GProgrammer User Manual

Version: 3.2

Release Date: 2024-08-21

Shenzhen Goodix Technology Co., Ltd.

Trademarks and Permissions

GOODiX and other Goodix trademarks are trademarks of Shenzhen Goodix Technology Co., Ltd. All other trademarks and trade names mentioned in this document are the property of their respective holders.

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: Floor 13, Phase B, Tengfei Industrial Building, Futian Free Trade Zone, Shenzhen, China

TEL: +86-755-33338828 Zip Code: 518000

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:

- Device user
- Developer
- Test engineer
- Technical support engineer

Release Notes

This document is the Eighteenth release of *GProgrammer User Manual*, corresponding to GProgrammer V2.0.1.

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	• Added GR5526 to descriptions specific to SoC models supported by GProgrammer. • 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*.
2.6	2023-01-19	Deleted the GR5515I0ND SoC.
2.7	2023-02-03	• Updated description in "GR5xxx_console.exe" and added commands supported by *GR5xxx_console.exe*, including "device" and "load". • Added description on the parameter "rand_number" to *GR5xxx_console.exe* commands. • Added description on the software package *GProgrammer- Version.tar.bz2* (running on Linux). • Added a tip in "Flash Configuration" in "External Flash". • Added description on viewing/reselecting the chip model in use. • Added description on refreshing and choosing the serial number of target boards in SWD connection mode. • Added more data file formats (except BIN files) support during downloading data to Flash memories. • Added the function button **Parse Para** for parsing data in the NVDS area.
2.8	2023-03-30	• Added **Detect USB** for UART connection of GR5526. • Optimized the data download to the Boot info space of GR5526. • Added the functionality to configure the ID of an NVDS parameter. • Updated commands supported by *GR5xxx_console.exe* and added the "dump" command. • Replaced the email for feedback with the Developer Community website.
2.9	2023-04-20	• Updated descriptions about GR5xx SoCs. • Added **OTP Layout** for GR533x SoCs.
3.0	2023-09-22	• Updated SoC models. • Updated descriptions in "Installation Steps" and "Hardware Connection". • Updated descriptions about limitations to forcible download in "Download Data". • Updated *GR5xxx_console.exe* commands and related parameters. • Introduced the ECDSA signature algorithm. • Added "FAQ".
3.1	2023-09-24	• Updated SoC models. • Added the firmware **Sign** functionality to GR553x SoCs.
3.2	2024-08-21	• Updated SoC models.

Version	Date	Description
		• Updated "GR5xxx_console". • Updated "FAQ".

Contents

1 Introduction

GProgrammer is a firmware programming tool that applies to Bluetooth Low Energy (Bluetooth LE) GR5xx System-on-Chips (SoCs). It 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
- Viewing One-time Programmable (OTP) contents
- Firmware encryption and signing
- Configuring Non-Volatile Data Storage (NVDS) parameters
- Displaying device logs
- Programming on GR5xxx_console

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

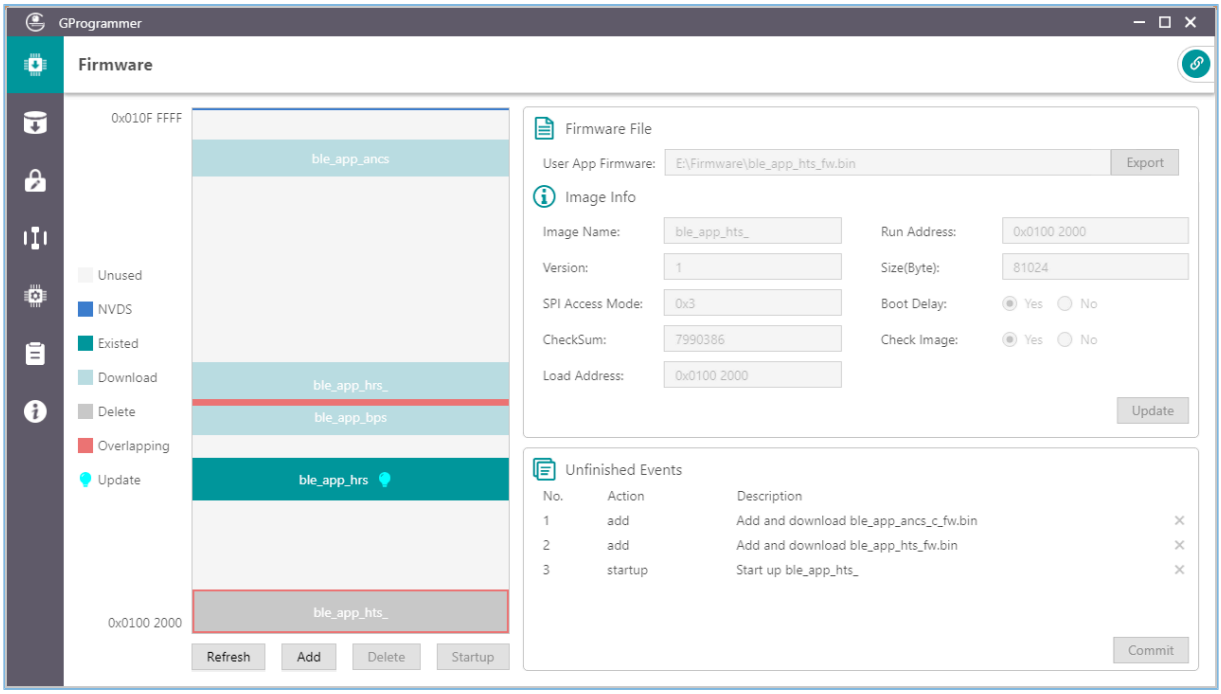

Figure 1-1 GProgrammer GUI

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)
Linux	Ubuntu 22.04 (64-bit)

2.2 Installation Steps

GProgrammer can be installed and run on both Windows and Linux.

- **Install GProgrammer.**
  - Windows: GProgrammer is provided in an executable installation package *GProgrammer_Windows_Version.exe*.
  - Linux: GProgrammer is installation-free in *GProgrammer_Linux_x64_Version.tar.bz2*. You can directly unzip the ZIP file and then double-click *gprogrammer* to launch GProgrammer.

Note:

Version indicates the GProgrammer software version number.

- **Installation steps:**

  On Windows, you can follow the steps below to install GProgrammer:

  1. Double-click *GProgrammer_Windows_Version.exe* to enter the installation interface (as shown in Figure 2-1), and follow the steps in the **GProgrammer Setup** wizard .

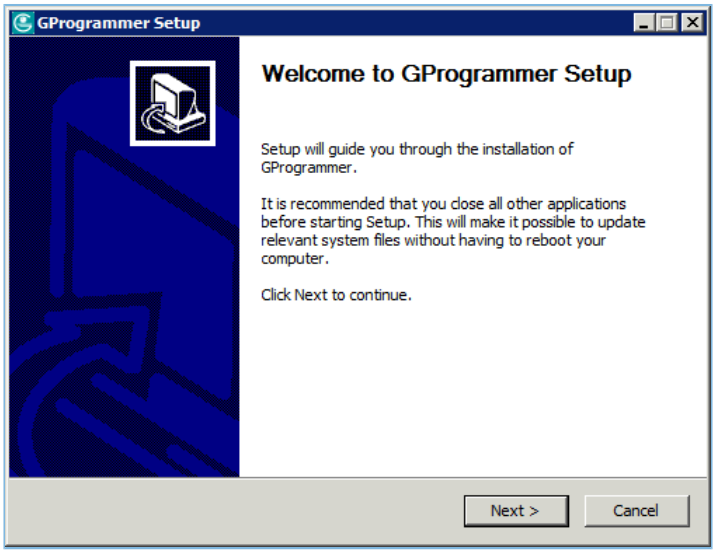

Figure 2-1 **GProgrammer Setup** installation wizard

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

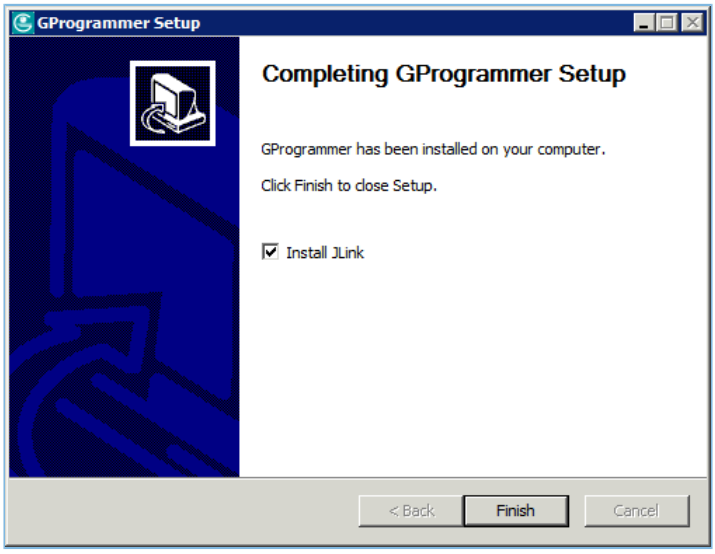

Figure 2-2 Prompt to install J-Link

Note:

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 GProgrammer Use Instructions

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, use a J-Link emulator with one end connecting to the PC through a USB cable and the other end connecting to SoC pins of the target board through Dupont wire cables.

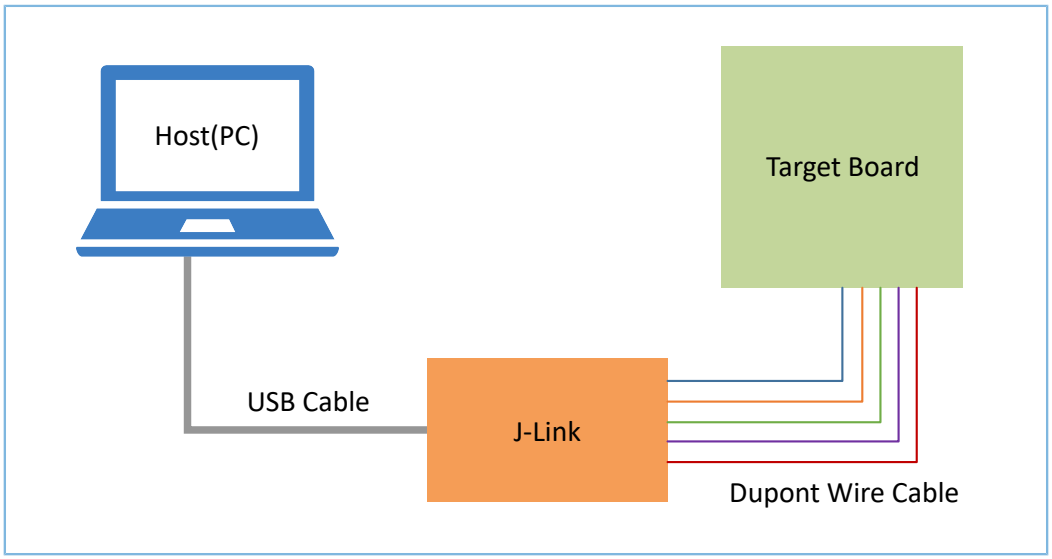

Figure 3-1 Hardware connection in SWD mode

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
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 USB cable.

- UART mode

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

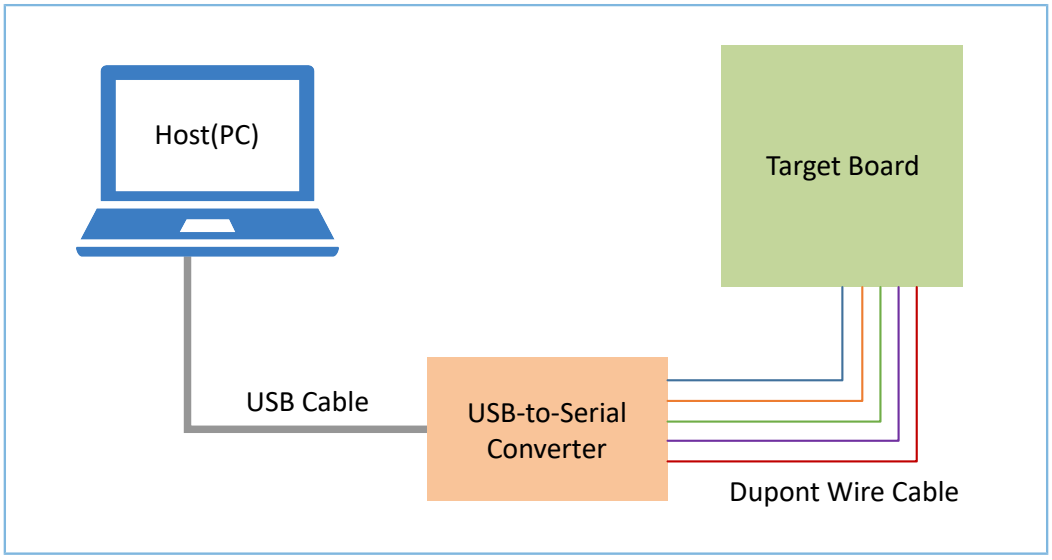

Figure 3-2 Hardware connection in UART mode

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

Note:

- 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 USB cable.
- When the SWD interface is closed, firmware information in the Flash is to be erased after the Host and the target board is connected through UART.

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**.

Note:

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

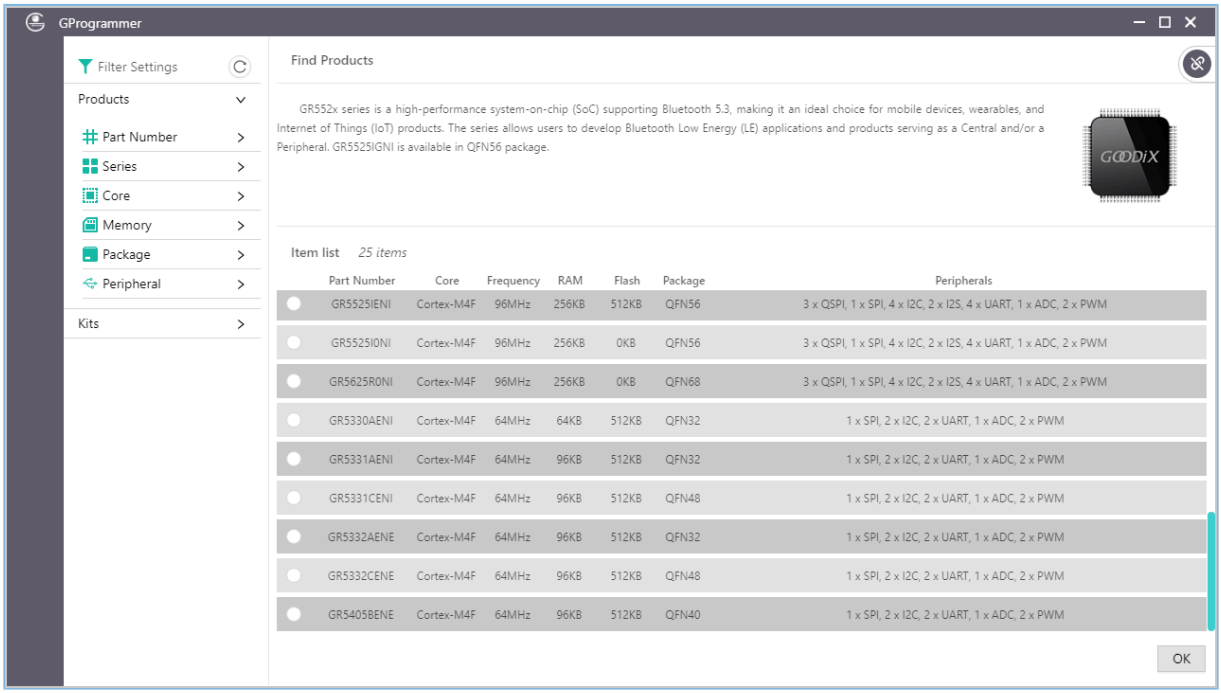

Figure 3-3 SoC selection interface

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**.

Note:

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.

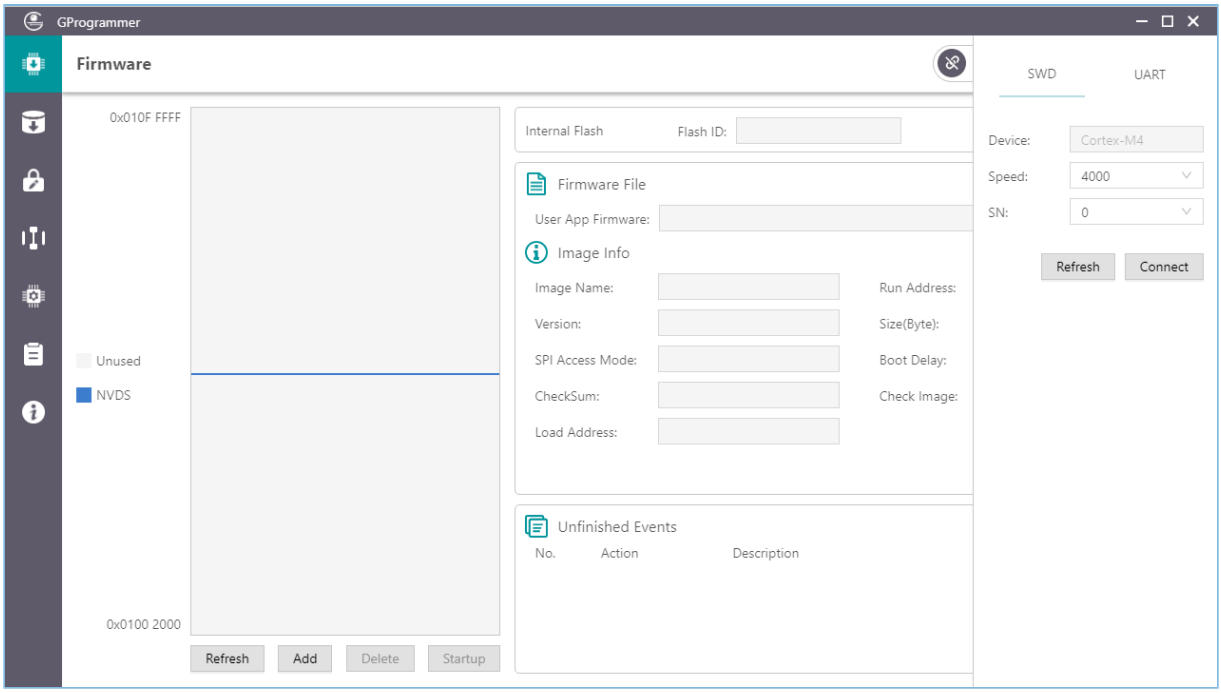

Figure 3-4 GProgrammer GUI

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
	-	View/Reselect the chip model in use. • You can view the selected chip model by moving the cursor onto this icon. • You can select another chip model by clicking this icon to return to the SoC selection interface. The current device is to be disconnected after clicking **OK** in the lower-right corner.
	Firmware	Display firmware-related operations.
	Flash	Display operations related to Flash memory.
	Encrypt & Sign	Display operations related to firmware encryption and signing.
	eFuse Layout	Display eFuse layout.
	Chip Configuration	Display operations related to chip configurations.
	Device Log	Display device logs.
	Help	Display help information.

Note:

GR533x and GR5405 support OTP, so an **OTP Layout** interface ( ) is added.

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 parameters below and click **Connect** to connect the target board to the host.

Table 3-4 Parameter description

Parameter	Description
Device	CPU of the on-board chip. It is **Cortex-M4** by default and cannot be modified.
Speed	Data transfer rate. The default value is **4000** kHz.
SN	Serial number of the target board. The default value is **0**. ◦ When the PC is connected with only one target board, you can keep the default value "0" or obtain the corresponding serial number by clicking **Refresh**. ◦ When the PC is connected with multiple target boards, you should obtain the corresponding serial numbers by clicking **Refresh**, and then choose the target one. In this case, if you keep the default value **0** and start device connection, a window will pop up to inform you of choosing the target board S/N when GProgrammer runs on Windows.

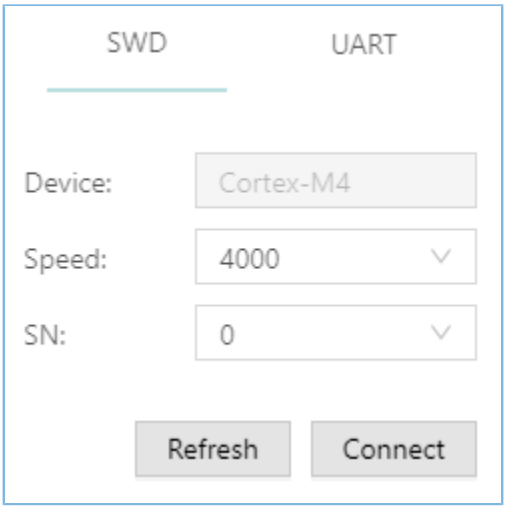

Figure 3-5 GProgrammer SWD connection

- 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.

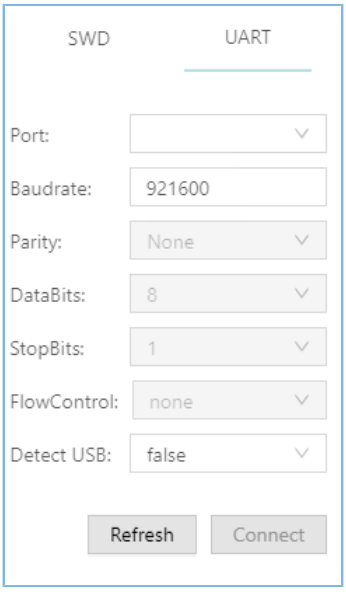

Figure 3-6 GProgrammer UART connection

Note:

The parameter **Detect USB** is applicable to USB connection for GR5526 only. Setting **Detect USB** to **true** will enable USB device detection. After clicking **Refresh**, you will be prompted to reset devices, and GProgrammer will search USB devices (a USB device will be enumerated as a serial device) and add the available devices to the **Port** list. Setting **Detect USB** to **false** will disable USB device detection. After you click **Refresh**, GProgrammer will add all serial devices to the **Port** list.

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**.

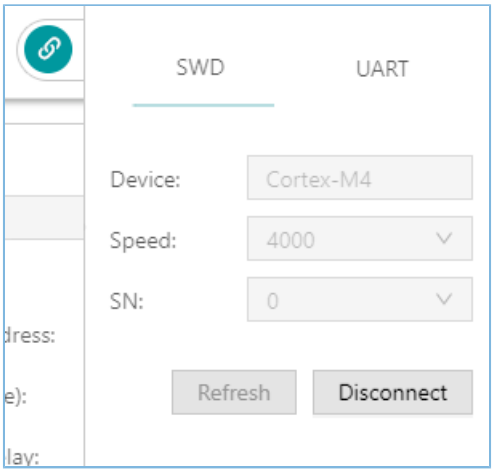

Figure 3-7 Clicking **Disconnect** on GProgrammer

3.5 Firmware

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

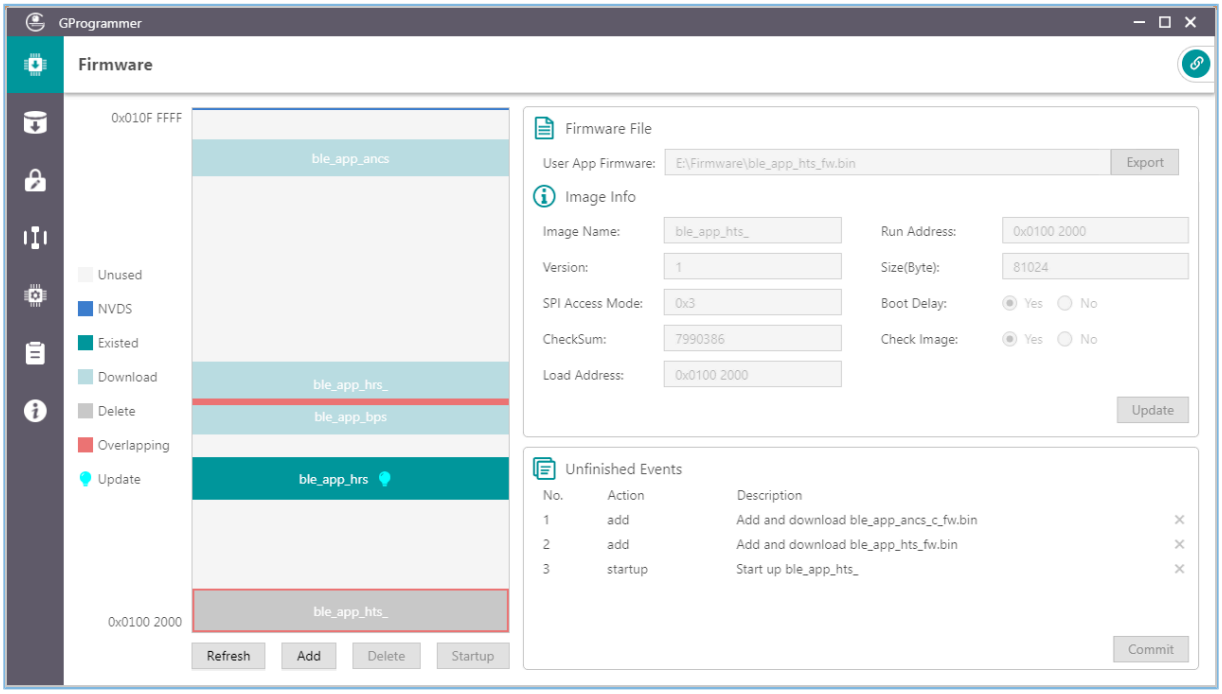

Figure 3-8 GProgrammer **Firmware** interface

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.

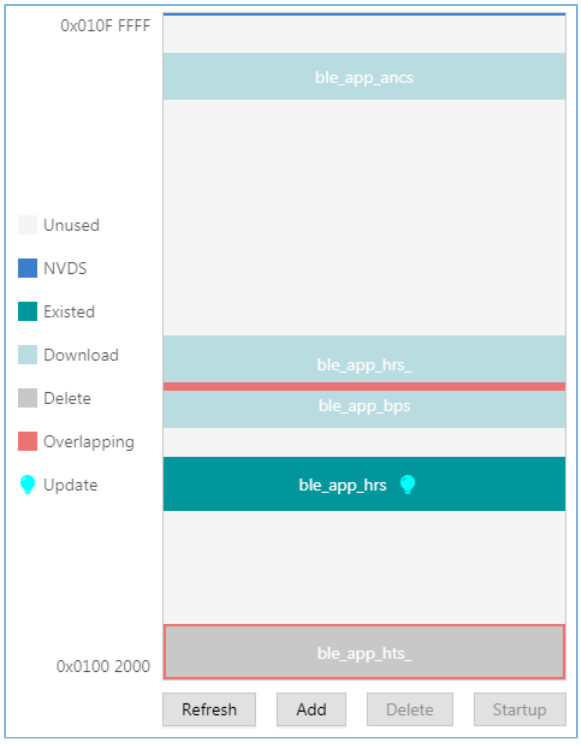

Figure 3-9 Flash firmware layout

- 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:

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

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.

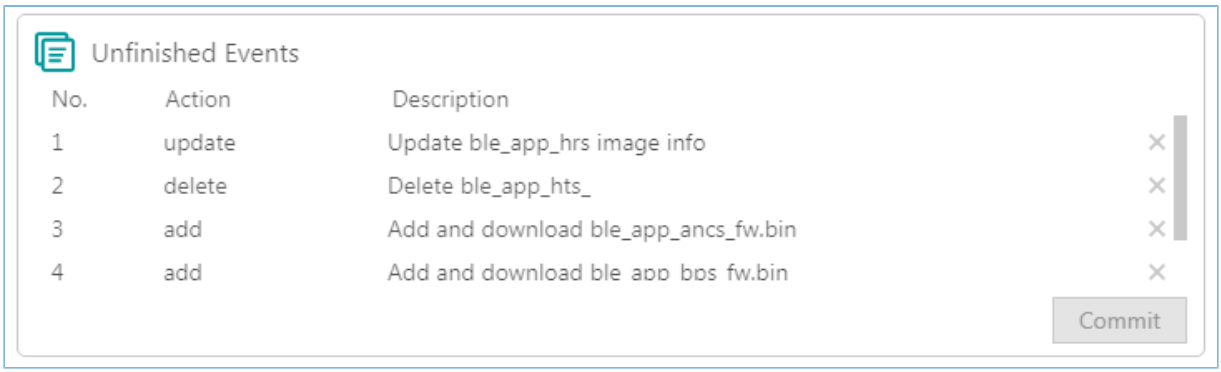

Figure 3-10 Action order

Executable actions for users are listed in the table below.

Table 3-5 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:** 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

Name	Button/Icon	Description
		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.

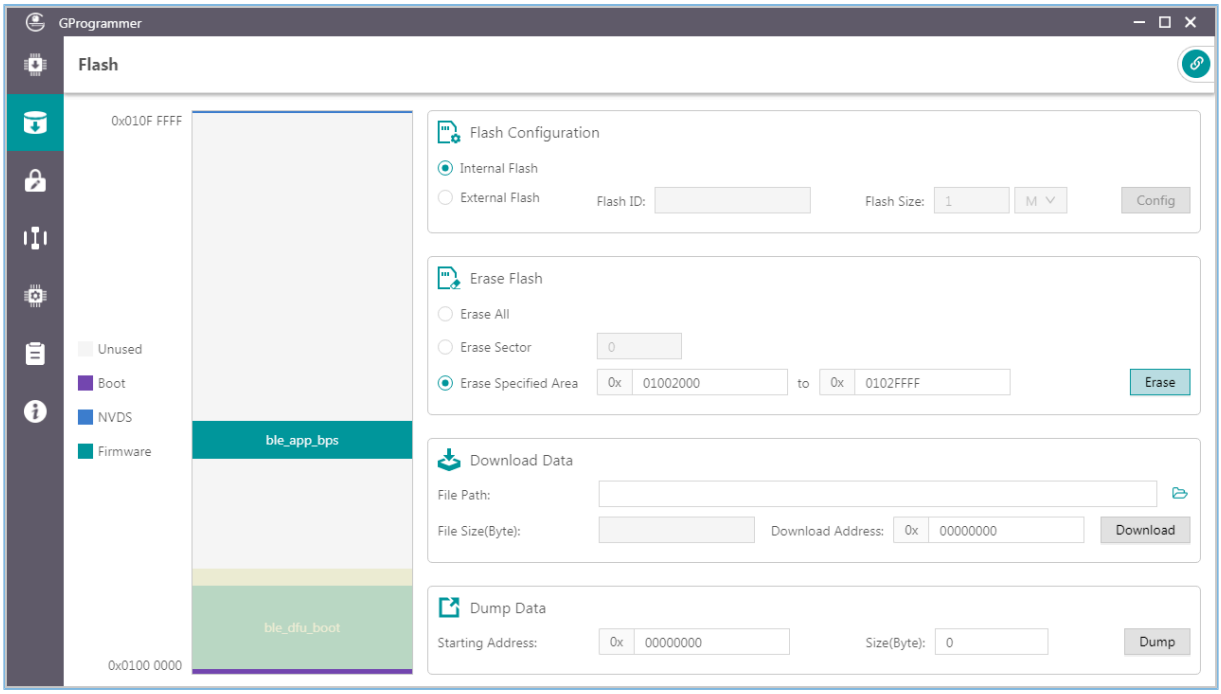

Figure 3-11 GProgrammer **Flash** interface

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.

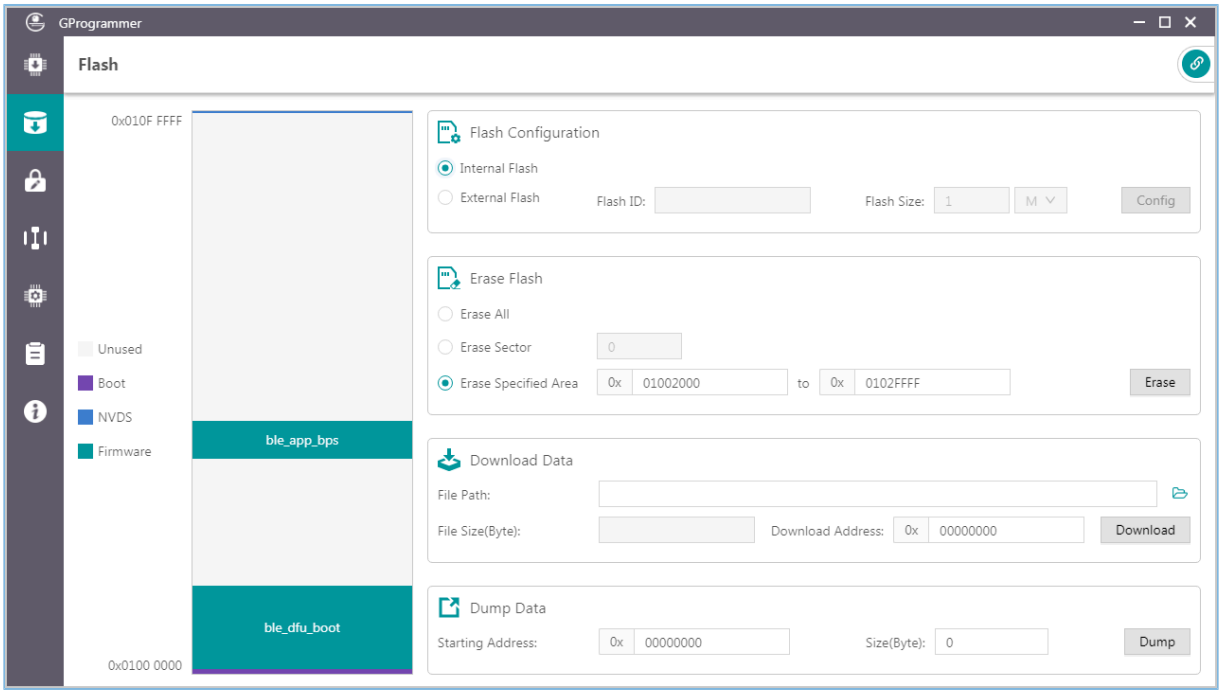

Figure 3-12 Selecting **Internal Flash**

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 spaces.

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

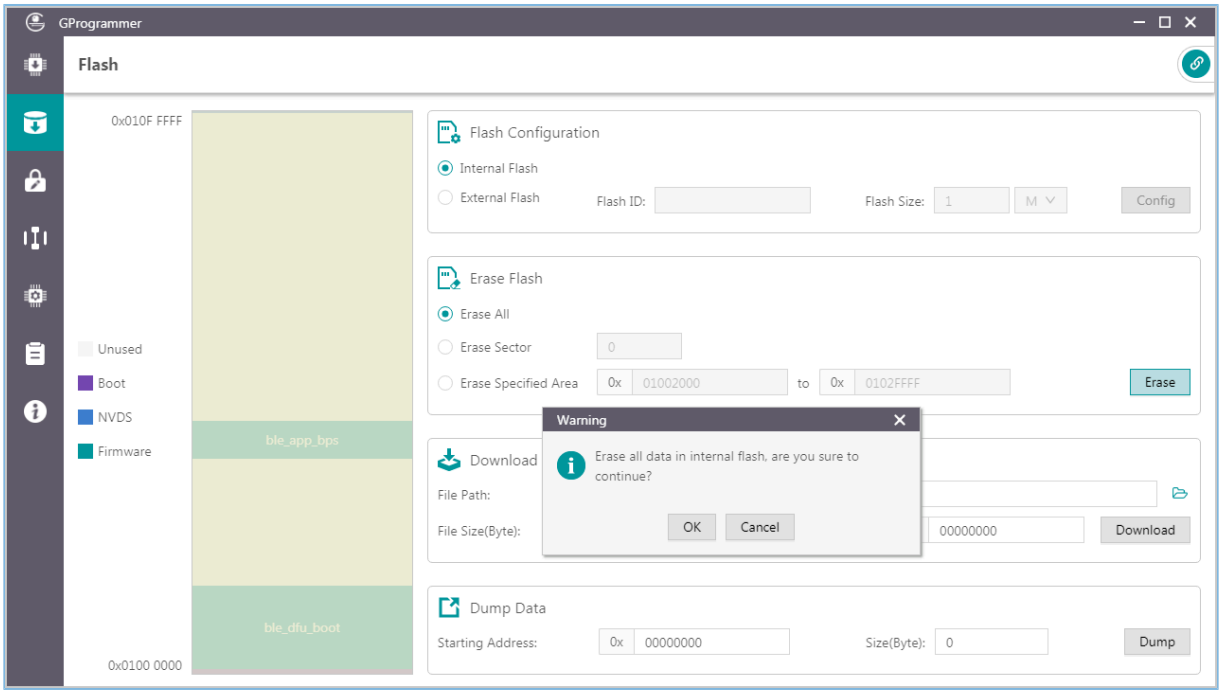

Figure 3-13 **Erase All** on GProgrammer

- **Erase Sector**

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

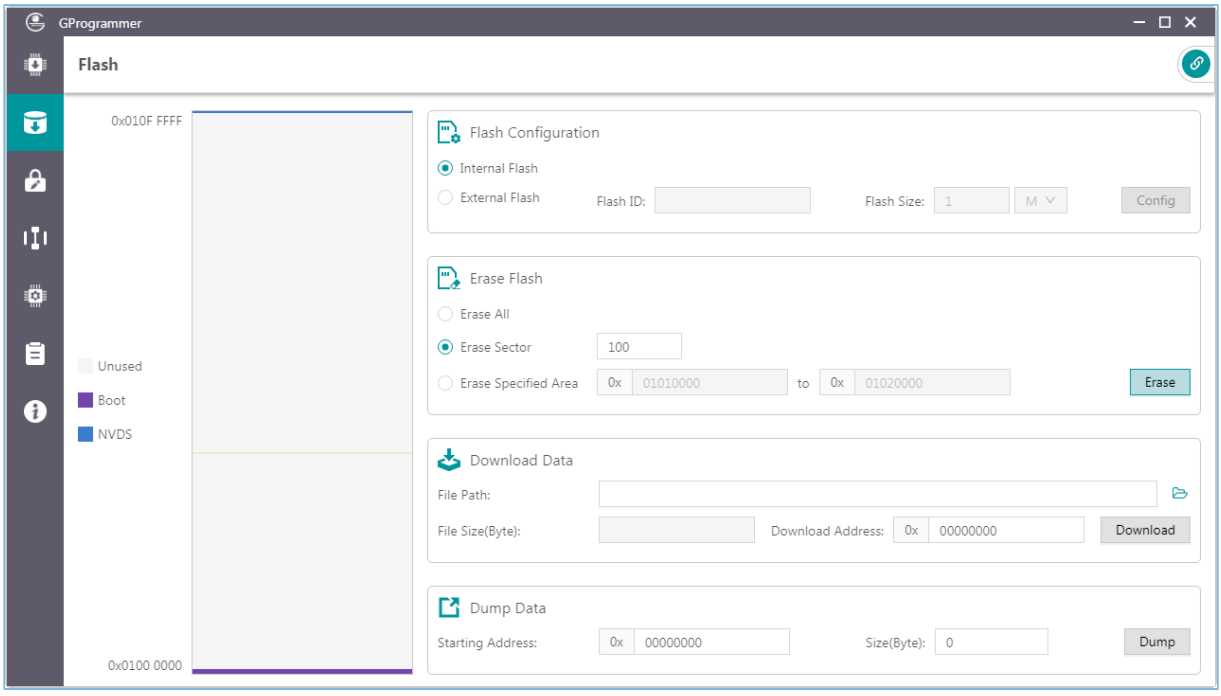

Figure 3-14 **Erase Sector** on GProgrammer

- **Erase Specified Area**

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

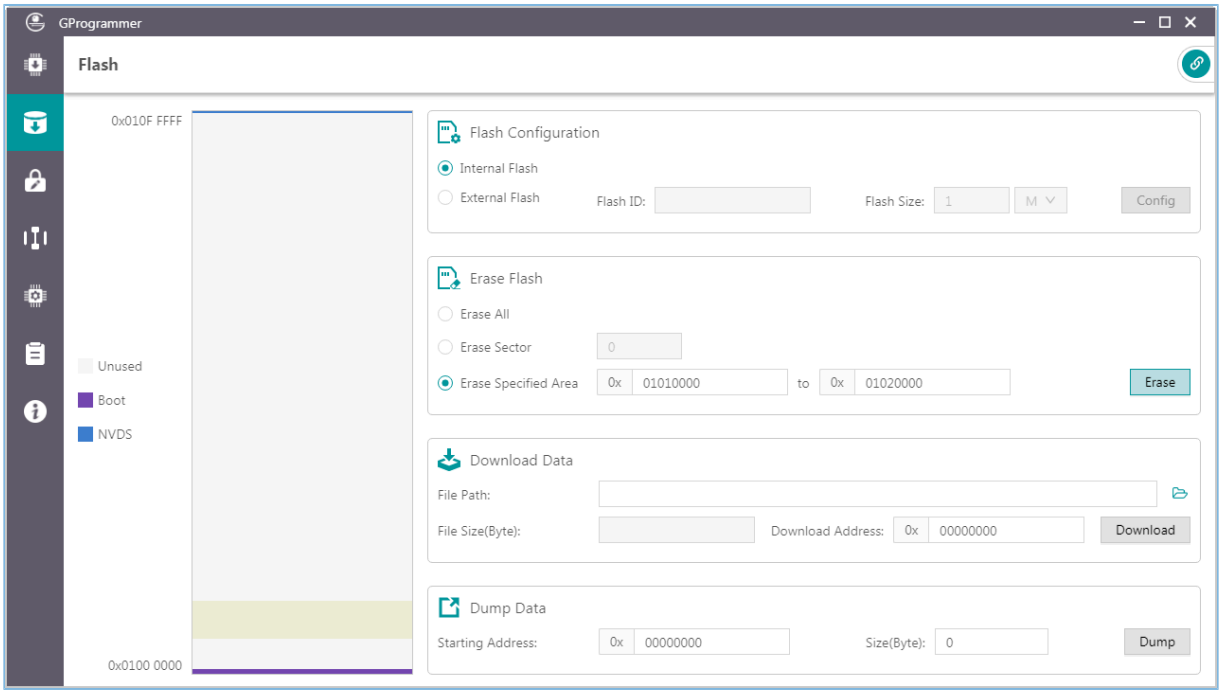

Figure 3-15 **Erase Specified Area** on GProgrammer

3.6.1.3 Download Data

When downloading data to Flash memories on GProgrammer, users only need to view and add the data file, as well as set a start address for downloading in **Download Address**.

Tip:

The download address shall be 4 KB-aligned.

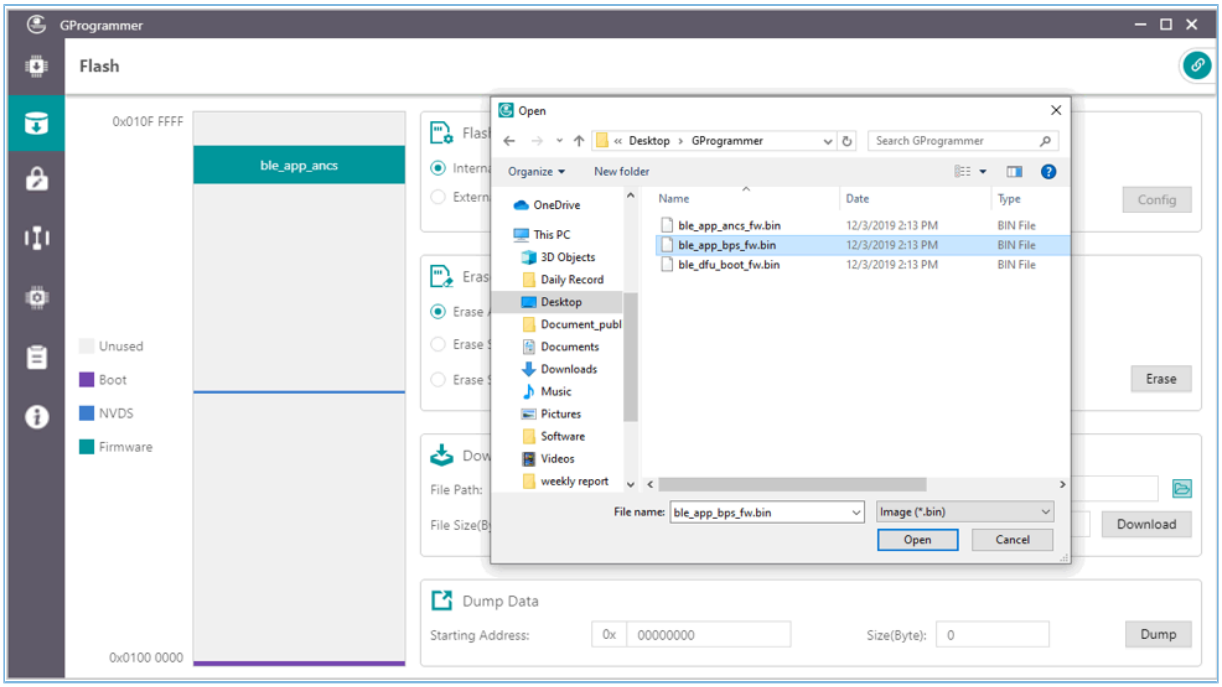

Figure 3-16 Viewing and selecting a data file to be downloaded

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

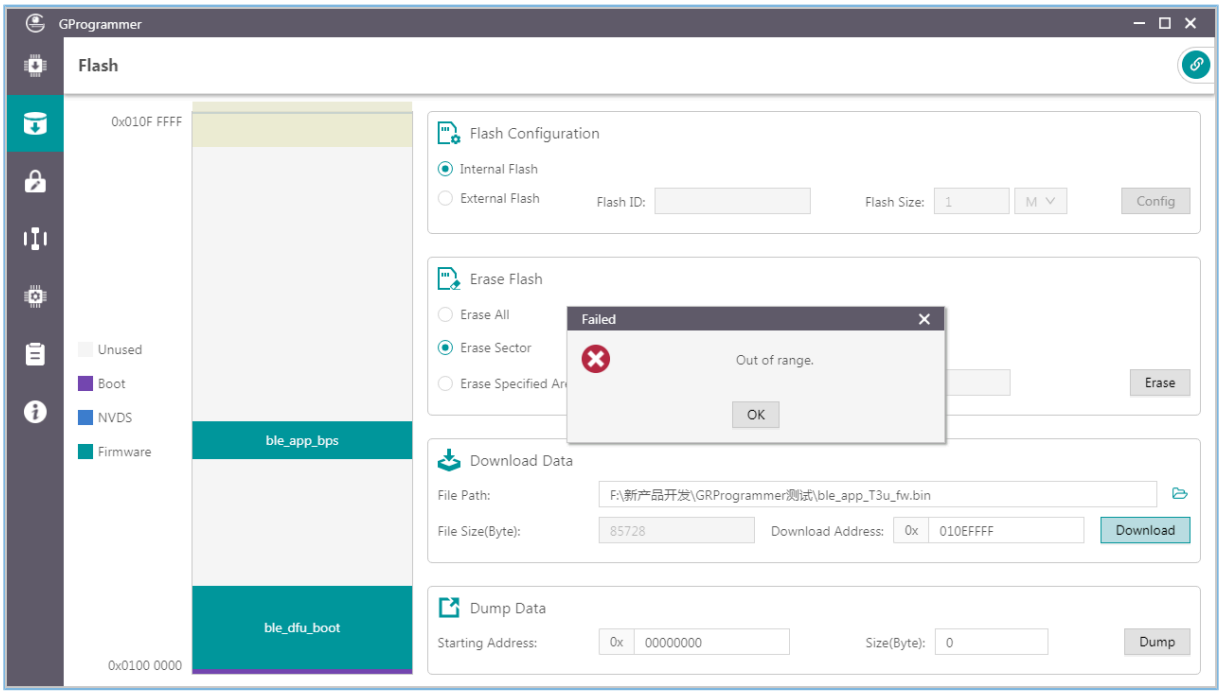

Figure 3-17 Flash overflow error

Note:

- In SWD connection mode, users are allowed to forcibly download data to the Boot info space (SCA).
- In UART mode, forcible download to the Boot info space is prohibited (for GR551x and GR5526 SoCs only).

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.

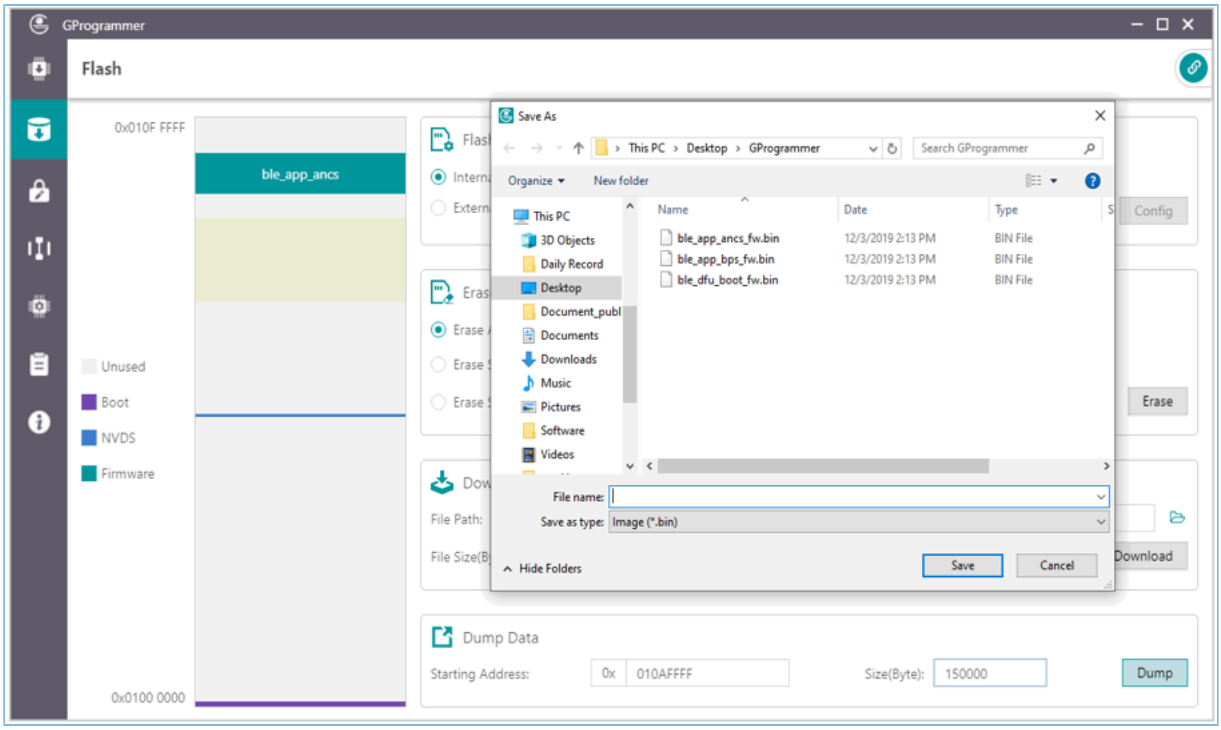

Figure 3-18 **Dump Data** on GProgrammer

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.

Note:

- 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.
- GPIO_0 and GPIO_1 are for device connection, and if they are configured as external Flash pins, GProgrammer will disconnect from the target board.

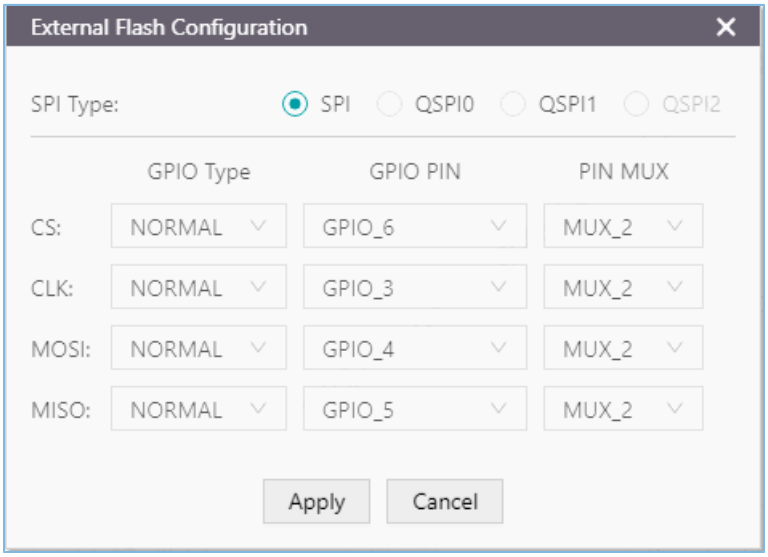

Figure 3-19 SPI configurations

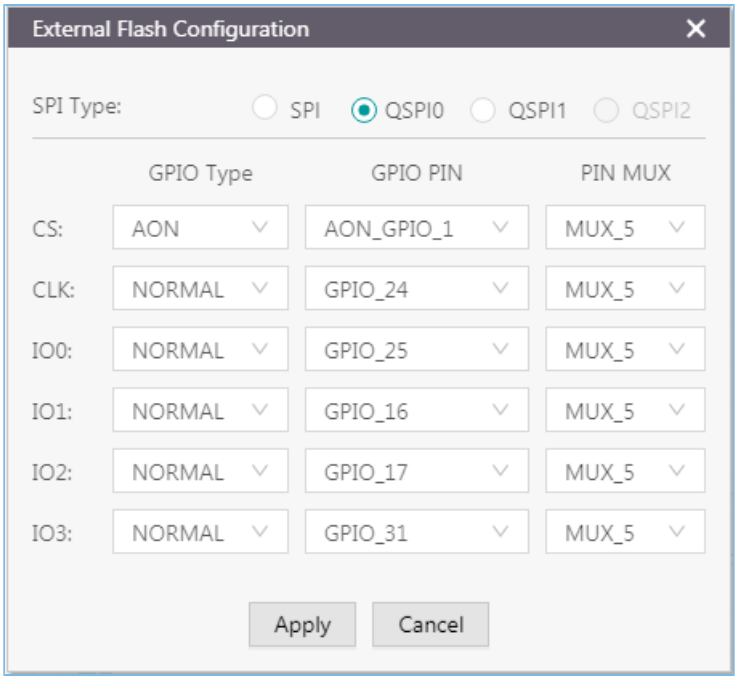

Figure 3-20 QSPI0 configurations

- 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.

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

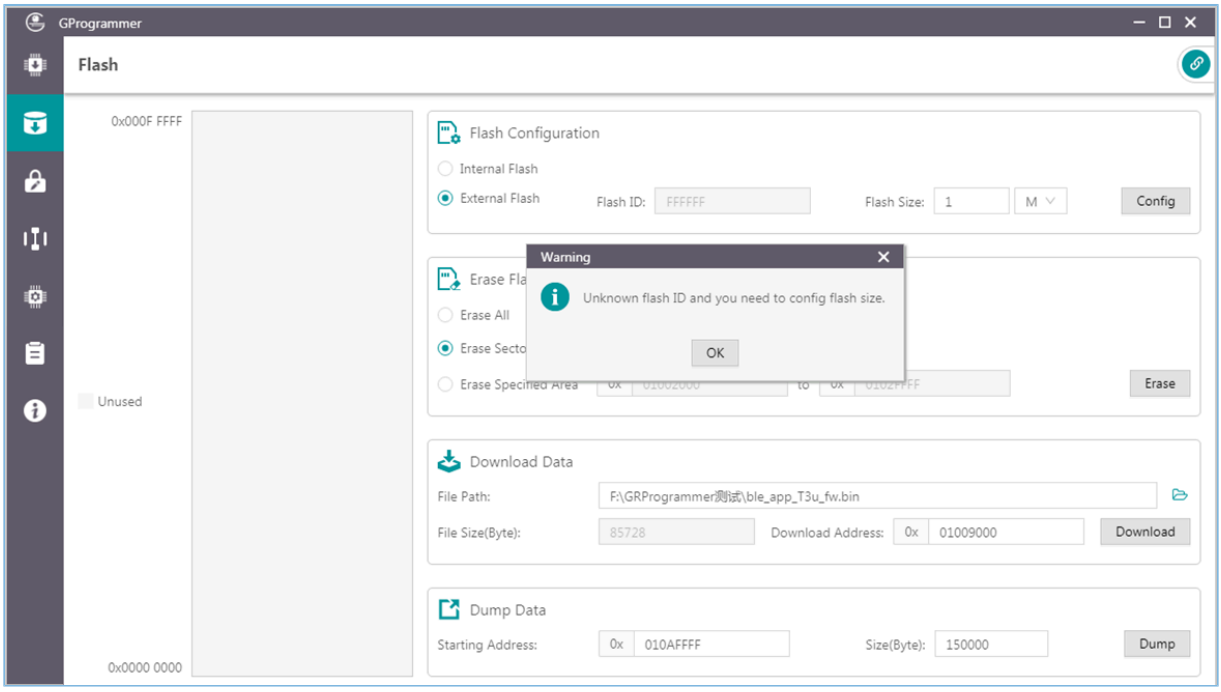

Figure 3-21 Unknown Flash ID

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.

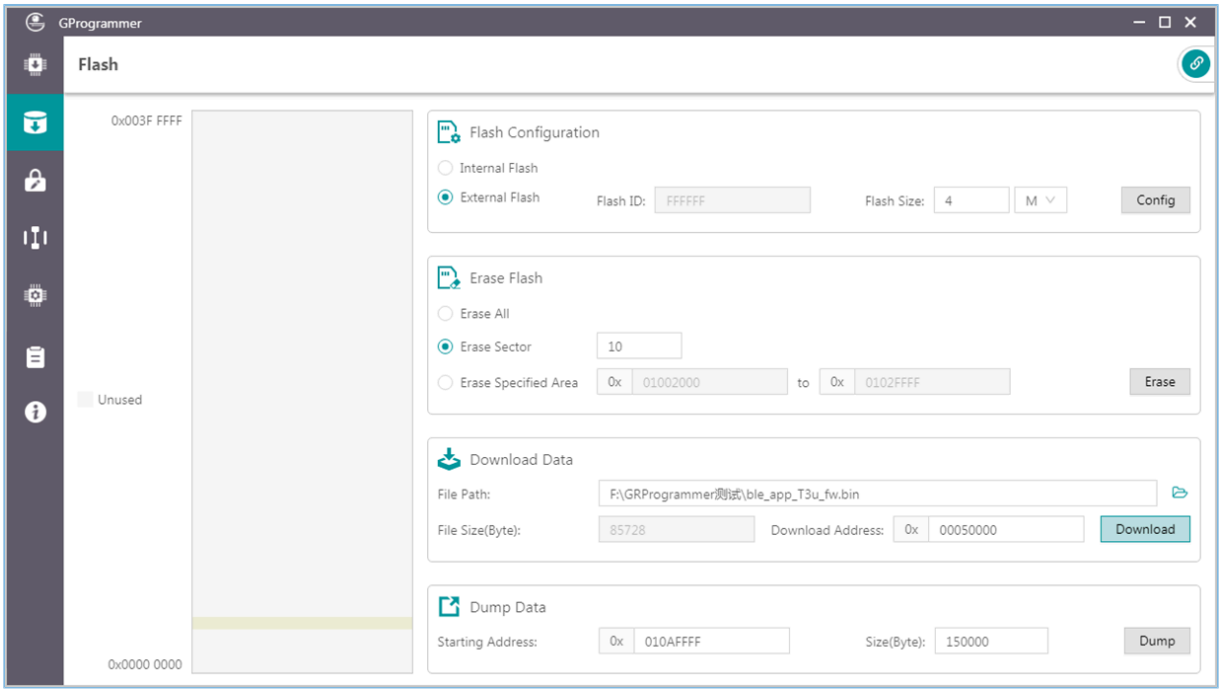

Figure 3-22 **Download Data** to external Flash on GProgrammer

Note:

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.

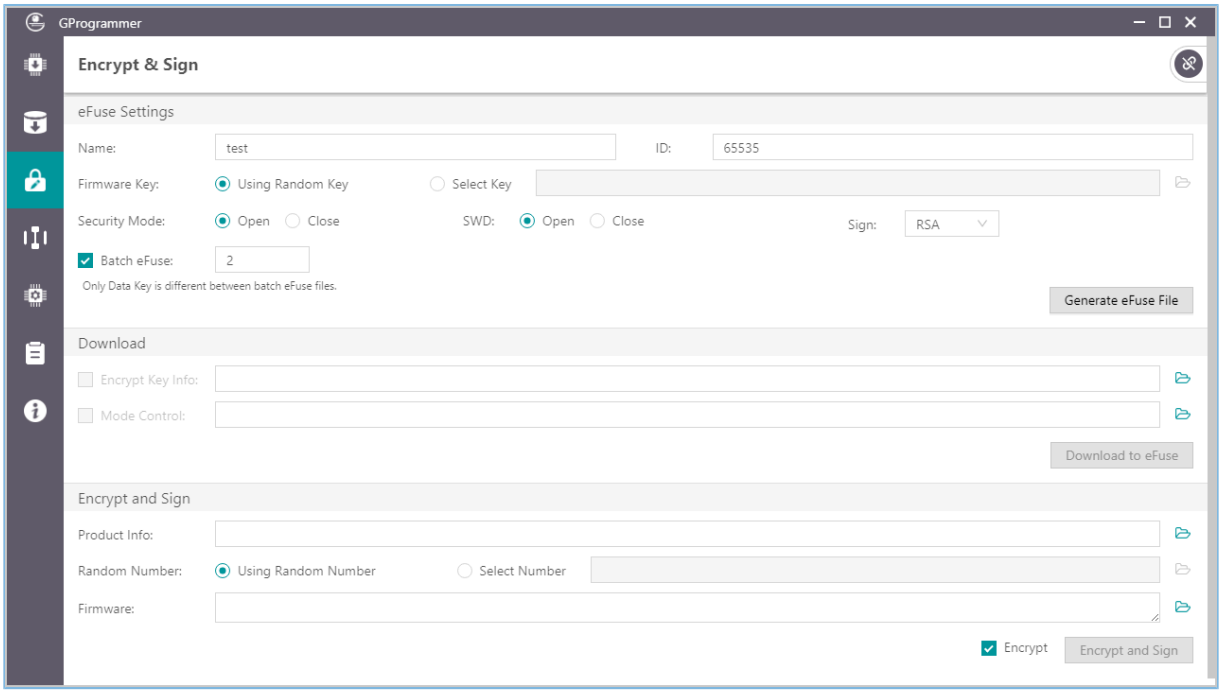

Figure 3-23 GProgrammer **Encrypt & Sign** interface

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.

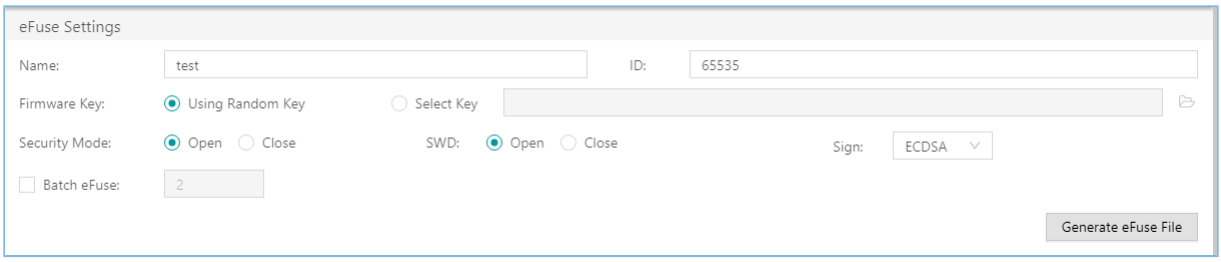

Figure 3-24 Setting eFuse parameters

Note:

- **Firmware Key** can be random keys generated by GProgrammer. Users can also add key files on demand.
- Selecting **Open** for **Security Mode** will enable the security mode, which cannot be disabled after being enabled.
- Selecting **Close** for **SWD** will disable SWD. In this case, you can upgrade the firmware through DFU.
- Select between RSA and ECDSA signature algorithms: For GR551x/GR5526 SoCs, RSA is used by default (with no configuration options on the page). For GR5525 SoCs, you can select **RSA** or **ECDSA**. For GR533x/GR5405 SoCs, you can use **ECDSA** only.

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:

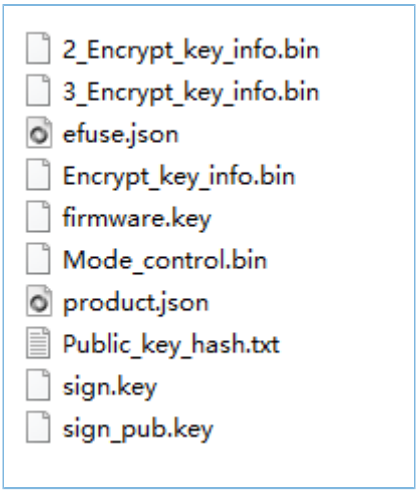

Figure 3-25 Generated files

- *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

Note:

Please keep the above files properly. These files are required for subsequent **Download to eFuse** and **Encrypt and Sign** operations.

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.

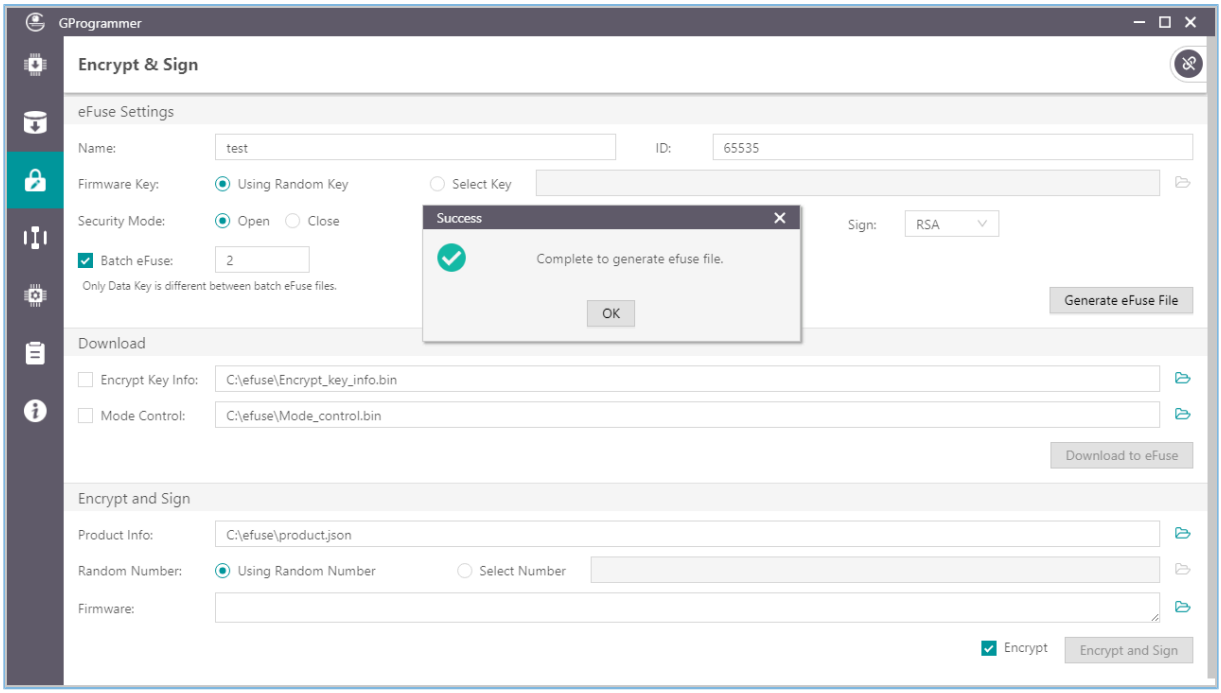

Figure 3-26 Paths for automatically loaded files

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.

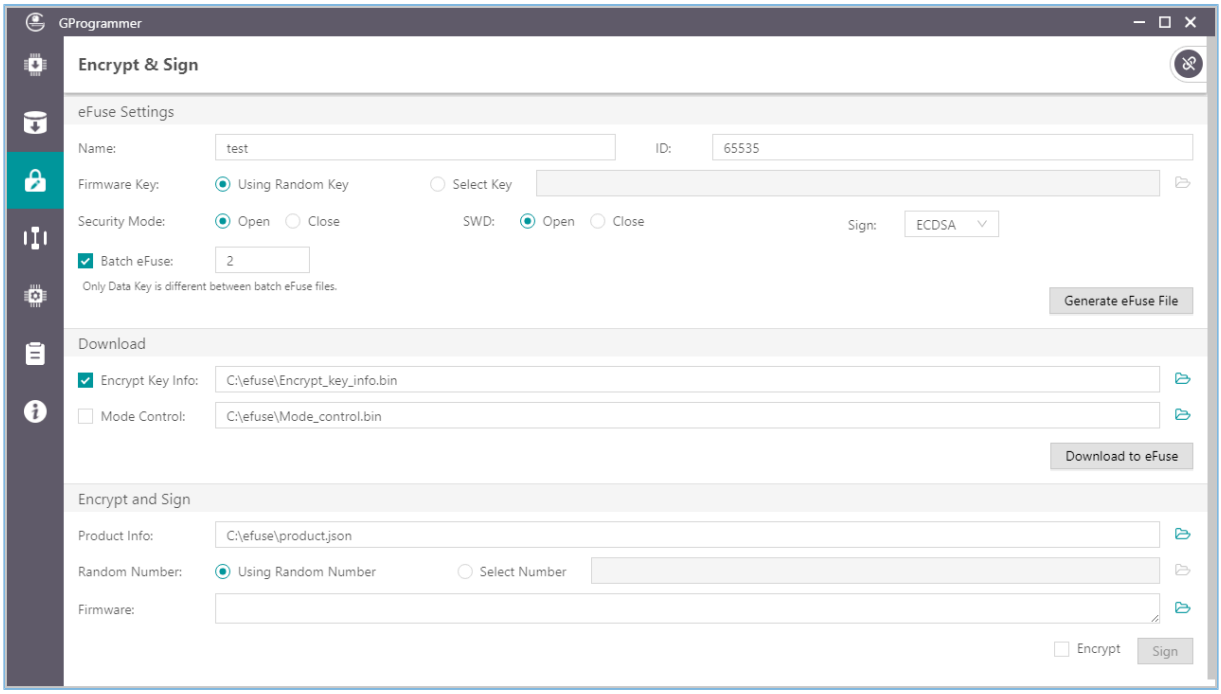

Figure 3-27 Downloading files to eFuse

Note:

- Download to eFuse can be performed only once.
- Download to eFuse is not supported by GR533x/GR5405 SoCs.

3.7.3 Encrypt & Sign

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**).

- Random Number: If **Using Random Number** is selected, random numbers automatically generated by software will be used for signing. You can also click **Select Number** to import a .bin file which contains customized numbers for signing.
- Firmware: Import unencrypted firmware files in this field. GProgrammer supports importing unencrypted firmware in both .hex and .bin formats and outputting (encrypted and) signed firmware in .bin format. When multiple firmware files are imported, add ";" between file paths, as shown below.

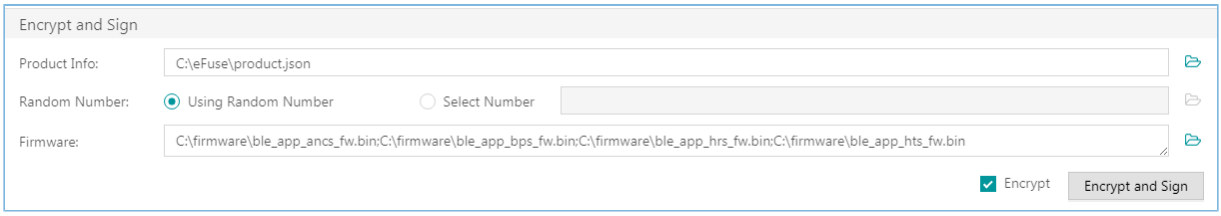

Figure 3-28 Adding more than one firmware file

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

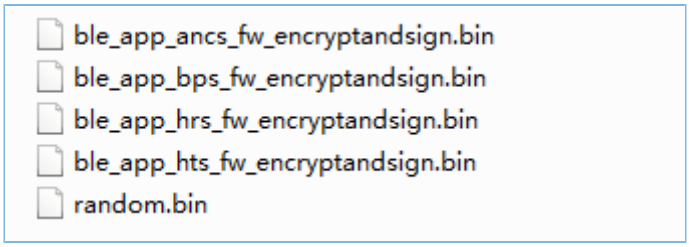

Figure 3-29 GProgrammer-generated files after encryption and signing

  - Files after being signed are listed below:

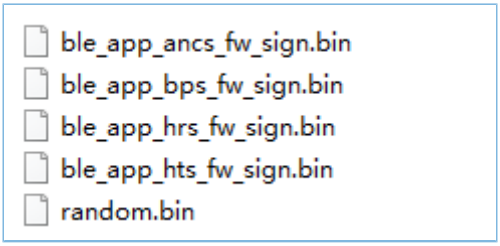

Figure 3-30 GProgrammer-generated files after signing

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.

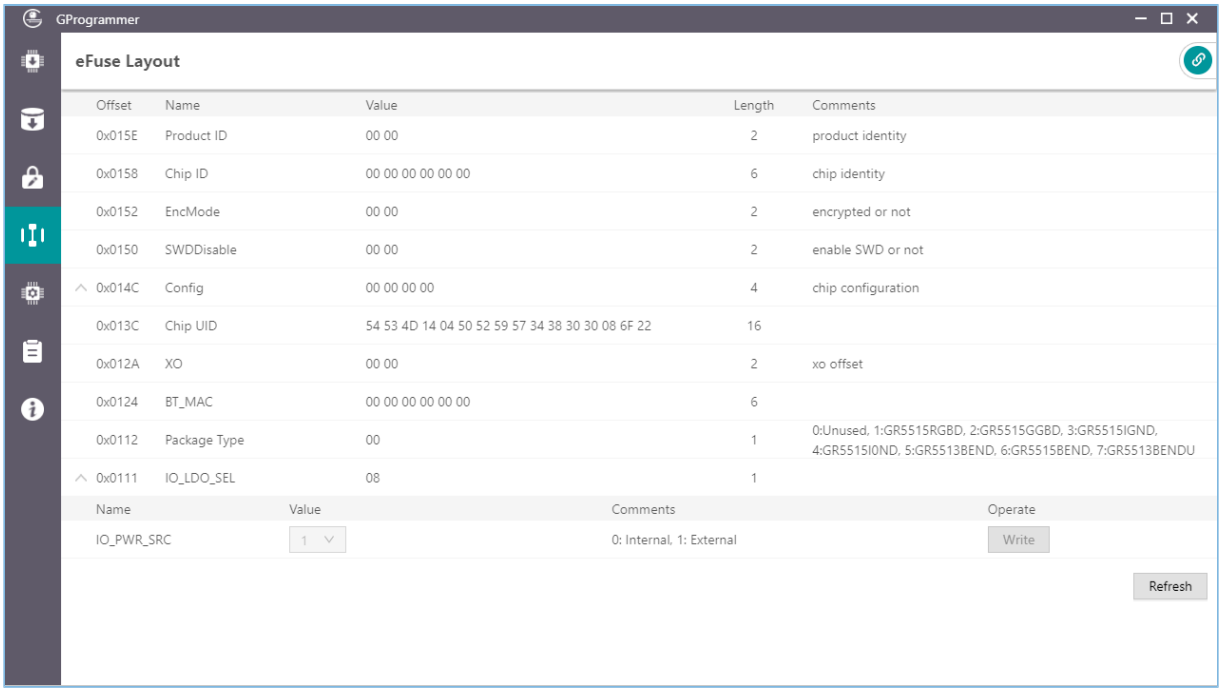

Figure 3-31 **eFuse Layout** interface

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.

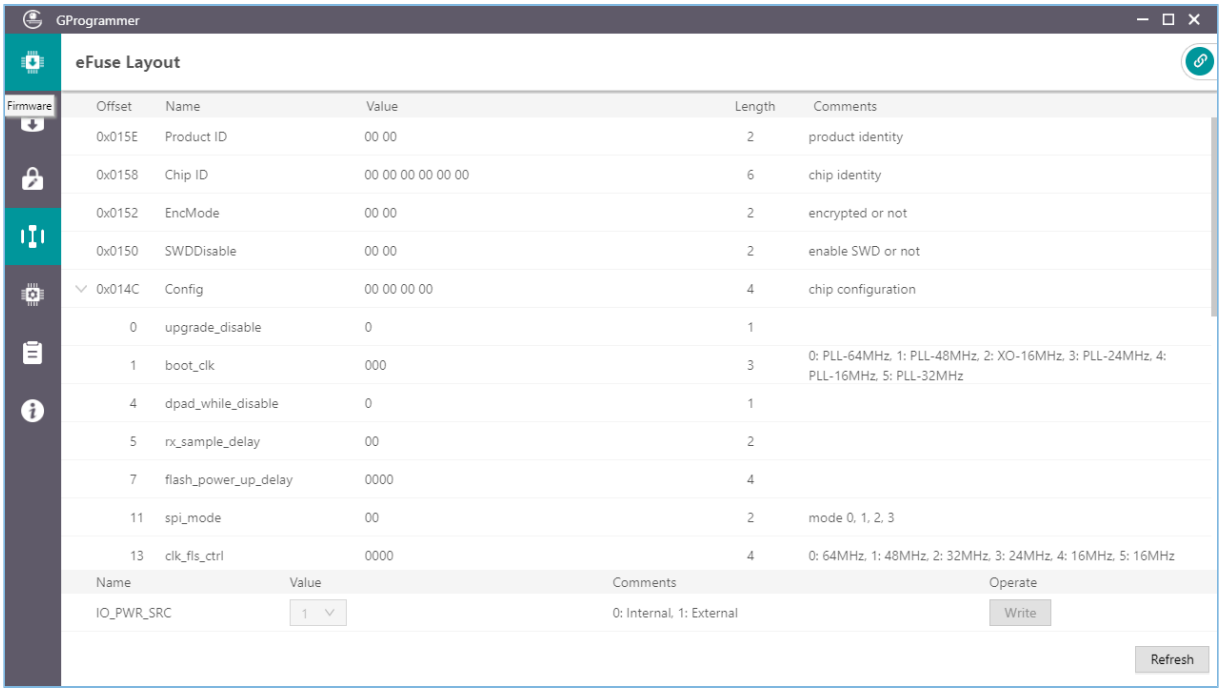

Figure 3-32 Expanded **Offset**

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 OTP Layout

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

Note:

This interface is applicable to GR533x/GR5405 only.

The field and bit field information shown in the list are from the *xxx_otp_config.json* file in the config folder, not all the information stored in the OTP.

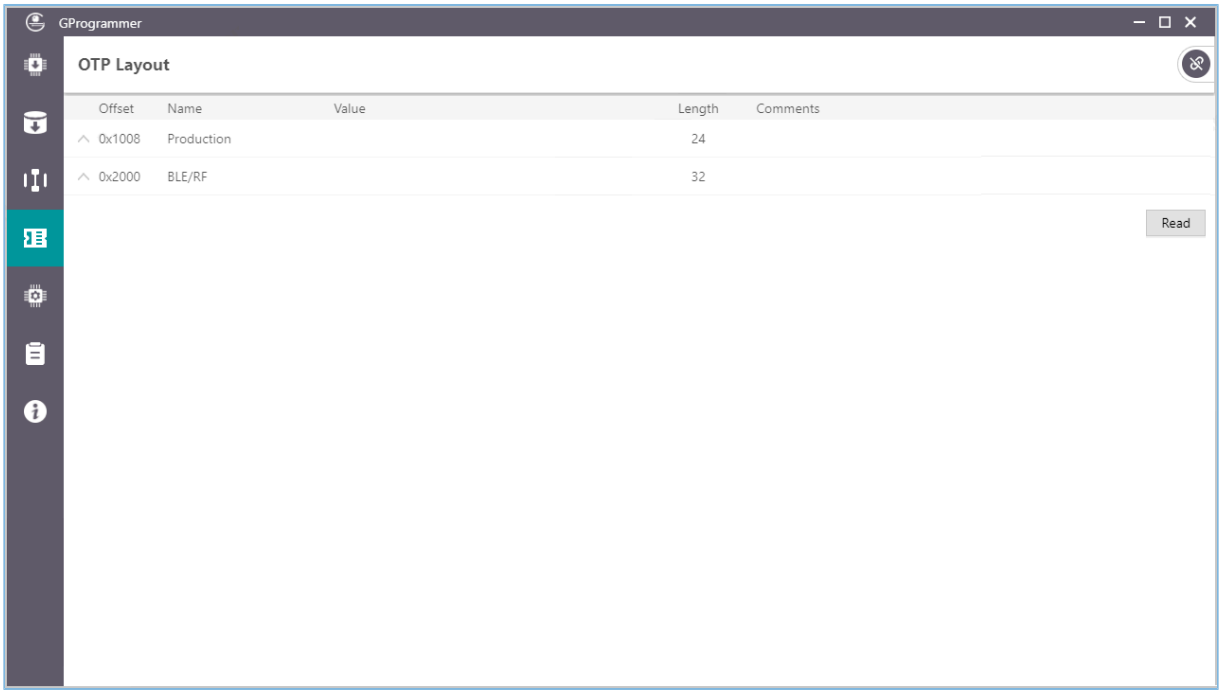

Figure 3-33 **OTP Layout** interface

3.10 Chip Configuration

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

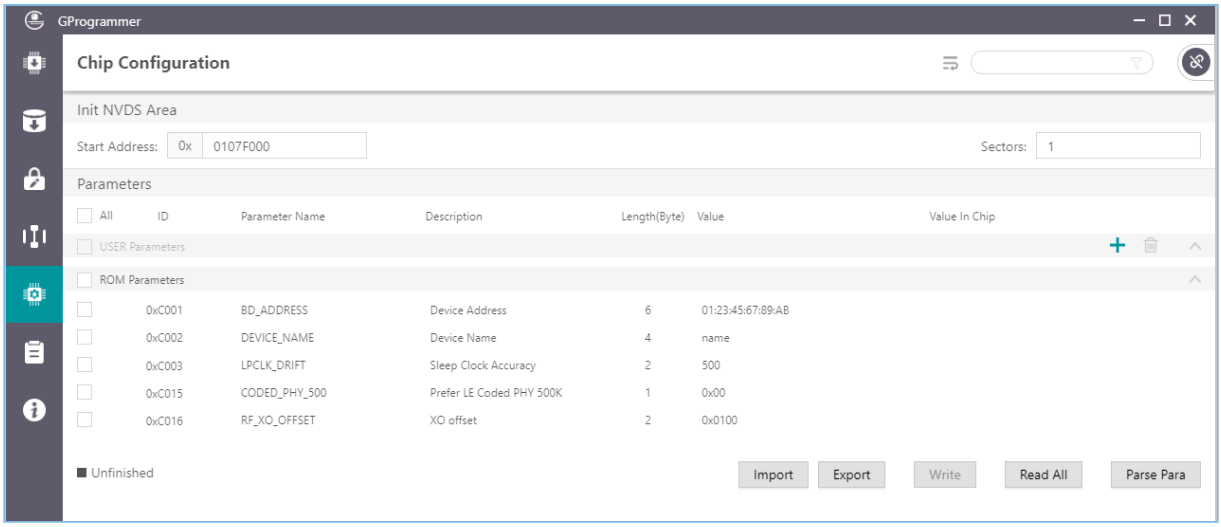

Figure 3-34 GProgrammer **Chip Configuration** interface

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 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 3-6.
- 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 3-6 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.
0xC003	LPCLK_DRIFT	This parameter sets the Sleep Clock Accuracy (SCA); range: 10 ppm to 500 ppm
0xC015	CODED_PHY_500	This parameter sets the default Coded PHY value; Value 0: 125 kbps; Value 1: 500 kbps
0xC016	RF_XO_OFFSET	This parameter sets the clock calibration byte; range: 0x000 to 0x1FF

3.10.1 Init NVDS Area

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

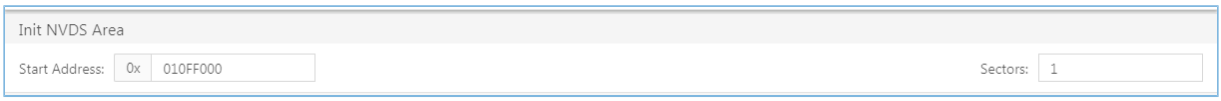

Figure 3-35 Setting the start address and sector quantity in the NVDS area

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

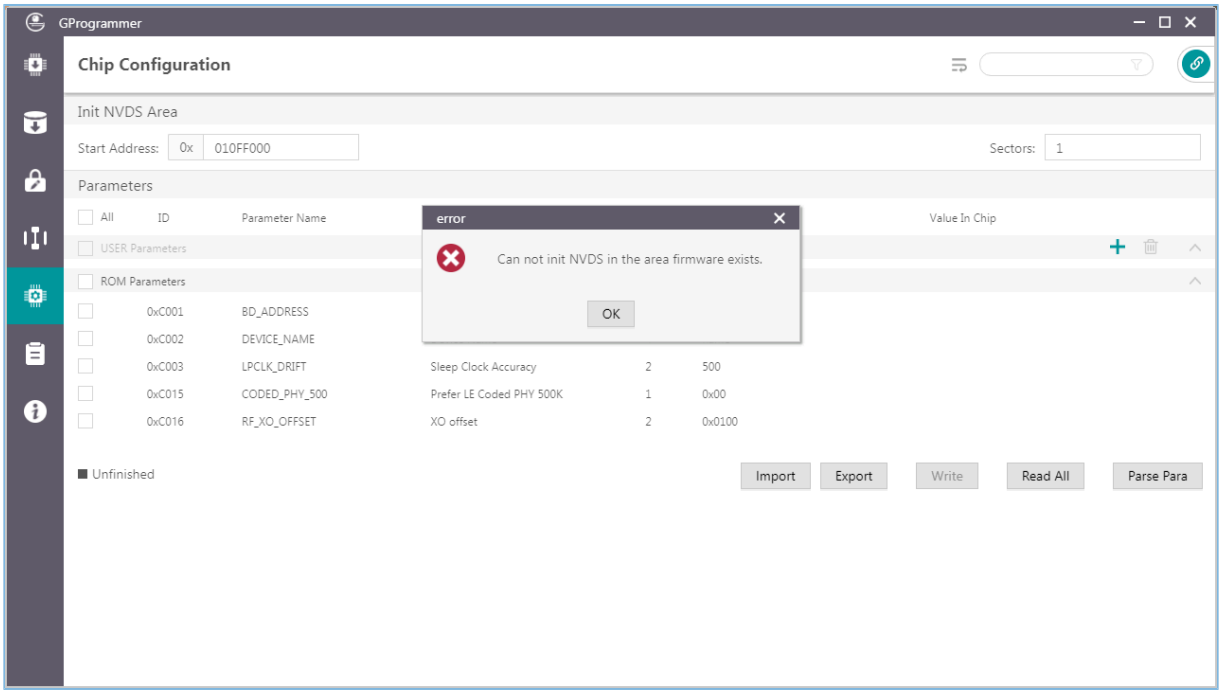

Figure 3-36 NVDS initialization failure

3.10.2 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 3-37) or ones that have been listed in the NVDS area but with a different parameter length (example: 0x4001 in Figure 3-37).
- **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 3-37)
- **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 3-37)

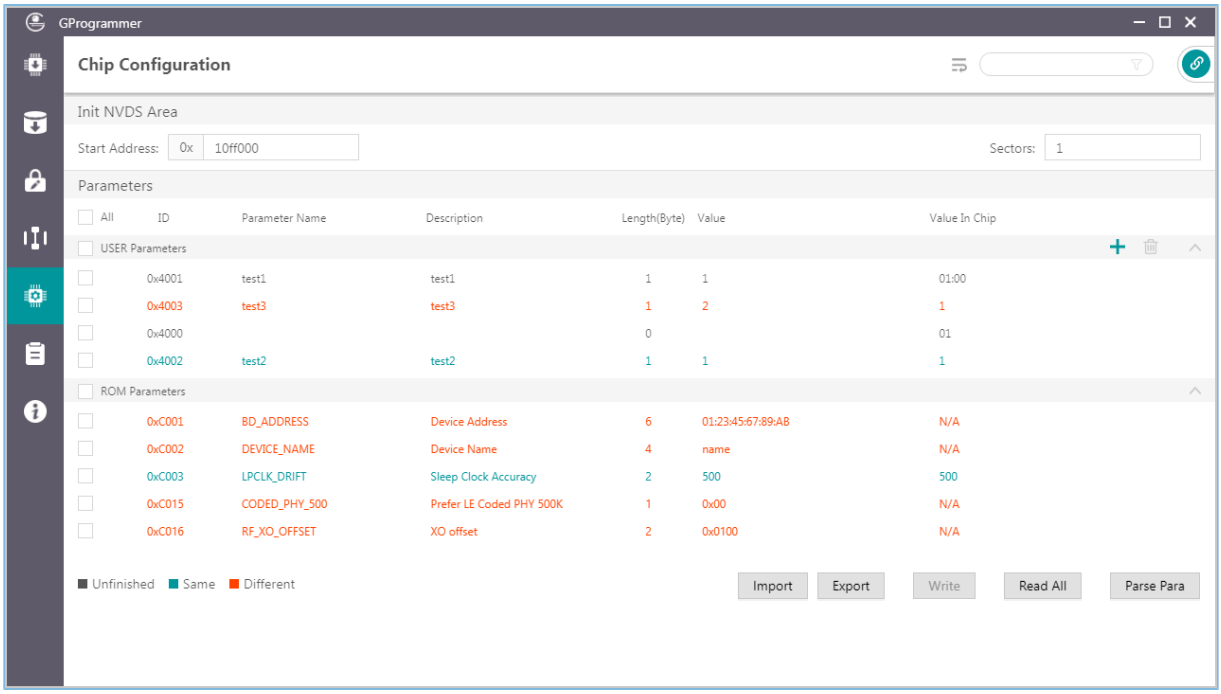

Figure 3-37 **Read All** interface

3.10.3 Write

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

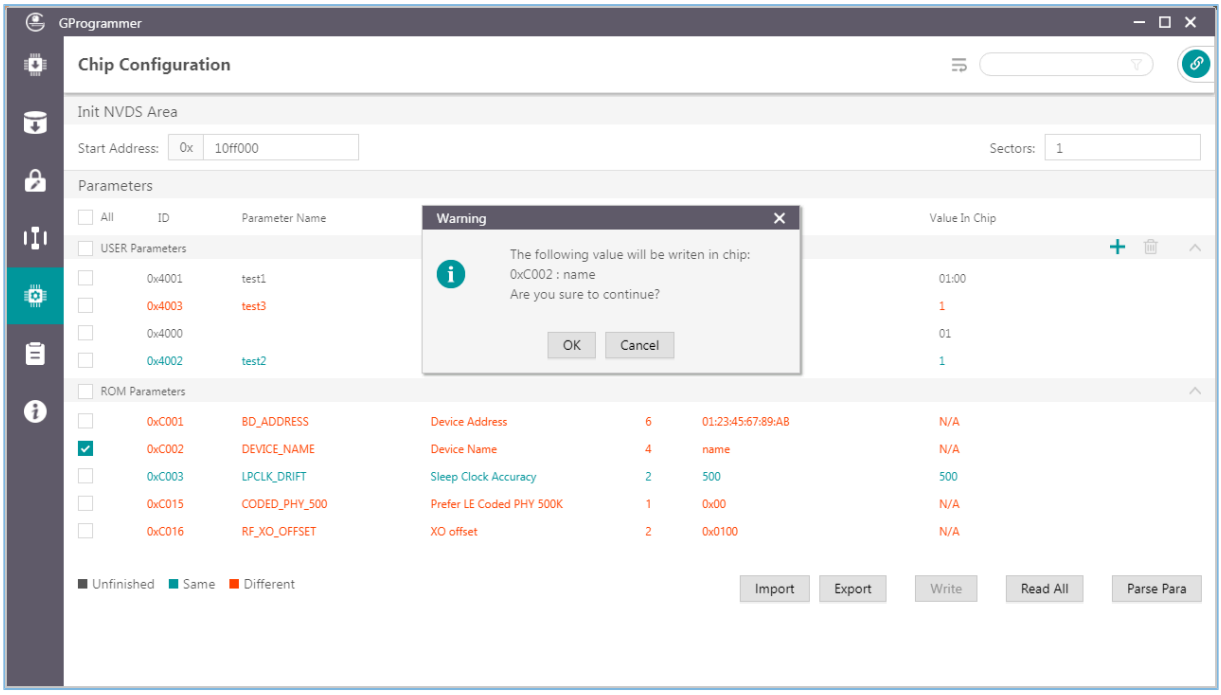

Figure 3-38 **Write** parameters to NVDS

Note:

- 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.

3.10.4 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**).

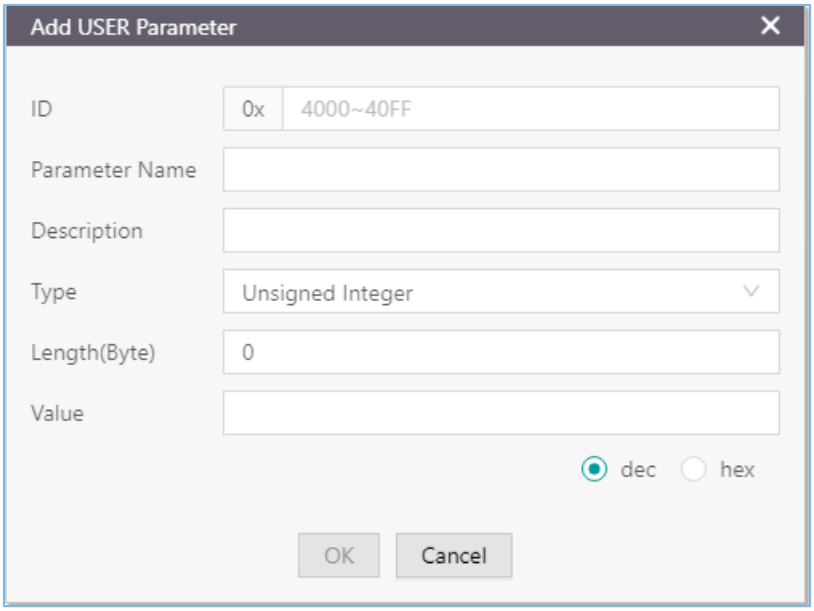

Figure 3-39 Adding a user parameter to NVDS

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 3-40.
- 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.

The default range of the ID is 0x4000–0x40FF. You can modify the valid range to 0x4000–0x7FFF in *nvds_common_config.json*. Note that if the range is too large, the parameter read duration will be affected.

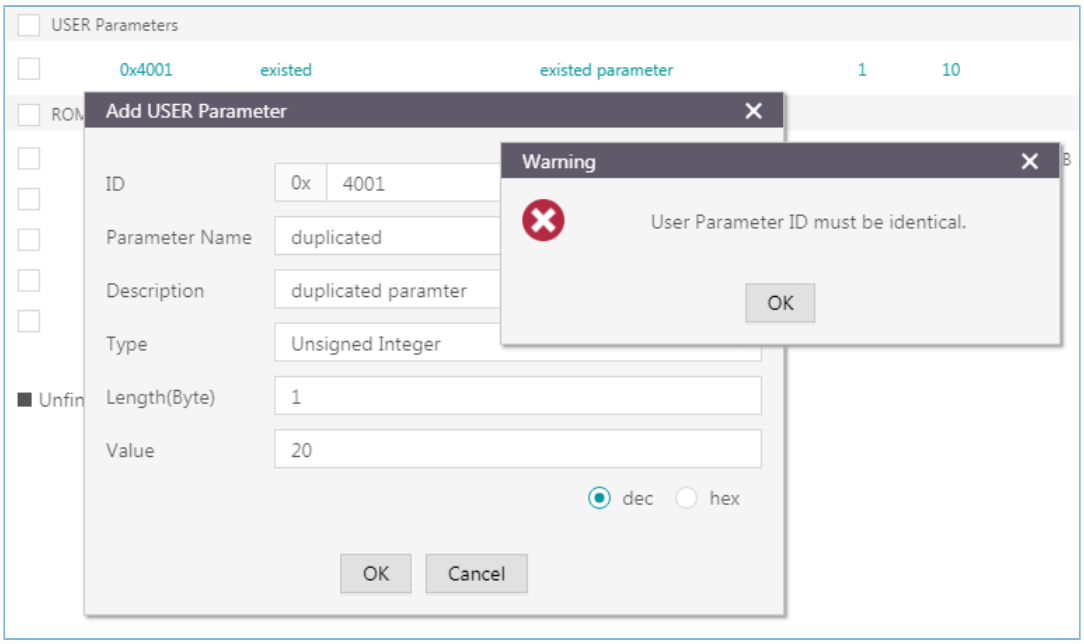

Figure 3-40 Failure to add a user parameter due to an identical parameter ID

3.10.5 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.

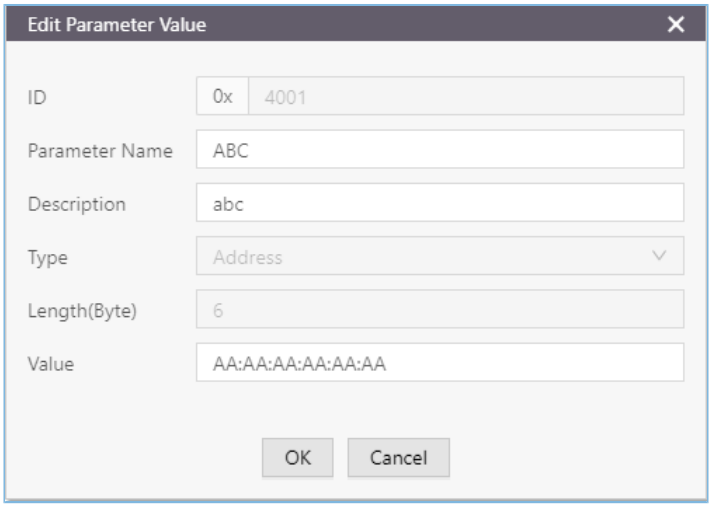

Figure 3-41 **Edit Parameter Value** window

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.

3.10.6 Remove a User Parameter

Users can remove user parameters only.

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

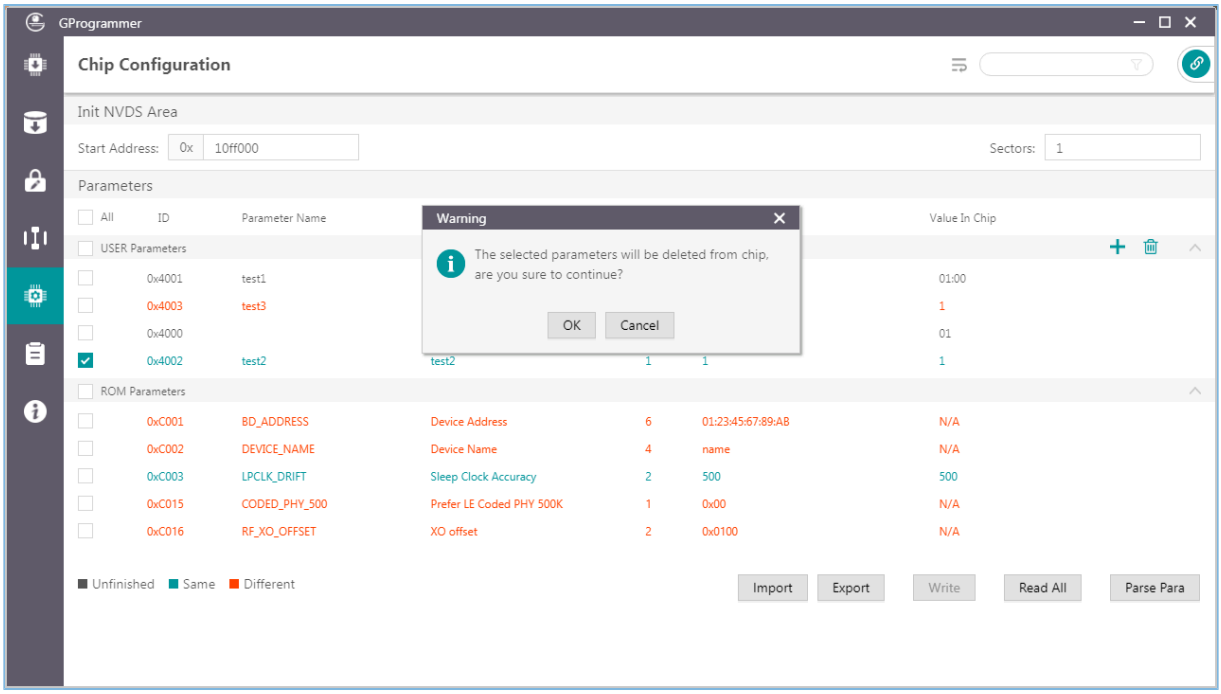

Figure 3-42 Removing a parameter

Note:

- You can select more than one parameter and click to implement a batch removal.
- When a ROM parameter is selected, is in gray.

3.10.7 Import and Export

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

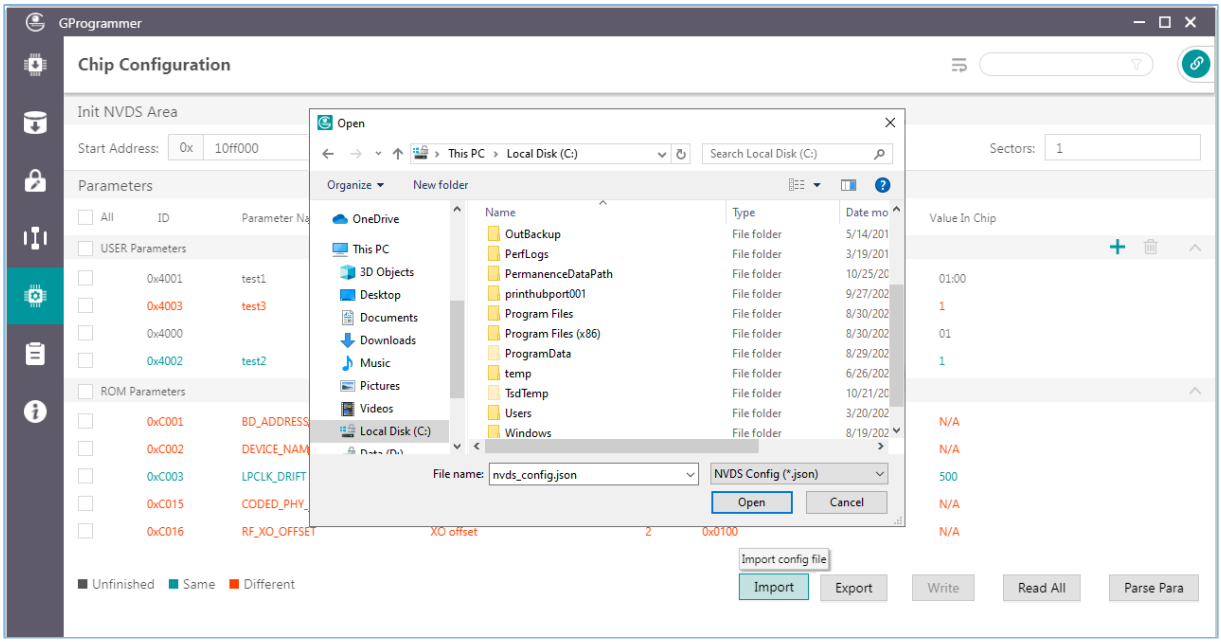

Figure 3-43 Importing local JSON configuration files to GProgrammer

Note:

- 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.

3.10.8 Parse Data in the NVDS Area

GProgrammer provides users with data parsing functionality **Parse Para**. It supports parsing data read from the NVDS area or loaded from a local data file.

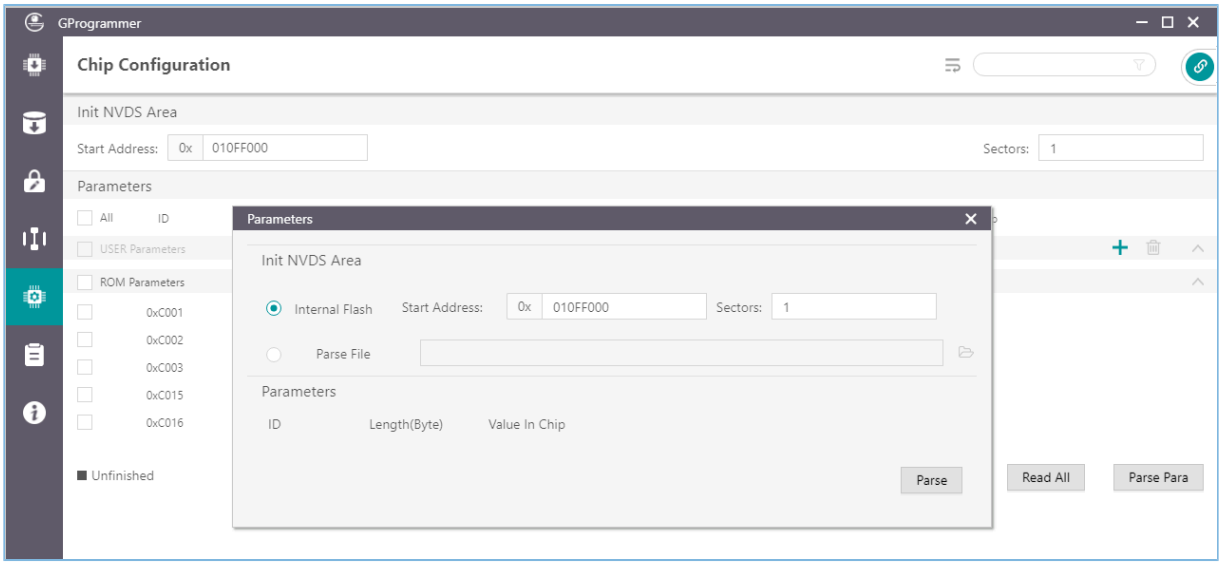

Figure 3-44 Configuring to-be-parsed data

- To parse data read from the NVDS area, choose **Internal Flash**, and then set the start address of the NVDS area (4 KB aligned) as well as the number of occupied sector(s).

  The area shall be in the range configured for NVDS parameters, with start address and the number of occupied sector(s) as detailed in "Section 3.10.1 Init NVDS Area". Otherwise, data parsing fails.

- To parse data loaded from a local data file, choose **Parse File**, and then select an exported NVDS data file locally.

Note:

- This functionality is applicable to non-encrypted data only.
- When parsing starts by clicking **Parse**, the sequence of data parsing and result display is identical with that of Flash memory data. **Value in Chip** is in little-endian mode.

3.11 Device Log

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

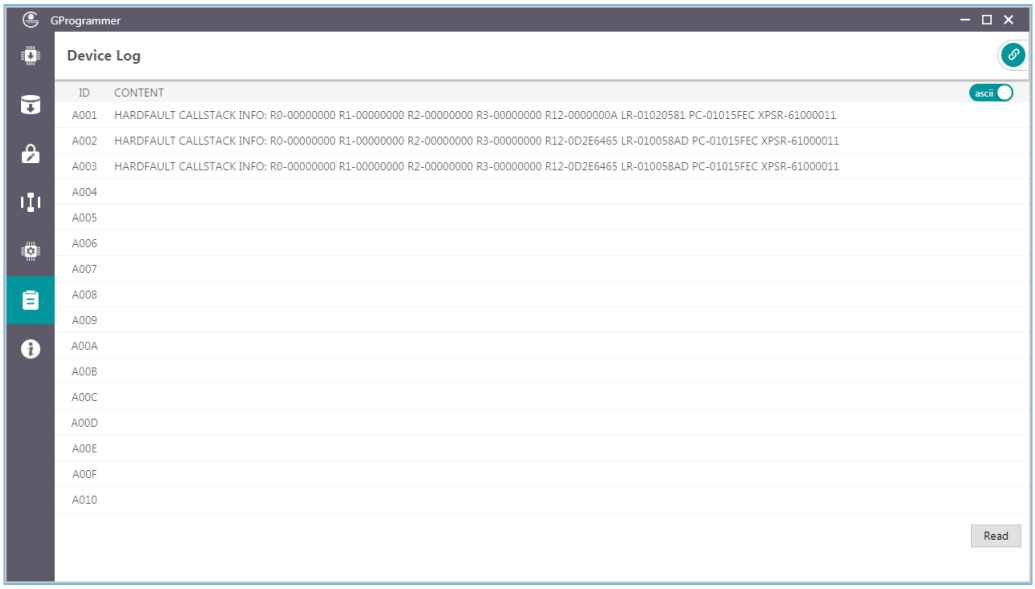

Figure 3-45 **Device Log** interface

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 3-46.
- stream: The device logs are displayed by byte stream as shown in Figure 3-47.

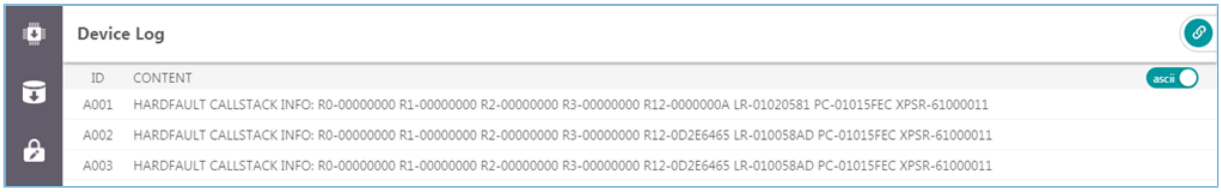

Figure 3-46 Device logs in ASCII characters

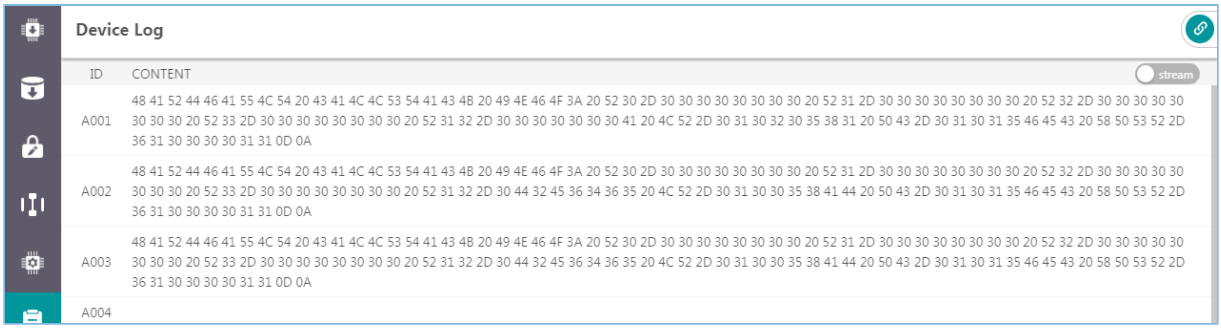

Figure 3-47 Device logs in byte streams

3.12 Command-line Programs

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

Note:

GR5xxx represents the name of SoC series.

- `GR5xxx_console` supports firmware programming and Flash/NVDS/OTP/eFuse operations in a command-line interface.
- `GR5xxx_encrypt_signature` supports firmware encryption and (or) signing in a command-line interface.

3.12.1 GR5xxx_console

- On Windows

  Follow the steps below to run `GR5xxx_console` on Windows:

  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_console.exe help` to check the command list.
  4. Type `GR5xxx_console.exe help <command>` (such as `GR5xxx_console.exe help program`) to view the descriptions and instructions on the commands.

- On Linux

  Follow the steps below to run `GR5xxx_console` on Linux:

  1. Start the device.
  2. Navigate to the GProgrammer installation directory by using `cd` command.
  3. Type `./GR5xxx_console help` to check the command list.
  4. Type `./GR5xxx_console help <command>` (such as `./GR5xxx_console help program`) to view the descriptions and instructions on the commands.

Tip:

To use commands from an older version (before V2.0.0), you can enter `GR5xxx_console.exe help jlink` (Windows) or `./GR5xxx_console help jlink` (Linux) to view the command list. It is recommended to use the latest version for a better user experience.

3.12.1.1 Common Parameters

Common parameters apply to all commands requiring device connection, excluding local commands not requiring device communication (such as help, merge, generate, and device commands). Common parameters mainly include connection parameters and SoC type parameters.

Table 3-7 Common parameters

Parameter	Description
--uart	Specify the UART communication mode. The parameter name is followed by the UART name, such as COM32 or ttyUSB0.
--baudrate (optional)	Specify the baud rate for UART communication, such as 921600 or 2000000.
--jlink	Specify the J-Link communication mode. The parameter name is followed by the J-Link SN. If there is only one device, SN can be 0.
--chip	Specify the SoC series, such as GR5515, GR5525, GR5332, or GR5405.

3.12.1.2 External Flash Configuration Parameters

To execute external Flash operation commands (writeFlash, updateFlash, readFlash, eraseFlash), you need to configure external Flash parameters.

GPIO_TYPE parameter description:

- GR551x:
  - 00 - Normal
    01 - AON
    02 - MSIO
- GR5525/GR5526:
  - 00 - GPIOA (GPIO0 - GPIO15)
  - 01 - GPIOB (GPIO16 - GPIO31)
  - 02 - GPIOC (GPIO32 - GPIO33)
  - 03 - AON (AON_GPIO_0 - AON_GPIO_7)
  - 04 - MSIOA (MSIO_A0 - MSIO_A7)

Table 3-8 External Flash parameters

Parameter	Description
--spi_type	SPI type: QSPI, SPI
--gpio_cs_type	GPIO CS type, refer to GPIO_TYPE parameter description
--gpio_cs_pin	GPIO CS pin
--gpio_cs_mux	GPIO CS MUX
--gpio_clk_type	GPIO clock type, refer to GPIO_TYPE parameter description
--gpio_clk_pin	GPIO clock pin
--gpio_clk_mux	GPIO clock MUX
--mosi_io0_type	GPIO IO0 (or MOSI) type, refer to GPIO_TYPE parameter description
--mosi_io0_pin	GPIO IO0 (or MOSI) pin
--mosi_io0_mux	GPIO IO0 (or MOSI) MUX
--miso_io1_type	GPIO IO1 (or MISO) type, refer to GPIO_TYPE parameter description
--miso_io1_pin	GPIO IO1 (or MISO) pin
--miso_io1_mux	GPIO IO1 (or MISO) MUX
--gpio_io2_type (optional)	GPIO IO2 type (refer to GPIO_TYPE parameter description), used to configure QSPI
--gpio_io2_pin (optional)	GPIO IO2 pin, used to configure QSPI
--gpio_io2_mux (optional)	GPIO IO2 MUX, used to configure QSPI
--gpio_io3_type (optional)	GPIO IO3 type (refer to GPIO_TYPE parameter description), used to configure QSPI
--gpio_io3_pin (optional)	GPIO IO3 pin, used to configure QSPI
--gpio_io3_mux (optional)	GPIO IO3 MUX, used to configure QSPI
--qspi_Id (optional)	QSPI module ID, used to configure QSPI

3.12.1.3 Help Command

- Command name: `help`
- Command description: List the commands and demonstrate how to use the commands.
- Parameter description:

Table 3-9 Command parameter description

Parameter	Description
Command name	1. If the help command is not followed by any parameter, list all commands. 2. If the help command is followed by a command name, view the description, parameters, and instruction on the command.

- Example command:

  1. List all commands.

```
help
```

2. View the description and instruction on the `program` command.

```
help program
```

3.12.1.4 Generating Firmware for Programming

- Command name: `generate`
- Command description: Generate programmable .bin firmware from the compiled .bin/.hex file during the development process.
- Parameter description:

Table 3-10 Command parameter description

Parameter	Description
--chip	Specify the SoC series, such as GR5515, GR5525, GR5332, or GR5405.
--input	Compiled .bin/.hex file during the development process
--output	Generated binary firmware for programming

- Example command:

  Use the compiled *app_bootloader.bin* to generate *app_bootloader_fw.bin* for programming.

```
generate --chip gr5525 --input app_bootloader.bin --output app_bootloader_fw.bin
```

3.12.1.5 Merging Firmware

- Command name: `merge`
- Command description: Merge multiple sub-firmware files into one firmware file for one-time programming.

Note:

Currently, the firmware merged using the `merge` command can only be programmed using the `program` command. It cannot be programmed through GProgrammer UI.

- Parameter description:

Table 3-11 Command parameter description

Parameter	Description
--files	Specify the non-startup firmware to be merged, and separate each firmware file with a space.
--boot	Specify the “startup” firmware. You need to specify a firmware program as the “startup” firmware before merging firmware files.

Parameter	Description
--out	Path to store the merged binary firmware

- Example command:

  Merge an application firmware file and a startup firmware file.

```
merge --files ble_app_template_dfu_fw.bin --boot ble_dfu_boot_fw.bin --out out.bin
```

3.12.1.6 Locking/Unlocking Flash

- Command name: `lockFlash`
- Command description: Lock and unlock the internal Flash.
- Parameter description:

Table 3-12 Command parameter description

Parameter	Description
Common parameter	Refer to “Section 3.12.1.1 Common Parameters”. **Note:** This command is supported in UART communication mode only.
--mode	◦ 0: Unlock the Flash. ◦ 1: Unlock and erase the Flash. ◦ 2: Lock the Flash.

- Example command:

  Unlock GR5405 Flash.

```
lockFlash --chip gr5405 --mode 0 --uart COM30
```

3.12.1.7 Resetting Device

- Command name: `reset`
- Command description: Reset a device.
- Parameter description:

Table 3-13 Command parameter description

Parameter	Description
Common parameter	Refer to “Section 3.12.1.1 Common Parameters”.

- Example command:

  Reset a device.

```
reset --chip gr5332 --jlink 0
```

3.12.1.8 Programming Firmware

- Command name: `program`
- Command description: Program the firmware to the internal Flash in the target device.
- Parameter description:

Table 3-14 Command parameter description

Parameter	Description
Common parameter	Refer to "Section 3.12.1.1 Common Parameters".
--file	Specify the file path of the firmware to be programmed, which can be a single firmware file or a merged firmware file using the `merge` command.
--run	Configure whether to run the program immediately after the firmware is programmed. ◦ true: Immediately run the program. ◦ false: Do not run the program.
--erase	Configure whether to erase Flash before firmware programming. ◦ 0: Do not erase Flash. ◦ 1: Erase the entire Flash. ◦ 2: Erase BootInfo.

- Example command:

1. Program *app_bootloader_fw.bin*; erase the entire Flash before programming, and run the program immediately after programming completes.

```
program --jlink 0 --chip GR5405 --file app_bootloader_fw.bin --erase 1 --run 1
```

2. Generate a firmware file and program it.

```
generate --chip GR5405 --input app_bootloader.bin --output app_bootloader_fw.bin
program --uart COM10 --chip GR5405 --file app_bootloader_fw.bin --erase 1 --run 1
```

3. Merge multiple firmware files and program the merged firmware.

```
merge --files ble_app_template_dfu_fw.bin --boot ble_dfu_boot_fw.bin --out out.bin
program --uart COM10 --chip GR5405 --file out.bin --erase 1 --run 1
```

3.12.1.9 Writing Data to Flash

- Command name: `writeFlash`
- Command description: Write data to Flash.

- Parameter description:

Table 3-15 Command parameter description

Parameter	Description
Common parameter	Refer to "Section 3.12.1.1 Common Parameters".
External Flash configuration parameters (optional)	When an external Flash is selected for **flashType**, you need to input the configuration parameters of the external Flash. For details, refer to "Section 3.12.1.2 External Flash Configuration Parameters".
--flashType (optional)	Flash type ◦ 0: internal Flash (default) ◦ 1: external Flash
--flashSize (recommended)	Flash size (unit: Kbyte)
--file	Data file; the parameter name is followed by the file path.
--hex	Hex data; the parameter name is followed by the hex character string.
--address	Target address to write data (in hexadecimal format)
--force	Configure whether to forcibly write data when the target address conflicts with the NVDS, BOOT_INFO or IMAGE address. ◦ true: Forcibly write data. ◦ false: Do not write data.

- Example command:

1. Write the data file to a specified address in the internal Flash.

```
writeFlash --chip GR5525 --file bootloader.bin --address 0x00210000 --force false --uart
 COM32
```

2. Write the hex data to a specified address in the internal Flash.

```
writeFlash --chip GR5405 --hex 0x11223344 --address 0x00210000 --force true --jlink 0
```

Note:

The data outside the target address may be erased using the `writeFlash` command. To avoid this, use the `updateFlash` command instead.

3.12.1.10 Updating Flash

- Command name: `updateFlash`
- Command description: Update the data in Flash.
- Parameter description:

Table 3-16 Command parameter description

Parameter	Description
Common parameter	Refer to "Section 3.12.1.1 Common Parameters".
External Flash configuration parameters (optional)	When an external Flash is selected for **flashType**, you need to input the configuration parameters of the external Flash. For details, refer to "Section 3.12.1.2 External Flash Configuration Parameters".
--flashType (optional)	Flash type ◦ 0: internal Flash (default) ◦ 1: external Flash
--flashSize (recommended)	Flash size (unit: Kbyte)
--file	Data file; the parameter name is followed by the file path.
--hex	Hex data; the parameter name is followed by the hex character string.
--address	Target address to update data (in hexadecimal format)

- Example command:

1. Update the data file to a specified address in the internal Flash.

```
updateFlash --chip GR5405 --file save.bin --address 0x00210000 --jlink 0
```

2. Update the hex data to a specified address in the internal Flash.

```
updateFlash --chip GR5405 --hex 0x11223344 --address 0x00210000 --force true --jlink 0
```

3.12.1.11 Reading Data from Flash

- Command name: `readFlash`
- Command description: Read data from a specified address in Flash.
- Parameter description:

Table 3-17 Command parameter description

Parameter	Description
Common parameter	Refer to "Section 3.12.1.1 Common Parameters".
External Flash configuration parameters (optional)	When an external Flash is selected for **flashType**, you need to input the configuration parameters of the external Flash. For details, refer to "Section 3.12.1.2 External Flash Configuration Parameters".
--flashType (optional)	Flash type ◦ 0: internal Flash (default) ◦ 1: external Flash

Parameter	Description
--flashSize (recommended)	Flash size (unit: Kbyte)
--save	Save the read data to a file. If this parameter is not entered, the first 1K data is displayed in hexadecimal format on the terminal device.
--address	Target address to read data
--size	Size of the data to be read (unit: byte)

- Example command:

  Read data from a specified address in the internal Flash, and save the data to a file.

```
readFlash --chip GR5515 --address 0x01070000 --size 10240 --save save.bin --uart COM32
```

3.12.1.12 Erasing Flash

- Command name: `eraseFlash`
- Command description: Erase Flash.
- Parameter description:

Table 3-18 Command parameter description

Parameter	Description
Common parameter	Refer to “Section 3.12.1.1 Common Parameters”.
External Flash configuration parameters (optional)	When an external Flash is selected for **flashType**, you need to input the configuration parameters of the external Flash. For details, refer to “Section 3.12.1.2 External Flash Configuration Parameters”.
--flashType (optional)	Flash type ◦ 0: internal Flash (default) ◦ 1: external Flash
--flashSize (recommended)	Flash size (unit: Kbyte)
--start	Represent the start address of the Flash area to be erased (in hexadecimal). The erasing operation starts from the sector where the start address locates.
--end	Represent the end address of the Flash area to be erased (in hexadecimal). The erasing operation ends in the sector where the end address locates.
--force	Configure whether to forcibly erase the Flash when the target address conflicts with the NVDS, BOOT_INFO or IMAGE address during partial erase. ◦ true: Forcibly erase Flash. ◦ false: Do not erase Flash.

- Example command:

1. Erase the entire internal Flash.

```
eraseFlash --all true --chip GR5332 --uart COM77
```

2. Erase the specified area in the internal Flash.

```
eraseFlash --start 0x00210000 --end 0x00212000 --chip GR5405 --jlink 0
```

3.12.1.13 Writing Data to NVDS

- Command name: `writeNvds`
- Command description: Write data to NVDS.
- Parameter description:

Table 3-19 Command parameter description

Parameter	Description
Common parameter	Refer to "Section 3.12.1.1 Common Parameters".
--address	NVDS start address (in hexadecimal format)
--sectors	Count of NVDS sectors, equal to the NVDS capacity divided by 4K
--tag	Target tag to write data
--hex	Input the data to be written as hex character strings.

- Example command:

  Write data to NVDS.

```
writeNvds --address 0x00270000  --sectors 1 --tag C001 --hex 0123456789BB --jlink 0 --chip
 GR5332
```

3.12.1.14 Reading Data from NVDS

- Command name: `readNvds`
- Command description: Read data from a specified tag in NVDS.
- Parameter description:

Table 3-20 Command parameter description

Parameter	Description
Common parameter	Refer to "Section 3.12.1.1 Common Parameters".
--address	NVDS start address (in hexadecimal format)
--sectors	Count of NVDS sectors, equal to the NVDS capacity divided by 4K
--tag	Target tag to read data

Parameter	Description
--save (optional)	Save the read data to a file. If this parameter is not entered, the first 1K data is displayed in hexadecimal format on the terminal device.

- Example command:

  Read data from a specified tag in NVDS.

```
readNvds --address 0x00270000  --sectors 1 --tag C001 --chip GR5332 --jlink 0
```

Note:

Ensure that data has been written to NVDS before reading; otherwise, reading will fail.

3.12.1.15 Erasing NVDS

- Command name: `eraseNvds`
- Command description: Erase the data for a specified tag in NVDS.
- Parameter description:

Table 3-21 Command parameter description

Parameter	Description
Common parameter	Refer to "Section 3.12.1.1 Common Parameters".
--address	NVDS start address (in hexadecimal format)
--sectors	Count of NVDS sectors, equal to the NVDS capacity divided by 4K
--tag	Target tag to erase data

- Example command:

  Erase the data for a specified tag in NVDS.

```
eraseNvds --address 0x00270000  --sectors 1 --tag C001 --chip GR5332 --jlink 0
```

3.12.1.16 Writing Data to eFuse

- Command name: `writeEfuse`
- Command description: Write data to User Region (0x00–0x1F) in eFuse.

Note:

This command is only applicable to GR551x/GR5525/GR5526 SoCs.

- Parameter description:

Table 3-22 Command parameter description

Parameter	Description
Common parameter	Refer to "Section 3.12.1.1 Common Parameters".
--offset	Target offset address (0x00–0x1C) to write data; 4-byte aligned
--file	Data file; the parameter name is followed by the file path, and the file size shall be 4-byte aligned.
--hex	Hex data; the parameter name is followed by the hexadecimal character string; 8-character aligned.

- Example command:

  Write hex data to eFuse.

```
writeEfuse --offset 0 --hex 12345678 --chip GR5525 --uart COM77
```

3.12.1.17 Reading Data from eFuse

- Command name: `readEfuse`
- Command description: Read data from eFuse.
- Parameter description:

Table 3-23 Command parameter description

Parameter	Description
Common parameter	Refer to "Section 3.12.1.1 Common Parameters".
--offset	Target offset address to read data; 4-byte aligned
--size	Size of the data to be read (unit: byte); 4-byte aligned
--save (optional)	Save the read data to a file. If this parameter is not entered, the data is output to the terminal device in hexadecimal format.

- Example command:

  Read data from eFuse.

```
readEfuse --offset 0 --size 32 --chip GR5332 --uart COM77
```

3.12.1.18 Writing Data to OTP

- Command name: `writeOtp`
- Command description: Write data to a specified address in OTP.

Note:

This command is for GR533x and GR5405 only.

- Parameter description:

Table 3-24 Command parameter description

Parameter	Description
Common parameter	Refer to "Section 3.12.1.1 Common Parameters".
--address	Target address to write data (in hexadecimal format)
--file	Data file; the parameter name is followed by the file path.
--hex	Hex data; the parameter name is followed by the hex character string.

- Example command:

  Write data to OTP.

```
writeOtp --address 0x2008 --hex 12345678 --uart COM77 --chip GR5332
```

3.12.1.19 Reading Data from OTP

- Command name: `readOtp`
- Command description: Read data from a specified address in OTP.

Note:

This command is for GR533x and GR5405 only.

- Parameter description:

Table 3-25 Command parameter description

Parameter	Description
Common parameter	Refer to "Section 3.12.1.1 Common Parameters".
--address	Target address to read data (in hexadecimal format)
--size	Size of the data to be read (unit: byte)
--save (optional)	Save the read data to a file. If this parameter is not entered, the data is output to the terminal device in hexadecimal format.

- Example command:

  Read data from OTP.

```
readOtp --address 0x2008 --size 0x10 --uart COM77 --chip GR5332
```

3.12.1.20 Erasing OTP

- Command name: `eraseOtp`
- Command description: Erase the data at a specified address in OTP.

Note:

This command is for GR533x and GR5405 only.

- Parameter description:

Table 3-26 Command parameter description

Parameter	Description
Common parameter	Refer to "Section 3.12.1.1 Common Parameters".
--address	Target address to erase data (in hexadecimal format)

- Example command:

  Erase the data in OTP.

```
eraseOtp --address 0x2008 --uart COM77 --chip GR5332
```

3.12.1.21 Displaying Device List

- Command name: `device`
- Command description: Display the device list.
- Parameter description:

Table 3-27 Command parameter description

Parameter	Description
--type	The type of devices to be displayed can be UART, J-Link or all. "all" indicates that both UART and J-Link devices are displayed.
--count (optional)	Count of devices to be displayed (≤ 50); default: 50

- Example command:

  1. Display all UART devices.

```
device --type uart
```

  2. Display all J-Link devices.

```
device --type jlink
```

  3. Display all UART and J-Link devices.

```
device --type all
```

3.12.2 GR5xxx_encrypt_signature

- On Windows

  Follow the steps below to run `GR5xxx_encrypt_signature` on Windows:

  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 3-28. To view all parameters, enter `GR5xxx_encrypt_signature.exe --help`.

- On Linux

  Follow the steps below to run `GR5xxx_encrypt_signature` on Linux:

  1. Start the device.
  2. Navigate to the GProgrammer installation directory by using `cd` command.
  3. Type `./GR5xxx_encrypt_signature --parameter` to complete corresponding operations.

     For most frequently used parameters, see Table 3-28. To view all parameters, enter `./GR5xxx_encrypt_signature --help`.

Table 3-28 Frequently used parameters for GR5xxx_encrypt_signature

Parameter	Description	Remarks
operation	Indicate the operation type. Options: • encryptandsign: Encrypt and sign firmware. • sign: Sign firmware only	
firmware_key	Show 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 File** in "Section 3.7.1 eFuse Settings".
signature_key	Show the directory of *sign.key*, which is used for firmware encryption and signing, or signing only.	
signature_pub_key	Show the directory of *sign_pub.key*, which is used for firmware encryption and signing, or signing only.	
product_json_path	Show the directory of *product.json*, which is used for firmware encryption and signing, or signing only.	
rand_number	Show the directory of *random.bin*, which is used for firmware encryption and signing, or signing only.	
ori_firmware	Show the directory that saves the firmware before encryption and signing, or signing only.	
output	Show the directory that saves the firmware after encryption and signing, or signing only.	

Parameter	Description	Remarks
random_output	Show the directory that saves the random numbers used for firmware encryption and signing, or signing only.	
base_addr	Set the start address in the Flash memories to which firmware files are downloaded. Value: • 0x01000000: for GR551x • 0x00200000: for GR5526/GR533x/GR5405/GR5525	
flash_size	Indicate 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	Indicate the SoC series. Valid value and description: • 0: GR551x • 1: GR5526 • 2: GR533x/GR5405 • 4: GR5525	
help	Display 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" --rand_number="D:/
test/eFuse/random.bin" --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 3-28.

- --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
- --rand_number: Refer to "Section 3.7.3 Encrypt & Sign" for specific descriptions on "Random Number".

Run the command to encrypt and sign the firmware.

3.12.3 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 represents the name of SoC series.

---

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.

3.13 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 Tool**

  This section provides version information and features of GProgrammer.

- **Feedback**

  If you have any questions or suggestions, please log in to Developer Community for feedback.

- **About Goodix**

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

3.14 FAQ

3.14.1 Why Does GProgrammer Open with a Blank Screen After Launch?

- Description

  After installing GProgrammer, double-click the icon to launch GProgrammer. However, it opens with a blank screen.

- Analysis

  The possible causes may include:

  - Cause 1: GProgrammer has been installed on the system disk (such as the C drive), which has protection measures for file read and write operations, causing abnormal display after launch.
  - Cause 2: A file in the installation package of GProgrammer is missing or damaged (such as being encrypted), causing abnormal display after launch.

- Solution

  You can choose from the following solutions to address this issue:

  - Solution 1: Run GProgrammer in administrator mode.
  - Solution 2: Reinstall GProgrammer on a non-system disk.

3.14.2 Why Does GProgrammer Connection Fail?

- Description

  Connection between GProgrammer and the device fails.

- Analysis and solution

  Follow the steps below for troubleshooting:

  1. Check whether correct SoC part number is selected before connection.
  2. Check whether hardware is properly connected. If the firmware program with sleep functionality enabled is running in the SoC, ensure that the reset pin of the J-Link emulator or the RST pin of the USB-to-serial converter is connected to the Chip_En pin of the SoC. If the two pins cannot be connected, you can reset the device manually. Click **Connect** in GProgrammer, and then press the **Reset** button on the SK Board immediately to reset the board. However, the success rate of manual reset will decrease.
  3. If multiple J-Link devices are inserted, leading to connection failure due to frequent pop-up windows, remove the devices that are currently not in use, and then try reconnecting.
  4. If a waiting dialog box is always displayed or command line execution is unresponsive during connection via J-Link, the program may get stuck in the J-Link driver. In this case, reinsert the device, and restart GProgrammer for reconnection.

3.14.3 Why Does Module Loading Fail when GProgrammer Starts on Linux?

- Description

  When GProgrammer starts on Linux, an error is reported: Failed to load module “canberra-gtk-module”.

- Analysis

  This problem is caused by the lack of canberra-gtk-module in the system.

- Solution

  Install canberra-gtk-module. For example, enter the following command to install canberra-gtk-module on Ubuntu.

```
sudo apt-get install libcanberra-gtk-module
```

3.14.4 Why Does the GProgrammer Interface Appear Blank or Device Connection Fail on Linux?

- Description

  After GProgrammer starts on Linux, GProgrammer interface appears blank, or device connection fails.

- Analysis

  The program may not be loaded correctly due to lack of access permissions for some files.

- Solution

  Ensure that all files in the GProgrammer directory are accessible to users, or users can start GProgrammer with root permissions. For example, execute the following command to start GProgrammer on Ubuntu:

```
sudo ./GProgrammer
```