curve to use. One of the various MBEDTLS_ECP_DP_XXX macros depending on configuration. f_rng: The RNG function to use. This must not be NULL. p_rng: The RNG context to be passed to f_rng. This may be NULL if f_rng does not need a context argument.
Return	0 on success. An MBEDTLS_ERR_ECP_XXX code on failure.
Description	This function generates an ECDSA keypair on the given curve.
int mbedtls_ecdsa_write_signature(mbedtls_ecdsa_context *ctx, mbedtls_md_type_t md_alg, const unsigned char *hash, size_t hlen, unsigned char *sig, size_t *slen, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)	
Prototype	int mbedtls_ecdsa_write_signature(mbedtls_ecdsa_context *ctx, mbedtls_md_type_t md_alg, const unsigned char *hash, size_t hlen, unsigned char *sig, size_t *slen, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)
Parameter	ctx: The ECDSA context to use. This must be initialized and have a group and private key bound to it, for example via mbedtls_ecdsa_genkey() or mbedtls_ecdsa_from_keypair(). md_alg: The message digest that was used to hash the message. hash: The message hash to be signed. This must be a readable buffer of length hlen bytes. hlen: The length of the hash in bytes. sig: The buffer to which to write the signature. This must be a writable buffer of a length at least twice as large as the size of the curve used, plus 9. For example, 73 bytes if a 256-bit curve is used. The buffer length of MBEDTLS_ECDSA_MAX_LEN is always safe. slen: The address at which to store the actual length of the signature written. Must not be NULL. f_rng: The RNG function. This must not be NULL if MBEDTLS_ECDSA_DETERMINISTIC is unset. Otherwise, it is unused and may be set to NULL. p_rng: The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or does not use a context.
Return	0 on success. An MBEDTLS_ERR_ECP_XXX, MBEDTLS_ERR_MPI_XXX or MBEDTLS_ERR_ASN1_XXX error code on failure.
Description	This function computes the ECDSA signature and writes it to a buffer, serialized as defined in RFC-4492: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS).
int mbedtls_ecdsa_read_signature(mbedtls_ecdsa_context *ctx, const unsigned char *hash, size_t hlen, const unsigned char *sig, size_t slen)	
Prototype	int mbedtls_ecdsa_read_signature(mbedtls_ecdsa_context *ctx, const unsigned char *hash, size_t hlen, const unsigned char *sig, size_t slen)
Parameter	ctx: The ECDSA context to use. This must be initialized and have a group and public key bound to it. hash: The message hash that was signed. This must be a readable buffer of length size bytes hlen: The size of the hash. sig: The signature to read and verify. This must be a readable buffer of length slen bytes. slen: The size of sig in bytes.
Return	0 on success. MBEDTLS_ERR_ECP_BAD_INPUT_DATA if signature is invalid. MBEDTLS_ERR_ECP_SIG_LEN_MISMATCH if there is a valid signature in sig, but its length is less than siglen. An MBEDTLS_ERR_ECP_XXX or MBEDTLS_ERR_MPI_XXX error code on failure for any other reason.
Description	This function reads and verifies an ECDSA signature.

16.1.8 Cryptographic Algorithms – Diffie-Hellman Key Exchange

The Diffie-Hellman-Merkle (DHM) key exchange sample application demonstrates common use cases of DHM key exchange on the client and server sides.

```
* DHM parameter load: passed
* Diffie-Hellman full exchange: passed
```

Figure 106. Result of crypto Diffie Hellman

16.1.8.1 Application Initialization

This example includes two types. The first function crypto_sample_dhm_parse_dhm, shows how Diffie-Hellman parameters can be loaded. The second function crypto_sample_dhm_do_dhm, shows how DA16200 works for Diffie-Hellman key exchange.

```
void crypto_sample_dhm()
{
    ret = crypto_sample_dhm_parse_dhm();

    for (idx = 0 ; crypto_sample_dhm_do_dhm_list[idx].title != NULL ; idx++) {
        ret = crypto_sample_dhm_do_dhm(crypto_sample_dhm_do_dhm_list[idx].title,
                                       crypto_sample_dhm_do_dhm_list[idx].radix_P,
                                       crypto_sample_dhm_do_dhm_list[idx].input_P,
                                       crypto_sample_dhm_do_dhm_list[idx].radix_G,
                                       crypto_sample_dhm_do_dhm_list[idx].input_G);
    }
}
```

16.1.8.2 How Diffie-Hellman Works

Sample codes and APIs show how the Diffie-Hellman works and is loaded over the `mbedTLS` library's API. The API details are as follows.

Table 47. APIs for Diffie-Hellman-Merkle

Item	Description
void mbedtls_dhm_init(mbedtls_dhm_context *ctx)	
Prototype	void mbedtls_dhm_init(mbedtls_dhm_context *ctx) (See Section 16.1.8.2.1)
Parameter	ctx: The DHM context to initialize.
Return	None
Description	This function initializes the DHM context.
int mbedtls_dhm_parse_dhm(mbedtls_dhm_context *dhm, const unsigned char *dhmin, size_t dhminlen)	
Prototype	int mbedtls_dhm_parse_dhm(mbedtls_dhm_context *dhm, const unsigned char *dhmin, size_t dhminlen) (See Section 16.1.8.2.2)
Parameter	dhm: The DHM context to import the DHM parameters into. This must be initialized. dhmin: The input buffer. This must be a readable buffer of length dhminlen bytes. dhminlen: The size of the input buffer dhmin, including the terminating NULL byte for PEM data.
Return	0 on success. An MBEDTLS_ERR_DHM_XXX or MBEDTLS_ERR_PEM_XXX error code on failure.
Description	This function parses DHM parameters in PEM or DER format.
void mbedtls_dhm_free(mbedtls_dhm_context *ctx)	
Prototype	void mbedtls_dhm_free(mbedtls_dhm_context *ctx) See Section 16.1.8.2.1)
Parameter	ctx: The DHM context to free and clear. This may be NULL, in which case this function is a no-op. If it is not NULL, it must point to an initialized DHM context.
Return	None

Item	Description
Description	This function frees and clears the components of a DHM context.
int mbedtls_dhm_make_params(mbedtls_dhm_context *ctx, int x_size, char *output, size_t *olen, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)	
Prototype	int mbedtls_dhm_make_params(mbedtls_dhm_context *ctx, int x_size, char *output, size_t *olen, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng) (See Section 16.1.8.2.2)
Parameter	ctx: The DHM context to use. This must be initialized and have the DHM parameters set. It may or may not already have imported the peer's public key. x_size: The private key size in bytes. output: The destination buffer. This must be a writable buffer of sufficient size to hold the reduced binary presentation of the modulus, the generator and the public key, each wrapped with a 2-byte length field. It is the responsibility of the caller to ensure that enough space is available. See the mbedtls_mpi_size() to compute the byte-size of an MPI. olen: The address at which to store the number of bytes written on success. This must not be NULL. f_rng: The RNG function. Must not be NULL. p_rng: The RNG context to be passed to f_rng. This may be NULL if f_rng does not need a context parameter.
Return	0 on success. An MBEDTLS_ERR_DHM_XXX error code on failure
Description	This function generates a DHM key pair and exports its public part together with the DHM parameters in the format used in a TLS ServerKeyExchange handshake message.
int mbedtls_dhm_read_params(mbedtls_dhm_context *ctx, unsigned char **p, unsigned char *end)	
Prototype	int mbedtls_dhm_read_params(mbedtls_dhm_context *ctx, unsigned char **p, unsigned char *end) (See Section 16.1.8.2.2)
Parameter	ctx: The DHM context to use. This must be initialized. p: On input, *p must be the start of the input buffer. On output, *p is updated to point to the end of the data that has been read. On success, this is the first byte past the end of the ServerKeyExchange parameters. On error, this is the point at which an error has been detected, which is usually not useful except for debug failures. end: The end of the input buffer.
Return	0 on success. An MBEDTLS_ERR_DHM_XXX error code on failure.
Description	This function parses the DHM parameters in a TLS ServerKeyExchange handshake message (DHM modulus, generator, and public key).
int mbedtls_dhm_make_public(mbedtls_dhm_context *ctx, int x_size, unsigned char *output, size_t olen, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)	
Prototype	int mbedtls_dhm_read_params(mbedtls_dhm_context *ctx, unsigned char **p, unsigned char *end) (See Section 16.1.8.2.2)
Parameter	ctx: The DHM context to use. This must be initialized. p: On input, *p must be the start of the input buffer. On output, *p is updated to point to the end of the data that has been read. On success, this is the first byte past the end of the ServerKeyExchange parameters. On error, this is the point at which an error has been detected, which is usually not useful except to debug failures. end: The end of the input buffer.
Return	0 on success. An MBEDTLS_ERR_DHM_XXX error code on failure.
Description	This function parses the DHM parameters in a TLS ServerKeyExchange handshake message (DHM modulus, generator, and public key).
int mbedtls_dhm_make_public(mbedtls_dhm_context *ctx, int x_size, unsigned char *output, size_t olen, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)	
Prototype	int mbedtls_dhm_make_public(mbedtls_dhm_context *ctx, int x_size, unsigned char *output, size_t olen, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng) (See Section 16.1.8.2.2)

Item	Description
Parameter	ctx: The DHM context to use. This must be initialized and have the DHM parameters set. It may or may not already have imported the peer's public key. x_size: The private key size in bytes. output: The destination buffer. This must be a writable buffer of size olen bytes. olen: The length of the destination buffer. This must be at least equal to ctx->len (the size of P). f_rng: The RNG function. This must not be NULL. p_rng: The RNG context to be passed to f_rng. This may be NULL if f_rng does not need a context argument.
Return	0 on success. An MBEDTLS_ERR_DHM_XXX error code on failure.
Description	This function creates a DHM key pair and exports the raw public key in big-endian format.
int mbedtls_dhm_read_public(mbedtls_dhm_context *ctx, const unsigned char *input, size_t ilen)	
Prototype	int mbedtls_dhm_read_public(mbedtls_dhm_context *ctx, const unsigned char *input, size_t ilen) (See Section 16.1.8.2.2)
Parameter	ctx: The DHM context to use. This must be initialized and have its DHM parameters set, for instance via mbedtls_dhm_set_group(). It may or may not already have generated its own private key. input: The input buffer containing the G^Y value of the peer. This must be a readable buffer of size ilen bytes. ilen: The size of the input buffer input in bytes.
Return	0 on success. An MBEDTLS_ERR_DHM_XXX error code on failure.
Description	This function imports the raw public value of the peer.
int mbedtls_dhm_calc_secret(mbedtls_dhm_context *ctx, unsigned char *output, size_t output_size, size_t *olen, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)	
Prototype	int mbedtls_dhm_calc_secret(mbedtls_dhm_context *ctx, unsigned char *output, size_t output_size, size_t *olen, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng) (See Section 16.1.8.2.2)
Parameter	ctx: The DHM context to use. This must be initialized and have its own private key generated and the peer's public key imported. output: The buffer to write the generated shared key to. This must be a writable buffer of size output_size bytes. output_size: The size of the destination buffer. This must be at least the size of ctx->len (the size of P). olen: On exit, holds the actual number of bytes written. f_rng: The RNG function, for blinding purposes. This may be NULL if blinding is not needed. p_rng: The RNG context. This may be NULL if f_rng does not need a context argument.
Return	0 on success. An MBEDTLS_ERR_DHM_XXX error code on failure.
Description	This function derives and exports the shared secret (G^Y)^X mod P.

16.1.8.2.1 How to Load Diffie-Hellman Parameters

```
The mbedtls_dhm_parse_dhm parses DHM parameters in PEM or DER format. The
crypto_sample_dhm_params is already defined in this sample.

int crypto_sample_dhm_parse_dhm()
{
    mbedtls_dhm_context *dhm = NULL;     // The DHM context structure.

    // Initialize the DHM context.
    mbedtls_dhm_init(dhm);
```

```
    // Parse DHM parameters in PEM or DER format.
    ret = mbedtls_dhm_parse_dhm(dhm,
                                (const unsigned char *)crypto_sample_dhm_params,
                                crypto_sample_dhm_params_len);

    // Free and clear the components of a DHM context.
    mbedtls_dhm_free(dhm);
}
```

16.1.8.2.2 How Diffie-Hellman Works

The sample code shows how Diffie-Hellman works over the API of the `mbedTLS` library. Diffie-Hellman operation is normally used during TLS Handshake, ServerKeyExchange, and ClientKeyExchange messages. To verify it, the code exchanges ServerKeyExchange and ClientKeyExchange messages.

```
int crypto_sample_dhm_do_dhm(char *title, int radix_P, char *input_P, int radix_G, char
*input_G)
{
    mbedtls_dhm_context ctx_srv;
    mbedtls_dhm_context ctx_cli;
    rnd_pseudo_info rnd_info;

    // Initialize the DHM context.
    mbedtls_dhm_init(&ctx_srv);
    mbedtls_dhm_init(&ctx_cli);

    // Set parameters
    MBEDTLS_MPI_CHK(mbedtls_mpi_read_string(&ctx_srv.P, radix_P, input_P));
    MBEDTLS_MPI_CHK(mbedtls_mpi_read_string(&ctx_srv.G, radix_G, input_G));

    x_size = mbedtls_mpi_size(&ctx_srv.P);
    pub_cli_len = x_size;

    /* Generate a DHM key pair and export its public part together
     * with the DHM parameters in the format.
     */
    ret = mbedtls_dhm_make_params(&ctx_srv, x_size, ske, &ske_len,
                                  &rnd_pseudo_rand, &rnd_info);

    // Parse the DHM parameters (DHM modulus, generator, and public key)
    ret = mbedtls_dhm_read_params(&ctx_cli, &p, ske + ske_len);

    // Create a DHM key pair and export the raw public key in big-endian format.
    ret = mbedtls_dhm_make_public(&ctx_cli, x_size, pub_cli, pub_cli_len,
                                  &rnd_pseudo_rand, &rnd_info);

    // Import the raw public value of the peer.
    ret = mbedtls_dhm_read_public(&ctx_srv, pub_cli, pub_cli_len);

    // Derive and export the shared secret (G^Y)^X mod P.
    ret = mbedtls_dhm_calc_secret(&ctx_srv, sec_srv, DHM_BUF_SIZE,
                                  &sec_srv_len, &rnd_pseudo_rand, &rnd_info);

    // Derive and export the shared secret (G^Y)^X mod P.
    ret = mbedtls_dhm_calc_secret(&ctx_cli, sec_cli, DHM_BUF_SIZE, &sec_cli_len,
                                  NULL, NULL);

    // Free and clear the components of a DHM context.
    mbedtls_dhm_free(&ctx_srv);
    mbedtls_dhm_free(&ctx_cli);
}
```

16.1.9 Cryptographic Algorithms – RSA PKCS#1

The RSA PKCS#1 sample application demonstrates common use cases of RSA PKCS#1 functions.

```
* RSA key validation: passed
* PKCS#1 encryption : passed
* PKCS#1 decryption : passed
* PKCS#1 data sign  : passed
* PKCS#1 sig. verify: passed
```

Figure 107. Result of crypto RSA

16.1.9.1 Application Initialization

This example shows RSA key validation, encryption, decryption, and verification of the signature. To verify the signature, a SHA-1 Hash algorithm is used.

```
void crypto_sample_rsa(ULONG arg)
{
    crypto_sample_rsa_pkcs1();

    return ;
}
```

16.1.9.2 How RSA PKCS#1 Works

The example application below shows how RSA PKCS#1 works over the API of the `mbedTLS` library. To verify, an RSA-1024 keypair and a SHA-1 Hash algorithm are used on RSA PKCS-1 v1.5.

```
int crypto_sample_rsa_pkcs1()
{
    mbedtls_rsa_context *rsa = NULL;        // The RSA context structure.
    unsigned char *sha1sum = NULL;

    // Initializes an RSA context.
    mbedtls_rsa_init(rsa, MBEDTLS_RSA_PKCS_V15, MBEDTLS_MD_NONE);

    PRINTF("* RSA key validation: ");

    // Check if a context contains at least an RSA public key.
    ret = mbedtls_rsa_check_pubkey(rsa);

    ret = mbedtls_rsa_check_privkey(rsa);

    PRINTF("* PKCS#1 encryption : ");

    memcpy(rsa_plaintext, RSA_PT, PT_LEN);

    // Add the message padding, then performs an RSA operation.
    ret = mbedtls_rsa_pkcs1_encrypt(rsa, myrand,
                                    NULL, MBEDTLS_RSA_PUBLIC, PT_LEN,
                                    rsa_plaintext, rsa_ciphertext);

    PRINTF("* PKCS#1 decryption : ");

    // Perform an RSA operation, then removes the message padding.
    ret = mbedtls_rsa_pkcs1_decrypt(rsa, myrand,
                                    NULL, MBEDTLS_RSA_PRIVATE, &len,
                                    rsa_ciphertext, rsa_decrypted,
                                    (PT_LEN * sizeof(unsigned char)));

    PRINTF("* PKCS#1 data sign  : ");

    mbedtls_sha1_ret(rsa_plaintext, PT_LEN, sha1sum);
```

```
    // Perform a private RSA operation to sign a message digest using PKCS#1.
    ret = mbedtls_rsa_pkcs1_sign(rsa, myrand,
                                 NULL, MBEDTLS_RSA_PRIVATE, MBEDTLS_MD_SHA1,
                                 0, sha1sum, rsa_ciphertext);


    PRINTF("* PKCS#1 sig. verify: ");

    // Perform a public RSA operation and checks the message digest.
    ret = mbedtls_rsa_pkcs1_verify(rsa, NULL,
                                   NULL, MBEDTLS_RSA_PUBLIC, MBEDTLS_MD_SHA1,
                                   0, sha1sum, rsa_ciphertext);

    // Free the components of an RSA key.
    mbedtls_rsa_free(rsa);
}
```

The API details are as follows.

Table 48. APIs for PKCS#11 RSA

Item	Description
void mbedtls_rsa_init(mbedtls_rsa_context *ctx, int padding, int hash_id)	
Prototype	void mbedtls_rsa_init(mbedtls_rsa_context *ctx, int padding, int hash_id)
Parameter	ctx: The RSA context to initialize. This must not be NULL. padding: The padding mode to use. This must be either MBEDTLS_RSA_PKCS_V15 or MBEDTLS_RSA_PKCS_V21. hash_id: The hash identifier of mbedtls_md_type_t type, if padding is MBEDTLS_RSA_PKCS_V21. It is otherwise unused.
Return	None
Description	This function initializes an RSA context.
int mbedtls_rsa_check_pubkey(const mbedtls_rsa_context *ctx)	
Prototype	int mbedtls_rsa_check_pubkey(const mbedtls_rsa_context *ctx)
Parameter	ctx: The initialized RSA context to check.
Return	0 on success. An MBEDTLS_ERR_RSA_XXX error code on failure.
Description	This function checks if a context contains at least an RSA public key. If the function runs successfully, it is guaranteed that enough information is present to do an RSA public key operation with mbedtls_rsa_public().
int mbedtls_rsa_check_privkey(const mbedtls_rsa_context *ctx)	
Prototype	int mbedtls_rsa_check_pubkey(const mbedtls_rsa_context *ctx)
Parameter	ctx: The initialized RSA context to check.
Return	0 on success. An MBEDTLS_ERR_RSA_XXX error code on failure.
Description	This function checks if a context contains an RSA private key and does basic consistency checks.
int mbedtls_rsa_pkcs1_encrypt(mbedtls_rsa_context *ctx, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t ilen, const unsigned char *input, unsigned char *output)	
Prototype	int mbedtls_rsa_pkcs1_encrypt(mbedtls_rsa_context *ctx, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t ilen, const unsigned char *input, unsigned char *output)

Item	Description
Parameter	ctx: The initialized RSA context to use. f_rng: The RNG to use. It is mandatory for PKCS#1 v2.1 padding encoding, and for PKCS#1 v1.5 padding encoding when used with mode set to MBEDTLS_RSA_PUBLIC. For PKCS#1 v1.5 padding encoding and mode set to MBEDTLS_RSA_PRIVATE, it is used for blinding and should be provided in this case. See mbedtls_rsa_private() for more information. p_rng: The RNG context to be passed to f_rng. May be NULL if f_rng is NULL or if f_rng does not need a context argument. mode: The mode of operation. This must be either MBEDTLS_RSA_PUBLIC or MBEDTLS_RSA_PRIVATE (deprecated). ilen: The length of the plaintext in bytes. input: The input data to encrypt. This must be a readable buffer of size ilen bytes. This must not be NULL. output: The output buffer. This must be a writable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.
Return	0 on success. An MBEDTLS_ERR_RSA_XXX error code on failure.
Description	This function adds the message padding, then does an RSA operation. It is the generic wrapper to do a PKCS#1 encryption operation with the mode from the context.
int mbedtls_rsa_pkcs1_decrypt(mbedtls_rsa_context *ctx, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)	
Prototype	int mbedtls_rsa_pkcs1_decrypt(mbedtls_rsa_context *ctx, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)
Parameter	ctx: The initialized RSA context to use. f_rng: The RNG function. If mode is MBEDTLS_RSA_PRIVATE, this is used for blinding and should be provided; see mbedtls_rsa_private() for more. If mode is MBEDTLS_RSA_PUBLIC, it is ignored. p_rng: The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or does not need context. mode: The mode of operation. This must be either MBEDTLS_RSA_PRIVATE or MBEDTLS_RSA_PUBLIC (deprecated). olen: The address at which to store the length of the plaintext. This must not be NULL input: The ciphertext buffer. This must be a readable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus. output: The buffer used to hold the plaintext. This must be a writable buffer of length output_max_len bytes. output_max_len: The length in bytes of the output buffer output.
Return	0 on success. An MBEDTLS_ERR_RSA_XXX error code on failure.
Description	This function does an RSA operation, then removes the message padding. It is the generic wrapper to do a PKCS#1 decryption operation with the mode from the context.
int mbedtls_rsa_pkcs1_sign(mbedtls_rsa_context *ctx, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)	
Prototype	int mbedtls_rsa_pkcs1_sign(mbedtls_rsa_context *ctx, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)

Item	Description
Parameter	ctx: The initialized RSA context to use. f_rng: The RNG function to use. If the padding mode is PKCS#1 v2.1, this must be provided. If the padding mode is PKCS#1 v1.5 and the mode is MBEDTLS_RSA_PRIVATE, it is used for blinding and should be provided. See mbedtls_rsa_private() for more information. It is otherwise ignored. p_rng: The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or does not need a context argument. mode: The mode of operation. This must be either MBEDTLS_RSA_PRIVATE or MBEDTLS_RSA_PUBLIC (deprecated). md_alg: The message-digest algorithm used to hash the original data. Use MBEDTLS_MD_NONE for signing raw data. hashlen: The length of the message digest. This is only used if md_alg is MBEDTLS_MD_NONE. hash: The buffer holding the message digest or raw data. If md_alg is MBEDTLS_MD_NONE, this must be a readable buffer of length hashlen bytes. If md_alg is not MBEDTLS_MD_NONE, it must be a readable buffer of length the size of the hash corresponding to md_alg sig: The buffer to hold the signature. This must be a writable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.
Return	0 on success. An MBEDTLS_ERR_RSA_XXX error code on failure.
Description	This function does a private RSA operation to sign a message digest with PKCS#1. It is the generic wrapper to do a PKCS#1 signature with the mode from the context.
int mbedtls_rsa_pkcs1_verify(mbedtls_rsa_context *ctx, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)	
Prototype	int mbedtls_rsa_pkcs1_verify(mbedtls_rsa_context *ctx, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)
Parameter	ctx: The initialized RSA public key context to use. f_rng: The RNG function to use. If mode is MBEDTLS_RSA_PRIVATE, this is used for blinding and should be provided; see mbedtls_rsa_private() for more. Otherwise, it is ignored. p_rng: The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or does not need context. mode: The mode of operation. This must be either MBEDTLS_RSA_PUBLIC or MBEDTLS_RSA_PRIVATE (deprecated). md_alg: The message-digest algorithm used to hash the original data. Use MBEDTLS_MD_NONE for signing raw data. hashlen: The length of the message digest. This is only used if md_alg is MBEDTLS_MD_NONE. hash: The buffer holding the message digest or raw data. If md_alg is MBEDTLS_MD_NONE, this must be a readable buffer of length hashlen bytes. If md_alg is not MBEDTLS_MD_NONE, it must be a readable buffer of length the size of the hash that corresponds to md_alg. sig: The buffer holding the signature. This must be a readable buffer of length ctx->len bytes. For example, 256 bytes for a 2048-bit RSA modulus.
Return	0 on success. An MBEDTLS_ERR_RSA_XXX error code on failure.
Description	This function does a public RSA operation and checks the message digest. This is the generic wrapper to do PKCS#1 verification with the mode from the context.
void mbedtls_rsa_free(mbedtls_rsa_context *ctx)	
Prototype	void mbedtls_rsa_free(mbedtls_rsa_context *ctx)
Parameter	ctx: The RSA context to free. If it is NULL, in which case this function is a no-op. If it is not NULL, it must point to an initialized RSA context.
Return	None

Item	Description
Description	This function frees the components of an RSA key.

16.1.10 Cryptographic Algorithms – ECDH

The Elliptic-curve Diffie-Hellman (ECDH) sample application demonstrates common use cases of ECDH key exchange. It is a variant of the Diffie-Hellman protocol that uses elliptic-curve cryptography.

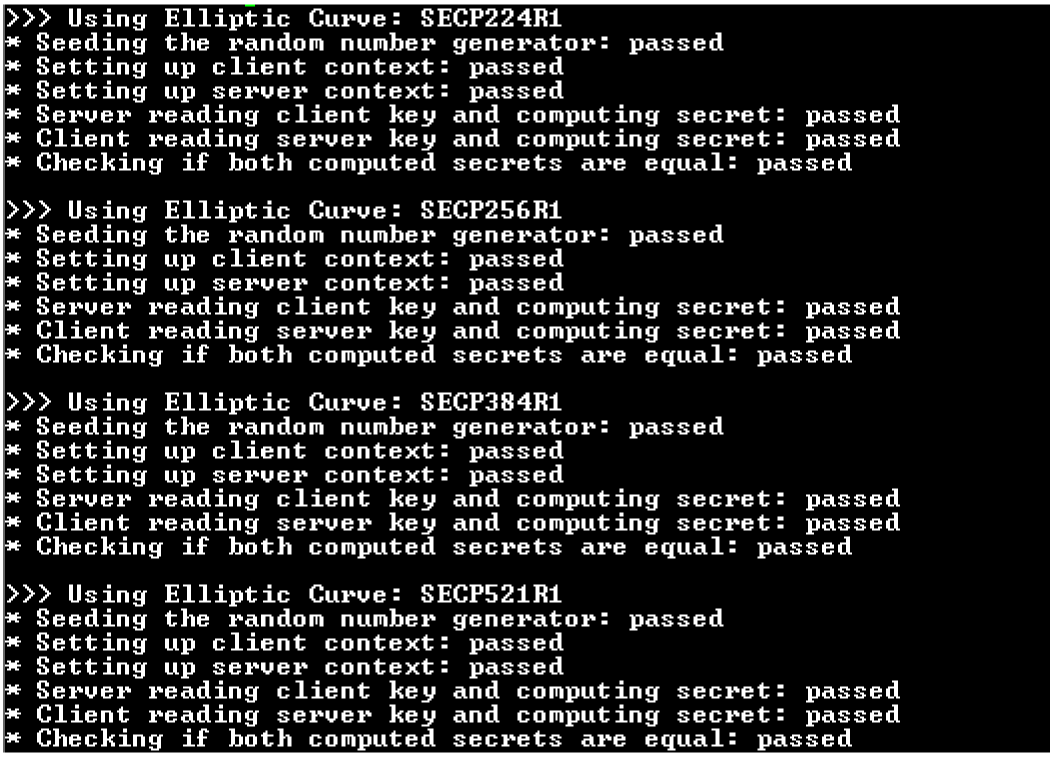


Figure 108. Result of crypto ECDH

16.1.10.1 Application Initialization

This example describes how the ECDH key exchange works with the use of Elliptic Curve SECP224R1, SECP256R1, SECP384R1, SECP521R1, and Curve25519.

```
void crypto_sample_ecdh(void *param)
{
    mbedtls_ecp_group_id ids[6] = {
        MBEDTLS_ECP_DP_SECP224R1, /*!< 224-bits NIST curve  */
        MBEDTLS_ECP_DP_SECP256R1, /*!< 256-bits NIST curve  */
        MBEDTLS_ECP_DP_SECP384R1, /*!< 384-bits NIST curve  */
        MBEDTLS_ECP_DP_SECP521R1, /*!< 521-bits NIST curve  */
        MBEDTLS_ECP_DP_CURVE25519,/*!< Curve25519           */
        MBEDTLS_ECP_DP_NONE
    };

    for (idx = 0, id = ids[idx] ; idx < 6 && id != MBEDTLS_ECP_DP_NONE ; idx++,
         id = ids[idx])
    {
        ret = crypto_sample_ecdh_key_exchange(id);
        if (ret) {
            break;
        }
    }
}
```

16.1.10.2 How ECDH Key Exchange Works

This sample application shows how ECDH works over the API of the "mbedTLS" library. In this example, the ECDH key exchange is verified on the server and client sides.

```
int crypto_sample_ecdh_key_exchange(mbedtls_ecp_group_id id)
{
    mbedtls_ecdh_context ctx_cli;
    mbedtls_ecdh_context ctx_srv;
    mbedtls_entropy_context entropy;
    mbedtls_ctr_drbg_context ctr_drbg;

    // Initialize an ECDH context.
    mbedtls_ecdh_init(&ctx_cli);
    mbedtls_ecdh_init(&ctx_srv);

    // Initialize the CTR_DRBG context.
    mbedtls_ctr_drbg_init(&ctr_drbg);

    // Initialize the entropy context.
    mbedtls_entropy_init(&entropy);

    PRINTF( ">>> Using Elliptic Curve: ");

    switch (id) {
        case MBEDTLS_ECP_DP_SECP224R1: {
            PRINTF("SECP224R1\r\n");
        }
        break;
        case MBEDTLS_ECP_DP_SECP256R1: {
            PRINTF("SECP256R1\r\n");
        }
        break;
        case MBEDTLS_ECP_DP_SECP384R1: {
            PRINTF("SECP384R1\r\n");
        }
        break;
        case MBEDTLS_ECP_DP_SECP521R1: {
            PRINTF("SECP521R1\r\n");
        }
        break;
        case MBEDTLS_ECP_DP_CURVE25519: {
            PRINTF("Curve25519\r\n");
        }
        break;
        default: {
            PRINTF("failed - [%s] Invalid Curve selected!\r\n");
        }
        goto cleanup;
    }
    // Initialize random number generation
    PRINTF("* Seeding the random number generator: ");

    ret = mbedtls_ctr_drbg_seed(&ctr_drbg, mbedtls_entropy_func, &entropy,
                                  (const unsigned char *)pers, sizeof(pers));

    // Client: inialize context and generate keypair
    PRINTF("* Setting up client context: ");

    // Sets up an ECP group context from a standardized set of domain parameters.
    ret = mbedtls_ecp_group_load(&(ctx_cli.grp), id);

    // Generate an ECDH keypair on an elliptic curve.
    ret = mbedtls_ecdh_gen_public(&(ctx_cli.grp), &(ctx_cli.d), &(ctx_cli.Q),
                                    mbedtls_ctr_drbg_random, &ctr_drbg);
    /* Export multi-precision integer (MPI) into unsigned binary data,
```

```
     * big endian (X coordinate of ECP point)
     */
    MBEDTLS_MPI_CHK(mbedtls_mpi_write_binary(&(ctx_cli.Q.X), cli_to_srv_x, buflen));

    /* Export multi-precision integer (MPI) into unsigned binary data,
     * big endian (Y coordinate of ECP point)
     */
    MBEDTLS_MPI_CHK(mbedtls_mpi_write_binary(&(ctx_cli.Q.Y), cli_to_srv_y, buflen));

    // Server: initialize context and generate keypair
    PRINTF("* Setting up server context: ");

    // Sets up an ECP group context from a standardized set of domain parameters.
    ret = mbedtls_ecp_group_load(&(ctx_srv.grp), id);

    // Generate a public key
    ret = mbedtls_ecdh_gen_public(&(ctx_srv.grp), &(ctx_srv.d), &(ctx_srv.Q),
                                  mbedtls_ctr_drbg_random, &ctr_drbg);

    /* Export multi-precision integer (MPI) into unsigned binary data,
     * big endian (X coordinate of ECP point).
     */
    MBEDTLS_MPI_CHK(mbedtls_mpi_write_binary(&(ctx_srv.Q.X), srv_to_cli_x, buflen));

    /* Export multi-precision integer (MPI) into unsigned binary data,
     * big endian (Y coordinate of ECP point).
     */

    MBEDTLS_MPI_CHK(mbedtls_mpi_write_binary(&(ctx_srv.Q.Y), srv_to_cli_y, buflen));
    /*
     * Server: read peer's key and generate shared secret
     */
    // Set the Z component of the peer's public value (public key) to 1
    MBEDTLS_MPI_CHK(mbedtls_mpi_lset(&(ctx_srv.Qp.Z), 1));

    /* Set the X component of the peer's public value based on
     * what was passed from client in the buffer.
     */
    MBEDTLS_MPI_CHK(mbedtls_mpi_read_binary(&(ctx_srv.Qp.X), cli_to_srv_x, buflen));

    /* Set the Y component of the peer's public value based on
     * what was passed from client in the buffer.
     */
    MBEDTLS_MPI_CHK(mbedtls_mpi_read_binary(&(ctx_srv.Qp.Y), cli_to_srv_y, buflen));

    // Compute the shared secret.
    ret = mbedtls_ecdh_compute_shared(&(ctx_srv.grp),
                                      &(ctx_srv.z), &(ctx_srv.Qp), &(ctx_srv.d),
                                      mbedtls_ctr_drbg_random, &ctr_drbg);

    // Client: read peer's key and generate shared secret
    PRINTF("* Client reading server key and computing secret: ");

    MBEDTLS_MPI_CHK(mbedtls_mpi_lset(&(ctx_cli.Qp.Z), 1));

    MBEDTLS_MPI_CHK(mbedtls_mpi_read_binary(&(ctx_cli.Qp.X), srv_to_cli_x, buflen));

    MBEDTLS_MPI_CHK(mbedtls_mpi_read_binary(&(ctx_cli.Qp.Y), srv_to_cli_y, buflen));

    // Compute the shared secret.
    ret = mbedtls_ecdh_compute_shared(&(ctx_cli.grp), &(ctx_cli.z),
                                      &(ctx_cli.Qp), &(ctx_cli.d),
                                      mbedtls_ctr_drbg_random, &ctr_drbg);

    // Verification: are the computed secrets equal?
    PRINTF("* Checking if both computed secrets are equal: ");
```

```
    MBEDTLS_MPI_CHK(mbedtls_mpi_cmp_mpi(&(ctx_cli.z), &(ctx_srv.z)));

    // Free ECDH context.
    mbedtls_ecdh_free(&ctx_cli);
    mbedtls_ecdh_free(&ctx_srv);

    // Free the data in the context.
    mbedtls_entropy_free(&entropy);

    // Clear CTR_CRBG context data.
    mbedtls_ctr_drbg_free(&ctr_drbg);
}
```

The API details are as follows.

Table 49. APIs for ECDH

Item	Description
void mbedtls_ecdh_init(mbedtls_ecdh_context *ctx)	
Prototype	void mbedtls_ecdh_init(mbedtls_ecdh_context *ctx)
Parameter	ctx: The ECDH context to initialize. This must not be NULL.
Return	None
Description	This function initializes an ECDH context.
int mbedtls_ecp_group_load(mbedtls_ecp_group *grp, mbedtls_ecp_group_id id)	
Prototype	int mbedtls_ecp_group_load(mbedtls_ecp_group *grp, mbedtls_ecp_group_id id)
Parameter	grp: The group context to set up. This must be initialized. id: The identifier of the domain parameter set to load.
Return	0 on success. MBEDTLS_ERR_ECP_FEATURE_UNAVAILABLE if the id does not correspond to a known group. Another negative error code on other kinds of failure.
Description	This function sets up an ECP group context from a standardized set of domain parameters.
int mbedtls_ecdh_gen_public(mbedtls_ecp_group *grp, mbedtls_mpi *d, mbedtls_ecp_point *Q, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)	
Prototype	int mbedtls_ecdh_gen_public(mbedtls_ecp_group *grp, mbedtls_mpi *d, mbedtls_ecp_point *Q, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)
Parameter	grp: The ECP group to use. This must be initialized and have domain parameters loaded, for example through mbedtls_ecp_load() or mbedtls_ecp_tls_read_group(). d: The destination MPI (private key). This must be initialized. Q: The destination point (public key). This must be initialized. f_rng: The RNG function to use. This must not be NULL. p_rng: The RNG context to be passed to f_rng. This may be NULL in case f_rng does not need a context argument.
Return	0 on success. Another MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.
Description	This function generates an ECDH keypair on an elliptic curve. This function is the first of two core computations implemented during the ECDH key exchange. The second core computation is done by mbedtls_ecdh_compute_shared().
int mbedtls_ecdh_compute_shared(mbedtls_ecp_group *grp, mbedtls_mpi *z, const mbedtls_ecp_point *Q, const mbedtls_mpi *d, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)	
Prototype	int mbedtls_ecdh_compute_shared(mbedtls_ecp_group *grp, mbedtls_mpi *z, const mbedtls_ecp_point *Q, const mbedtls_mpi *d, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)

Item	Description
Parameter	grp: The ECP group to use. This must be initialized and have domain parameters loaded, for example through mbedtls_ecp_load() or mbedtls_ecp_tls_read_group(). z: The destination MPI (shared secret). This must be initialized. Q: The public key from another party. This must be initialized. d: Our secret exponent (private key). This must be initialized. f_rng: The RNG function. This may be NULL if randomization of intermediate results during the ECP computations is not needed (discouraged). See the documentation of mbedtls_ecp_mul() for more information. p_rng: The RNG context to be passed to f_rng. This may be NULL if f_rng is NULL or does not need a context argument.
Return	0 on success. Another MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.
Description	This function computes the shared secret. This function is the second of two core computations implemented during the ECDH key exchange. The first core computation is done by mbedtls_ecdh_gen_public().
void mbedtls_ecdh_free(mbedtls_ecdh_context *ctx)	
Prototype	void mbedtls_ecdh_free(mbedtls_ecdh_context *ctx)
Parameter	ctx: The context to free. This may be NULL, in which case this function does nothing. If it is not NULL, it must point to an initialized ECDH context.
Return	None
Description	This function frees a context.

16.1.11 Cryptographic Algorithms – KDF

The Key Derivation Function (KDF) sample application demonstrates common use cases of PKCS#5 functions.

Figure 109. Result of crypto KDF

16.1.11.1 Application Initialization

This example uses a password-based Key Derivation Function specified in PKCS#5 PBKDF2 and implemented in mbedTLS in function mbedtls_pkcs5_pdkdf2_hmac.

```
void crypto_sample_kdf(void *param)
{
    crypto_sample_pkcs5();
}
```

16.1.11.2 How KDF Works

This example application shows how KDF works over the API of the mbedTLS library. In this example, PKCS#5 PBKDF2 is used. To verify, a SHA-1 Hash algorithm is used.

```
int crypto_sample_pkcs5()
{
    mbedtls_md_context_t sha1_ctx;
    const mbedtls_md_info_t *info_sha1;

    // Initialize a SHA-1 context.
    mbedtls_md_init(&sha1_ctx);

    // Get the message-digest information associated with the given digest type.
    info_sha1 = mbedtls_md_info_from_type(MBEDTLS_MD_SHA1);
```

```
    // Select the message digest algorithm to use, and allocate internal
    // structures.
    ret = mbedtls_md_setup(&sha1_ctx, info_sha1, 1);

    PRINTF("* PBKDF2 (SHA1): ");

    // Derive a key from a password using PBKDF2 function with HMAC
    ret = mbedtls_pkcs5_pbkdf2_hmac(&sha1_ctx,
                                    pkcs5_password, pkcs5_plen,
                                    pkcs5_salt, pkcs5_slen,
                                    pkcs5_it_cnt,
                                    pkcs5_key_len, key);

    /* Clear the internal structure of ctx and free any embedded internal
       structure,
     * but does not free ctx itself.
     */
    mbedtls_md_free(&sha1_ctx);
}
```

The API details are as follows.

Table 50. APIs for PKCS#5 PBKDF2

Item	Description
int mbedtls_pkcs5_pbkdf2_hmac(mbedtls_md_context_t *ctx, const unsigned char *password, size_t plen, const unsigned char *salt, size_t slen, unsigned int iteration_count, uint32_t key_length, unsigned char *output)	
Prototype	int mbedtls_pkcs5_pbkdf2_hmac(mbedtls_md_context_t *ctx, const unsigned char *password, size_t plen, const unsigned char *salt, size_t slen, unsigned int iteration_count, uint32_t key_length, unsigned char *output)
Parameter	ctx: Generic HMAC context. password: Password to use when generating a key. plen: Length of password. salt: Salt to use when generating a key. slen: Length of salt. iteration_count: Iteration count. key_length: Length of generated key in bytes. output: Generated key. Must be at least as big as key_length.
Return	0 on success, or a MBEDTLS_ERR_XXX code if verification fails.
Description	PKCS#5 PBKDF2 using HMAC.

16.1.12 Cryptographic Algorithms – Public Key Abstraction Layer

The `mbedTLS` library provides the Public Key abstraction layer for confidentiality, integrity, authentication, and non-repudiation based on asymmetric algorithms, using traditional RSA or Elliptic Curves. The Public Key abstraction layer sample application demonstrates common use cases of the APIs.

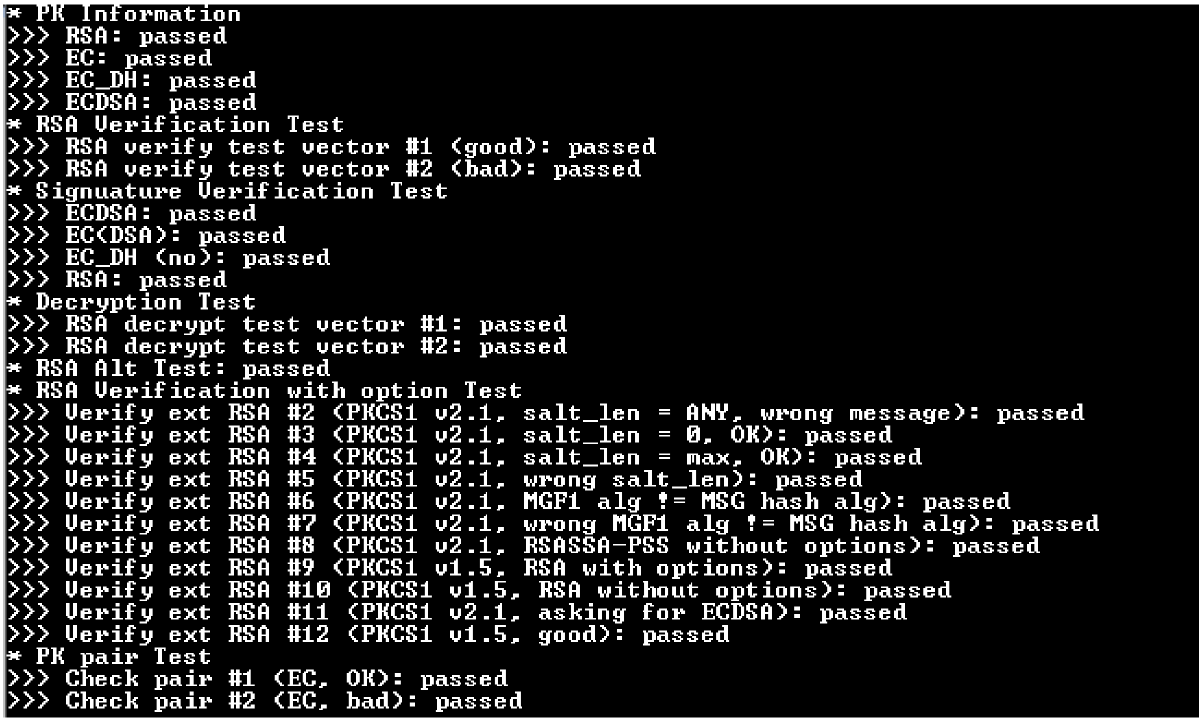


Figure 110. Result of crypto public key

16.1.12.1 Application Initialization

This example shows how to use the Public Key Abstraction Layer of the `mbedTLS` library.

```
void crypto_sample_pk(void *param)
{
    PRINTF("* PK Information\n");
    ret = crypto_sample_pk_utils(crypto_sample_pk_utils_list[i].type,
                            crypto_sample_pk_utils_list[i].size,
                    crypto_sample_pk_utils_list[i].len,
            crypto_sample_pk_utils_list[i].name);

    PRINTF("* RSA Verification Test\n");
    ret = crypto_sample_pk_rsa_verify_test_vec(
              crypto_sample_pk_rsa_verify_test_vec_list[i].title,
              crypto_sample_pk_rsa_verify_test_vec_list[i].message_hex_string,
              crypto_sample_pk_rsa_verify_test_vec_list[i].digest,
              crypto_sample_pk_rsa_verify_test_vec_list[i].mod,
              crypto_sample_pk_rsa_verify_test_vec_list[i].radix_N,
              crypto_sample_pk_rsa_verify_test_vec_list[i].input_N,
              crypto_sample_pk_rsa_verify_test_vec_list[i].radix_E,
              crypto_sample_pk_rsa_verify_test_vec_list[i].input_E,
              crypto_sample_pk_rsa_verify_test_vec_list[i].result_hex_str,
              crypto_sample_pk_rsa_verify_test_vec_list[i].result);

    PRINTF("* Signuature Verification Test\n");
    ret = crypto_sample_pk_sign_verify(
              crypto_sample_pk_sign_verify_list[i].title,
              crypto_sample_pk_sign_verify_list[i].type,
              crypto_sample_pk_sign_verify_list[i].sign_ret,
              crypto_sample_pk_sign_verify_list[i].verify_ret);

    PRINTF("* Decryption Test\n");
    ret = crypto_sample_pk_rsa_decrypt_test_vec(
              crypto_sample_pk_rsa_decrypt_list[i].title,
              crypto_sample_pk_rsa_decrypt_list[i].cipher_hex,
```

```
                crypto_sample_pk_rsa_decrypt_list[i].mod,
                crypto_sample_pk_rsa_decrypt_list[i].radix_P,
                crypto_sample_pk_rsa_decrypt_list[i].input_P,
                crypto_sample_pk_rsa_decrypt_list[i].radix_Q,
                crypto_sample_pk_rsa_decrypt_list[i].input_Q,
                crypto_sample_pk_rsa_decrypt_list[i].radix_N,
                crypto_sample_pk_rsa_decrypt_list[i].input_N,
                crypto_sample_pk_rsa_decrypt_list[i].radix_E,
                crypto_sample_pk_rsa_decrypt_list[i].input_E,
                crypto_sample_pk_rsa_decrypt_list[i].clear_hex,
                crypto_sample_pk_rsa_decrypt_list[i].result);

    ret = crypto_sample_pk_rsa_alt();

    PRINTF("* RSA Verification with option Test\n");
    ret = crypto_sample_pk_rsa_verify_ext_test_vec(
                crypto_sample_pk_rsa_verify_ext_list[i].title,
                crypto_sample_pk_rsa_verify_ext_list[i].message_hex_string,
                crypto_sample_pk_rsa_verify_ext_list[i].digest,
                crypto_sample_pk_rsa_verify_ext_list[i].mod,
                crypto_sample_pk_rsa_verify_ext_list[i].radix_N,
                crypto_sample_pk_rsa_verify_ext_list[i].input_N,
                crypto_sample_pk_rsa_verify_ext_list[i].radix_E,
                crypto_sample_pk_rsa_verify_ext_list[i].input_E,
                crypto_sample_pk_rsa_verify_ext_list[i].result_hex_str,
                crypto_sample_pk_rsa_verify_ext_list[i].pk_type,
                crypto_sample_pk_rsa_verify_ext_list[i].mgf1_hash_id,
                crypto_sample_pk_rsa_verify_ext_list[i].salt_len,
                crypto_sample_pk_rsa_verify_ext_list[i].result);

    PRINTF("* PK pair Test\n");
    ret = crypto_sample_pk_check_pair(
                crypto_sample_pk_check_pair_list[i].title,
                crypto_sample_pk_check_pair_list[i].pub_file,
                crypto_sample_pk_check_pair_list[i].prv_file,
                crypto_sample_pk_check_pair_list[i].result);
}
```

16.1.12.2 How to Use Public Key Abstraction Layer

The mbedTLS library provides the Public Key Abstraction Layer for confidentiality, integrity, authentication, and non-repudiation based on asymmetric algorithms, using traditional RSA or Elliptic Curves. The user needs to check which public key could be supported by the mbedTLS library. The example code below shows how to get and check public key information.

```
int crypto_sample_pk_utils(mbedtls_pk_type_t type, int size, int len, char *name)
{
    mbedtls_pk_context pk;

    // Initialize a mbedtls_pk_context.
    mbedtls_pk_init(&pk);

    /* Initialize a PK context with the information given
     * and allocates the type-specific PK subcontext.
     */
    ret = mbedtls_pk_setup(&pk, mbedtls_pk_info_from_type(type));

    // Get the key type.
    if (mbedtls_pk_get_type(&pk) != type) {
    }
```

```
    // Tell if a context can do the operation given by type.
    if (!mbedtls_pk_can_do(&pk, type)) {
    }

    // Get the size in bits of the underlying key.
    if (mbedtls_pk_get_bitlen(&pk) != (unsigned)size) {
    }

    // Get the length in bytes of the underlying key.
    if (mbedtls_pk_get_len(&pk) != (unsigned)len) {
    }

    // Access the type name.
    if ((ret = strcmp(mbedtls_pk_get_name(&pk), name)) != 0) {
    }

    // Free the components of a mbedtls_pk_context.
    mbedtls_pk_free(&pk);
}
```

The API details are as follows.

Table 51. APIs for public key abstraction layer

Item	Description
void mbedtls_pk_init(mbedtls_pk_context *ctx)	
Prototype	void mbedtls_pk_init(mbedtls_pk_context *ctx)
Parameter	ctx: The context to initialize. This must not be NULL.
Return	None
Description	Initialize an mbedtls_pk_context (as NONE).
int mbedtls_pk_setup(mbedtls_pk_context *ctx, const mbedtls_pk_info_t *info)	
Prototype	int mbedtls_pk_setup(mbedtls_pk_context *ctx, const mbedtls_pk_info_t *info)
Parameter	ctx: Context to initialize. It must not have been set up yet (type MBEDTLS_PK_NONE) info: Information to use.
Return	0 on success, MBEDTLS_ERR_PK_BAD_INPUT_DATA on invalid input, MBEDTLS_ERR_PK_ALLOC_FAILED on allocation failure.
Description	Initialize a PK context with the information given and allocate the type-specific PK sub context.
mbedtls_pk_type_t mbedtls_pk_get_type(const mbedtls_pk_context *ctx)	
Prototype	mbedtls_pk_type_t mbedtls_pk_get_type(const mbedtls_pk_context *ctx)
Parameter	ctx: The PK context to use. It must have been initialized.
Return	MBEDTLS_PK_NONE for a context that has not been set up.
Description	Get the key type.
int mbedtls_pk_can_do(const mbedtls_pk_context *ctx, mbedtls_pk_type_t type)	
Prototype	int mbedtls_pk_can_do(const mbedtls_pk_context *ctx, mbedtls_pk_type_t type)
Parameter	ctx: The context to query. It must have been initialized. type: The desired type.
Return	1 if the context can do operations on the given type. 0 if the context cannot do the operations on the given type. This is always the case for a context that has been initialized but not set up, or that has been cleared with mbedtls_pk_free().
Description	Tell if a context can do the operation given by the type.

Item	Description
size_t mbedtls_pk_get_bitlen(const mbedtls_pk_context *ctx)	
Prototype	size_t mbedtls_pk_get_bitlen(const mbedtls_pk_context *ctx)
Parameter	ctx: The context to query. It must have been initialized.
Return	Key size in bits, or 0 on error.
Description	Get the size in bits of the underlying key.
static inline size_t mbedtls_pk_get_len(const mbedtls_pk_context *ctx)	
Prototype	static inline size_t mbedtls_pk_get_len(const mbedtls_pk_context *ctx)
Parameter	ctx: The context to query. It must have been initialized.
Return	Key size in bits, or 0 on error.
Description	Get the length in bytes of the underlying key.
const char* mbedtls_pk_get_name(const mbedtls_pk_context *ctx)	
Prototype	const char* mbedtls_pk_get_name(const mbedtls_pk_context *ctx)
Parameter	ctx: The PK context to use. It must have been initialized.
Return	Type name on success, or "invalid PK."
Description	Access the type name
void mbedtls_pk_free(mbedtls_pk_context *ctx)	
Prototype	void mbedtls_pk_free(mbedtls_pk_context *ctx)
Parameter	ctx: The context to clear. It must have been initialized. If this is NULL, this function does nothing.
Return	None
Description	Free the components of a mbedtls_pk_context.

Function crypto_sample_pk_genkey describes how to generate a public key with the given algorithms (RSA or Elliptic curves).

```
int crypto_sample_pk_genkey(mbedtls_pk_context *pk)
{
    mbedtls_entropy_context *entropy = NULL;
    mbedtls_ctr_drbg_context *ctr_drbg = NULL;

    // Initialize the entropy context.
    mbedtls_entropy_init(entropy);

    // Initialize the CTR_DRBG context.
    mbedtls_ctr_drbg_init(ctr_drbg);

    // Seed and sets up the CTR_DRBG entropy source for future reseeds.
    mbedtls_ctr_drbg_seed(ctr_drbg, mbedtls_entropy_func, entropy, NULL, 0);

#if defined(MBEDTLS_RSA_C) && defined(MBEDTLS_GENPRIME)
    if (mbedtls_pk_get_type(pk) == MBEDTLS_PK_RSA) {
        // Generate the RSA key pair.
        ret = mbedtls_rsa_gen_key(mbedtls_pk_rsa(*pk),
                                  rnd_std_rand,
                                  ctr_drbg,
                                  RSA_KEY_SIZE, 3);
    }
#endif

#if defined(MBEDTLS_ECP_C)
    if ((mbedtls_pk_get_type(pk) == MBEDTLS_PK_ECKEY)
     || (mbedtls_pk_get_type(pk) == MBEDTLS_PK_ECKEY_DH)
```

```
      || (mbedtls_pk_get_type(pk) == MBEDTLS_PK_ECDSA)) {

         // Set a group using well-known domain parameters.
         ret = mbedtls_ecp_group_load(&mbedtls_pk_ec(*pk)->grp,
                                      MBEDTLS_ECP_DP_SECP192R1);

         // Generate key pair, wrapper for conventional base point
         ret = mbedtls_ecp_gen_keypair(&mbedtls_pk_ec(*pk)->grp,
                                       &mbedtls_pk_ec(*pk)->d,
                                       &mbedtls_pk_ec(*pk)->Q,
                                       rnd_std_rand, ctr_drbg);
     }
#endif
    mbedtls_ctr_drbg_free(ctr_drbg);
    mbedtls_entropy_free(entropy);
}
```

The API details are as follows.

Table 52. APIs for generating key pair

Item	Description
int mbedtls_rsa_gen_key(mbedtls_rsa_context *ctx, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, unsigned int nbits, int exponent)	
Prototype	int mbedtls_rsa_gen_key(mbedtls_rsa_context *ctx, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng, unsigned int nbits, int exponent)
Parameter	ctx: The initialized RSA context used to hold the key. f_rng: The RNG function to be used for key generation. This must not be NULL. p_rng: The RNG context to be passed to f_rng. This may be NULL if f_rng does not need a context. nbits: The size of the public key in bits. exponent: The public exponent to use. For example, 65537. This must be odd and greater than 1.
Return	0 on success. An MBEDTLS_ERR_RSA_XXX error code on failure.
Description	This function generates an RSA keypair.
int mbedtls_ecp_gen_keypair(mbedtls_ecp_group *grp, mbedtls_mpi *d, mbedtls_ecp_point *Q, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)	
Prototype	int mbedtls_ecp_gen_keypair(mbedtls_ecp_group *grp, mbedtls_mpi *d, mbedtls_ecp_point *Q, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)
Parameter	grp: The ECP group to generate a key pair for. This must be initialized and have group parameters set, for example through mbedtls_ecp_group_load(). d: The destination MPI (secret part). This must be initialized. Q: The destination point (public part). This must be initialized. f_rng: The RNG function. This must not be NULL. p_rng: The RNG context to be passed to f_rng. This may be NULL if f_rng does not need a context argument.
Return	0 on success. An MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.
Description	This function generates an ECP keypair.

Function crypto_sample_pk_rsa_verify_test_vec describes how to verify RSA signatures with Public Key abstraction Layer functions.

```
int crypto_sample_pk_rsa_verify_test_vec(char *title, char *message_hex_string,
mbedtls_md_type_t digest, int mod, int radix_N, char *input_N, int radix_E, char *input_E,
char *result_hex_str, int result)
```

```
{
    mbedtls_rsa_context *rsa = NULL;
    mbedtls_pk_context pk;

    // Initialize a mbedtls_pk_context.
    mbedtls_pk_init(&pk);

    /* Initialize a PK context with the information given
     * and allocates the type-specific PK subcontext.
     */
    ret = mbedtls_pk_setup(&pk, mbedtls_pk_info_from_type(MBEDTLS_PK_RSA));

    // Quick access to an RSA context inside a PK context.
    rsa = mbedtls_pk_rsa(pk);

    rsa->len = mod / 8;

    MBEDTLS_MPI_CHK(mbedtls_mpi_read_string(&rsa->N, radix_N, input_N));
    MBEDTLS_MPI_CHK(mbedtls_mpi_read_string(&rsa->E, radix_E, input_E));

    msg_len = unhexify(message_str, message_hex_string);

    unhexify(result_str, result_hex_str);

    // Get the message-digest information associated with the given digest type.
    if (mbedtls_md_info_from_type(digest) != NULL) {

        /* Calculates the message-digest of a buffer,
         * with respect to a configurable message-digest algorithm in a single call.
         */
        ret = mbedtls_md(mbedtls_md_info_from_type(digest),
                         message_str, msg_len,
                         hash_result);
    }

    // Verify signature (including padding if relevant) & Check result with
    // expected result.
    ret = mbedtls_pk_verify(&pk, digest, hash_result, 0, result_str,
                            mbedtls_pk_get_len(&pk));

    // Free the components of a mbedtls_pk_context.
    mbedtls_pk_free(&pk);
}
```

The API details are as follows.

Table 53. APIs for verifing signature

Item	Description
int mbedtls_pk_verify(mbedtls_pk_context *ctx, mbedtls_md_type_t md_alg, const unsigned char *hash, size_t hash_len, const unsigned char *sig, size_t sig_len)	
Prototype	int mbedtls_pk_verify(mbedtls_pk_context *ctx, mbedtls_md_type_t md_alg, const unsigned char *hash, size_t hash_len, const unsigned char *sig, size_t sig_len)
Parameter	ctx: The PK context to use. It must have been set up. md_alg: Hash algorithm used. hash: Hash of the message to sign. hash_len: Hash length or 0. sig: Signature to verify. sig_len: Signature length.

Item	Description
Return	0 on success (signature is valid), MBEDTLS_ERR_PK_SIG_LEN_MISMATCH if there is a valid signature in sig but its length is less than siglen, or a specific error code.
Description	Verify signature (including padding if relevant).

Function crypto_sample_pk_sign_verify describes how to generate a key, make a signature, and verify this with the given cryptographic algorithms.

```
int crypto_sample_pk_sign_verify(char *title, mbedtls_pk_type_t type, int sign_ret, int
verify_ret)
{
    mbedtls_pk_context pk;

    // Initialize a mbedtls_pk_context.
    mbedtls_pk_init(&pk);

    /* Initialize a PK context with the information given
     * and allocates the type-specific PK subcontext.
     */
    ret = mbedtls_pk_setup(&pk, mbedtls_pk_info_from_type(type));

    // Generate key pair by the type.
    ret = crypto_sample_pk_genkey(&pk);

    // Make signature, including padding if relevant and Check result with expected
    // result.
    ret = mbedtls_pk_sign(&pk, MBEDTLS_MD_SHA256,
                          hash, 64, sig, &sig_len,
                          rnd_std_rand, NULL);

    // Verify signature (including padding if relevant) and Check result with
    // expected result.
    ret = mbedtls_pk_verify(&pk, MBEDTLS_MD_SHA256, hash, 64, sig, sig_len);

    // Free the components of a mbedtls_pk_context.
    mbedtls_pk_free(&pk);
}
```

The API details are as follows.

Table 54. APIs for making signature

Item	Description
int mbedtls_pk_sign(mbedtls_pk_context *ctx, mbedtls_md_type_t md_alg, const unsigned char *hash, size_t hash_len, unsigned char *sig, size_t *sig_len, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)	
Prototype	int mbedtls_pk_sign(mbedtls_pk_context *ctx, mbedtls_md_type_t md_alg, const unsigned char *hash, size_t hash_len, unsigned char *sig, size_t *sig_len, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)
Parameter	ctx: The PK context to use. Must have been set up with a private key. md_alg: Hash algorithm used. hash: Hash of the message to sign. hash_len: Hash length or 0. sig: Place to write the signature. sig_len: Number of bytes written. f_rng: RNG function. p_rng: RNG parameter.
Return	0 on success, or a specific error code.

Item	Description
Description	Make a signature, including padding if relevant.

Function crypto_sample_pk_rsa_decrypt_test_vec describes how RSA is decrypted using Public Key Abstraction Layer's functions. Encryption could also be used. But this example only explains RSA decryption.

```
int crypto_sample_pk_rsa_decrypt_test_vec(char *title, char *cipher_hex, int mod, int
radix_P, char *input_P, int radix_Q, char *input_Q, int radix_N, char *input_N, int radix_E,
char *input_E, char *clear_hex, int result)
{
    rnd_pseudo_info *rnd_info = NULL;
    mbedtls_rsa_context *rsa = NULL;
    mbedtls_pk_context pk;

    // Initialize a mbedtls_pk_context.
    mbedtls_pk_init(&pk);

    /* Initialize a PK context with the information given
     * and allocates the type-specific PK subcontext.
     */
    ret = mbedtls_pk_setup(&pk, mbedtls_pk_info_from_type(MBEDTLS_PK_RSA));

    // Quick access to an RSA context inside a PK context.
    rsa = mbedtls_pk_rsa(pk);

    // Import a set of core parameters into an RSA context.
    ret = mbedtls_rsa_import(rsa, &N, &P, &Q, NULL, &E);

    // Retrieve the length of RSA modulus in bytes.
    if (mbedtls_rsa_get_len(rsa) != (size_t)(mod / 8)) {
    }

    // Complete an RSA context from a set of imported core parameters.
    ret = mbedtls_rsa_complete(rsa);

    // Decrypt message (including padding if relevant).
    ret = mbedtls_pk_decrypt(&pk, cipher, cipher_len,
                             output, &olen, (1000 * sizeof(unsigned char)),
                             rnd_pseudo_rand, rnd_info);

    // Free the components of a mbedtls_pk_context.
    mbedtls_pk_free(&pk);
}
```

The API details are as follows.

Table 55.APIs for PKCS#11 RSA

Item	Description
int mbedtls_rsa_import(mbedtls_rsa_context *ctx, const mbedtls_mpi *N, const mbedtls_mpi *P, const mbedtls_mpi *Q, const mbedtls_mpi *D, const mbedtls_mpi *E)	
Prototype	int mbedtls_rsa_import(mbedtls_rsa_context *ctx, const mbedtls_mpi *N, const mbedtls_mpi *P, const mbedtls_mpi *Q, const mbedtls_mpi *D, const mbedtls_mpi *E)
Parameter	ctx: The initialized RSA context to store the parameters in. N: The RSA modulus. This may be NULL. P: The first prime factor of N. This may be NULL. Q: The second prime factor of N. This may be NULL. D: The private exponent. This may be NULL. E: The public exponent. This may be NULL.

Item	Description
Return	0 on success. A non-zero error code on failure.
Description	This function imports a set of core parameters into an RSA context.
int mbedtls_rsa_complete(mbedtls_rsa_context *ctx)	
Prototype	int mbedtls_rsa_complete(mbedtls_rsa_context *ctx)
Parameter	ctx: The initialized RSA context holding imported parameters.
Return	0 on success. MBEDTLS_ERR_RSA_BAD_INPUT_DATA if the attempted derivations failed.
Description	This function completes an RSA context from a set of imported core parameters. To set up an RSA public key, precisely N and E must have been imported. To set up an RSA private key, sufficient information must be present for the other parameters to be derivable. The default implementation supports the following: > Derive P, Q from N, D, E > Derive N, D from P, Q, E Alternative implementations need not support these. If this function runs successfully, it guarantees that the RSA context can be used for RSA operations without the risk of failure or crash.
int mbedtls_pk_decrypt(mbedtls_pk_context *ctx, const unsigned char *input, size_t ilen, unsigned char *output, size_t *olen, size_t osize, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)	
Prototype	int mbedtls_pk_decrypt(mbedtls_pk_context *ctx, const unsigned char *input, size_t ilen, unsigned char *output, size_t *olen, size_t osize, int (*f_rng)(void *, unsigned char *, size_t), void *p_rng)
Parameter	ctx: The PK context to use. It must have been set up with a private key. input: Input to decrypt. ilen: Input size. output: Decrypted output. olen: Decrypted message length. osize: Size of the output buffer. f_rng: RNG function. p_rng: RNG parameter.
Return	0 on success, or a specific error code.
Description	Decrypt message (including padding if relevant).

Function crypto_sample_pk_rsa_alt describes how RSA ALT context creates and decrypts a signature.

```
int crypto_sample_pk_rsa_alt()
{
    /*
     * An rsa_alt context can only do private operations (decrypt, sign).
     * Test it against the public operations (encrypt, verify) of a
     * corresponding rsa context.
     */

    mbedtls_rsa_context *raw = NULL;
    mbedtls_pk_context rsa, alt;
    mbedtls_pk_debug_item *dbg_items = NULL;

    // Initialize an RSA context.
    mbedtls_rsa_init(raw, MBEDTLS_RSA_PKCS_V15, MBEDTLS_MD_NONE);

    // Initialize a mbedtls_pk_context.
    mbedtls_pk_init(&rsa);
    mbedtls_pk_init(&alt);
```

```
    /* Initialize a PK context with the information given
     * and allocates the type-specific PK subcontext.
     */
    ret = mbedtls_pk_setup(&rsa, mbedtls_pk_info_from_type(MBEDTLS_PK_RSA));

    // Generate key pair by the type.
    ret = crypto_sample_pk_genkey(&rsa);

    // Copy the components of an RSA context.
    ret = mbedtls_rsa_copy(raw, mbedtls_pk_rsa(rsa));

    // Initialize PK RSA_ALT context
    ret = mbedtls_pk_setup_rsa_alt(&alt, (void *)raw,
                                   crypto_sample_rsa_decrypt_func,
                                   crypto_sample_rsa_sign_func,
                                   crypto_sample_rsa_key_len_func);

    // Encrypt message (including padding if relevant).
    ret = mbedtls_pk_encrypt(&rsa, msg, 50, cipher,
                             &cipher_len, 1000, rnd_std_rand, NULL);

    // Decrypt message (including padding if relevant).
    ret = mbedtls_pk_decrypt(&alt, cipher, cipher_len,
                              test, &test_len, 1000, rnd_std_rand, NULL);

    // Free the components of an RSA key.
    mbedtls_rsa_free(raw);

    // Free the components of a mbedtls_pk_context.
    mbedtls_pk_free(&rsa);
    mbedtls_pk_free(&alt);
}
```

The API details are as follows.

Table 56. APIs for initializing RSA

Item	Description
int mbedtls_pk_setup_rsa_alt(mbedtls_pk_context *ctx, void * key, mbedtls_pk_rsa_alt_decrypt_func decrypt_func, mbedtls_pk_rsa_alt_sign_func sign_func, mbedtls_pk_rsa_alt_key_len_func key_len_func)	
Prototype	int mbedtls_pk_setup_rsa_alt(mbedtls_pk_context *ctx, void * key, mbedtls_pk_rsa_alt_decrypt_func decrypt_func, mbedtls_pk_rsa_alt_sign_func sign_func, mbedtls_pk_rsa_alt_key_len_func key_len_func)
Parameter	ctx: Context to initialize. It must not have been set up yet (type MBEDTLS_PK_NONE) key: RSA key pointer. decrypt_func: Decryption function. sign_func: Signing function. key_len_func: Function returning key length in bytes.
Return	0 on success, or MBEDTLS_ERR_PK_BAD_INPUT_DATA if the context was not already initialized as RSA_ALT.
Description	Initialize an RSA-alt context.

The code example shows how to check if a public and private pair of keys matches.

```
int crypto_sample_pk_check_pair(char *title, char *pub_file, char *prv_file, int result)
{
    mbedtls_pk_context pub, prv, alt;
```

```
    // Initialize a mbedtls_pk_context.
    mbedtls_pk_init(&pub);
    mbedtls_pk_init(&prv);

    // Parse a public key in PEM or DER format.
    ret = mbedtls_pk_parse_public_key(&pub,
                                      (const unsigned char *)pub_file,
                                      (strlen(pub_file) + 1));

    // Parse a private key in PEM or DER format.
    ret = mbedtls_pk_parse_key(&prv,
                               (const unsigned char *)prv_file,
                               (strlen(prv_file) + 1), NULL, 0);

    // Check if a public-private pair of keys matches.
    ret = mbedtls_pk_check_pair(&pub, &prv);

    mbedtls_pk_free(&pub);
    mbedtls_pk_free(&prv);
}
```

The API details are as follows.

Table 57. APIs for parsing private and public key

Item	Description
int mbedtls_pk_parse_public_key(mbedtls_pk_context *ctx, const unsigned char *key, size_t keylen)	
Prototype	int mbedtls_pk_parse_public_key(mbedtls_pk_context *ctx, const unsigned char *key, size_t keylen)
Parameter	ctx: The PK context to fill. It must have been initialized but not set up. key: Input buffer to parse. The buffer must contain the input exactly, with no extra trailing material. For PEM, the buffer must contain a null-terminated string. keylen: Size of key in bytes. For PEM data, this includes the terminating null byte, so keylen must be equal to strlen(key) + 1.
Return	0 if successful, or a specific PK or PEM error code.
Description	Parse a public key in PEM or DER format.
int mbedtls_pk_parse_key(mbedtls_pk_context *pk, const unsigned char *key, size_t keylen, const unsigned char *pwd, size_t pwdlen)	
Prototype	int mbedtls_pk_parse_key(mbedtls_pk_context *pk, const unsigned char *key, size_t keylen, const unsigned char *pwd, size_t pwdlen)
Parameter	pk: The PK context to fill. It must have been initialized but not set up. key: Input buffer to parse. The buffer must contain the input exactly, with no extra trailing material. For PEM, the buffer must contain a null-terminated string. keylen: Size of key in bytes. For PEM data, this includes the terminating null byte, so keylen must be equal to strlen(key) + 1. pwd: Optional password for decryption. Pass NULL if expecting a non-encrypted key. Pass a string of pwdlen bytes if expecting an encrypted key; a non-encrypted key is also accepted. The empty password is not supported. pwdlen: Size of the password in bytes. Ignored if pwd is NULL.
Return	0 if successful, or a specific PK or PEM error code.
Description	Parse a private key in PEM or DER format.
int mbedtls_pk_check_pair(const mbedtls_pk_context *pub, const mbedtls_pk_context *prv)	
Prototype	int mbedtls_pk_check_pair(const mbedtls_pk_context *pub, const mbedtls_pk_context *prv)

Item	Description
Parameter	pub: Context holding a public key. prv: Context holding a private (and public) key.
Return	0 on success or MBEDTLS_ERR_PK_BAD_INPUT_DATA.
Description	Check if a public-private pair of keys matches.

16.1.13 Cryptographic Algorithms – Generic Cipher Wrapper

The Generic cipher wrapper sample application demonstrates common use cases of a generic cipher wrapper API of the `mbedTLS` library that is included in the DA16200 SDK.

Figure 111. Result of generic cipher

16.1.13.1 Application Initialization

The generic cipher wrapper contains an abstraction interface for use with the cipher primitives that the library provides. It provides a common interface to all the available cipher operations.

```
void crypto_sample_cipher(void *param)
{
      crypto_sample_cipher_wrapper();

      vTaskDelete(NULL);

      return ;
}
```

16.1.13.2 How Generic Cipher Wrapper is Used

This example describes how to encrypt and decrypt with generic cipher wrapper functions.

```
int crypto_sample_cipher_wrapper()
{
    mbedtls_cipher_type_t cipher_type = MBEDTLS_CIPHER_NONE;
    mbedtls_cipher_context_t cipher_ctx;
    mbedtls_cipher_info_t *cipherinfo = NULL;
    mbedtls_cipher_mode_t cipher_mode = MBEDTLS_MODE_NONE;

    for (cipher_type = MBEDTLS_CIPHER_AES_128_ECB ;
         cipher_type <= MBEDTLS_CIPHER_CAMELLIA_256_CCM ;
         cipher_type++) {

        flag_pass = FALSE;
```

```
        // Initialize a cipher_context as NONE.
        mbedtls_cipher_init(cipher_ctx);

        // Retrieve the cipher-information structure associated with the given
        // cipher type.
        cipherinfo = (mbedtls_cipher_info_t *)mbedtls_cipher_info_from_type
                     (cipher_type);

        // Initialize and fill the cipher-context structure with the appropriate
        // values.
        mbedtls_cipher_setup(&cipher_ctx, cipherinfo);

        // Return the key length of the cipher.
        cipher_keylen = mbedtls_cipher_get_key_bitlen(&cipher_ctx);

        // Return the mode of operation for the cipher.
        cipher_mode = mbedtls_cipher_get_cipher_mode(&cipher_ctx);

        // Return the size of the IV or nonce of the cipher, in bytes.
        cipher_ivlen = mbedtls_cipher_get_iv_size(&cipher_ctx);

        // Return the block size of the given cipher.
        cipher_blksiz = mbedtls_cipher_get_block_size(&cipher_ctx);

        // Return the name of the given cipher as a string.
        cipher_name = (char *)mbedtls_cipher_get_name(&cipher_ctx);

        PRINTF("* %s", cipher_name);

        PRINTF("(enc, ");

        if (cipher_adlen == 0) { // No CCM or GCM
            // Set the key to use with the given context.
            cipher_status = mbedtls_cipher_setkey(&cipher_ctx,
                                                   cipher_key, cipher_keylen,
                                                   MBEDTLS_ENCRYPT);

            // Set the initialization vector (IV) or nonce.
            cipher_status = mbedtls_cipher_set_iv(&cipher_ctx,
                                                   cipher_iv, cipher_ivlen);

            // Reset the cipher state.
            cipher_status = mbedtls_cipher_reset(&cipher_ctx);

            // Encrypt or decrypt using the given cipher context.
            cipher_status = mbedtls_cipher_update(&cipher_ctx,
                                                   plain_in, plain_inlen,
                                                   ciphertext, &ciphertext_len);

            // Finish the operation.
            cipher_status = mbedtls_cipher_finish(&cipher_ctx,
                                                  &(ciphertext[ciphertext_len]),
                                                  &ciphertext_finlen);

        } else {
            // Set the key to use with the given context.
            cipher_status = mbedtls_cipher_setkey(&cipher_ctx,
                                                   cipher_key, cipher_keylen,
                                                   MBEDTLS_ENCRYPT);
```

```
            // Perform autenticated encryption (AEAD).
            cipher_status = mbedtls_cipher_auth_encrypt(&cipher_ctx,
                                                        cipher_iv, cipher_ivlen,
                                                        cipher_ad, cipher_adlen,
                                                        plain_in, plain_inlen,
                                                        ciphertext, &ciphertext_len,
                                                        cipher_tag, cipher_taglen);
        }

        PRINTF("dec): ");

        if (cipher_adlen == 0) { // No CCM or GCM
            // Set the key to use with the given context.
            cipher_status = mbedtls_cipher_setkey(&cipher_ctx,
                                                  cipher_key, cipher_keylen,
                                                  MBEDTLS_DECRYPT);

            // Set the initialization vector (IV) or nonce.
            cipher_status = mbedtls_cipher_set_iv(&cipher_ctx,
                                                  cipher_iv, cipher_ivlen);
            // Reset the cipher state.
            cipher_status = mbedtls_cipher_reset(&cipher_ctx);

            // Encrypt or decrypt using the given cipher context.
            cipher_status = mbedtls_cipher_update(&cipher_ctx,
                                     ciphertext, (ciphertext_len + ciphertext_finlen),
                                     plain_out, &plain_outlen);

            // Finish the operation.
            cipher_status = mbedtls_cipher_finish(&cipher_ctx,
                                                   &(plain_out[plain_outlen]),
                                                   &plain_finlen);
        } else {
            // Set the key to use with the given context.
            cipher_status = mbedtls_cipher_setkey(&cipher_ctx,
                                                  cipher_key, cipher_keylen,
                                                  MBEDTLS_DECRYPT);

            // Perform autenticated decryption (AEAD).
            cipher_status = mbedtls_cipher_auth_decrypt(&cipher_ctx,
                                                        cipher_iv, cipher_ivlen,
                                                        cipher_ad, cipher_adlen,
                                                        ciphertext, ciphertext_len,
                                                        plain_out, &plain_outlen,
                                                        cipher_tag, cipher_taglen);
        }

        // Free and clear the cipher-specific context of ctx.
        mbedtls_cipher_free(&cipher_ctx);
    }
}
```

The API details are as follows.

Table 58. APIs for generic cipher wrapper

Item	Description
void mbedtls_cipher_init(mbedtls_cipher_context_t *ctx)	
Prototype	void mbedtls_cipher_init(mbedtls_cipher_context_t *ctx)
Parameter	ctx: The context to be initialized. This must not be NULL.

Item	Description
Return	None
Description	This function initializes a cipher_context as NONE.
void mbedtls_cipher_free(mbedtls_cipher_context_t *ctx)	
Prototype	void mbedtls_cipher_free(mbedtls_cipher_context_t *ctx)
Parameter	ctx: The context to be freed. If this is NULL, the function has no effect, otherwise this must point to an initialized context.
Return	None
Description	This function frees and clears the cipher-specific context of ctx. Freeing ctx itself remains the responsibility of the caller.
const mbedtls_cipher_info_t* mbedtls_cipher_info_from_type(const mbedtls_cipher_type_t cipher_type)	
Prototype	Const mbedtls_cipher_info_t* mbedtls_cipher_info_from_type(const mbedtls_cipher_type_t cipher_type)
Parameter	cipher_type: Type of the cipher to search for.
Return	The cipher information structure is associated with the given cipher_type. NULL if the associated cipher information is not found.
Description	This function retrieves the cipher-information structure associated with the given cipher type.
int mbedtls_cipher_setup(mbedtls_cipher_context_t *ctx, const mbedtls_cipher_info_t *cipher_info)	
Prototype	int mbedtls_cipher_setup(mbedtls_cipher_context_t *ctx, const mbedtls_cipher_info_t *cipher_info)
Parameter	ctx: The context to initialize. This must be initialized. cipher_info: The cipher to use.
Return	0 on success. MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure. MBEDTLS_ERR_CIPHER_ALLOC_FAILED if allocation of the cipher-specific context fails.
Description	This function initializes and fills the cipher-context structure with the appropriate values. It also clears the structure.
static inline int mbedtls_cipher_get_key_bitlen(const mbedtls_cipher_context_t *ctx)	
Prototype	static inline int mbedtls_cipher_get_key_bitlen(const mbedtls_cipher_context_t *ctx)
Parameter	ctx: The context of the cipher. This must be initialized.
Return	The key length of the cipher in bits. MBEDTLS_KEY_LENGTH_NONE if ctx has not been initialized.
Description	This function returns the key length of the cipher.
static inline mbedtls_cipher_mode_t mbedtls_cipher_get_cipher_mode(const mbedtls_cipher_context_t *ctx)	
Prototype	static inline mbedtls_cipher_mode_t mbedtls_cipher_get_cipher_mode(const mbedtls_cipher_context_t *ctx)
Parameter	ctx: The context of the cipher. This must be initialized.
Return	The mode of operation. MBEDTLS_MODE_NONE if ctx has not been initialized.
Description	This function returns the mode of operation for the cipher.
static inline int mbedtls_cipher_get_iv_size(const mbedtls_cipher_context_t *ctx)	
Prototype	static inline int mbedtls_cipher_get_iv_size(const mbedtls_cipher_context_t *ctx)
Parameter	ctx: The context of the cipher. This must be initialized.

Item	Description
Return	The recommended IV size if no IV has been set. 0 for ciphers not using an IV or nonce. The actual size if an IV has been set.
Description	This function returns the size of the IV or nonce of the cipher, in bytes.
static inline unsigned int mbedtls_cipher_get_block_size(const mbedtls_cipher_context_t *ctx)	
Prototype	static inline unsigned int mbedtls_cipher_get_block_size(const mbedtls_cipher_context_t *ctx)
Parameter	ctx: The context of the cipher. This must be initialized.
Return	The block size of the underlying cipher. 0 if ctx has not been initialized.
Description	This function returns the block size of the given cipher.
static inline const char *mbedtls_cipher_get_name(const mbedtls_cipher_context_t *ctx)	
Prototype	static inline const char *mbedtls_cipher_get_name(const mbedtls_cipher_context_t *ctx)
Parameter	ctx: The context of the cipher. This must be initialized.
Return	The name of the cipher. NULL if ctx is not initialized.
Description	This function returns the name of the given cipher as a string.
int mbedtls_cipher_setkey(mbedtls_cipher_context_t *ctx, const unsigned char *key, int key_bitlen, const mbedtls_operation_t operation)	
Prototype	int mbedtls_cipher_setkey(mbedtls_cipher_context_t *ctx, const unsigned char *key, int key_bitlen, const mbedtls_operation_t operation)
Parameter	ctx: The generic cipher context. This must be initialized and bound to a cipher information structure. key: The key to use. This must be a readable buffer of at least key_bitlen Bits. key_bitlen: The key length to use, in Bits. operation: The operation that the key is used for: MBEDTLS_ENCRYPT or MBEDTLS_DECRYPT.
Return	0 on success. MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure. A cipher-specific error code on failure.
Description	This function sets the key to use with the given context
int mbedtls_cipher_set_iv(mbedtls_cipher_context_t *ctx, const unsigned char *iv, size_t iv_len)	
Prototype	int mbedtls_cipher_set_iv(mbedtls_cipher_context_t *ctx, const unsigned char *iv, size_t iv_len)
Parameter	ctx: The generic cipher context. This must be initialized and bound to a cipher information structure. iv: The IV to use, or NONCE_COUNTER for CTR-mode ciphers. This must be a readable buffer of at least iv_len bytes. iv_len: The IV length for ciphers with variable-size IV. This parameter is discarded by ciphers with fixed-size IV.
Return	0 on success. MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure.
Description	This function sets the initialization vector (IV) or nonce.
int mbedtls_cipher_reset(mbedtls_cipher_context_t *ctx)	
Prototype	int mbedtls_cipher_reset(mbedtls_cipher_context_t *ctx)
Parameter	ctx: The generic cipher context. This must be initialized.

Item	Description
Return	0 on success. MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure.
Description	This function resets the cipher state.
int mbedtls_cipher_update(mbedtls_cipher_context_t *ctx, const unsigned char *input, size_t ilen, unsigned char *output, size_t *olen)	
Prototype	int mbedtls_cipher_update(mbedtls_cipher_context_t *ctx, const unsigned char *input, size_t ilen, unsigned char *output, size_t *olen)
Parameter	ctx: The generic cipher context. This must be initialized and bound to a key. input: The buffer holding the input data. This must be a readable buffer of at least ilen bytes ilen: The length of the input data. output: The buffer for the output data. This must be able to hold at least ilen + block_size. This must not be the same buffer as input. olen: The length of the output data, to be updated with the actual number of bytes written. This must not be NULL.
Return	0 on success. MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure. MBEDTLS_ERR_CIPHER_FEATURE_UNAVAILABLE on an unsupported mode for a cipher. A cipher-specific error code on failure.
Description	The generic cipher update function. It encrypts or decrypts using the given cipher context. Writes as many block-sized blocks of data as possible to output. Any data that cannot be written immediately is either added to the next block or flushed when mbedtls_cipher_finish() is called. Exception: For MBEDTLS_MODE_ECB, expects a single block in size. For example, 16 bytes for AES.
int mbedtls_cipher_finish(mbedtls_cipher_context_t *ctx, unsigned char *output, size_t *olen)	
Prototype	int mbedtls_cipher_finish(mbedtls_cipher_context_t *ctx, unsigned char *output, size_t *olen)
Parameter	ctx: The generic cipher context. This must be initialized and bound to a key. output: The buffer to write data to. This needs to be a writable buffer of at least block_size bytes olen: The length of the data written to the output buffer. This may not be NULL.
Return	0 on success. MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure. MBEDTLS_ERR_CIPHER_FULL_BLOCK_EXPECTED on decryption expecting a full block but not receiving one. MBEDTLS_ERR_CIPHER_INVALID_PADDING on invalid padding while decrypting. A cipher-specific error code on failure.
Description	The generic cipher finalization function. If data still needs to be flushed from an incomplete block, the data contained in it is padded to the size of the last block and written to the output buffer.
int mbedtls_cipher_auth_encrypt(mbedtls_cipher_context_t *ctx, const unsigned char *iv, size_t iv_len, const unsigned char *ad, size_t ad_len, const unsigned char *input, size_t ilen, unsigned char *output, size_t *olen, unsigned char *tag, size_t tag_len)	
Prototype	int mbedtls_cipher_auth_encrypt(mbedtls_cipher_context_t *ctx, const unsigned char *iv, size_t iv_len, const unsigned char *ad, size_t ad_len, const unsigned char *input, size_t ilen, unsigned char *output, size_t *olen, unsigned char *tag, size_t tag_len)

Item	Description
Parameter	ctx: The generic cipher context. This must be initialized and bound to a key. iv: The IV to use, or NONCE_COUNTER for CTR-mode ciphers. This must be a readable buffer of at least iv_len bytes. iv_len: The IV length for ciphers with variable-size IV. This parameter is discarded by ciphers with fixed-size IV. ad: The additional data to authenticate. This must be a readable buffer of at least ad_len bytes ad_len: The length of ad. input: The buffer holding the input data. This must be a readable buffer of at least ilen bytes ilen: The length of the input data. output: The buffer for the output data. This must be able to hold at least ilen bytes. olen: The length of the output data, to be updated with the actual number of bytes written. This must not be NULL. tag: The buffer for the authentication tag. This must be a writable buffer of at least tag_len bytes tag_len: The desired length of the authentication tag.
Return	0 on success. MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure. A cipher-specific error code on failure.
Description	The generic authenticated encryption (AEAD) function.
int mbedtls_cipher_auth_decrypt(mbedtls_cipher_context_t *ctx, const unsigned char *iv, size_t iv_len, const unsigned char *ad, size_t ad_len, const unsigned char *input, size_t ilen, unsigned char *output, size_t *olen, const unsigned char *tag, size_t tag_len)	
Prototype	int mbedtls_cipher_auth_decrypt(mbedtls_cipher_context_t *ctx, const unsigned char *iv, size_t iv_len, const unsigned char *ad, size_t ad_len, const unsigned char *input, size_t ilen, unsigned char *output, size_t *olen, const unsigned char *tag, size_t tag_len)
Parameter	ctx: The generic cipher context. This must be initialized and bound to a key. iv: The IV to use, or NONCE_COUNTER for CTR-mode ciphers. This must be a readable buffer of at least iv_len bytes. iv_len: The IV length for ciphers with variable-size IV. This parameter is discarded by ciphers with fixed-size IV. ad: The additional data to be authenticated. This must be a readable buffer of at least ad_len bytes. ad_len: The length of ad. input: The buffer holding the input data. This must be a readable buffer of at least ilen bytes ilen: The length of the input data. output: The buffer for the output data. This must be able to hold at least ilen bytes. olen: The length of the output data, to be updated with the actual number of bytes written. This must not be NULL. tag: The buffer holding the authentication tag. This must be a readable buffer of at least tag_len bytes. tag_len: The length of the authentication tag.
Return	0 on success. MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA on parameter-verification failure. MBEDTLS_ERR_CIPHER_AUTH_FAILED if data is not authentic. A cipher-specific error code on failure.
Description	The generic authenticated decryption (AEAD) function.

17. Peripheral and System Examples

17.1 UART

Along with a UART0 interface for the debug console, the DA16200 SDK has a UART1 or UART2 interface to communicate with an external MCU. GPIOA[4] and GPIOA[5] can be used to this interface.

17.1.1 Introduction

The DA16200/DA16600 has two UARTs (Universal Asynchronous Receiver-Transmitter), which have the following features:

- Programmable use of UART
- Compliance to the AMBA AHB bus specification for easy integration into SoC implementation
- Support both byte and word access for reduction of bus burden
- Support both RS-232 and RS-485
- Separate 32x8 bit transmit and 32x12 bit receive FIFO memory buffers to reduce CPU interrupts
- Programmable FIFO disabling for 1-byte depth
- Programmable baud rate generator
- Standard asynchronous communication bits (start, stop and parity). These are added before transmission and removed upon reception
- Independent masking of transmit FIFO, receive FIFO, receive timeout
- Support for Direct Memory Access (DMA)
- False start bit detection
- Programmable flow control
- Fully programmable serial interface characteristics:
  - Data can be 5, 6, 7 or 8 bits
  - Even, odd, stick or no-parity bit generation and detection
  - 1 or 2 stop bit generation
  - Baud rate generation

Table 59. UART pin configuration

Pin name	Pin number		I/O	Function name
	QFN	fcCSP		
UART0_RXD	12	M10	I	UART0_RXD
UART0_TXD	11	L9	O	UART0_TXD
GPIOA7	31	E1	I	UART1_RXD
GPIOA5	33	D2	I	-
GPIOA3	36	D4	I	-
GPIOA1	38	C3	I	-
GPIOA6	32	E3	O	UART1_TXD
GPIOA4	34	F4	O	-
GPIOA2	37	B2	O	-
GPIOA0	39	A3	O	-
GPIOA5	33	D2	I	UART1_CTS
GPIOA4	34	F4	O	UART1_RTS
GPIOA11	27	G1	I	UART2_RXD
GPIOC7	9	K12	I	
F_IO2	16	J7	I	

Pin name	Pin number		I/O	Function name
	QFN	fcCSP		
GPIOA10	28	F2	O	UART2_TXD
GPIOC6	10	L11	O	
F_IO3	17	K6	O	

17.1.2 API

Table 60. APIs for UART interface

Item		Description
HANDLE UART_CREATE(UART_UNIT_IDX dev_idx)		
Parameter	dev_idx	Device index
Return		If it succeeds, return handle for such device. If it fails, return NULL.

Description		Function to create a handle with parameter dev_idx designated.
		The DA16200/DA16600 has two UART ports.
		`typedef enum __uart_unit__ {`
		`UART_UNIT_0 = 0,`
		`UART_UNIT_1,`
		`UART_UNIT_MAX`
		`} UART_UNIT_IDX;`
		Normally, UART0 is used for debug console, and UART1 is used for data transfer.

int UART_INIT (HANDLE handler)		
Parameter	handler	Device handle
Return		TRUE if it succeeds, or FALSE if it fails.
Description		The UART configuration should be set before this function is called. After this function is called, UART operation starts.
int UART_CHANGE_BAUERATE (HANDLE handler, UINT32 baudrate)		
Parameter	handler	Device handle
	baudrate	Baud rate to set.
Return		TRUE if it succeeds, or FALSE if it fails.
Description		This function changes the baud rate of UART during UART operation.
int UART_IOCTL(HANDLE handler, UINT32 cmd, VOID *data)		
Parameter	handler	Device handle
	cmd	Commands are defined in <uart.h> in the DA16200/DA16600 SDK.
	data	Data pointer.
Return		TRUE if it succeeds, or FALSE if it fails.

Item	Description
Description	The user can set the configuration of UART with this function. Configurations of UART should be called before the UART_INIT() function. Commands are as below: ▪ UART_GET_DEVREG = 1, // get device physical address ▪ UART_SET_CLOCK, // set base clock ▪ UART_SET_BAUDRATE, // set baud rate ▪ UART_GET_BAUDRATE, // get baud rate ▪ UART_SET_LINECTRL, // set line control ▪ UART_GET_LINECTRL, // get line control ▪ UART_SET_CONTROL, // set UART control ▪ UART_GET_CONTROL, // get UART control ▪ UART_SET_SW_RX_QUESIZE, // set queue size ▪ UART_SET_INT, // set interrupt configuration ▪ UART_GET_INT, // get interrupt configuration ▪ UART_SET_FIFO_INT_LEVEL, // set FIFO level ▪ UART_GET_FIFO_INT_LEVEL, // get FIFO level ▪ UART_SET_USE_DMA, // set DMA use ▪ UART_GET_USE_DMA, // get DMA use ▪ UART_CHECK_RXEMPTY, // check RX FIFO empty ▪ UART_CHECK_RXFULL, // check RF FIFO full ▪ UART_CHECK_TXEMPTY, // check TX FIFO empty ▪ UART_CHECK_TXFULL, // check TX FIFO full ▪ UART_CHECK_BUSY, // check UART busy ▪ UART_SET_RX_SUSPEND, // set the RX function to suspend ▪ UART_GET_RX_SUSPEND, // get the RX function to suspend ▪ UART_SET_SW_FLOW_CONTROL, // set the flow control to enable ▪ UART_GET_SW_FLOW_CONTROL, // get the flow control to enable ▪ UART_SET_WORD_ACCESS, // set word-access-enable register ▪ UART_GET_WORD_ACCESS, //get word-access-enable register ▪ UART_SET_RW_WORD, // set whether write and read in word or byte ▪ UART_GET_RW_WORD, // get whether write and read in word or byte ▪ UART_SET_RS485, // set the RS485 function to enable ▪ UART_GET_RS485, // get the RS485 function to enable ▪ UART_CLEAR_ERR_INT_CNT, // clear error interrupt counter ▪ UART_GET_ERR_INT_CNT, // get error interrupt counter ▪ UART_SET_ERR_INT_CALLBACK, // set error interrupt callback function ▪ UART_CLEAR_FRAME_INT_CNT, //clear frame error interrupt counter ▪ UART_GET_FRAME_INT_CNT, // get frame error interrupt counter ▪ UART_SET_FRAME_INT_CALLBACK, // set frame error interrupt callback ▪ UART_CLEAR_PARITY_INT_CNT, // clear parity error interrupt counter ▪ UART_GET_PARITY_INT_CNT, // get frame error interrupt counter ▪ UART_SET_PARITY_INT_CALLBACK, // set frame error interrupt callback ▪ UART_CLEAR_BREAK_INT_CNT, // clear break error interrupt counter ▪ UART_GET_BREAK_INT_CNT, // get break error interrupt counter ▪ UART_SET_BREAK_INT_CALLBACK, // set break error interrupt callback ▪ UART_CLEAR_OVERRUN_INT_CNT, // clear overrun error interrupt counter ▪ UART_GET_OVERRUN_INT_CNT, // get overrun error interrupt counter ▪ UART_SET_OVERRUN_INT_CALLBACK, // set overrun interrupt callback

Item		Description
int UART_READ (HANDLE handler, VOID *p_data, UINT32 p_dlen)		
Parameter	handler	Device handle
	p_data	Data pointer
	p_dlen	Length to read.
Return		If it succeeds, return number of received data, else negative number.
Description		User can use the UART_SET_RX_SUSPEND ioctl command to set the UART READ operation to suspend or not.
int UART_WRITE (HANDLE handler, VOID *p_data, UINT32 p_dlen)		
Parameter	handler	Device handle
	p_data	Data pointer
	p_dlen	Length to write
Return		Number of sent data
Description		UART write command
int UART_DMA_READ_TIMEOUT (HANDLE handler, VOID *p_data, UINT32 p_dlen, UINT32 timeout)		
Parameter	handler	Device handle
	p_data	Data pointer
	p_dlen	Length to read
	timeout	Wait option to receive data
Return		Number of received data
Description		The operation of this function is the same with UART_DMA_READ with waiting timeout.
int UART_DMA_READ (HANDLE handler, VOID *p_data, UINT32 p_dlen)		
Parameter	handler	Device handle
	p_data	Data pointer
	p_dlen	Length to read
Return		Number of received data
Description		The operation of this function is the same with UART_READ, except DMA is used.
int UART_DMA_WRITE (HANDLE handler, VOID *p_data, UINT32 p_dlen)		
Parameter	handler	Device handle
	p_data	Data pointer
	p_dlen	Length to write
Return		Number of sent data
Description		The operation of this function is the same with UART_WRITE, except DMA is used.
int UART_FLUSH(HANDLE handler)		
Parameter	handler	Device handle
Return		TRUE if it succeeds, or FALSE if it fails.
Description		Flush the FIFO buffer of UART
int UART_CLOSE(HANDLE handler)		
Parameter	handler	Device handle
Return		TRUE if it succeeds, or FALSE if it fails.
Description		UART driver close command.

17.1.3 How to Run

1. In the e[2] studio, import a project for the UART sample application as follows.
   `~/SDK/apps/common/examples/Peripheral/UART1/projects/da16200`
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. The start log message is shown in the console terminal and UART1 terminal.
4. To test with UART1, input test data (hexa or ascii) on the UART1 terminal and click the Enter key to send data to DA16200. Then the console terminal shows the received data in hexadecimal and sends the message "- Data receiving OK…" to UART1.
   a. UART1 terminal

Figure 112. Result of UART #1

   b. Console terminal

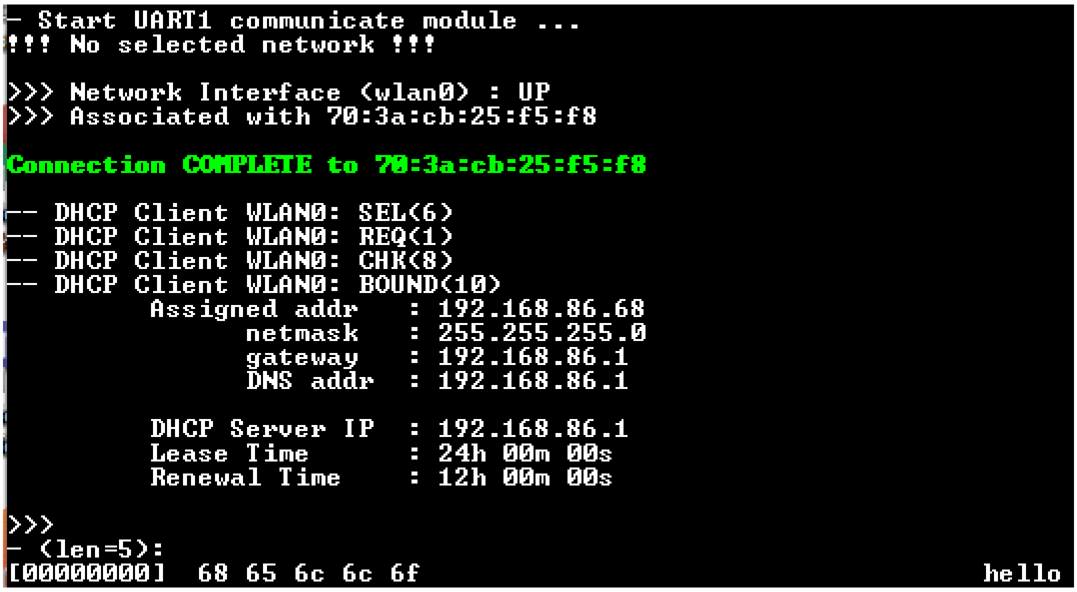


Figure 113. Result of UART #2

17.1.4 Sample Code

17.1.4.1 Application Initialization

This is an example of a user application to initialize and communicate between the DA16200 and an MCU that is connected through the UART1 interface. Function `user_uart1_init()` initializes the UART1 hardware resource and then `uart1_monitor_sample()` is run to communicate with the host through the UART1 interface.

```
~/SDK/apps/common/examples/Peripheral/UART1/src/uart_sample.c

/* Local static variables */
static int  sample_uart_idx = UART_UNIT_1;    // UART_UNIT_1, UART_UNIT_2

/*
 * For configuring UART devices,
 *
 * "user_UART_config_info" data should be located in /SDK/customer/src/user_uart.c.
 *
 * This data is temporary for sample application.
 */
static uart_info_t sample_UART_config_info =
{
```

```
    UART_BAUDRATE_115200,   /* baud */
    UART_DATABITS_8,        /* bits */
    UART_PARITY_NONE,       /* parity */
    UART_STOPBITS_1,        /* stopbit */
    UART_FLOWCTL_OFF        /* flow control */
};

void run_uart1_sample(UINT32 arg)
{
    int status;

    *
    * int set_user_UART_conf(int uart_idx, uart_info_t *uart_conf_info, char
      atcmd_flag)
    */
    status = set_user_UART_conf(UART_UNIT_1, &sample_UART_config_info, FALSE);
    if (status != 0)
    {
        PRINTF("[%S] Error to configure for UART1 !!!\n", __func__);
        return;
    }


    *
    * int UART_init(int uart_idx);
    */
    status = UART_init(sample_uart_idx);
    if (status != 0)
    {
        PRINTF("[%S] Error to initialize UART1 with sample_UART_config !!!\n",
        __func__);
    return;
    }

    /* Start UART monitor */
    uart1_sample();
}
```

Function `uart1_sample()` invokes function `get_data_from_uart1()` repeatedly to read data from UART1. User can enable/disable the UART echo function by setting `echo_enable`.

```
static void uart1_sample(void)
{
    int i;
    char *init_str = "- Start UART1 communicate module ...\r\n";
    char *rx_buf = NULL;
    char *tx_buf = "\r\n- Data receiving OK...\r\n";
    int tx_len;

    /* Print-out test string to console and to UART1 device */
    PRINTF((const char *)init_str); // For Console
    puts_UART(sample_uart_idx, init_str, strlen((const char *)init_str));

    echo_enable = TRUE;
    rx_buf = malloc(USER_UART1_BUF_SZ);

    while (1)
    {
        memset(rx_buf, 0, USER_UART1_BUF_SZ);

        /* Get on byte from uart1 comm port */
        get_data_from_uart1(rx_buf);
```

```
        … …
    }
}
```

17.1.4.2 Data Read/Write

Use `getchar_UART()` to read a character from UART1 or UART2. This example shows how to read data from UART device until it meets characters '\n' or '\r'. Users can modify this function for customized application operation.

NOTE
After UART_INIT() is called, it tries to receive UART_RX data even if UART_READ() does not call. Flush the UART_RX data if there is no need to use data before UART_READ().

```
#define USER_DELIMITER_0    '\0'
#define USER_DELIMITER_1    '\n'
#define USER_DELIMITER_2    '\r'

static void get_data_from_uart1(char *buf)
{
    char    ch = 0;
    int     i = 0;

    while (1) {
        /* Get on byte from uart1 comm port */
        ch = getchar_UART(sample_uart_idx, portMAX_DELAY);

        if (ch == 0) {
            vTaskDelay(1);
            continue;
        }

        if (echo_enable == TRUE) {
            puts_UART(sample_uart_idx, &ch, sizeof(char)); // echo
        }

        /* check data length */
        if (i >= (USER_UART1_BUF_SZ - 1)) {
            i = USER_UART1_BUF_SZ - 2;
        }

        if (ch == USER_DELIMITER_1 || ch == USER_DELIMITER_2) {
            buf[i++] = USER_DELIMITER_0;
            break;
        } else {
            buf[i++] = ch;
        }
    }
}
```

Also, this example shows how to send data to UART1 using `puts_UART()` API.

```
~/SDK/core/system//include/common/common_uart.h

/**
 *****************************************************************************************
 * @brief  Put character string to UART device
 * @param[in] uart_idx  Index value of UART interface (UART_UNIT_1, UART_UNIT_2)
 * @param[in] *data     Text string to write
```

```
 * @param[in] len        Write length
 * @return               None
 ****************************************************************************************
 */
void    puts_UART(int uart_idx, char *data, int data_len);
```

17.2 GPIO

This application shows how to read/write the GPIO port and use the GPIO interrupt.

17.2.1 Introduction

All digital pads can be used as GPIO. Each GPIO port is mixed with a multi-functional interface. The GPIO features for this device are:

- Input or output lines in a programmable direction
- Word and half word read/write access
- Address-masked byte writes to facilitate quick bit set and clear operations
- Address-based byte reads to facilitate quick bit test operations
- Make a GPIO pin to an interrupt pin possible to be the output signal of PWM [3:0], external Interrupt, SPI_CSB [3:1], RF_SW [1:0] and UART_TXDOE [1:0] on any GPIO pin

It provides special functions for GPIO pin use. PWM [3:0], external interrupt, SPI_CSB [3:1], RF_SW [1:0] and UART_TXDOE [1:0] signals can be output if any of the unused pins among the GPIO pins are selected. It is possible to select the function to be output from the GPIO register setting and select the remaining GPIO pin and not output the specific function to any desired GPIO pin.

Table 61. GPIO pin configuration

Pin name	Pin number	I/O	Pin selection	Function name
GPIOA0	39	I/O	Reg. GPIO_SEL.AMUX9	GPIOA[0]
GPIOA1	38	I/O	Reg. GPIO_SEL.AMUX9	GPIOA[1]
GPIOA2	37	I/O	Reg. GPIO_SEL.BMUX9	GPIOA[2]
GPIOA3	36	I/O	Reg. GPIO_SEL.BMUX9	GPIOA[3]
GPIOA4	34	I/O	Reg. GPIO_SEL.CMUX9	GPIOA[4]
GPIOA5	33	I/O	Reg. GPIO_SEL.CMUX9	GPIOA[5]
GPIOA6	32	I/O	Reg. GPIO_SEL.DMUX9	GPIOA[6]
GPIOA7	31	I/O	Reg. GPIO_SEL.DMUX9	GPIOA[7]
GPIOA8	30	I/O	Reg. GPIO_SEL.EMUX9	GPIOA[8]
GPIOA9	29	I/O	Reg. GPIO_SEL.EMUX9	GPIOA[9]
GPIOA10	28	I/O	Reg. GPIO_SEL.FMUX7	GPIOA[10]
GPIOA11	27	I/O	Reg. GPIO_SEL.FMUX7	GPIOA[11]
GPIOC6	10	I/O	Reg. GPIO_SEL.UMUX2	GPIOC[6]
GPIOC7	9	I/O	Reg. GPIO_SEL.UMUX2	GPIOC[7]
GPIOC8	8	I/O	Reg. GPIO_SEL.UMUX2	GPIOC[8]

When keeping GPIO PIN state high or low in sleep state, use one of the following API functions:

- _GPIO_RETAIN_HIGH()
- _GPIO_RETAIN_LOW().

GPIOA [11:4] and GPIOC [8:6] can only be set as GPIO retention high or low. When using GPIO and GPIO Retention API, the status of GPIO PIN is shown in Table 62.

Table 62. Status of GPIO pin

	PIN information	Before sleep (RTOS booting)	Sleep period	Sleep period (with SAVE_PULLUP_PINS_INFO)	After sleep (wake-up)
GPIO input configured	GPIOA[3:0]	High-Z	High-Z	High-Z	I-PD
	GPIOA[11:8], GPIOC[8:6]	High-Z	Low (PD)	High-Z	I-PD
GPIO output high configured	GPIOA[3:0]	High	High-Z	High-Z	I-PD
	GPIOA[11:8], GPIOC[8:6]	High	Low (PD)	High-Z	I-PD
GPIO output low configured	GPIOA[3:0]	Low	High-Z	High-Z	I-PD
	GPIOA[11:8], GPIOC[8:6]	Low	low (PD)	High-Z	I-PD
GPIO retention high configured	GPIOA[11:8], GPIOC[8:6]	High	High	High	High
GPIO retention low configured	GPIOA[11:8], GPIOC[8:6]	Low	Low	Low	Low

When keeping GPIO PIN in high-z state in sleep period, use the API described in the Section 17.2.2:

- SAVE_PULLUP_PINS_INFO()

This function should be used when an external pull-up register is connected to a GPIO PIN. If this function is not used, leakage current may occur.

17.2.2 API

Table 63. APIs for GPIO interface

Item		Description
HANDLE GPIO_CREATE(UINT32 dev_type)		
Parameter	dev_type	Device index
Return		If it succeeds, return the handle for the device. If it fails, return NULL.
Description		The DA16200/DA16600 can set GPIO_UNIT_A and GPIO_UNIT_C.
int GPIO_INIT (HANDLE handler)		
Parameter	handler	Device handle
Return		TRUE if it succeeds, or FALSE if it fails.
Description		Configure the GPIO setting.
int GPIO_IOCTL(HANDLE handler, UINT32 cmd, VOID *data)		
Parameter	handler	Device handle
	cmd	Commands are defined <gpio.h> in our SDK.
	data	Data pointer.
Return		TRUE if it succeeds, or FALSE if it fails.

Item		Description
Description		The necessary configuration of GPIO can be set with this function. Commands are as below: ▪ GPIO_GET_DEVREG = 1, ▪ GPIO_SET_OUTPUT, // set gpio as an output ▪ GPIO_SET_INPUT, // set gpio as an input ▪ GPIO_GET_DIRECTION, // get gpio direction ▪ GPIO_SET_INTR_MODE, // set gpio interrupt mode [edge/level] ▪ GPIO_GET_INTR_MODE, // get gpio interrupt mode ▪ GPIO_SET_INTR_ENABLE, // enable gpio interrupt ▪ GPIO_SET_INTR_DISABLE, // disable gpio interrupt ▪ GPIO_GET_INTR_ENABLE, // get gpio interrupt enable status ▪ GPIO_GET_INTR_STATUS, // get gpio interrupt pending status ▪ GPIO_SET_INTR_CLEAR, // clear gpio interrupt status ▪ GPIO_SET_MODE_ALT, // set alternate function ▪ GPIO_SET_MODE_NOALT, // clear alternate function ▪ GPIO_GET_MODE_ALT, // get alternate function ▪ GPIO_SET_CALLACK, // set a callback function for gpio interrupt
int GPIO_READ (HANDLE handler, UINT32 addr, UINT16 *pdata, UINT32 dlen)		
Parameter	handler	Device handle
	addr	GPIO index
	p_data	Data buffer pointer
	p_dlen	Data buffer length
Return		TRUE if it succeeds, or FALSE if it fails.
Description		GPIO value contained in p_data.
int GPIO_WRITE (HANDLE handler, UINT32 addr, VOID *p_data, UINT32 p_dlen)		
Parameter	handler	Device handle
	addr	GPIO index
	p_data	Data buffer pointer
	p_dlen	Data buffer length
Return		TRUE if it succeeds, or FALSE if it fails.
Description		GPIO value contained in p_data.
int GPIO_CLOSE(HANDLE handler)		
Parameter	handler	Device handle
Return		TRUE if it succeeds, or FALSE if it fails.
Description		GPIO close command.
INT32 GPIO_GET_ALT_FUNC (HANDLE handler, GPIO_ALT_FUNC_TYPE altFuncType, UINT32 * regVal)		
Parameter	handler	Device handle
	altFuncType	GPIO alternate function type
	regVal	GPIO alternate function setting value
Return		If it succeeds, return 0.
Description		Get GPIO alternate function setting value.

Item		Description
INT32 GPIO_SET_ALT_FUNC(HANDLE handler, GPIO_ALT_FUNC_TYPE altFuncType, GPIO_ALT_GPIO_NUM_TYPE gpioType)		
Parameter	handler	Device handle
	altFuncType	GPIO alternate function type
	gpioType	GPIO number
Return		If it succeeds, return 0.
Description		Set GPIO alternate function.
INT32 _GPIO_RETAIN_HIGH(UINT32 gpio_port, UINT32 gpio_num)		
Parameter	gpio_port	GPIO port number
	gpio_num	GPIO pin number
Return		TRUE if successfully configured, else FALSE.
Description		GPIOA[11:4] and GPIOC[8:6] are only available to set GPIO retention high. And this API function should not be called from the "config_pin_mux" function.
INT32 _GPIO_RETAIN_LOW(UINT32 gpio_port, UINT32 gpio_num)		
Parameter	gpio_port	GPIO port number
	gpio_num	GPIO pin number
Return		TRUE if successfully configured, else FALSE.
Description		GPIOA[11:4] and GPIOC[8:6] are only available to set GPIO retention high. And this API function should not be called from the "config_pin_mux" function.
void SAVE_PULLUP_PINS_INFO(UINT32 port_num, UINT32 pinnum)		
Parameter	port_num	GPIO port number
	pinnum	GPIO pin number
Description		It keeps GPIO PIN in High-Z state during sleep period. This function should be used when an external pull-up register is connected to a GPIO PIN. If this function is not used, leakage current may occur.

17.2.3 How to Run

1. In the e² studio, import a project for the GPIO sample application as follows.
   `~/SDK/apps/common/examples/Peripheral/GPIO/projects/da16200`
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. The status of GPIOA[0] and GPIOA[1] is printed every 1 second.
   - GPIOA[0] output low, GPIOA[4] output low, GPIOA[1] input low
   - GPIOA[0] output high, GPIOA[4] output high, GPIOA[1] input low
   - GPIOA[0] output low, GPIOA[4] output low, GPIOA[1] input low

17.2.4 Sample Code

1. Create and initialize a GPIO handle.

```
HANDLE gpio;
gpio = GPIO_CREATE(GPIO_UNIT_A);
GPIO_INIT(gpio);
```

2. Set pin multiplexing.

```
/* AMUX to GPIOA[1:0] */
_da16x_io_pinmux(PIN_AMUX, AMUX_GPIO);
```

```
/* BMUX to GPIOA[3:2] */
_da16x_io_pinmux(PIN_BMUX, BMUX_GPIO);

/* CMUX to GPIOA[5:4] */
_da16x_io_pinmux(PIN_CMUX, CMUX_GPIO);
```

3. Set GPIOA[0] and GPIOA[4] as output mode and GPIOA[1] as input mode.

```
/* GPIOA[0],GPIOA[4] output high low toggle */
pin = GPIO_PIN0 | GPIO_PIN4;
GPIO_IOCTL(gpio, GPIO_SET_OUTPUT, &pin); /* GPIOA[1] input */
pin = GPIO_PIN1;
GPIO_IOCTL(gpio, GPIO_SET_INPUT, &pin);
```

4. Set GPIOA[2] as an interrupt source with active low and register a callback function.

```
static int set_gpio_interrupt(HANDLE handler, UINT8 pin_num, UINT8 int_type, UINT8 int_pol,
void *callback_func)
{
      UINT16 pin, int_en_status;
      UINT32 ioctldata[3];
      int ret;

      if (15 < pin_num )
        return FALSE;

      if(handler == NULL){
        return FALSE;
      }

      pin = 0x01<<pin_num;
      ret = GPIO_IOCTL(handler, GPIO_SET_INPUT, &pin);

      ret = GPIO_IOCTL(handler, GPIO_GET_INTR_MODE, &ioctldata[0]);
      /* interrupt type 1: edge, 0: level*/
      ioctldata[0] &= ~(1 << pin_num); // clear the bit first
      ioctldata[0] |= (int_type << pin_num);
      /* interrupt pol 1: high active, 0: low active */
      ioctldata[1] &= ~(1 << pin_num); // clear the bit first
      ioctldata[1] |= (int_pol << pin_num);
      ret = GPIO_IOCTL(handler, GPIO_SET_INTR_MODE, &ioctldata[0]);

      /* register callback function */
      ioctldata[0] = pin; /* interrupt pin */
      ioctldata[1] = (UINT32) callback_func; /* callback function */
      ioctldata[2] = (UINT32) pin_num; /* param data */
      ret = GPIO_IOCTL(handler, GPIO_SET_CALLACK, ioctldata);

      ret = GPIO_IOCTL(handler, GPIO_GET_INTR_ENABLE, &int_en_status);
      int_en_status |= pin;
      ret = GPIO_IOCTL(handler, GPIO_SET_INTR_ENABLE, &int_en_status);

      return ret;
}

/* GPIOA[2] interrupt active low , Edge trigger */
set_gpio_interrupt(gpio, 2, GPIO_INT_TYPE_EDGE, GPIO_INT_POL_LOW, (void*)gpio_callback );
```

5. Set GPIOA[3] as an interrupt source with active high and register a callback function.

```
/*GPIOA[3] interrupt active high, Edge trigger */
set_gpio_interrupt(gpio, 3, GPIO_INT_TYPE_EDGE, GPIO_INT_POL_HIGH, (void*)gpio_callback );
```

6. Write GPIOA[0] and GPIOA[4] and read GPIOA[1].

```
if (toggle) {
    /* GPIOA[0],GPIOA[4] to high */
    write_data = GPIO_PIN0 | GPIO_PIN4;
    GPIO_WRITE(gpio, GPIO_PIN0 | GPIO_PIN4, &write_data, sizeof(UINT16));
    toggle = 0;
} else {
    /* GPIOA[0],GPIOA[4] to low*/
    write_data = 0;
    GPIO_WRITE(gpio, GPIO_PIN0 | GPIO_PIN4, &write_data, sizeof(UINT16));
    toggle = 1;
}

GPIO_READ(gpio, GPIO_PIN1, &read_data, sizeof(UINT16));
```

7. Set the PAD pull condition by using PAD_PULL_CONTROL.

```
#if PAD_PULL_CONTROL
    /*
      * GPIOA[1] input pull control it can make gpio pad pull up or pull down or HIZ
      */
    _da16x_gpio_set_pull(GPIO_UNIT_A, GPIO_PIN1, PULL_UP);
    /* or */
    _da16x_gpio_set_pull(GPIO_UNIT_A, GPIO_PIN1, PULL_DOWN);
    /* or */
    _da16x_gpio_set_pull(GPIO_UNIT_A, GPIO_PIN1, HIGH_Z);
#endif
```

8. Activate the RTC_GPO example by using RTC_GPO_CONTROL.

```
#ifdef RTC_GPO_CONTROL
     RTC_GPO_OUT_INIT(1);                // 0: auto, 1: manual
RTC_GPO_OUT_CONTROL(1);               // Set High
#endif
```

9. Both edges of interrupt are not supported by hardware but can be supported by software.

   Activate the GPIO interrupt according to the GPIO read value.

```
GPIO_READ(gpioc, GPIO_PIN6, &read_data, sizeof(UINT16));
set_gpio_interrupt(gpioc, 6, GPIO_INT_TYPE_EDGE, !(GPIO_PIN6&read_data),
(void*)gpioc_callback );
```

   And change the interrupt polarity at every GPIO callback function. See the "GPIOC6_BOTH_EDGE_INTERRUPT" example for details.

17.3 GPIO Retention

This application shows how to use GPIO retention. If the GPIO pin is set to retention high, it is kept in the high state during the sleep period. If the GPIO pin is set to retention low, it is kept in the low state during the sleep period.

17.3.1 How to Run

1. In the e² studio, import a project for the GPIO Retention sample application.

~/SDK/apps/common/examples/Peripheral/GPIO_Retention/projects/da16200

a. Build the main project, download the image to the DA16200 EVB, and reboot.
b. Toggle switch 13 (SW13).
c. Use an oscilloscope to check that the GPIOA [10: 8] and GPIOC [7] keep their PIN states.

17.3.2 Sample Code

1. Set pin multiplexing.

```
/*
 * 1. Set to GPIOA[11:8], GPIOC[8:6]
 * 2. Need be written to "config_pin_mux" function.
 */
_da16x_io_pinmux(PIN_EMUX, EMUX_GPIO);
_da16x_io_pinmux(PIN_FMUX, FMUX_GPIO);
_da16x_io_pinmux(PIN_UMUX, UMUX_GPIO);
```

2. Set GPIO retention config.

```
/* Set GPIOA[9:8] to retention high */
ret = _GPIO_RETAIN_HIGH(GPIO_UNIT_A, GPIO_PIN8 | GPIO_PIN9);
if(ret == FALSE)
    PRINTF("GPIO_RETAIN_HIGH() return false.\n");

/* Set GPIOA[10] to retention low */
ret = _GPIO_RETAIN_LOW(GPIO_UNIT_A, GPIO_PIN10);
if (ret == FALSE)
    PRINTF("GPIO_RETAIN_LOW() return false.\n");

/* Set GPIOC[7] to retention high */
ret = _GPIO_RETAIN_HIGH(GPIO_UNIT_C, GPIO_PIN7);
if(ret == FALSE)
    PRINTF("GPIO_RETAIN_HIGH() return false.\n");
```

3. Power down.

```
char * _argv[4] = {"down", "sec", "10", "1"};
cmd_power_down_config(4, _argv);

/* Set GPIOC[7] to retention high */
ret = _GPIO_RETAIN_HIGH(GPIO_UNIT_C, GPIO_PIN7);
if(ret == FALSE)
    PRINTF("GPIO_RETAIN_HIGH() return false.\n");
```

17.4 I2C

This section shows how to use the I2C interface.

17.4.1 Introduction

17.4.1.1 I2C Master

The DA16200/DA16600 includes an I2C master module. There are two supportable clock speeds for I2C in the DA16200/DA16600; the standard speed is 100 kbps, and fast mode is 400 kbps. Table 64 shows the pin definition of the I2C master interface in GPIO Pin Configuration.

Table 64. I2C master pin configuration

Pin name	Pin number		I/O	Function name
	QFN	fcCSP		
GPIOA1	38	C3	O	I2C_CLK
GPIOA5	33	D2	O	
GPIOA9	29	H2	O	
GPIOA0	39	A3	I/O	I2C_SDA
GPIOA4	34	F4	I/O	
GPIOA8	32	G3	I/O	

For more details, see Ref. [2].

17.4.1.2 I2C Slave

The DA16200/DA16600 supports the I2C slave interface controlled by an external host. The pin mux configurations are defined in Table 65. The I2C slave interface also supports the standard (100 kbps) or fast (400 kbps) transmission speeds.

Table 65. I2C slave pin configuration

Pin name	Pin number		I/O	Function name
	QFN	fcCSP		
GPIOA1	38	C3	I	I2C_CLK
GPIOA3	36	D4	I	
GPIOA5	33	D2	I	
GPIOA7	31	E1	I	
GPIOA0	39	A3	I/O	I2C_SDA
GPIOA2	37	B2	I/O	
GPIOA4	34	F4	I/O	
GPIOA6	32	E3	I/O	

For more details, see Ref. [2].

17.4.2 API

Table 66. APIs for I2C interface

Item		Description
HANDLE DRV_I2C_CREATE(UINT32 dev_id)		
Parameter	dev_id	Device ID number to create a handle.
Return		If it succeeds, return the handle for the device. If it fails, return NULL.
Description		Create a handle with the parameter "dev_id" designated.
Int DRV_I2C_INIT(HANDLE handler)		
Parameter	handler	Device handle to initialize.
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		Initialize the I2C interface.
int DRV_I2C_IOCTL(HANDLE handler, UINT32 cmd, VOID *data)		
Parameter	handler	Device handle to control.
	cmd	See **sys_i2c.h** in our SDK.
	*data	Data pointer when there are any. If not, NULL.

Item		Description	
Return		If it succeeds, return TRUE. If it fails, return FALSE.	
Description		This function controls miscellaneous I2C controller.	
int DRV_I2C_IOCTL(HANDLE handler, UINT32 cmd, VOID *data)			
I2C_GET_CONFIG		Get "i2c_cr0" Register Value. See Register Map	Read
I2C_GET_STATUS		Get "i2c_sr" Register Value. See Register Map	Read
I2C_SET_DMA_WR		I2C Write via uDMA TX Enable/Disable	[TRUE/FALSE]
I2C_SET_DMA_RD		I2C READ via uDMA RX Enable/Disable	[TRUE/FALSE]
I2C_GET_DMA_WR		Get uDMA TX Enabled	[0x2/FALSE]
I2C_GET_DMA_RD		Get uDMA RX Enabled	[TRUE/FALSE]
I2C_SET_RESET		Set I2C Device Reset/set	[TRUE/FALSE]
I2C_SET_CHIPADDR		Set I2C Slave Device Address (8 bits)	Write
I2C_GET_CHIPADDR		Get I2C Slave Device Address (8 bits)	Read
I2C_SET_CLOCK		Set I2C Clock [kHz] (Max = 1200)	Write
int DRV_I2C_WRITE_DMA(HANDLE handler, VOID *p_data, UINT32 p_dlen, UINT32 dummy)			
Parameter	handler	Device handle to write with DMA	
	*p_data	Buffer pointer to write	
	p_dlen	Length to write	
	dummy	Reserved (set to ‘0’)	
Return		If it succeeds, return TRUE. If it fails, return FALSE.	
Description		I2C write function through DMA.	
int DRV_I2C_WRITE(HANDLE handler, VOID *p_data, UINT32 p_dlen, UINT32 stopen, UINT32 dummy)			
Parameter	handler	Device handle to write	
	*p_data	Buffer pointer to write	
	p_dlen	Length to read	
	stopen	Flag stop bit enables	
	dummy	Reserved (set to ‘0’)	
Return		If it succeeds, return TRUE. If it fails, return FALSE.	
Description		I2C write function	
int DRV_I2C_READ(HANDLE handler, VOID *p_data, UINT32 p_dlen, UINT32 addr_len,UINT32 dummy)			
Parameter	handler	Device handle to read	
	*p_data	Buffer pointer to read	
	p_dlen	Length to read	
	addr_len	Length of register address inside of slave device. if 0, read only operation.	
	dummy	Reserved (set to ‘0’)	
Return		If it succeeds, return TRUE. If it fails, return FALSE.	
Description		I2C read function	
Int DRV_I2C_CLOSE(HANDLE handler);			
Parameter	handler	Device handle to close	
Return		If it succeeds, return TRUE. If it fails, return FALSE.	
Description		I2C driver close	

Item		Description
void DRV_I2C_REGISTER_INTERRUPT (HANDLE handler);		
Parameter	handler	Device handle to register Interrupt Handler
Return		NULL
Description		I2C Interrupt Registration

17.4.3 How to Run

1. In the e[2] studio, import a project for the I2C sample application.
   `~/SDK/apps/common/examples/Peripheral/I2C/projects/da16200`
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
   `The sample application code is written in the following source file:`
   `~/SDK/apps/common/examples/Peripheral/I2C/src/i2c_sample.c`

17.4.3.1 Test Procedure

1. Remove resistors R6 and R7.
2. Connect the AT24C512 EEPROM with the EVB.
3. Connect each 1,2 kΩ Pull-Up resistor with GPIOA0 and GPIOA1.
   GPIOA0= SDA, GPIOA1=SCL
4. Run I2C example code.

17.4.3.2 Sample Code for Using I2C

1. Initialize I2C.

```
// GPIO Select for I2C working. GPIO1 = SCL, GPIO0= SDA
Board_initialization();
DA16X_CLOCK_SCGATE->Off_DAPB_I2CM = 0;
DA16X_CLOCK_SCGATE->Off_DAPB_APBS = 0;

// Create Handle for I2C Device
I2C = DRV_I2C_CREATE(i2c_0);

// Initialization I2C Device
DRV_I2C_INIT(I2C);
```

2. I2C address.

```
// Device Address for AT24C512
UINT32 addr = 0xa0;
DRV_I2C_IOCTL(I2C, I2C_SET_CHIPADDR, &addr);
```

3. I2C clock.

```
// Set I2C Working Clock. Unit = kHz
DRV_I2C_IOCTL(I2C, I2C_SET_CLOCK, &i2c_clock);
```

4. Write I2C.

```
// Data Random Write  to EEPROM
// Address = 0, Length = 32, Word Address Length = 2
// [Start] - [Device addr. W] -  [1st word addr.] - [2nd word addr.] - [wdata0] ~
   [wdata31] - [Stop]
```

```
i2c_data[0] = AT_I2C_FIRST_WORD_ADDRESS; //Word Address to Write Data. 2 bytes.
                                           refer at24c512 DataSheet
i2c_data[1] = AT_I2C_SECOND_WORD_ADDRESS; //Word Address to Write Data. 2 bytes.
                                           refer at24c512 DataSheet

// Fill Ramp Data
for (int i = 0; i < AT_I2C_DATA_LENGTH; i++) {
    i2c_data[i+AT_I2C_LENGTH_FOR_WORD_ADDRESS] = i;
}

status = DRV_I2C_WRITE(I2C, i2c_data,
// Handle, buffer, length, stop enable, dummy
AT_I2C_DATA_LENGTH + AT_I2C_LENGTH_FOR_WORD_ADDRESS, 1, 0);

if (status != TRUE) {
    PRINTF("ret : 0x%08x\r\n", status);
}
```

5. Read I2C.

```
// Data Random Read from EEPROM
// Address = 0, Length = 32, Word Address Length = 2
// [Start] - [Device addr. W] - [1st word addr.] - [2nd word addr.]  - [Start] -
   [Device addr. R] - [rdata0] ~ [rdata31] - [Stop]

// Word Address to Write Data. 2 bytes. refer at24c512 DataSheet
i2c_data_read[0] = AT_I2C_FIRST_WORD_ADDRESS;

//Word Address to Write Data. 2 bytes. refer at24c512 DataSheet
i2c_data_read[1] = AT_I2C_SECOND_WORD_ADDRESS;

// Handle, buffer, length, address length, dummy
status = DRV_I2C_READ(I2C, i2c_data_read, AT_I2C_DATA_LENGTH,
                      AT_I2C_LENGTH_FOR_WORD_ADDRESS, 0);

if (status != TRUE) {
    PRINTF("ret : 0x%08x\r\n", status);
}

// Check Data
for (int i = 0; i < AT_I2C_DATA_LENGTH; i++) {
    if (i2c_data_read[i] != i2c_data[i + AT_I2C_LENGTH_FOR_WORD_ADDRESS]) {
        PRINTF("%dth data is different W:0x%02x, R:0x%02x\r\n", i,
               i2c_data[i + AT_I2C_LENGTH_FOR_WORD_ADDRESS],
               i2c_data_read[i]);
        status = AT_I2C_ERROR_DATA_CHECK;
    }
}

if (status != AT_I2C_ERROR_DATA_CHECK) {
    PRINTF("***** 32 bytes Data Write and Read Success *****\r\n");
}
```

6. I2C read_nostop.

```
// Data Current Address  Read from EEPROM
// Length = 32, Word Address Length = 0
// [Start] -[Device addr. R] - [rdata0] ~ [rdata31] - [Stop]

// Handle, buffer, length, address length, dummy
```

```
status = DRV_I2C_READ(I2C, i2c_data_read, 4, 0, 0);

if (status != TRUE) {
    PRINTF("ret : 0x%08x\r\n", status);
}
```

17.5 I2S

This section shows how to use the I2S interface.

17.5.1 How to Run

1. In the e[2] studio, import a project for the I2S sample application.
   `~/SDK/apps/common/examples/Peripheral/I2S/projects/da16200`
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
   `The sample application code is written in the following source file:`
   `~/SDK/apps/common/examples/Peripheral/I2S/src/i2s_sample.c`

17.5.2 User Task

The user task of the I2S application is added as shown in the example below and is executed by the system. SAMPLE_I2S should be a unique name to create a task. The port number does not need to be set, because this is a non-network task.

```
~/SDK/apps/common/examples/Peripheral/I2S/src/sample_apps.c
static const app_task_info_t      sample_apps_table[] =
{ I2S_SAMPLE, i2s_sample, 512, (tskIDLE_PRIORITY + 7), FALSE, FALSE,
  UNDEF_PORT,RUN_ALL_MODE },
};
```

17.5.3 Sample Code

1. Create and initialize an I2S handle.

```
HANDLE gi2shandle = NULL;
I2S_HANDLER_TYPE *i2s;
unsigned int mode, data;

DA16X_CLOCK_SCGATE->Off_DAPB_I2S = 0;
DA16X_CLOCK_SCGATE->Off_DAPB_APBS = 0;

gi2shandle = DRV_I2S_CREATE(I2S_0);
i2s = (I2S_HANDLER_TYPE *) gi2shandle;

if (!gi2shandle) {
    vTaskDelete(NULL);
    return;
}

/* Set I2S Output Mode */
if (DRV_I2S_INIT(gi2shandle, mode) == FALSE) {
    vTaskDelete(NULL);
    return;
}
```

2. Set the internal DAC or the external DAC.

```
// GPIO[3] - I2S_LRCK,  GPIO[2] - I2S_SDO
_da16x_io_pinmux(PIN_BMUX, BMUX_I2S);
// GPIO[1] - I2S_MCLK,  GPIO[0] - I2S_BCLK
```

```
_da16x_io_pinmux(PIN_AMUX, AMUX_I2S);

DRV_I2S_SET_CLOCK(gi2shandle, I2S_CLK_SOURCE_INTERNAL, 0);
```

3. Set additional configuration.

```
data = TRUE;
DRV_I2S_IOCTL(i2s, I2S_SET_STEREO, &data); /* Set Stereo Output Mode */

#ifdef I2S_SAMPLE_SET_MODE_RX
data = I2S_RESOLUTION_RX_16B;
#else
data = I2S_RESOLUTION_TX_16B;
#endif

DRV_I2S_IOCTL(i2s, I2S_SET_PCM_RESOLUTION, &data); /* Set 16bit resolution Mode */
```

4. Write and read data.

```
for(int i=0;i<2;i++)
{
#ifdef I2S_SAMPLE_SET_MODE_RX
    rd_len = DRV_I2S_READ(i2s, (unsigned int *)rx_buf[i], 768, 0);
#else
    DRV_I2S_WRITE(i2s, (unsigned int *) sinewave_pattern, 768, 0);
#endif
    xEventGroupWaitBits(i2s_sample_event, 0x1, pdTRUE, pdFALSE, 20);
}
```

17.6 PWM

This section shows how to use the PWM interface.

17.6.1 Introduction

Pulse-Width Modulation (PWM) is a modulation technique used to encode a message into a pulse signal. The blocks are designed to adjust the output pulse duration by means of the CPU bus clock (HCLK).

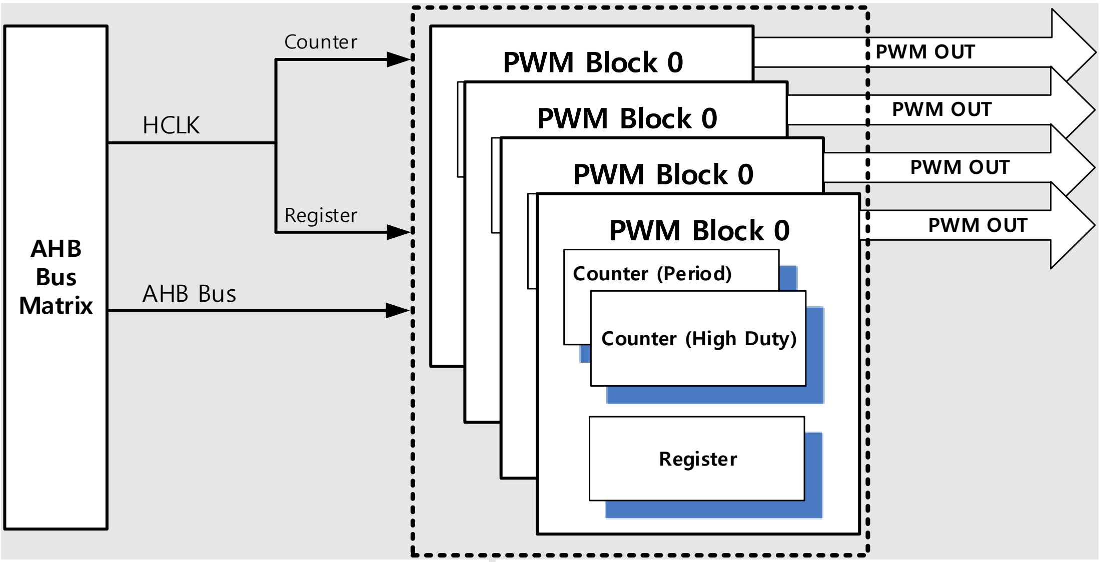


Figure 114. PWM block diagram

Table 67. PWM pin configuration

Pin name	Pin number	I/O	Pin selection	Function name
GPIOx		O	Reg. GPIO_SEL.xMUXx	PWM[3:0] output

For more details, see Ref. [2].

17.6.2 API

Table 68. APIs for PWM interface

Item		Description
HANDLE DRV_PWM_CREATE(UINT32 dev_id)		
Parameter	dev_id	Device number to create handle.
Return		If it succeeds, return the handle for such device. If it fails, return NULL.
Description		Function create handle with the parameter "dev_id" designated.
int DRV_PWM_INITf(HANDLE handler)		
Parameter	handler	Device handle to initialize.
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		Change GPIO multiplex to PWM mode.
int DRV_PWM_START(HANDLE handler, UINT32 period_us, UINT32 hduty_percent, UINT32 dummy)		
Parameter	handler	Device handle to enable PWM device output.
	Period_us	1 cycle period in microsecond.
	Hduty_percent	Output high time in percentage while every 1 cycle.
	dummy	TBD
Return		If it succeeds, return TRUE. If it fails, return FALSE.

Description		Enable PWM block in the DA16200/DA16600 with specified parameters.
		`period = (((period_us * 10) * (clock / 1000000))/10)-1;`
		`// minimum system clock 1 MHz`
		`hduty = (((period + 1) * hduty_percent) / 100)-1;`

int DRV_PWM_STOP(HANDLE handler, UINT32 dummy)		
Parameter	handler	Device handle to stop PWM out.
	cmd	See **pwm.h** in our SDK
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		Disable PWM block in the DA16200/DA16600.
int DRV_PWM_CLOSE(HANDLE handler)		
Parameter	handler	Device handle to close and de-initialize device.
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		Destroy handle

17.6.3 How to Run

1. In the e^2 studio, import a project for the PWM sample application.
```
~/SDK/apps/common/examples/Peripheral/PWM/projects/da16200
```
2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
```
The sample application code is written in the following source file:
~/SDK/apps/common/examples/Peripheral/PWM/src/pwm_sample.c
```

17.6.3.1 Test Procedure

1. Remove resistors R6~R9.
2. Run the PWM example command.
3. Get waveform from P7~P9 in connector J4.
4. Compare the waveform with the PWM setting inside the example code.

17.6.3.2 Sample Code

1. Set GPIO.

```
Board_Init();
DA16X_CLOCK_SCGATE->Off_CAPB_PWM = 0;

gpio = GPIO_CREATE(GPIO_UNIT_A);
GPIO_INIT(gpio);
GPIO_SET_ALT_FUNC(gpio, GPIO_ALT_FUNC_PWM_OUT0, GPIO_ALT_FUNC_GPIO0);
GPIO_SET_ALT_FUNC(gpio, GPIO_ALT_FUNC_PWM_OUT1, GPIO_ALT_FUNC_GPIO1);
GPIO_SET_ALT_FUNC(gpio, GPIO_ALT_FUNC_PWM_OUT2, GPIO_ALT_FUNC_GPIO2);
GPIO_SET_ALT_FUNC(gpio, GPIO_ALT_FUNC_PWM_OUT3, GPIO_ALT_FUNC_GPIO3);
```

2. Initialize PWM.

```
pwm[0] = DRV_PWM_CREATE(pwm_0);
pwm[1] = DRV_PWM_CREATE(pwm_1);
pwm[2] = DRV_PWM_CREATE(pwm_2);
pwm[3] = DRV_PWM_CREATE(pwm_3);

DRV_PWM_INIT(pwm[0]);
DRV_PWM_INIT(pwm[1]);
DRV_PWM_INIT(pwm[2]);
DRV_PWM_INIT(pwm[3]);
```

3. Set start_time.

```
period = 10; // 10 µs
duty_percent = 30; //30%, duration high 3 µs s per 10 µs
DRV_PWM_START(pwm[0], period, duty_percent, PWM_DRV_MODE_US); //PWM Start

period = 20; // 20 µs
duty_percent = 40; //40%, duration high 8 µs per 10 µs
DRV_PWM_START(pwm[1], period, duty_percent, PWM_DRV_MODE_US); //PWM Start

period = 40; // 40 µs
duty_percent = 50; //50%, duration high 20 µs per 10 µs
DRV_PWM_START(pwm[2], period, duty_percent, PWM_DRV_MODE_US); //PWM Start

period = 80; // 80 µs
duty_percent = 80; //80%, duration high 64 µs per 10 µs
DRV_PWM_START(pwm[3], period, duty_percent, PWM_DRV_MODE_US); //PWM Start
```

4. Set start_cycle.

```
// 2400 cycles(=30us @ 80 MHz), cycle = value + 1
cycle = 2400-1;
// 1680 cycles(=21us@80 MHz, 70% Duty High), duty_cycle = value + 1
duty_cycle = 1680-1;
DRV_PWM_START(pwm[0], cycle, duty_cycle, PWM_DRV_MODE_CYC); //PWM Start
```

```
// 2400 cycles(=30us @ 80 MHz), cycle = value + 1
cycle = 2400-1;
// 1680 cycles(=21us@ 80 MHz, 70% Duty High), 70% Duty High), duty_cycle = value + 1
duty_cycle = 1680-1;
DRV_PWM_START(pwm[1], cycle, duty_cycle, PWM_DRV_MODE_CYC); //PWM Start

// 2400 cycles(=30us @ 80 MHz), cycle = value + 1
cycle = 2400-1;
// 1680 cycles(=21us@ 80 MHz, 70% Duty High), 70% Duty High), duty_cycle = value + 1
duty_cycle = 1680-1;
DRV_PWM_START(pwm[2], cycle, duty_cycle, PWM_DRV_MODE_CYC); //PWM Start

// 2400 cycles(=30us @ 80 MHz), cycle = value + 1
cycle = 2400-1;
// 1680 cycles(=21us@ 80 MHz, 70% Duty High), 70% Duty High), duty_cycle = value + 1
duty_cycle = 1680-1;
DRV_PWM_START(pwm[3], cycle, duty_cycle, PWM_DRV_MODE_CYC); //PWM Start
```

5. Stop PWM.

```
DRV_PWM_STOP(pwm[0], 0);
DRV_PWM_STOP(pwm[1], 0);
DRV_PWM_STOP(pwm[2], 0);
DRV_PWM_STOP(pwm[3], 0);
```

17.7 ADC

This section shows how to use the ADC interface.

17.7.1 Introduction

The DA16200/DA16600 has Analog-to-Digital Converters (ADC): a four-channel single-end ADC of 12-bit resolution. Analog input is measured by means of 4 pins from GPIO0 to GPIO3, and the pin selection is changed through the register setting. See Figure 115 and Table 69.

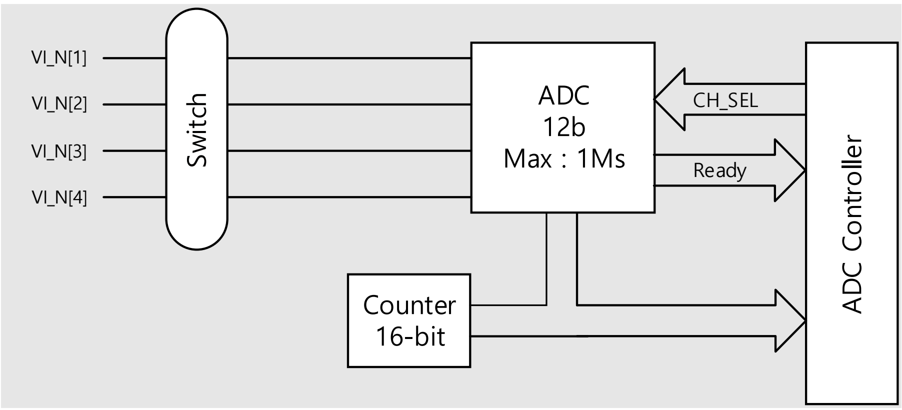


Figure 115. ADC control block diagram

Table 69. AUX ADC pin configuration

Pin name	Pin number		I/O	Function name
	QFN	fcCSP		
GPIOA3	36	D4	A	Analog signal
GPIOA2	37	B2	A	Analog signal
GPIOA1	38	C3	A	Analog signal
GPIOA0	39	A3	A	Analog signal

For more details, see Ref. [2].

17.7.2 API

Table 70. APIs for ADC interface

Item		Description
HANDLE DRV_ADC_CREATE(UINT32 dev_id)		
Parameter	dev_id	Device number to create a handle.
Return		If it succeeds, return the handle for such device. If it fails, return NULL.
Description		Function create handle with the parameter dev_id designated.
int DRV_ADC_INIT(HANDLE handler, unsigned int use_timestamp)		
Parameter	handler	Device handle to initialize.
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		ADC Initialization command.
Int DRV_ADC_IOCTL(HANDLE handler, UINT32 cmd, VOID *data)		
Parameter	handler	N/A
	cmd	N/A
	data	N/A
Return		N/A
Description		ADC IOCTL command
int DRV_ADC_START(HANDLE handler, UINT32 divider12, UINT32 dummy)		
Parameter	handler	Device handle to start.
	divider12	Fs = sys_clk/15/(div12 +1)
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		ADC starts command.
int DRV_ADC_STOP(HANDLE handler, UINT32 dummy)		
Parameter	handler	Device handle to stop.
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		ADC stop command.
Int DRV_ADC_CLOSE(HANDLE handler)		
Parameter	handler	Device handle to close.
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		ADC driver close.
int DRV_ADC_READ(HANDLE handler, UINT32 channel, UINT32 *data, UINT32 dummy)		
Parameter	handler	Device handle to read.
	channel	Channel number to read instant ADC value.
	*data	Buffer to read.
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		ADC read command.
int DRV_ADC_READ_DMA(HANDLE handler, UINT32 channel, UINT16 *p_data, UINT32 p_dlen, UINT32 dummy)		
Parameter	handler	Device handle to read with specified length.
	channel	Channel number to read.

Item		Description
	*p_data	Buffer block to read.
	p_dlen	Number of samples to read with DMA, not buffer length.
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		ADC read commands through DMA.
int DRV_ADC_ENABLE_CHANNEL(HANDLE handler, UINT32 channel, unsigned int sel_adc, UINT32 dummy)		
Parameter	handler	Device handle
	channel	Channel number to set ADC devices.
	sel_adc	12: SMI 12B ADC, 0: disable
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		ADC channel enables command.
int DRV_ADC_SET_INTERRUPT(HANDLE handler, UINT32 channel, UINT32 enable, UINT32 type, UINT32 dummy)		
Parameter	handler	Device handle
	channel	Channel number to set interrupt.
	enable	1: enable interrupt, 0: disable interrupt
	type	ADC_INTERRUPT_FIFO_HALF (0) ADC_INTERRUPT_FIFO_FULL (1) ADC_INTERRUPT_THD_OVER (2) ADC_INTERRUPT_THD_UNDER (3) ADC_INTERRUPT_THD_DIFF (4) ADC_INTERRUPT_ALL (0xf)
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		ADC interrupt set command.
int DRV_ADC_SET_THD_VALUE(HANDLE handler, UINT32 type, UINT32 enable, UINT32 thd, UINT32 dummy);		
Parameter	handler	Device handle
	type	ADC_THRESHOLD_TYPE_12B_OVER (0) ADC_THRESHOLD_TYPE_12B_UNDER (2) ADC_THRESHOLD_TYPE_12B_DIFF (4)
	thd	Interrupt threshold. 0 ~ 65535 range. Upper 12 bits of 16-bit data are valid values.
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		ADC interrupt threshold set command.
int DRV_ADC_WAIT_INTERRUPT(HANDLE handler, UNSIGNED *mask_evt);v		
Parameter	handler	Device handle

Item		Description
	*mask_evt	Mask for waiting interrupt bit[19] : Interrupt status for Threshold Difference of CHANNEL 3 bit[18] : Interrupt status for Threshold Difference of CHANNEL 2 bit[17] : Interrupt status for Threshold Difference of CHANNEL 1 bit[16] : Interrupt status for Threshold Difference of CHANNEL 0 bit[15] : Interrupt status for Threshold Under level of CHANNEL 3 bit[14] : Interrupt status for Threshold Under level of CHANNEL 2 bit[13] : Interrupt status for Threshold Under level of CHANNEL 1 bit[12] : Interrupt status for Threshold Under level of CHANNEL 0 bit[11] : Interrupt status for Threshold Over level of CHANNEL 3 bit[10] : Interrupt status for Threshold Over level of CHANNEL 2 bit[9] : Interrupt status for Threshold Over level of CHANNEL 1 bit[8] : Interrupt status for Threshold Over level of CHANNEL 0 bit[7] : Interrupt status for full level of CHANNEL 3 bit[6] : Interrupt status for full level of CHANNEL 2 bit[5] : Interrupt status for full level of CHANNEL 1 bit[4] : Interrupt status for full level of CHANNEL 0 bit[3] : Interrupt status for half level of CHANNEL 3 bit[2] : Interrupt status for half level of CHANNEL 2 bit[1] : Interrupt status for half level of CHANNEL 1 bit[0] : Interrupt status for half level of CHANNEL 0
Return		If it succeeds, return masked interrupt. If it fails, return FALSE.
Description		Wait ADC interrupt

17.7.3 Interrupt Description

ADC_INTERRUPT_FIFO_HALF: this interrupt occurs when the FIFO Level is 4 or higher.

ADC_INTERRUPT_FIFO_FULL: this interrupt occurs when FIFO Level is 8.

ADC_INTERRUPT_THD_OVER: this interrupt occurs when the current input value to the ADC device is greater than the value set in the "ADC_THRESHOLD_TYPE_12B_OVER" type.

ADC_INTERRUPT_THD_UNDER: this interrupt occurs when the current input value to the ADC device is smaller than the value set in the " ADC_THRESHOLD_TYPE_12B_UNDER " type.

ADC_INTERRUPT_THD_DIFF: this interrupt occurs when the difference between the current input value to the ADC device and the previous input value is greater than the value set in "ADC_INTERRUPT_THD_DIFF" type.

17.7.4 How to Run

1. In the e^2 studio, import a project for the ADC sample application.

   `~/SDK/apps/common/examples/Peripheral/ADC/projects/da16200`

2. There are three types of preprocessor statements defined in the ADC example code.
   - ADC_SAMPLE_READ
     - Read and print ADC input values
   - ADC_SAMPLE_INTERRUPT
     - Set the interrupt to occur at 0.7 V or less and verify that the setting works
   - ADC_SAMPLE_DPM
     - Set the ADC value of 0.15 V or more before entering Sleep mode 2, and it wakes up from Sleep mode 2 when ADC input is 0.15 V or more.
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. The sample application code is written in the following source file:

~/SDK/apps/common/examples/Peripheral/ADC/src/adc_sample.c

17.7.5 Sample Code – SAMPLE_READ

17.7.5.1 Test Procedure

1. Provide 0~1.3 V to P7 ~ P9, in connector J4.
2. Run the ADC example and read the ADC value.
3. Compare the value with the voltage supplied.

17.7.5.2 Sample Code for Reading ADC

1. Initialize ADC.

```
PRINTF("ADC_SAMPLE\n");
DA16X_CLOCK_SCGATE->Off_DAPB_AuxA = 0;
DA16X_CLOCK_SCGATE->Off_DAPB_APBS = 0;

// Set PAD Mux. GPIO_0 (ADC_CH0), GPIO_1(ADC_CH1)
_da16x_io_pinmux(PIN_AMUX, AMUX_AD12);

// Create Handle
hadc = DRV_ADC_CREATE(DA16200_ADC_DEVICE_ID);

// Initialization
DRV_ADC_INIT(hadc, DA16x_ADC_NO_TIMESTAMP);
```

2. Start ADC.

```
// Start. Set Sampling Frequency. 12B ADC Set to 200 kHz
// Clock = 1 MHZ / (value + 1)
// Ex) If Value = 4, Clock = 1 MHz / (4+1) = 200 kHz
DRV_ADC_START(hadc, DA16x_ADC_DIVIDER_12, 0);
```

3. Enable ADC.

```
// Set ADC_0 to 12- Bit ADC, ADC_1 to 12-Bit ADC
DRV_ADC_ENABLE_CHANNEL(hadc, DA16200_ADC_CH_0, DA16x_ADC_SEL_ADC_12, 0);
DRV_ADC_ENABLE_CHANNEL(hadc, DA16200_ADC_CH_1, DA16x_ADC_SEL_ADC_12, 0);
```

4. Read ADC Using DMA.

```
// Read 16ea ADC_0 Value. 12B ADC, Bit [15:4] is valid adc_data, [3:0] is zero
DRV_ADC_READ_DMA(hadc, DA16200_ADC_CH_0, data0, DA16x_ADC_NUM_READ * 2,
                   DA16x_ADC_TIMEOUT_DMA, 0);

// Read 16ea ADC_1 Value
DRV_ADC_READ_DMA(hadc, DA16200_ADC_CH_1, data1, DA16x_ADC_NUM_READ * 2,
                   DA16x_ADC_TIMEOUT_DMA, 0);
```

5. Read ADC.

```
// Read Current ADC_0 Value. Caution!! When read current adc value consequently,
// need delay at each read function bigger than Sampling Frequency
DRV_ADC_READ(hadc, DA16200_ADC_CH_0, &data, 0);
```

6. Close ADC.

```
// Close ADC
DRV_ADC_CLOSE(hadc);
0.
```

17.7.6 Sample Code – ADC_SAMPLE_INTERRUPT

17.7.6.1 Test Procedure

1. Provide 1.3 voltage to P7 ~ P9, in connector J4.

2. Run the ADC example.
3. Change the power supply to the ADC to 0.7 V or lower to see if an interrupt occurs.

17.7.6.2 Sample Code for ADC Interrupt

1. Initialize ADC.

```
HANDLE hadc;
int status, int_handling_mode;
unsigned int data, type, thd;

PRINTF("ADC_SAMPLE\n");
DA16X_CLOCK_SCGATE->Off_DAPB_AuxA = 0;
DA16X_CLOCK_SCGATE->Off_DAPB_APBS = 0;

// Set PAD Mux. GPIO_0 (ADC_CH0), GPIO_1(ADC_CH1)
_da16x_io_pinmux(PIN_AMUX, AMUX_AD12);

// Create Handle
hadc = DRV_ADC_CREATE(DA16200_ADC_DEVICE_ID);

// Initialization
status = DRV_ADC_INIT(hadc, DA16x_ADC_NO_TIMESTAMP);
PRINTF("ADC-INIT: %d\n", status);
```

2. Start ADC.

```
// Start. Set Sampling Frequency. 12B ADC Set to 200 kHz
// Clock = 1 MHZ / (value + 1)
// Ex) If Value = 4, Clock = 1 MHz / (4+1) = 200 kHz
status = DRV_ADC_START(hadc, DA16x_ADC_DIVIDER_12, 0);
RINTF("ADC start: %d\n", status);

// Set ADC_0 to 12-Bit ADC
status = DRV_ADC_ENABLE_CHANNEL(hadc, DA16200_ADC_CH_0, DA16x_ADC_SEL_ADC_12, 0);
PRINTF("ADC enable: %d\n", status);

//Set Data type offset binary, 0 : 2's complement ,  1 : offset binary
type = 1;
DRV_ADC_IOCTL(hadc, ADC_SET_DATA_MODE, &type);
```

3. Read ADC.

```
// Read Current ADC_0 Value. Caution!! When read current adc value consequently, need delay at each
read function bigger than Sampling Frequency
DRV_ADC_READ(hadc, DA16200_ADC_CH_0, &data, 0);
PRINTF("Current ADC Value = 0x%x\r\n", GET_VALID_ADC_VALUE(data));
```

4. Set ADC Interrupt.

```
//set threshold
//value 0x800(=0.7V/1.4V * 4095), max=4095(1.4V), min=0(0V), increase linearly) means about 0.7Volt in
EVK environment.
thd = 0x800;
//thd value must be shifted 4 bits right.
status = DRV_ADC_SET_THD_VALUE(hadc, ADC_THRESHOLD_TYPE_12B_UNDER, CONVERT_TO_THD_VALUE(thd), 0);
PRINTF("ADC-set threshold: %d\n", status);

// set Interrupt
status = DRV_ADC_SET_INTERRUPT(hadc, DA16200_ADC_CH_0, TRUE, ADC_INTERRUPT_THD_UNDER, 0);
PRINTF("ADC-set interrupt: %d\n", status);

//set Interrupt Handling Mode
//ADC_INTERRUPT_MODE_EVENT mode : in interrupt handler, call set_event function.
//ADC_INTERRUPT_MODE_MASK mode  : in interrupt handler, disable interrupt.
int_handling_mode = ADC_INTERRUPT_MODE_EVENT | ADC_INTERRUPT_MODE_MASK;
DRV_ADC_IOCTL(hadc, ADC_SET_INTERRUPT_MODE, (void *)(&int_handling_mode));
```

5. Wait ADC Interrupt.

```
//Wait Interrupt
UNSIGNED interrupt_status ;
DRV_ADC_WAIT_INTERRUPT(hadc, &interrupt_status);

vTaskDelay(5);
```

6. Print and Disable Interrupt.

```
// disable Interrupt
status = DRV_ADC_SET_INTERRUPT(hadc, DA16200_ADC_CH_0, FALSE, ADC_INTERRUPT_THD_UNDER, 0);
PRINTF("ADC-reset interrupt: %d\n", status);

// Read Current ADC_0 Value. Caution!! When read current adc value consequently, need delay at each
read function bigger than Sampling Frequency
DRV_ADC_READ(hadc, DA16200_ADC_CH_0, &data, 0);
PRINTF("Interrupt Occured with Value = 0x%x\r\n", GET_VALID_ADC_VALUE(data));

vTaskDelete(NULL);
```

17.7.7 Sample Code – ADC_SAMPLE_DPM

17.7.7.1 Test Procedure

1. Provide 0 V to P7 ~ P9, in connector J4.
2. Run the ADC example.
3. After the DA16200 enters Sleep mode, verify that it wakes up by changing the voltage supplied to the ADC to at least 0.15 V.

17.7.7.2 Sample Code for Wake Up DPM

1. Initialize ADC.

```
HANDLE hadc;
int status, int_handling_mode;
unsigned int data, type, val;
unsigned long long  wakeup_time;
UINT32 wakeup_src;

DA16X_CLOCK_SCGATE->Off_DAPB_AuxA = 0;
DA16X_CLOCK_SCGATE->Off_DAPB_APBS = 0;

// Set PAD Mux. GPIO_0 (ADC_CH0), GPIO_1(ADC_CH1)
_da16x_io_pinmux(PIN_AMUX, AMUX_AD12);

// Create Handle
```

```
hadc = DRV_ADC_CREATE(DA16200_ADC_DEVICE_ID);

// Initialization
status = DRV_ADC_INIT(hadc, DA16200_ADC_NO_TIMESTAMP);
//PRINTF("ADC-INIT: %d\n",status);
```

2. Start ADC.

```
// Start. Set Sampling Frequency. 12B ADC Set to 200 kHz
// Clock = 1 MHZ / (value + 1)
// Ex) If Value = 4, Clock = 1 MHz / (4+1) = 200 kHz
status = DRV_ADC_START(hadc, DA16200_ADC_DIVIDER_12, 0);
//PRINTF("ADC start: %d\n",status);

// Set ADC_0 to 12-Bit ADC
status = DRV_ADC_ENABLE_CHANNEL(hadc, DA16200_ADC_CH_0,
DA16200_ADC_SEL_ADC_12, 0);
//PRINTF("ADC enable: %d\n",status);

//Set Data type offset binary, 0 : 2's complement ,  1 : offset binary
type = 1;
DRV_ADC_IOCTL(hadc, ADC_SET_DATA_MODE, &type);
```

3. Read ADC.

```
//Read Current ADC_0 Value.
DRV_ADC_READ(hadc, DA16200_ADC_CH_0, &data, 0);
PRINTF("Current ADC Value = 0x%x\r\n", data & 0xffff);
```

4. Set ADC Interrupt.

```
//set threshold
//value 0x8000 means about 0.7Volt in EVK environment.
status = DRV_ADC_SET_THD_VALUE(hadc, ADC_THRESHOLD_TYPE_12B_OVER, 0x1B60,
               0);
PRINTF("ADC-set threshold: %d\n", status);

//set Interrupt
status = DRV_ADC_SET_INTERRUPT(hadc, DA16200_ADC_CH_0, TRUE,
ADC_INTERRUPT_THD_OVER, 0);
PRINTF("ADC-set interrupt: %d\n", status);

//set Interrupt Mode
int_handling_mode = ADC_INTERRUPT_MODE_EVENT | ADC_INTERRUPT_MODE_MASK;
DRV_ADC_IOCTL(hadc, ADC_SET_INTERRUPT_MODE, (void*) (&int_handling_mode));

//Set ADC for Sleep mode 2 //
DRV_ADC_SET_THRESHOLD(hadc, DA16200_ADC_CH_0, 0x1B6,
ADC_RTC_THRESHOLD_TYPE_OVER);
val = 1;
DRV_ADC_IOCTL(hadc, ADC_SET_RTC_CYCLE_BEFORE_ON, &val);
val = 1;
DRV_ADC_IOCTL(hadc, ADC_SET_RTC_CYCLE_BEFORE_CAPTURE, &val);
val = 0;
DRV_ADC_IOCTL(hadc, ADC_SET_CAPTURE_STEP, &val);
DRV_ADC_SET_SLEEP_MODE(hadc, 1, 0xf, 1);
DRV_ADC_SET_RTC_ENABLE_CHANNEL(hadc, DA16200_ADC_CH_0, 1);

drv_adc_sensor_out_enable(hadc);
vTaskDelay(50);
```

5. Enter Sleep mode 2.

```
wakeup_src = da16x_boot_get_wakeupmode();
PRINTF("\nWakeup source is 0x%x\n", wakeup_src);

wakeup_time = 10000000000 * 1000000;
start_dpm_sleep_mode_2(wakeup_time, TRUE);
PRINTF("Sample: Go to Sleep mode 2 ... \n");
```

17.8 SPI

This section shows how the SPI loopback operation works.

17.8.1 Introduction

17.8.1.1 SPI Master

The SPI master communicates in full duplex mode that uses master-slave architecture with a single master. The master device originates the frame to be read or written. Multiple slave-devices are supported with the selection of individual chip select (CS) lines.

Table 71 shows the pin definition of the SPI master interface. To use as an SPI master, the CSB signal can be used with any of the GPIO pins. CSB [3:2] can be selected from the GPIO special function. This is done through register settings in the GPIO.

Table 71. SPI master pin configuration

Pin name	Pin number		I/O	Function name
	QFN	fcCSP		
GPIOx			O	E_SPI_CSB[3:1]
GPIOA6	32	E3	O	E_SPI_CSB[0]
GPIOA7	31	E1	O	E_SPI_CLK
GPIOA8	30	G3	I/O	E_SPI_MOSI or E_SPI_D[0]
GPIOA9	29	H2	I/O	E_SPI_MISO or E_SPI_D[1]
GPIOA10	28	F2	I/O	E_SPI_D[2]
GPIOA11	27	G1	I/O	E_SPI_D[3]

17.8.1.2 SPI Slave

The SPI slave interface enables support to control the DA16200/DA16600 from an external host. The range of the SPI clock speed is the same as that of the internal bus clock speed. The SPI slave supports both burst mode and non-burst mode. In the burst mode, SPI_CSB remains active from the start to the end of communication. In the non-burst mode, SPI_CLK remains active at every 8-bit.

The communication protocols of the SPI slave interface use either 4-byte or 8-byte control signals. Between the two available communication protocols, the CPU chooses one before initiating the control.

Table 72. SPI slave pin configuration

Pin name	Pin number		I/O	Function name
	QFN	fcCSP		
GPIOA2	37	B2	I	SPI_CSB
GPIOA6	32	E3	I	
GPIOA3	36	D4	I	SPI_CLK
GPIOA7	31	E1	I	
GPIOA1	38	C3	I	SPI_MOSI
GPIOA9	29	H2	I	
GPIOA11	27	G1	I	

Pin name	Pin number		I/O	Function name
	QFN	fcCSP		
GPIOA0	39	A3	O	SPI_MISO
GPIOA8	30	G3	O	
GPIOA10	28	F2	O	

17.8.2 API

Table 73. APIs for SPI master interface

Item		Description
HANDLE SPI_CREATE(UINT32 dev_id)		
Parameter	dev_id	Instance Number of SPI (UINT32).
Return		Handler of SPI Driver (HANDLE).
Description		Returns the SPI Handler that is defined in "spi.h" file. ▪ create the GPIO handler for chip selection.
int SPI_INIT (HANDLE handler)		
Parameter	Handler	SPI Driver (HANDLE)
Return		TRUE/FALSE (int)
Description		Initializes the SPI Handler to set up GPIO and activate the ISR. ▪ create the MUTEX for support to control multi-slaves.
int SPI_IOCTL(HANDLE handler, UINT32 cmd, VOID *data)		
Parameter	Handler	SPI Driver (HANDLE)
	Cmd	IOCTL command
	data	IOCTL parameters
Return		TRUE/FALSE (int)
int SPI_WRITE(HANDLE handler, void *pdata, UINT32 dlen)		
Parameter	Handler	SPI Driver (HANDLE)
Return		zero – false, non-zero – data length (int)
Description		SPI write operation. ▪ pdata: TX data buffer ▪ dlen: byte length
int SPI_WRITE_READ(HANDLE handler, void *snddata, UINT32 sndlen, void *rcvdata, UINT32 rcvlen)		
Parameter	Handler	SPI Driver (HANDLE)
Return		zero – false, non-zero – data length (int)
Description		SPI write and read operation (write before read). This function runs in concatenation mode internally. ▪ snddata: TX data buffer ▪ sndlen: byte length ▪ rcvdata: TX data buffer rcvlen: byte length
Int SPI_TRANSMIT(HANDLE handler, VOID *snddata, UINT32 sndlen, VOID *rcvdata, UINT32 rcvlen)		
Parameter	Handler	SPI Driver (HANDLE)
Return		zero - false, non-zero - data length (int)

Item		Description
Description		Basic operation running once in SPI burst mode (send before receiving). This function does not support changing bus mode automatically. ▪ snddata: TX data buffer ▪ sndlen: byte length ▪ rcvdata: TX data buffer rcvlen: byte length
Int SPI_CLOSE(HANDLE handler)		
Parameter	Handler	SPI Driver (HANDLE)
Return		TRUE/FALSE (int)
Description		Release the SPI handler

Table 74. APIs for SPI slave interface

void host_spi_slave_init(void)
Change Slave I/F to SPI protocol. Enable clock to SPI slave device and GPIO Interrupt Set.
void host_i2c_slave_init(void)
Change Slave I/F to I2C protocol. Enable clock to I2C slave device and GPIO Interrupt Set.

17.8.3 How to Run

1. In the e^2 studio, import a project for the SPI sample application.

   `~/SDK/apps/common/examples/Peripheral/SPI/projects/da16200`

2. Connect the SPI master pins and SPI slave pins.
   - GPIOA[0] (SPI_MISO) - GPIOA[9] (E_SPI_DIO1)
   - GPIOA[1] (SPI_MOSI) - GPIOA[8] (E_SPI_DIO0)
   - GPIOA[2] (SPI_CSB) - GPIOA[6] (E_SPI_CSB)
   - GPIOA[3] (SPI_CLK) - GPIOA[7] (E_SPI_CLK)
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. The SPI loopback communication works as shown in Figure 116.

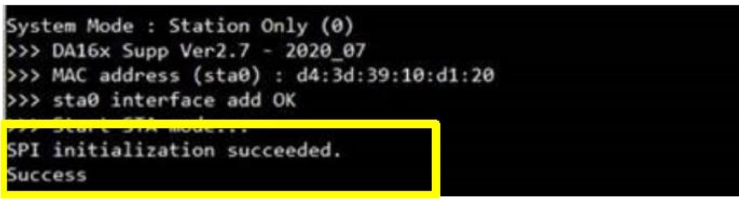


Figure 116. SPI loopback communication

17.8.4 Sample Code

1. Create an SPI handle and configure the interface.

```
spi = SPI_CREATE(SPI_UNIT_0);
if(spi == NULL) {
    PRINTF("[%s]failed to create instance\n", __func__);
    return;
}

_sys_clock_read( ioctldata, sizeof(UINT32));
SPI_IOCTL(spi, SPI_SET_CORECLOCK, ioctldata);

/* set SPI speed */
ioctldata[0] = SMC_SPI_SPEED * MHz;
```

```
SPI_IOCTL(spi, SPI_SET_SPEED, ioctldata);

/* set SPI polarity */
ioctldata[0] = SMC_SPI_POLARITY;
SPI_IOCTL(spi, SPI_SET_FORMAT, ioctldata);

/* set SPI DMA config */
ioctldata[0] = SPI_DMA_MP0_BST(8)
              | SPI_DMA_MP0_IDLE(1)
              | SPI_DMA_MP0_HSIZE(XHSIZE_DWORD)
              | SPI_DMA_MP0_AI(SPI_ADDR_INCR);
SPI_IOCTL(spi, SPI_SET_DMA_CFG, ioctldata);
SPI_IOCTL(spi, SPI_SET_DMAMODE, NULL);

/* set SPI chip select, io operation type */
ioctldata[0] = SMC_SPI_CS;
ioctldata[1] = SMC_IO_OPERTATION_TYPE;
SPI_IOCTL( spi, SPI_SET_WIRE, (VOID *)ioctldata);

/* set SPI delay index */
ioctldata[0] = SPI_DELAY_INDEX_LOW;
SPI_IOCTL(spi, SPI_SET_DELAY_INDEX, ioctldata);

/* SPI initialization */
status = SPI_INIT(spi);
```

2. Set pin multiplexing as SPI master and SPI slave.

```
/* pinmux config for SPI Slave - GPIOA[3:0] */
_da16x_io_pinmux(PIN_AMUX, AMUX_SPIs);
_da16x_io_pinmux(PIN_BMUX, BMUX_SPIs);

/* pinmux config for SPI Host  - GPIOA[9:6] */
_da16x_io_pinmux(PIN_DMUX, DMUX_SPIm);
_da16x_io_pinmux(PIN_EMUX, EMUX_SPIm);
```

3. Write data.

```
/* generate host interface protocol header */
_buf[0] = (addr >> 8) & 0xff;
_buf[1] = (addr >> 0) & 0xff;
_buf[2] = (HPC_WRITE_CMD & 0xff)
       | (HPC_COMMON_ADDR_MODE << 5)
       | (HPC_REF_LEN<<4)|((length>>8)&0xf);
_buf[3] = (length)&0xff;

/* copy data to buf */
memcpy(&(_buf[4]), data, length);

/* Bus Lock : CSEL0 */
ioctldata[0] = TRUE;
ioctldata[1] = portMAX_DELAY;
ioctldata[2] = SPI_CSEL_0;
SPI_IOCTL(spi, SPI_SET_LOCK, (VOID *)ioctldata);

status = SPI_WRITE(spi, _buf, (HPC_HEADER_LEN + length));

/* Bus Unlock */
ioctldata[0] = FALSE;
ioctldata[1] = portMAX_DELAY;
```

```
ioctldata[2] = SPI_CSEL_0;
SPI_IOCTL(spi, SPI_SET_LOCK, (VOID *)ioctldata);
```

4. Read data.

```
_buf[0] = (addr >> 8) & 0xff;
_buf[1] = (addr >> 0) & 0xff;
_buf[2] = HPC_READ_CMD
        | (HPC_COMMON_ADDR_MODE << 5)
        | (HPC_REF_LEN<<4)|((len>>8)&0xf);
_buf[3] = (len)&0xff;

/* Bus Lock : CSEL0 */
ioctldata[0] = TRUE;
ioctldata[1] = portMAX_DELAY;
ioctldata[2] = SPI_CSEL_0;
SPI_IOCTL(spi, SPI_SET_LOCK, (VOID *)ioctldata);

status = SPI_WRITE_READ(spi, _buf, 4, rx_data, len);

/* Bus Unlock */
ioctldata[0] = FALSE;
ioctldata[1] = portMAX_DELAY;
ioctldata[2] = SPI_CSEL_0;
SPI_IOCTL(spi, SPI_SET_LOCK, (VOID *)ioctldata);
```

17.9 SDIO

The DA16200 can be accessed with the SDIO interface. If the user wants to test it, then another host system is needed.

17.9.1 Introduction

17.9.1.1 SDIO Master

Secure Digital Input Output (SDIO) is a full/high speed card suitable for memory card and I/O card applications with low power consumption. The full/high speed card supports SPI, 1-bit SD and 4-bit SD transfer modes at the full clock range of 0~50 MHz. To be compatible with the serviceable SDIO clock, the internal BUS clock should be set to a minimum of 50 MHz. The CIS and CSA areas are inside the internal memory, and the SDIO registers (CCCR and FBR) are programmed by the SD host. For more details, see Ref. [2].

17.9.1.2 SDIO Slave

The GPIO4 and GPIO5 pins are set to SDIO CMD and CLK by default. If SDIO initialization is done and SDIO communication is enabled, then the SDIO data pin setting is done automatically. In other words, when SDIO communication is detected, the pin used as the SDIO data among the GPIO pins is automatically activated in the SDIO use mode. However, the auto setting function is not supported for the F_xx pin used as the flash function.

17.9.2 API

Table 75. APIs for SDIO master interface

Item		Description
HANDLE EMMC_CREATE(void);		
Parameter	None	-
Return		If it succeeds, return the handle for such device. If it fails, return NULL.
Description		Create EMMC handle.
int EMMC_INIT(HANDLE handler)		
Parameter	handler	Device handle

Item		Description
Return		If it succeeds, return ERR_NONE. If it fails, return ERR_MMC_INIT.
Description		Initialize the SD/eMMC or SDIO card.
int EMMC_CLOSE(HANDLE handler)		
Parameter	handler	Device handle
Return		If it succeeds, return ERR_NONE.
int SDIO_ENABLE_FUNC(HANDLE handler, UINT32 func_num)		
Parameter	handler	Device handle
	func_num	Function number to enable.
Return		If it succeeds, return ERR_NONE.
int SDIO_DISABLE_FUNC(HANDLE handler, UINT32 func_num)		
Parameter	handler	Device handle
	func_num	Function number to disable.
Return		If it succeeds, return ERR_NONE.
int SDIO_SET_BLOCK_SIZE(HANDLE handler, UINT32 func_num, UINT32 blk_size)		
Parameter	handler	Device handle
	func_num	Function number
	blk_size	Block size
Return		If it succeeds, return ERR_NONE.
int SDIO_READ_BYTE(HANDLE handler, UINT32 func_num, UINT32 addr, UINT8 *data)		
Parameter	handler	Device handle
	func_num	Function number
	addr	Address in the function
	data	Data pointer
Return		If it succeeds, return ERR_NONE. And byte data is stored in data.
int SDIO_WRITE_BYTE(HANDLE handler, UINT32 func_num, UINT32 addr, UINT8 *data)		
Parameter	handler	Device handle
	func_num	Function number
	addr	Address in the function
	data	Data pointer
Return		If it succeeds, return ERR_NONE.
int SDIO_READ_BURST(HANDLE handler, UINT32 func_num, UINT32 addr, UINT32 incr_addr, UINT8 *data, UINT32 count, UINT32 blksz)		
Parameter	handler	Device handle
	func_num	Function number
	addr	Function address
	Incr_addr	Increase address option (1: address increase, 0: address fix)
	data	Data pointer
	count	Count of blocks
	blksz	Block size
Return		If it succeeds, return ERR_NONE. If it fails, Error Code return, see also EMMC.h.

Item		Description
int SDIO_WRITE_BURST(HANDLE handler, UINT32 func_num, UINT32 addr, UINT32 incr_addr, UINT8 *data, UINT32 count, UINT32 blksz)		
Parameter	handler	Device handle
	func_num	Function number
	addr	Function address
	Incr_addr	Increase address option (1: address increase, 0: address fix)
	data	Data pointer
	count	Count of blocks
	blksz	Block size
Return		If it succeeds, return ERR_NONE.

Table 76. SDIO slave pin configuration

Pin name	Pin number		I/O	Function name
	QFN	fcCSP		
GPIOA4	34	F4	I/O	SDIO_CMD
F_CSN	18	J5	I/O	
GPIOA5	33	D2	I	SDIO_CLK
F_CLK	19	K4	I	
GPIOA9	29	H2	I/O	SDIO_D0
F_IO0	14	K8	I/O	
GPIOA8	30	G3	I/O	SDIO_D1
F_IO1	15	L7	I/O	
GPIOA7	31	E1	I/O	SDIO_D2
F_IO2	16	J7	I/O	
GPIOA6	32	E3	I/O	SDIO_D3
F_IO3	17	K6	I/O	

For more details, see Ref. [2].

17.9.3 How to Run

1. In the e² studio, import a project for the SDIO sample application.

   `~/SDK/apps/common/examples/Peripheral/SDIO/projects/da16200`

2. The sample application code is written in the following source file:

   `~/SDK/apps/common/examples/Peripheral/SDIO/src/sdio_sample.c`

   - GPIOA[9:4] needs to connect to the HOST system
   - GPIOA[9] - SDIO_D0, GPIOA[8] - SDIO_D1
   - GPIOA[7] - SDIO_D2, GPIOA[6] - SDIO_D3
   - GPIOA[5] - SDIO_CLK, GPIOA[4] - SDIO_CMD

3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. The sample runs as soon as the boot up is completed.

```
[/DA16200]  # sdio_slave start
 Now the sdio host can access the DA16200
[/DA16200] #
```

Now the DA16200 is ready to receive an SDIO command.

17.9.4 Sample Code

In the DA16200, the loopback test between SD host and sdio_slave is not supported. Instead, in the sample code provided, SDIO is just waiting for a request from the host after initialization.

```
/*
 * SDIO Slave
 */
// GPIO[9] - SDIO_D0,  GPIO[8] - SDIO_D1
_da16x_io_pinmux(PIN_EMUX, EMUX_SDs);
// GPIO[5] - SDIO_CLK,  GPIO[4] - SDIO_CMD
_da16x_io_pinmux(PIN_CMUX, CMUX_SDs);
// GPIO[7] - SDIO_D2,  GPIO[6] - SDIO_D3
_da16x_io_pinmux(PIN_DMUX, DMUX_SDs);

// clock enable sdio_slave
DA16X_CLOCK_SCGATE->Off_SSI_M3X1 = 0;
DA16X_CLOCK_SCGATE->Off_SSI_SDIO = 0;

SDIO_SLAVE_INIT();
/* now the sdio host can access the DA16200 */
Printf("now the sdio host can access the DA16200\r\n");
```

17.10 SD/eMMC

This section shows how to use the SD/eMMC interface.

17.10.1 Introduction

The SD/eMMC host interface of the DA16200/DA16600 provides access to SD or eMMC cards. The SD/eMMC host interface supports a 4-bit data bus with a maximum clock rate of 48 MHz giving a maximum data rate of 24 MB/s (192 Mbps). The SD/eMMC pin mux condition is defined in Table 77.

Table 77. SD/eMMC master pin configuration

Pin name	Pin number		I/O	Function name
	QFN	fcCSP		
GPIOA4	34	F4	I/O	SD/eMMC_CMD
GPIOA5	33	D2	O	SD/eMMC_CLK
GPIOA9	29	H2	I/O	SD/eMMC_D0
GPIOA8	30	G3	I/O	SD/eMMC_D1
GPIOA7	31	E1	I/O	SD/eMMC_D2
GPIOA6	32	E3	I/O	SD/eMMC_D3
GPIOA10	28	F2	I	SD/eMMC_WRP
GPIOA1	38	C3	I	

For more details, see Ref. [2].

17.10.2 API

Table 78. APIs for SD/eMMC interface

Item		Description
HANDLE EMMC_CREATE(void)		
Parameter	None	
Return		If it succeeds, return the handle for such device. If it fails, return NULL.

Item		**Description**
Description		Function create handle. If memory allocation fails, return NULL.
int EMMC_INIT(HANDLE handler)		
Parameter	handler	Device handle
Return		If it succeeds, return ERR_NONE. If it fails, return ERR_MMC_INIT.
Description		Initialize the SD/eMMC or SDIO card. If the function returns ERR_NONE, the card information is stored in the handle.
int EMMC_READ(HANDLE handler, UINT32 dev_addr, VOID *p_data, UINT32 block_count)		
Parameter	handler	Device handle
	dev_addr	Address
	p_data	Data pointer
	block_count	Block counter for read
Return		If it succeeds, return ERR_NONE.
Description		EMMC read command
int EMMC_WRITE(HANDLE handler, UINT32 dev_addr, VOID *p_data, UINT32 block_count)		
Parameter	handler	Device handle
	dev_addr	Address
	p_data	Data pointer
	block_count	Block counter for write
Return		If it succeeds, return ERR_NONE.
Description		EMMC write command
void EMMC_SEND_CMD(HANDLE handler, UINT32 cmd, UINT32 cmd_arg)		
Parameter	handler	Device handle
	cmd	SDIO command without response. Defined in <SDIO.h>.
	cmd_arg	SDIO command argument
Return		If it succeeds, return TRUE. If it fails, return FALSE.
Description		Send command to SDIO
void EMMC_SEND_CMD_RES(HANDLE handler, UINT32 cmd, UINT32 cmd_arg, UINT32 *rsp)		
Parameter	handler	Device handle
	cmd	SDIO command with response
	cmd_arg	SDIO command argument
	rsp	Response pointer
Return		None
Description		After this function call, the response is stored in rsp.
int EMMC_IOCTL(HANDLE handler, UINT32 cmd, VOID *data)		
Parameter	handler	Device handle
	cmd	The command that is defined in EMMC.h.
	data	Data pointer
Return		If it succeeds, return ERR_NONE.
Description		EMMC IOCTL command

Item		Description
int EMMC_CLOSE(HANDLE handler)		
Parameter	handler	Device handle
Return		If it succeeds, return ERR_NONE.
Description		EMMC driver close command

17.10.3 How to Run

1. In the e² studio, import a project for the SD_EMMC sample application.
   `~/SDK/apps/common/examples/Peripheral/SD_EMMC/projects/da16200`
2. The sample application code is written in the following source file:
   `~/SDK/apps/common/examples/Peripheral/SD_EMMC/src/sd_emmc_sample.c`
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. The sample runs as soon as the boot is complete.

```
[/DA16200]  # emmc sample start
fail / total 0 / 100
[/DA16200] #
```

5. If the SD card is not ready, then the message "emmc_init fail" is returned.
6. Connect GPIOA[9:4] to the SD card socket as shown below.

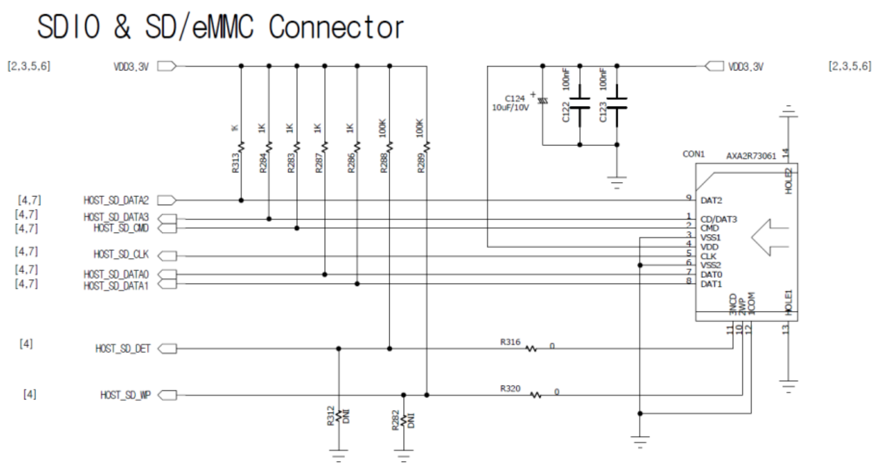


Figure 117. SDIO and SD/eMMC connector

NOTE
The CMD and DATA pins of the SD card connections are open-drain at initialization. When the SD card initialization is not working normally, it needs to use smaller pull-up resisters for CMD and DATA pins or a shorter length jumper wire of the SD card connections.

- GPIOA[9] - mSDeMMCIO_D0, GPIOA[8] - mSDeMMCIO_D1
- GPIOA[7] - mSDeMMCIO_D2, GPIOA[6] - mSDeMMCIO_D3
- GPIOA[5] - mSDeMMCIO_CLK, GPIOA[4] - mSDeMMCIO_CMD
- GPIOA[10] is not mandatory (for write protect function).

17.10.4 Sample Code

This sample code shows how the eMMC host writes random data to a slave memory card and reads back the written data to check if that data matches.

Function Emmc_verify() compares the written data with the data read from the SD memory card. The sector size of the SD memory card is 512 bytes. The "addr" variable value (210) in the code is just an example sector number in the SD memory card.

```
void emmc_init() {
...
    DA16X_CLOCK_SCGATE->Off_SysC_HIF = 0;
    DA16X_SYSCLOCK->CLK_DIV_EMMC = EMMC_CLK_DIV_VAL;
    DA16X_SYSCLOCK->CLK_EN_SDeMMC = 0x01;       // clock enable
...
```

- Set pin multiplexing

```
/*
 * SDIO Master
 */
// GPIO[9] - mSDeMMCIO_D0,  GPIO[8] - mSDeMMCIO_D1
_da16x_io_pinmux(PIN_EMUX, EMUX_SDm);
// GPIO[5] - mSDeMMCIO_CLK,  GPIO[4] - mSDeMMCIO_CMD
_da16x_io_pinmux(PIN_CMUX, CMUX_SDm);
// GPIO[7] - mSDeMMCIO_D2,  GPIO[6] - mSDeMMCIO_D3
_da16x_io_pinmux(PIN_DMUX, DMUX_SDm);
```

- Create and initialize an SD/eMMC handle

```
if (_emmc == NULL) {
    _emmc = EMMC_CREATE();
}
err = EMMC_INIT(_emmc);
```

17.11 User SFLASH Read/Write Example

17.11.1 How to Run

1. In the e² studio, import a project for the SD_EMMC sample application.
   `~/SDK/apps/common/examples/Peripheral/Sflash_API/projects/da16200`
2. The sample application code is written in the following source file:
   `~/SDK/apps/common/examples/Peripheral/Sflash_API/src/sflash_sample.c`
3. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
4. After booting, the sample starts automatically.

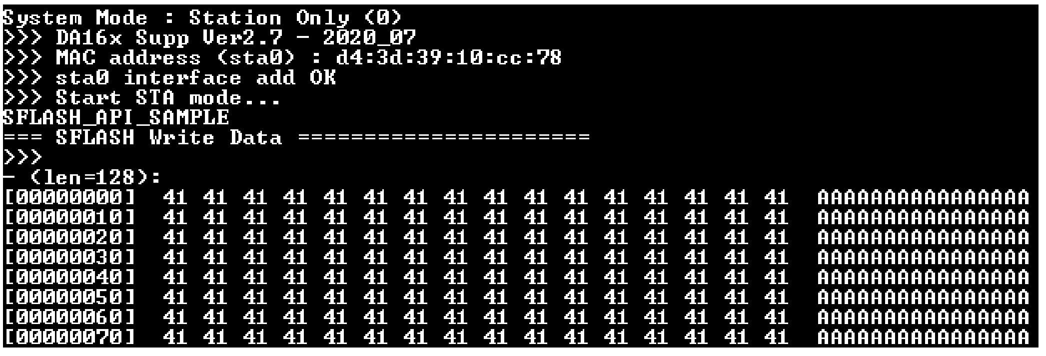


Figure 118. SFlash example sample test

17.11.2 User Task

The user task of the sflash api sample application is defined as below and executed by the system. SAMPLE_SFLASH should be a unique name to create a task. This test is not related to network initialization and DPM mode.

```
~/SDK/apps/common/examples/Peripheral/Sflash_API/src/sample_apps.c
static const app_task_info_t sample_apps_table[] = {
  SAMPLE_SFLASH, user_sflash_test,1024, USER_PRI_APP(1), FALSE, FALSE,     UNDEF_PORT,
      RUN_ALL_MODE },
};
```

17.11.3 Sample Code

17.11.3.1 Application Initialization

The user_sflash_test function is run after the basic initialization is complete.

```
void SFLASH_API_sample(void *param)
{
    /* DO SOMETHING */
    PRINTF("SFLASH_API_SAMPLE\n");

    test_sflash_write();
    vTaskDelay(10);  // Dealy 100 msec

    test_sflash_read();
    vTaskDelete(NULL);

    return ;
}
```

17.11.3.2 SFlash Read and Write

```
// user sflash APIs

extern UINT user_sflash_read(UINT sflash_addr, VOID *rd_buf, UINT rd_size);
sflash_addr: see above user sflash area
rd_buf: buffer to which data is copied
rd_size: data size

extern UINT user_sflash_write(UINT sflash_addr, UCHAR *wr_buf, UINT wr_size);
sflash_addr: see above user sflash area
rd_buf: buffer from which data is copied to sflash_addr
rd_size: data size

...

void test_sflash_write(void)
{
    UCHAR     *wr_buf = NULL;
    UINT      wr_addr;

#define       SFLASH_WR_TEST_ADDR  SFLASH_USER_AREA_START
#define       TEST_WR_SIZE         SF_SECTOR_SZ

    wr_buf = (UCHAR *)malloc(TEST_WR_SIZE);

    if (wr_buf == NULL) {
        PRINTF("[%s] malloc fail ...\n", __func__);
        return;
    }
```

```
    memset(wr_buf, 0, TEST_WR_SIZE);

    for (int i = 0; i < TEST_WR_SIZE; i++) {
        wr_buf[i] = 0x41;   // A
    }

    wr_addr = SFLASH_WR_TEST_ADDR;

    PRINTF("=== SFLASH Write Data ======================\n");
    user_sflash_write(wr_addr, wr_buf, TEST_WR_SIZE);
}


void test_sflash_read(void)
{
    UCHAR     *rd_buf = NULL;
    UINT      rd_addr;
    UINT      status;

#define       SFLASH_RD_TEST_ADDR  SFLASH_USER_AREA_START
#define       TEST_RD_SIZE         (1 * 1024)

    rd_buf = (UCHAR *)malloc(TEST_RD_SIZE);

    if (rd_buf == NULL) {
        PRINTF("[%s] malloc fail ...\n", __func__);
        return;
    }

    memset(rd_buf, 0, TEST_RD_SIZE);

    rd_addr = SFLASH_RD_TEST_ADDR;
    status = user_sflash_read(rd_addr, (VOID *)rd_buf, TEST_RD_SIZE);

    if (status == TRUE) {
        hex_dump(rd_buf, 128);
    }

    free(rd_buf);
}
```

NOTE
user_sflash_read/write is a blocking function. Take special care when running this code under DPM mode enabled (sleep mode 2 or sleep mode 3 applications): when invoking user_sflash_write( ), make sure to get the result before the DPM sleeping API is invoked.

17.12 OTP

17.12.1 Introduction

The DA16200/DA16600 includes a one-time electrically field programmable non-volatile CMOS memory. This memory is to protect and manage major information essential for mass production and management of products, such as booting information, MAC address, serial number, and others.

OTP is also used for storing secret information which is used for the advanced security functions such as secure booting, secure debugging, and secure asset storage. But this secret information cannot be accessed in a normal way of CPU read or write access so that it is protected from external access.

Table 79. OTP map

Offset	Field	Size (Bytes)
0x000	Renesas Reserved	1024
0x100	MAC Address #0 Low	4
0x101	MAC Address #0 High	4
0x102	MAC Address #1 Low	4
0x103	MAC Address #1 High	4
0x104	MAC Address #2 Low	4
0x105	MAC Address #2 High	4
0x106	MAC Address #3 Low	4
0x107	MAC Address #3 High	4
0x10A	XTAL Offset #0	4
0x10B	XTAL Offset #1	4
0x10C to 0x1FE	User Area	972

17.12.2 API

Table 80. OTP API list

Item		Description
void otp_mem_create(void)		
Parameter	None	
Return		None

Description		Initialize OTP hardware.
		Before calling this function, it needs otp_clock_enable.
		{
		DA16200_SYSCLOCK ->PLL_CLK_EN_4_PHY = 1;
		DA16200_SYSCLOCK ->CLK_EN_PHYBUS = 1;
		extern void DA16X_SecureBoot_OTPLock(unsigned int mode);
		DA16X_SecureBoot_OTPLock(1); // unlock
		#define CLK_GATING_OTP 0x50006048
		MEM_BYTE_WRITE(CLK_GATING_OTP, 0x00);
		otp_mem_create();
		}

void otp_mem_close(void)		
Parameter	None	
Return		None
Description		Close the OTP hardware.
int otp_mem_read(UINT32 offset, UINT32 *data)		
Parameter	offset	OTP memory offset (0x00 ~ 0x1FE)
	data	[out] data pointer of buffer
Return		OTP_OK if it succeeds.
Description		Each offset stores 32-bit data. Offset 0x00 to 0x2c used for security purposes. So, it may not be accessible. See Table 79.

Item		Description
int otp_mem_write (UINT32 offset, UINT32 data)		
Parameter	offset	OTP memory offset (0x00 ~ 0x1FE)
	data	Data to write
Return		OTP_OK if it succeeds.
Description		Offset 0x00 to 0x2c used for security purposes. Do not write any data within. See Table 79.
int otp_mem_lock_read (UINT32 offset, UINT32 *data)		
Parameter	offset	Lock status offset. Always (0xFFF).
	data	Data pointer of lock status
Return		OTP_OK if it succeeds.
Description		The OTP memory can be locked. Each lock bit can lock range ~ 0x40. For example: lock status value 0x00000002, it means that offset 0x40~0x7F OTP memory locked. lock bit 0 lock offset 0 ~ 0x3F lock bit 1 lock offset 0x40 ~ 0x7F lock bit 2 lock offset 0x80 ~ 0xBF lock bit 3 lock offset 0xC0 ~ 0xFF lock bit 4 lock offset 0x100 ~ 0x13F lock bit 5 lock offset 0x140 ~ 0x17F lock bit 6 lock offset 0x180 ~ 0x1BF lock bit 7 lock offset 0x1C0 ~ 0x1FE
int otp_mem_lock_write (UINT32 offset, UINT32 *data)		
Parameter	offset	Lock status offset. Always (0xFFF).
	data	Lock status value.
Return		OTP_OK if it succeeds.
Description		Refer otp_mem_lock_read().

17.13 Bluetooth LE Coexistence

The Bluetooth® Low Energy (LE) coexistence feature can be enabled and disabled through a configuration register. The activation scenarios based on the status of each pin are as follows:

- BT_sig0 (oWlanAct)
  - When asserted, the external Bluetooth/Bluetooth® LE device is expected to stop occupying RF.
  - This signal can be configured as always high by software to block the external Bluetooth/Bluetooth® LE device from occupying RF. See Section 17.13.4 for details
- BT_sig1 (iBtAct)
  - When asserted, the DA16200/DA16600 stops occupying RF
- BT_sig2 (iBtPri)
  - This is optional.
  - When iBtPri is active, the DA16200/DA16600 stops occupying RF when iBtAct is active even if a Wi-Fi transmission is in progress.

When both DA16200/DA16600 and Bluetooth/Bluetooth® LE try to transmit a packet at the same time, there is a configuration in the DA16200/DA16600 that determines which has the priority over the other.

When the priority of the DA16200/DA16600 is set to be higher than Bluetooth/Bluetooth® LE, it ignores iBtAct signal and transmits its packet anyway. When the priority of the DA16200/DA16600 is set to be lower than

Bluetooth/Bluetooth® LE, it delays transmission of its packet until Bluetooth/Bluetooth® LE de-asserts the iBtAct signal.

Priority can be set through an API which is described in Section 17.13.4.

17.13.1 Pin Configuration

Table 81 shows the 3-pin configuration of Bluetooth® LE coexistence interface. If the iBTPri pin is not controlled, it must be configured as pull-up or pull-down and not high-z to avoid any leakage.

Table 81. 3-Pin Bluetooth® LE coexistence pin configuration

Signals	Description	I/O	DA16200 GPIO No.	Pin number			
				DA16200	DA16200MOD	DA16600MOD	
						DA16200	DA14531
oWlanAct	Wi-Fi active signal	O	GPIOA8	30	25	35	48 (P0_5)
iBtAct	Bluetooth active signal	I	GPIOA9	29	24	34	3 (P0_6)
iBtPri	Bluetooth priority	I	GPIOA10	28	23	33	47 (P0_7)

When GPIOA8 and GPIOA9 are assigned as either an SDIO or SPI interface, only the GPIOA10 pin should be used for Bluetooth® LE Coexistence. In this case, the GPIOA10 pin must be connected to the iBtAct pin of the Bluetooth/Bluetooth® LE chipset to coordinate the use of the RF signal between the DA16200/DA16600 and Bluetooth/Bluetooth® LE chipsets Table 82 shows 1-pin configuration of Bluetooth® LE coexistence.

Table 82. 1-Pin Bluetooth® LE coexistence pin configuration

Signals	Description	I/O	DA16200 GPIO No.	Pin number			
				DA16200	DA16200MOD	DA16600MOD	
						DA16200	DA14531
iBtAct	Bluetooth active signal	I	GPIOA10	28	23	33	3 (P0_6)

17.13.2 Pin Multiplex

Pin multiplexing for the Bluetooth® LE coexistence feature can be configured by modifying the `initialize_bt_coex(void)` function in the "rf_meas_api.c" file as follows:

```
// pin mux setup for Bluetooth® LE coexistence
#ifndef __SUPPORT_BTCOEX_1PIN__
      _da16x_io_pinmux(PIN_EMUX, EMUX_BT);
#endif
     _da16x_io_pinmux(PIN_FMUX, FMUX_GPIOBT);
```

17.13.3 SDK Feature Definition

The Bluetooth® LE coexistence feature can be enabled in the DA16200/DA16600 SDK in the `config_generic_sdk.h` file as follows:

- 3-Pin Bluetooth® LE Coexistence

```
#define __SUPPORT_BTCOEX__              // BT Coexistences
```

- 1-Pin Bluetooth® LE Coexistence

```
#define __SUPPORT_BTCOEX__              // BT Coexistences
#define __SUPPORT_BTCOEX_1PIN__         // BT Coexistences with 1 pin
```

NOTE
When 1-pin Bluetooth® LE Coexistence is defined in "config_generic_sdk.h," the GPIOA10 pin should be connected to the iBtAct pin of the Bluetooth/Bluetooth® LE chipset.

17.13.4 API

Table 83. APIs for Bluetooth® LE coexistence

Item	Description
void rf_meas_btcoex(uint8_t enable, uint8_t priority, uint8_t polarity);	
enable	0 or 1 (1: enable, 0: disable)
priority	Priority: 0, 1, 2, 3 0: BT > WLAN (BT priority is higher than WLAN) 1: BT = WLAN (BT and WLAN priorities are equal) 2: BT < WLAN (WLAN priority is higher than BT) 3: BT < WLAN, oWlanAct is forced to be always high
polarity	Polarity: 0 and 1 for oWlanAct, iBtAct, and iBtPri PIN 0: Normal (Active-High) 1: Inverted (Active-low)
Return	None

Example code for the Bluetooth® LE coexistence API when using 3 pins:

```
// The 1st Element means BT coexistence is enabled
rf_meas_btcoex(1, 0, 0); // 0: Bluetooth® win in conflict
rf_meas_btcoex(1, 2, 0); // 2: WLAN win in conflict
rf_meas_btcoex(1, 3, 0); // 3: Set oWlanAct pin high always, available in SDK v3.2.6.0 or
later
```

Example code for the Bluetooth® LE coexistence API when using 1 pin:

```
rf_meas_btcoex(1, 0, 0); // 1: BT coexistence is enabled, 0: Bluetooth® win in conflict
```

To change the default setting, after starting the system, `rf_meas_btcoex(1, 0, 0)` is called in `user_main()` to set the default configuration of the Bluetooth® LE coexistence interface.

The `rf_meas_btcoex()` function can be called at any time after system startup to reconfigure the priorities without requiring a reboot.

17.14 RTC Timer in DPM

This sample code describes how to use the RTC timer for waking up from Sleep mode 2 or Sleep mode 3 and, see Ref. [2] for further details.

17.14.1 How to Run

1. In the e² studio, import a project for the RTC timer sample application.

   `~/SDK/apps/common/examples/Peripheral/RTC_Timer_DPM/projects/da16200`

2. Build the DA16200 SDK, download the RTOS image to the DA16200 EVB, and reboot.
3. Use the console terminal of DA16200 EVB to set up the Wi-Fi station interface and enable DPM mode.
4. After rebooting, the RTC timer sample application starts automatically.

   The user can select Sleep modes in the sample code.

```
/* Defines for sample */
#undef    SAMPLE_FOR_DPM_SLEEP_2          // Sleep mode 2
#define   SAMPLE_FOR_DPM_SLEEP_3          // Sleep mode 3
```

17.14.2 Timer Creation: Sleep Mode 2

Sleep mode 2 powers off all components of DA16200 except RTC. However, for test purposes, retention memory can be powered on. External wake-up or RTC timer can wake up a DUT (Device Under Test) in Sleep mode 2. If the RTC timer is not set, the DUT in Sleep mode 2 is not woken up by the RTC timer. When the DUT is woken up by wake-up sources, it works using the data of retention memory.

To go to Sleep mode 2, run API dpm_sleep_start_mode_2().

```
Void rtc_timer_sample(void * param)
{
    unsigned long long     wakeup_time;

    /* Just work in case of RTC timer wakeup */
    if ( dpm_mode_is_wakeup() == DPM_WAKEUP
          && dpm_get_wakeup_source() != WAKEUP_COUNTER_WITH_RETENTION) {
        dpm_app_sleep_ready_set(SAMPLE_RTC_TIMER);
        return;
    }

    /* TRUE : Maintain RTM area for DPM operation */
    wakeup_time = MICROSEC_FOR_ONE_SEC * RTC_TIMER_WAKEUP_ONCE;
    dpm_sleep_start_mode_2(wakeup_time, TRUE);
}
```

17.14.3 Timer Creation: Sleep Mode 3

Sleep mode 3 powers off all components except RTC, retention memory, and pTIM (should be in running state to be connected to Wi-Fi). Sleep mode 3 is always connected to Wi-Fi, but Sleep mode 2 needs to connect to the Wi-Fi network first and transmits or receives data. For more detailed information on Sleep mode 3, see Ref. [3].

This sample code shows how to create the one-shot RTC timer and a periodic RTC timer.

```
void rtc_timer_sample(void * param)
{
    ULONG     status;

    if (dpm_mode_is_wakeup() == NORMAL_BOOT) {

       /*
        * Create a timer only once during normal boot.
        */

       dpm_app_sleep_ready_clear(SAMPLE_RTC_TIMER);

       /* One-Shot timer */
       status = dpm_timer_create(SAMPLE_RTC_TIMER,
                                 "timer1",
                                  rtc_timer_dpm_once_cb,
                                  RTC_TIMER_WAKEUP_ONCE,
                                  0);
       switch ((int)status) {
        case DPM_MODE_NOT_ENABLED   :
        case DPM_TIMER_SEC_OVERFLOW :
        case DPM_TIMER_ALREADY_EXIST:
        case DPM_TIMER_NAME_ERROR   :
        case DPM_UNSUPPORTED_RTM    :
        case DPM_TIMER_REGISTER_FAIL:
        case DPM_TIMER_MAX_ERR :
            PRINTF(">>> Fail to create %s timer (err=%d)\n",
                   SAMPLE_CUR_TIME_DPM, (int)status);
            // Delay to display above message on console ...
            vTaskDelay(2);

            break;
```

```
        }

        /* Periodic timer */
        status = dpm_timer_create(SAMPLE_RTC_TIMER,
                              "timer2",
                              rtc_timer_dpm_periodic_cb,
                              RTC_TIMER_WAKEUP_PERIOD,
                              RTC_TIMER_WAKEUP_PERIOD);
              switch ((int)status) {
        case DPM_MODE_NOT_ENABLED   :
        case DPM_TIMER_SEC_OVERFLOW :
        case DPM_TIMER_ALREADY_EXIST:
        case DPM_TIMER_NAME_ERROR   :
        case DPM_UNSUPPORTED_RTM    :
        case DPM_TIMER_REGISTER_FAIL:
        case DPM_TIMER_MAX_ERR      :
            PRINTF(">> Fail to create %s timer (err=%d)\n",
                   SAMPLE_CUR_TIME_DPM, (int)status);

         // Delay to display above message on console ...
         vTaskDelay(2);

         break;
       }

        dpm_app_sleep_ready_set(SAMPLE_RTC_TIMER);
    } else {
        /* Notice initialize done to DPM module */
        dpm_app_wakeup_done(SAMPLE_RTC_TIMER);
    }

    while (1) {
        /* Nothing to do... */
       vTaskDelay(100);
    }
}
```

18. DA16600 Example Applications

The DA16600 module is comprised of the DA16200 (Wi-Fi) and the DA14531 (Bluetooth® LE) SoC. The two chips exchange data with each other through the GTL interface. This document describes the test steps and code walkthrough of four example applications with the DA16600.

The DA16600 SDK has four example applications as listed below and all of them support Wi-Fi provisioning and OTA download.

- Gas leak detection sensor
- TCP client in DPM
- DA14531 peripheral driver
- IoT sensor gateway

The applications 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**

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

To develop a function that talks to a Bluetooth® LE peer (for example, can talk to the Provisioning Mobile Application), see Ref. [9] for details.

To develop a local function such as driver function in the DA14531 (for example, to handle a custom GTL message that should be handled in the DA14531), the user also needs to understand the local APIs of the DA14531. See Ref. [10] or the API documentation included in the DA14531 SDK.

For the hardware configuration of the DA16600 EVB, see Ref. [3].

18.1 Source Structure and Common APIs

The DA16600 example applications working with Bluetooth® LE (DA14531) are added into the DA16200 SDK. The Wi-Fi (DA16200) and Bluetooth® LE (DA14531) chips are connected through a four-wire UART (TX, RX, RTS, and CTS, the baud rate is 115200 by default) and communicating with each other over Renesas Electronics' proprietary GTL interface. In the GTL architecture, a Bluetooth® LE application is running on the external host (DA16200).

As the GTL architecture and the DA16200 based SDK are used in the DA16600, the application developer should understand and know how to use the user APIs for both host platforms – the DA16200 SDK and Bluetooth® LE platform (DA14531).

18.1.1 DA16600 Bluetooth® Source Structure

Figure 119 shows the folder structure for Wi-Fi and Bluetooth® LE applications:

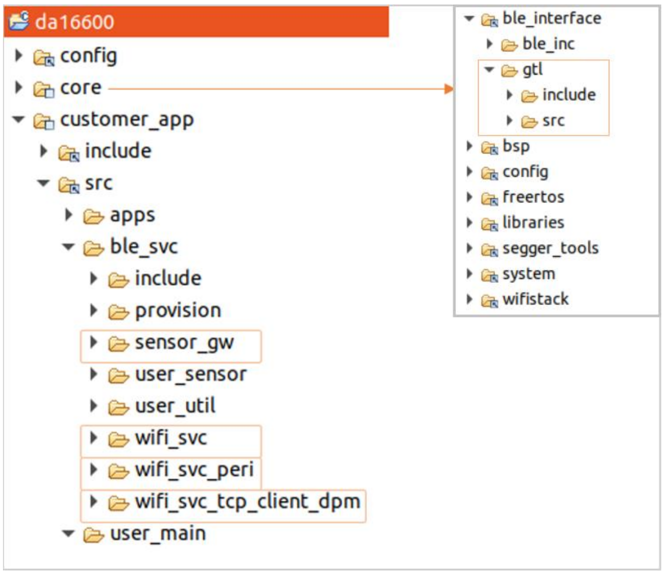


Figure 119. DA16600 Bluetooth® source structure

- **core/ble_interface/gtl** folder contains Bluetooth® LE image load, boot, and reset
- **customer_app/src/ble_svc** folder contains four example applications: The examples support the Wi-Fi provisioning application – GATT Server implementation to communicate with Bluetooth® LE peer applications (Wi-Fi Provisioning Mobile APP) and OTA download as default
  - **sensor_gw**: **IoT Sensor Gateway**

    GAP Central example application based on a Bluetooth® LE example; this is a GATT Client application used in this application.

  - **wifi_svc**: **Gas leak detection sensor application**

    GAP Peripheral example application based on a Bluetooth® LE example, works with a gas leak sensor (virtual) that is locally connected to the DA14531 chip. When a gas leak event occurs, this application posts a message to a network server in TCP/IP network.

  - **wifi_svc_tcp_client_dpm**: **TCP Client DPM application (default enabled)**

    GAP Peripheral example application based on a Bluetooth® LE example, a pure TCP/IP network application that communicates with a TCP Server in the connected network.

  - **wifi_svc_peri**: **DA14531 Peripheral driver sample application**

    GAP Peripheral example application based on a Bluetooth® LE example, this configures and runs some peripheral devices locally attached to DA14531.

18.1.2 Application APIs and Console Commands

Table 84 and Table 85 show the list of common APIs used in the example applications.

Table 84. Application functions

Item	Description
system_start	Entry point for customer main.
combo_init	4-wire UART initialization, the DA14531 firmware load.
wlaninit	WLAN initialization
gtl_main	Main GTL message handler
gtl_host_ifc_mon	GTL/UART1 RX monitoring task
ble_app_usr_cmd_hdler	Bluetooth® LE Application User Command handler
BleReceiveMsg	Receive a message from the DA14531 through the GTL.
BleSendMsg	Send a message to the DA14531 through the GTL.

Item	Description
start_user_apps	Start system application, Entry point of user's applications defined in user_apps_table[].
initialize_bt_coex	Initialize the DA16600 Wi-Fi and Bluetooth® LE Combo module.

Table 85. Major console commands

Root commands	Description
help	CMD-list display and help command.
top	Go to the ROOT directory.
up	Go to the upper directory.
dpm	DPM enable/disable
factory	Factory reset, if type ‘y’ after this command, factory reset is complete.
getwlanmac	Show MAC_addr
ping	Ping help
reboot	Device reboot command
ver	Version display
Sub-commands	
ble	Bluetooth® LE application commands
net	Network commands
nvram	NVRAM commands
sys	System commands
user	User commands

18.2 Environment Setup

The DA16600 module consists of two SoC chips – DA16200 and DA14531. The firmware images of the two chips are stored in the SPI flash memory of the DA16600 module. The flash memory is only accessible by DA16200 and not accessible by DA14531 directly. That is, the DA16200 reads and transfers it to DA14531.

The list of firmware images required to run each SoC chip is as follows:

DA16200: Wi-Fi chip

- Two image files:
  - **FBOOT** image: secondary bootloader
  - **FRTOS** image: main operation software which user applications are built
- Code storage memory
  - SFlash of DA16200

DA14531: Bluetooth® LE chip

- Single image file:
  - Main image: main operation software which user applications are built
- Code storage memory
  - SFlash of DA16200 or DA14531 OTP memory (32 kB allocated for OTP image)
    - OTP memory can be used to burn a default image but since OTP can only be written once, the firmware cannot be updated. If an OTA firmware update is required, then the SFlash of the DA16200 should be used to store the DA14531 firmware.

18.2.1 SFlash Memory Map

The following code shows the DA16200 source code which defines the address where the Bluetooth® LE firmware is stored in SFlash.

```
.\core\bsp\driver\include\DA16200\da16200_map.h

...
/*
 * 0x003A_D000 BLE Area  0x10000 ~ 0x15000 ( 64 KB MIN ~ 84 KB MAX)
 * Bluetooth Firmware Size Max 0x10000 ~ 0x14000 ( 64 KB MIN ~ 80 KB MAX)
 * BLE Security DB       0x00000 ~ 0x01000 ( 00 KB MIN ~ 04 KB MAX)
…
*/
…
#define SFLASH_14531_BLE_AREA_START
  (SFLASH_NVRAM_BACKUP + SFLASH_NVRAM_BACKUP - SFLASH_NVRAM_ADDR) // 0x003AD000

/* DA14531 Bluetooth LE Firmware start */
#define SFLASH_BLE_FW_BASE        (SFLASH_14531_BLE_AREA_START)
```

Two image banks are defined in the SFlash memory map for storing firmware, one for the DA16200 image and one for the DA14531 image.

The following sections describe how to build the firmware for the DA16200 and the DA14531 based on the memory map above.

18.2.2 Build the DA16600 SDK

To build the FreeRTOS based version of the DA16600 SDK, install the e^2 studio. See Ref. [3] for details on how to install and set up the development environment.

This section describes four example applications and provides instructions on how to configure and build the SDK. Each application supports provisioning of the Wi-Fi interface through the Bluetooth® LE device and OTA firmware download through the Wi-Fi interface. Before building one of the following four applications, the key features for the selected application should be configured in the following main header file:

`.\apps\da16600\get_started\include\apps\user_custom_config.h`

18.2.2.1 Gas Leak Detection Sensor Example Feature

This application supports the example described in Section 18.5.

- Change the features as follows.

  ```
  #define __BLE_PERI_WIFI_SVC__
  #undef __BLE_PERI_WIFI_SVC_TCP_DPM__
  #undef __BLE_PERI_WIFI_SVC_PERIPHERAL__
  #undef __BLE_CENT_SENSOR_GW__
  ```

18.2.2.1.1 How to Add Security Feature

In addition to Gas Leak Detection Sensor Example Feature, if security needs to be enabled, then enable it in `ble_combo_features.h(.\apps\da16600\get_started\include\apps\)` as follows:

- `#define __WIFI_SVC_SECURITY__`

NOTE
There are two pairing modes in Bluetooth pairing mechanism and they depend on the test mobile phone's Bluetooth® authentication capability configuration: **Legacy Pairing**: the mobile application may show an input box and ask a user to enter a passkey that can be found on the display of the DA16600 hardware, and then a user needs to enter the exact passkey on the test smartphone to successfully connect to the DA16600. **Secure Connection Pairing** (SC Pairing): the mobile application may show a PIN code on the test mobile and ask a user to compare the PIN code with the one printed on the display of the DA16600 hardware. If the PIN code matches, click the **OK** button to connect the DA16600 to test the mobile application. The DA16600 example currently can save bond information for up to ten Bluetooth® LE peers.

NOTE
When the bond information is stored in the test mobile phone, the test mobile phone's Bluetooth® LE peer application can be connected to the DA16600 without the need to repeat the pairing process. If either party loses the pairing credentials, the pairing process starts again when trying to reconnect.

18.2.2.2 TCP Client in DPM Example Feature

This application supports the examples described in Section 18.6.

- **Change the features as follows**:

```
#undef __BLE_PERI_WIFI_SVC__
#define __BLE_PERI_WIFI_SVC_TCP_DPM__
#undef __BLE_PERI_WIFI_SVC_PERIPHERAL__
#undef __BLE_CENT_SENSOR_GW__
```

18.2.2.3 Peripherals in DA14531 Driver Example Feature

This application supports the examples described in Section 18.7.

- **Change the features as follows**:

```
#undef __BLE_PERI_WIFI_SVC__
#undef __BLE_PERI_WIFI_SVC_TCP_DPM__
#define __BLE_PERI_WIFI_SVC_PERIPHERAL__
#undef __BLE_CENT_SENSOR_GW__
```

18.2.2.4 IoT Sensor Gateway Example Feature

This application supports the examples described in Section 18.8.

- **Change the features as follows**:

```
#undef __BLE_PERI_WIFI_SVC__
#undef __BLE_PERI_WIFI_SVC_TCP_DPM__
#undef __BLE_PERI_WIFI_SVC_PERIPHERAL__
#define __BLE_CENT_SENSOR_GW__
```

18.2.2.5 Build SDK in e^2 studio IDE

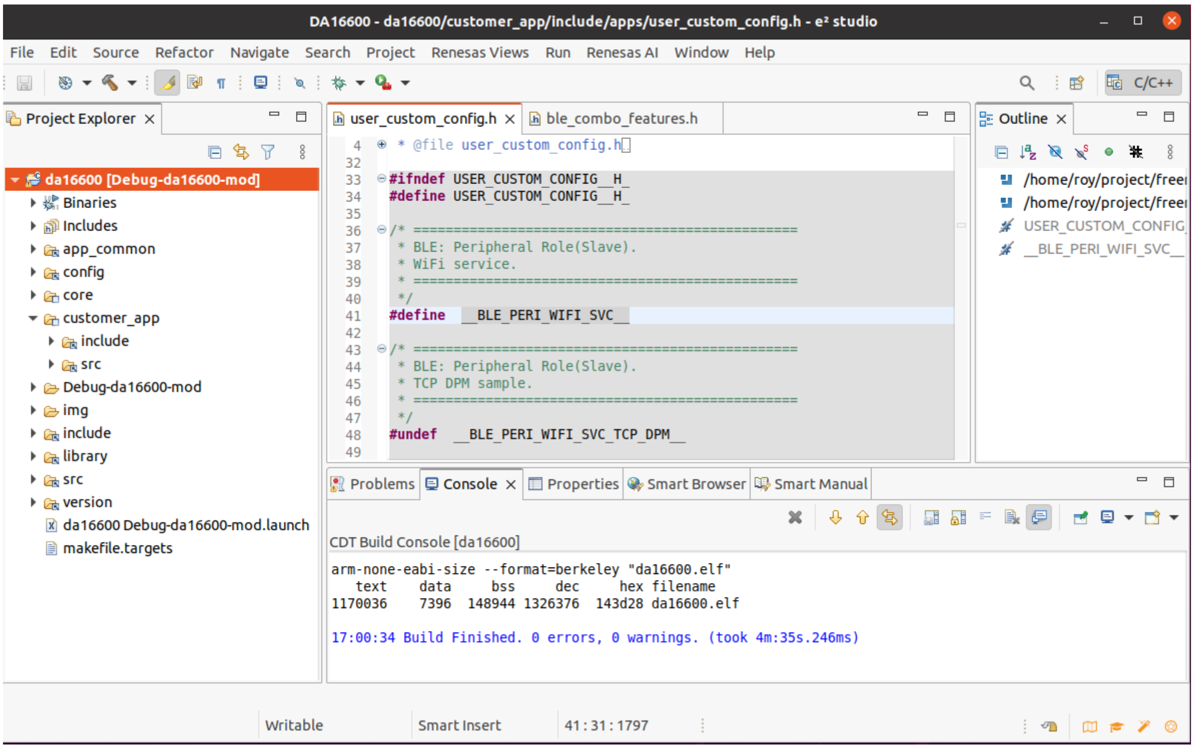


Figure 120. Project view

To build the SDK based on FreeRTOS, right-click the **da16600** folder in the project explorer panel, and then select **Build project** in the list. After the build is complete, the following two DA16200 images can be found in the `[DA16600_SDK_ROOT]\apps\da16600\get_started\projects\da16600\img\` folder:

- DA16600_FRTOS-*.img
- DA16600_FBOOT-*.img

18.2.3 Build DA14531 SDK

The DA14531 software (Bluetooth® LE Software) used in the DA16600 SDK is based on the DA14531 SDK version 6.0.14.1114. To build the DA14531 software, it needs a DA14531 SDK 6.0.14.1114 that is specifically adapted for DA16600. It is available in `[DA16600_SDK_ROOT]\utility\combo\da14531_sdk_v_xxx.zip`. The DA14531 SDK project is determined by which DA16600 example application is required.

18.2.3.1 DA14531 Peripheral Role Project

The **peripheral role project** (used for 18.2.2.1, 18.2.2.2, and 18.2.2.3) is located in

[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex.

18.2.3.2 DA14531 Central Role Project

The **central role project** (used for 18.2.2.4) is located in
[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_monitor_aux_ext_coex.

18.2.3.3 Install Keil

For information on the Keil installation, see Ref. [8].

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

18.2.3.4 Build Project

To build a project in Keil, go to **Project** > **Open Project**, and then select the **.uvprojx** file.
For **peripheral** role project example, open

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

Or for **central** role project example, open `[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_monitor_aux_ext_coex\Keil_5\prox_monitor_ext.uvprojx`.

On the project window as shown in Figure 121.

1. Go to **Project** > **Clean Targets**.
2. Click **Rebuild**.

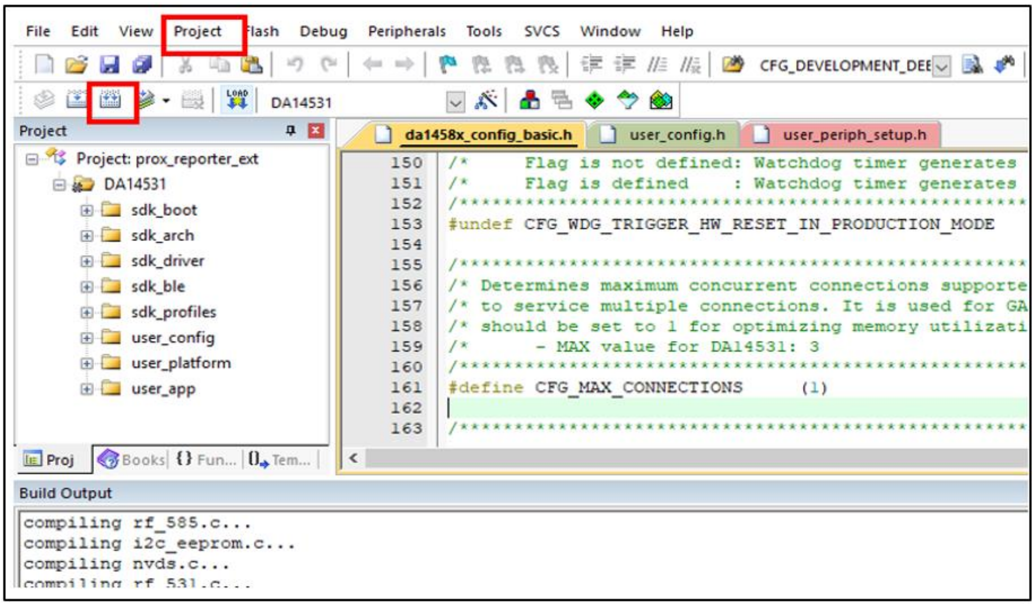


Figure 121. Keil – build

18.2.3.4.1 Peripheral Role Image

After **Peripheral role projects** are built with the Keil IDE, the pxr_coex_ext_*.bin file is generated in `[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex\Keil_5\out_DA14531\Objects\`.

Based on the bin file, following *.img files are generated as well for Sflash image update in

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

- da14531_multi_part_proxr.img: for Sflash image update
- `pxr_sr_coex_ext_***_ota.img`: for image update by OTA

18.2.3.4.2 Central Role Image

After **Central role project** are built with the Keil IDE, the pxm_coex_ext_*.bin file is generated in `[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prom_monitor_aux_ext_coex\Keil_5\out_DA14531\Objects\`.

Based on the bin file, following *.img files are generated as well for Sflash image update in

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

- `da14531_multi_part_proxm.img`: for Sflash image update
- `pxm_sr_coex_ext_***_ota.img`: for image update by OTA

NOTE
To quickly test an example without building the DA14531 software, use the pre-built DA14531 image - da14531_multi_part_prox*.img. See the folder `[DA16600_SDK_ROOT]\apps\da16600\get_started\projects\da16600\img\DA14531_x`: DA14531_P – **Peripheral role** DA14531_C – **Central role** If you want to run multiple DA16600 boards at the same time (with the same example project), this feature is available to get random BD address - USE_BLE_RANDOM_STATIC_ADDRESS, otherwise you need to build the DA14531 projects with different BD addresses and update the DA14531 firmware. 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).

18.2.4 Firmware Image Update

After building DA16200 SDK, the relevant firmware images are located properly in

`.\apps\da16600\get_started\projects\da16600\img\` folder.

But after building DA14531 SDK, the built image - da14531_multi_part_prox*.img should be copied to the above folder manually. And some notes are similar to items listed below for the DA14531 images.

- da14531_multi_part_prox*.img: this is a multi-part format image and the file used for direct download to flash, when new built image is required, copy this file to the folder where the DA16200 images are located (`.\apps\da16600\get_started\projects\da16600\img\`, described in Section 18.2.2.5), then download into the flash as described in the sections 18.2.4.1 or 18.2.4.2.
- px*_coex_ext_*_ota.img: this file is a single-part format image for the OTA update used in Section 18.4.

18.2.4.1 Firmware Update with *.ttl File

In the `[DA16600_SDK_ROOT]\apps\da16600\get_started\projects\da16600\img\` folder, the following .ttl file can be found in

- da16600_da14531_**P**_download.ttl

  This script file is used for **peripheral** example applications using in Sections 18.2.2.1, 18.2.2.2, and 18.2.2.3
- da16600_da14531_**C**_download.ttl

  This script file is used for **central** example application using in Section 18.2.2.4

Make sure that all images are ready in the `[DA16600_SDK_ROOT]\apps\da16600\get_started\projects\da16600\img\` folder as follows:

- DA16600_FBOOT-*.img
- DA16600_FRTOS-*.img
- da14531_multi_part_prox**r**.img in DA14531_P
- da14531_multi_part_prox**m**.img in DA14531_C

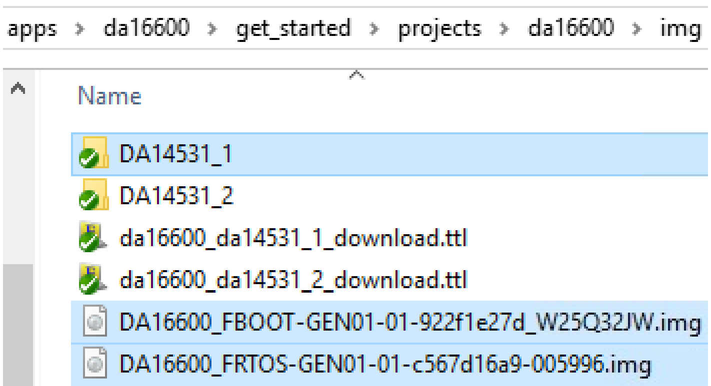


Figure 122. DA16600 images and .ttl files to program

When the files are ready, complete the following steps:

1. Put a micro-USB cable in the CN1 USB port of the DA16600 board (see Ref. [3]).
   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 your computer to make a "power cycle" of the board easy during the test.
2. Run Tera Term.
   Windows detects two USB ports (for example, COM34 and COM35).
3. Connect Tera Term to the lowest of the two COM port numbers that were detected (for example, COM34, which is UART0 of DA16200).
4. Make sure that the baud rate is 230400.
5. Press **Enter** several times to check that it is online with the DA16600 EVB.
6. Type `reset` and then press **Enter**. Check the `[MROM]` prompt.
7. In the Tera term, go to **Control** > **Macro**, then browse and select the .ttl file in the img folder.

The three images are programed step by step and then rebooted. Now it is ready to test.

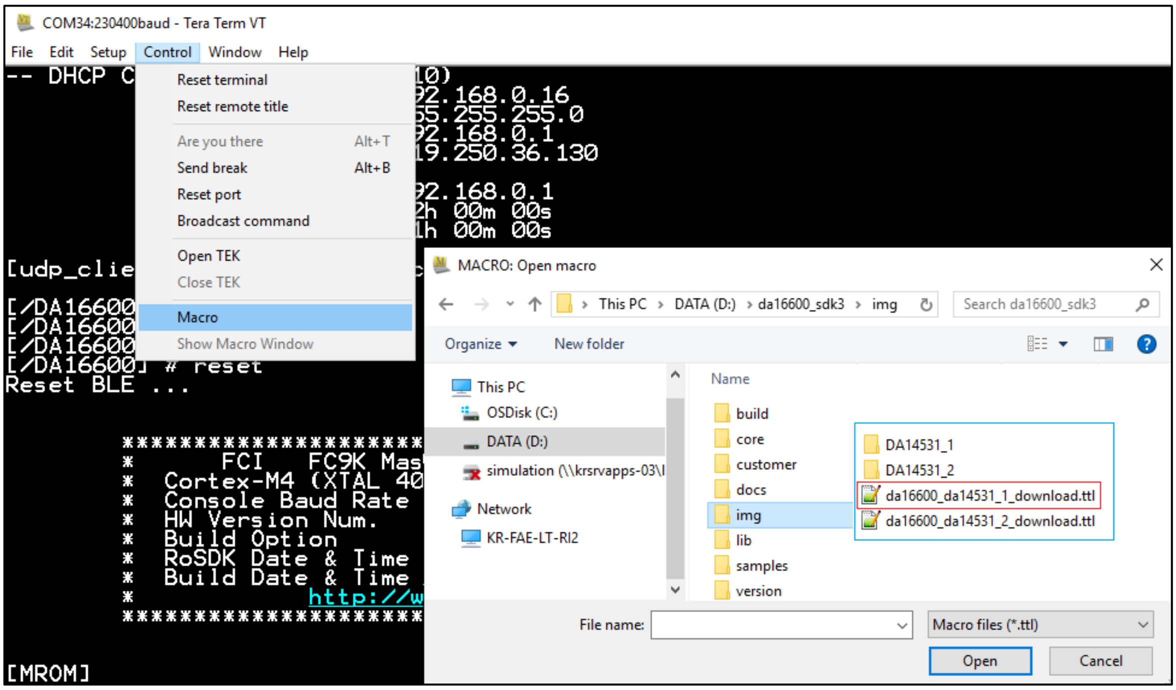


Figure 123. Steps to program by .ttl file

18.2.4.2 Firmware Update Without .ttl File

To program SFlash without .ttl file, check the steps (until step 6) described in Section 18.2.4.1.

- Type `reset` and then press **Enter**. Check the `[MROM]` prompt)

Then, complete the following steps:

1. Before entering a command, copy the location path to the folder where the three firmware images are stored (FBOOT, RTOS, and DA14531 image) in advance for sure so that you do not get a timeout error while running the loady command. If you select a file slowly, the Tera Term's ymodem transfer progress dialog is stuck or not working. Then, you need to re-run the loady command.
2. Type `loady 0 1000` and press **Enter**.
3. Go to **File** > **Transfer** > **YMODEM** > Send to quickly find file DA16600_FBOOT-*.img, see Figure 124.
   These shortcut keys can be used: keep the **Alt + F and T, Y, S**.

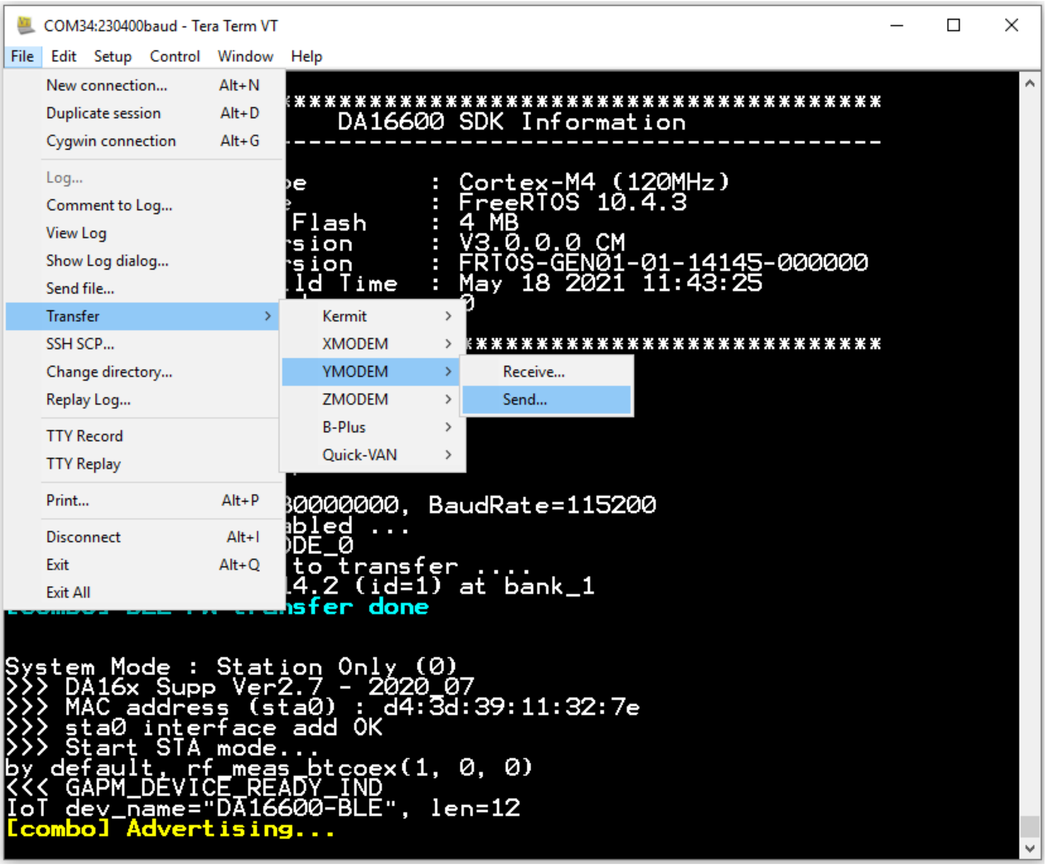


Figure 124. Tera Term

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

4. Similar to step 3, type `loady 23000 1000` and press **Enter.** Then, select **DA16600_FRTOS-*.img**. This ymodem transfer takes the longest time to complete.
5. Similar to step 3, type `loady 3ad000 1000 bin` and press **Enter.** Then, select **da14531_multi_part_proxr.img** or **da14531_multi_part_proxm.img**.

NOTE
When downloading only one image file (RTOS or da14531), Renesas strongly recommends downloading the FBOOT image first (`loady 0 1000`) and then either `loady 23000 1000` for RTOS or `loady 3ad000 1000 bin` for da14531.

6. After all images are transferred to SFlash, type `boot`, and then press **Enter**.
   Some debug messages are printed in Tera Term.
7. Press **Enter** several times until the prompt `[/DA16600] #` shows.

IMPORTANT
Switch off and on the USB port (if a user USB hub has a power switch per port, then toggle it) or remove the USB cable completely and then put it back in. This is needed to change the DA14531 to Bootloader mode and wait to get an image from the DA16600.

8. Wait until the Tera Term is connected to COMxx (for example, COM34). Or simply reconnect with serial. This step is not needed if Tera Term is not used during the test.

NOTE
When Tera Term is connected, type reboot in the Tera Term console to check early boot messages.

18.2.5 Run DA16600 with JTAG

When the steps in Section 18.2.4 are complete, it is ready to run the example applications. Users can also use JTAG to run DA16600 because the DA16600 has two chips – Wi-Fi (DA16200) and Bluetooth® LE (DA14531) – and each chip has its JTAG port.

18.2.5.1 Run DA16200 with JTAG

The JTAG cable should be connected to JTAG PIN (ID 5 or J7) of the DA16600 EVB. See Section of Debugging with J-Link Debug Probe in Ref. [3] on how to run the JTAG debugging. If you want to boot the DA16600 EVB in "non" JTAG mode again after JTAG is used, then SPI re-programming with two DA16200 images is required. This is because the memory map is different for JTAG boot and normal boot – as DA16200 is using XIP: JTAG writes code in SFlash by its memory map which is not the same as the DA16600's memory map.

18.2.5.2 Run DA14531 with JTAG

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

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

1. Build the DA16600 SDK with `__DA14531_BOOT_FROM_UART__` disabled (see `[DA16600_SDK_ROOT]\apps\da16600\get_started\include\apps\ble_combo_features.h`), and program SFlash with the three DA16200 images (FBOOT and FRTOS). The DA14531 image does not need to be programmed.
2. Set DA16600 EVB's DIP switch configuration (SW4 - P0_2, P0_10 on DA16600 EVB figure in Ref. [3]) as shown in Figure 143.
3. Connect a USB cable to the CN6 USB Port (DA14531 JTAG Port) of the DA16600 EVB. See the Components on DA16600 EVB figure in Ref. [3].
4. Connect a USB cable to the CN1 USB Port (see Figure 2. DA16600 EVB hardware configuration of Ref. [3] of the DA16600 EVB for a Tera Term connection.
5. Switch ON (in a USB hub) the two USB cable connections.
6. Run the Keil IDE and open a DA14531 project.
7. Click the  icon as shown in Figure 125.

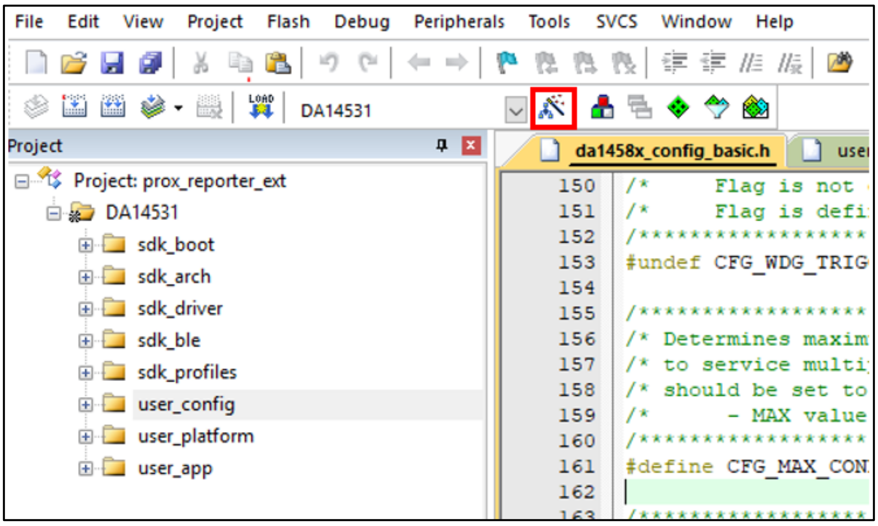


Figure 125. Keil – option

8. Select the **Debug** tab and click **Settings**. See Figure 126.

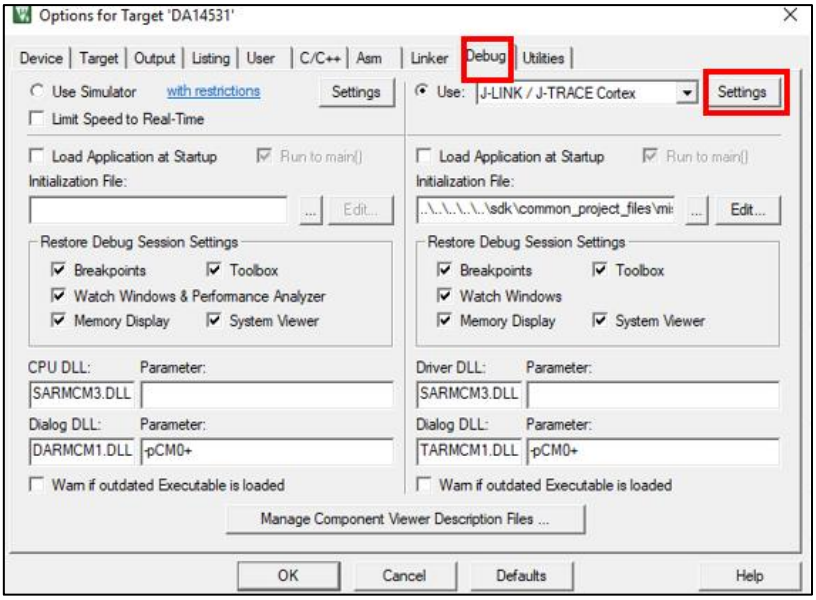


Figure 126. Keil – debug

9. Set **SN** and **SWD** fields with valid values. See Figure 127.

NOTE
If there is invalid value for SN or SWD, the JTAG/J-Link firmware does not exist or is not enabled in the JTAG chip of the DA16600, or the JTAG is not working for some reason. In this case, contact Renesas Electronics to update the JTAG firmware on the DA16600 EVB.

10. If the DA14531 JTAG is successfully recognized, click **OK**.
11. Switch OFF the power to the two USB cables.
12. Switch ON the power to the two USB cables.

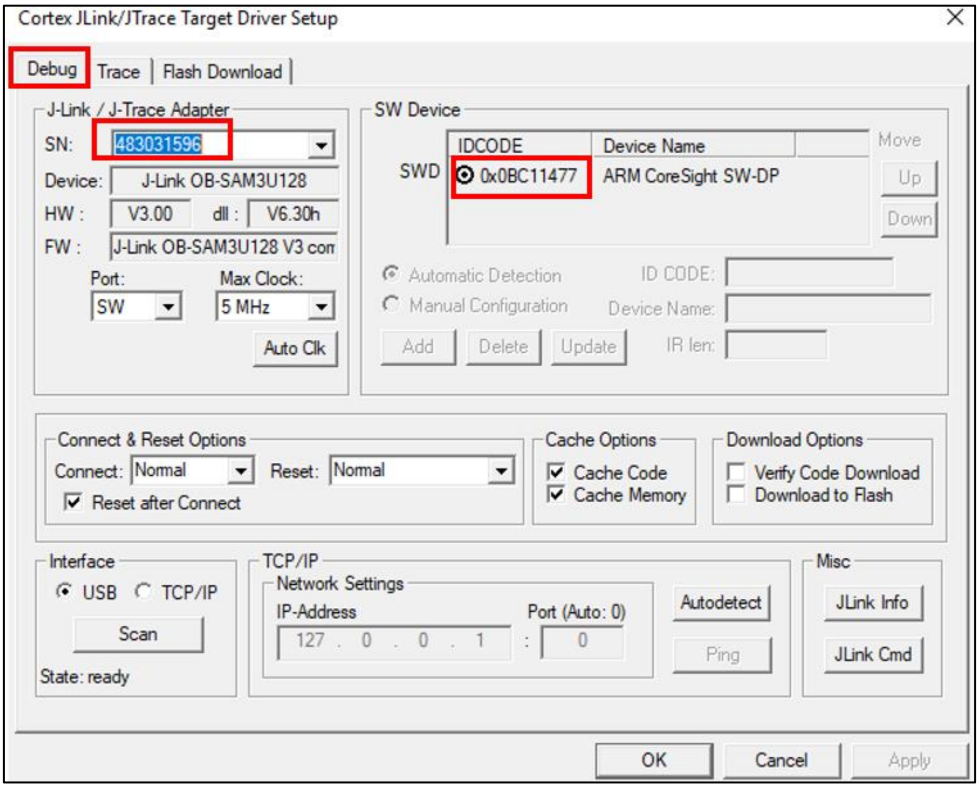


Figure 127. Keil – JTAG device

13. In Tera Term (to which the lower number COM port is connected), make sure that the DA16200 boots successfully. If it succeeds, the following messages are shown as in Figure 128.

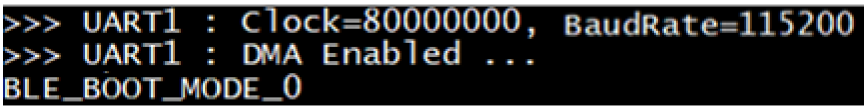


Figure 128. Tera Term – DA16200 waiting for DA14531 to connect

Enable JTAG SWD pins by changing a compiler flag:

```
[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex\include\ext_host_ble_a
ux_task.h
…
#undef __DISABLE_JTAG_SWD_PINS_IN_BLE__
…
```

14. After the build is done, click the **Start Debugger** button. See Figure 129.

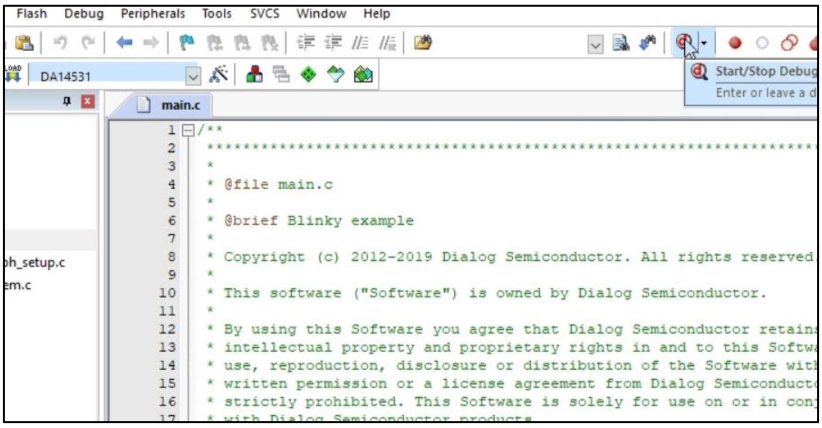


Figure 129. Keil – start debugger

15. Click **OK**.

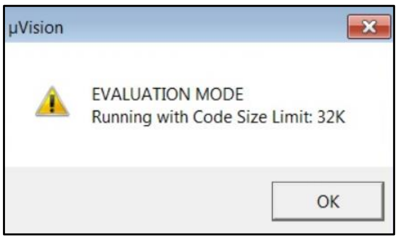


Figure 130. Keil – evaluation mode dialog

16. Click the **Run** button. See Figure 131.

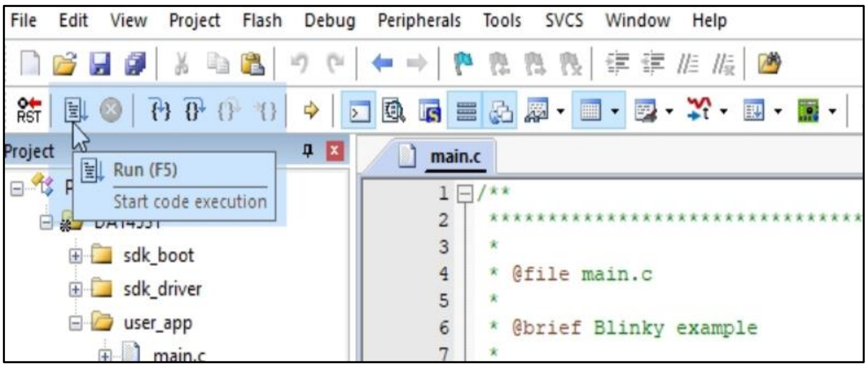


Figure 131. Keil – run

If you see the message as shown in Tera Term, then the DA14531 is successfully started.

```
[combo] BLE FW transfer done
...
<<< GAPM_DEVICE_READY_IND
IoT dev_name="DA16600-327E", len=12
[combo] Advertising...
```

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

18.2.6 Test Environment Setup

The items in the following sections are used in the examples for the test.

18.2.6.1 Wi-Fi Access Point

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

18.2.6.2 Bluetooth® LE Peers

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

18.2.6.2.1 Bluetooth® LE Mobile App

Renesas provides a sample mobile application (Android/IOS App) called "Wi-Fi Provisioning" to test example applications. This mobile application is used to give Wi-Fi provision information (Wi-Fi router connection information plus any customer proprietary information to configure the DA16600) to the DA16600 board.

NOTE
You can also download (from the App Store) and use a general-purpose Bluetooth® LE mobile application that supports a GATT Client (that can read/write a GATT characteristic of a GATT Server). If you are familiar with the Wi-Fi SVC GATT Server database structure and JSON application protocols (see Section 18.3.5), you can use a general Bluetooth® LE Mobile App as well to send a command.

18.2.6.2.2 Bluetooth® LE Sensors

To test the IoT Sensor Gateway Example (Bluetooth® LE Central) application, one or two (up to three) Bluetooth® LE peer devices are required. In these Bluetooth peer devices, it should be implemented for a simple GATT server application, and this is implemented in the DA16600 SDK with the feature (__USER_SENSOR__) to be referred.

For example, there are one or two DA16600 EVBs which run Gas Leak Detection Sensor Example (Bluetooth® LE Peripheral) application, then they can be connected to the IoT Sensor Gateway application.

18.2.6.3 Laptop to Control Bluetooth® LE Peers and DA16600 Boards

1. Use a LAN cable to connect your computer to **MyAP**.
2. Use PuTTy to open two ssh windows (to RBP3_1) – **login as root**.
3. Enter `cd /home/pi` for two ssh windows (let us say **ssh_win_1**).

   Mobile Application (UDP or TCP) can be used instead of the ssh_win_1 accordingly
4. Open one Tera Term window and connect to the COM port (the lower port number of the two, with baud rate 230400) of the DA16600. Let us call this Tera Term window "**da16_tera_win**" from here onwards.
5. Open another Tera Term window and connect to the other COM port (the higher port number of the two, with Baud rate 115200) of the DA16600. Let us call this Tera Term window "**da14_tera_win**" from here onwards (this Tera Term is connected to UART2 of DA14531 chip).

18.3 Wi-Fi Provisioning Over Bluetooth® LE

Each application supports Wi-Fi provisioning and Wi-Fi plays the main role, and Bluetooth® LE assists with "Wi-Fi Provisioning" for the initial setup (Out-of-Box). It allows users to configure Wi-Fi (such as the SSID, password, and server information for user's Wi-Fi router) into the DA16600 module. Figure 132 shows how the devices are connected and worked.

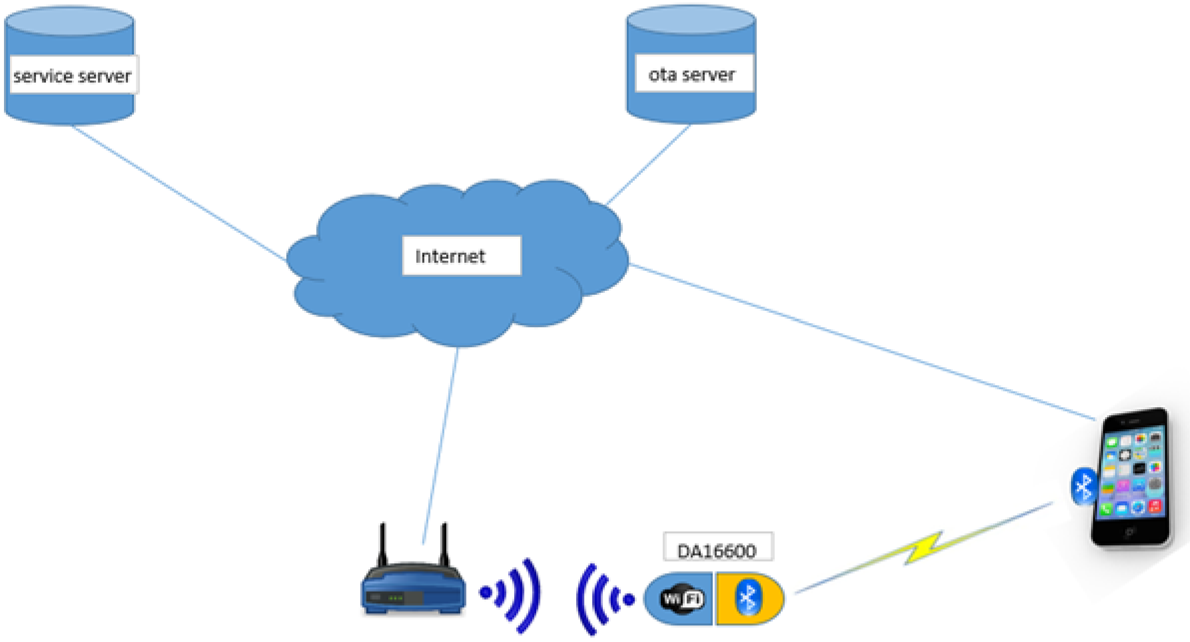


Figure 132. Bluetooth® LE assisted with Wi-Fi provisioning

18.3.1 Description and Requirements

A Bluetooth® LE mobile application is used for the Wi-Fi provisioning. The host Bluetooth application in smart phone can talk to the DA14531 of the DA16600 EVB to start Wi-Fi provisioning.

18.3.2 Test Procedure

After the DA16600 EVB boots up, it starts advertising. Then, the host Bluetooth® LE application in smart phone starts scanning, it is connected by selecting the DA16600 EVB on the Mobile application.

1. Power **ON** DA16600 EVB or else.
   a. Do a Power On Reset (POR) boot: plug out and then plug in the USB cable.
   b. After boot-up, run the command - `factory` to clear any existing NVRAM content. The command - `factory` also triggers a reboot after NVRAM is cleared.
   c. Make sure that "*Advertising …*" is printed on da16_tera_win.

   This should work as described for the Bluetooth peripheral-based examples but in case of the IoT Sensor Gateway Example (Bluetooth Central), it needs to type below command to do the Bluetooth based provisioning. After the command below, the DA14531 role is switched to the peripheral role to start advertising, and it gets back to the central role mode after provisioning is done.

   ```
   [/DA16600] # ble. monitor provision_mode
   ```
2. Run the Wi-Fi provisioning App shown in Figure 133, which is available in the Google Play Store or iOS App Store.
3. Configure Wi-Fi environment as shown in Ref. [4].
4. Steps: Start DA16600-based > Start > Select 'DA16600-XXXX' > Wait for some seconds > Scan Wi-Fi network > select [MyAP name] (input Password if need) > Connect to [MyAP name], then the provisioning information is transferred to DA16600, which saves the information into NVRAM and rebooted. After rebooting, Wi-Fi is connected to the selected AP.

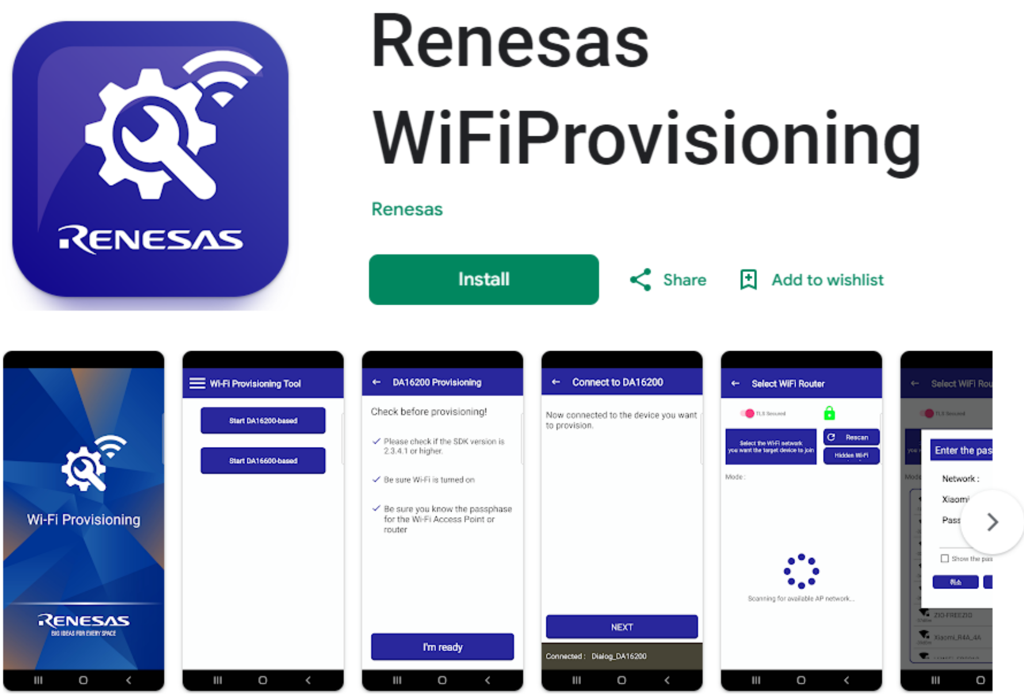


Figure 133. Renesas Wi-Fi provisioning app

To remove the provisioning information,

1. Establish a Bluetooth® LE connection again with DA16600 EVB.
2. Click the Reset the device button.

   Now you can start provisioning again.

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

18.3.3 GTL Workflow

1. Initialization until advertising.

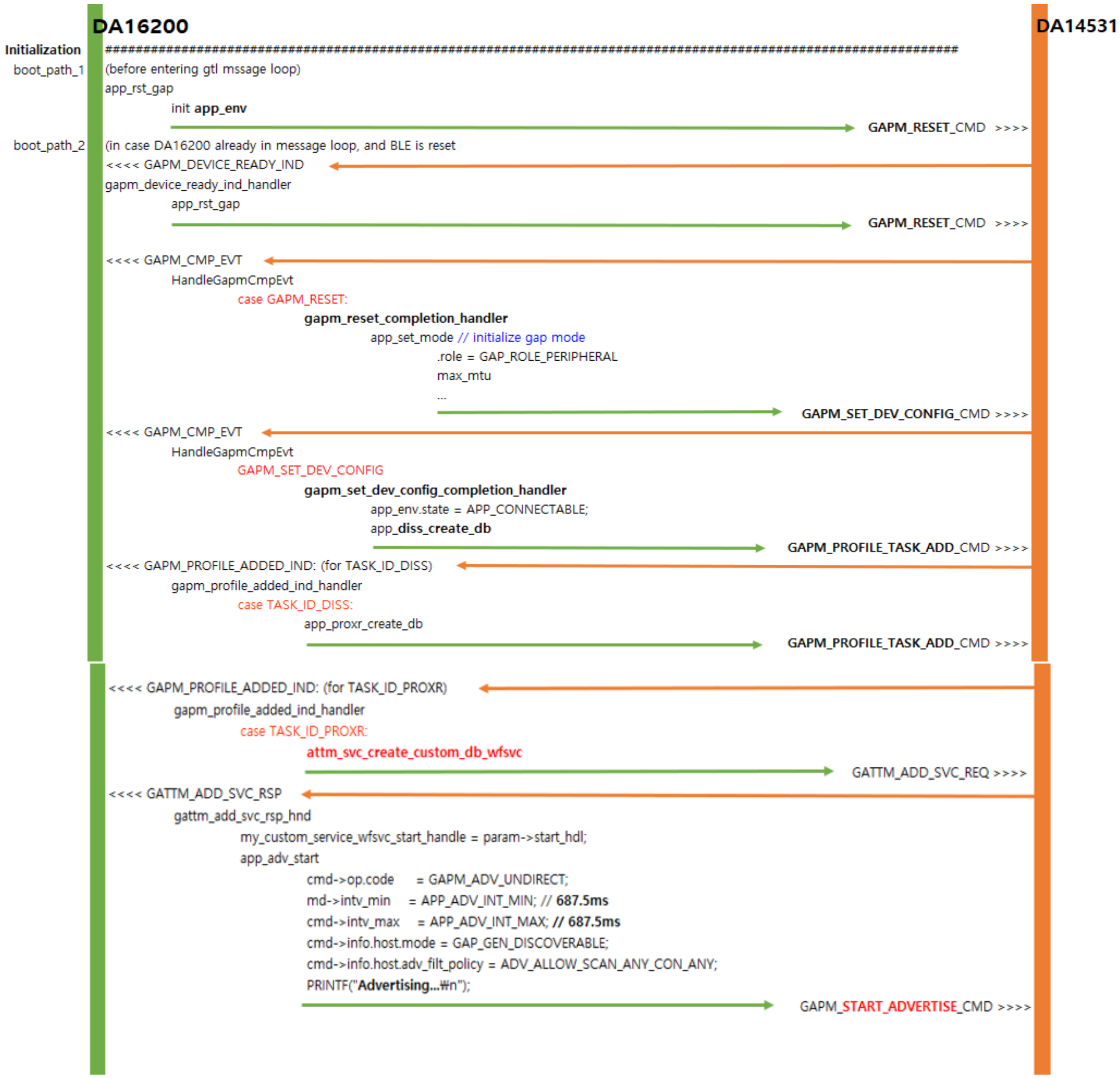


Figure 134. GTL message sequence chart – initialization

2. Connection Request and Characteristic "Write" Request.

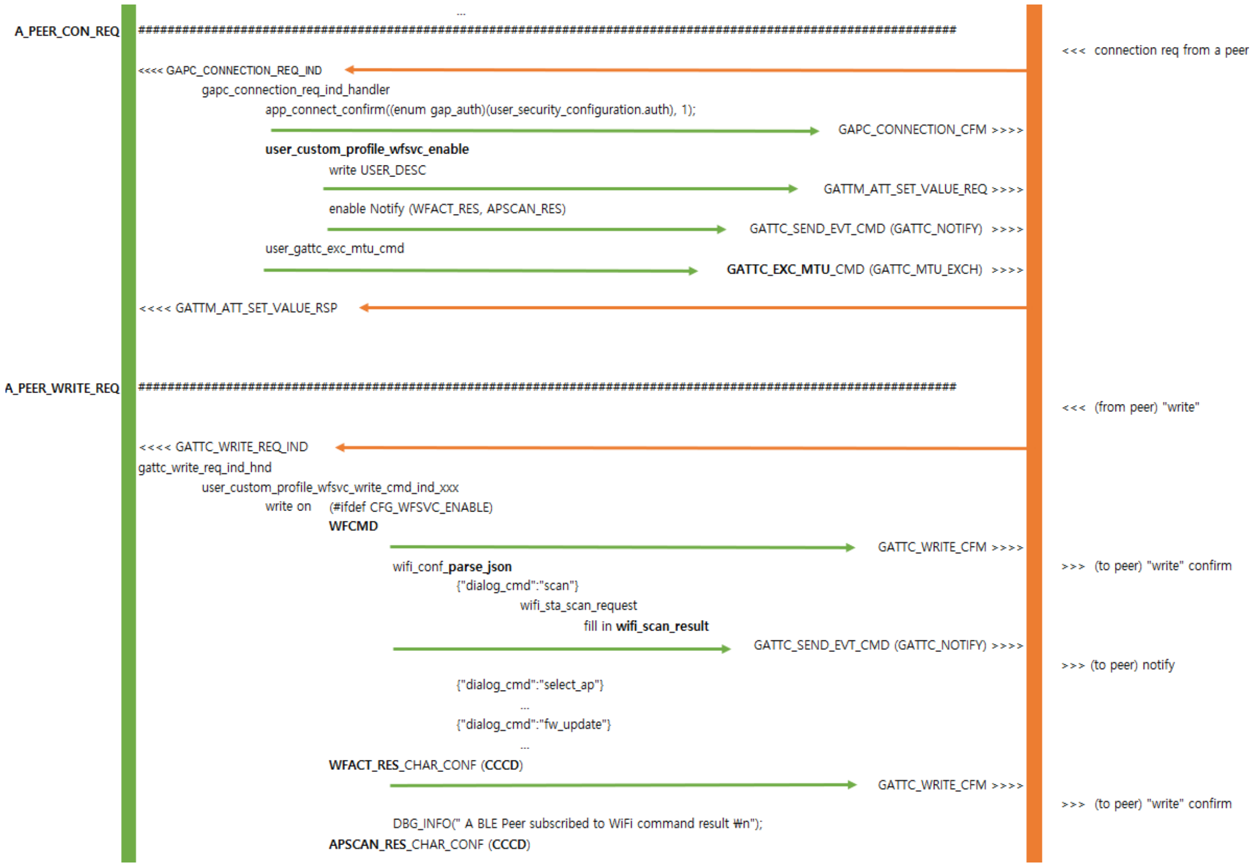


Figure 135. GTL message sequence chart – connect and write

3. Characteristic "Read" request.

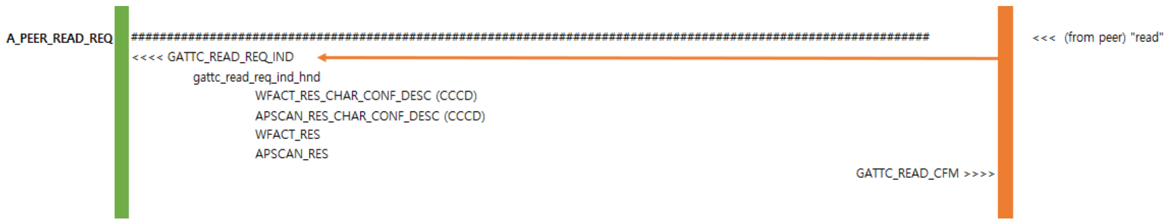


Figure 136. GTL message sequence chart – read

18.3.4 Wi-Fi Service GATT Database Design

The Wi-Fi Service sample – GATT Server database is added as a reference in the application source. It may need to modify the database or create a different one. See the `*_user_custom_profile.c/h (for example, wifi_svc_user_custom_profile.c/h)` file and `gattm_svc_desc_wfsvc` variable for more details.

18.3.5 Wi-Fi Service Application Protocol

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: scan Wi-Fi routers, (Mobile host application -> DA16600)

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

- "network_info" command: provide the network information necessary during the provisioning. (Mobile host application -> DA16600)

```
{
"dialog_cmd":"network_info",
"ping_addr":"8.8.8.8",
"svr_addr":"172.16.0.100",
"svr_port":10195,
"svr_url":"www.google.com"
}
```

- "select_ap" command: select the AP in the AP list received by the scan command. (Mobile host application DA16600), The DA16600 device tries to connect to the selected AP with the information after the command, upon receipt of this command, the DA16600 stores the credentials in permanent storage (NVRAM) and reboots

```
{
    "dialog_cmd":"select_ap",
    "SSID":"linksys",
    "security_type":3,
    "password":"123456789",
"isHidden":0
}
```

- "fw_update" command: download new firmware from a specified OTA server, (Mobile host application -> DA16600)

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

- "factory_reset" command: remove the Wi-Fi network profile saved in the DA16600 EVB, the EVB reboots after the reset. (Mobile host application -> DA16600)

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

- "reboot" command: reboot the DA16600 device. DA16600 tries to connect to the selected AP after rebooted if the provisioning is completed before (Mobile host application -> DA16600)

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

- "wifi_status" command: check the Wi-Fi connection status (connected or disconnected). The Bluetooth® LE peer can be notified of or read, the status via the "Wi-Fi Action Result" characteristic (DA16600 -> Mobile host application)

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

- "disconnect" command: disconnect a Wi-Fi connection from the connected AP. If the user sends the command "reboot", then it can reconnect to the connected AP before. (Mobile host application -> DA16600)

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

- If you want to add a new custom command, use the following:

  `enum WIFI_CMD` > define a new custom command

  > `user_custom_profile_wfsvc_write_cmd_ind_xxx( ):` add the handler of the command

  > `wifi_conf_parse_json` : add the parser of the command

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

```
// Wi-Fi Action Result
enum WIFI_ACTION_RESULT {
      COMBO_WIFI_CMD_SCAN_AP_SUCCESS = 1,
      COMBO_WIFI_CMD_SCAN_AP_FAIL,
      COMBO_WIFI_CMD_FW_BLE_DOWNLOAD_SUCCESS,
      COMBO_WIFI_CMD_FW_BLE_DOWNLOAD_FAIL,
      COMBO_WIFI_CMD_INQ_WIFI_STATUS_CONNECTED,
      COMBO_WIFI_CMD_INQ_WIFI_STATUS_NOT_CONNECTED,
      COMBO_WIFI_PROV_DATA_VALIDITY_CHK_ERR,
      COMBO_WIFI_PROV_DATA_SAVE_SUCCESS,
      COMBO_WIFI_CMD_MEM_ALLOC_FAIL,
      COMBO_WIFI_CMD_UNKNOWN_RCV
};
```

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

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

18.4 Bluetooth® LE Firmware OTA Download Through Wi-Fi

When the Wi-Fi has been successfully provisioned and the DA16600 device can receive notifications over Wi-Fi from a remote service server indicating that there is new Wi-Fi or Bluetooth® LE firmware available for the DA16600. Upon receipt of the notification, the DA16600 can securely download the new firmware from an OTA server through Wi-Fi, store the firmware in its flash memory and then trigger the firmware update. Figure 132 shows how the devices are connected and worked.

18.4.1 Description and Requirements

OTA firmware download service (Wi-Fi download of firmware) is supported and enabled in all examples, and AP and HTTP(s) server are required to set up and the mobile APP is used to trigger the OTA download as well.

18.4.2 Test Procedure

For this test, make sure that an HTTP(s) server is running on the MyAP network and complete the following steps.

1. Install HTTP server on a personal computer with Apache as the web server that is connected to MyAP. Go to apache.org to download Apache and for instructions.
2. Increase the version number (2 > 3) of `.\binaries\da14531\mkimage\app_version.h` as follows and build the project. This is to get a different image version to compare the previous image.
   - `#define SDK_VERSION "6.0.14.1114.3"`
3. Copy file **px*_coex_ext_531_6_0_14_1_ota.img** (described in Section 18.2.4.1) to the **htdocs** folder of the Apache server (or any http server that you want to use).
4. Make sure that your computer is connected to MyAP and the **px*_coex_ext_531_6_0_14_1_ota.img** file is available through a web browser with the link. The IP address is just an example: http://192.168.0.230/pxr_sr_coex_ext_531_6_0_14_1114_1_ota.img.
5. See the following console commands.

```
[/DA16600] # nvram
[/DA16600/NVRAM] setenv URI_BLE
http://192.168.0.230/pxr_sr_coex_ext_531_6_0_14_1114_1_ota.img
[/DA16600/NVRAM] reboot
...
BLE FW VER to transfer ....
>>> v_6.0.14.1114.2 (id=1) at bank_1 // check current version and bank number.

[/DA16600] # nvram
[/DA16600/NVRAM] # getenv
...
URI_BLE (STR,53) ......... http://192.168.0.230/pxr_sr_coex_ext_531_6_0_14_1114_1_ota.img
```

6. In the Bluetooth® LE Mobile App (for example, mobile phone), send the firmware update command.

Figure 137. Provisioning application – custom command

   a. Bluetooth® LE Mobile Application > Start DA16600-based > **Start** > Connect to "DA16600-BLE" > **Custom command** (see Figure 137), type the command **{"dialog_cmd":"fw_update"**} and click **Send**.
   b. To enter the command easily, open a notepad on the smartphone to type the command, and then copy and paste the command on the Bluetooth® LE Mobile Application.

7. When the command "fw_update" is reached to the DA16600, it tries to connect to an OTA server (**192.168.0.230**) to download a Bluetooth® LE firmware file. This log shows some details for the steps.

```
[/DA16600/NVRAM] #
<<< GAPC_CONNECTION_REQ_IND
...
<<< GATTC_WRITE_REQ_IND
Receive - FW_UPDATE
 COMBO_WIFI_CMD_FW_BLE_DOWNLOAD received
[ota_fw_update_combo] uri_rtos =
[ota_fw_update_combo] uri_ble = http://192.168.0.230/pxr_sr_coex_ext_531_6_0_14_1114_1_ota.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....
...
Wake-up source is 0x00 // rebooted ...
...
BLE FW VER to transfer ....
   >>> v_6.0.14.1114.3 (id=2) at bank_2 // new FW boots from bank_2 (< bank_1)
...
```

18.4.3 Working Flow

There are two ways to trigger an OTA firmware download service:

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

NOTE
Option 1 can be used for unattended/automatic OTA operation. It works as follows: As a DA16600 device (DA16200 Wi-Fi chip included) is 'always' in the connected state with a Wi-Fi router connected to the Internet, a Service Server/Cloud Server can talk with this DA16600 device when there is new firmware available on an OTA Server. A Service Server can contact a user through 4G/Wi-Fi (in the form of a push message) for firmware update confirmation. Upon receipt of 'user confirmation', the Service Server may ask the DA16600 device to download new firmware by giving an HTTP(s) URI to an OTA Server. When the new firmware is received, that firmware is stored in SFlash connected to the DA16600 device, and the DA16600 device restarts to boot with the new software.

For Option 2, a Bluetooth® LE Peer App should 'write' the following command on the characteristic "Wi-Fi CMD" to trigger an OTA firmware update.

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

Upon receipt of the command, `GATTC_WRITE_REQ_IND` is sent to DA14531 and DA16200, and then a handler is invoked. See the following steps.

- Host Bluetooth® LE application (Mobile phone) 'fw_update' command
  > DA16600 receives the command and calls the following functions
  > `HandleBleMsg (bType = GATTC_WRITE_REQ_IND)`
  > `gattc_write_req_ind_hnd()`
  > `user_custom_profile_wfsvc_write_cmd_ind_xxx()`
  > `wifi_conf_parse_json()`
  > `ota_fw_update_combo()`

> `ota_update_start_download()`

> `ota_update_http_client_update_proc()`: handles http connection, http download, and firmware renewing process

18.5 Gas Leak Detection Sensor Example (Bluetooth® LE Peripheral)

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

A virtual gas density check sensor is attached to the Bluetooth wireless interface of the DA16600. The DA16600 operates in low-power mode and periodically checks the gas density level at a time interval defined by the user. When a certain gas density level value is reached, the Wi-Fi device is activated, and a "gas leak" event is created and sent to the Wi-Fi device. The Wi-Fi device then posts the "gas leak" event to a cloud server where a user is notified, and action can be taken. Figure 138 show this example works.

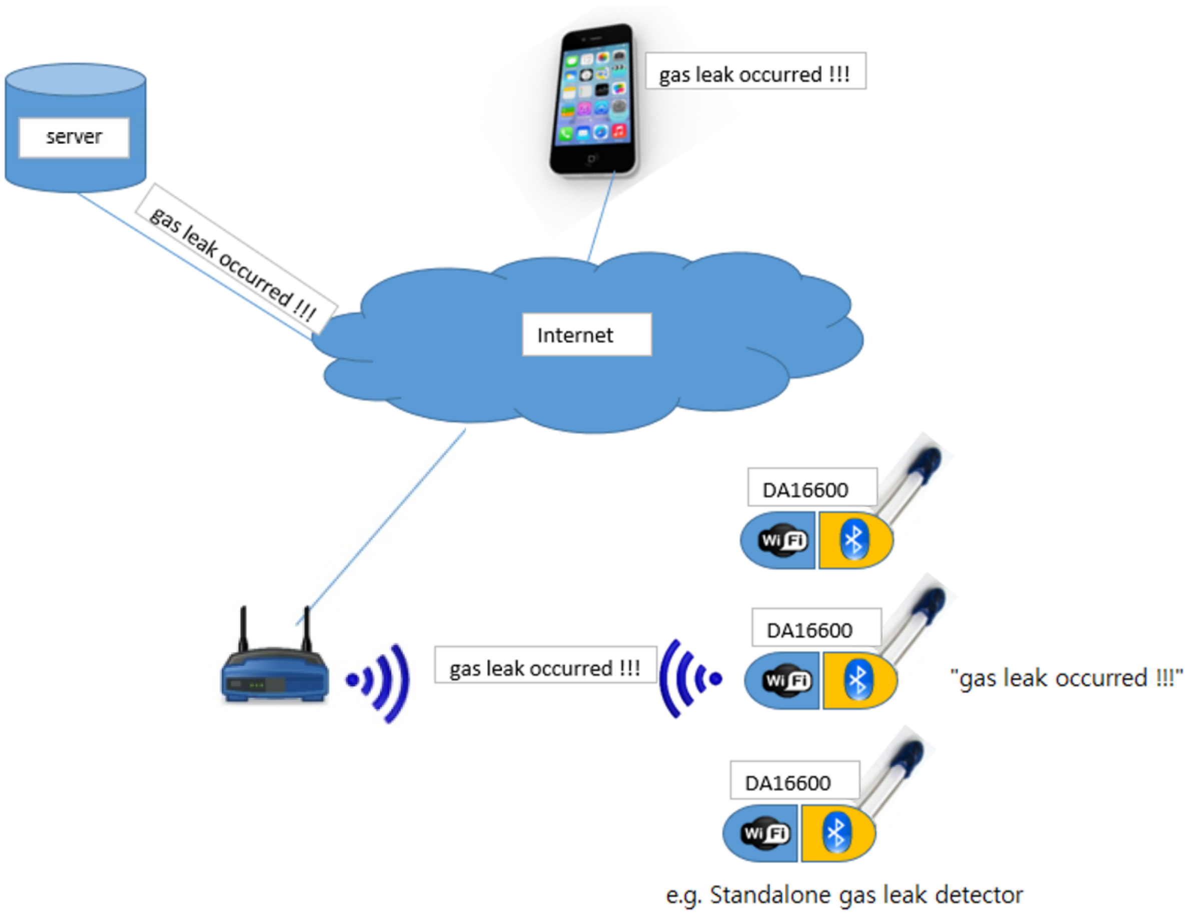


Figure 138. Standalone gas leak detection sensor

18.5.1 Description and Requirements

To build and run for this application, see Sections 18.2.2 and 18.2.2.1. The DA16200 sends the command to DA14531 to get the gas leak sensor started in DA14531. When a Gas leak is detected, the DA16200 is woken up by the DA14531 and receives the event and then sends the information to the UDP server. Sleep mode 2 is used in this application.

NOTE
Connection with a Bluetooth® LE Peer is not required in the Gas Leak Detection Sensor application.

18.5.2 Test Procedure

1. ssh_win_1: type the following command to start the UDP server (for example, Raspberry-pi or can be set up on android/iOS phone).

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

On the console, the provisioning command JSON string "network_info" of the Provisioning App (see Section 18.3.5) should have the following data.

a. "svr_addr": "172.16.30.136"

b. "svr_port": 10954

2. Type dpm on command after provisioning, the device is rebooted. Then type the bold font (command) in the log box below. The DA16600 (Wi-Fi) goes to sleep (while in sleep, keyboard input is not working, wait for some minutes).

```
[/DA16600] # ble
[/DA16600/ble] # iot_sensor start
...

sleep (rtm ON) entered
...
```

3. After the DA16600 wakes up and posts a message to a server, and then it goes to sleep mode again.

```
...
>>> [msg_sent] : gas_leak occurred, plz fix it !!!
sleep (rtm ON) entered
```

4. On the server, the following message is shown (for example, ssh_win_1).

```
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!!!
```

Then, the following occur:

- If you run the command `iot_sensor start`, the command is sent over (through 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 the DA16200 and sends the event ("gas leak occurred!") to the 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.

18.5.3 Workflow

The gas leak detection example starts by typing the command – **ble.iot_sensor start** in the console. The following function is invoked after the command.

- `[/DA16600] # ble.iot_sensor start` >
  - > `ConsoleSendSensorStart()`
  - > `ConsoleEvent_handler(CONSOLE_IOT_SENSOR_START` or `STOP)`
  - > `app_sensor_start()` or `app_sensor_stop()`
  - > `BleMsgAlloc(APP_GAS_LEAK_SENSOR_START, TASK_ID_EXT_HOST_BLE_AUX, TASK_ID_GTL, 0);`
- The message " `APP_GAS_LEAK_SENSOR_START`" is sent to Bluetooth® LE (DA14531) which starts the sensor reading task (periodically reads a gas-leak sensor)

In the DA14531, to exchange messages or commands to an external host (DA16200), the following code should be implemented on both Wi-Fi and Bluetooth® LE SDKs.

- For both DA16200 SDK and DA14531 SDK, define custom messages:
  - **DA16600 SDK:** send a custom user-defined message through the GTL interface with `TASK_ID_EXT_HOST_BLE_AUX` as the destination task.
  - **DA14531 SDK:** enable the DA14531 AUX task (`TASK_ID_EXT_HOST_BLE_AUX`) to receive user-defined custom messages.

- The same `ext_host_ble_aux_task_msg_t` should be defined on both the `app.h` (in DA16600 SDK) and the `ext_host_ble_aux_task.h` (in DA14531 SDK).

```
typedef enum {
...
    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;
```

- Regarding the message handlers in the DA14531 SDK/`ext_host_ble_aux_task.c`,
  - DA16600/`BleMsgAlloc(APP_GAS_LEAK_SENSOR_START)`
    > DA14531/`ext_host_ble_aux_task_handler (APP_GAS_LEAK_SENSOR_START)`
    > DA14531/`app_gas_leak_sensor_start_cfm_send` with `APP_GAS_LEAK_SENSOR_START_CFM`
    > DA16200

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

- `system_start()` > `sleep2_monitor_start()`
- `gtl_init()` > `sleep2_monitor_regi()`
- `app_sensor_event_ind_hnd()`
  > `sleep2_monitor_set_state(SLEEP2_CLR)`
  > set `iot_sensor_data_info.is_gas_leak_happened = TRUE`
- `udp_client()`: send the warning message to server when gas leak occurred and tell sleep2_monitor to go into sleep

18.6 TCP Client in DPM Example (Bluetooth® LE Peripheral)

This example demonstrates how the DA16600 module runs a TCP client in a low-power mode where the DA16200 stays in DPM mode and the DA14531 stays in Extended Sleep mode. The DA16600 module wakes up from low-power or Sleep mode, then receives and processes a Wi-Fi packet from a network peer or Bluetooth® LE data from a Bluetooth peer. After either a Wi-Fi packet or Bluetooth data has been handled, DA16600 enters Sleep mode again to save power.

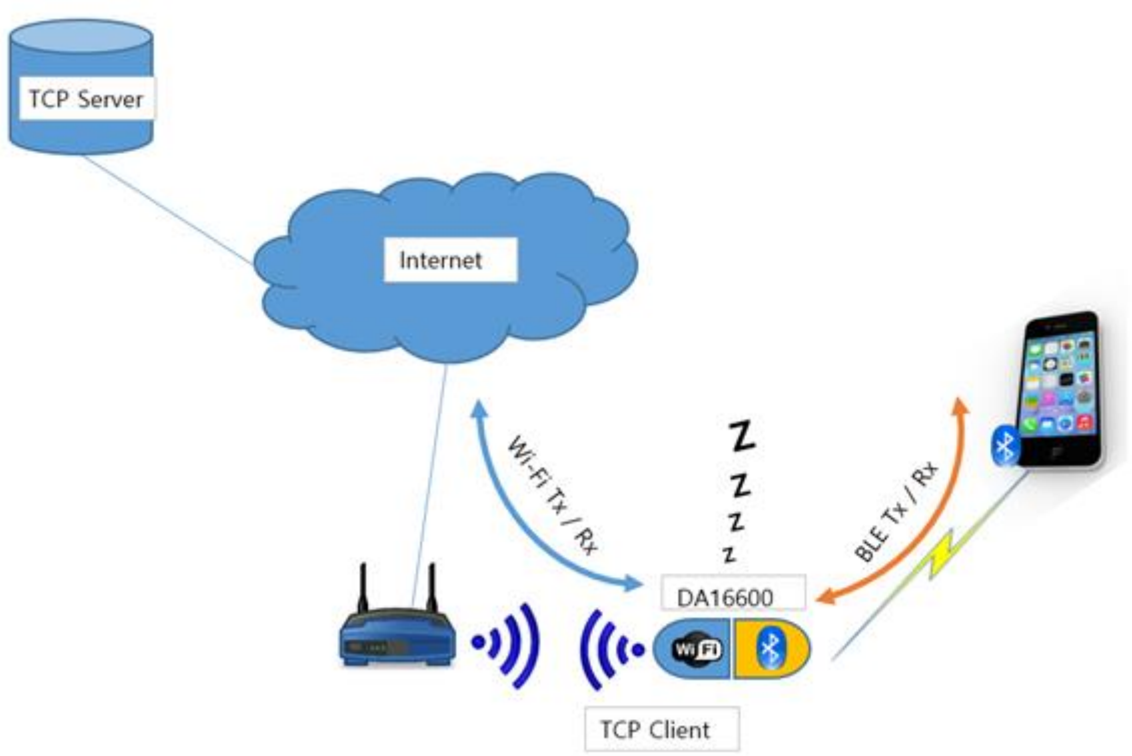


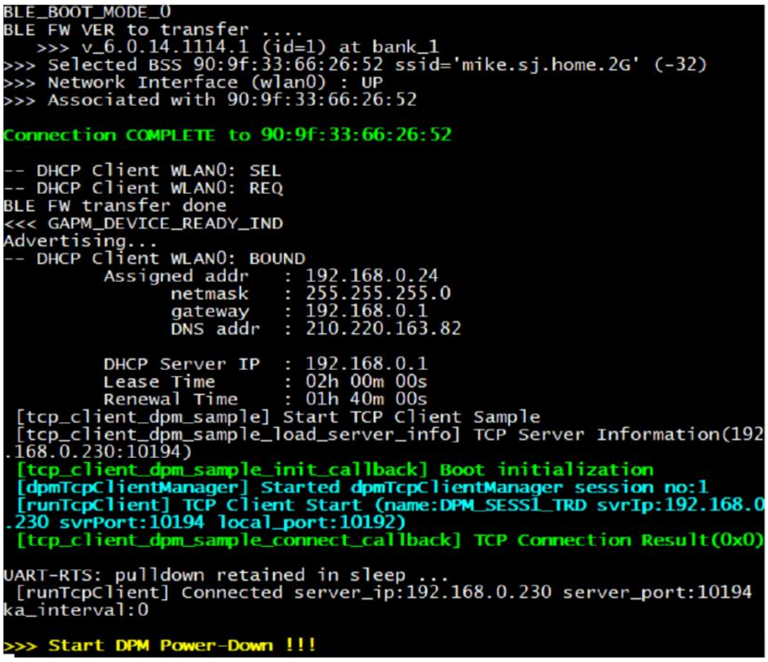
Figure 139. DA16600 TCP client in DPM

18.6.1 Description and Requirements

To build and run for this application, see Sections 18.2.2 and 18.2.2.2. In this example, the DA16600 receives TCP packets while in DPM mode, and AP and a TCP server utility are required as well. Sleep mode 3 is used in this application.

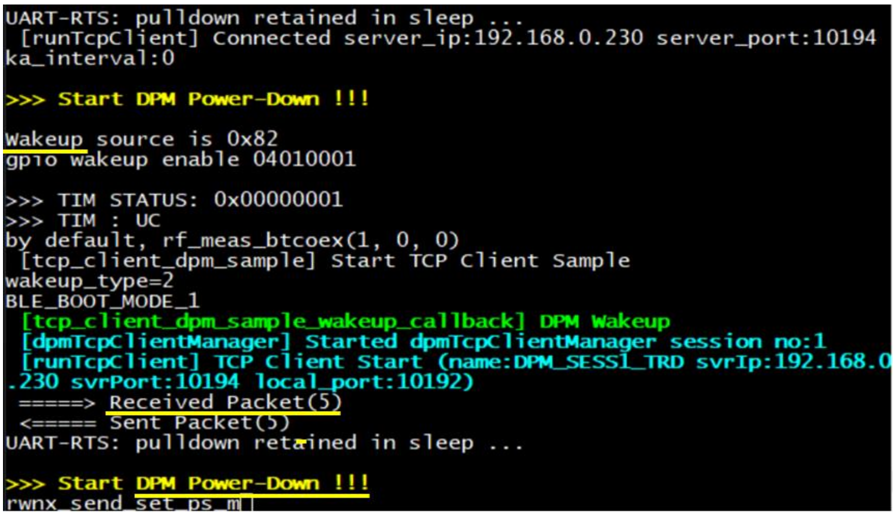
18.6.2 Test Procedure

1. Wi-Fi Router: MyAP.
2. TCP Client: DA16600 EVB.
3. Two Tera Term windows: da16_tera_win, and da14_tera_win.
4. TCP Server: any TCP Server utilities are OK such as IONINJA, or Android/IOS TCP network tool.
5. TCP Server machine (Windows utility/mobile application).
   a. Connect to MyAP (either through a wired port or Wi-Fi port – wired connection preferred).
   b. Run TCP Server tool (with the port number set to 10194).
   c. Take note of TCP Server information: IP = 192.168.0.230, Port = 10194.
6. TCP Client.
   a. da16_tera_win
   b. type `factory` > type `y`
   c. Run Wi-Fi Provisioning to connect to MyAP. See Section 18.3.2.
   d. Type this command to set the server information in NVRAM.
      ```
      nvram.setenv TCPC_SERVER_IP 192.168.0.230
      nvram.setenv TCPC_SERVER_PORT 10194
      ```
   e. Type `dpm on`.
   f. DA16600 EVB is rebooted and enters DPM Sleep as in Figure 140
7. TCP Server Tool: Send a text to TCP Client.
8. TCP Client – TCP Client wakes up, receives, processes data, and enters sleep as shown in Figure 141.



```
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 140. TCP client in DPM sleep



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

>>> Start DPM Power-Down !!!

Wakeup source is 0x82
gpio wakeup enable 04010001

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

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

Figure 141. TCP client – wake up from DPM sleep

18.6.3 Workflow

TCP Client thread which uses DPM manager – tcp_client_dpm_sample is run when network is alive. User callbacks for TCP events are registered in `tcp_client_dpm_sample_init_user_config()` including `tcp_client_dpm_sample_recv_callback()`. In the receive callback, when a user receives and processes data, then calls the `dpm_mng_job_done()`.

`tcp_client_dpm_sample()`

> `tcp_client_dpm_sample_init_user_config()`: callback functions are registered

> `tcp_client_dpm_sample_recv_callback()`: called when received the packet, process the data

> `dpm_mng_job_done()`

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

- `gtl_init()` > `dpm_app_register()`: register to DPM sub-system
- `gapc_connection_req_ind_handler()`

> `dpm_app_sleep_ready_clear()`: if a peer is connected (until disconnected), tell DPM sub-system to hold going into sleep

> `dpm_abnormal_chk_hold()`: told 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

- `gapc_disconnect_ind_handler()`

> `dpm_app_sleep_ready_set()`: tell DPM sub-system to enter sleep as the peer is disconnected

> `dpm_abnormal_chk_resume()`: tell DPM Abnormal Checker to resume its work

18.7 DA14531 Peripheral Driver Example (Bluetooth® LE Peripheral)

This example shows the way to control the peripherals in the DA14531 devices by DA16200, the peripherals in DA14531 can be configured and used as the GPIO, I2C, SPI, and PWM.

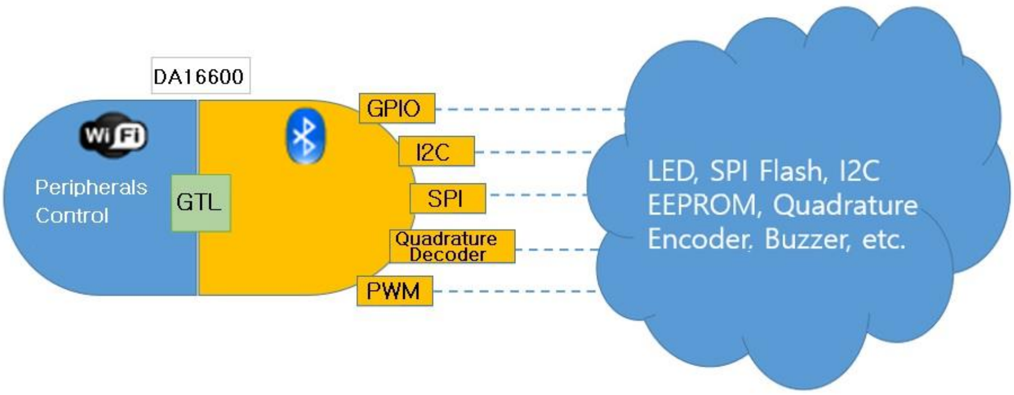


Figure 142. DA14531 peripheral device control

18.7.1 Description and Requirements

To build and run for this application, see Sections 18.2.2 and 18.2.2.3. For this example, some proper components or connections are required for each test, and the DA14531 GPIO pins can be controlled by the DA16200 in the DA16600.

18.7.2 Test Environment Setup

18.7.2.1 DA16600 EVB Setup

See the EVB configuration of Ref. [3] for the components such as SW4, SW5, SW6, SW8, J2, J3, and GPIO pins. To test this example, use the following configurations.

18.7.2.1.1 Configuration_1

GPIO Pins	
P0_2	GPIO
P0_8	GPIO
P0_9	UART2_RX
P0_10	GPIO
P0_11	GPIO

SW4	1-8	OFF	P0_8 to USBA/Rx
	2-7	ON	P0_9 to USBA/Tx
	3-6	ON	P0_2 to SWCLK
	4-5	ON	P0_10 to SWDIO
SW5	1-4	OFF	GPIOC-6 to USBA/Rx
	2-3	OFF	GPIOC-7 to USBA/Tx
SW6 (1wire UART)	1-4	OFF	P0_5 to USBA/Tx
	2-3	OFF	P0_5 to USBA/Rx
SW8	1-4	ON	GPIOA6 to WPS Button
	2-3	ON	GPIOA7 to Factory Reset Button

Figure 143. DA16600 EVB SW config. 1

18.7.2.1.2 Configuration_2

GPIO Pins	
P0_2	GPIO
P0_8	GPIO
P0_9	UART2_RX
P0_10	GPIO
P0_11	GPIO

Switch	Pin	State	Connection
SW4	1-8	OFF	P0_8 to USBA/Rx
	2-7	ON	P0_9 to USBA/Tx
	3-6	OFF	P0_2 to SWCLK
	4-5	OFF	P0_10 to SWDIO
SW5	1-4	OFF	GPIOC-6 to USBA/Rx
	2-3	OFF	GPIOC-7 to USBA/Tx
SW6 (1wire UART)	1-4	OFF	P0_5 to USBA/Tx
	2-3	OFF	P0_5 to USBA/Rx
SW8	1-4	ON	GPIOA6 to WPS Button
	2-3	ON	GPIOA7 to Factory Reset Button

Figure 144. DA16600 EVB SW config. 2

18.7.2.2 Tera Term Setup

Two Tera Term windows are required.

- Teraterm_1 (da16_tera_win): connect to COMxx (lower one) with 230400 as baud rate. This is the debug console of the DA16200 where the user command is entered
- Teraterm_2 (da14_tera_win): connect to COMxx (higher one) with 115200 as baud rate. This is the debug console of the DA14531. Test progress is printed

18.7.2.3 DA14531 Peripheral Driver Samples

Ten peripheral samples are described in this section. The list of the DA14531 Peripheral Driver samples follows the commands bolded.

```
[/DA16600] # ble
[/DA16600/ble] # peri
--------------------------------------------------
peri : Run DA14531 Peripheral Driver Sample
       type a command below
--------------------------------------------------
[01] peri blinky     : blinking LED sample
[02] peri systick    : systick timer sample
[03] peri timer0_gen : timer0 general sample
[04] peri timer0_buz : timer0 PWM buzzer sample
[05] peri timer2_pwm : timer2 PWM LED array sample
[06] peri batt_lvl   : battery level read sample
[07] peri i2c_eeprom : I2C EEPROM read/write sample
[08] peri spi_flash  : SPI_flash read/write sample
[09] peri gpio       : GPIO contorl(High/Low)
```

18.7.3 Test Procedure

18.7.3.1 peri blinky

One GPIO is used to blink the LED connected to the GPIO.

1. DA16600 EVB Configuration (See Figure 143).

   By default, P0_8 is used to connect to LED. Connect J3:P0_8 to any pins in P10 or P11 (#17 or #18 of DA16600 EVB Hardware Configuration figure in Ref. [3]).
2. Run command as in Figure 145. The LED that is connected to the GPIO specified blinks.

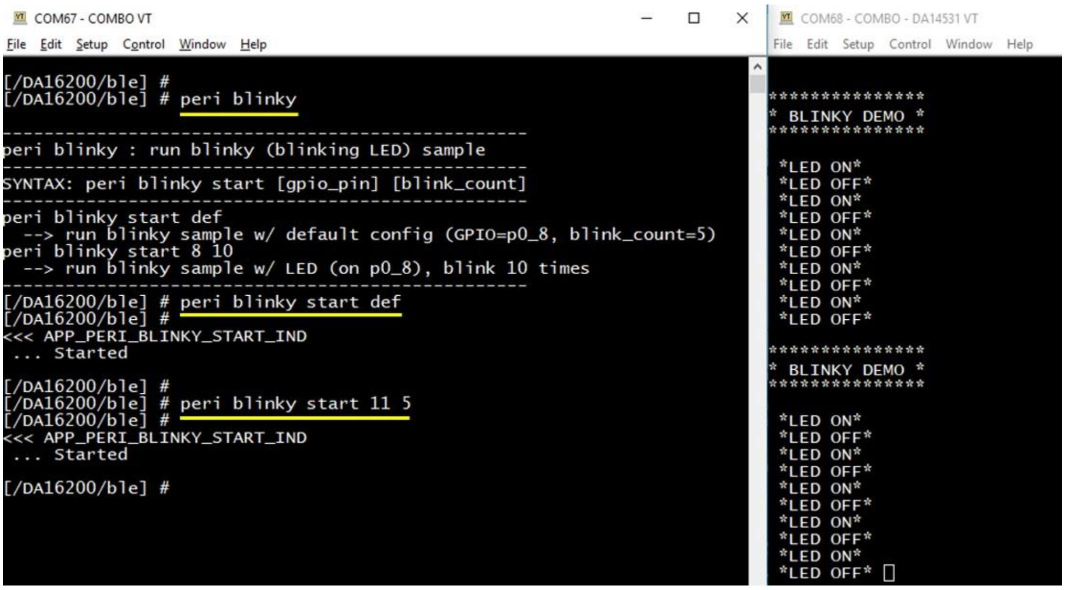


Figure 145. Peri blinky

18.7.3.2 peri systick

The systick timer of the DA14531 is used in this sample.

1. This sample uses 1 GPIO to change the state of the LED that is connected to the GPIO.
2. DA16600 EVB Configuration (See Figure 143).

   By default, P0_8 is used to connect to LED. Connect J3:P0_8 to any pins in P10 or P11 (#17 or #18 of DA16600 EVB Hardware Configuration figure in Ref. [3]).
3. Run command as in Figure 146. Whenever the systick timer expires, it toggles the LED state.

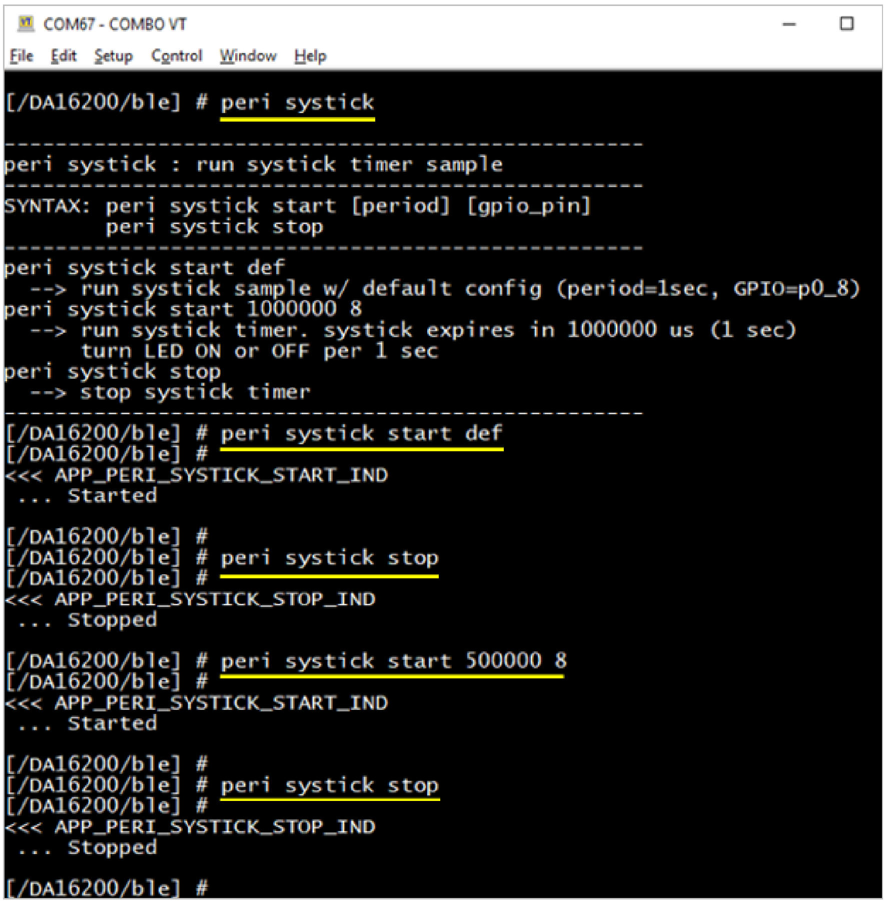


Figure 146. Peri systick

18.7.3.3 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.

1. Use one GPIO connected to an LED to show how TIMER0 can be used.
2. DA16600 EVB Configuration (See Figure 143).

   By default, P0_8 is used to connect to LED. Connect J3:P0_8 to any pins in P10 or P11 (#17 or #18 of DA16600 EVB Hardware Configuration figure in Ref. [3]).
3. Run command as in Figure 147 and check LED.

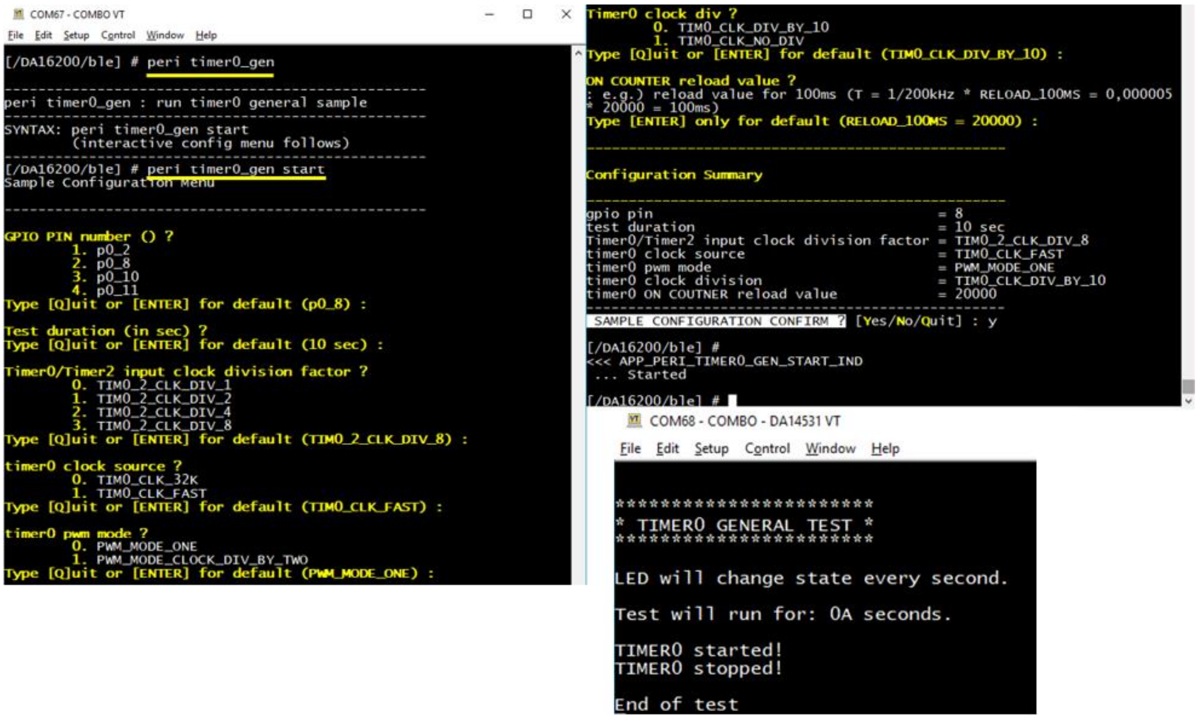


Figure 147. Peri Timer0_gen

18.7.3.4 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.

1. Use two GPIOs connected to an external buzzer.

   DA16600 EVB Configuration: By default, P0_8 and P0_11 are used to connect to a buzzer. Connect J3:P0_8 and J2:P0_11 to a buzzer. See Ref. [3] for J2 and J3.

2. Run commands as shown in Figure 148 and Figure 149.

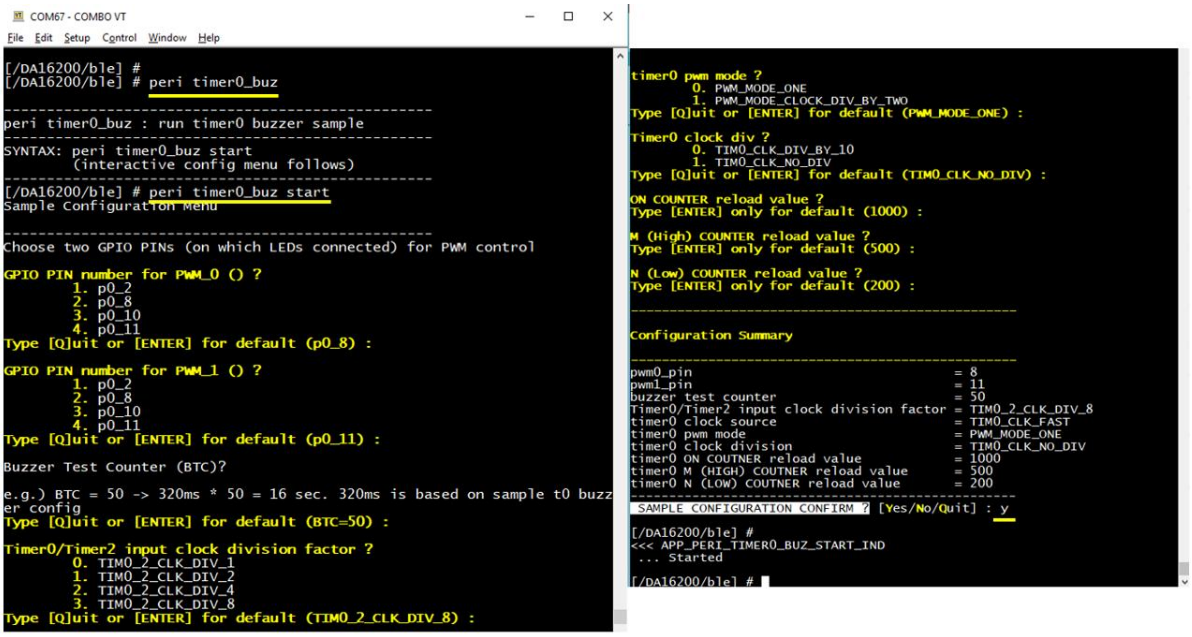


Figure 148. Peri Timer0_buz

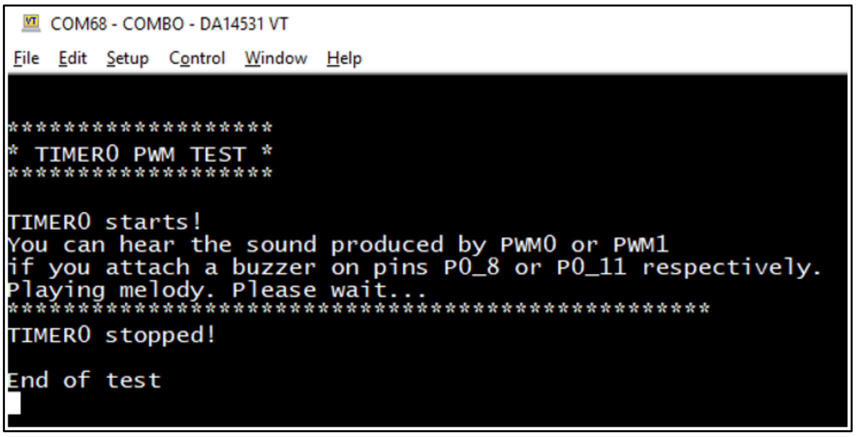


Figure 149. Peri Timer0_buz (Continued)

18.7.3.5 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.

1. Use three GPIOs connected to an LED to show how TIMER2 PWM can be used.
2. DA16600 EVB Configuration (See Figure 143).

   By default, P0_8, P0_11, and P0_2 are used to connect to an LED.

   Connect J3:P0_8 and J2:P0_11, and J3:P0_2 to an LED. For J2 and J3, see Ref. [3].
3. Run command as in Figure 150.

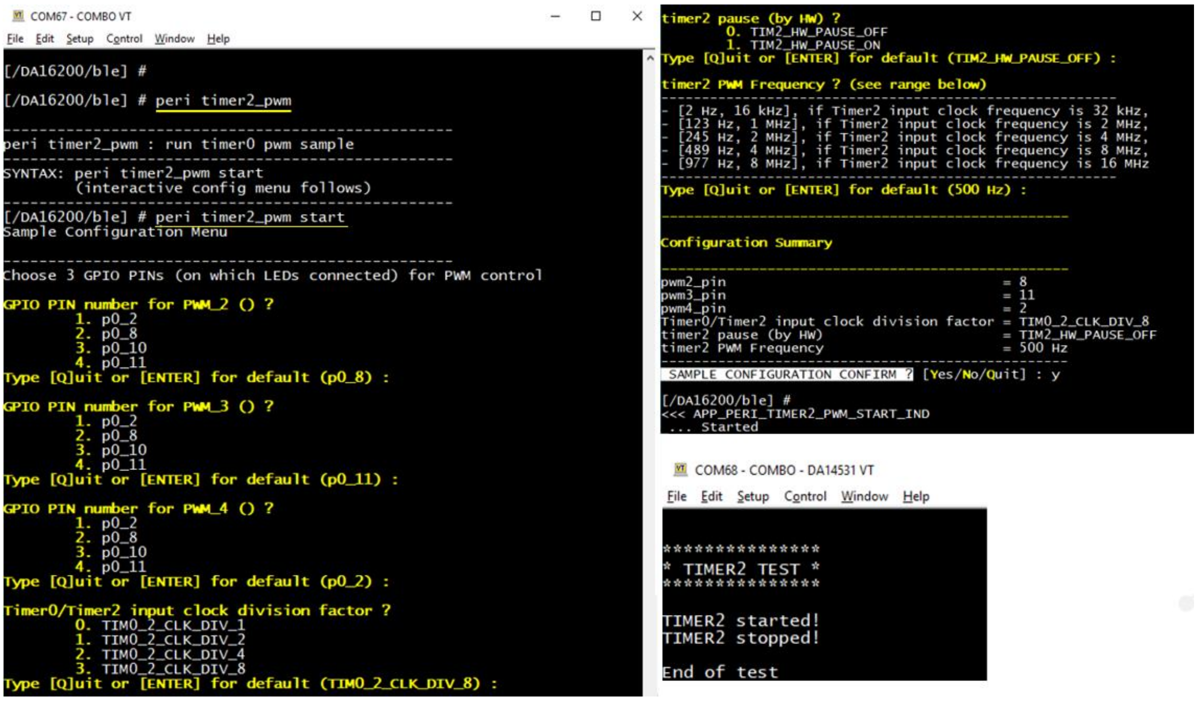


Figure 150. Peri Timer2_pwm

18.7.3.6 peri batt_lvl

The Battery example demonstrates how to read the level of the battery connected to DA14531.

1. DA16600 EVB Configuration (See Figure 143).
2. Run commands as in Figure 151.

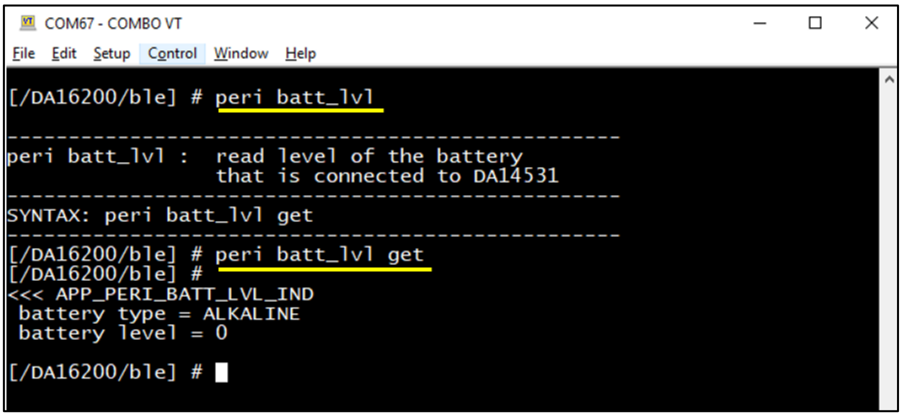


Figure 151. Peri Batt_lvl

18.7.3.7 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.

1. DA16600 EVB Configuration (See Figure 143).

   By default, P0_8 (SCL – Serial Clock) and P0_11 (SDA – Serial Data) are used. Connect J3:P0_8, J2:P0_11 to an external I2C_EEPROM. See Ref. [3] for J2 and J3.

2. Run command as in Figure 152 and Figure 153.

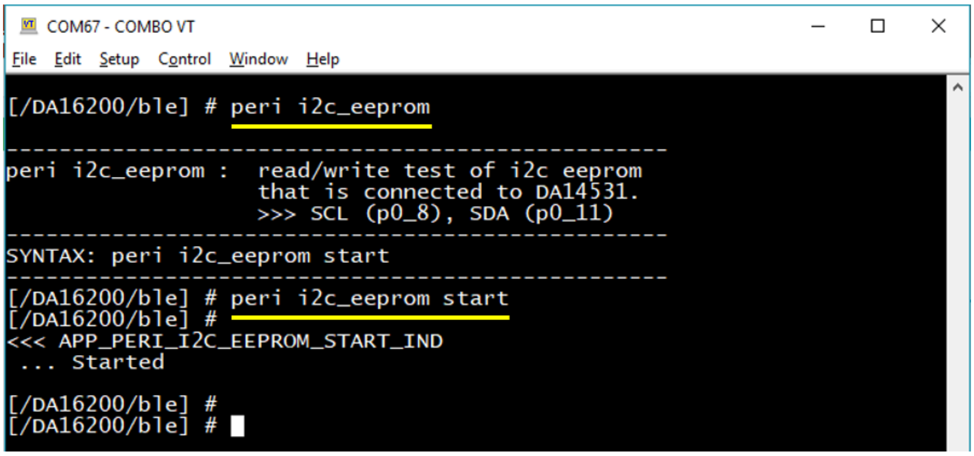


Figure 152. Peri I2c_eeprom

Figure 153. Peri I2c_eeprom read/write

18.7.3.8 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.

1. The following are the pre-defined characteristics configured in the DA14531 image:

```
#define SPI_MS_MODE             SPI_MS_MODE_MASTER
#define SPI_CP_MODE             SPI_CP_MODE_0
#define SPI_WSZ                 SPI_MODE_8BIT
#define SPI_CS                  SPI_CS_0
#define SPI_FLASH_DEV_SIZE      (256 * 1024)
```

2. DA16600 EVB Configuration (See Figure 144).

   By default, 4 GPIO pins are used: SPI_EN (J3: P0_8), SPI_CLK (J2:P0_11), SPI_DO (J3:P0_2), SPI_DI (J3:P0_10). For J2 and J3, see DA16600 EVB Hardware Configuration figure in Ref. [3].

3. Download the following DA14531 image for this test.

   `[DA16600_SDK_ROOT]\apps\da16600\get_started\projects\da16600\img\DA14531_P\peri_spi_flash\da14531_multi_part_proxr_s.img.`

   If you do not use the image above, the message shown in Figure 154 appears.

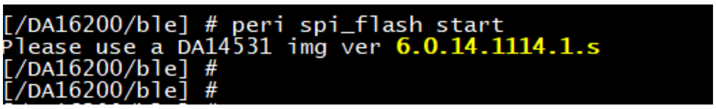


Figure 154. Peri Spi_flash – wrong image warning

4. After booting with a correct Bluetooth® LE test image, you can find the version string printed at boot as in Figure 155.

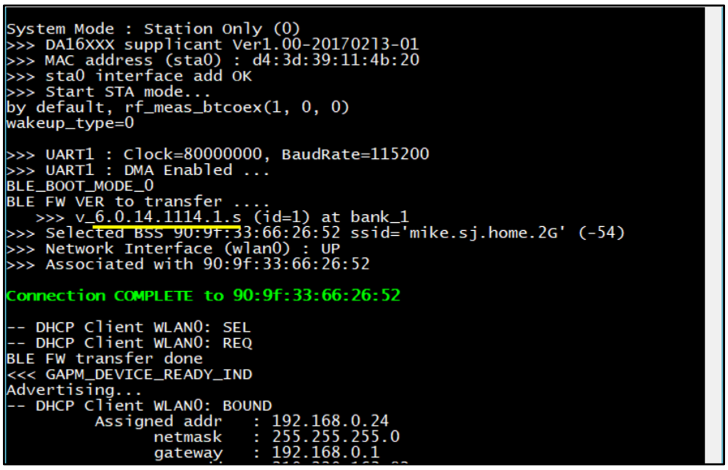


Figure 155. Correct image version for peri Spi_flash sample

5. Run command as shown in Figure 156 and Figure 157.

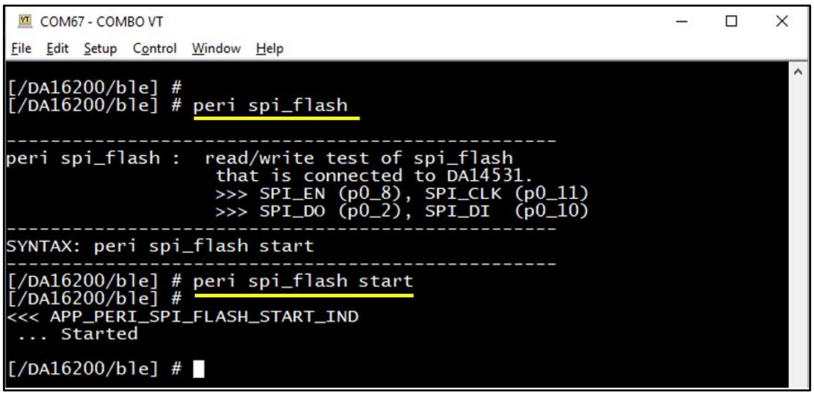


Figure 156. Peri Spi_flash

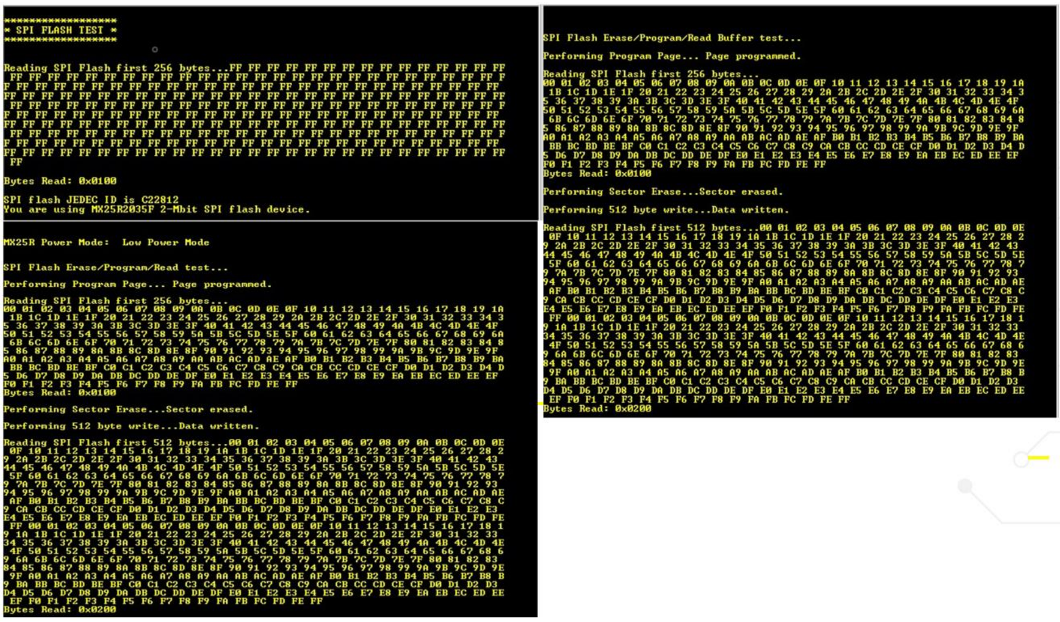


Figure 157. Peri Spi_flash read/write

18.7.3.9 peri gpio

The GPIO example demonstrates how to set/get the state of a GPIO of DA14531 and how to set as input or output. If you set the state of a GPIO of DA14531 to either HIGH or LOW in output mode, the state is kept although DA14531 is in sleep. If you do not want to control the GPIO state of DA14531 anymore, then you need to set the GPIO to 0xFF.

1. DA16600 EVB Configuration (See Figure 143).

   See Section 18.7.5 to check available GPIOs in DA14531.

2. The DA14531 image should be the same as the following image for this test.

   `[DA16600_SDK_ROOT]\apps\da16600\get_started\projects\da16600\img\DA14531_P\`
   `da14531_multi_part_proxr.img`

3. The DA14531 GPIO can be configured to output and input with some options.

   - Syntax: peri gpio set port_no pin_no state [func]
     - port_no, pin_no: port/pin number.
     - state: 1 (high), 0 (low), FF (Not used)
     - func: 0 (INPUT), 1 (INPUT_PULLUP), 2 (INPUT_PULLDOWN), 3 (OUTPUT) - Default OUTPUT

   a. Example: set the P0_8 to high in output mode.

      peri gpio set 0 8 1 3

   b. Example: set the P0_8 as INPUT.

      peri gpio set 0 8 0 0

   c. Example: get the P0_8 status.

      peri gpio get 0 8

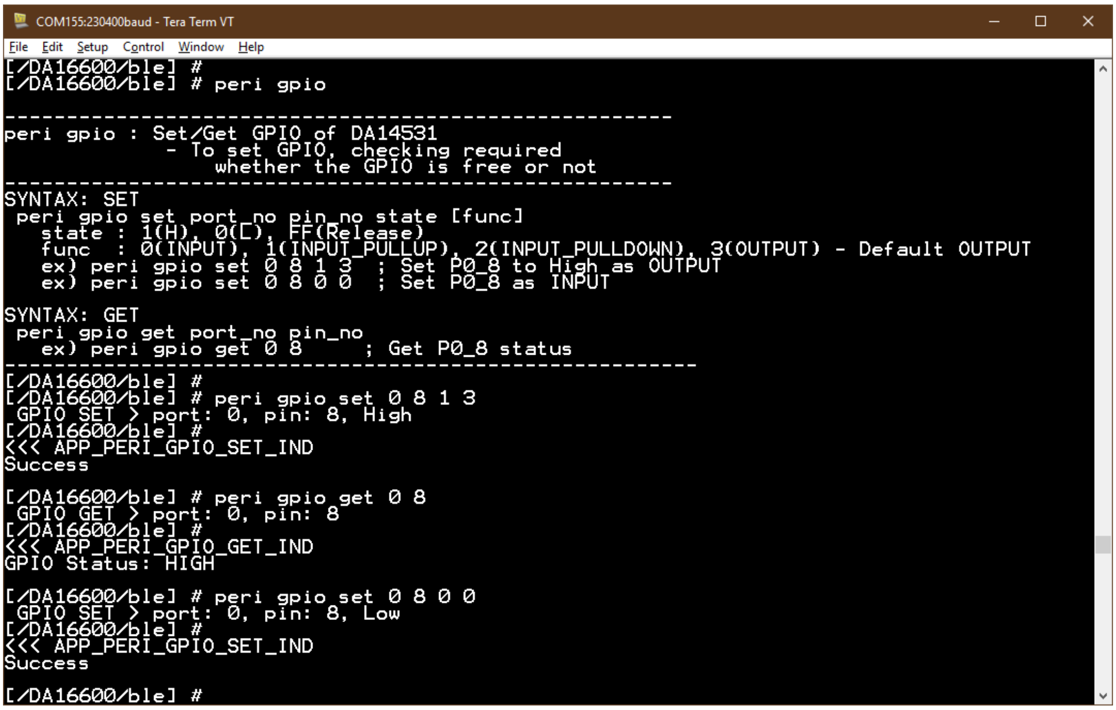


Figure 158. Peri GPIO configuration

18.7.4 Workflow

This example application is not required for Wi-Fi provisioning and controls the DA14531 peripherals (GPIOs) by sending commands in the DA16200 and each example to the DA14531 through the GTL interface.

- Most types and definitions of GTL messages are defined in the `app.h`.

  `[DA16600_SDK_ROOT]\apps\da16600\get_started\src\ble_svc\include\app.h`

- `ext_host_ble_aux_task_msg_t` must be exactly same as in both DA16600 and DA14531 SDK.
- Timer buz console command flow in the DA16600, all test cases should have similar workflow.

ble.peri timer0_buz start

> cmd_peri_sample()

> app_peri_timer0_buz_start_send()

> BleSendMsg()

- Timer buz console command flow in the DA14531.

  GTL > ext_host_ble_aux_task_handler (msgid = APP_PERI_TIMER0_BUZ_START)

  > app_peri_timer0_buz_start_ind_send(): run a timer to start the sample action and send back the GTL "APP_PERI_TIMER0_BUZ_START_IND" to DA16200 indicating timer0_buz sample soon starts. On receipt of this message, DA16200 prints that TIMER0_BUZ started

  > ext_host_ble_aux_task_handler(): handles for the other commands as well

  > app_peri_timer0_buz_run()

  > pwm0_user_callback_function()

18.7.5 GPIO PINs in DA14531

The DA14531 has 12 GPIOs. Among them, seven GPIOs are reserved and being used by GTL (four-wire UART, thus four pins) and BT-Wi-Fi Coex (3-wire, thus 3 pins), therefore, there are five GPIOs free for peripheral devices. Depending on the actual design, the default usage of pins may vary.

- **P0_2**: SWD. Free if __DISABLE_JTAG_SWD_PINS_IN_BLE__ is defined (by default, SWD disabled)
- **P0_8/P0_9**: used as UART2 for peripheral driver sample print-out
- **P0_10**: SWD. Free if __DISABLE_JTAG_SWD_PINS_IN_BLE__ is defined (by default, SWD disabled)
- **P0_11**: available as the RESET pin after booting up
- **P0_5**: used as Coex: wlanAct in default DA14531 image

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

18.8 IoT Sensor Gateway Example (Bluetooth® LE Central)

In this example, the DA16600 plays the role of a gateway device for multiple Bluetooth® LE temperature sensors. A Bluetooth® LE sensor posts the current temperature value periodically via the notify function of Bluetooth® LE. The Bluetooth® LE chip of DA16600 gathers the information as a central (host) device and asks Wi-Fi to (periodically) post notifications to a service server in the cloud. Figure 159 shows how the data is transferred to the server.

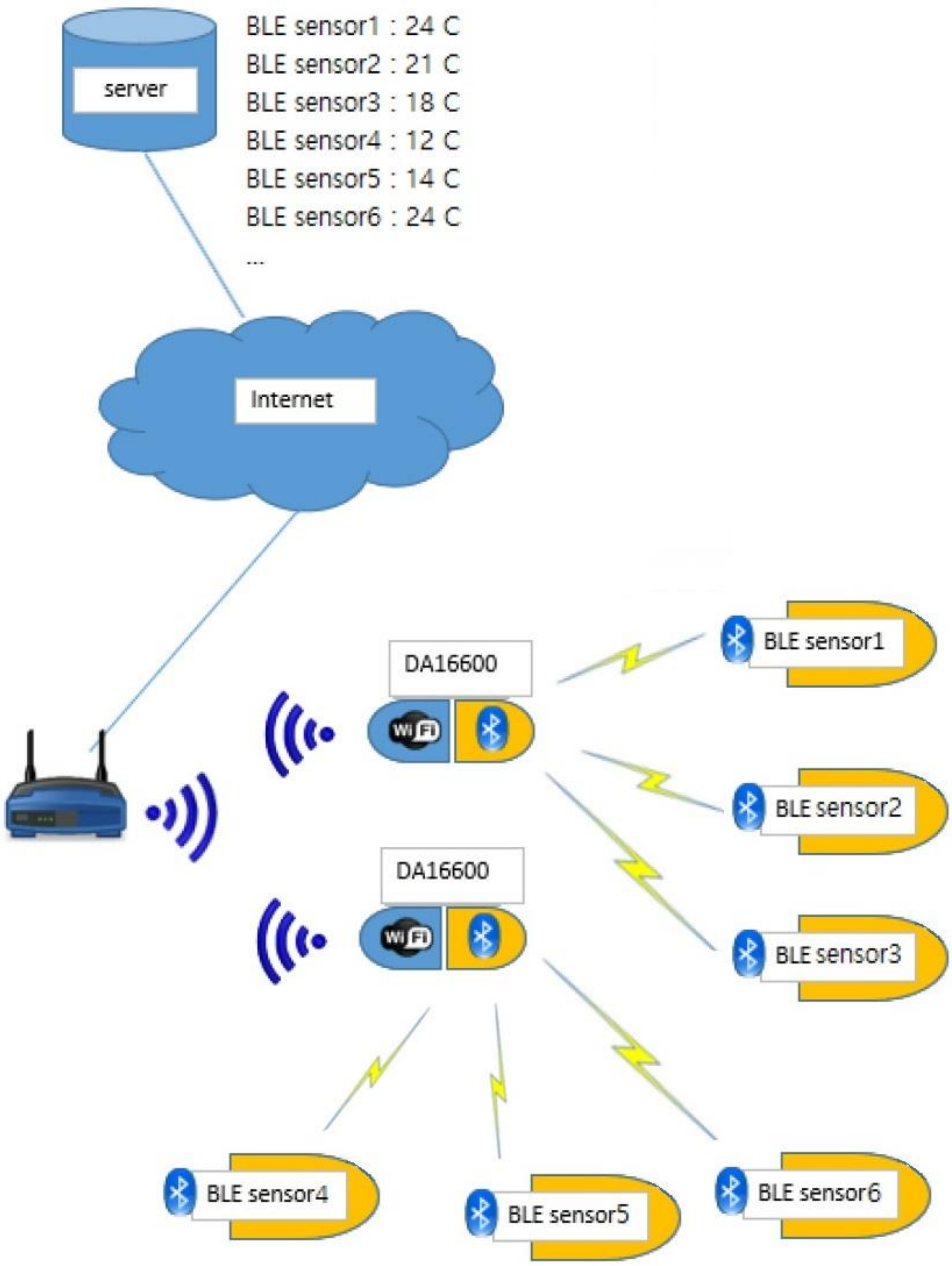


Figure 159. **IoT sensor gateway**

18.8.1 Description and Requirements

To build and run for this application, see Sections 18.2.2 and 18.2.2.4. In this example, a few Bluetooth® LE peripheral devices are used (See Section 18.5). The DA16600 receives sensor data through the DA14531 which works as central mode, coming from peripheral devices and sends them to the server.

18.8.2 Test Setup and Procedure

Set up the DA16600 boards (up to three) as peripheral sensor devices.

NOTE
Gas Leak Detection Sensor application starts advertising after booting up. That is the application is waiting to be connected by a GAP Central – this example application.

1. da16_tera_win:
   a. Connect the USB cable to IoT Sensor Gateway Example.

b. Connect Tera Term to the EVB.

By default, the sensor gateway application is running as "Bluetooth® LE GAP Central" that scans neighbor GAP Peripheral devices, the provisioning should be done before this test. See Section 18.3.2 and run the provisioning procedure if it is not done yet.

2. The Gateway device in Bluetooth® LE is in scanning mode, when the scan is finished, the list is as shown below when there are two peripheral sensor boards.

```
#####################################################
          #    DA14531 Proximity Monitor demo application    #
#####################################################
#  No.  bd_addr                Name                   Rssi           #
#  1    b8:00:00:00:00:01       DA16600-AAAA           -48 dB         #
#  2    b8:00:00:00:00:02       DA16600-BBBB           -42 dB         #
#  3    ec:00:00:00:00:00       Mi Smart Band 4        -62 dB         #
#  4    80:ea:ca:80:00:01       Dialog SOC Demo        -58 dB         #
Scanning... Wait GAPM_ADV_REPORT_IND
>>> Connect or rescan. Type in "[/DA16200] # ble. monitor" for cmd options
```

3. By typing `ble` and `monitor` command below, the commands available in the example are displayed in the following log box.

```
[/DA16600/] # ble
[/DA16600/ble] # monitor
...
        monitor [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
```

4. To connect the devices (NO 1, NO 2) in the scan list, type the following commands:

```
[/DA16600/ble] # monitor conn 1
...
[/DA16600/ble] # monitor conn 2
...
#  No. Model No.        BDA                  Bonded  RSSI   LLA  TX_PL  Temp
#  * 1 iot_sensor       b8:00:00:00:00:01     NO
```

```
#    2 iot_sensor       b8:00:00:00:00:02   NO
#    3                --   Empty Slot --
```

5. To enable the notification from the peer device - the IoT sensor device, type the command in the log box. Then the temperatures should be posted from the peer device to the gateway device.

```
[/DA16600/ble] # monitor peer 1 J
...

[/DA16600/ble] # monitor peer 2 J
...
#  No. Model No.        BDA                Bonded  RSSI   LLA  TX_PL  Temp
#  * 1 iot_sensor       b8:00:00:00:00:01   NO                          35
#    2 iot_sensor       b8:00:00:00:00:02   NO                          13
#    3                --   Empty Slot --
```

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

6. Command "J" let sensor start temperature posting.
7. ssh_win_1 or mobile application: the temperature data are posted to the server as follows.

```
...
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 = 33
 >>> iot_sensor[2], temperature value = 11
 >>> iot_sensor[1], temperature value = 31
 >>> iot_sensor[2], temperature value = 8
...
```

NOTE

Details of sensor service are that implement the following GATT service on a user Bluetooth® LE device:

- Temperature characteristic: UUID = '12345678-1234-5678-1234-56789abcdef1', Permission = READ | NOTIFY, Value size = 1 byte
- If subscription (on CCCD) is requested by a peer, periodic notification starts every five sec (the temperature value is notified every five seconds)
- If subscription (on CCCD) is requested by a peer, periodic notification starts every five sec (the temperature value is notified every five seconds)

18.8.3 Workflow

After provisioned and booted up, the DA16600 tries to scan and connect the peripheral devices. When the connection between the DA14531 and peripheral is established, the sensor data is transmitted to DA16600 (DA14531 - GTL - DA16200), and then DA16200 sends the data to the server.

- The DA16600 tries to scan automatically after booting up or by typing the following scan command to start

```
[/DA16600] # ble. monitor scan
```

NOTE

For DA14531/Bluetooth® LE scan activity, if the DA16600 is connected to the network, the duration available for Wi-Fi network should be more than 110ms, it can be calculated by scan interval and window duration as follows.

- Available duration for Wi-Fi = interval x 0.625(time slot) – window x 0.625(time slot)
- It can be tuned in the app_inq() function in sensor_gw_app.c (.\apps\da16600\get_started\src\ble_svc\sensor_gw\src)
- Renesas recommends that separate the Wi-Fi activities and Bluetooth® LE activities if it takes long time

- `ConsoleEvent()`: UI handler for controlling sensor gateway, the console commands are sent to the DA14531 through the GTL.
- Connect the devices and enable the notification to the peripheral devices.
- The peripheral devices send the data to the DA14531.
- Transfer the data to the DA16200 and post it to the server.

18.8.4 GTL Message Flow

18.8.4.1 Initialization

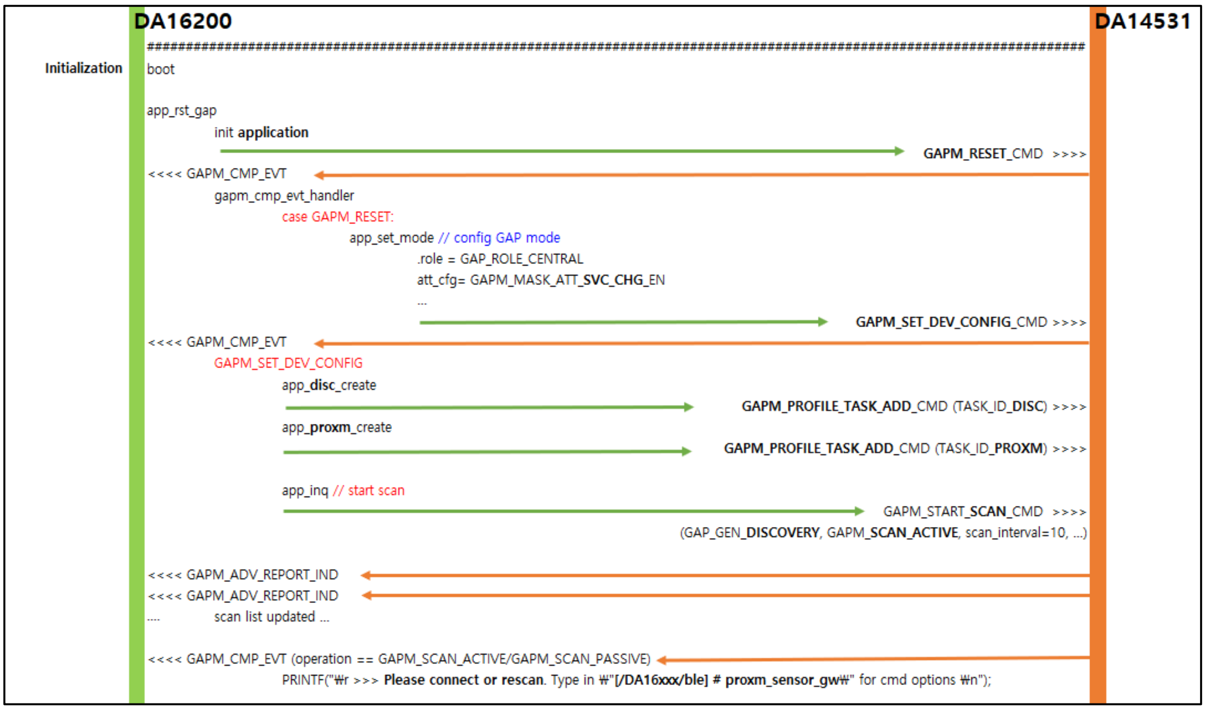


Figure 160. GTL message sequence chart – initialization

18.8.4.2 Provisioning Mode

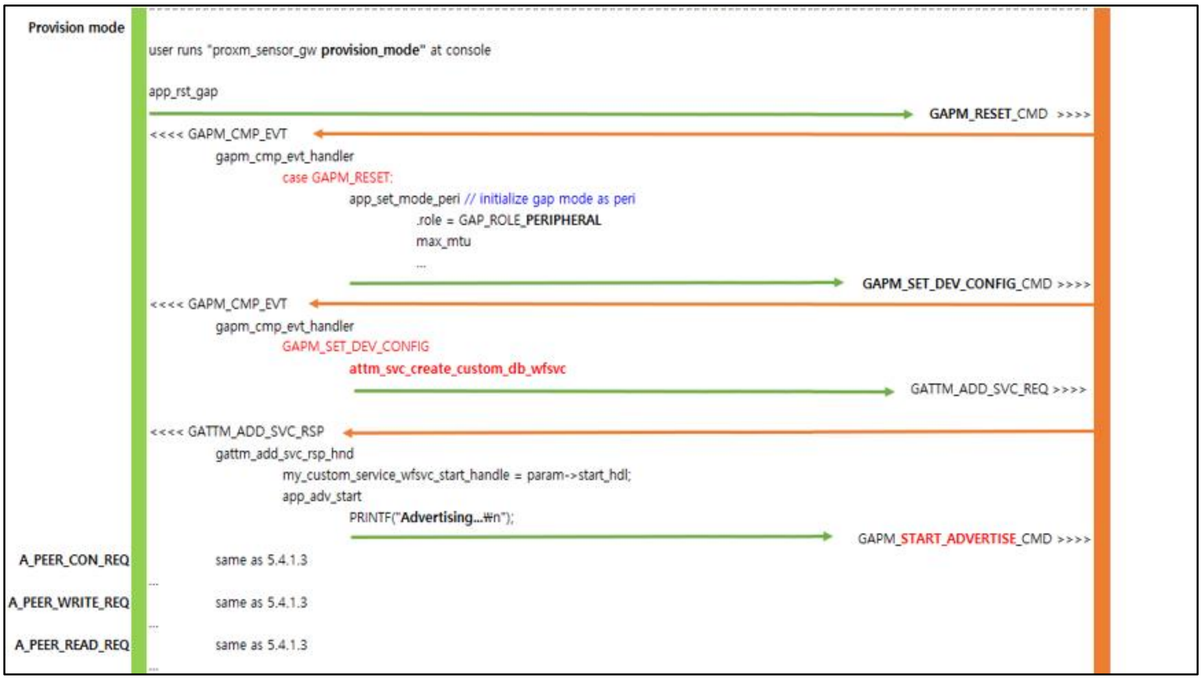
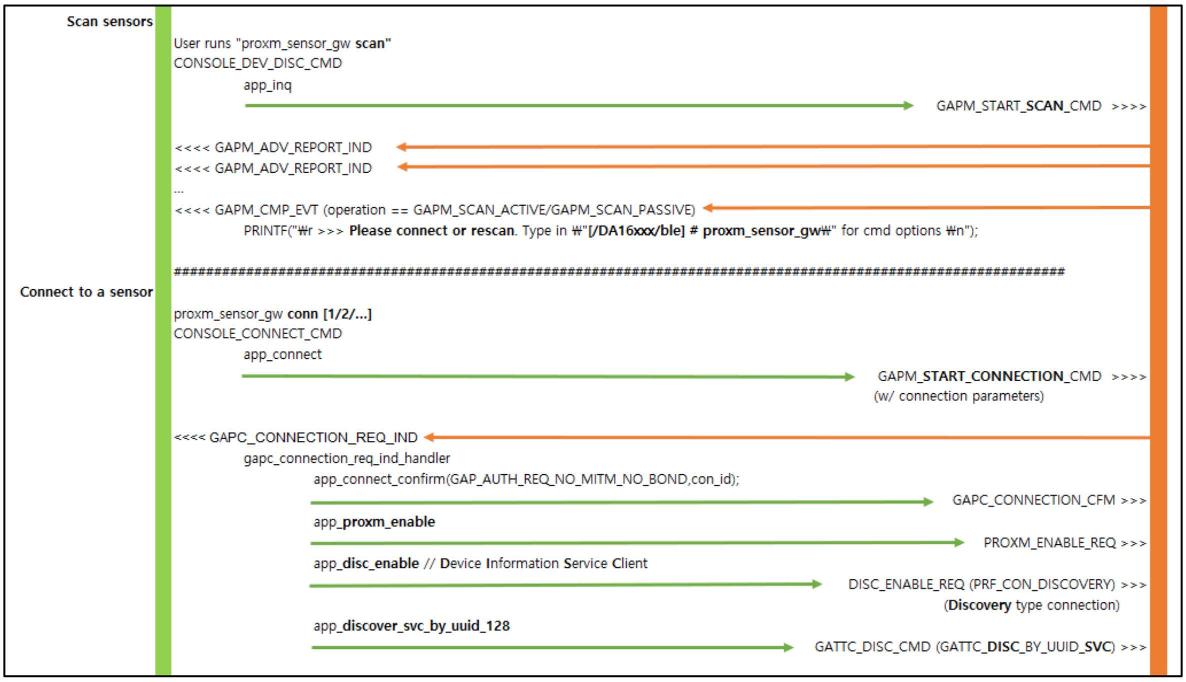
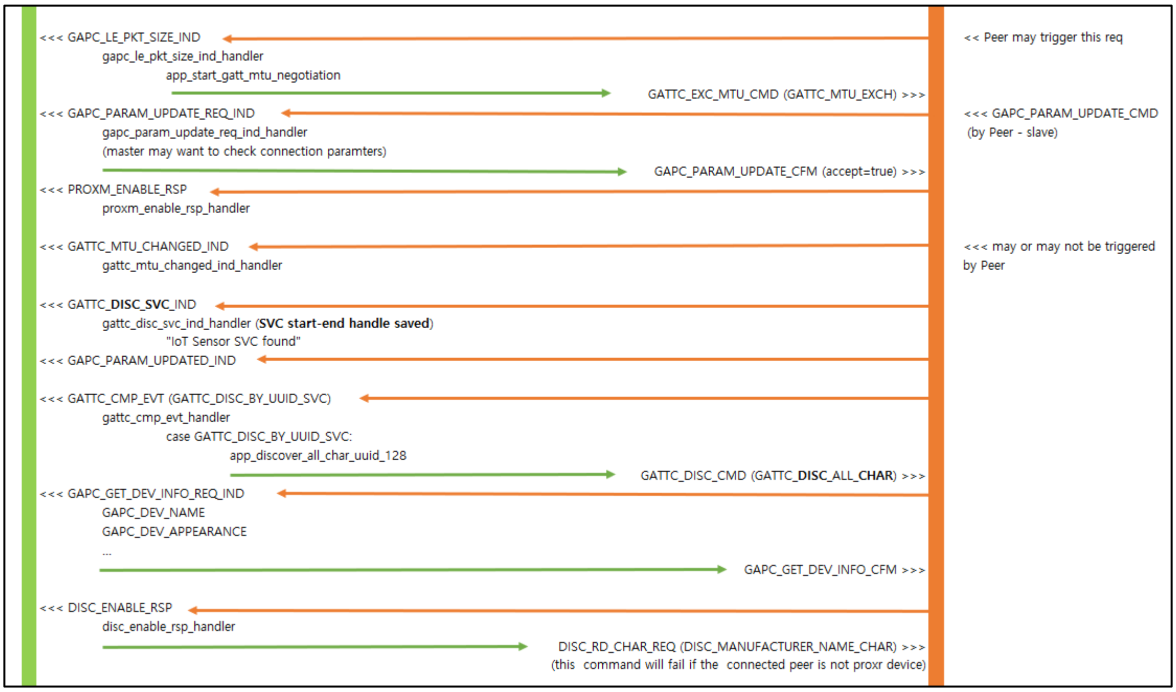


Figure 161. GTL message sequence chart – provisioning mode

18.8.4.3 Scan and Connect to Sensor





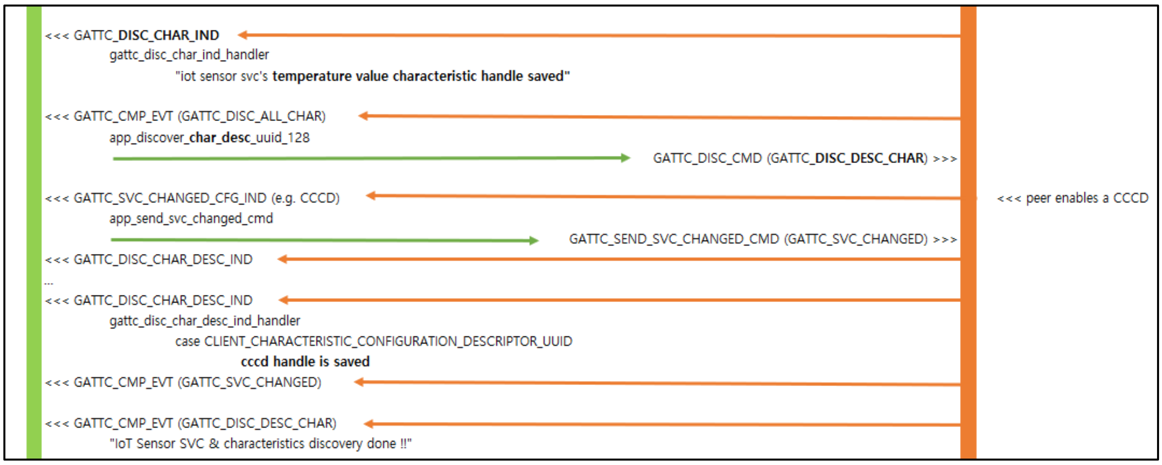


Figure 162. GTL message sequence chart – scan and connect

18.8.4.4 Enable Sensor Posting

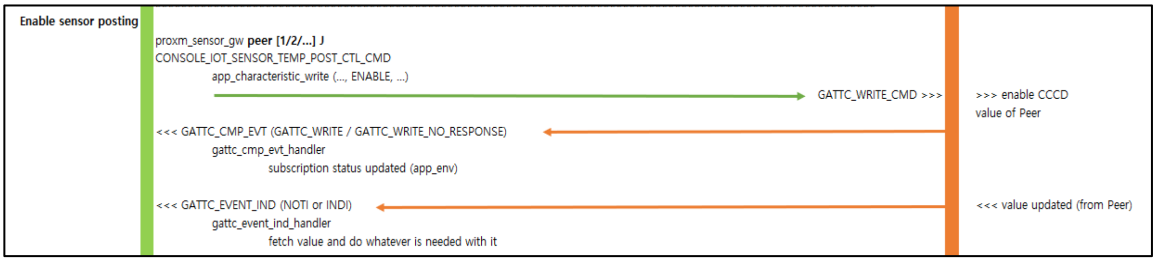


Figure 163. GTL message sequence chart – enable sensor posting

18.8.4.5 Disable Sensor Posting

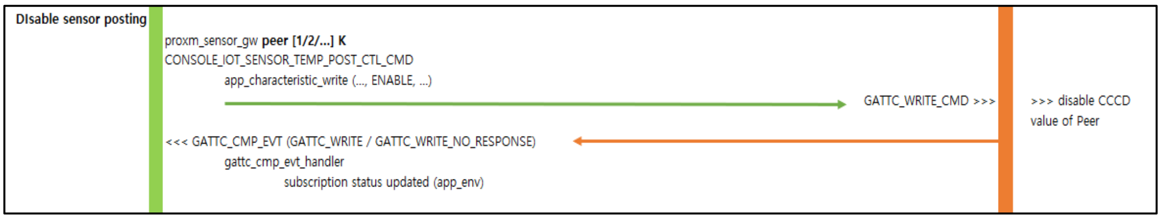


Figure 164. GTL message sequence chart – disable sensor posting

Appendix A License Information

A.1 Mosquitto 1.4.14 License

```
Eclipse Distribution License 1.0

Copyright (c) 2007, Eclipse Foundation, Inc. and its licensors.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided
that the following conditions are met:

    * Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright notice,
      this list of conditions and the following disclaimer
      in the documentation and/or other materials provided with the distribution.
    * Neither the name of the Eclipse Foundation, Inc.
      nor the names of its contributors may be used to endorse or
      promote products derived from this software without
      specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

A.2 MiniUPnPc License

```
Copyright (c) 2005-2016, Thomas BERNARD

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided
that the following conditions are met:

    * Redistributions of source code must retain the above copyright notice,
      this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright notice,
      this list of conditions and the following disclaimer in the documentation
      and/or other materials provided with the distribution.
    * The name of the author may not be used to endorse or promote products
          derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
-----------------------------------------------------------------------------------------------------
-----------------------------
UMAC GPL License
Linux kernel 3.9.0 rc3 version (backport 4.2.6-1)
```

Appendix B TX Power Table Edit

The DA16200/DA16600 SDK allows users to tune and edit TX Power (per channel) for FCC or country-dependent product customization/optimization. The country code and channels describe in Table 87.

Ch.2	11b				11g								11n							
Power Index	1Mbps	2Mbps	5.5Mbps	11Mbps	6Mbps	9Mbps	12Mbps	18Mbps	24Mbps	36Mbps	48Mbps	54Mbps	MCS0	MCS1	MCS2	MCS3	MCS4	MCS5	MCS6	MCS7
0	20.9	20.9	21.0	21.1	19.0	19.0	19.0	19.0	17.4	17.5	16.3	15.3	18.9	18.9	18.8	17.3	17.3	16.1	15.4	15.3
1	20.4	20.4	20.5	20.6	18.4	18.4	18.4	18.4	16.8	16.9	15.5	14.6	18.2	18.2	18.3	16.7	16.7	15.5	14.6	14.7
2	19.7	19.7	19.8	19.8	17.6	17.6	17.6	17.6	15.9	16.0	14.6	13.7	17.4	17.5	17.5	15.9	15.9	14.7	13.8	13.6
3	19.1	19.1	19.2	19.2	17.0	17.1	16.9	17.0	15.3	15.4	14.0	13.1	16.9	16.9	16.8	15.2	15.2	14.0	13.1	13.1
4	18.0	18.0	18.1	18.1	15.9	16.0	15.8	15.9	14.1	14.2	12.8	11.9	15.8	15.8	15.7	14.0	14.1	12.8	12.0	11.8
5	16.8	16.7	16.8	16.9	14.8	14.9	14.8	14.8	13.4	13.6	12.1	11.2	14.7	14.7	14.7	13.3	13.3	12.1	11.2	11.1
6	16.2	16.1	16.3	16.2	14.2	14.2	14.2	14.2	12.8	12.9	11.5	10.5	14.0	14.1	14.1	12.8	12.7	11.4	10.5	10.5
7	15.4	15.4	15.4	15.5	13.4	13.4	13.4	13.3	11.9	12.0	10.6	9.7	13.2	13.2	13.3	11.8	11.9	10.6	9.7	9.7
8	14.8	14.8	14.9	14.9	12.8	12.8	12.8	12.8	11.2	11.3	9.8	9.0	12.7	12.7	12.7	11.1	11.1	9.8	9.0	8.9
9	13.8	13.8	13.8	13.8	11.7	11.7	11.7	11.7	10.2	10.3	8.9	8.0	11.5	11.6	11.5	10.2	10.1	9.0	8.1	8.0
10	13.1	13.1	13.2	13.2	11.0	11.1	11.0	11.0	9.7	9.7	8.3	7.5	10.9	10.9	10.9	9.5	9.5	8.3	7.4	7.4
11	12.6	12.6	12.7	12.7	10.5	10.5	10.4	10.5	8.5	8.5	7.1	6.2	10.3	10.4	10.3	8.3	8.4	7.1	6.2	6.2

Figure 165. TX power table

NOTE
The 2.4 GHz band is divided into 14 channels at 5 MHz intervals centered at 2.412 GHz, starting with channel 1. The last channel (CH 14) has additional restrictions or cannot be used for all regulatory areas. ▪ TX power setting value range: 0x0 ~ 0xB ▪ Setting value for unsupported channel: 0xF

B.1 Tune TX Power

Users can tune and test TX power through CLI command and AT GUI tool. When the TX power value increases by 1 step, the actual TX power decreases by 0.8 dB. Unsupported channels should be set to 0xF. See the corresponding section of Ref. [5] on how to tune TX power using CLI or AT GUI tool.

This example shows the range of values for TX power.

Table 86. TX power setting value range

▪ TX power gap per value step: 0.8 dB ▪ TX power setting value range: 0x0 ~ 0xB ▪ Setting value for unsupported channel: 0xF

B.2 Apply Tuned TX Power to Main Image

The following procedure describes how to set the tuned TX power indices to the Main image.

1. In the DA16200/DA16600 SDK, open `~/FreeRTOS_SDK/core/system/src/common/main/sys_user_feature.c`.

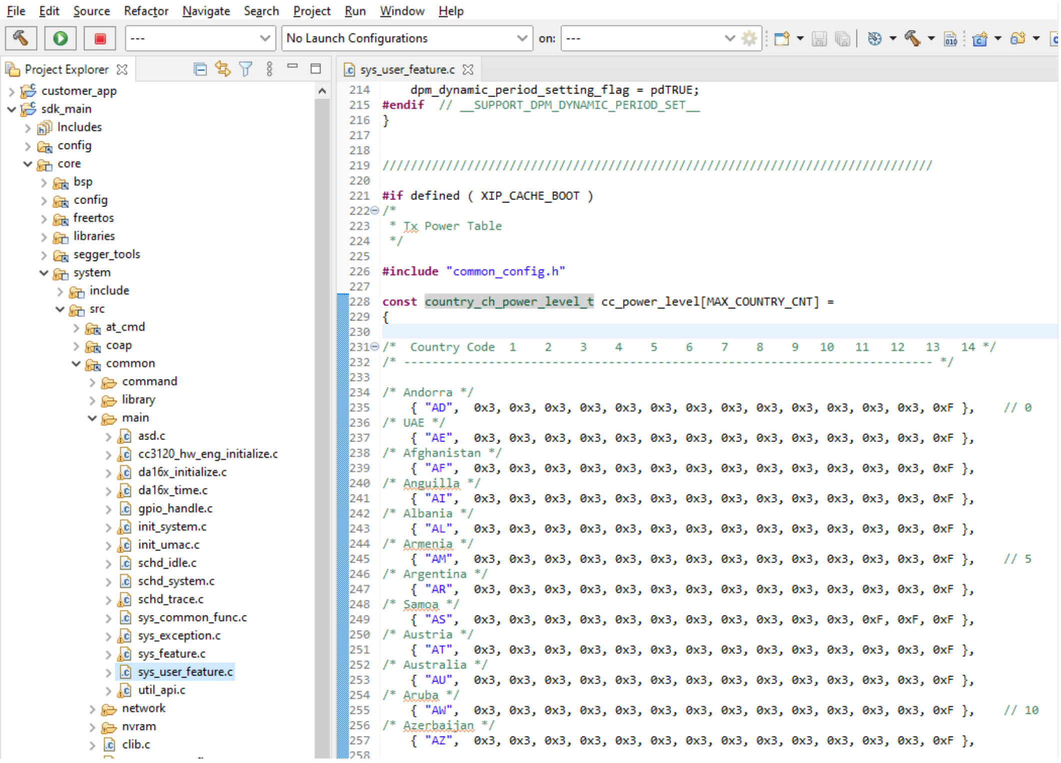


Figure 166. TX power table source code

The array cc_power_level contains the default values customized for FCC. Edit the power values for a specific country or the desired countries with tuned values. See Table 86 for more information.

2. Re-build the SDK.

   When the rebuilt software is started and the country is selected, the corresponding Tx power value set for that channel takes effect.

Appendix C Tips

C.1 Find/Optimize Stack Size for Applications

To check the stack size of the application, the DA16200/DA16600 has a tool (a console command) called **ps** that shows the list of threads and the status of each application stack. Figure 167 is a snapshot of command ps when `tcp_client_sample.c` is run.

```
[/DA16200] # ps

 <<< Task information >>>
 Task count: 20  TotalTime: 507 Ticks
 ===============================================================================
 No TaskName     State    Run-Tm   Run-Per     Prio Stack-B Stack-E  S-Size Stack-L
 ===============================================================================
  1 system_laun Blocked       3      <1%         1 0xaf2b8 0xafeb0    3072    1476
  2 IDLE        Ready       496      97%         0 0xaff50 0xb00d8     400     316
  3 Tmr_Svc     Blocked       0      <1%        18 0xb0500 0xb0cf8    2048    1836
  4 Console_OUT Blocked       0      <1%         4 0xb4ce0 0xb50d8    1024     788
  5 Console_IN  Running       0      <1%         3 0xb58e0 0xb70d8    6144    5380
  6 wdt_kicking Blocked       0      <1%        31 0xb7590 0xb7788     512     420
  7 UmTaskletSv Blocked       0      <1%        20 0xb9130 0xb9528    1024     892
  8 UmTxNiTd    Blocked       0      <1%        19 0xb9a20 0xbb218    6144    5484
  9 UmRxMacTd   Blocked       1      <1%        20 0xbb2b8 0xbc2b0    4096    3188
 10 UmMacNiTd   Blocked       0      <1%        20 0xbc350 0xbd348    4096    3764
 11 @LmacMain   Blocked       1      <1%        22 0xbda58 0xbea50    4096    3776
 12 @UmacRx     Blocked       0      <1%        21 0xbeb58 0xbfd50    4608    4224
 13 UmWrkqSvc   Blocked       0      <1%        20 0xbff20 0xc0f18    4096    2964
 14 LwIP_init   Blocked       0      <1%        10 0xc1af0 0xc22e8    2048    1104
 15 lwIP        Blocked       1      <1%        21 0xc25f0 0xc2fe8    2560    1600
 16 wifi_ev_mon Blocked       0      <1%        17 0xc3088 0xc3680    1536     864
 17 da16x_supp  Blocked       5      <1%        18 0xc3988 0xc5980    8192    6028
 18 sntp_thread Blocked       0      <1%         7 0xc7a90 0xc8a88    4096    3284
 19 poll_gpio   Blocked       0      <1%         0 0xc8b28 0xc8f20    1024     932
 22 TCPC        Blocked       0      <1%         7 0xd41f0 0xd51e8    4096    2572
 ===============================================================================
```

Figure 167. Check stack size

TCPC is the name of the tread for this sample application, and the stack size is 1020 (which is defined in sample_apps.c).

This is a sample code of TCP Client.

```
…
#if defined (__TCP_CLIENT_SAMPLE__)
      { SAMPLE_TCP_CLI, tcp_client_sample, 1024, (tskIDLE_PRIORITY + 7), TRUE, FALSE,
      TCP_CLI_TEST_PORT, RUN_ALL_MODE },
#endif // (__TCP_CLIENT_SAMPLE__)
…
```

Command `ps` shows the following information:

- Stack-B/E: the stack address
- S-Size: the stack size allocated
- Stack-L: peak usage size of the stack

To find and optimize the stack size for this application, for example if this application has four use cases, follow the steps below:

1. First, over-allocate stack memory as a precaution, like 2K, "just to be safe".
2. Run each use case and examine the peak stack usage with command `ps`.
3. Allocate optimal memory based on peak usage information. If all the possible use case scenarios are confirmed, allocate the stack size with extra memory as a precaution.

C.2 How to Make/Write User Data to User Area of Flash Externally

Many different methods are available to generate user data and program it into Flash, and here is a simple method to get started. User data can be generated using any hex editor. Figure 168 shows a snapshot for editing data in hex editor, Notepad++. The hex data (left side) or string (right side) can be added and updated.

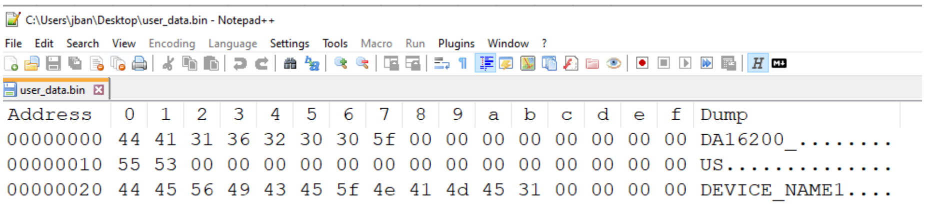


Figure 168. Snapshot of hex editor

User data can be programmed to the user area of flash in DA16200 using MultiDownloader. See Ref. [3] for more details. Figure 169 shows the settings of MultiDownloader for programming data to a specific address.

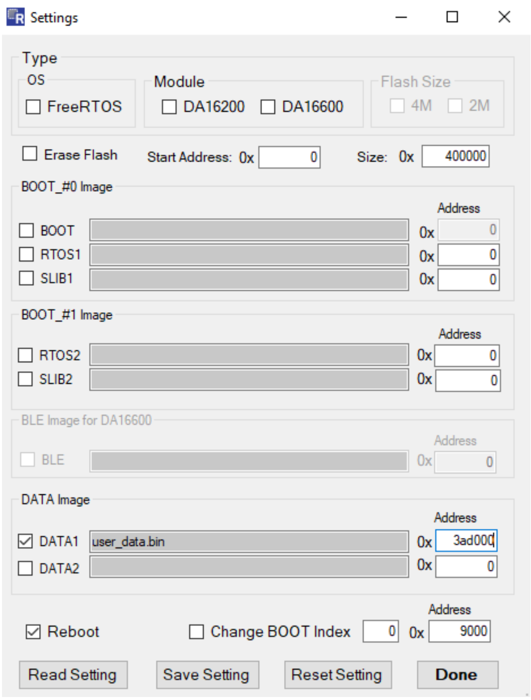


Figure 169. Settings of multidownloader

User data can be confirmed using "sflash read address size" command in MROM. See Figure 170.

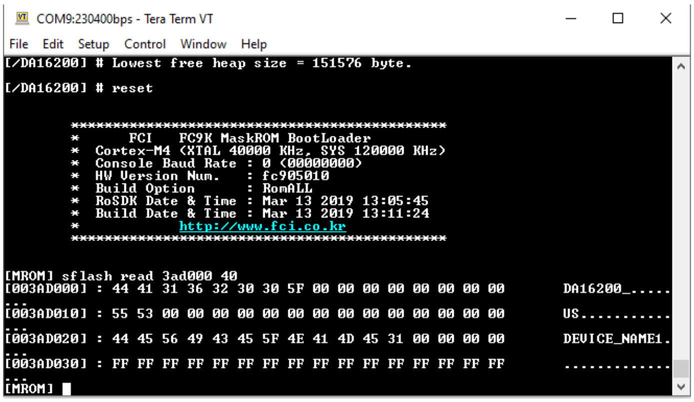


Figure 170. Read user data using command

Appendix D Country Code and TX Power

This section lists the country codes that the DA16200/DA16600 supports and the supported channels of 2.4 GHz bandwidth in the STA and the Soft AP mode.

D.1 Country Code and Channels

Table 87. Country code

Country code	Country	STA channels	Soft AP channels
"AD"	Andorra	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“AE”	United Arab Emirates	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“AF”	Afghanistan	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“AI”	Anguilla	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“AL”	Albania	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“AM”	Netherlands Antilles	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“AR”	Argentina	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“AS”	American Samoa	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
“AT”	Austria	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“AU”	Australia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“AW”	Aruba	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“AZ”	Azerbaijan	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BA”	Bosnia and Herzegovina	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BB”	Barbados	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BD”	Bangladesh	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BE”	Belgium	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BF”	Burkina Faso	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BG”	Bulgaria	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BH”	Bahrain	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BL”	Saint-Barthelemy	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BM”	Bermuda	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
“BN”	Brunei Darussalam	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BO”	Bolivia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BR”	Brazil	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BS”	Bahamas	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BT”	Bhutan	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BY”	Belarus	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“BZ”	Belize	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“CA”	Canada	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
“CF”	Central African Republic	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“CH”	Switzerland	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“CI”	Ivory Coast	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“CL”	Chile	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“CN”	China	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13

Country code	Country	STA channels	Soft AP channels
"CO"	Colombia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"CR"	Costa Rica	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"CU"	Cuba	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"CX"	Christmas Island	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"CY"	Cyprus	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"CZ"	Czech Republic	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"DE"	Germany	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"DK"	Denmark	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"DM"	Dominica	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
"DO"	Dominican Republic	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
"DZ"	Algeria	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"EC"	Ecuador	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"EE"	Estonia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"EG"	Egypt	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"ES"	Spain	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"ET"	Ethiopia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"EU"	Europe	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"FI"	Finland	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"FM"	Micronesia, Federated States of	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
"FR"	France	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"GA"	Gabon	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"GB"	United Kingdom	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"GD"	Grenada	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
"GE"	Georgia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"GF"	French Guiana	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"GH"	Ghana	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"GL"	Greenland	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"GP"	Guadeloupe	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"GR"	Greece	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"GT"	Guatemala	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
"GU"	Guam	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
"GY"	Guyana	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"HK"	Hong Kong	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"HN"	Honduras	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"HT"	Haiti	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
"HU"	Hungary	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"ID"	Indonesia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"IE"	Ireland	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"IL"	Israel	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13

Country code	Country	STA channels	Soft AP channels
“IN”	India	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“IR”	Iran, Islamic Republic of	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“IS”	Iceland	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“IT”	Italy	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“JM”	Jamaica	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“JO”	Jordan	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"JP”	Japan	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“KE”	Kenya	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“KH”	Cambodia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“KN”	Saint Kitts and Nevis	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“KP”	North Korea	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“KR”	South Korea	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“KW”	Kuwait	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“KY”	Cayman Islands	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“KZ”	Kazakhstan	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“LB”	Lebanon	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“LC”	Saint Lucia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“LI”	Liechtenstein	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“LK”	Sri Lanka	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“LS”	Sesotho	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“LT”	Lithuania	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“LU”	Luxembourg	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“LV”	Latvia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MA”	Morocco	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MC”	Monaco	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MD”	Moldova	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“ME”	Montenegro	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MF”	Saint-Martin (French part)	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MH”	Marshall Islands	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
“MK”	Macedonia, Republic of	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MN”	Mongolia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MO”	Macao	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MP”	Northern Mariana Islands	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
“MQ”	Martinique	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MR”	Mauritania	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MT”	Malta	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MU”	Mauritius	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MV”	Maldives	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MW”	Malawi	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“MX”	Mexico	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13

Country code	Country	STA channels	Soft AP channels
“MY”	Malaysia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“NG”	Nigeria	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“NI”	Nicaragua	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
“NL”	Netherlands	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“NO”	Norway	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“NP”	Nepal	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“NZ”	New Zealand	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"OM”	Oman	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“PA”	Panama	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
“PE”	Peru	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“PF”	French Polynesia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“PG”	Papua New Guinea	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“PH”	Philippines	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“PK”	Pakistan	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“PL”	Poland	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“PM”	Saint Pierre and Miquelon	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“PR”	Puerto Rico	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
“PT”	Portugal	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“PW”	Palau	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
“PY”	Paraguay	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“QA”	Qatar	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“RE”	Reunion	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“RO”	Romania	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“RS”	Serbia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“RU”	Russian Federation	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“RW”	Rwanda	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“SA”	Saudi Arabia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“SE”	Sweden	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“SG”	Singapore	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“SI”	Slovenia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“SK”	Slovak Republic	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“SN”	Senegal	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“SR”	Suriname	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“SV”	El Salvador	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“SY”	Syrian Arab Republic	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“TC”	Turks and Caicos Islands	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“TD”	Chad	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“TG”	Togo	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“TH”	Thailand	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“TN”	Tunisia	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13

Country code	Country	STA channels	Soft AP channels
“TR”	Turkey	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“TT”	Trinidad and Tobago	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“TW”	Taiwan	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“TZ”	Tanzania	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“UA”	Ukraine	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
"UG”	Uganda	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“US”	United States of America	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
“UY”	Uruguay	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“UZ”	Uzbekistan	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“VC”	Saint Vincent and Grenadines	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“VE”	Venezuela	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“VI”	Virgin Islands	1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11
“VN”	Vietnam	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“VU”	Vanuatu	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“WF”	Walls and Futuna Islands	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“WS”	Samoa	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“YE”	Yemen	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“YT”	Mayotte	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“ZA”	South Africa	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“ZW”	Zimbabwe	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“00”	Worldwide	1,2,3,4,5,6,7,8,9,10,11,12,13	1,2,3,4,5,6,7,8,9,10,11,12,13
“XX”		1,2,3,4,5,6,7,8,9,10,11	1,2,3,4,5,6,7,8,9,10,11

D.2 Programming

All of the power level settings are 0x0 as default setting, so it should be set as required for the customer’s specifications and requirement. The power table consists of two types, one is for OFDM and the other is for DSSS. DA16200/DA16600 refers to the levels either of the cc_power_level table for OFDM mode or the cc_power_level_dsss for DSSS mode.

In the DA16200/DA16600 SDK, users can change the supporting "country code list" for their product. See Table 86 and Table 88.

NOTE
The 2.4 GHz band is divided into 14 channels at 5 MHz intervals centered at 2.412 GHz, starting with channel 1. The last channel (CH 14) has additional restrictions or cannot be used in all regulatory areas. ▪ TX power setting value range: 0x0 ~ 0xB ▪ Setting value for unsupported channel: 0xF

- FreeRTOS_SDK/apps/da6200(da16600)/get_started/src/apps/main/user_system_feature.c

This is programming examples of country codes.

Table 88. Programming example for country code

```
const country_ch_power_level_t cc_power_level[MAX_COUNTRY_CNT] =
{

/*  Country  1    2    3    4    5    6    7    8    9   10   11   12   13  14 */
```

```
/* ---------------------------------------------------------------------------- */

/* Andorra */
  { "AD",  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xF },
/* UAE */
  { "AE",  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xF },
/* Afghanistan */
  { "AF",  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xF },
```

```
const country_ch_power_level_t cc_power_level_dsss[MAX_COUNTRY_CNT] =
{

/*  Country  1    2    3    4    5    6    7    8    9   10   11   12   13  14 */
/* ---------------------------------------------------------------------------- */

/* Andorra */
  { "AD",  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xF },
/* UAE */
  { "AE",  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xF },
/* Afghanistan */
  { "AF",  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xF },
```

Appendix E How to Use J-Link Debugger

To debug DA16200 and DA16600, J-Link debug probe and J-Link software are required. See the Debugging with J-Link Debug Probe section of Ref. [3] on how to use J-Link debugger.

Appendix F Create RTOS Image for fcCSP Using SDK v3.2.7.1 or Earlier

For fcCSP type package, to create a RTOS image with the DA16200/DA16600 SDK, change the files as shown below, and then follow the Build SDK instructions described in Section 3.7.

- Library file for Low-Power:
  - ~/FreeRTOS_SDK/library/liblmac.a.fcCSP_LP (or liblmac.fcCSP.LP.a) → ~/FreeRTOS_SDK/library/liblmac.a
- Library file for Normal-Power:
  - ~/FreeRTOS_SDK/library/liblmac.a.fcCSP_NP (or liblmac.fcCSP.NP.a) → ~/FreeRTOS_SDK/library/liblmac.a
- Compile feature:
  - ~/FreeRTOS_SDK/apps/da16200/get_started/include/user_main/sys_common_features.h
    - `#undef __FOR_FCCSP_SDK__` → `#define __FOR_FCCSP_SDK__`
    - In case of Low-Power: `#define __FCCSP_LOW_POWER__`
    - In case of Normal-Power: `#undef __FCCSP_LOW_POWER__`

Appendix G Bluetooth® LE Customization

G.1 How to Change Bluetooth® LE Device Name

The Bluetooth® LE device name can be changed by editing the definition - USER_DEVICE_NAME in the \SDK_ROOT\apps\da16600\get_started\src\ble_svc\include\user_config.h file.

```
#define __APPEND_EXTRA_INFO_AT_DEVICE_NAME

#ifdef __APPEND_EXTRA_INFO_AT_DEVICE_NAME
#define USER_DEVICE_NAME        ("DA16600-XXXX")
#else
#define USER_DEVICE_NAME        ("DA16600")
#endif
```

If `__APPEND_EXTRA_INFO_AT_DEVICE_NAME` is defined, then the string("`XXXX`") is replaced by the last 4 digits of the MAC address stored at the EVB or target board in the `app_rst_gap()` function. If it is not defined, then the fixed device name is used. The change may not be immediately seen because of the phone or APP caching the previous ADV device name, so the cache of the user's APP and BT system APP should be cleared before testing with a new device name.

G.2 How to Change Bluetooth® LE ADV Interval

The advertising interval can be changed by the define - `USER_CFG_DEFAULT_ADV_INTERVAL_MS` in `[SDK_ROOT]\apps\da16600\get_started\src\ble_svc\include\app.h` file for Example applications (Bluetooth® LE peripheral) in `[SDK_ROOT]\apps\da16600\get_started\src\ble_svc\sensor_gw\inc\app.h` file for Example application (Bluetooth® LE central).

```
/* Advertising and Connection related parameters */
#define USER_CFG_DEFAULT_ADV_INTERVAL_MS                (687.5)

/// Advertising minimum interval
#define APP_ADV_INT_MIN                 MS_TO_BLESLOTS(USER_CFG_DEFAULT_ADV_INTERVAL_MS)
/// Advertising maximum interval
#define APP_ADV_INT_MAX                 MS_TO_BLESLOTS(USER_CFG_DEFAULT_ADV_INTERVAL_MS)
```

The ADV interval should not be too long, Renesas recommends not to set it bigger than 1 or 2 seconds because it may not be scanned well by the host due to its longer advertising. The shorter interval can be scanned faster by the hosts of course, but to save the power consumption we have set it to the appropriate value – 687.5 ms.

G.3 How to Configure Bluetooth® LE Hardware Reset

To configure Bluetooth® LE hardware reset, the GPIOC_8/DA16200 must be connected to P0_11/DA14531 first, then users should define the `__CFG_ENABLE_BLE_HW_RESET__` as follows in the DA16600 SDK.

```
// [SDK_ROOT]\apps\da16600\get_started\include\apps\user_custom_config.h

#if defined(__BLE_PERI_WIFI_SVC__) ||
defined(__BLE_PERI_WIFI_SVC_TCP_DPM__) ||
defined(__BLE_CENT_SENSOR_GW__)
      #define __CFG_ENABLE_BLE_HW_RESET__ // Enable hardware reset of Bluetooth
#endif
```

And it needs to configure the POR register in DA14531 for P0_11 working as a reset pin, the `CFG_ENABLE_POR_PIN` should be defined in the DA14531 SDK as well, build the SDK and replace the image file – da14531_multi_part_proxr.img or da14531_multi_part_proxm.img.

```
//[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_reporter_sensor_ext_coex\src\conf
ig\da1458x_config_advanced.h (reporter project) or
[DA14531_SDK_ROOT]\projects\target_apps\ble_examples\prox_monitor_aux_ext_coex\src\config\da1
458x_config_advanced.h (Monitor/Central project)
```

```
/******************************************************************/
/* Enable HW RESET by P0_11 if need                               */
/******************************************************************/
#define CFG_ENABLE_POR_PIN
```

The GPIOC_8/DA16200 and P0_11 should not be used in any other cases apart from this purpose to use the Bluetooth® LE hardware reset properly.

Appendix H QSPI Clock Selection

The XFC driver in the ROM decides the QSPI clock based on the maximum frequency information in the SFDP and core clock during boot. The clock is (core clock/N) which is less than or equal to the maximum frequency in SFDP. N is larger than 1.

For example, if the maximum frequency in the SFDP is 100 MHz and the core clock is 120 MHz, then the QSPI clock is configured as 60 MHz (120/2 <= max. frequency 100 MHz in SFDP). If the maximum frequency in SFDP is 40 MHz and the core clock is 120 MHz, the QSPI clock is 120/3 = 40 MHz.

Appendix I Power Down Step

To power down, complete the following steps:

1. Put the DA16200/DA16600 into Sleep mode 2 or Sleep mode 3. See `dpm_sleep_start_mode_2()` or `dpm_sleep_start_mode_3()` used in Section 17.14 APIs or sleep AT commands (AT+SETSLEEP2EXT=0,0 or AT+SETSLEEP3EXT=10000) in Ref. [7].
2. Then, put the DA16200 to Sleep mode 1 using RTC Power Key.

19. Revision History

Revision	Date	Description
2.5	Apr 16, 2025	▪ Updated TLS certificate in Selection 6. ▪ Added the note for network latency and HTTP timeout in Section 15.4.4 ▪ Added Appendix C.2 and Appendix H. ▪ Updated loady command for FBOOT in Section 18.2.4.2 ▪ Added Section 3.7.2. ▪ Added Section 13.3 and updated Section 13.4.4.
2.4	May 17, 2024	▪ DA16600 - Change cli command name "proxm_sensor_gw" to "monitor" ▪ DA16600 - Change Bluetooth® image folder name to DA14531_P (from DA14531_1) and DA14531_C (from DA14531_2) ▪ Section 18.3 title updated
2.3	Mar. 21, 2024	▪ Added Wi-Fi Functionality ▪ Added UM-WI-052 DA16600 FreeRTOS Example Application Manual ▪ Added Wake-up sources
2.2	Aug. 18, 2023	▪ Updated MCU transmission protocol changes in Section 15.6.2 ▪ Added Certificate API in Section 6 ▪ Added parameters into Section 9.1.1 • DA16X_CONF_INT_HIDDEN_0 • DA16X_CONF_INT_AUTH_MODE_0 • DA16X_CONF_INT_EAP_PHASE1_0 • DA16X_CONF_INT_EAP_PHASE2_0 ▪ Added parameters into Section 9.1.2 • DA16X_CONF_STR_EAP_IDENTITY • DA16X_CONF_STR_EAP_PASSWORD ▪ Added WPA enterprise Sample code into Section 9.1.3 ▪ Added Watchdog service in Section 8 ▪ Changed IDE to e^2 studio ▪ Changed description about fcCSP Low Power RTOS Image in Section 3.7.1
2.1	May. 31, 2023	Added missing section of the Introduction 3
2.0	May. 19, 2023	▪ Merged MQTT Programmer Guide, FreeROTS OTA Update, and FreeRTOS Example Application Manual ▪ Added the details for Bluetooth® LE Coexistence part in Section 17.13 ▪ Modified wake-up status in Table 62
1.9	Jan. 04 2023	▪ Added Bluetooth® LE Coexistence part in Section 17.13 ▪ Memory map updated: adjustable Bluetooth® LE firmware size and user area in DA16600 ▪ Updated Figures of Eclipse IDE screen capture
1.8	Oct. 24, 2022	Added TX power setting value range and step in Appendix B.1
1.7	Aug. 11, 2022	Changed TLS certificate area index number of SFlash area : #0, #1, #2, #3 → #1, #2, #3, #4 in Section 15.2
1.6	Jun. 14, 2022	▪ Updated company name of Reference documents ▪ Updated Sflash memory map for the DA16200/DA16600in Section 15.2 ▪ Updated TX power table programming in Appendix B.2
1.5	Mar. 28, 2022	Updated logo, disclaimer, and copyright.
1.4	Dec. 22, 2021	Added description about fcCSP Low Power RTOS Image.
1.3	Nov. 26, 2021	The title was changed.

Revision	Date	Description
1.2	Nov. 09, 2021	TW Editorial.
1.1	Oct. 25, 2021	Added description about OTP.
1.0	Apr. 13, 2021	First Release.

Status Definitions

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

RoHS Compliance

Renesas Electronics' 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.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/