# GR551x Second Boot Example Application

**Version: 2.0**

**Release Date: 2022-02-20**

**Shenzhen Goodix Technology Co., Ltd.**

## Trademarks and Permissions

## Disclaimer

Information contained in this document is intended for your convenience only and is subject to change without prior notice. It is your responsibility to ensure its application complies with technical specifications.

Shenzhen Goodix Technology Co., Ltd. (hereafter referred to as “Goodix”) makes no representation or guarantee for this information, express or implied, oral or written, statutory or otherwise, including but not limited to representation or guarantee for its application, quality, performance, merchantability or fitness for a particular purpose. Goodix shall assume no responsibility for this information and relevant consequences arising out of the use of such information.

Without written consent of Goodix, it is prohibited to use Goodix products as critical components in any life support system. Under the protection of Goodix intellectual property rights, no license may be transferred implicitly or by any other means.

**Shenzhen Goodix Technology Co., Ltd.**

Headquarters: 2F. & 13F., Tower B, Tengfei Industrial Building, Futian Free Trade Zone, Shenzhen, China

TEL: +86-755-33338828 FAX: +86-755-33338099

Website: www.goodix.com

# Preface

### Purpose

This document introduces how to use and verify the Second Boot example in the GR551x SDK, to help users quickly get started with secondary development.

### Audience

This document is intended for:

- GR551x user
- GR551x developer
- GR551x tester
- Hobbyist developer

### Release Notes

This document is the sixth release of *GR551x Second Boot Example Application*, corresponding to GR551x System-on-Chip (SoC) series.

### Revision History

| Version | Date | Description |
|---|---|---|
| 1.5 | 2020-08-30 | Initial release |
| 1.6 | 2020-11-25 | • Added operations required before downloading *ble_tem_dfu.bin* in “Firmware Download”.<br>• Added operations required before and after OTA DFU by using the Second Boot example, and described the subsequent influences in “Second Boot OTA”.<br>• Described the operations to recompile firmware by enabling the Second Boot mode in Keil in “Validity Check, Redirection, and Operation of Application Firmware”. |
| 1.7 | 2020-12-25 | • Added description on operations before downloading *second_boot.bin* in “Firmware Download”.<br>• Added description in “ Checking Validity, Jumping to, and Running Application Firmware”.<br>• Added an FAQ about wake-up and warm boot failure of the application firmware. |
| 1.8 | 2021-04-19 | • Updated parameters in *user_config.h* in “ Firmware Download”.<br>• Added “Custom Strategies for Firmware Update, Verification, and Jumping”. |
| 1.9 | 2021-12-29 | • Deleted the "Hardware Connection" section,<br>• Updated the "Firmware Download" section. |
| 2.0 | 2022-02-20 | Modified the file name of the example firmware based on Software Development Kit (SDK) changes. |

# Contents

# 1 Introduction

The Second Boot example performs the functions of device firmware update (DFU), checking validity, jumping to, and running application firmware, as well as secure signature verification over Bluetooth Low Energy (Bluetooth LE) transmission and second boot of firmware, providing users with flexible, reliable, and secure Over-the-Air (OTA) functions.

- Background dual-bank DFU by copying the firmware: Update the firmware by copying the firmware from one bank to another through OTA over Bluetooth LE transmission.
- Checking validity, jumping to, and running application firmware: Compare the application firmware information with the information in APP Image Info. Jump to and run the application firmware (*ble_tem_dfu.bin* is used as an example in this document) if the information from the two sources matches.
- Secure verification: Sign the firmware to protect it against tampering and achieve non-repudiation. The Second Boot example verifies the signature before update.

Before getting started, you can refer to the following documents.

Table 1-1 Reference documents

| **Name** | **Description** |
|---|---|
| GR551x Developer Guide | Introduces GR551x Software Development Kit (SDK) and how to develop and debug applications based on the SDK. |
| GR551x DFU Application Note | Introduces the principles and methods of Device Firmware Update for GR551x System-on-Chips (SoCs). |
| GR551x OTA Example Application | Introduces how to implement Over The Air for GR551x firmware on GRToolbox. |
| GProgrammer User Manual | Lists GProgrammer operational instructions, including downloading firmware to and encrypting firmware on GR551x SoCs. |
| GR551x Firmware Encryption Application Note | Introduces how to encrypt data and firmware of GR551x SoCs. |
| J-Link/J-Trace User Guide | Provides J-Link operational instructions. Available at http://www.segger.com/downloads/jlink/UM08001_JLink.pdf. |
| Keil User Guide | Offers detailed Keil operational instructions. Available at https://www.keil.com/support/man/docs/uv4/. |

# 2 Flash Layout

The Flash layout of the GR551x Second Boot example is shown in the figure below.

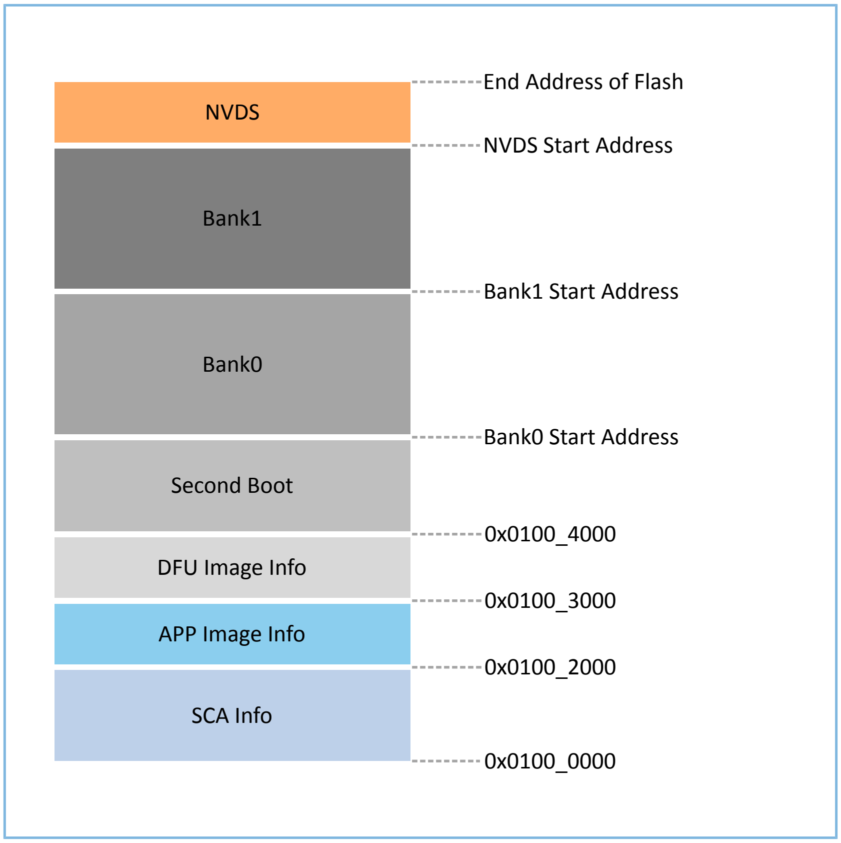


Figure 2-1 Flash layout of Second Boot example

- SCA Info: an area to store system information and the boot information of the Second Boot example
- APP Image Info: an area to store the operation settings for application firmware
- DFU Image Info: an area to store information about the firmware for DFU, which is used to check the validity of the firmware to be copied
- Second Boot: an area that stores the Second Boot example and in which the example is implemented
- Bank0: an area that stores the application firmware and in which the example is implemented
- Bank1: an area that buffers the firmware for DFU; the firmware that passes the validity check will be copied to Bank0.
- NVDS: Non-volatile Data Storage area

# 3 Initial Operation

This chapter introduces how to run and verify the GR551x Second Boot example in the GR551x SDK.

**Note**:

SDK_Folder is the root directory of GR551x SDK.

## 3.1 Preparation

Perform the following tasks before running the Second Boot example.

- **Hardware preparation**

Table 3-1 Hardware preparation

| Name | Description |
|---|---|
| J-Link debug probe | JTAG emulator launched by SEGGER. For more information, visit http://www.segger.com/products/debug-probes/j-link/. |
| Development board | GR5515 Starter Kit Board (SK Board) |
| Connection cable | A micro USB 2.0 serial cable |
| Android phone | A phone running on Android 5.0 (KitKat) or later versions |

- **Software preparation**

Table 3-2 Software preparation

| Name | Description |
|---|---|
| Windows | Windows 7/Windows 10 |
| J-Link driver | A J-Link driver. Available at www.segger.com/downloads/jlink/. |
| Keil MDK5 | An integrated development environment (IDE). MDK-ARM Version 5.20 or later is required. Available at www.keil.com/download/product/. |
| GProgrammer (Windows) | A programming tool. Available in `SDK_Folder\tools\GProgrammer`. |
| GRUart (Windows) | A serial port debugging tool. Available in `SDK_Folder\tools\GRUart`. |
| GRToolbox (Android) | A Bluetooth LE debugging tool. Available in `SDK_Folder\tools\GRToolbox`. |

## 3.2 Firmware Programming

To get started, users shall first erase the Flash memory in the GR551x SoC with GProgrammer, and then programme *second_boot.bin* and *ble_tem_dfu.bin* to the SK Board.

Before programming the firmware, it is required to:

- For *ble_tem_dfu.bin*: Enable USE_SECOND_BOOT_MODE in Keil (for details, see "Section 3.3.2 Validity Check, Redirection, and Operation of Application Firmware"). Then, recompile the firmware file before programming it to the SK Board.

- For *second_boot.bin*: Configure *user_config.h* (available in `SDK_Folder\projects\ble\dfu\second_boot\Src\config`), to set the parameters and hash values of the public key. After the configuration completes, recompile the firmware file before programming it to the SK Board.

Table 3-3 Parameters in *user_config.h*

| Macro | Description |
|---|---|
| BOOTLOADER_DEFAULT_STRATEGY_ENABLE | Use the default firmware for update, verification, and jumping strategies or not.<br>• 0: Use the custom firmware.<br>• 1: Use the default firmware. |
| BOOTLOADER_WDT_ENABLE | Enable the WDT of the Second Boot example or not.<br>• 0: Disable<br>• 1: Enable |
| BOOTLOADER_OTA_ENABLE | Enable Second Boot OTA or not.<br>• 0: Disable<br>• 1: Enable |
| BOOTLOADER_SIGN_ENABLE | Enable the signing and verification solution for the Second Boot example or not, valid when BOOTLOADER_DEFAULT_STRATEGY_ENABLE is set to 1.<br>• 0: Disable<br>• 1: Enable<br>**Note:**<br>Refer to "Section 3.3.3 Secure Signature Verification" for details about enabling the secure verification function. |
| USER_FW_COMMENTS | Define the application firmware comments, valid when BOOTLOADER_DEFAULT_STRATEGY_ENABLE is set to 1. Compare the information in the application firmware comments to search for the Image Info of the firmware. The macro is up to 12 bytes. The current default value is "ble_tem_dfu_". |
| APP_FW_RUN_ADDRESS | The run address of application firmware, valid when BOOTLOADER_DEFAULT_STRATEGY_ENABLE is set to 0.<br>**Note:**<br>See "Section 4.2.3 Custom Strategies for Firmware Update, Verification, and Jumping" for details |

For details about using GProgrammer, see *GProgrammer User Manual*.

**Note**:

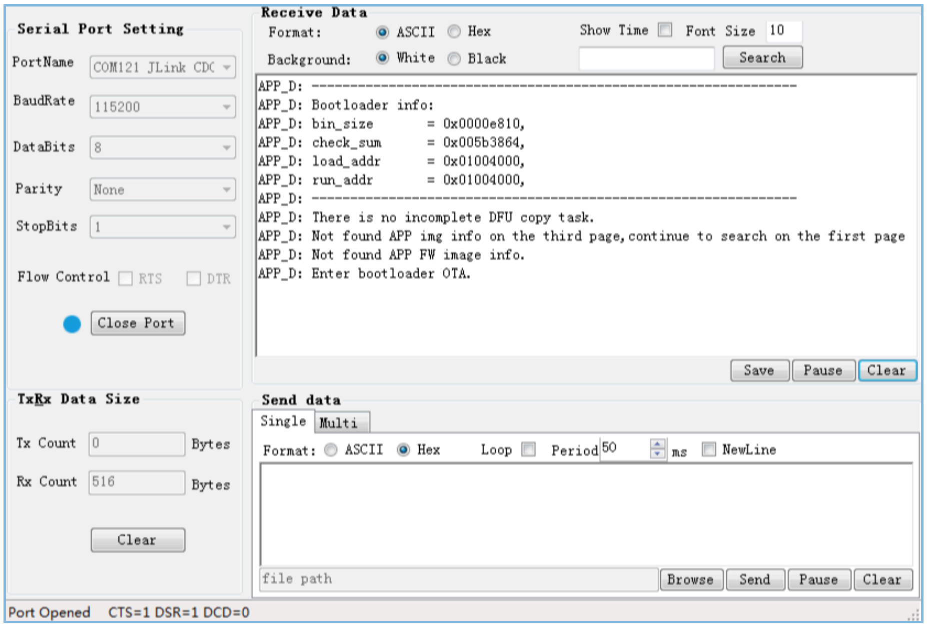
1. *second_boot.bin* is in `SDK_Folder\projects\ble\dfu\second_boot\build\`. The default run address is 0x01004000.
2. *ble_tem_dfu.bin* is in `SDK_Folder\projects\ble\ble_peripheral\ble_app_template_dfu\build`. The default run address is 0x01040000.
3. If the run addresses of *second_boot.bin* and *ble_tem_dfu.bin* need to be modified, make sure no conflict exists in the memory addresses of the two pieces of firmware.
4. If users prefer custom strategies for firmware update (by copying the firmware), verification, and jumping, set BOOTLOADER_DEFAULT_STRATEGY_ENABLE to 0, and implement vendor_fw_copy_update() and vendor_fw_verify() based on customization.

## 3.3 Test and Verification

This section explains how to quickly verify the Second Boot example by introducing Second Boot OTA, checking validity, jumping to, and running application firmware, as well as secure signature verification.

### 3.3.1 Second Boot OTA

1. Before programming *second_boot.bin* to the SK Board with GProgrammer, erase the Flash memory of the GR551x SoC with GProgrammer, to make sure no OTA copying task or application firmware is in the Flash memory.
2. Programme *second_boot.bin* to the SK Board, and enter the OTA process for Second Boot to wait for firmware update (see Step 3 in "Section 4.2 Interaction Process and Main Code" for the mechanism). The interface of GRUart is shown as below.



PortName: COM121 JLink CDC
BaudRate: 115200
DataBits: 8
Parity: None
StopBits: 1

Tx Count: 0 Bytes
Rx Count: 516 Bytes

Format: ● ASCII ○ Hex
Background: ● White ○ Black
Show Time □ Font Size 10

```
APP_D: ------------------------------------------------------------
APP_D: Bootloader info:
APP_D: bin_size       = 0x0000e810,
APP_D: check_sum      = 0x005b3864,
APP_D: load_addr      = 0x01004000,
APP_D: run_addr       = 0x01004000,
APP_D: ------------------------------------------------------------
APP_D: There is no incomplete DFU copy task.
APP_D: Not found APP img info on the third page,continue to search on the first page
APP_D: Not found APP FW image info.
APP_D: Enter bootloader OTA.
```

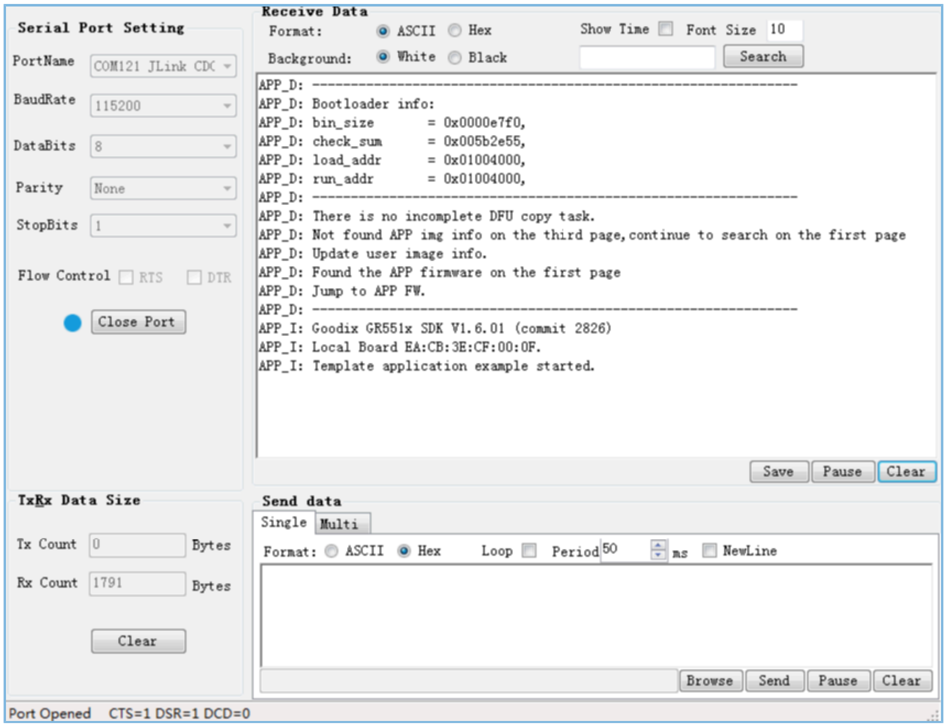
Format: ○ ASCII ● Hex Loop □ Period 50 ms □ NewLine

Port Opened CTS=1 DSR=1 DCD=0

Figure 3-1 Entering OTA process after Flash erase

3. Turn on Bluetooth on the Android phone and open GRToolbox. Scan for devices, and if “Goodix_Boot” is discovered, it means the Second Boot firmware runs normally.

4. Bluetooth LE OTA function is integrated in the Second Boot firmware. For details about OTA, see “Update Firmware in ble_app_template_dfu” in *GR551x OTA Example Application*. After firmware update completes, the system automatically jumps to and runs the newly updated firmware. The interface of GRUart is shown as below.



Serial Port Setting

PortName: COM121 JLink CDC

BaudRate: 115200

DataBits: 8

Parity: None

StopBits: 1

Flow Control: □ RTS □ DTR

Receive Data

Format: ● ASCII ○ Hex  Show Time □  Font Size 10

Background: ● White ○ Black

```
APP_D: -----------------------------------------------------------------
APP_D: Bootloader info:
APP_D: bin_size       = 0x0000e7f0,
APP_D: check_sum      = 0x005b2e55,
APP_D: load_addr      = 0x01004000,
APP_D: run_addr       = 0x01004000,
APP_D: -----------------------------------------------------------------
APP_D: There is no incomplete DFU copy task.
APP_D: Not found APP img info on the third page,continue to search on the first page
APP_D: Update user image info.
APP_D: Found the APP firmware on the first page
APP_D: Jump to APP FW.
APP_D: -----------------------------------------------------------------
APP_I: Goodix GR551x SDK V1.6.01 (commit 2826)
APP_I: Local Board EA:CB:3E:CF:00:0F.
APP_I: Template application example started.
```

TxRx Data Size

Tx Count: 0 Bytes

Rx Count: 1791 Bytes

Send data

Format: ○ ASCII ● Hex  Loop □  Period 50 ms  □ NewLine

Port Opened  CTS=1 DSR=1 DCD=0

Figure 3-2 Firmware running after successful update

**Note**:

- During OTA DFU in Second Boot mode, check **Copy Mode** on the **DFU** page in GRToolbox. Then, contents in the area specified by **Copy Address** will be overwritten. Therefore, improper configuration will lead to loss of the original information in this area.
- After OTA DFU in Second Boot mode, the updated firmware information will not be displayed in GProgrammer.

## 3.3.2 Validity Check, Redirection, and Operation of Application Firmware

1. Erase the Flash memory of the GR551x SoC with GProgrammer, to make sure no OTA copying task or application firmware is in the Flash memory.

2. Modify the example project ble_app_template_dfu in Keil, and then recompile the firmware:

   (1). Enter the directory of example project `SDK_Folder\projects\ble\ble_peripheral\ble_app_template_dfu\Keil_5`. Double-click *ble_app_template_dfu.uvprojx* to open the example project in Keil.

(2). Click (**Options for Target**) on Keil toolbar. Then, choose the **C/C++** tab in the popped up window **Options for Target 'GR551x_SK'**.

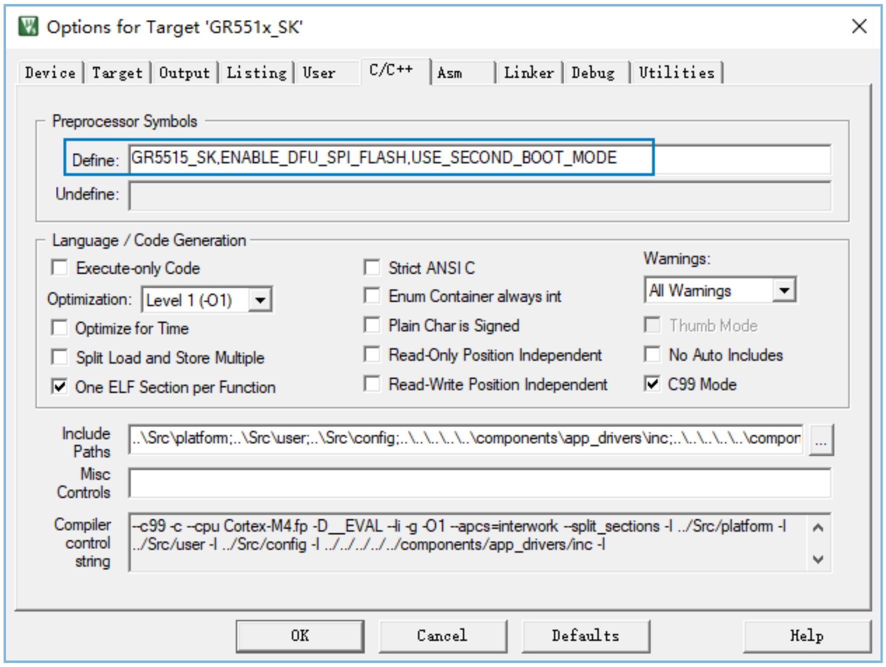
(3). Add **USE_SECOND_BOOT_MODE** in the **Define** field in the **Preprocessor Symbols** area.

---

**Note**:

The added **USE_SECOND_BOOT_MODE** shall be separated from the previous macro with a comma.

---



Define: GR5515_SK,ENABLE_DFU_SPI_FLASH,USE_SECOND_BOOT_MODE

Undefine:

□ Execute-only Code

Optimization: Level 1 (-O1)

□ Optimize for Time

□ Split Load and Store Multiple

☑ One ELF Section per Function

□ Strict ANSI C

□ Enum Container always int

□ Plain Char is Signed

□ Read-Only Position Independent

□ Read-Write Position Independent

Warnings: All Warnings

□ Thumb Mode

□ No Auto Includes

☑ C99 Mode

Include Paths: ..\Src\platform;..\Src\user;..\Src\config;..\..\..\..\..\components\app_drivers\inc;..\..\..\..\..\compor

Misc Controls:

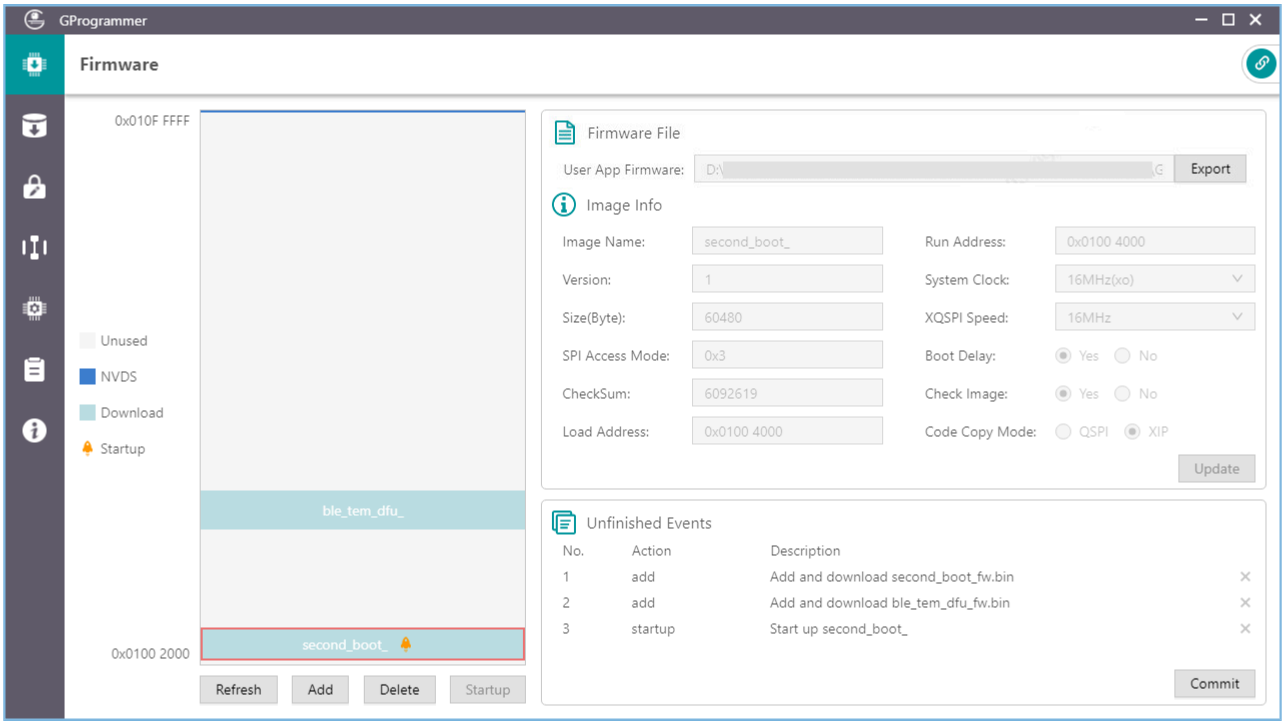
Compiler control string:

```
--c99 -c --cpu Cortex-M4.fp -D__EVAL -li -g -O1 --apcs=interwork --split_sections -I ../Src/platform -I
../Src/user -I ../Src/config -I ../../../../../components/app_drivers/inc -I
```

Figure 3-3 To enable the Second Boot mode

(4). After saving the settings, click on the Keil toolbar to compile the example project, and generate a .bin file.

3. Over GProgrammer, download *second_boot.bin* and *ble_tem_dfu.bin* to the SK Board, and set *second_boot.bin* for startup.



Firmware File

User App Firmware: D:\[redacted]G

Image Info

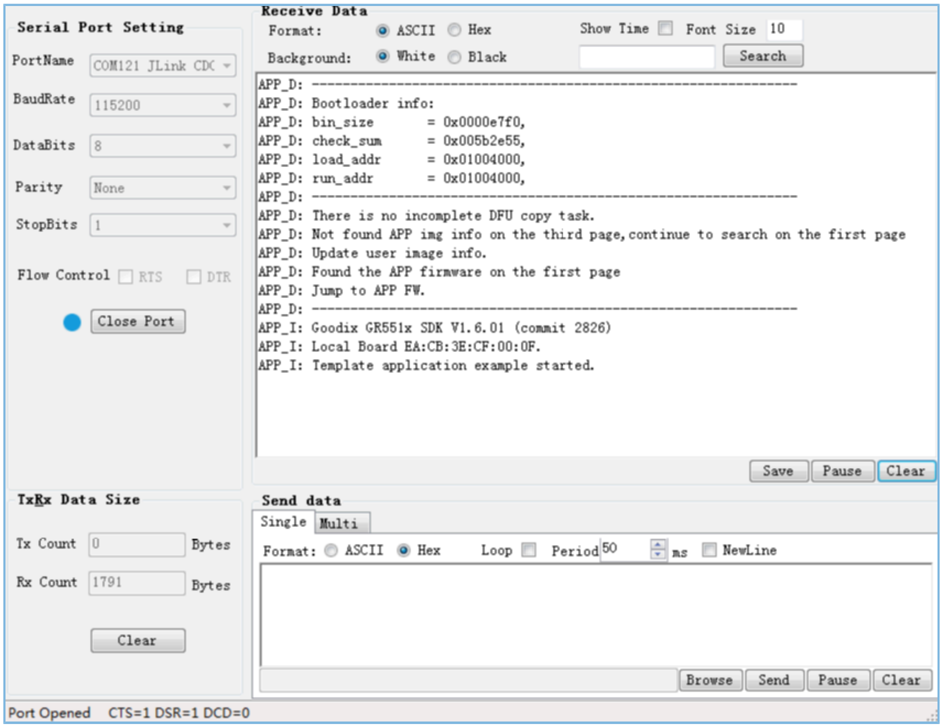
| | | | |
|---|---|---|---|
| Image Name: | second_boot_ | Run Address: | 0x0100 4000 |
| Version: | 1 | System Clock: | 16MHz(xo) |
| Size(Byte): | 60480 | XQSPI Speed: | 16MHz |
| SPI Access Mode: | 0x3 | Boot Delay: | ● Yes ○ No |
| CheckSum: | 6092619 | Check Image: | ● Yes ○ No |
| Load Address: | 0x0100 4000 | Code Copy Mode: | ○ QSPI ● XIP |

Unfinished Events

| No. | Action | Description |
|---|---|---|
| 1 | add | Add and download second_boot_fw.bin |
| 2 | add | Add and download ble_tem_dfu_fw.bin |
| 3 | startup | Start up second_boot_ |

Figure 3-4 Choosing *second_boot.bin* for startup

4. *ble_tem_dfu.bin* is detected when the GR551x SoC is started. After the firmware passes validity check, the GR551x SoC jumps to the start address of the application firmware and starts to run the firmware. The interface of GRUart is shown as below.



| | |
|---|---|
| PortName | COM121 JLink CDC |
| BaudRate | 115200 |
| DataBits | 8 |
| Parity | None |
| StopBits | 1 |
| Flow Control | □ RTS □ DTR |
| Tx Count | 0 Bytes |
| Rx Count | 1791 Bytes |

Format: ● ASCII ○ Hex  Show Time □  Font Size 10

Background: ● White ○ Black

```
APP_D: ------------------------------------------------------------
APP_D: Bootloader info:
APP_D: bin_size        = 0x0000e7f0,
APP_D: check_sum       = 0x005b2e55,
APP_D: load_addr       = 0x01004000,
APP_D: run_addr        = 0x01004000,
APP_D: ------------------------------------------------------------
APP_D: There is no incomplete DFU copy task.
APP_D: Not found APP img info on the third page,continue to search on the first page
APP_D: Update user image info.
APP_D: Found the APP firmware on the first page
APP_D: Jump to APP FW.
APP_D: ------------------------------------------------------------
APP_I: Goodix GR551x SDK V1.6.01 (commit 2826)
APP_I: Local Board EA:CB:3E:CF:00:0F.
APP_I: Template application example started.
```

Format: ○ ASCII ● Hex  Loop □  Period 50 ms  □ NewLine

Port Opened  CTS=1 DSR=1 DCD=0

Figure 3-5 Application firmware running after successful update

### 3.3.3 Secure Signature Verification

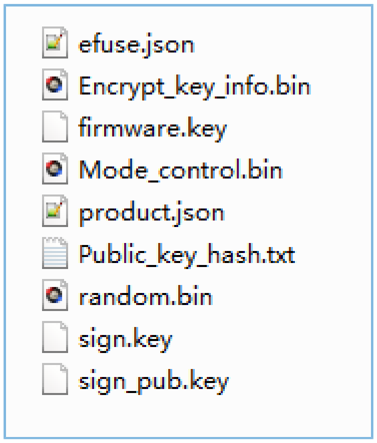
Secure signature verification on OTA firmware is supported in the Second Boot example. Users can choose to enable/ disable the function as needed. To enable the function, set BOOTLOADER_SIGN_ENABLE = 1 in *user_config.h* in the project directory of the Second Boot example.

Before signature verification, users can sign the application firmware by using GProgrammer. The process for signing and verification is as follows:

1. Generate the hash values of the public key and the private key.

   For operations about generating signatures, see “Encrypt & Sign” in *GProgrammer User Manual*. For related mechanisms, see “Digital Signature” in *GR551x Firmware Encryption Application Note*.

   The files used for encryption and signing generated through GProgrammer are shown as below:

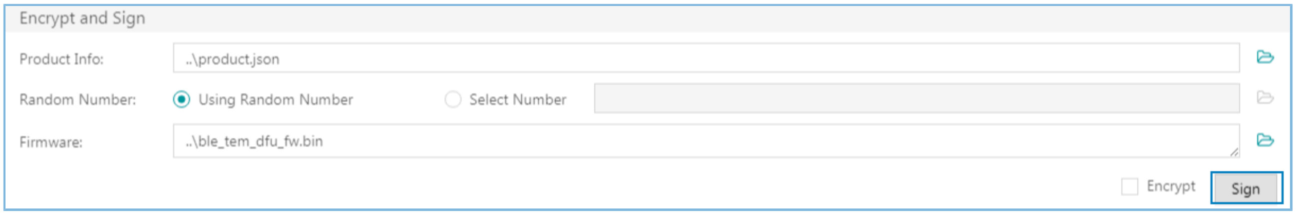


efuse.json
Encrypt_key_info.bin
firmware.key
Mode_control.bin
product.json
Public_key_hash.txt
random.bin
sign.key
sign_pub.key

Figure 3-6 Files used for encryption and signing

2. Sign the firmware.

   Import *product.json* and *ble_tem_dfu.bin*, and click **Sign**, as shown in Figure 3-7.



Encrypt and Sign

Product Info: ..\product.json

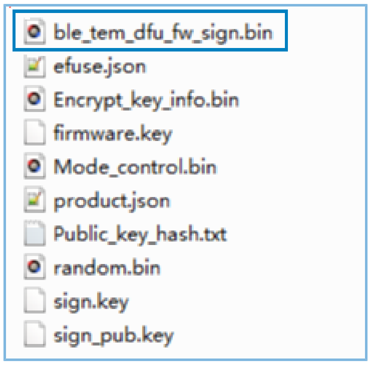
Random Number: ● Using Random Number ○ Select Number

Firmware: ..\ble_tem_dfu_fw.bin

□ Encrypt

Figure 3-7 Signing the application firmware

   Specify the path for signed files, and the signed application firmware file is generated (the one whose file name ends with _sign, which is *ble_tem_dfu_fw_sign.bin* in this example), as shown in Figure 3-8:



ble_tem_dfu_fw_sign.bin
efuse.json
Encrypt_key_info.bin
firmware.key
Mode_control.bin
product.json
Public_key_hash.txt
random.bin
sign.key
sign_pub.key

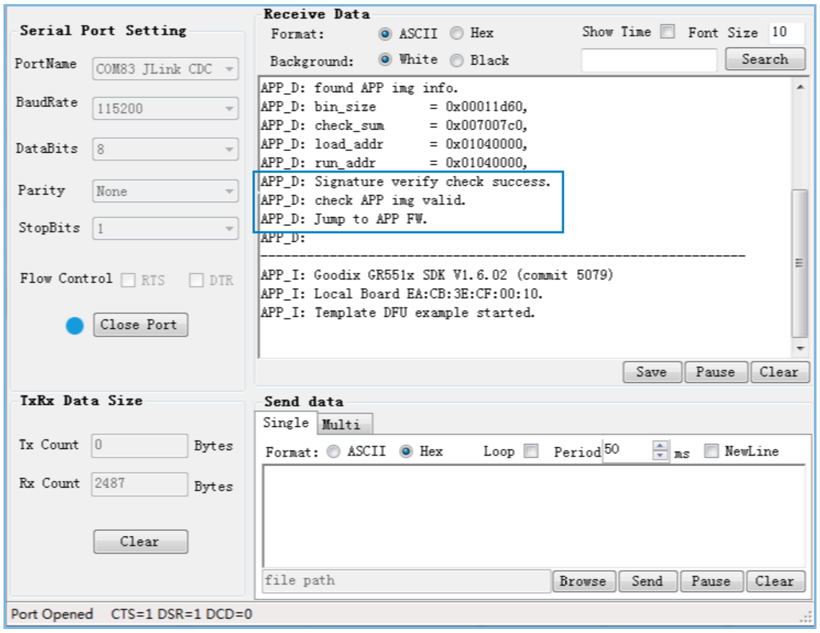
Figure 3-8 Signed firmware file

3. Copy the hash value of the public key in *Public_key_hash.txt* to the public_key_hash array in *user_config.h* and re-compile *second_boot.bin*.

```
//Hash value of the signed public key
static const uint8_t public_key_hash[] =
{
    0x08,0x57,0x41,0xDD,0x34,0x17,0x0C,0x01,0x43,0xFB,0xCA,0xA5,0x5C,0x51,0x81,0xF5
};
```

4. Verify the signed firmware.

   Download the recompiled *second_boot.bin* file and the signed *ble_tem_dfu_fw_sign.bin* file to the SK Board; set *second_boot.bin* as the firmware for startup, and run the firmware. The Second Boot firmware checks and verifies the signed *ble_tem_dfu_sign.bin* file. When the application firmware passes the checking and verification, the system jumps to and runs the application firmware. GRUart shows as follows:



PortName: COM83 JLink CDC
BaudRate: 115200
DataBits: 8
Parity: None
StopBits: 1
Flow Control: □ RTS □ DTR
Format: ● ASCII ○ Hex  Show Time □  Font Size 10
Background: ● White ○ Black

```
APP_D: found APP img info.
APP_D: bin_size       = 0x00011d60,
APP_D: check_sum      = 0x007007c0,
APP_D: load_addr      = 0x01040000,
APP_D: run_addr       = 0x01040000,
APP_D: Signature verify check success.
APP_D: check APP img valid.
APP_D: Jump to APP FW.
APP_D:
----------------------------------------------------------------
APP_I: Goodix GR551x SDK V1.6.02 (commit 5079)
APP_I: Local Board EA:CB:3E:CF:00:10.
APP_I: Template DFU example started.
```

Tx Count: 0 Bytes
Rx Count: 2487 Bytes
Format: ○ ASCII ● Hex  Loop □  Period 50 ms  □ NewLine

Port Opened  CTS=1 DSR=1 DCD=0

Figure 3-9 Verifying the signed firmware

# 4 Application Details

This chapter introduces the project directory, the main interaction processes, and related code of the Second Boot example.

## 4.1 Project Directory

The source code and the project file of the Second Boot example are in `SDK_Folder\projects\ble\dfu\second_boot\Keil_5`.

Double-click the project file, *second_boot.uvprojx*, to check the project directory structure of the Second Boot example in Keil. Related files are described in Table 4-1.

Table 4-1 Project files of Second Boot example

| Group | File | Description |
|---|---|---|
| gr_profiles | otas.c | Implements OTA Service. |
| user_callback | user_gap_callback.c | Implements GAP callbacks, such as connection, disconnection, and GAP parameter update. |
| | user_gatt_common_callback.c | Implements GATT common callbacks, such as MTU update. |
| user_platform | user_periph_setup.c | Configures APP logs and the WDT. |
| user_app | main.c | Contains the main() function. |
| | user_app.c | Initializes OTA Service and handles Bluetooth LE events. |
| | user_dfu.c | Initializes the DFU service. |
| | user_boot.c | Checks the validity of firmware and enables jumping to the firmware. |
| | sign_verify.lib | This is the static library that verifies firmware signatures. |
| | user_config.h | Configures WDT and firmware signature verification. |

## 4.2 Interaction Process and Main Code

This section introduces the process and the critical code for copying and upgrading the firmware for DFU, checking, jumping to, and running the application firmware, to help users better understand the operating mechanism of the Second Boot example.

The process for running the Second Boot example is shown in Figure 4-1.

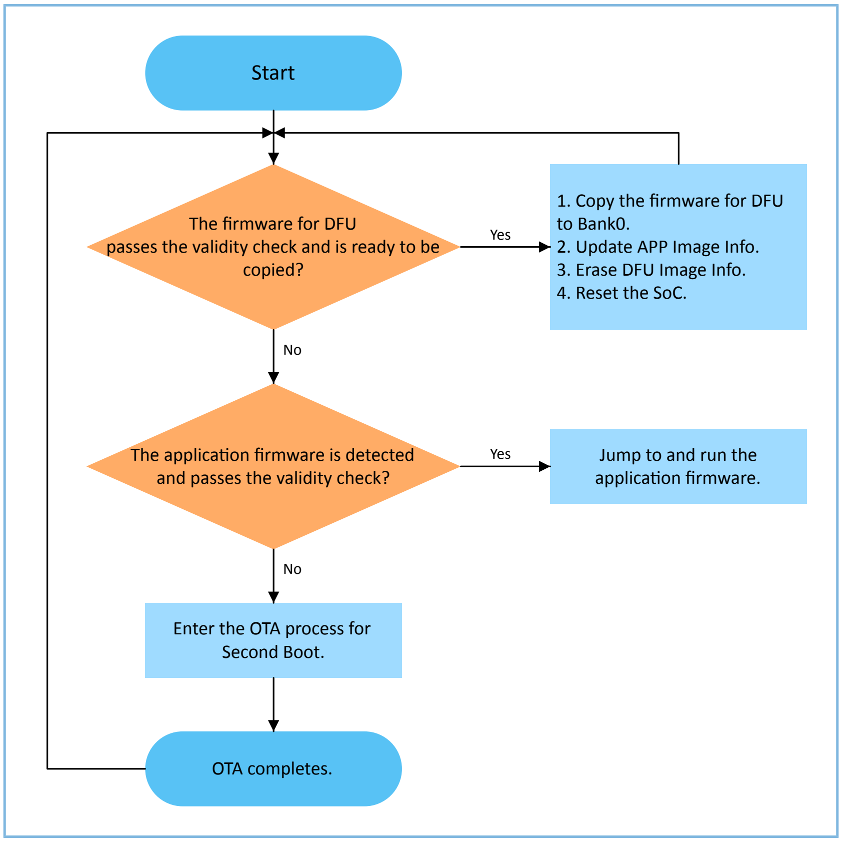

Start
The firmware for DFU passes the validity check and is ready to be copied?
Yes
1. Copy the firmware for DFU to Bank0.
2. Update APP Image Info.
3. Erase DFU Image Info.
4. Reset the SoC.
No
The application firmware is detected and passes the validity check?
Yes
Jump to and run the application firmware.
No
Enter the OTA process for Second Boot.
OTA completes.

Figure 4-1 Process for running Second Boot example

1. Read DFU Image Info. When firmware for DFU in Bank1 needs to be copied to Bank0, and the firmware has passed validity check, proceed to Step 2. Otherwise, proceed to Step 3.
2. Copy the firmware for DFU from Bank1 to Bank0. After updating APP Image Info and erasing DFU Image Info, reset the GR551x SoC.
3. Read APP Image Info. When application firmware is in Bank0 and the firmware has passed validity check, the system jumps to and runs the application firmware. If the firmware has not passed validity check, proceed to Step 4.
4. Enter Second Boot OTA. After OTA completes, update APP Image Info, and reset the GR551x SoC.

### 4.2.1 Copying Firmware for DFU

Application firmware of the GR551x SoC adopts dual-bank background update for OTA:

1. Save the firmware for DFU in Bank1, and update the related information in the DFU Image Info area;
2. Reset the GR551x SoC and run the Second Boot firmware, to copy the firmware for DFU from Bank1 to Bank0.

Code for copying the firmware for DFU is described below:

**Path:** `user_app\user_boot.c` under the project directory

**Name:** is_fw_need_copy();

is_fw_need_copy() is used to read DFU Image Info, to check whether any firmware copying task for DFU is waiting.

```
static bool is_fw_need_copy(void)
{
    copy_load_addr = 0;
    hal_flash_read_judge_security(IMG_INFO_DFU_ADDR, (uint8_t*)&copy_load_addr, 4);
    memset((uint8_t*)&dfu_img_info, 0, sizeof(img_info_t));
    hal_flash_read_judge_security(IMG_INFO_DFU_ADDR+4, (uint8_t*)&dfu_img_info,
 sizeof(img_info_t));

    if (dfu_img_info.pattern ! = 0x4744 ||\
        (memcmp(dfu_img_info.comments, USER_FW_COMMENTS, strlen(USER_FW_COMMENTS)) ! = 0))
    {
        APP_LOG_DEBUG("There is no incomplete DFU copy task." );
        return false;
    }

    APP_LOG_DEBUG("---------------------------------------------------------------");
    APP_LOG_DEBUG("copy addr      = 0x%08x", copy_load_addr);
    APP_LOG_DEBUG("DFU fw boot info:" );
    log_boot_info(&dfu_img_info.boot_info);
    APP_LOG_DEBUG("---------------------------------------------------------------");

    APP_LOG_DEBUG("There is incomplete DFU copy task." );
    return true;
}
```

**Path:** `user_app\user_boot.c` under the project directory

**Name:** incplt_dfu_task_continue();

incplt_dfu_task_continue() is used to check the validity of the firmware for DFU. After the firmware passes the validity check, copy the firmware from Bank1 to Bank0, update APP Image Info, and erase DFU Image Info. Reset the GR551x SoC. The system then jumps to and runs the firmware in Bank0. The code snippet is as follows:

```
static void incplt_dfu_task_continue(void)
{
    if (!boot_fw_valid_check(copy_load_addr, &dfu_img_info.boot_info))
    {
        APP_LOG_DEBUG("DFU FW image valid check fail." );
        return;
    }
    if(copy_load_addr ! = dfu_img_info.boot_info.load_addr)
    {
        uint32_t copy_size = dfu_img_info.boot_info.bin_size + 48;
        APP_LOG_DEBUG("DFU FW image start copy." );

        if(sys_security_enable_status_check())
        {
            copy_size += 856;
        }
        else
        {
        #if BOOTLOADER_SIGN_ENABLE
            copy_size += 856;
        #endif
        }
        dfu_fw_copy(dfu_img_info.boot_info.load_addr, copy_load_addr, copy_size);
    }
    user_img_info_update(&dfu_img_info);
    hal_flash_erase(IMG_INFO_DFU_ADDR, CODE_PAGE_SIZE);//clear copy info
hal_nvic_system_reset();
}
```

### 4.2.2 Checking Validity, Jumping to, and Running Application Firmware

When no firmware copying task is waiting, the Second Boot example checks the validity of the application firmware, and jumps to and runs the firmware if it passes the validity check.

**Path:** `user_app\user_boot.c` under the project directory

**Name:** is_jump_user_fw();

is_jump_user_fw() is used to check the validity of the application firmware before the system jumps to and runs the firmware.

is_jump_user_fw() reads the comments in APP Image Info and compares the comments with those of the application firmware (USER_FW_COMMENTS).

If the comments from the two sources are the same, it means the application firmware has been copied to Bank0. Then, check the validity of the application firmware in APP Image Info, and the system jumps to and runs the firmware after it passes the validity check.

If the comments from the two sources are different, it means the application firmware has not been copied to Bank0. In this case, search for and read comments in the SCA Info area, and compare the comments with USER_FW_COMMENTS. If comments from the two sources are the same, check the validity of the application firmware in the SCA Info area. If the firmware passes validity check, update the application firmware in APP Image Info with the firmware in SCA Info. If comments from the two sources are different, or the validity check fails, the system cannot jump to the application firmware.

```
static bool is_jump_user_fw(void)
{
    memset((uint8_t*)&app_img_info, 0, sizeof(img_info_t));
    hal_flash_read_judge_security(IMG_INFO_APP_ADDR, (uint8_t*)&app_img_info,
 sizeof(img_info_t));

    if ((app_img_info.pattern == 0x4744) &&\
        (0 == memcmp(app_img_info.comments, USER_FW_COMMENTS, strlen(USER_FW_COMMENTS))))
    {
        APP_LOG_DEBUG("found APP img info." );
        log_boot_info(&app_img_info.boot_info);
        if (boot_fw_valid_check(app_img_info.boot_info.load_addr, &app_img_info.boot_info))
        {
            APP_LOG_DEBUG("check APP img valid." );
            return true;
        }
    }
    APP_LOG_DEBUG("Not found APP img info on the third page,continue to search on the first
 page");

    img_info_t img_info_main;
    for (uint8_t i = 0; i < IMG_INFO_SAVE_NUM_MAX; i++)
    {
        fw_img_info_get(BOOT_INFO_ADDR + 0x40, i, &img_info_main);

        if (0 == memcmp(img_info_main.comments, USER_FW_COMMENTS, strlen(USER_FW_COMMENTS)))
        {
            if (boot_fw_valid_check(img_info_main.boot_info.load_addr,
 &img_info_main.boot_info))
```

```
            {
                user_img_info_update(&img_info_main);
                memcpy(&app_img_info, &img_info_main, sizeof(img_info_t));
                APP_LOG_DEBUG("Found the APP firmware on the first page");
                return true;
            }
        }
    }

    APP_LOG_DEBUG("Not found APP FW image info." );
    return false;
}
```

**Path:** `user_app\user_boot.c` under the project directory

**Name1:** jump_user_fw();

**Name2:** sec_boot_jump();

Before the system jumps to the firmware, it is required to update the information for warm boot, set the main stack pointer (MSP), and relocate the vector table.

```
static void jump_user_fw(void)
{
    APP_LOG_DEBUG("Jump to APP FW." );
    APP_LOG_DEBUG("------------------------------------------------------------------");
    sec_boot_jump(&app_img_info.boot_info);
}
 static void sec_boot_jump(boot_info_t *p_boot_info)
{
    extern void rom_init(void);
    extern void jump_app(uint32_t addr);
    extern boot_info_t bl1_boot_info;
    extern void bl_xip_dis(void);
    uint16_t enc_mode = *(uint16_t*)0x30000020;
    bool mirror_mode = false;

    if(p_boot_info->run_addr ! = p_boot_info->load_addr)//mirror mode
    {
        mirror_mode = true;
        if(!enc_mode)
            SET_CODE_LOAD_FLAG();
        memcpy((uint8_t*)p_boot_info->run_addr, (uint8_t*)p_boot_info->load_addr,
 p_boot_info->bin_size);
    }
    if(enc_mode)
    {
        REG(0xA000C578UL) &= ~0xFFFFFC00;
        REG(0xA000C578UL) |= (p_boot_info->run_addr & 0xFFFFFC00);
    }

    memcpy(&bl1_boot_info, p_boot_info, sizeof(boot_info_t));
    if(mirror_mode)
    {
        if(enc_mode)
        {
            REG(0xa000d470) = ENCRY_CTRL_DISABLE;
        }
    }
    sys_firmware_jump(p_boot_info->run_addr);
}
```

**Note**:

To ensure that the SK Board jumps to the application firmware directly upon the firmware warm boot after wake-up from the sleep mode, assign boot_info of the application firmware to the global variable bl1_boot_info: “memcpy(&bl1_boot_info, p_boot_info, sizeof(boot_info_t));”. The value shall not be modified.

### 4.2.3 Custom Strategies for Firmware Update, Verification, and Jumping

To use the custom strategies, set BOOTLOADER_DEFAULT_STRATEGY_ENABLE to 0, and implement vendor_fw_copy_update() and vendor_fw_verify() based on customization for firmware update (by copying the firmware) and verification, and vendor_fw_jump() (customization is not compulsory) to jump to the application firmware.

The three functions are available in `user_app\user_boot.c`, and all can be customized for extended functionalities.

# 5 FAQ

This chapter describes possible problems, reasons, and solutions when using and verifying the Second Boot example.

## 5.1 Why Does OTA DFU by Using the Second Boot Example Fail?

- Description

  When I perform OTA DFU by using the Second Boot example, signature verification fails.

- Analysis

  It fails to obtain the public key when users perform signature verification for firmware update.

- Solution

  Make sure the private key for signing pairs with the public key for verification. Copy the hash value of the public key in *Public_key_hash.txt* to the public_key_hash array in *user_config.h*.

## 5.2 Why Do I Fail to Wake up the Application Firmware from Sleep Mode?

- Description

  I cannot wake up the application firmware from the sleep mode when using the Second Boot example.

- Analysis

  The code in the Second Boot firmware file for firmware verification and jumping has been modified, and boot_info of the current application firmware has not been assigned to bl1_boot_info, resulting in warm boot failure from the sleep mode.

- Solution

  Assign boot_info of the application firmware to the global variable bl1_boot_info in sec_boot_jump().