GProgrammer User Manual

Version: 2.5

Release Date: 2022-02-20

Shenzhen Goodix Technology Co., Ltd.

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 install GProgrammer and operate its functional modules, enabling users to quickly get started with GProgrammer.

Audience

This document is intended for:

- GR551x user
- GR551x developer
- GR551x tester

Release Notes

This document is the eleventh release of *GProgrammer User Manual*, corresponding to GProgrammer V1.2.26.

Revision History

Version	Date	Description
1.5	2020-05-30	Initial release
1.6	2020-06-30	• Updated sector-related description in "Chip Configuration". • Added "GR551x_console.exe", introducing a command-line program to erase and download commands; added "GR551x_encrypt_signature.exe" and "User-defined Windows Scripts". • Introduced the public key hashes to verify signatures, updated the file name extension for encrypted and signed files, and introduced the firmware signing function in "Encrypt & Sign".
1.7	2020-08-30	• Introduced the GR5515I0ND System-on-Chip (SoC) for GR551x SoCs in "SoC/MCU Selection". • Changed icons for **Delete** and **Startup** in "Firmware".
1.8	2020-09-30	Added description on firmware download failure in "Download Firmware".
1.9	2020-11-26	Updated UI screenshots for software version.
2.0	2021-01-05	Updated software UI screenshots for SoC/MCU selection and firmware operations.
2.1	2021-03-02	• Added file modification description to "eFuse Settings". • Added file export description to "Import and Export". • Updated descriptions concerning operations prior to viewing device logs in "Device Log". • Added description of **IO_LDO_SEL** field to "eFuse Layout". • Deleted the parameter of nvds in erase and download commands in "GR551x_console.exe".
2.2	2021-05-13	Deleted functionalities for GMF03x series.
2.3	2021-07-16	Updated software UI figures for SoC selection.

Version	Date	Description
2.4	2021-09-06	Updated software UI figures for SoC selection.
2.5	2022-02-20	• Updated the “Firmware” and “Encrypt & Sign” sections where adding HEX firmware files via GProgrammer is supported. • Updated the "Firmware" section where **Export** is added to the **Firmware** interface and exporting BIN firmware files by GProgrammer is supported. • Modified the "External Flash" section where **QSPI2** is added to **SPI Type** and relevant configuration items are modified. • Updated commands supported by *GR5xxx_console.exe* and *GR5xxx_encrypt_signature.exe*.

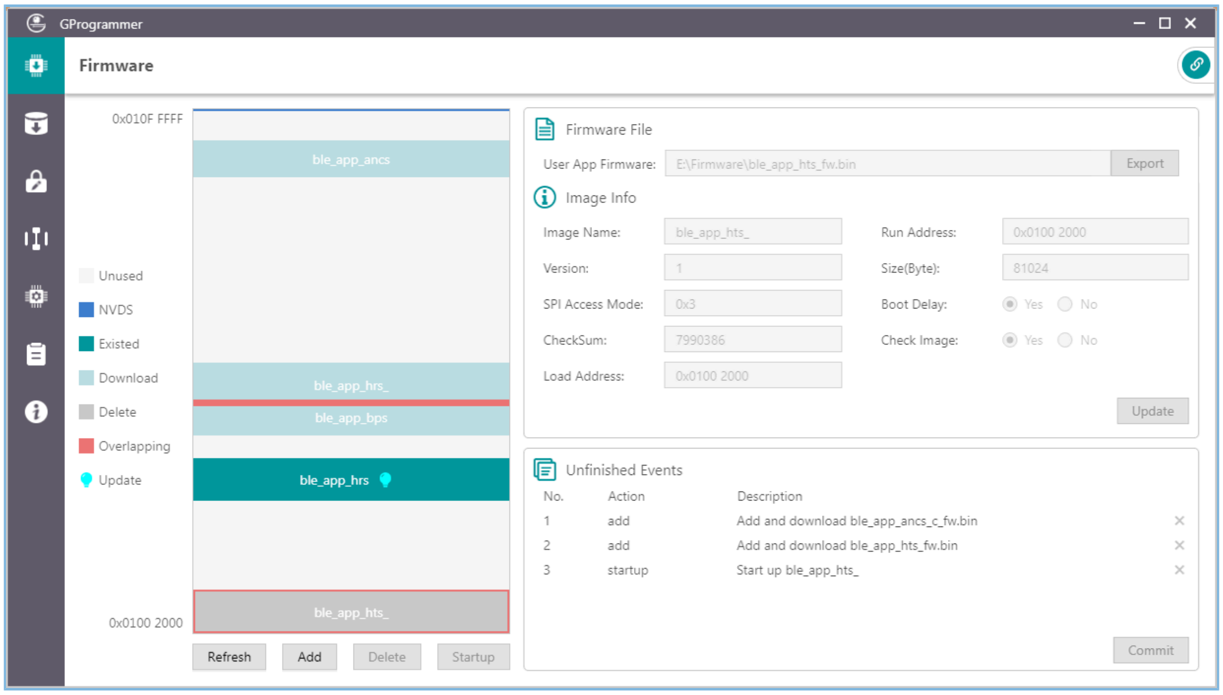
Contents

1 Introduction

GProgrammer supports programming of flash memories on GR551x System-on-Chips (SoCs). It runs on Windows only and provides the following features.

- Connection via SWD and UART
- Firmware download
- Flash programming & erasing
- Inputting product information (ID, name, description, and value)
- Downloading files to eFuse
- Viewing eFuse contents
- Firmware encryption and signing
- Configuring Non-Volatile Data Storage (NVDS) parameters
- Displaying device logs
- Programming on GR551x_console

Figure 1-1 shows the Graphical User Interface (GUI) of GProgrammer.



User App Firmware: E:\Firmware\ble_app_hts_fw.bin

Image Name:	ble_app_hts_	Run Address:	0x0100 2000
Version:	1	Size(Byte):	81024
SPI Access Mode:	0x3	Boot Delay:	[✓ Yes] No
CheckSum:	7990386	Check Image:	[✓ Yes] No
Load Address:	0x0100 2000		

No.	Action	Description
1	add	Add and download ble_app_ancs_c_fw.bin
2	add	Add and download ble_app_hts_fw.bin
3	startup	Start up ble_app_hts_

Figure 1-1 GProgrammer GUI

2 Installation Instructions

This chapter describes the environment requirements as well as installation steps for installing GProgrammer.

2.1 Installation Requirements

- **Hardware environment**

Table 2-1 Hardware environment

Name	Description
CPU	1.6 GHz and faster
RAM	1 GB and larger

- **Operating system**

Table 2-2 Operating system

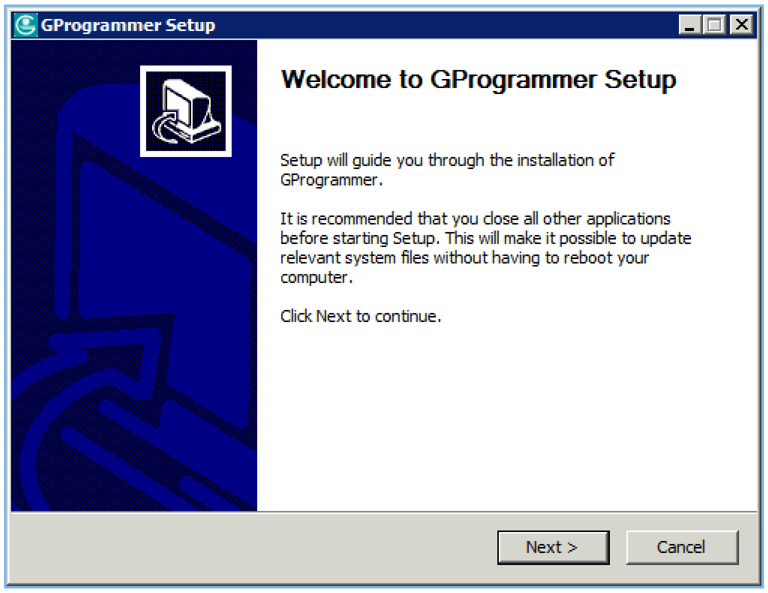
Name	Description
Windows	Windows 7/Windows 10 (32-bit/64-bit)

2.2 Installation Steps

GProgrammer runs on Windows only with an executable installation package: *GProgrammer Setup Version.exe*.

Users can follow the steps below when installing GProgrammer:

1. Double-click *GProgrammer Setup Version.exe*, and follow the steps in the **GProgrammer Setup** wizard (see Figure 2-1).



Welcome to GProgrammer Setup

Setup will guide you through the installation of GProgrammer.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

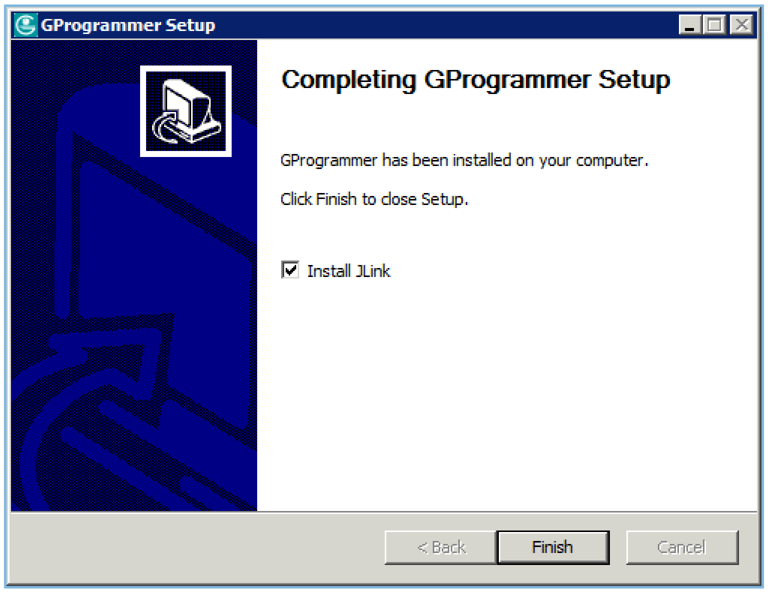
Click Next to continue.

Figure 2-1 **GProgrammer Setup** installation wizard

Note:

Version indicates the GProgrammer software version number.

2. After installing GProgrammer, you are prompted to install J-Link on demand. See Figure 2-2.



GProgrammer Setup

Completing GProgrammer Setup

GProgrammer has been installed on your computer.

Click Finish to close Setup.

☑ Install JLink

Figure 2-2 Prompt to install J-Link

Tip:

For users who have installed J-Link on their PCs before installing GProgrammer, clear **Install J-Link** in the installation wizard.

---

3. After installing J-Link, you can start the GProgrammer by clicking the GProgrammer shortcut on desktop or **Start** menu.

3 Programming Flash with GProgrammer

This chapter elaborates on how to use functional modules of GProgrammer.

3.1 Hardware Connection

Before starting GProgrammer, make sure the host (PC) is correctly connected to the target board. You can establish the connection in either SWD mode or UART mode.

- SWD mode

  In SWD mode, users need a J-Link emulator with one end connecting to the host through a Micro USB cable and the other end connecting to SoC pins of the target board through Dupont wire cables.

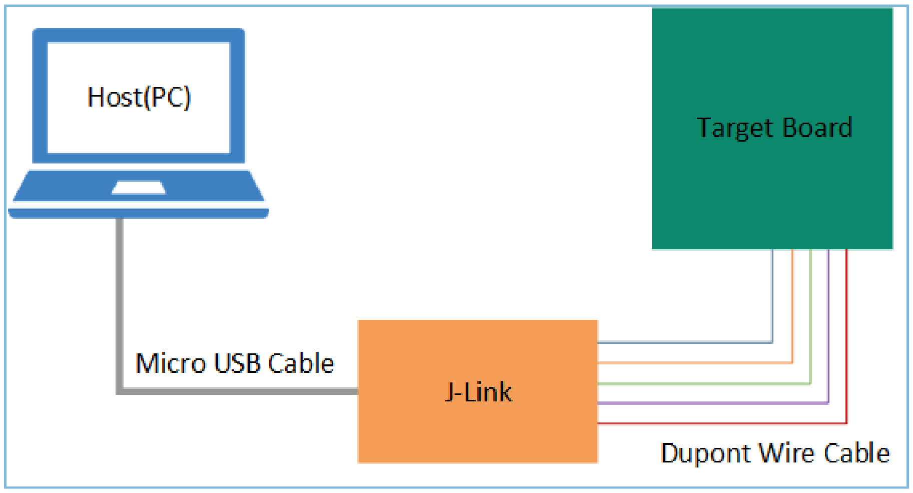


Figure 3-1 Host-target-board connection in SWD mode

The table below lists the mapping relations between J-Link emulator pins and SoC pins.

Table 3-1 Mapping relations between J-Link emulator pins and SoC pins

J-Link Emulator Pin	SoC Pin
VCC	VCC
GND	GND
SWDIO	GPIO_1

J-Link Emulator Pin	SoC Pin
SWCLK	GPIO_0

Note:

- For target boards that have been integrated with J-Link emulator chips, you can connect the host to the target board directly through a Micro USB cable.
- For Goodix Starter Kit (SK) Boards, you cannot connect an SK Board to a PC directly via the on-board Micro USB connector (J49) for firmware programming because the integrated ROM upgrade program in the SoC shall be implemented based on a baud rate of 921600, a value which the integrated J-Link emulator chips on the SK Board fails to support.

- UART mode

  In UART mode, users need a USB-to-serial converter with one end connecting to the host through a Micro USB cable and the other end connecting to SoC pins of the target board through Dupont wire cables.

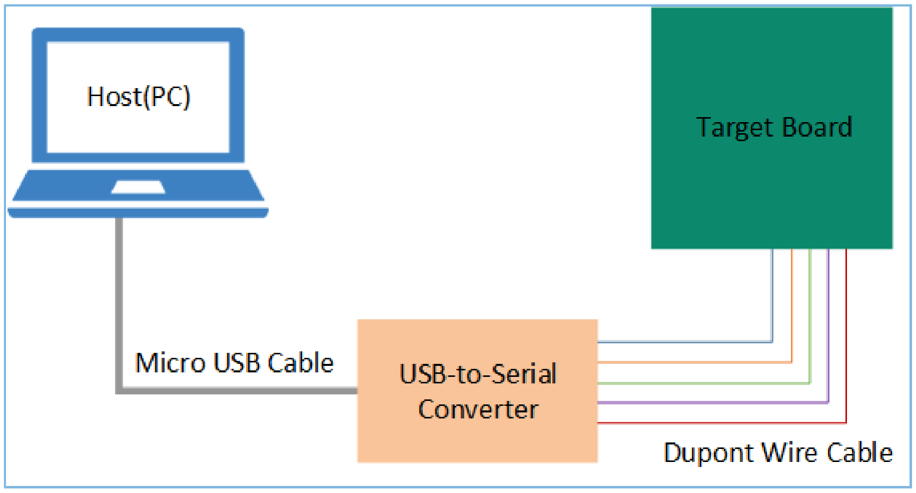


Figure 3-2 Host-target-board connection in UART mode

The table below lists the mapping relations between USB-to-serial converter pins and SoC pins.

Table 3-2 Mapping relations between USB-to-serial converter pins and SoC pins

USB-to-Serial Converter Pin	SoC Pin
VCC	VCC
GND	GND
TX	GPIO_1
RX	GPIO_0
RTS	CHIP_EN

Tip:

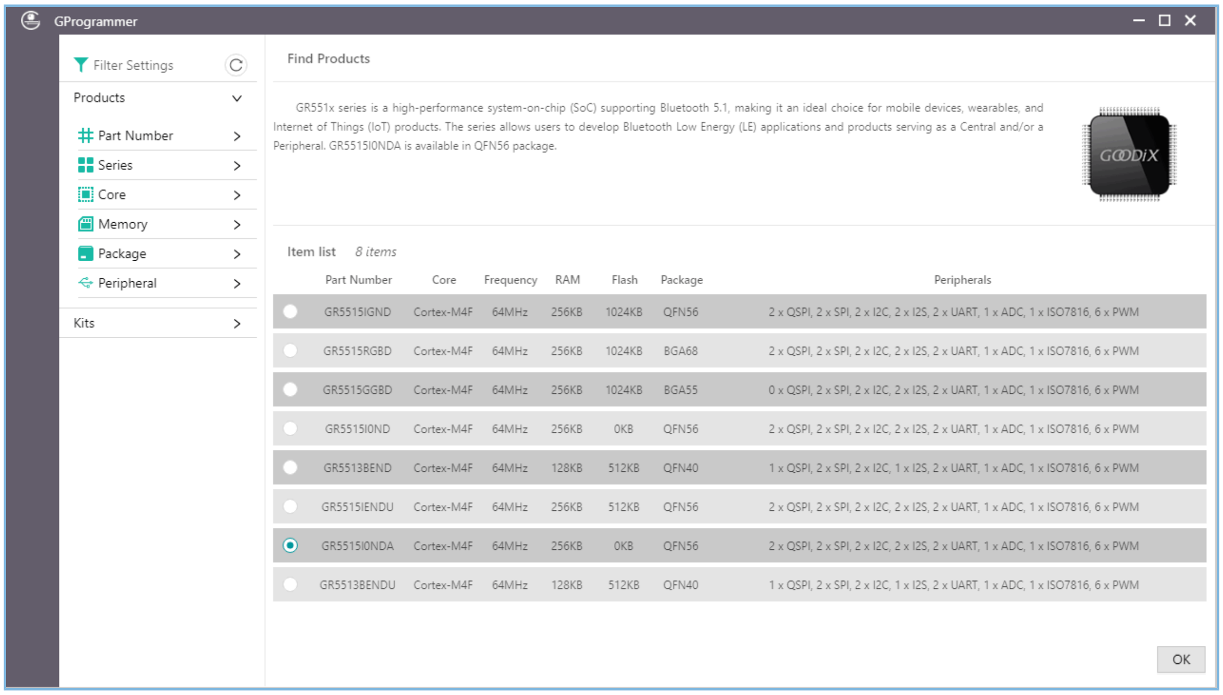
For target boards that have been integrated with USB-to-serial converter chips, you can connect the host to the target board directly through a Micro USB cable.

3.2 SoC Selection

Start GProgrammer. Prior to other operations, you are required to choose the SoC model on your target board and click **OK**.

Tip:

By default, GProgrammer opens the SoC selection interface when being started.



GR551x series is a high-performance system-on-chip (SoC) supporting Bluetooth 5.1, making it an ideal choice for mobile devices, wearables, and Internet of Things (IoT) products. The series allows users to develop Bluetooth Low Energy (LE) applications and products serving as a Central and/or a Peripheral. GR5515I0NDA is available in QFN56 package.

Item list *8 items*

	Part Number	Core	Frequency	RAM	Flash	Package	Peripherals
	GR5515IGND	Cortex-M4F	64MHz	256KB	1024KB	QFN56	2 x QSPI, 2 x SPI, 2 x I2C, 2 x I2S, 2 x UART, 1 x ADC, 1 x ISO7816, 6 x PWM
	GR5515RGBD	Cortex-M4F	64MHz	256KB	1024KB	BGA68	2 x QSPI, 2 x SPI, 2 x I2C, 2 x I2S, 2 x UART, 1 x ADC, 1 x ISO7816, 6 x PWM
	GR5515GGBD	Cortex-M4F	64MHz	256KB	1024KB	BGA55	0 x QSPI, 2 x SPI, 2 x I2C, 2 x I2S, 2 x UART, 1 x ADC, 1 x ISO7816, 6 x PWM
	GR5515I0ND	Cortex-M4F	64MHz	256KB	0KB	QFN56	2 x QSPI, 2 x SPI, 2 x I2C, 2 x I2S, 2 x UART, 1 x ADC, 1 x ISO7816, 6 x PWM
	GR5513BEND	Cortex-M4F	64MHz	128KB	512KB	QFN40	1 x QSPI, 2 x SPI, 2 x I2C, 1 x I2S, 2 x UART, 1 x ADC, 1 x ISO7816, 6 x PWM
	GR5515IENDU	Cortex-M4F	64MHz	256KB	512KB	QFN56	2 x QSPI, 2 x SPI, 2 x I2C, 2 x I2S, 2 x UART, 1 x ADC, 1 x ISO7816, 6 x PWM
●	GR5515I0NDA	Cortex-M4F	64MHz	256KB	0KB	QFN56	2 x QSPI, 2 x SPI, 2 x I2C, 2 x I2S, 2 x UART, 1 x ADC, 1 x ISO7816, 6 x PWM
	GR5513BENDU	Cortex-M4F	64MHz	128KB	512KB	QFN40	1 x QSPI, 2 x SPI, 2 x I2C, 1 x I2S, 2 x UART, 1 x ADC, 1 x ISO7816, 6 x PWM

Figure 3-3 SoC selection interface

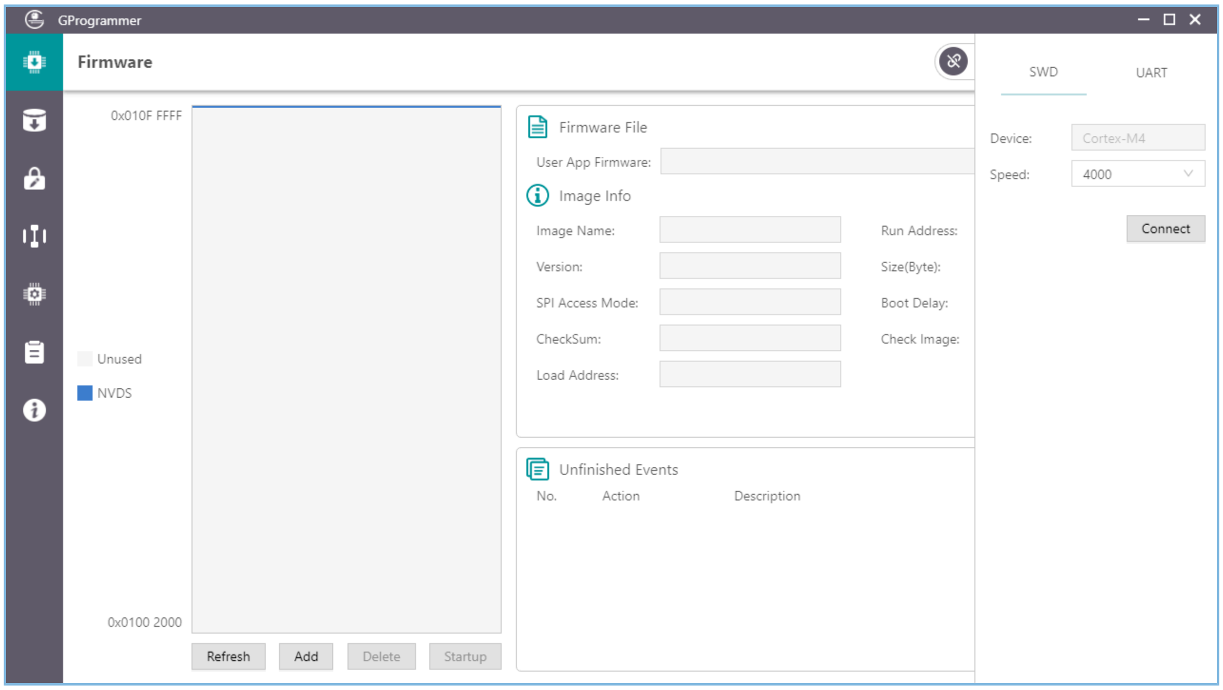
On the SoC selection interface, the left pane lists **Products** and **Kits** options, and the right pane shows the available choices. You can select an SoC by defining its **Part Number**, **Series**, **Core**, **Memory**, **Package**, or **Peripheral**.

Tip:

Peripherals listed on the SoC selection interface are only part of the peripherals of an SoC. For details of all peripherals, see the datasheet corresponding to SoC series.

3.3 GProgrammer GUI

After you choose an SoC, the main operational interface opens, as shown in the figure below.



Device: Cortex-M4

Speed: 4000

Figure 3-4 GProgrammer GUI

The GUI comprises a functional navigation bar on the left (see Table 3-3) and a function operational zone on the right.

Table 3-3 Options on the functional navigation bar

Icon	Function Name	Description
⚙	Firmware	Displays firmware-related operations.
▤	Flash	Displays operations related to flash memory.
🔒	Encrypt & Sign	Displays operations related to firmware encryption and signing.
▥	eFuse Layout	Displays eFuse layout.
⚙	Chip Configuration	Displays operations related to chip configurations.
📋	Device Log	Displays device logs.
ⓘ	Help	Displays help information.

3.4 Connection Management

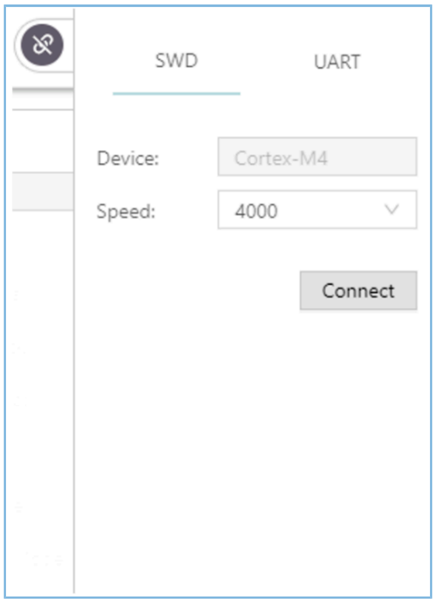
GProgrammer helps users manage and control the connection between your host and target board.

Click ⊗ in the upper-right corner of the interface to open or hide the connection management window of GProgrammer.

GProgrammer supports two connection modes: SWD and UART.

- SWD

Users need to configure **Speed** (data transfer rate) only and click **Connect** to connect the target board to the host.



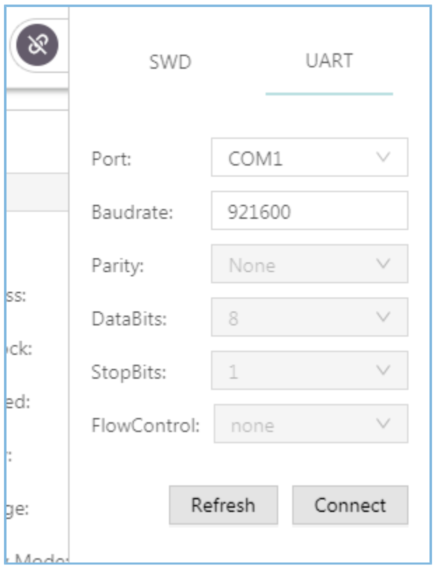
Device: Cortex-M4
Speed: 4000

Figure 3-5 GProgrammer SWD connection

- UART

Users need to configure **Port** (click **Refresh** and select a correct **Port** value) and **Baudrate** on demand. The default configurations of other parameters (**Parity**, **DataBits**, **StopBits**, and **FlowControl**) cannot be modified.

After setting these parameters, click **Connect** to connect the target board to the host.

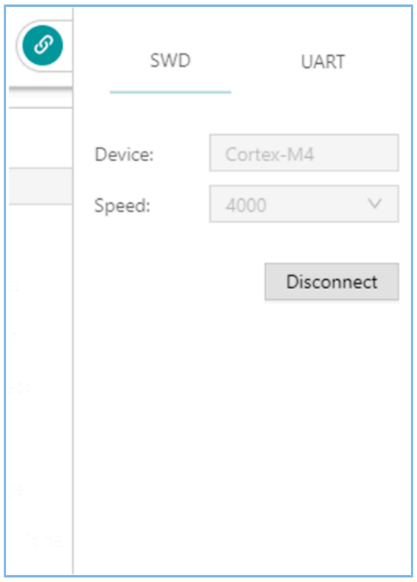


Port: COM1
Baudrate: 921600
Parity: None
DataBits: 8
StopBits: 1
FlowControl: none

Figure 3-6 GProgrammer UART connection

After the connection is successfully established, the connection management window automatically hides with the button turning into , which indicates successful connection establishment.

To disconnect the host from the board, click to open the connection management window, and click **Disconnect**.

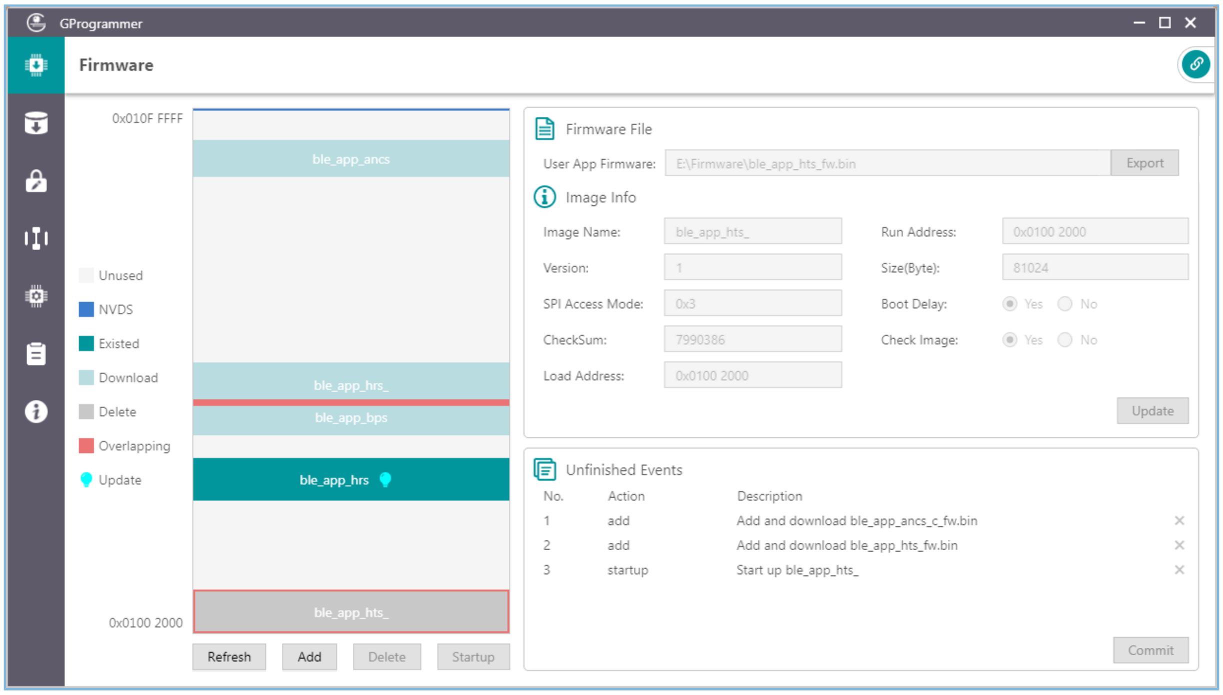


Device: Cortex-M4
Speed: 4000

Figure 3-7 Clicking **Disconnect** on GProgrammer

3.5 Firmware

Click on the left side of the main interface of GProgrammer to open the **Firmware** interface.



Firmware File

User App Firmware: E:\Firmware\ble_app_hts_fw.bin

Image Info

Image Name:	ble_app_hts_	Run Address:	0x0100 2000
Version:	1	Size(Byte):	81024
SPI Access Mode:	0x3	Boot Delay:	[✓ Yes] No
CheckSum:	7990386	Check Image:	[✓ Yes] No
Load Address:	0x0100 2000		

Unfinished Events

No.	Action	Description
1	add	Add and download ble_app_ancs_c_fw.bin
2	add	Add and download ble_app_hts_fw.bin
3	startup	Start up ble_app_hts_

Figure 3-8 GProgrammer **Firmware** interface

You can download your application firmware to the contiguous space of flash memories, ranging from 0x01002000 to 0x010FFFFF.

Note:

The start and end addresses of Flash memories to which firmware can be downloaded vary depending on the Flash size of the specific SoC.

3.5.1 Downloading Firmware

GProgrammer graphically displays the flash memory space layout occupied by firmware (see Figure 3-9), which helps you easily learn the flash occupation status.

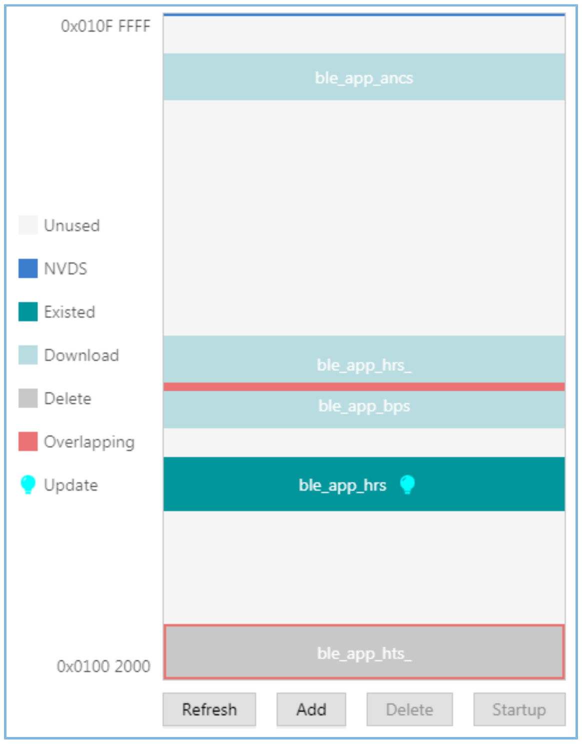


Figure 3-9 Flash firmware layout

- ⬜ represents flash space to which data can be downloaded.
- 🟦 represents default NVDS area to which firmware cannot be downloaded.
- ▨ indicates space for storing to-be-deleted firmware. Example: ble_app_hts_.
- ⬜ indicates space for storing to-be-downloaded firmware. Example: ble_app_ancs.
- 🟩 indicates space for storing downloaded firmware in flash memories. Example: ble_app_hrs.
- 🟥 indicates space overlapped by two pieces of firmware. Examples: ble_app_hrs_ and ble_app_bps.

Follow the steps below to download firmware to a flash memory by using GProgrammer:

1. Click **Add** to add a local firmware file (HEX/BIN) to GProgrammer. GProgrammer presents details of the added firmware such as firmware directory (**User App Firmware**) and **Image Info**.

   In the **Firmware File** area, click **Export** to convert the imported firmware file to an unencrypted BIN file that can be used by the SoC.

2. Click **Commit** to download the firmware to flash memories.

After downloading, the color of the firmware turns from ▢ to ■, indicating the firmware has been successfully downloaded.

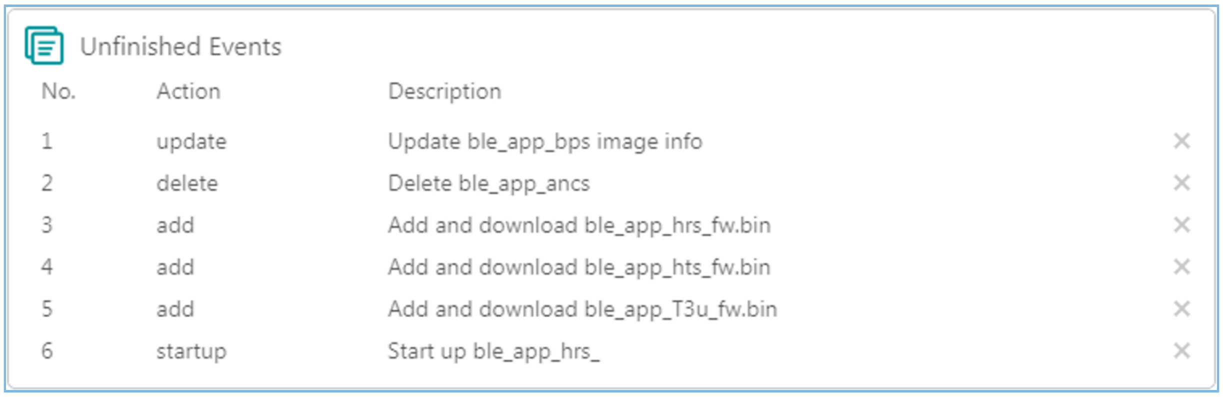
Note:

1. GProgrammer automatically reads firmware existing in the flash memories after being connected a target board.

2. If J-Link cannot be connected when you download firmware, connection/firmware download to the SK Board fails. At this moment, the SoC may be in sleep mode (the firmware keeps running in sleep mode). You can press **RESET** on the SK Board, wait for around one second, and re-download the firmware. If this approach does not work, erase the flash and re-download the firmware.

3.5.2 Action Order

You can execute multiple actions at a time. For example, download multiple pieces of firmware to flash memories and set one piece of firmware as **Startup**. The user-defined actions are executed by clicking **Commit**. The action orders are displayed in **Unfinished Events**, as shown in Figure 3-10.



Unfinished Events

No.	Action	Description	
1	update	Update ble_app_bps image info	×
2	delete	Delete ble_app_ancs	×
3	add	Add and download ble_app_hrs_fw.bin	×
4	add	Add and download ble_app_hts_fw.bin	×
5	add	Add and download ble_app_T3u_fw.bin	×
6	startup	Start up ble_app_hrs_	×

Figure 3-10 Action order

Executable actions for users are listed in the table below.

Table 3-4 Executable actions for users on GProgrammer

Name	Button/Icon	Description
Add firmware	Add	Click **Add** to add a local firmware file to GProgrammer. Alternatively, you can add a local firmware file to GProgrammer by directly dragging the file to GProgrammer from **Windows/File Explorer**. **Note:**

Name	Button/Icon	Description
		Do not click **Open** after dragging the file to GProgrammer.
Refresh firmware	Refresh	Click **Refresh** to obtain the information of firmware downloaded in the flash memories of a target board. Unexecuted actions of flash firmware on the living target board in the **Unfinished Events** pane, such as those labeled as **startup** or **update** are withdrawn with modified parameters being reset to values before refresh.
Delete firmware	Delete	Click the **Delete** button to delete existing firmware in flash memories. Select firmware to be deleted in the flash firmware layout, and click **Delete**. The firmware color turns to ⬜. An action labelled as **delete** is added to the **Unfinished Events**. Note: Delete operations result in deleting only image info of the selected firmware. The firmware information stored in the area will not be deleted.
Start execution	Startup	Set firmware as **startup** to run the firmware immediately. Select firmware in the flash firmware layout, and click the **Startup** button. 🔔 displays on the right of the firmware. An action labelled as **startup** is added to the **Unfinished Events**. The host automatically disconnects from the target board after running the firmware.
Update firmware information	Update	Click the **Update** button to update the information of existing firmware in flash memories on a target board. Select firmware to be updated in the flash firmware layout, and modify the firmware information (the color of modified parameters turns to 🟥). Click **Update**, and the 💡 icon displays on the right side of the firmware. An action labelled as **update** is added to the **Unfinished Events**. Execute update actions, and all parameters involved are locked. No editing is allowed. If modification is required, withdraw the previous update action.

Note:

- In the action order list, you can withdraw an action by clicking ✕ on the right side of the action.
- For two associated actions, withdrawal of the associated action may lead to automatic withdrawal of the previous action. For example, add a firmware file to flash memories, and set it as **startup**. Withdrawal of **Add** leads to withdrawal of **Startup**.

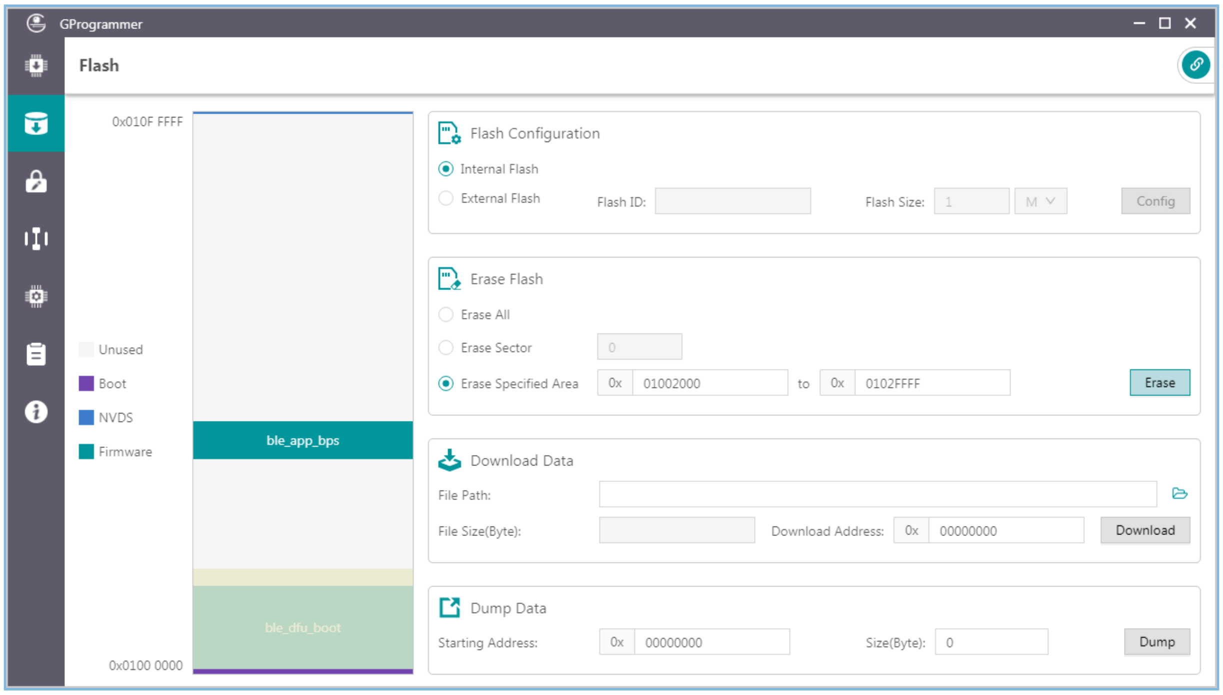
In addition, if there is overlapped space for firmware, **Commit** will not be available until the conflict is resolved.

Note:

For two pieces of firmware totally overlapping with each other, you can click the overlapping space to select one piece of firmware and double-click the space to select the other.

3.6 Flash

Click on the left side of the main interface of GProgrammer to open the **Flash** interface.



Flash Configuration: [✓ Internal Flash] External Flash; Flash ID: ; Flash Size: 1 M

Erase Flash: Erase Sector 0; [✓ Erase Specified Area] 0x 01002000 to 0x 0102FFFF

Download Address: 0x 00000000

Starting Address: 0x 00000000; Size(Byte): 0

Figure 3-11 GProgrammer **Flash** interface

GProgrammer allows users to program internal and external flash memories of SoCs. Detailed programming actions include **Erase Flash**, **Download Data**, and **Dump Data**.

Similar to the firmware layout, the Flash module presents the flash space occupation in a graphic manner.

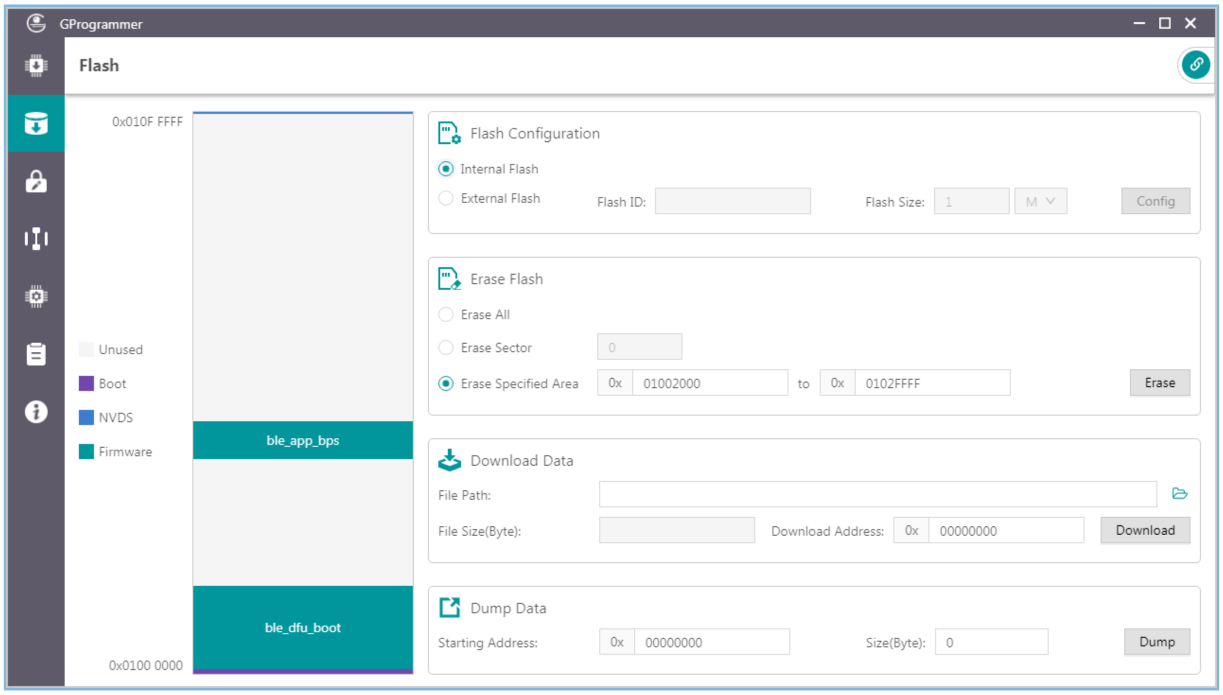
- ⬜ unused flash space
- 🟦 space for NVDS
- 🟪 Boot info space (0x01000000 to 0x01002000, specific to SoC model). The Boot info space is automatically loaded and displayed when users choose internal flash memories.
- 🟩 space for storing downloaded firmware in flash memories. Example: ble_app_bps
- 🟨 space to be operated, such as flash space to be erased

3.6.1 Internal Flash

3.6.1.1 Flash Configuration

Select **Internal Flash** in the **Flash Configuration** list to program internal flash memories.

The flash layout on the left side of the **Flash** interface automatically synchronizes with updated firmware layout information to obtain the firmware, NVDS, and Boot info space.



[● Internal Flash] External Flash

Flash ID:

Flash Size: 1 M

Erase All

Erase Sector 0

[● Erase Specified Area] 0x 01002000 to 0x 0102FFFF

File Path:

File Size(Byte):

Download Address: 0x 00000000

Starting Address: 0x 00000000

Size(Byte): 0

Figure 3-12 Selecting **Internal Flash**

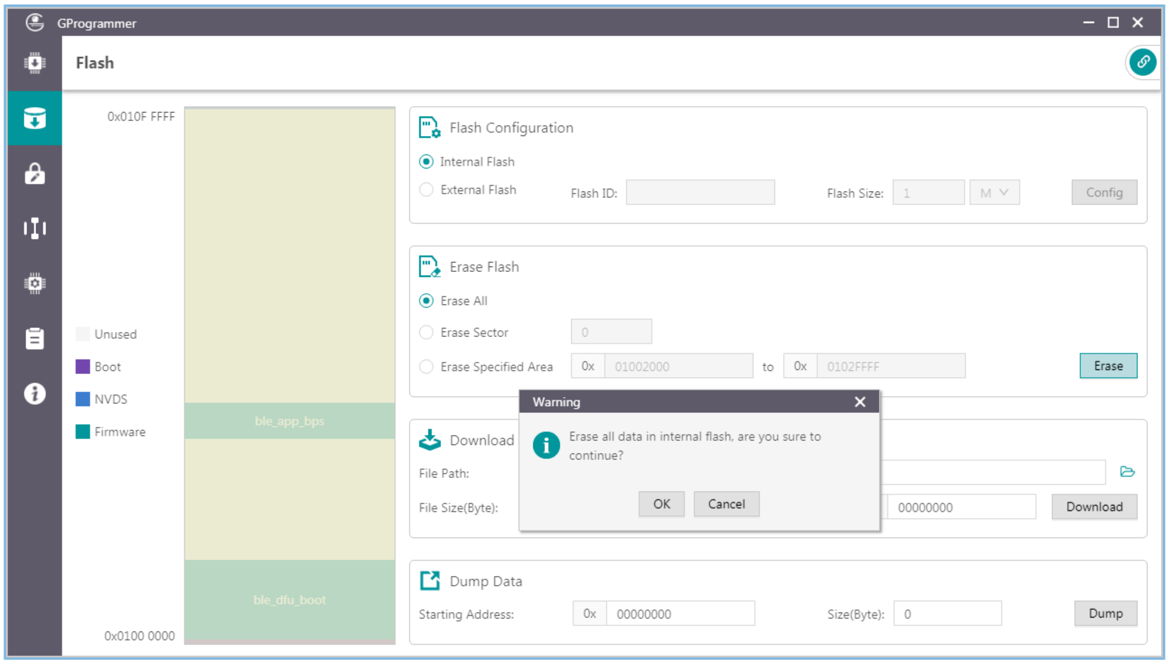
3.6.1.2 Erase Flash

GProgrammer provides three flash erasing mechanisms: **Erase All**, **Erase Sector**, and **Erase Specified Area**.

- **Erase All**

  The mechanism helps erase all flash space.

  The Boot info and NVDS space is cleared with all firmware deleted.



Internal Flash: selected; Flash Size: 1

Erase All: selected; Erase Sector: 0; Erase Specified Area: 0x 01002000 to 0x 0102FFFF

Warning: Erase all data in internal flash, are you sure to continue?

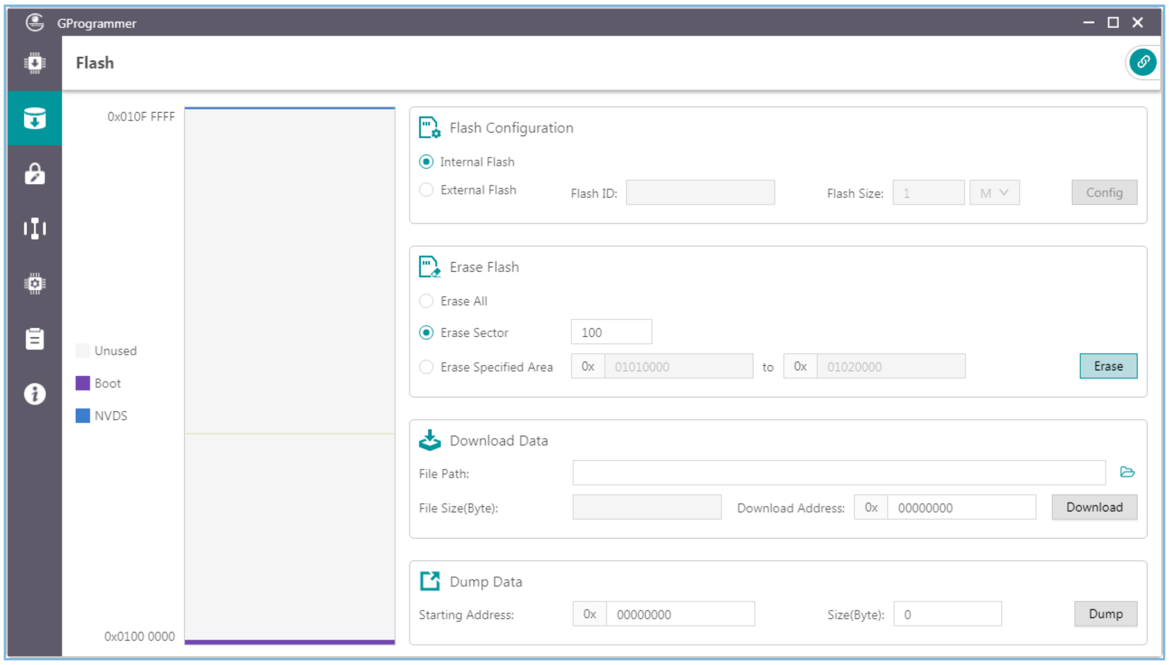
Download address: 00000000

Starting Address: 0x 00000000; Size(Byte): 0

Figure 3-13 **Erase All** on GProgrammer

- **Erase Sector**

  The mechanism helps erase a specified flash sector (size: 4 KB).



Internal Flash: selected; Flash Size: 1

Erase Sector: selected, 100; Erase Specified Area: 0x 01010000 to 0x 01020000

Download Address: 0x 00000000

Starting Address: 0x 00000000; Size(Byte): 0

Figure 3-14 **Erase Sector** on GProgrammer

- **Erase Specified Area**

  The mechanism helps erase an area within a specified address range, by sector.

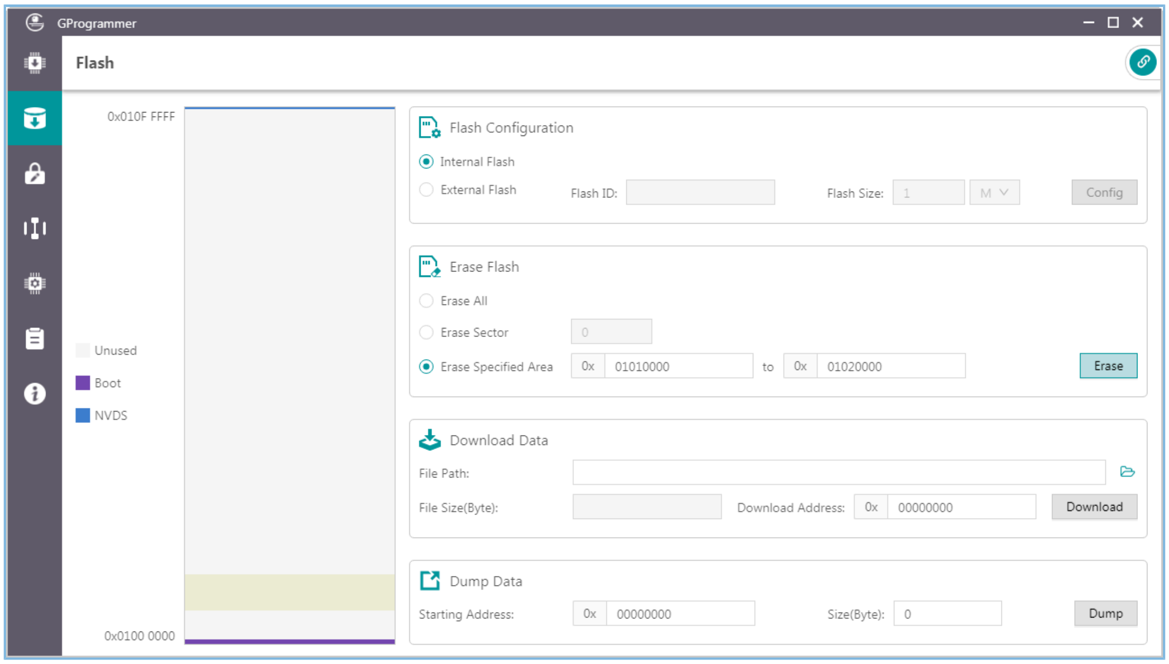


Figure 3-15 **Erase Specified Area** on GProgrammer

3.6.1.3 Download Data

When downloading data to flash memories on GProgrammer, users only need to view and add the BIN files of the data, as well as set a starting address for downloading in **Download Address**.

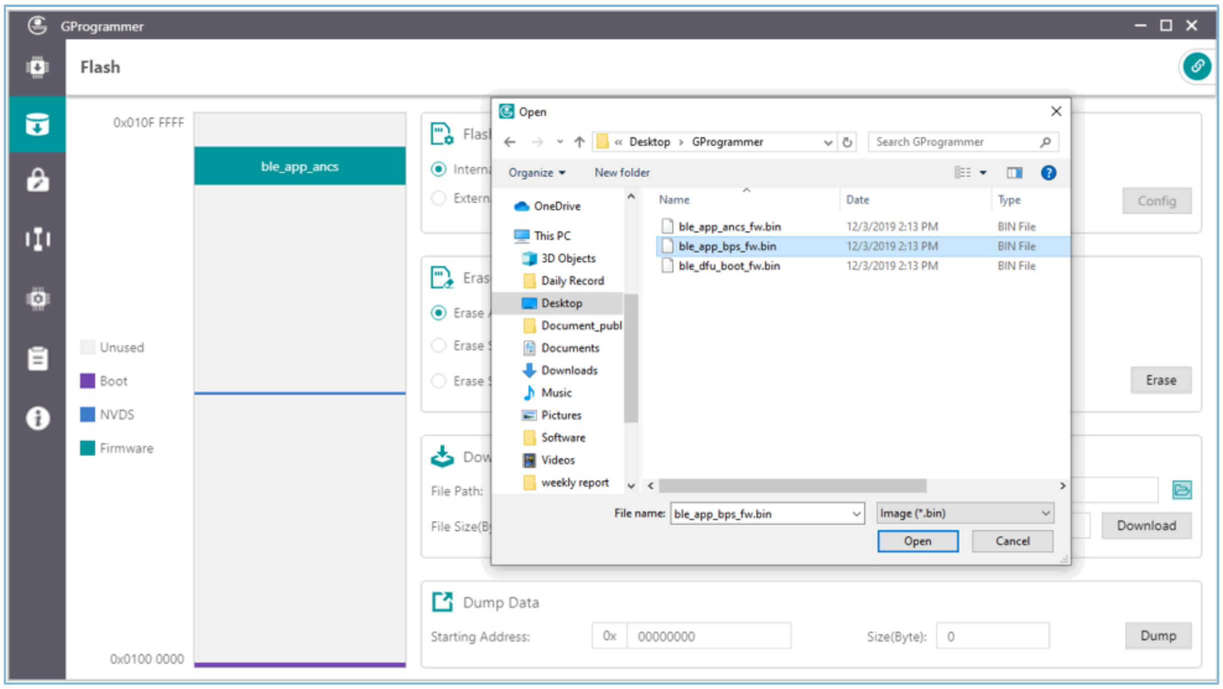
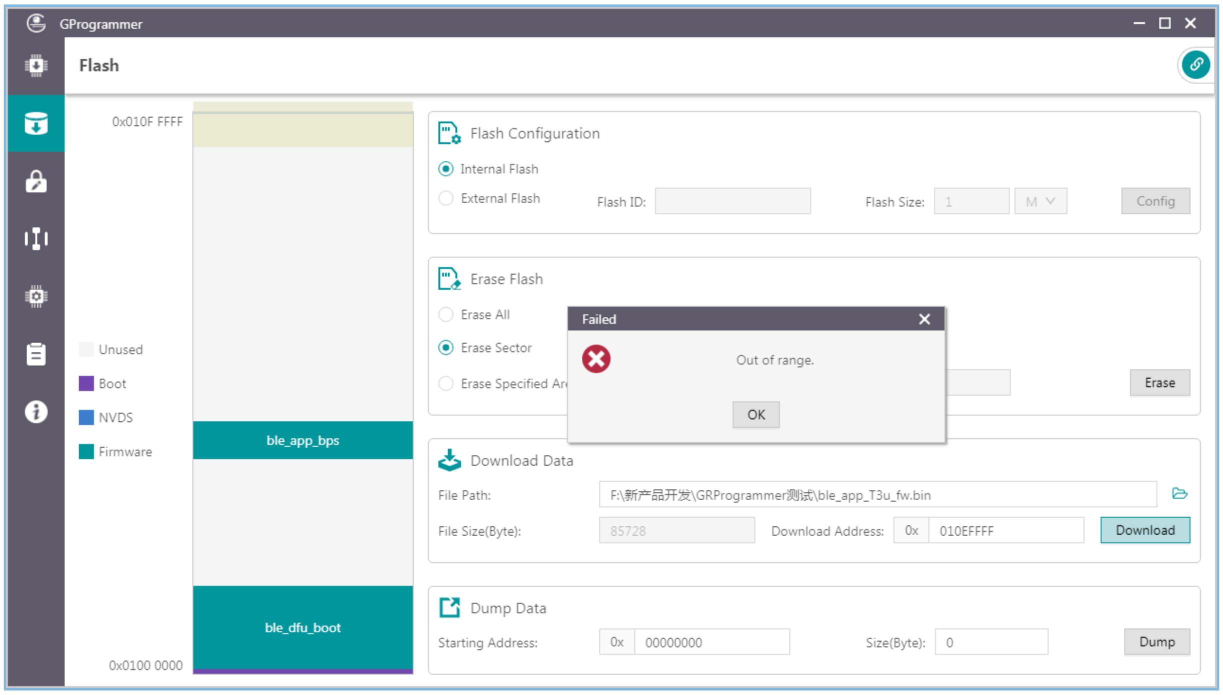


Figure 3-16 Viewing and selecting a data file to be downloaded

A flash overflow error occurs when the downloaded file size is excessively large or the starting address is out of range.



Failed: Out of range.

File Path: F:\新产品开发\GRProgrammer测试\ble_app_T3u_fw.bin

File Size(Byte): 85728

Download Address: 0x 010EFFFF

Starting Address: 0x 00000000

Size(Byte): 0

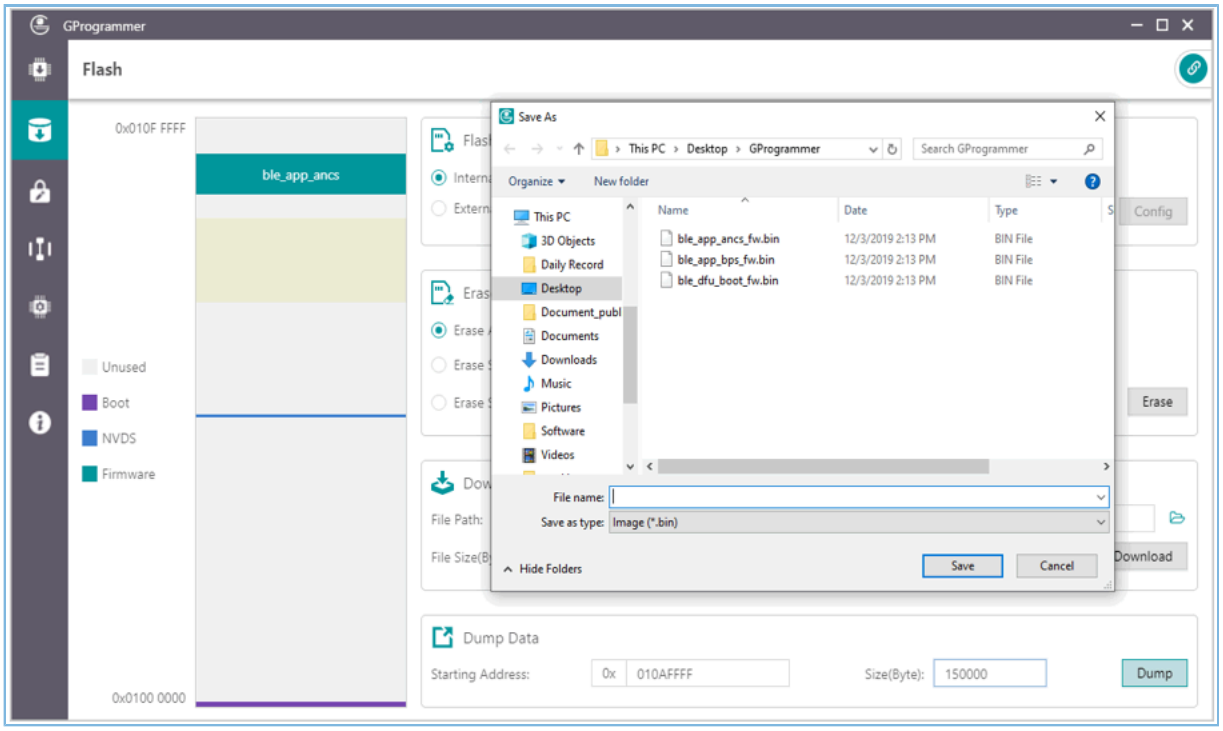
Figure 3-17 Flash overflow error

Tip:

Users are allowed to forcibly download data to the Boot info space in SWD connection mode only. In UART mode, force download to the Boot info space is prohibited.

3.6.1.4 Dump Data

Users can dump any data in flash memories to a local file by specifying a starting dump address and the data size.



Name	Date	Type
ble_app_ancs_fw.bin	12/3/2019 2:13 PM	BIN File
ble_app_bps_fw.bin	12/3/2019 2:13 PM	BIN File
ble_dfu_boot_fw.bin	12/3/2019 2:13 PM	BIN File

File name:
Save as type: Image (*.bin)

Starting Address: 0x 010AFFFF
Size(Byte): 150000

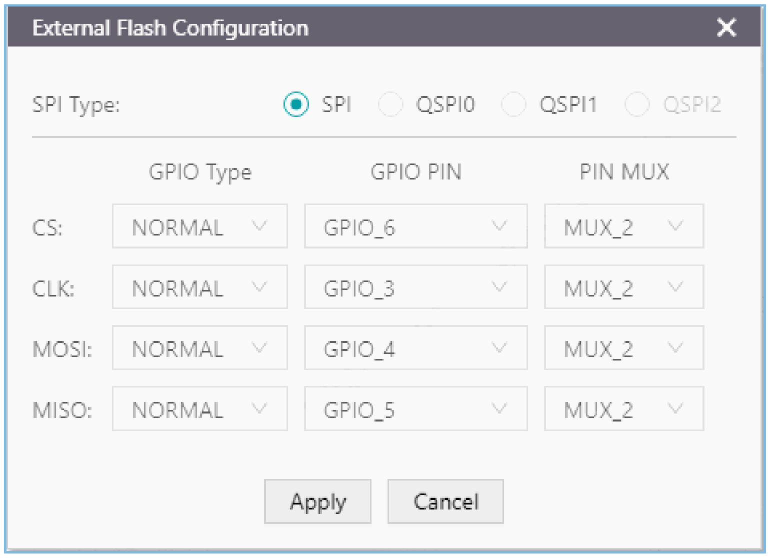
Figure 3-18 **Dump Data** on GProgrammer

3.6.2 External Flash

3.6.2.1 Flash Configuration

Select **External Flash** in the **Flash Configuration** list to program external flash memories. Click **Config** to configure the SPI Type and pins based on actual demands.

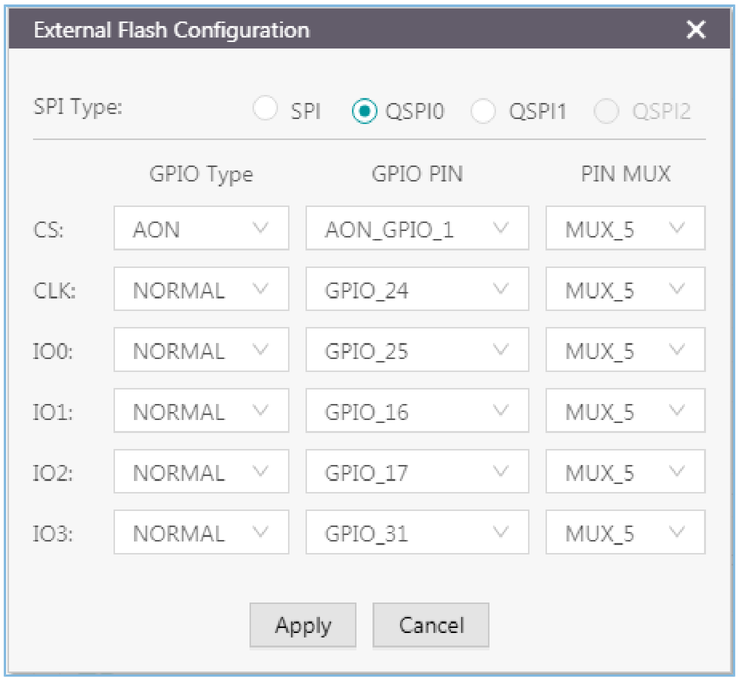
Click **Apply** to complete the configuration.



SPI Type: [✓ SPI] QSPI0 QSPI1 QSPI2

	GPIO Type	GPIO PIN	PIN MUX
CS:	NORMAL	GPIO_6	MUX_2
CLK:	NORMAL	GPIO_3	MUX_2
MOSI:	NORMAL	GPIO_4	MUX_2
MISO:	NORMAL	GPIO_5	MUX_2

Figure 3-19 SPI configurations



SPI Type: ( ) SPI [✓ QSPI0] ( ) QSPI1 ( ) QSPI2

	GPIO Type	GPIO PIN	PIN MUX
CS:	AON	AON_GPIO_1	MUX_5
CLK:	NORMAL	GPIO_24	MUX_5
IO0:	NORMAL	GPIO_25	MUX_5
IO1:	NORMAL	GPIO_16	MUX_5
IO2:	NORMAL	GPIO_17	MUX_5
IO3:	NORMAL	GPIO_31	MUX_5

Figure 3-20 QSPI0 configurations

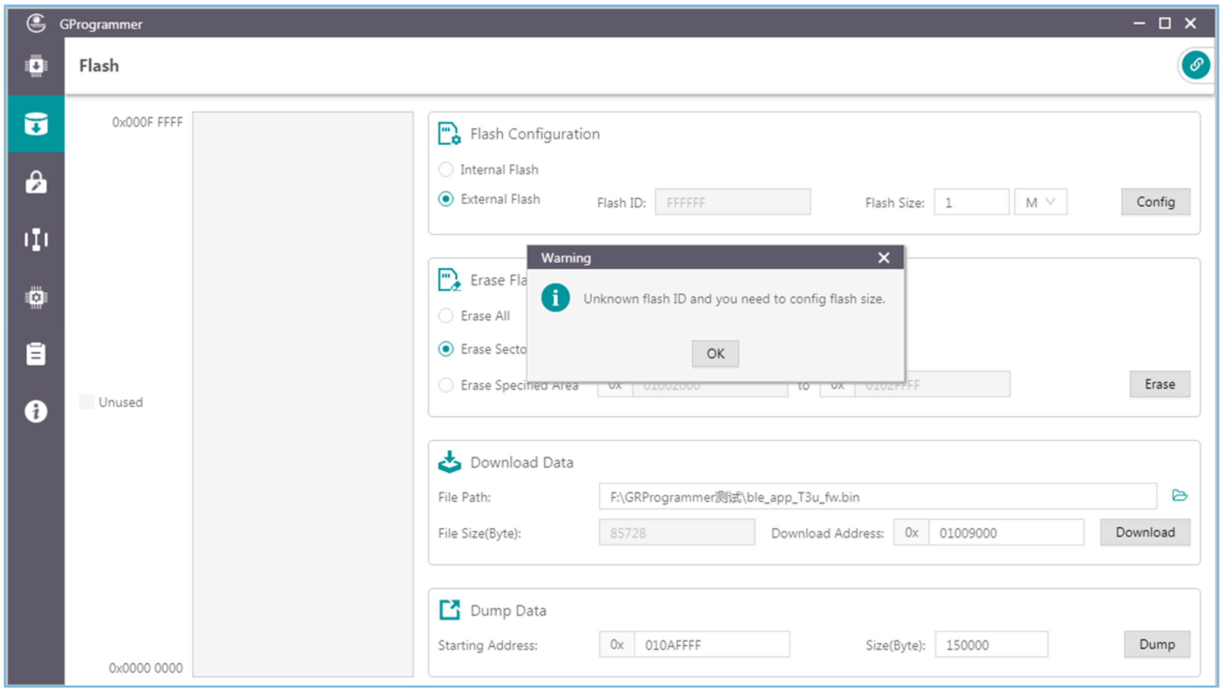
- Configure **Flash Size**

After users apply the pin configurations, GProgrammer reads and displays the external **Flash ID** based on which the **Flash Size** is automatically set.

Tip:

Before clicking **Apply**, make sure external flash memories are correctly connected to the target board in accordance with pin configurations. Incorrect connections lead to failures in communications between external flash and the board.

Users need to manually set the **Flash Size** when GProgrammer fails to get the flash size based on the accessed flash ID.


GProgrammer

Flash

Flash Configuration

Internal Flash [✓ External Flash]

Flash ID: FFFFFF

Flash Size: 1 M

Warning

Unknown flash ID and you need to config flash size.

OK

Erase Sector [✓]

Download Data

File Path: F:\GRProgrammer测试\ble_app_T3u_fw.bin

File Size(Byte): 85728

Download Address: 0x 01009000

Dump Data

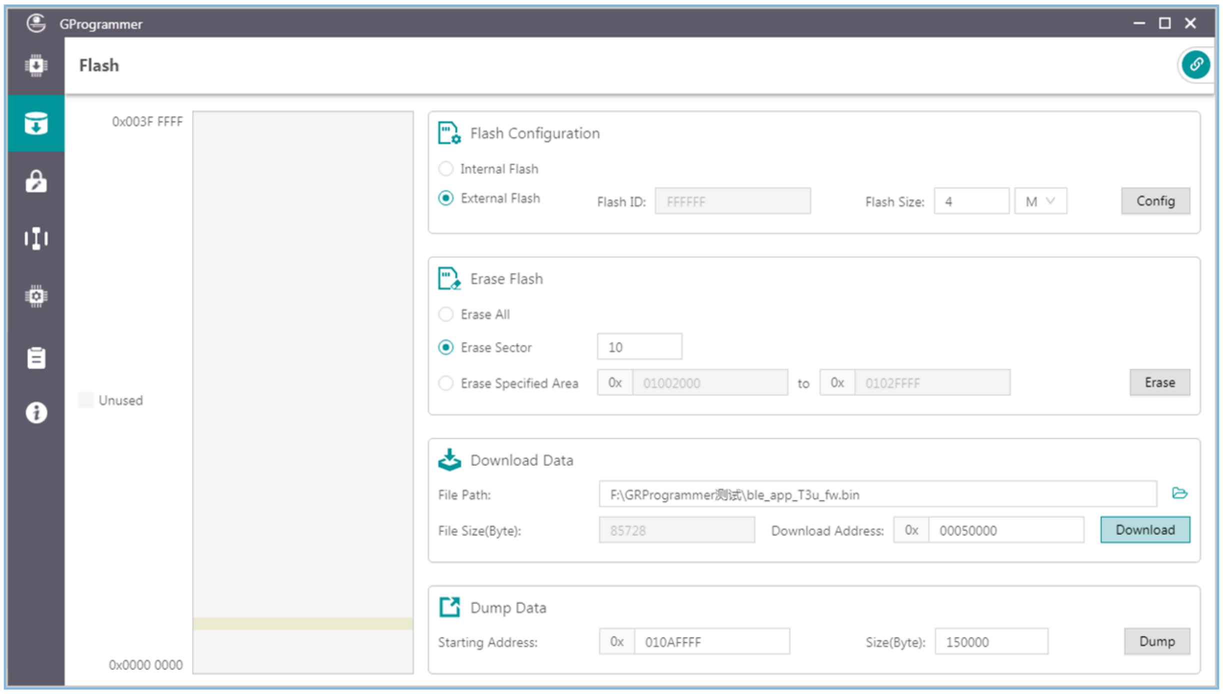
Starting Address: 0x 010AFFFF

Size(Byte): 150000

Figure 3-21 Unknown flash ID

3.6.2.2 External Flash Programming

GProgrammer allows users to program flash memories (erase flash, download data to flash, and dump data to a local file) within a valid address range.


GProgrammer

Flash

Flash Configuration

Internal Flash [✓ External Flash]

Flash ID: FFFFFF

Flash Size: 4 M

Erase Flash

Erase All [✓ Erase Sector] 10

Erase Specified Area 0x 01002000 to 0x 0102FFFF

Download Data

File Path: F:\GRProgrammer测试\ble_app_T3u_fw.bin

File Size(Byte): 85728

Download Address: 0x 00050000

Dump Data

Starting Address: 0x 010AFFFF

Size(Byte): 150000

Figure 3-22 **Download Data** to external flash on GProgrammer

Tip:

No operation on external flash is allowed before completing pin configurations.

3.7 Encrypt & Sign

Click on the left side of the main interface of GProgrammer to open the **Encrypt & Sign** interface.

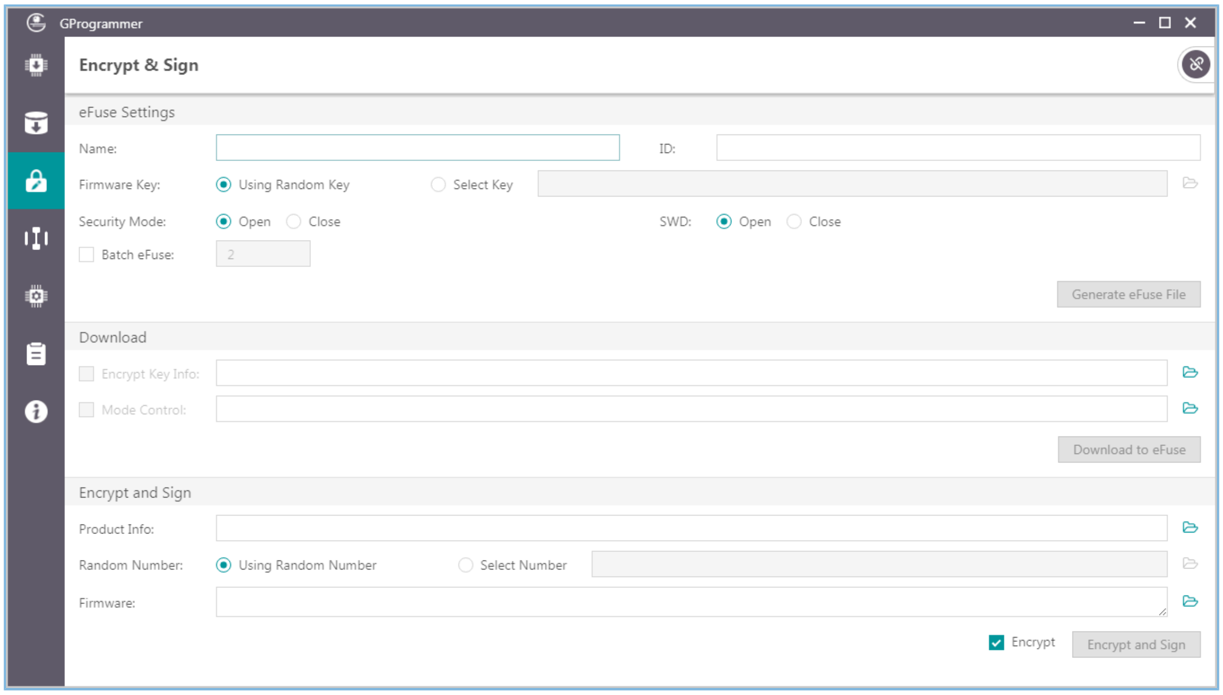


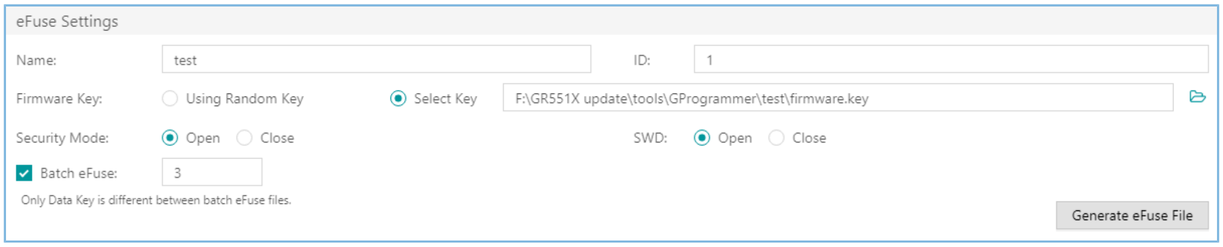
Figure 3-23 GProgrammer **Encrypt & Sign** interface

The selected SoCs support Security Mode and Non-security Mode. The mode is determined by the security mode of the product written in eFuse. When Security Mode is enabled, only firmware that has been encrypted and signed can be downloaded to flash memories.

3.7.1 eFuse Settings

eFuse is a one-time programmable (OTP) memory with random access interfaces on SoCs. The eFuse stores product configurations, security mode control information, and keys for encryption and signing.

When using GProgrammer, users can generate eFuse files by specifying product names, IDs, and firmware keys, and by configuring security mode and SWD interfaces.



eFuse Settings

Name: test

ID: 1

Firmware Key: Using Random Key [✓ Select Key] F:\GR551X update\tools\GProgrammer\test\firmware.key

Security Mode: [✓ Open] Close

SWD: [✓ Open] Close

[✓ Batch eFuse:] 3

Only Data Key is different between batch eFuse files.

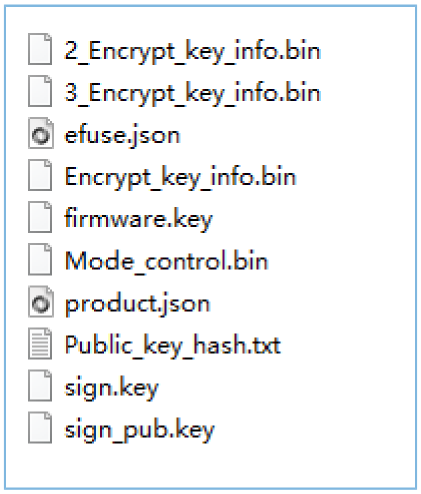
Figure 3-24 Setting eFuse parameters

Note:

- Firmware keys can be random keys generated by GProgrammer. Users can also add key files on demand.
- When **Security Mode** is enabled, users can choose to **Open** or **Close** the SWD interface.

GProgrammer allows users to generate multiple *Encrypt_key_info.bin* files in batches by checking **Batch eFuse**. The generated files are unique, meeting requirements of scenarios demanding one key for one device. For example, when users input "3" in the **Batch eFuse** box, GProgrammer generates three *Encrypt_key_info.bin* files: *Encrypt_key_info.bin*, *2_Encrypt_key_info.bin*, and *3_Encrypt_key_info.bin*.

Generated files are listed in the figure below:

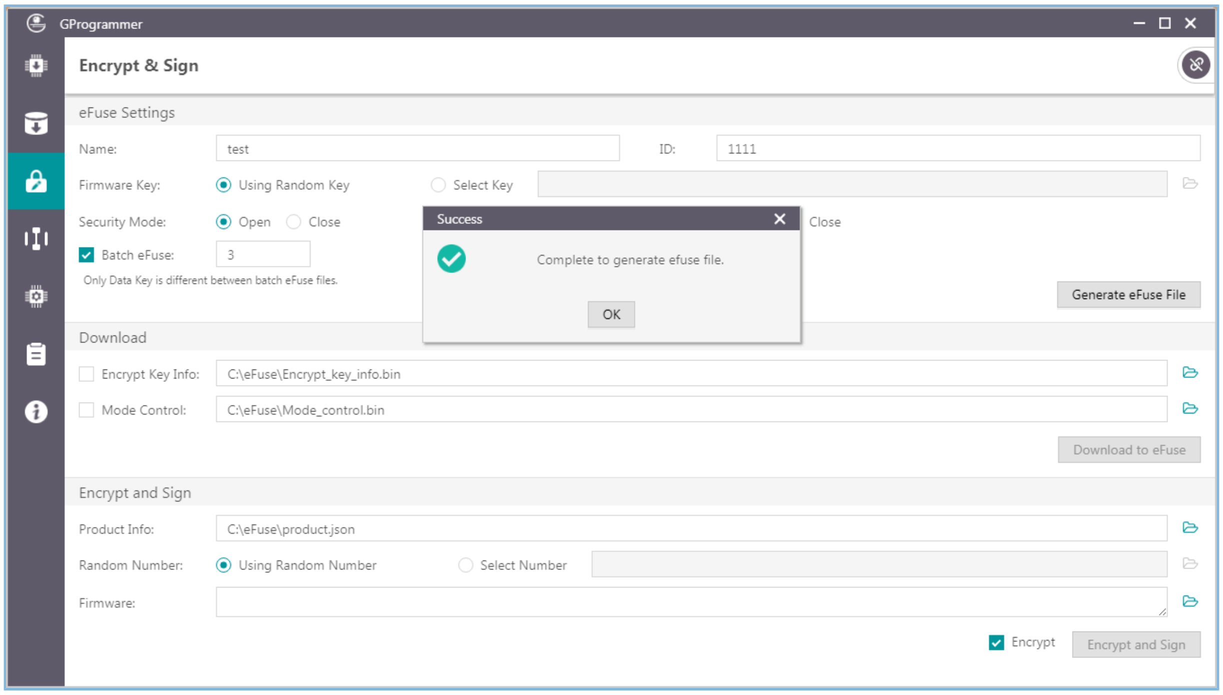


2_Encrypt_key_info.bin
3_Encrypt_key_info.bin
efuse.json
Encrypt_key_info.bin
firmware.key
Mode_control.bin
product.json
Public_key_hash.txt
sign.key
sign_pub.key

Figure 3-25 Generated files

- *efuse.json*: a temporary file
- *Encrypt_key_info.bin*, *2_Encrypt_key_info.bin*, and *3_Encrypt_key_info.bin*: files to be downloaded to eFuse, covering information on products, encryption, and signing. These files shall be downloaded to and stored in eFuse.
- *firmware.key*: a private key for encrypting firmware
- *Mode_control.bin*: an eFuse file covering information on security mode and SWD. This file shall be downloaded to and stored in eFuse.
- *product.json*: a product information file. This file shall be imported to a GProgrammer when encrypting or signing firmware.
- *sign.key*: a private key to generate signatures
- *sign_pub.key*: a public key to verify signatures
- *Public_key_hash.txt*: a public key hash file to verify signatures

To make files download to eFuse or firmware encryption and signing user-friendly, GProgrammer automatically loads the paths of the *Encrypt_key_info.bin* file and the *Mode_control.bin* file to the **Download** area, and the path of the *product.json* file to the **Product Info** pane in the **Encrypt and Sign** area, as shown in the figure below.



Success

Complete to generate efuse file.

Name: test

ID: 1111

Firmware Key: [✓ Using Random Key] Select Key

Security Mode: [✓ Open] Close

Batch eFuse: ✓ 3

Only Data Key is different between batch eFuse files.

Download

Encrypt Key Info: C:\eFuse\Encrypt_key_info.bin

Mode Control: C:\eFuse\Mode_control.bin

Encrypt and Sign

Product Info: C:\eFuse\product.json

Random Number: [✓ Using Random Number] Select Number

Firmware:

✓ Encrypt

Figure 3-26 Paths for automatically loaded files

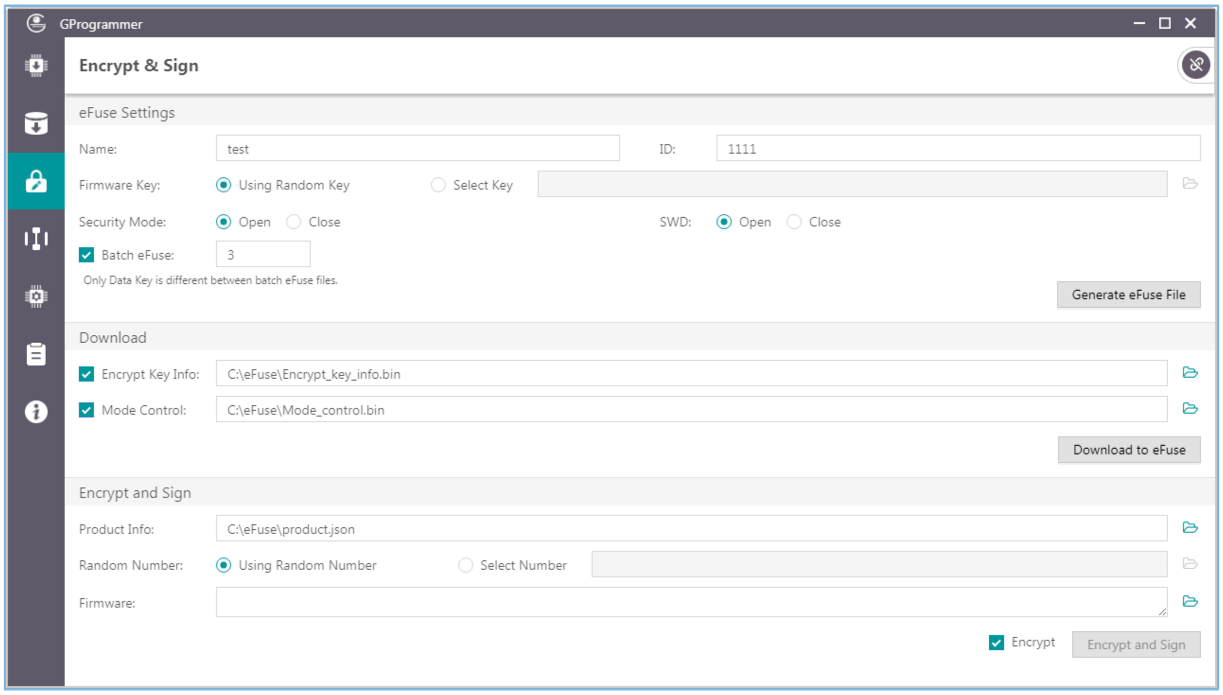
Note:

No modification of eFuse-generated files is allowed because any modification may lead to firmware encryption and signing failures.

3.7.2 Download

For users who have clicked **Generate eFuse File** to generate *Encrypt_key_info.bin* and *Mode_control.bin* files in the **eFuse Settings** pane, select **Encrypt Key Info** and **Mode Control** in the **Download** pane, and click **Download to eFuse** to download the files to eFuse.

Otherwise, users need to manually add *Encrypt_key_info.bin* and *Mode_control.bin* files before downloading the files to eFuse.


GProgrammer

Encrypt & Sign

eFuse Settings

Name: test   ID: 1111

Firmware Key: [✓ Using Random Key] Select Key

Security Mode: [✓ Open] Close   SWD: [✓ Open] Close

✓ Batch eFuse: 3

Only Data Key is different between batch eFuse files.

Generate eFuse File

Download

✓ Encrypt Key Info: C:\eFuse\Encrypt_key_info.bin

✓ Mode Control: C:\eFuse\Mode_control.bin

Download to eFuse

Encrypt and Sign

Product Info: C:\eFuse\product.json

Random Number: [✓ Using Random Number] Select Number

Firmware:

✓ Encrypt   Encrypt and Sign

Figure 3-27 Downloading files to eFuse

Note:

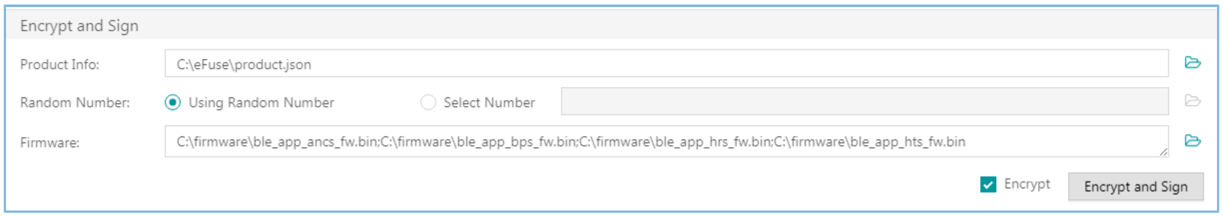
eFuse information cannot be repeatedly downloaded to firmware.

3.7.3 Encrypt & Sign

When Security Mode is enabled, only firmware that has been encrypted and signed can be downloaded to flash memories. GProgrammer allows users to encrypt and sign, or to sign multiple firmware files (HEX/BIN) by using one set of product information (**Product Info**) and one random number (**Random Number**).

The **Random Number** can be manually set by users or generated by GProgrammer.

When adding more than one firmware file, separate each file path with a semicolon (;), as shown in Figure 3-28.


Encrypt and Sign

Product Info: C:\eFuse\product.json

Random Number: [✓ Using Random Number] Select Number

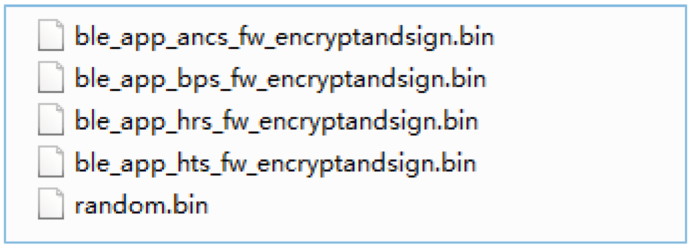
Firmware: C:\firmware\ble_app_ancs_fw.bin;C:\firmware\ble_app_bps_fw.bin;C:\firmware\ble_app_hrs_fw.bin;C:\firmware\ble_app_hts_fw.bin

✓ Encrypt   Encrypt and Sign

Figure 3-28 Adding more than one firmware file

To encrypt and sign the firmware, check the **Encrypt** box, and the button changes from **Sign** to **Encrypt and Sign**; to sign the firmware only, clear the **Encrypt** box, and the button changes back to **Sign**. Choose the directory to save the (encrypted and) signed firmware, and click **Encrypt and Sign**/**Sign**.

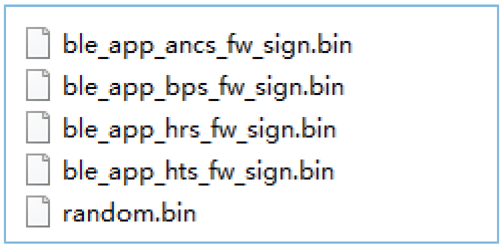
Files after being encrypted and signed are generated in BIN formats with details listed below:



ble_app_ancs_fw_encryptandsign.bin
ble_app_bps_fw_encryptandsign.bin
ble_app_hrs_fw_encryptandsign.bin
ble_app_hts_fw_encryptandsign.bin
random.bin

Figure 3-29 GProgrammer-generated files after encryption and signing

Files after being signed are listed below:



ble_app_ancs_fw_sign.bin
ble_app_bps_fw_sign.bin
ble_app_hrs_fw_sign.bin
ble_app_hts_fw_sign.bin
random.bin

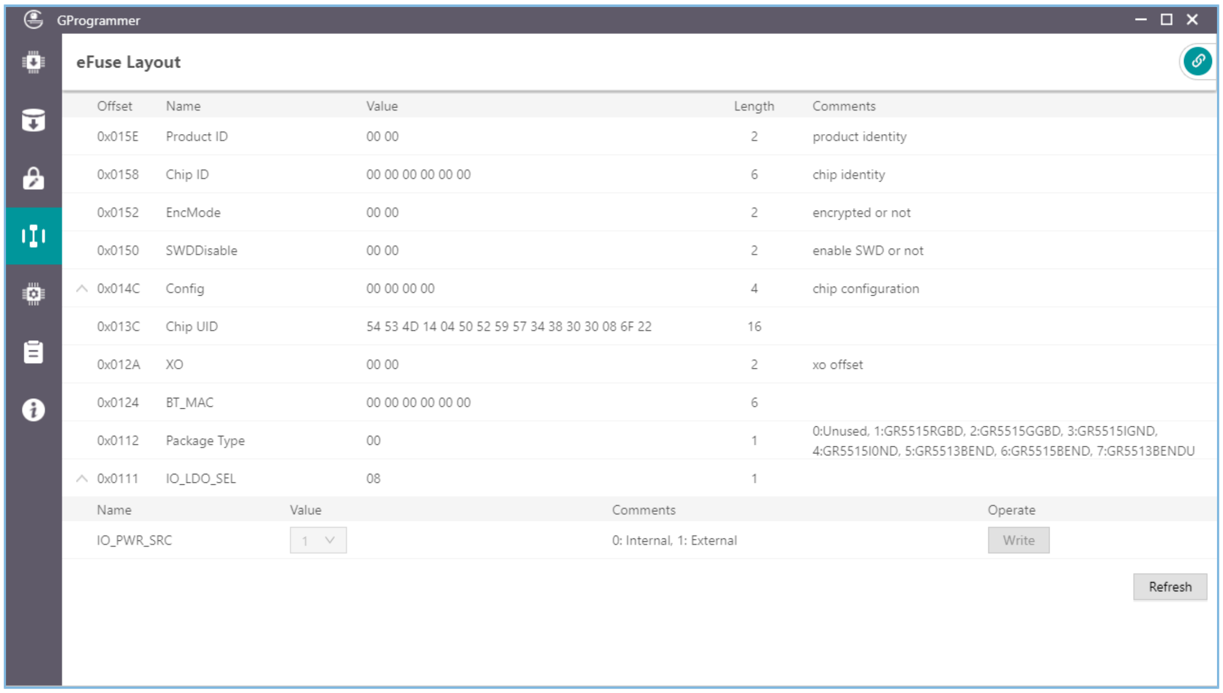
Figure 3-30 GProgrammer-generated files after signing

Note:

The random number generated by GProgrammer is for encryption algorithms. After users perform encryption and signing of firmware files, the *random.bin* file is stored in the same directory as encrypted and signed firmware files. Users can view and add the *random.bin* file to GProgrammer next time they use the random number for firmware encryption and signing.

3.8 eFuse Layout

Click on the left side of the main interface of GProgrammer to open the **eFuse Layout** interface.



Offset	Name	Value	Length	Comments
0x015E	Product ID	00 00	2	product identity
0x0158	Chip ID	00 00 00 00 00 00	6	chip identity
0x0152	EncMode	00 00	2	encrypted or not
0x0150	SWDDisable	00 00	2	enable SWD or not
0x014C	Config	00 00 00 00	4	chip configuration
0x013C	Chip UID	54 53 4D 14 04 50 52 59 57 34 38 30 30 08 6F 22	16	
0x012A	XO	00 00	2	xo offset
0x0124	BT_MAC	00 00 00 00 00 00	6	
0x0112	Package Type	00	1	0:Unused, 1:GR5515RGBD, 2:GR5515GGBD, 3:GR5515IGND, 4:GR5515I0ND, 5:GR5513BEND, 6:GR5515BEND, 7:GR5513BENDU
0x0111	IO_LDO_SEL	08	1	

Name	Value	Comments	Operate
IO_PWR_SRC	1	0: Internal, 1: External	Write

Figure 3-31 **eFuse Layout** interface

GProgrammer presents users with eFuse layout information: **Offset**, **Value**, **Length**, and **Comments** of fields including but not limited to **Product ID**, **Chip ID**, **EncMode**, **SWDDisable**, **Config**, and **IO_LDO_SEL**. Among them, the **Config** and **IO_LDO_SEL** fields contain multiple bit fields.

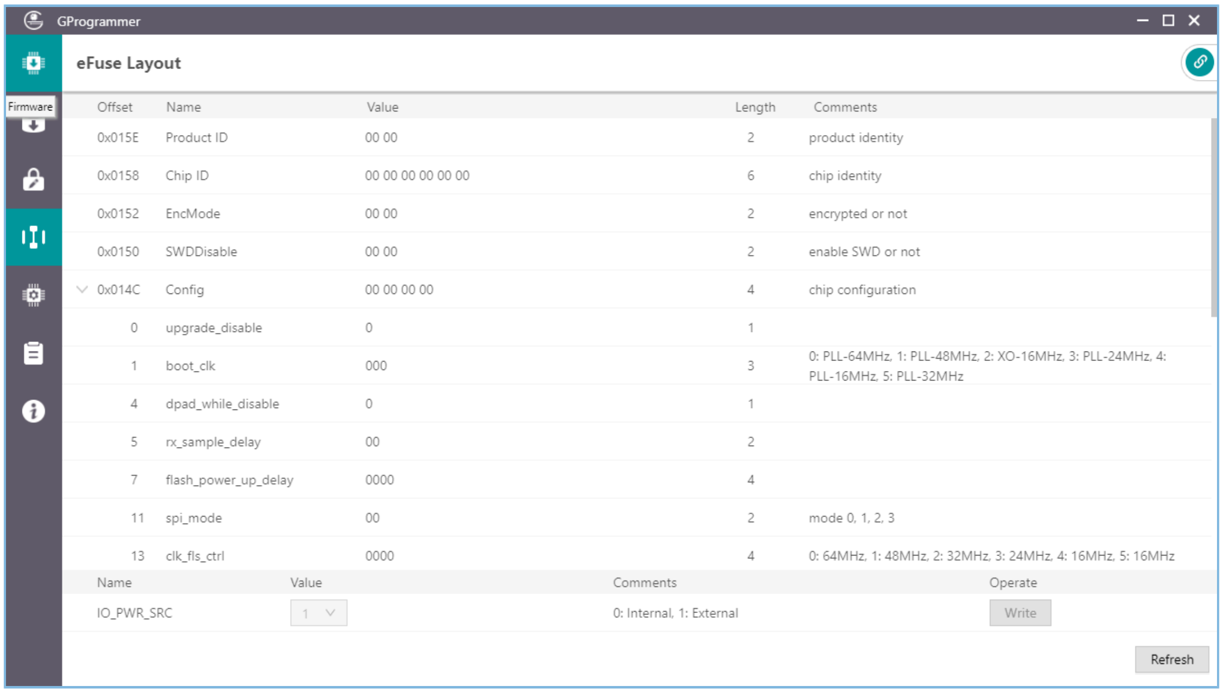
Click **Refresh** to obtain the values of all fields or bit fields.

Click ^ before **Offset** of **Config** or **IO_LDO_SEL** to expand the detailed bits, as shown in the figure below. Click ˅ or double-click **Config** or **IO_LDO_SEL** to collapse the detailed bits.

You can change the **IO_PWR_SRC** value in the **IO_LDO_SEL** field to set the power source of peripherals.

Note:

You can only change the **IO_PWR_SRC** value from "0" to "1". The contrary direction is not allowed.



Offset	Name	Value	Length	Comments
0x015E	Product ID	00 00	2	product identity
0x0158	Chip ID	00 00 00 00 00 00	6	chip identity
0x0152	EncMode	00 00	2	encrypted or not
0x0150	SWDDisable	00 00	2	enable SWD or not
0x014C	Config	00 00 00 00	4	chip configuration
0	upgrade_disable	0	1	
1	boot_clk	000	3	0: PLL-64MHz, 1: PLL-48MHz, 2: XO-16MHz, 3: PLL-24MHz, 4: PLL-16MHz, 5: PLL-32MHz
4	dpad_while_disable	0	1	
5	rx_sample_delay	00	2	
7	flash_power_up_delay	0000	4	
11	spi_mode	00	2	mode 0, 1, 2, 3
13	clk_fls_ctrl	0000	4	0: 64MHz, 1: 48MHz, 2: 32MHz, 3: 24MHz, 4: 16MHz, 5: 16MHz

Name	Value	Comments	Operate
IO_PWR_SRC	1	0: Internal, 1: External	Write

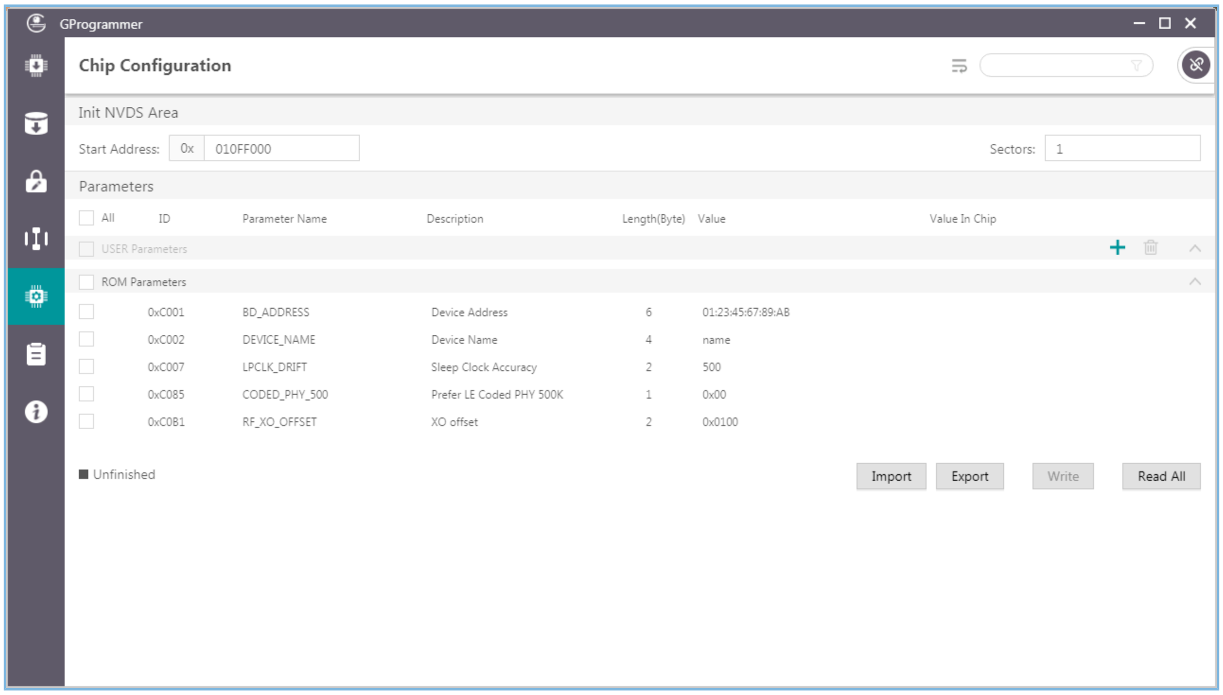
Figure 3-32 Expanded **Offset**

Note:

The fields and bit fields listed in the interface are stored in the *efuse_config.json* file in the config folder. Information stored in eFuse is more than just the listed fields and bit fields.

3.9 Chip Configuration

Click on the left side of the main interface of GProgrammer to open the **Chip Configuration** interface.



Chip Configuration

Init NVDS Area

Start Address: 0x 010FF000

Sectors: 1

Parameters

All	ID	Parameter Name	Description	Length(Byte)	Value	Value In Chip
USER Parameters						
ROM Parameters						
	0xC001	BD_ADDRESS	Device Address	6	01:23:45:67:89:AB	
	0xC002	DEVICE_NAME	Device Name	4	name	
	0xC007	LPCLK_DRIFT	Sleep Clock Accuracy	2	500	
	0xC085	CODED_PHY_500	Prefer LE Coded PHY 500K	1	0x00	
	0xC0B1	RF_XO_OFFSET	XO offset	2	0x0100	

■ Unfinished

Figure 3-33 GProgrammer **Chip Configuration** interface

GProgrammer allows users to set the parameters (including **USER Parameters** and **ROM Parameters**) stored in the NVDS area.

- **USER Parameters**: user-defined parameters that can be added, deleted, and modified
- **ROM Parameters**: ROM parameters stored on GR551x SoCs, which can be modified only by users. Neither parameter addition nor deletion is allowed.

Note:

- The default ROM parameters listed in the interface are stored in the *nvds_config.json* file in the config folder. The parameters are not results accessed in real time from the NVDS area. For more information about ROM parameters, see Table 3-5.
- Click in the upper-right corner of the **Chip Configuration** interface to enable display of complete value contents of a parameter.
- Look up parameters quickly by using the screening box in the upper-right corner of the interface.

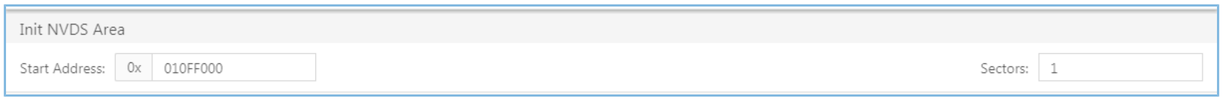
Table 3-5 NVDS ROM parameters

ID	Parameter Name	Description
0xC001	BD_ADDRESS	This parameter sets the Bluetooth device address.
0xC002	DEVICE_NAME	This parameter sets the device name.
0xC007	LPCLK_DRIFT	This parameter sets the Sleep Clock Accuracy (SCA); range: 10 ppm to 500 ppm
0xC085	CODED_PHY_500	This parameter sets the default Coded PHY value; Value 0: 125 kbps; Value 1: 500 kbps

ID	Parameter Name	Description
0xC0B1	RF_XO_OFFSET	This parameter sets the clock calibration byte; range: 0x000 to 0x1FF

3.9.1 Init NVDS Area

Prior to configuring NVDS parameters, users need to specify a starting address (4 KB aligned) and the number of occupied sectors in the NVDS area.

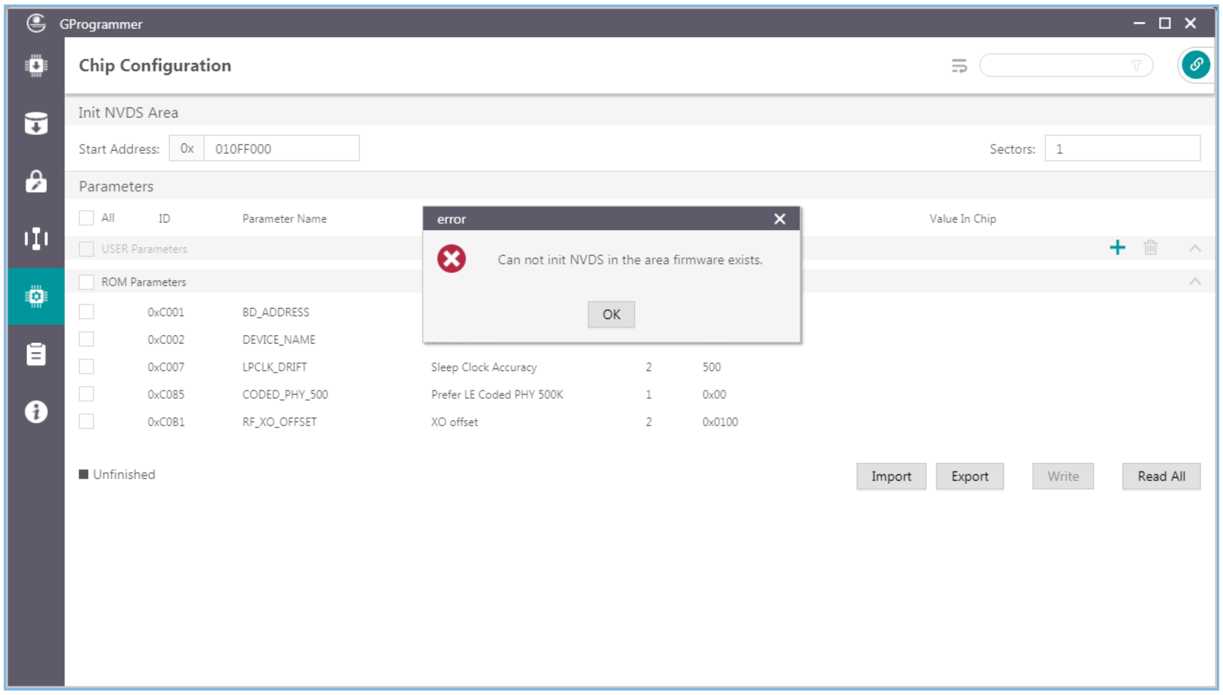

Init NVDS Area

Start Address: 0x 010FF000

Sectors: 1

Figure 3-34 Setting the starting address and sector quantity in the NVDS area

NVDS initialization fails when the configured NVDS area overlaps with the existing firmware area.



Init NVDS Area

Start Address: 0x 010FF000

Sectors: 1

Parameters

All	ID	Parameter Name				Value In Chip
USER Parameters						
ROM Parameters						
	0xC001	BD_ADDRESS				
	0xC002	DEVICE_NAME				
	0xC007	LPCLK_DRIFT	Sleep Clock Accuracy	2	500	
	0xC085	CODED_PHY_500	Prefer LE Coded PHY 500K	1	0x00	
	0xC0B1	RF_XO_OFFSET	XO offset	2	0x0100	

error

Can not init NVDS in the area firmware exists.

Figure 3-35 NVDS initialization failure

3.9.2 Read All

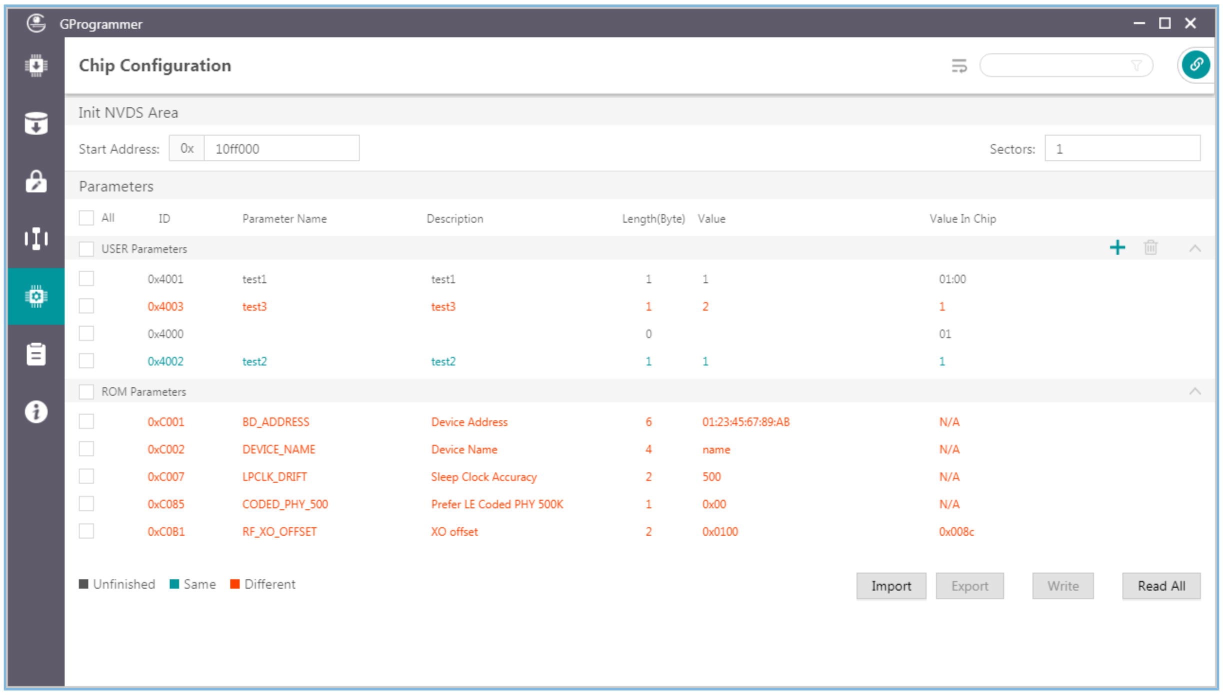
GProgrammer can read all parameters in the current NVDS area and display them in the **Parameters** pane.

To prevent operation failures in user applications due to parameter overlapping in the NVDS area, users are recommended to click **Read All** after connecting the target board to the host.

GProgrammer provides three parameter states: **Unfinished**, **Same**, and **Different**, which help you quickly identify the parameter state in the current NVDS. Details are listed below:

- **Unfinished**: Parameters in unfinished state are presented in black. These parameters are either new ones different from the default listed parameters after users click **Read All** (example: 0x4000 in Figure 3-36) or ones that have been listed in the NVDS area but with a different parameter length (example: 0x4001 in Figure 3-36).

- **Same**: Parameters in same state are presented in green, indicating the parameters already exist in the NVDS area and have the same length and value as those in the default list (example: 0x4002 in Figure 3-36)
- **Different**: Parameters in different state are presented in orange, indicating the parameters already exist in the NVDS area and have the same length as but a different value from default listed parameters (example: 0x4003 in Figure 3-36)



Init NVDS Area

Start Address: 0x 10ff000 Sectors: 1

Parameters

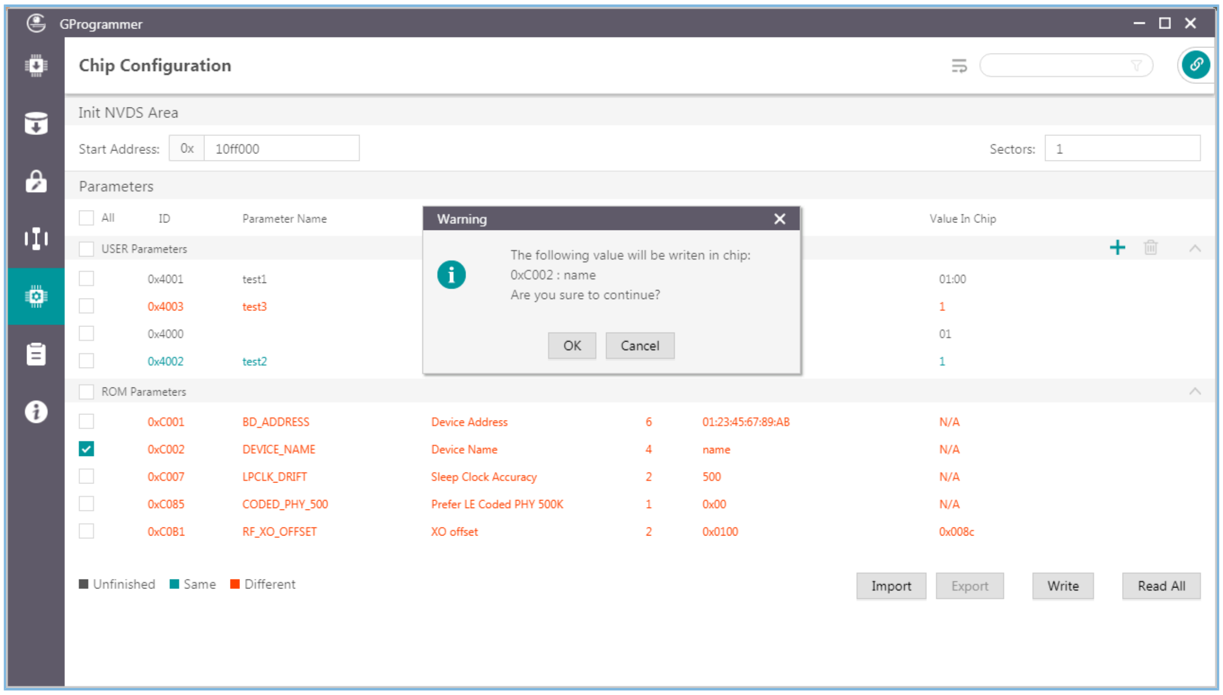
All	ID	Parameter Name	Description	Length(Byte)	Value	Value In Chip
USER Parameters						
	0x4001	test1	test1	1	1	01:00
	0x4003	test3	test3	1	2	1
	0x4000			0		01
	0x4002	test2	test2	1	1	1
ROM Parameters						
	0xC001	BD_ADDRESS	Device Address	6	01:23:45:67:89:AB	N/A
	0xC002	DEVICE_NAME	Device Name	4	name	N/A
	0xC007	LPCLK_DRIFT	Sleep Clock Accuracy	2	500	N/A
	0xC085	CODED_PHY_500	Prefer LE Coded PHY 500K	1	0x00	N/A
	0xC0B1	RF_XO_OFFSET	XO offset	2	0x0100	0x008c

■ Unfinished 🟦 Same 🟧 Different

Figure 3-36 **Read All** interface

3.9.3 Write

Select parameters to be written to NVDS, and click **Write**.


GProgrammer

Chip Configuration

Init NVDS Area

Start Address: 0x 10ff000 Sectors: 1

Parameters

Warning

The following value will be writen in chip:
0xC002 : name
Are you sure to continue?

OK Cancel

All	ID	Parameter Name				Value In Chip
USER Parameters						
☐	0x4001	test1				01:00
☐	0x4003	test3				1
☐	0x4000					01
☐	0x4002	test2				1
ROM Parameters						
☐	0xC001	BD_ADDRESS	Device Address	6	01:23:45:67:89:AB	N/A
✓	0xC002	DEVICE_NAME	Device Name	4	name	N/A
☐	0xC007	LPCLK_DRIFT	Sleep Clock Accuracy	2	500	N/A
☐	0xC085	CODED_PHY_500	Prefer LE Coded PHY 500K	1	0x00	N/A
☐	0xC0B1	RF_XO_OFFSET	XO offset	2	0x0100	0x008c

■ Unfinished 🟩 Same 🟥 Different

Import Export Write Read All

Figure 3-37 **Write** parameters to NVDS

Tip:

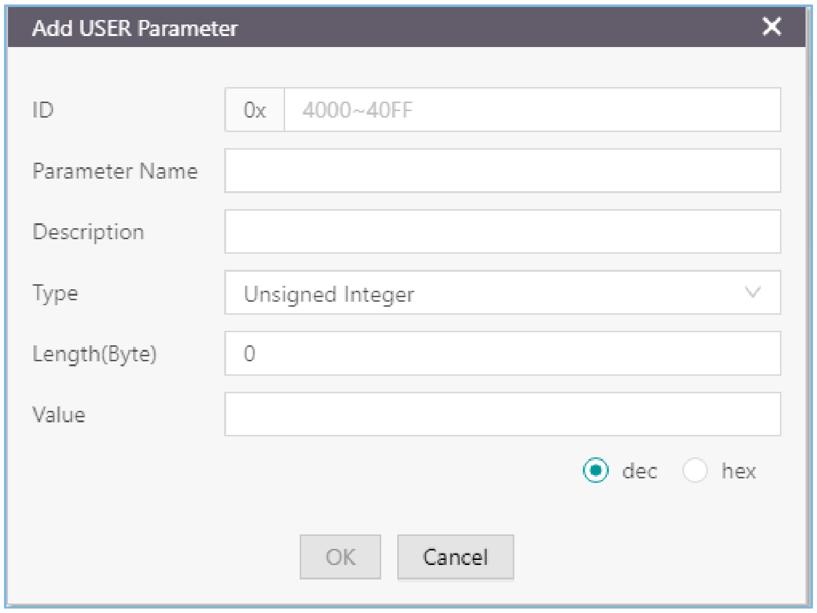
- Parameters in unfinished state cannot be written to NVDS directly.
- You can select more than one parameter to implement a batch write.
- When an unfinished parameter is selected, **Write** is unavailable.

3.9.4 Add a User Parameter

Follow the steps below to add a user parameter to NVDS.

1. Click + to open the **Add USER Parameter** window.

2. Specify the **ID**, **Parameter Name**, **Description**, **Type**, **Length(Byte)**, **Value**, and data presentation format (**dec** or **hex**).


Add USER Parameter

ID	0x 4000~40FF
Parameter Name	
Description	
Type	Unsigned Integer
Length(Byte)	0
Value	

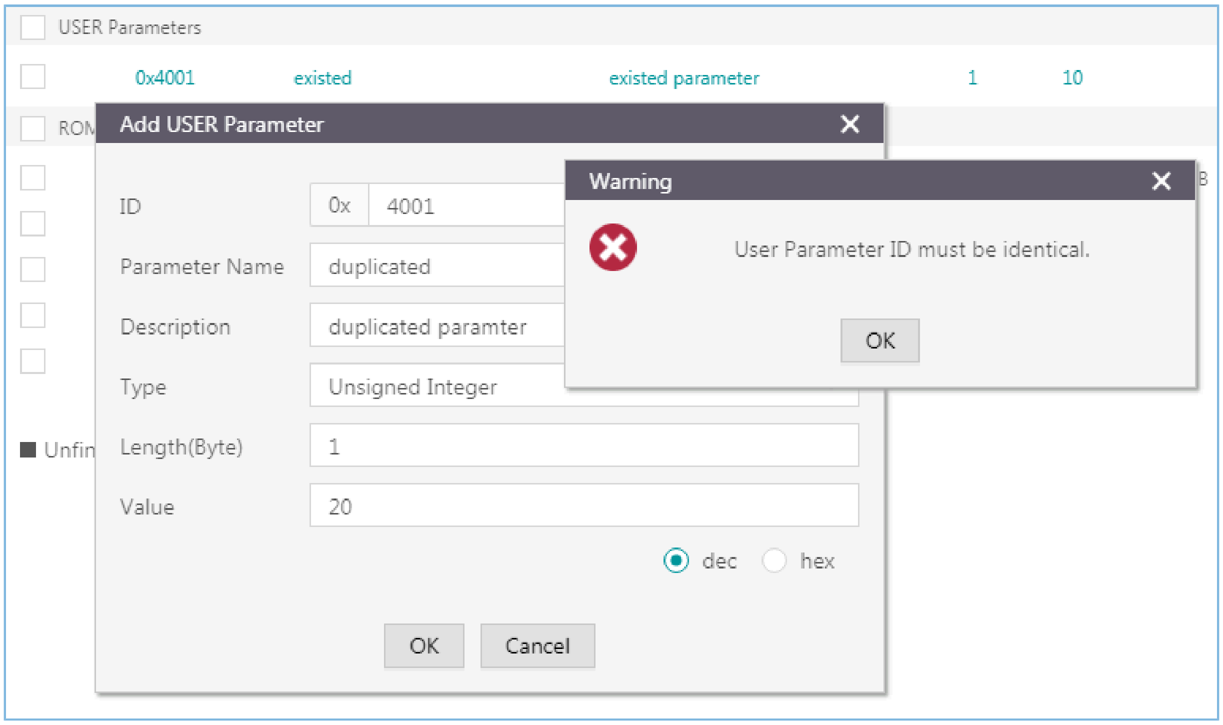
[✓ dec] hex

Figure 3-38 Adding a user parameter to NVDS

3. Click **OK** to complete the adding.

Note:

- You cannot input a parameter ID that is identical with those listed in the **Parameters** pane. Otherwise, a warning dialog box pops up, as shown in Figure 3-39.
- If the added ID is different from those existing in the NVDS, the added parameter is directly written to NVDS.
- If the ID of a to-be-added parameter already exists in NVDS and the two parameters with the same ID are of the same length, the to-be-added parameter is written to NVDS.
- If the ID of a to-be-added parameter already exists in NVDS but the two parameters with the same ID are of different lengths, the to-be-added parameter is not written to NVDS. Users need to modify the parameter length before writing it to NVDS.


USER Parameters

0x4001 existed existed parameter 1 10

ROM

Unfin

Add USER Parameter

ID 0x 4001

Parameter Name duplicated

Description duplicated paramter

Type Unsigned Integer

Length(Byte) 1

Value 20

[✓ dec] hex

OK Cancel

Warning

User Parameter ID must be identical.

OK

Figure 3-39 Failure to add a user parameter due to an identical parameter ID

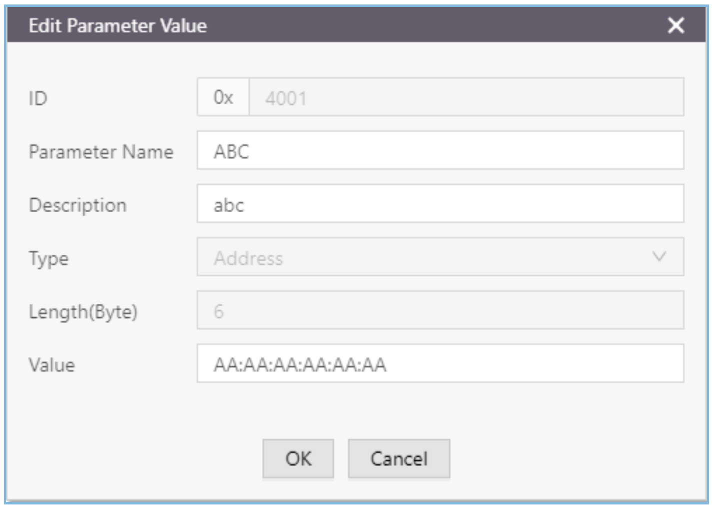
3.9.5 Modify NVDS Parameters

Users can modify both the **USER Parameters** and **ROM Parameters**.

ROM Parameters: You can modify the **Parameter Name**, **Description**, and **Value** of a ROM parameter. The modification on a parameter value does not lead to changes in the parameter length (except varying-length character strings).

USER Parameters: For user parameters in same and different states, the **Parameter Name**, **Description**, and **Value** can be modified. For user parameters in unfinished state, the **Type** and **Length(Byte)** can be modified.

Double-click a parameter to be modified, and edit the parameter information in the pop-up window. Click **OK** to write the modifications into NVDS.



Edit Parameter Value

ID	0x 4001
Parameter Name	ABC
Description	abc
Type	Address
Length(Byte)	6
Value	AA:AA:AA:AA:AA:AA

Figure 3-40 **Edit Parameter Value** window

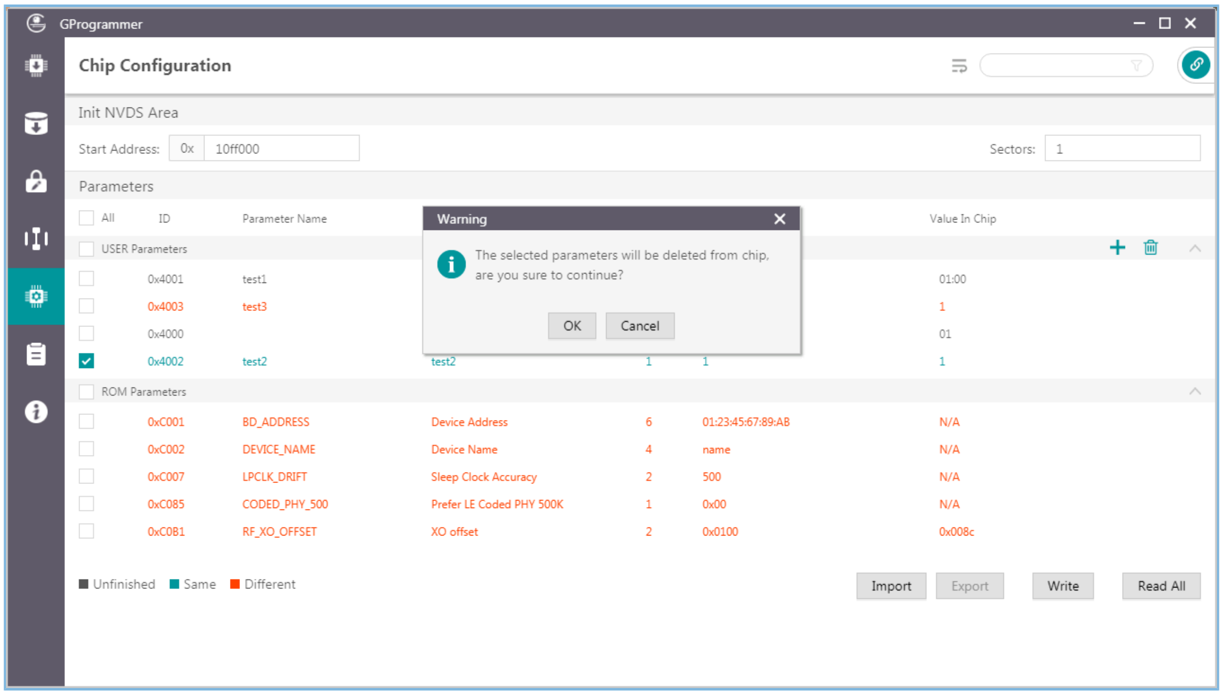
Note:

Parameters in unfinished state with a modified length that is different from that in the NVDS remain unfinished. Such parameters cannot be automatically written into the NVDS.

3.9.6 Remove a User Parameter

Users can remove user parameters only.

Select a parameter to be removed, and click Delete to remove the parameter from the NVDS.



Chip Configuration

Init NVDS Area

Start Address: 0x 10ff000

Sectors: 1

Parameters

All	ID	Parameter Name				Value In Chip
USER Parameters						
☐	0x4001	test1				01:00
☐	0x4003	test3				1
☐	0x4000					01
✓	0x4002	test2	test2	1	1	1
ROM Parameters						
☐	0xC001	BD_ADDRESS	Device Address	6	01:23:45:67:89:AB	N/A
☐	0xC002	DEVICE_NAME	Device Name	4	name	N/A
☐	0xC007	LPCLK_DRIFT	Sleep Clock Accuracy	2	500	N/A
☐	0xC085	CODED_PHY_500	Prefer LE Coded PHY 500K	1	0x00	N/A
☐	0xC0B1	RF_XO_OFFSET	XO offset	2	0x0100	0x008c

Warning

The selected parameters will be deleted from chip, are you sure to continue?

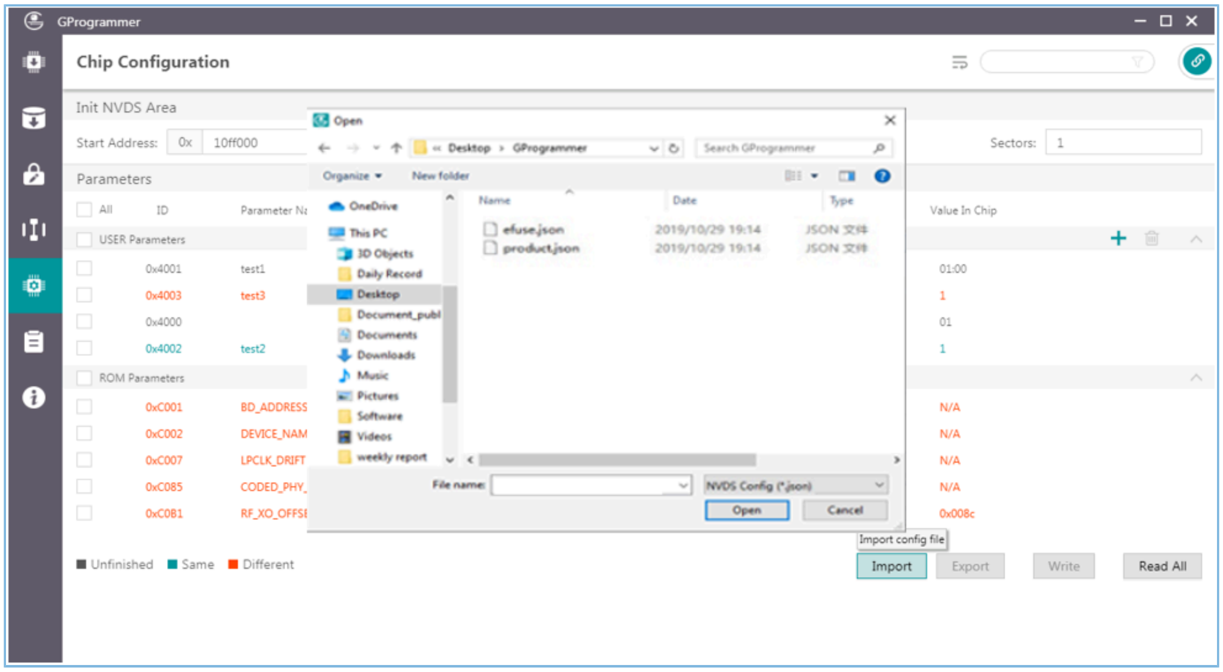
Figure 3-41 Removing a parameter

Tip:

- You can select more than one parameter and click Delete to implement a batch removal.
- When a ROM parameter is selected, **Remove** is unavailable (Delete is in grey).

3.9.7 Import and Export

GProgrammer allows users to export all parameter data (**Parameter Name**, **Description**, **Length**, and **Value**) to a local JSON configuration file as well as import local JSON configuration files to GProgrammer.



Start Address: 0x 10ff000

Sectors: 1

Desktop › GProgrammer

Search GProgrammer

Name	Date	Type
efuse.json	2019/10/29 19:14	JSON 文件
product.json	2019/10/29 19:14	JSON 文件

File name:

NVDS Config (*.json)

All	ID	Parameter Na	Value In Chip
USER Parameters			
	0x4001	test1	01:00
	0x4003	test3	1
	0x4000		01
	0x4002	test2	1
ROM Parameters			
	0xC001	BD_ADDRESS	N/A
	0xC002	DEVICE_NAM	N/A
	0xC007	LPCLK_DRIFT	N/A
	0xC085	CODED_PHY_	N/A
	0xC0B1	RF_XO_OFFSE	0x008c

■ Unfinished ■ Same ■ Different

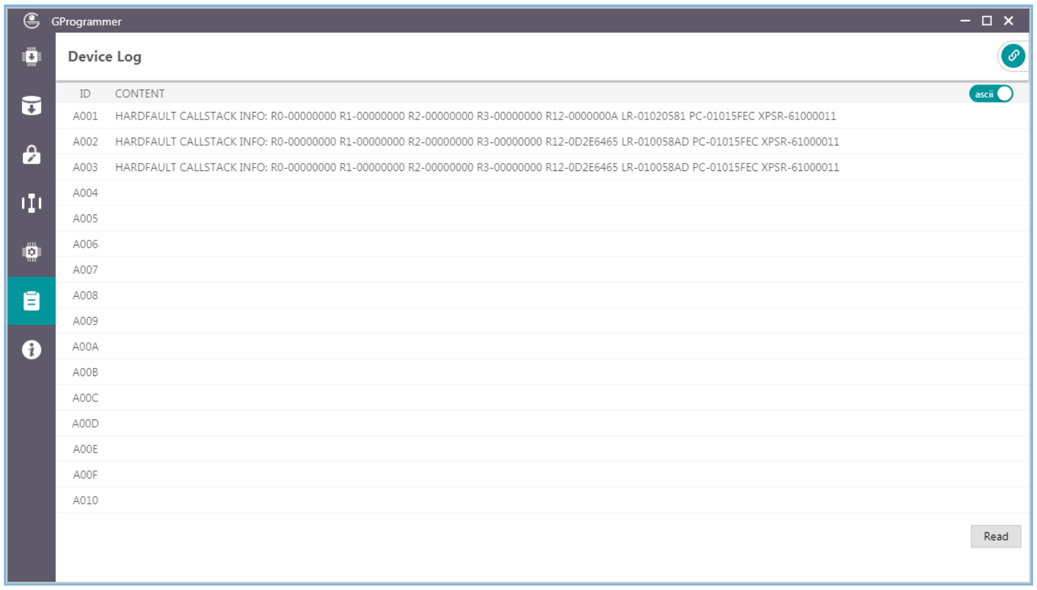
Figure 3-42 Importing local JSON configuration files to GProgrammer

Tip:

- Parameters in the imported JSON files replace all those listed in the **Parameters** pane.
- Export modified parameter data to a local JSON file to prevent repeated modification.
- **Export** is unavailable when parameters in unfinished state exist.
- All data displayed in the Chip Configuration interface can be exported by clicking **Export**.

3.10 Device Log

Click on the left side of the main interface of GProgrammer to open the **Device Log** interface.



ID	CONTENT
A001	HARDFAULT CALLSTACK INFO: R0-00000000 R1-00000000 R2-00000000 R3-00000000 R12-0000000A LR-01020581 PC-01015FEC XPSR-61000011
A002	HARDFAULT CALLSTACK INFO: R0-00000000 R1-00000000 R2-00000000 R3-00000000 R12-0D2E6465 LR-010058AD PC-01015FEC XPSR-61000011
A003	HARDFAULT CALLSTACK INFO: R0-00000000 R1-00000000 R2-00000000 R3-00000000 R12-0D2E6465 LR-010058AD PC-01015FEC XPSR-61000011
A004	
A005	
A006	
A007	
A008	
A009	
A00A	
A00B	
A00C	
A00D	
A00E	
A00F	
A010	

Figure 3-43 **Device Log** interface

Users can view device logs, mainly error information during SoC running, on GProgrammer. Click **Read** to retrieve the device logs.

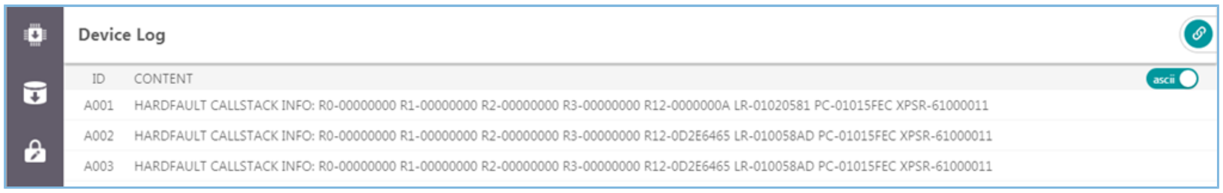
Note:

Prior to viewing device logs, make sure you have performed the following:

- Write device error code into the NVDS by using the application firmware (NVDS ID: A001–A010).
- Initialize the NVDS area correctly on GProgrammer, and the initialization result is identical with the value defined in the application firmware.

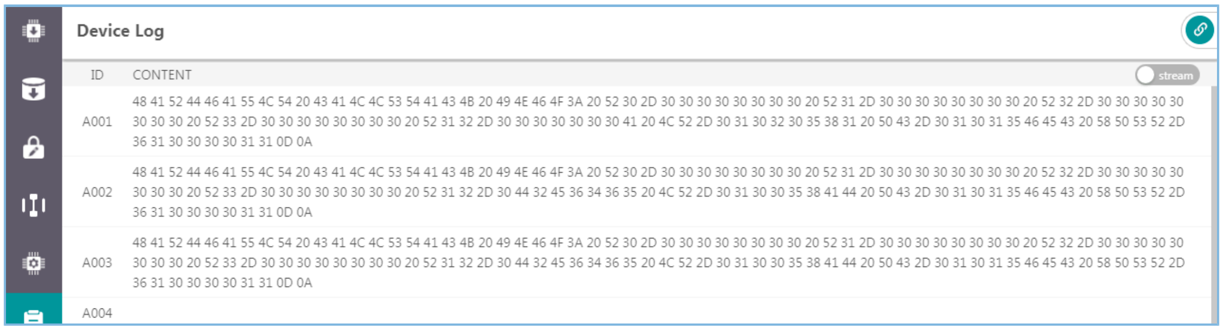
In the interface, click ascii or stream in the upper-right corner to switch the mode in displaying device logs between ASCII and stream.

- ascii: The device logs are displayed by ASCII character as shown in Figure 3-44.
- stream: The device logs are displayed by byte stream as shown in Figure 3-45.



ID	CONTENT
A001	HARDFAULT CALLSTACK INFO: R0-00000000 R1-00000000 R2-00000000 R3-00000000 R12-0000000A LR-01020581 PC-01015FEC XPSR-61000011
A002	HARDFAULT CALLSTACK INFO: R0-00000000 R1-00000000 R2-00000000 R3-00000000 R12-0D2E6465 LR-010058AD PC-01015FEC XPSR-61000011
A003	HARDFAULT CALLSTACK INFO: R0-00000000 R1-00000000 R2-00000000 R3-00000000 R12-0D2E6465 LR-010058AD PC-01015FEC XPSR-61000011

Figure 3-44 Device logs in ASCII characters



```
ID     CONTENT
A001   48 41 52 44 46 41 55 4C 54 20 43 41 4C 4C 53 54 41 43 4B 20 49 4E 46 4F 3A 20 52 30 2D 30 30 30 30 30 30 30 30 20 52 31 2D 30 30 30 30 30 30 30 30 20 52 32 2D 30 30 30 30 30
       30 30 30 20 52 33 2D 30 30 30 30 30 30 30 30 20 52 31 32 2D 30 30 30 30 30 30 30 41 20 4C 52 2D 30 31 30 32 30 35 38 31 20 50 43 2D 30 31 30 31 35 46 45 43 20 58 50 53 52 2D
       36 31 30 30 30 30 31 31 0D 0A
A002   48 41 52 44 46 41 55 4C 54 20 43 41 4C 4C 53 54 41 43 4B 20 49 4E 46 4F 3A 20 52 30 2D 30 30 30 30 30 30 30 30 20 52 31 2D 30 30 30 30 30 30 30 30 20 52 32 2D 30 30 30 30 30
       30 30 30 20 52 33 2D 30 30 30 30 30 30 30 30 20 52 31 32 2D 30 44 32 45 36 34 36 35 20 4C 52 2D 30 31 30 30 35 38 41 44 20 50 43 2D 30 31 30 31 35 46 45 43 20 58 50 53 52 2D
       36 31 30 30 30 30 31 31 0D 0A
A003   48 41 52 44 46 41 55 4C 54 20 43 41 4C 4C 53 54 41 43 4B 20 49 4E 46 4F 3A 20 52 30 2D 30 30 30 30 30 30 30 30 20 52 31 2D 30 30 30 30 30 30 30 30 20 52 32 2D 30 30 30 30 30
       30 30 30 20 52 33 2D 30 30 30 30 30 30 30 30 20 52 31 32 2D 30 44 32 45 36 34 36 35 20 4C 52 2D 30 31 30 30 35 38 41 44 20 50 43 2D 30 31 30 31 35 46 45 43 20 58 50 53 52 2D
       36 31 30 30 30 30 31 31 0D 0A
A004
```

Figure 3-45 Device logs in byte streams

3.11 Command-line Programs

Goodix provides two command-line programs in the GProgrammer installation directory: *GR5xxx_console.exe* and *GR5xxx_encrypt_signature.exe*.

Note:

GR5xxx, representing the name of SoC series, includes GR551x.

- *GR5xxx_console.exe* supports firmware download and flash programming in SoCs in a command-line interface.
- *GR5xxx_encrypt_signature.exe* supports firmware encryption and (or) signing in a command-line interface.

3.11.1 *GR5xxx_console.exe*

Follow the steps below to run *GR5xxx_console.exe*:

1. Open the **Command Prompt** window from the **Start** menu or by entering **cmd** in the **Run** window.
2. Navigate to the GProgrammer installation directory by using `cd` command.
3. Type the `GR5xxx_console.exe` command to complete corresponding operations. The details about the command are shown in Table 3-6.

Table 3-6 GR5xxx_console supported commands

Command	Functional Description	Command Format and Parameter Description	Remarks
program	Programs firmware files to internal SoC flash memories.	program <firmware file path> <run immediately:y | n> <flash start address(hex)> <flash size> <product type> Parameter description: • <firmware file path>: It sets the path of the to-be-downloaded firmware file. • <run immediately:y | n>: It decides on whether to run the firmware immediately after downloading.	The following parameters apply to all commands: • <flash start address(hex)>: It sets the start address in the Flash memories to which firmware files are downloaded. Value:
erase	Erases flash memory data within an SoC based on a specified address range.	erase <start address<hex>> <end address<hex>><force erase when conflict with firmware/bootinfo:y | n> <flash start address(hex)> <flash size> <product type>	

Command	Functional Description	Command Format and Parameter Description	Remarks
		Parameter description: • <start address<(hex)>>: It represents the start address of the storage area to be erased (in hexadecimal). • <end address<(hex)>>: It represents the end address of the storage area to be erased (in hexadecimal). • <force erase when conflict with firmware/bootinfo:y | n>: This parameter decides whether to forcibly erase the flash memory data when its address conflicts with that of firmware, Boot info, or NVDS.	◦ 0x01000000: for GR551x • <flash size>: It indicates the Flash size (unit: KB) of the selected SoC. For value details, see the **Flash** column in Figure 3-3. **Note:** For SoCs with 0 KB Flash, the external Flash size applies. • <product type>: It indicates the SoC series. Valid value and description: ◦ 0: GR551x
eraseall	Erases all flash memory data within an SoC.	eraseall <product type>	
download	Downloads data files to internal SoC flash memories.	download <data file path> <start address<(hex)>> <force download when conflict with firmware/ bootinfo:y | n> <flash start address(hex)> <flash size> <product type> Parameter description: • <data file path>: It sets the path of the to-be-downloaded data file. • <start address<(hex)>>: It represents the start address of the download area (in hexadecimal). • <force download when conflict with firmware/ bootinfo:y | n>: This parameter decides whether to forcibly download the data files to internal SoC flash memories when their addresses conflict with that of firmware or Boot info.	
writeefuse	Writes Encrypt Key Info and Mode Control files to eFuse.	writeefuse <Encrypt Key Info file Path> <Mode Control file Path> <product type> Parameter description: • <Encrypt Key Info file Path>: It sets the path of Encrypt Key Info file. • <Mode Control file Path>: It sets the path of Mode Control file.	
reset	Resets the GR551x SoC.	reset <product type>	
generate	Converts firmware files into BIN files that can be used by the SoC	generate <input firmware file path> <output firmware file path> <flash start address(hex)> <flash size> <product type> Parameter description:	

Command	Functional Description	Command Format and Parameter Description	Remarks
		• <input firmware file path>: It indicates the path of an imported file (HEX/BIN file generated via integrated development environment tools). • <output firmware file path>: It indicates the output file of an exported file.	
help	Displays all help information.	help	

Take GR551x SoC as an example. The code below shows how to use the `program` command to download a firmware file to SoC flash memories and run the firmware immediately after downloading. Command line:

```
GR5xxx_console.exe program "D:/test/test_fw.bin" y "0x01000000" 1024 0
```

The parameter descriptions are listed below:

- "D:/test/test_fw.bin": It indicates the path for the to-be-downloaded firmware BIN file
- "0x01000000" 1024 0: It represents the start address in Flash to which the firmware is downloaded (0x01000000), the Flash size (1024 KB), and SoC model (GR551x) respectively.

The downloading progress is displayed in real time during executing the `program` command.

Note:

You cannot operate *GR5xxx_console.exe* while GProgrammer is running.

3.11.2 *GR5xxx_encrypt_signature.exe*

Follow the steps below to run *GR5xxx_encrypt_signature.exe*:

1. Open the **Command Prompt** window from the **Start** menu or by entering **cmd** in the **Run** window.
2. Navigate to the GProgrammer installation directory by using `cd` command.
3. Type `GR5xxx_encrypt_signature.exe --parameter` to complete corresponding operations.

   For most frequently used parameters, see Table 3-7. To view all parameters, enter `GR5xxx_encrypt_signature.exe --help`.

Table 3-7 Frequently used parameters for *GR5xxx_encrypt_signature.exe*

Parameter	Description	Remarks
operation	Indicates the operation type. Options: • encryptandsign: Encrypt and sign firmware. • sign: Sign firmware only	
firmware_key	Shows the directory of *firmware.key*, which is used for firmware encryption and signing, or signing only.	The directories correspond to the paths you have set when you click **Generate eFuse**

Parameter	Description	Remarks
signature_key	Shows the directory of *sign.key*, which is used for firmware encryption and signing, or signing only.	**File** in "Section 3.7.1 eFuse Settings".
signature_pub_key	Shows the directory of *sign_pub.key*, which is used for firmware encryption and signing, or signing only.	
product_json_path	Shows the directory of *product.json*, which is used for firmware encryption and signing, or signing only.	
rand_number	Shows the directory of *random.bin*, which is used for firmware encryption and signing, or signing only.	
ori_firmware	Shows the directory that saves the firmware before encryption and signing, or signing only.	
output	Shows the directory that saves the firmware after encryption and signing, or signing only.	
base_addr	Sets the start address in the Flash memories to which firmware files are downloaded. Value: • 0x01000000: for GR551x	
flash_size	Indicates the Flash size (unit: KB) of the selected SoC. For value details, see the **Flash** column in Figure 3-3. **Note:** For SoCs with 0 KB Flash, the external Flash size applies.	
product_type	Indicates the SoC series. Valid value and description: • 0: GR551x	
random_output	Shows the directory that saves the random numbers used in firmware encryption and signing, or signing only.	
help	Displays help information.	

Take GR551x SoC as an example. The code below shows how to encrypt and sign firmware by using *GR5xxx_encrypt_signature.exe*:

```
GR5xxx_encrypt_signature.exe --operation="encryptandsign" --firmware_key="D:/test/eFuse/
firmware.key" --signature_key="D:/test/eFuse/sign.key" --signature_pub_key="D:/test/eFuse/
sign_pub.key" --product_json_path="D:/test/eFuse/product.json" --ori_firmware="D:/test/
firmware/test_fw.bin" --output="D:/test/firmware_encryptAndSign/test_fw_encryptAndSign.bin"
 --random_output="D:/test/firmware_encryptAndSign/random.bin" --base_addr="0x01000000" --
flash_size="1024" --product_type="0"
```

In the code snippet above, the **D:/test/eFuse/** directories show the user-defined folders where files are saved after users click **Generate eFuse File**, as described in "Section 3.7.1 eFuse Settings". For descriptions of other parameter, see Table 3-7.

- --ori_firmware="D:/test/firmware/test_fw.bin": the directory of the firmware before any operation

- --output="D:/test/firmware_encryptAndSign/test_fw_encryptAndSign.bin": the directory of the encrypted and signed firmware
- --base_addr="0x01000000" --flash_size="1024" --product_type="0": the start address in Flash to which the firmware is downloaded (0x01000000), the Flash size (1024 KB), and SoC model (GR551x) respectively

Run the command to encrypt and sign the firmware.

3.11.3 User-defined Windows Scripts

Users can also write custom scripts on Windows to call command-line programs. Two sample script files are provided in the GR5xxx_script file in the GProgrammer installation directory.

Note:

GR5xxx, representing the name of SoC series, includes GR551x.

---

encryptAndSignatureFirmware.bat can encrypt and sign firmware with *firmware_origin.bin* in the same directory and the files saved in the eFuse directory. The encrypted and signed firmware is available in `firmware_encryptAndSign\firmware_encryptAndSign.bin`.

program_Firmware_EncryptAndSign.bat can erase all internal flash memories, and download the firmware `firmware_encryptAndSign\firmware_encryptAndSign.bin` and save the firmware file in the internal flash memories.

3.12 Help

Click on the left side of the main interface of GProgrammer to open the **Help** interface.

GProgrammer offers help and support to users.

- **About GProgrammer**

  This section provides version information and features of GProgrammer.

- **Feedback**

  If you have any questions or suggestions, please send an email to *software@reg.goodix.com*.

- **About Goodix**

  For more information, please visit Goodix official website: www.goodix.com.