GProgrammer User Manual

Version: 2.3

Release Date: 2021-07-16

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: 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 users
- GR551x developers
- GR551x testers

Release Notes

This document is the ninth release of *GProgrammer User Manual*, corresponding to GProgrammer V1.2.20.

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 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 figures for software version.
2.0	2021-01-05	Updated software UI figures 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.

Contents

1 Introduction

GProgrammer supports programming of flash memories on GR551x 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.

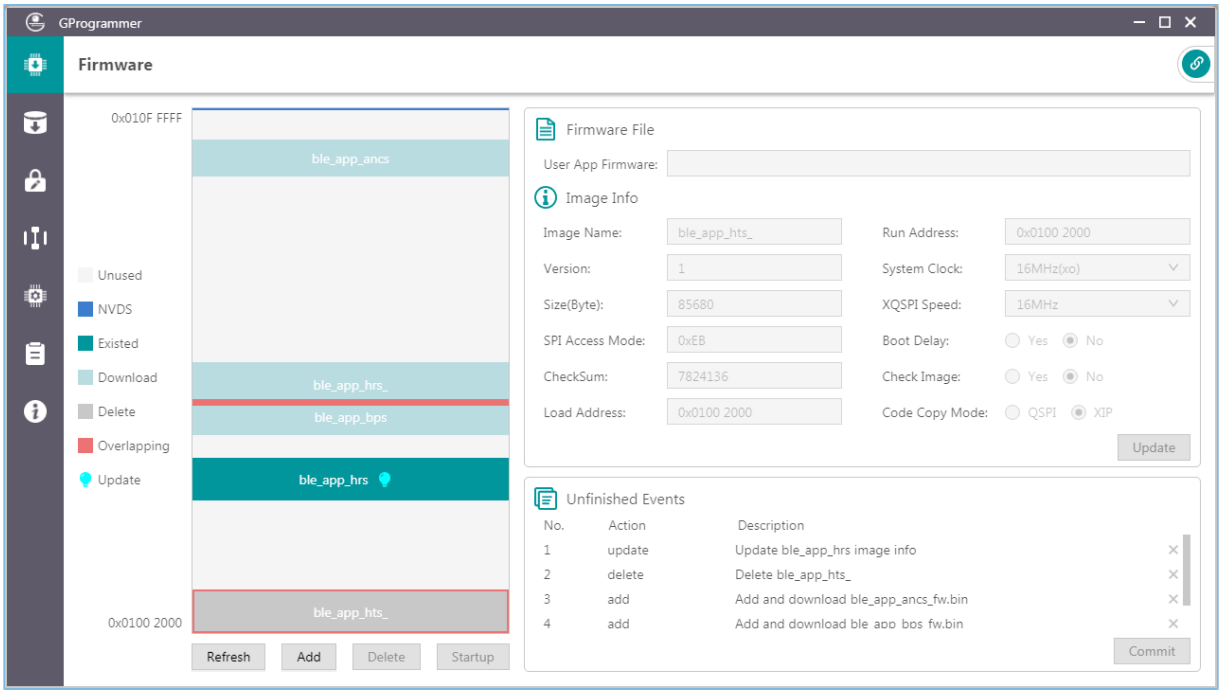

Figure 1-1 GProgrammer GUI

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).

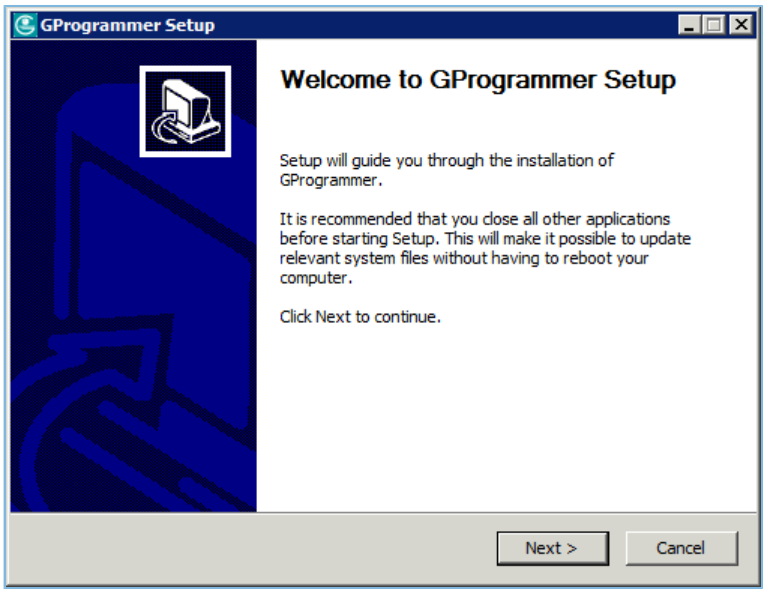

Figure 2-1 **GProgrammer Setup** installation wizard

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.

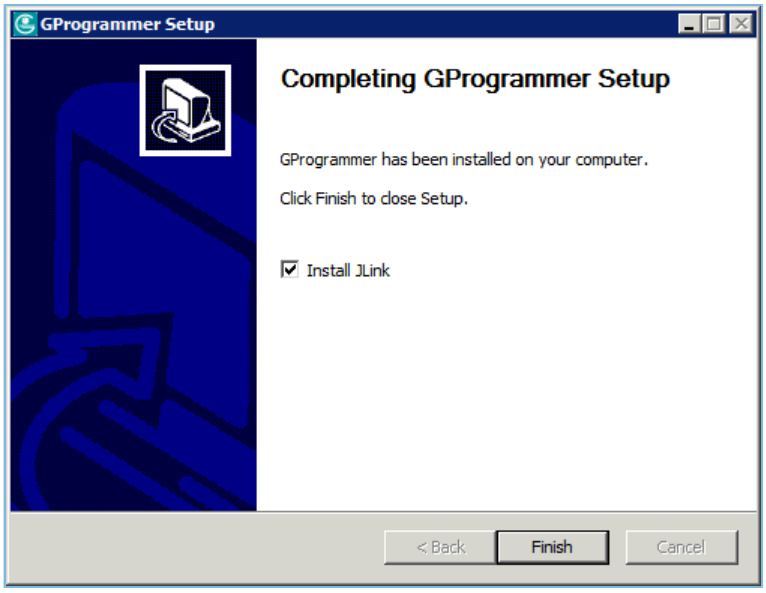

Figure 2-2 Prompt to install J-Link

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.

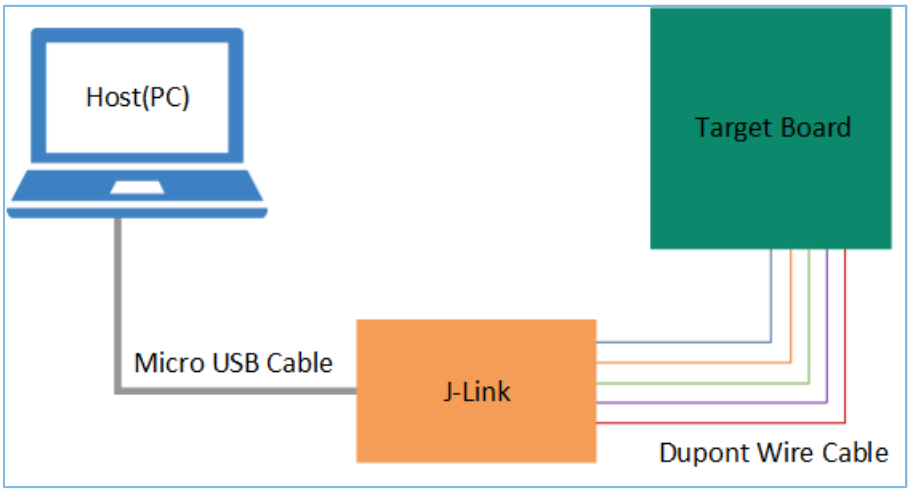

Figure 3-1 Host-target-board connection in SWD mode

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	GR551x SoC Pin
VCC	VCC
GND	GND
SWDIO	GPIO_1
SWCLK	GPIO_0

Tip:

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.

- 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.

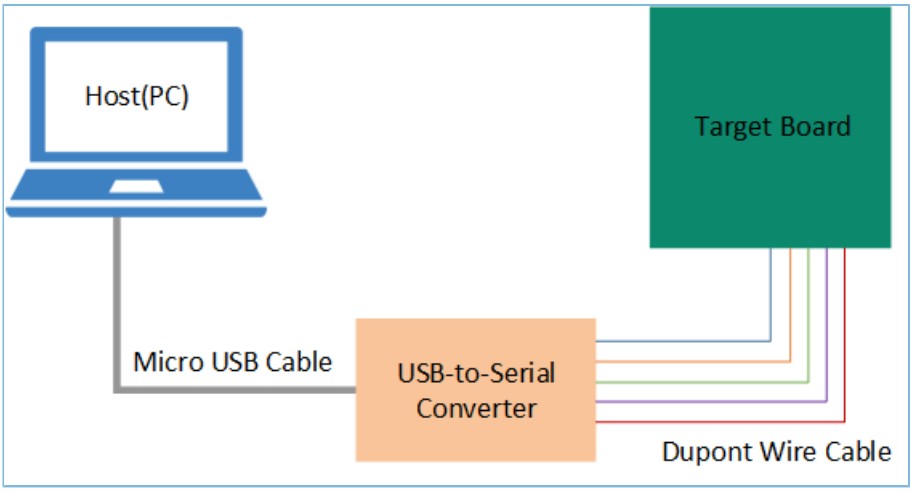

Figure 3-2 Host-target-board connection in UART mode

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	GR551x 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.

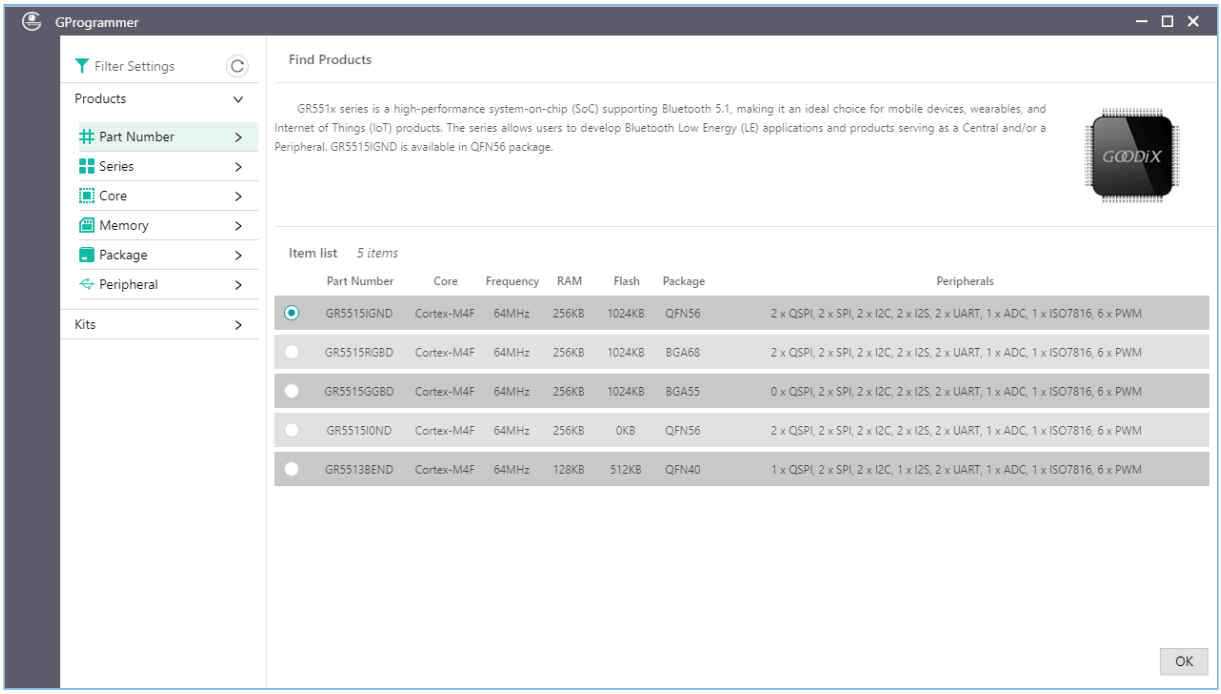

Figure 3-3 SoC selection interface

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 a SoC. For details of all peripherals, see the datasheet corresponding to SoC series.

3.3 Main Operational Interface

After you choose a GR551x SoC, the main operational interface opens, as shown in the figure below.

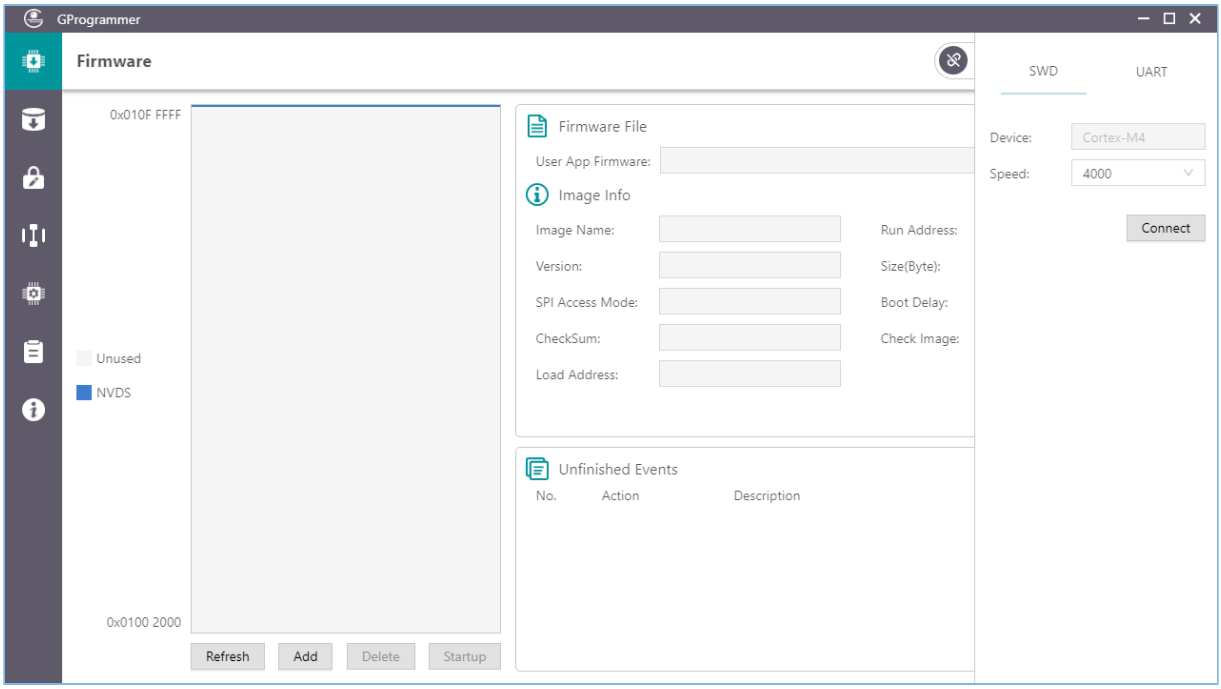

Figure 3-4 GProgrammer GUI (for GR551x series)

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.

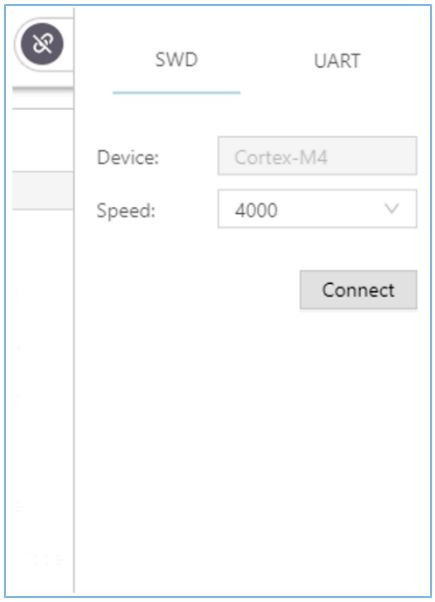

Figure 3-5 GProgrammer SWD connection

- 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.

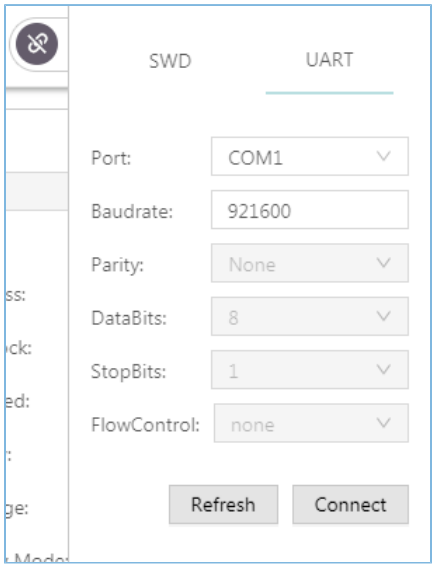

Figure 3-6 GProgrammer UART connection

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**.

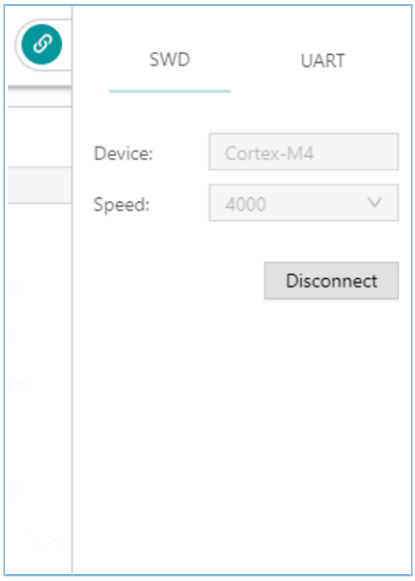

Figure 3-7 Clicking **Disconnect** on GProgrammer

3.5 Firmware

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

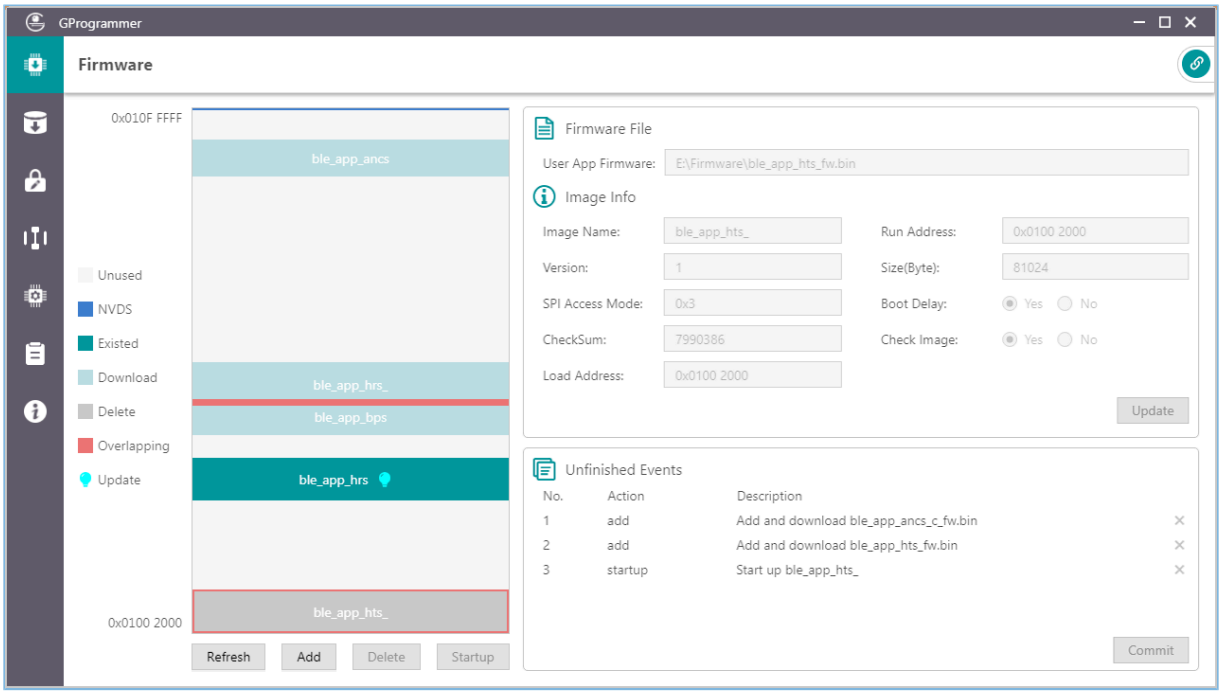

Figure 3-8 GProgrammer **Firmware** interface

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

3.5.1 Download 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.

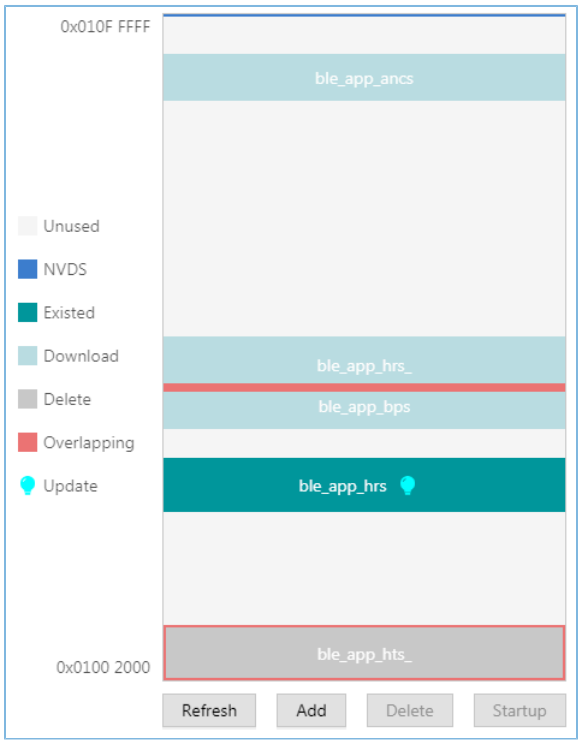

Figure 3-9 Flash firmware layout

- 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_ancs.
- indicates space for storing to-be-downloaded firmware. Example: ble_app_hrs.
- indicates space for storing downloaded firmware in flash memories. Example: ble_app_bps.
- indicates space overlapped by two pieces of firmware. Examples: ble_app_T3u and ble_app_hts.

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

1. Click **Add** to add a local firmware file to GProgrammer. GProgrammer presents details of the added firmware such as firmware directory (**User App Firmware**) and **Image Info**.
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 GR5515 SK Board fails. At this moment, the GR551x SoC may be in sleep mode (the firmware keeps running in sleep mode). You can press **RESET** on the GR5515 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.

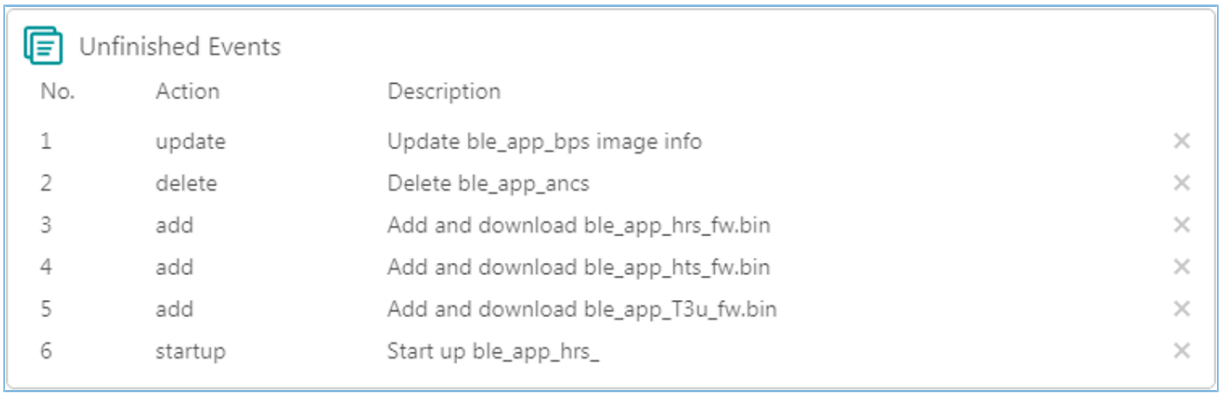

Figure 3-10 Action order

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:** 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.

Name	Button/Icon	Description
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.

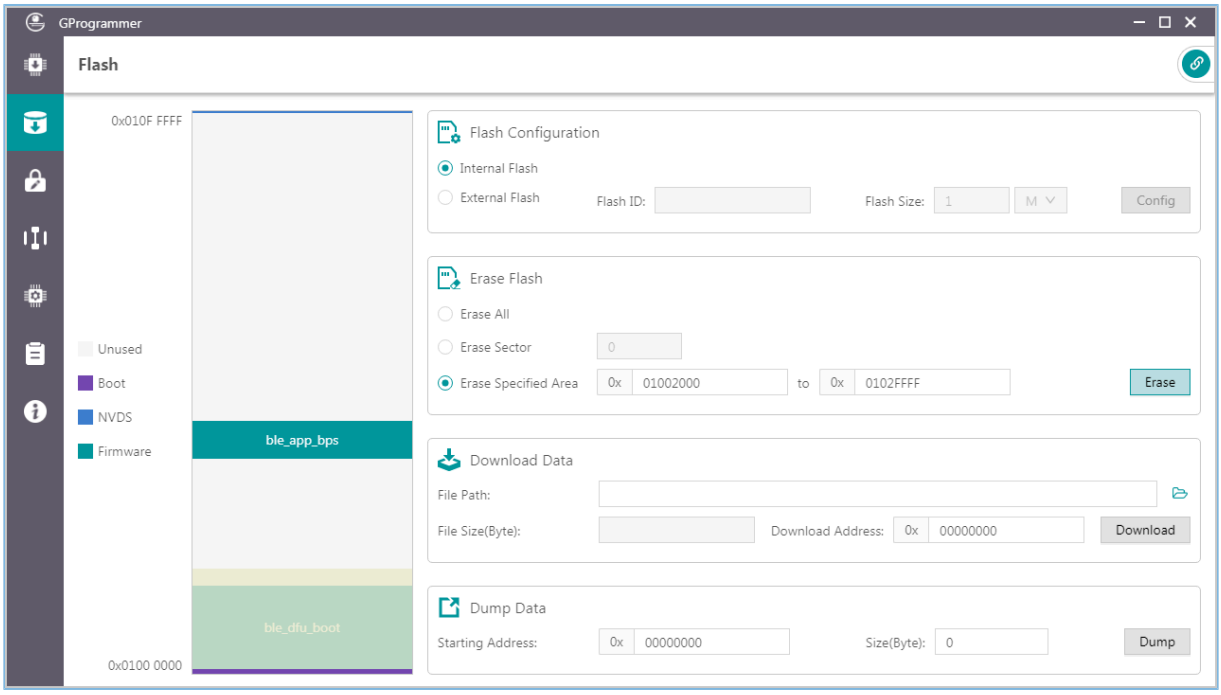

Figure 3-11 GProgrammer **Flash** interface

GProgrammer allows users to program internal and external flash memories of GR551x 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). 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.

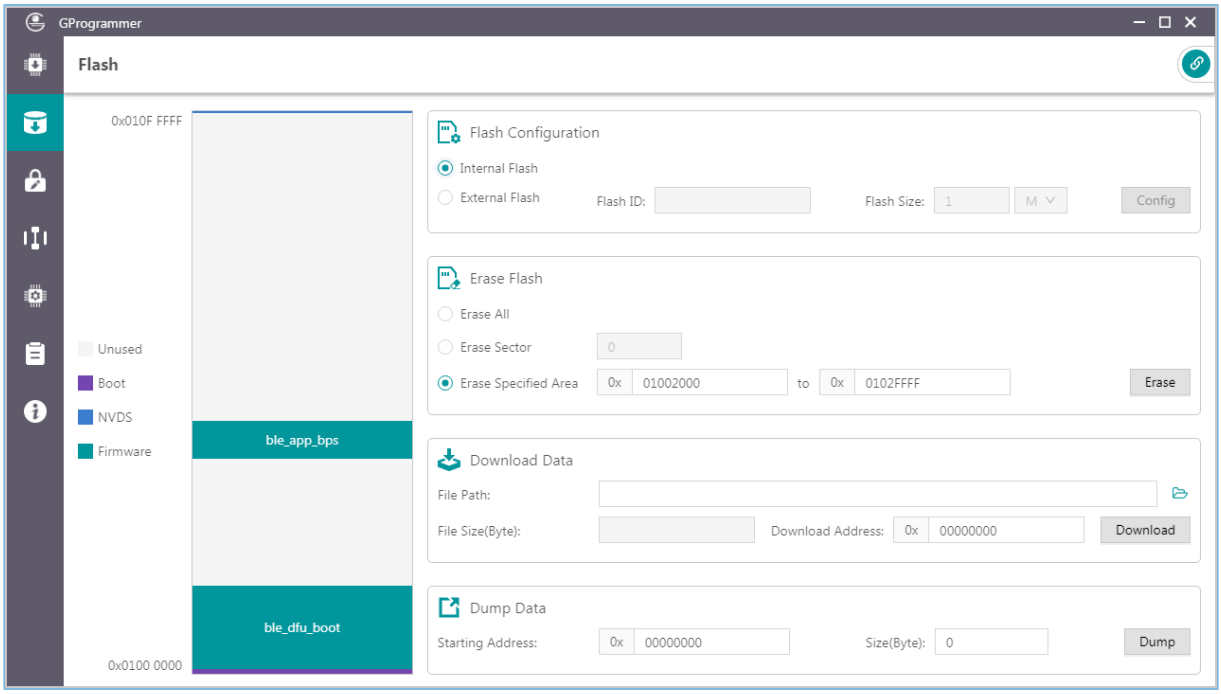

Figure 3-12 Selecting **Internal Flash**

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 space.

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

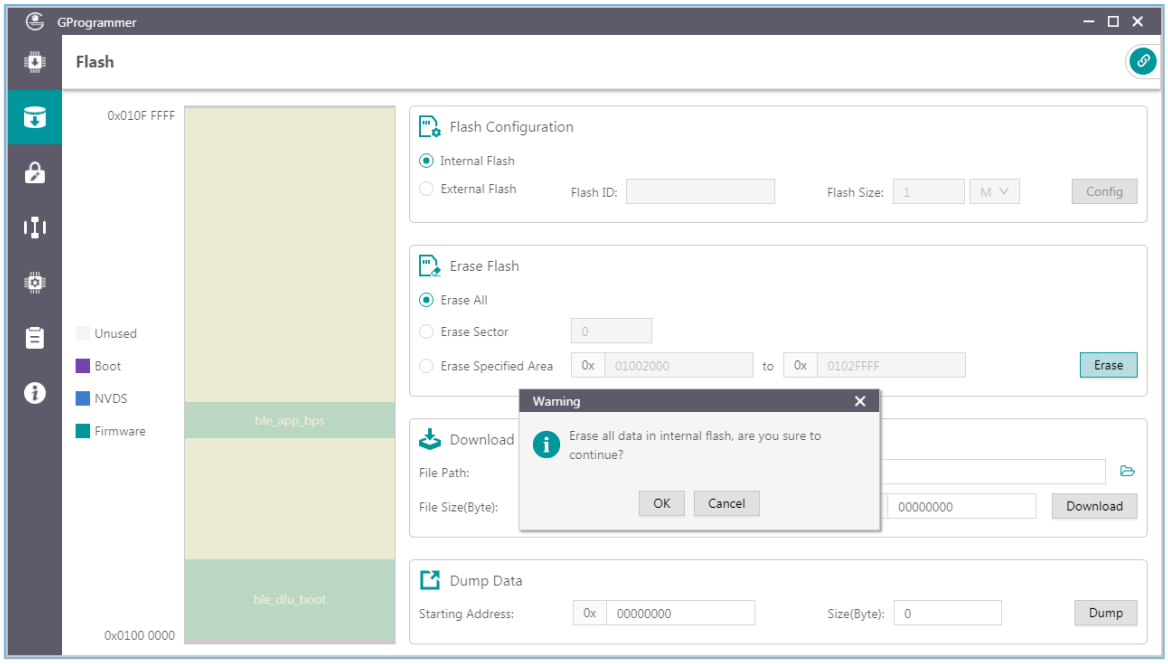

Figure 3-13 **Erase All** on GProgrammer

- **Erase Sector**

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

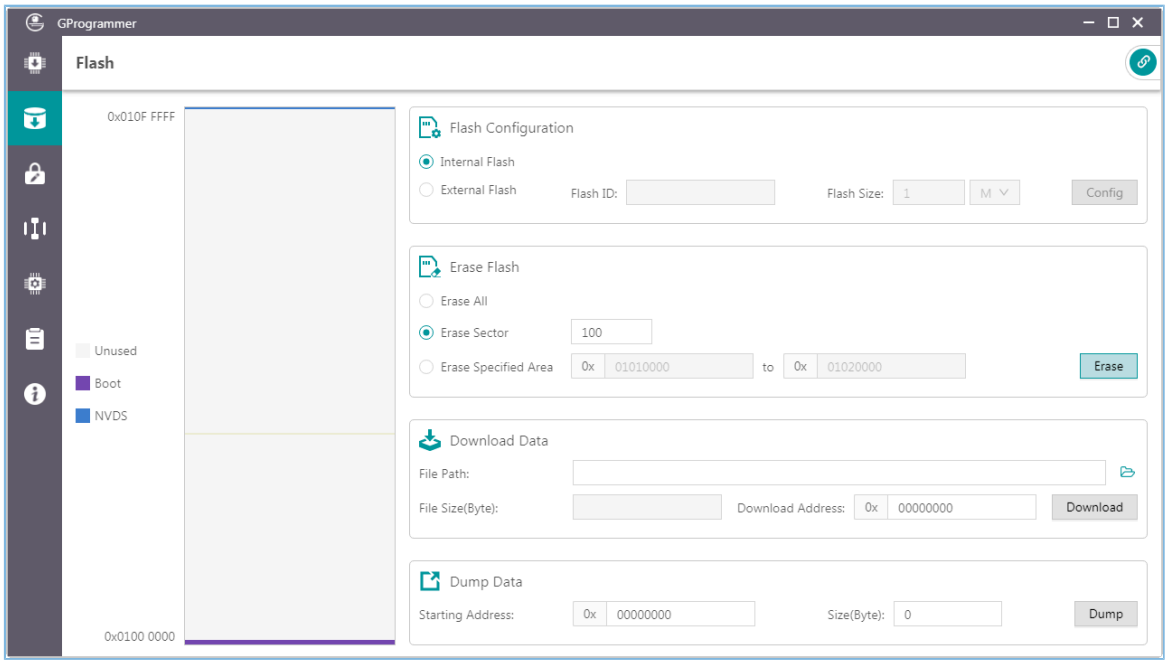

Figure 3-14 **Erase Sector** on GProgrammer

- **Erase Specified Area**

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

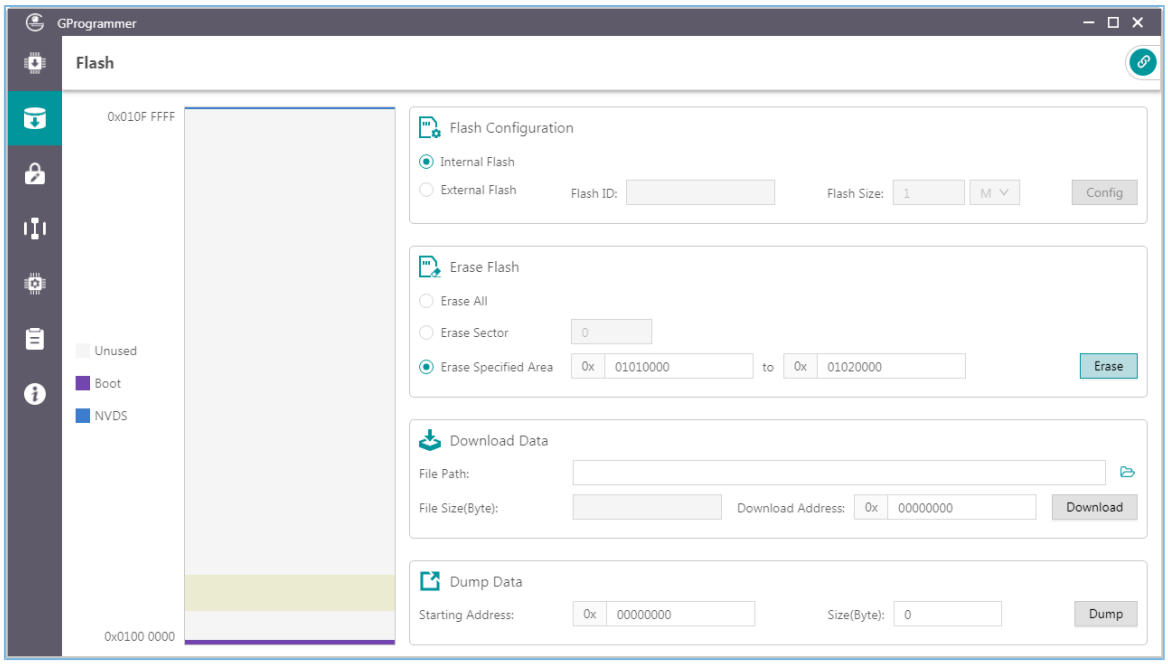

Figure 3-15 **Erase Specified Area** on GProgrammer

3.6.1.3 Download Data

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

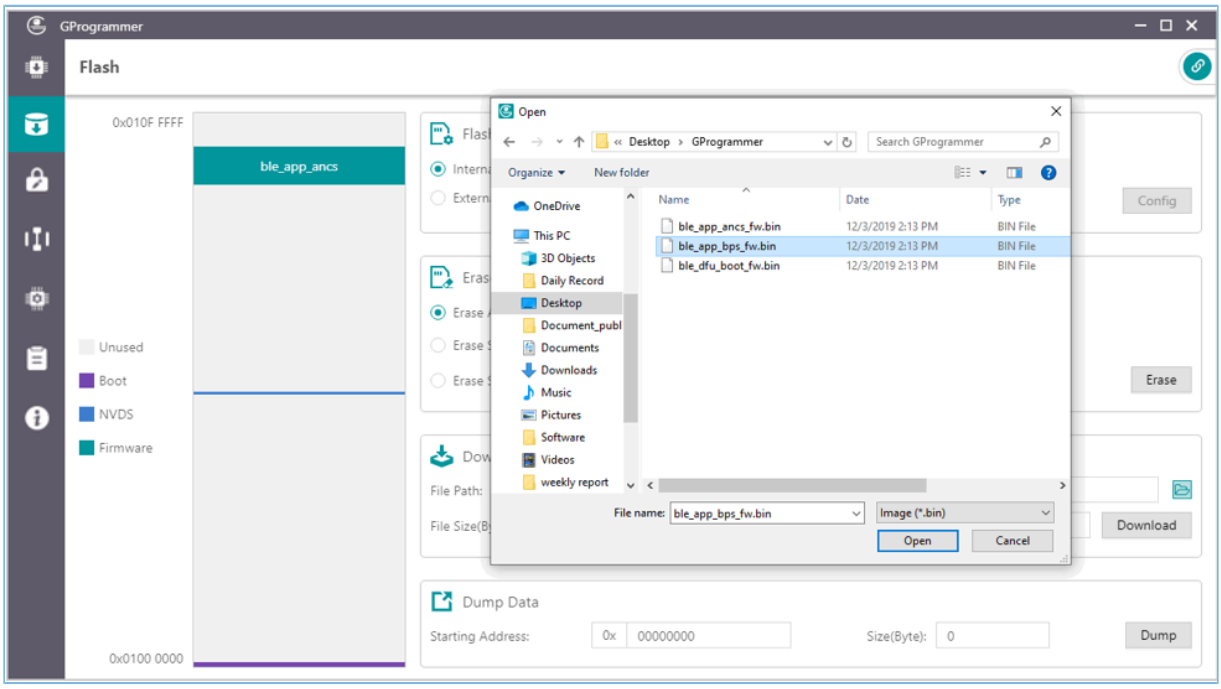

Figure 3-16 Viewing and selecting a data file to be downloaded

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

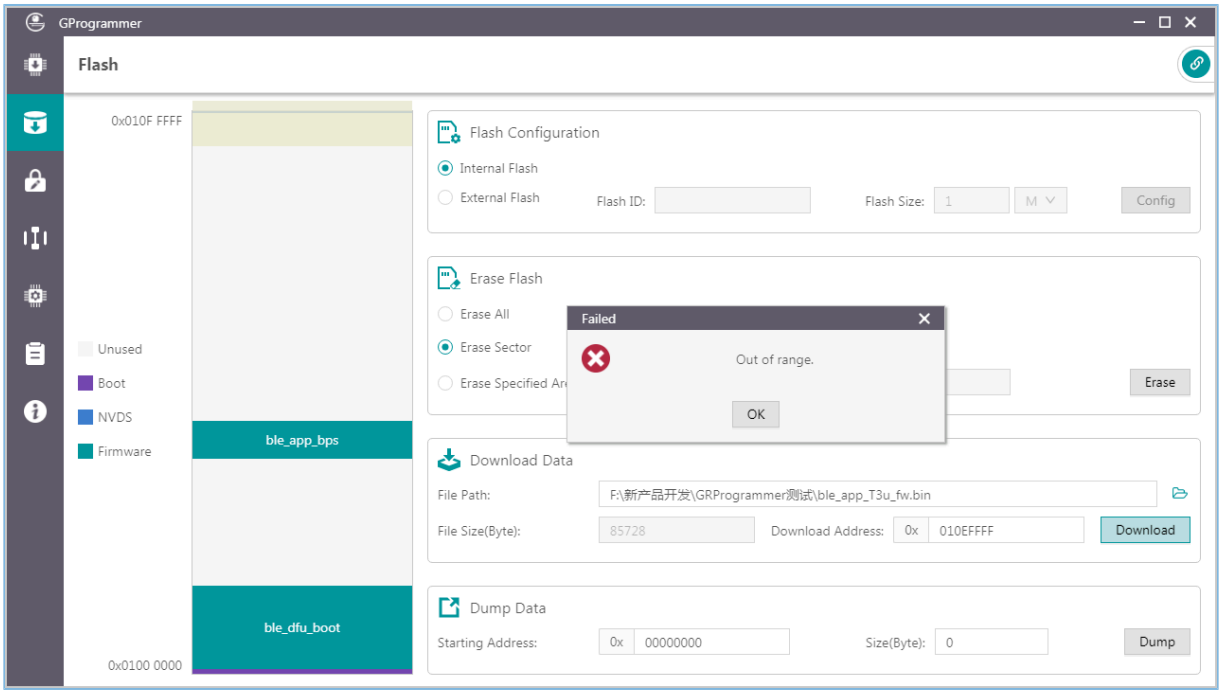

Figure 3-17 Flash overflow error

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.

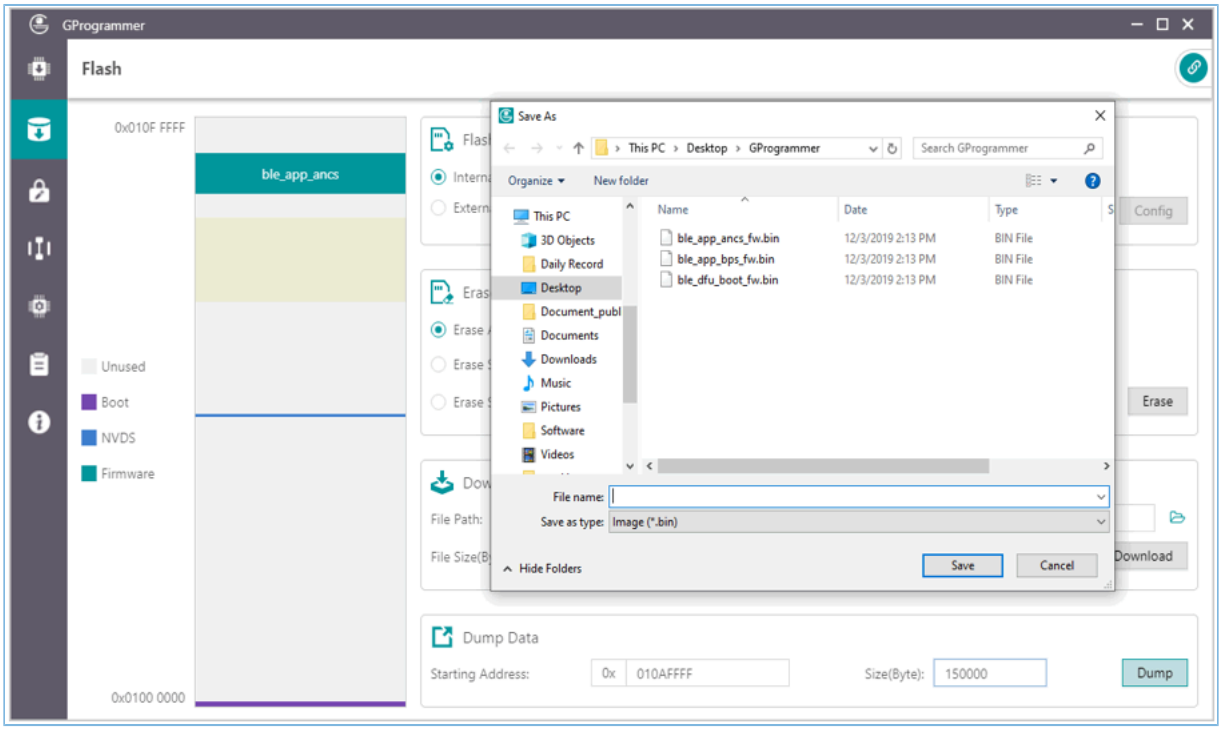

Figure 3-18 **Dump Data** on GProgrammer

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.

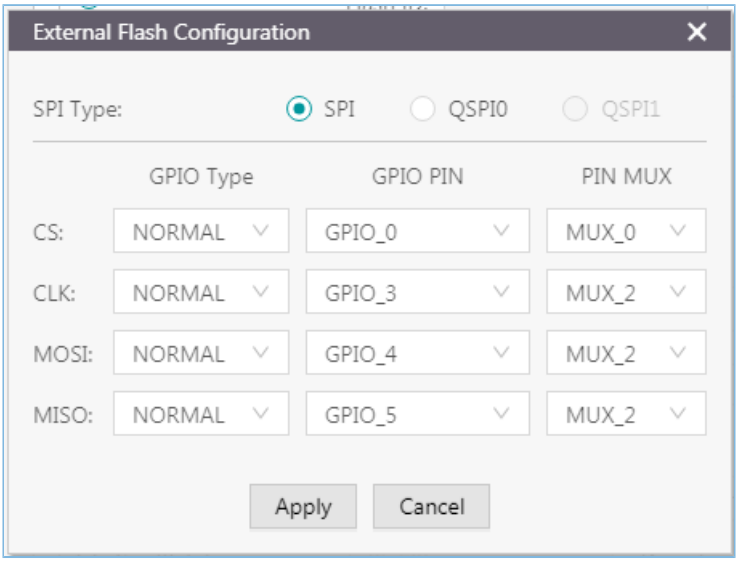

Figure 3-19 SPI configurations

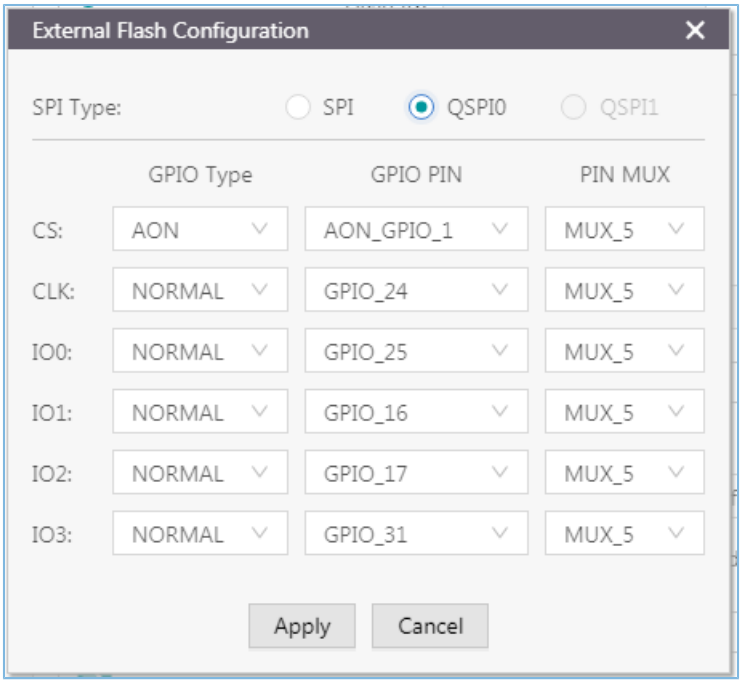

Figure 3-20 QSPI0 configurations

- 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.

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.

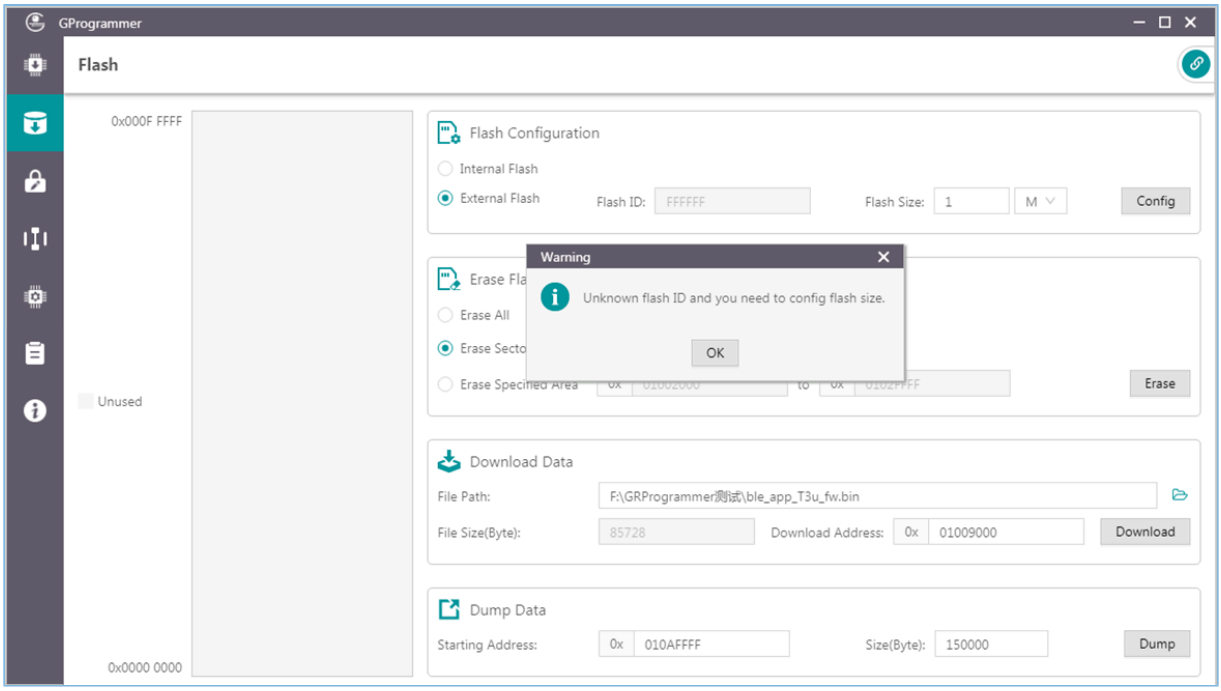

Figure 3-21 Unknown flash ID

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.

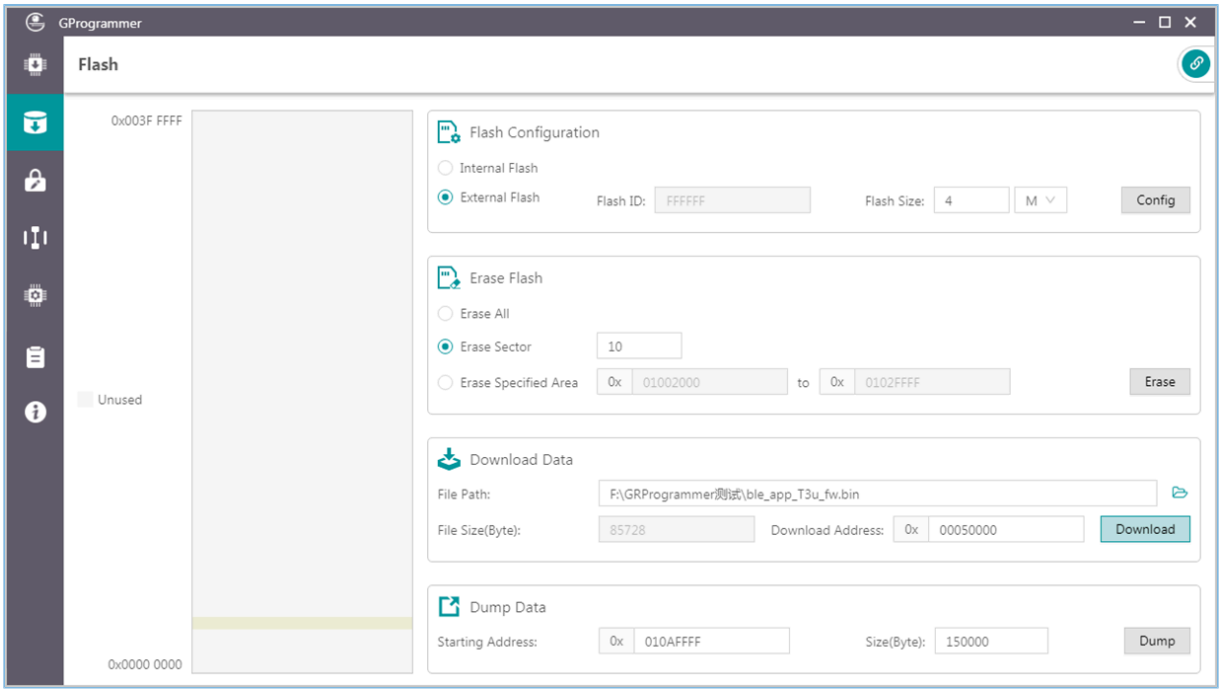

Figure 3-22 **Download Data** to external flash on GProgrammer

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.

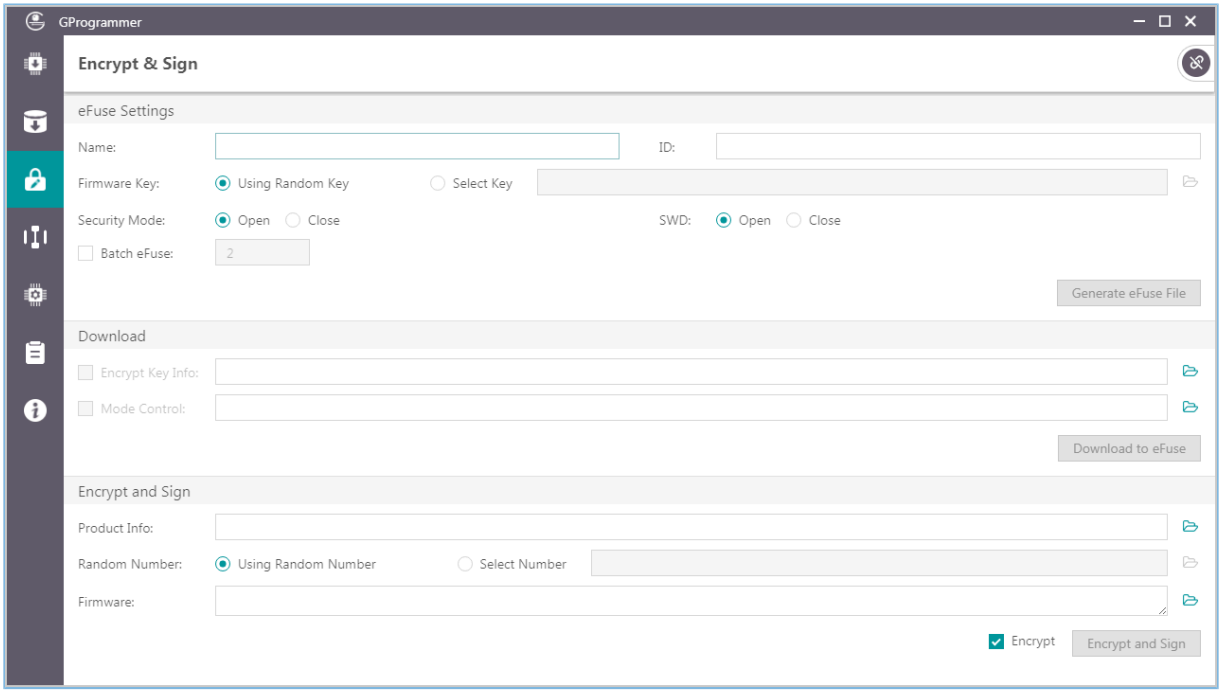

Figure 3-23 GProgrammer **Encrypt & Sign** interface

GR551x 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 GR551x 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.

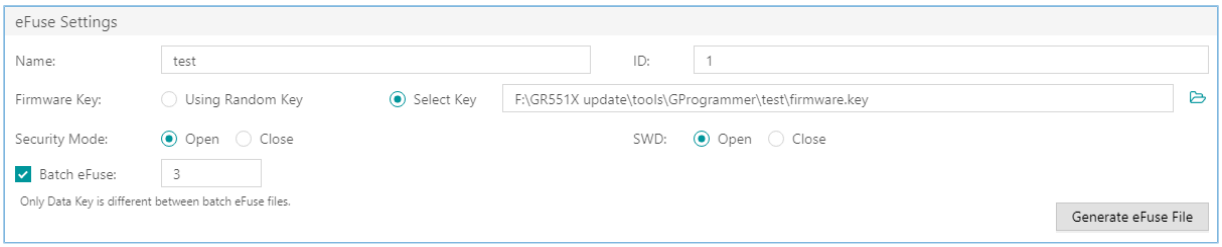

Figure 3-24 Setting eFuse parameters

Note:

- Firmware 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:

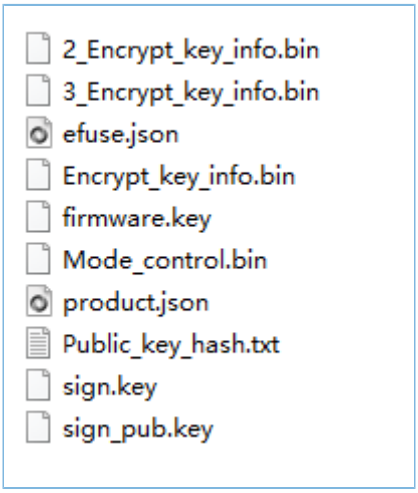

Figure 3-25 Generated files

- *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.

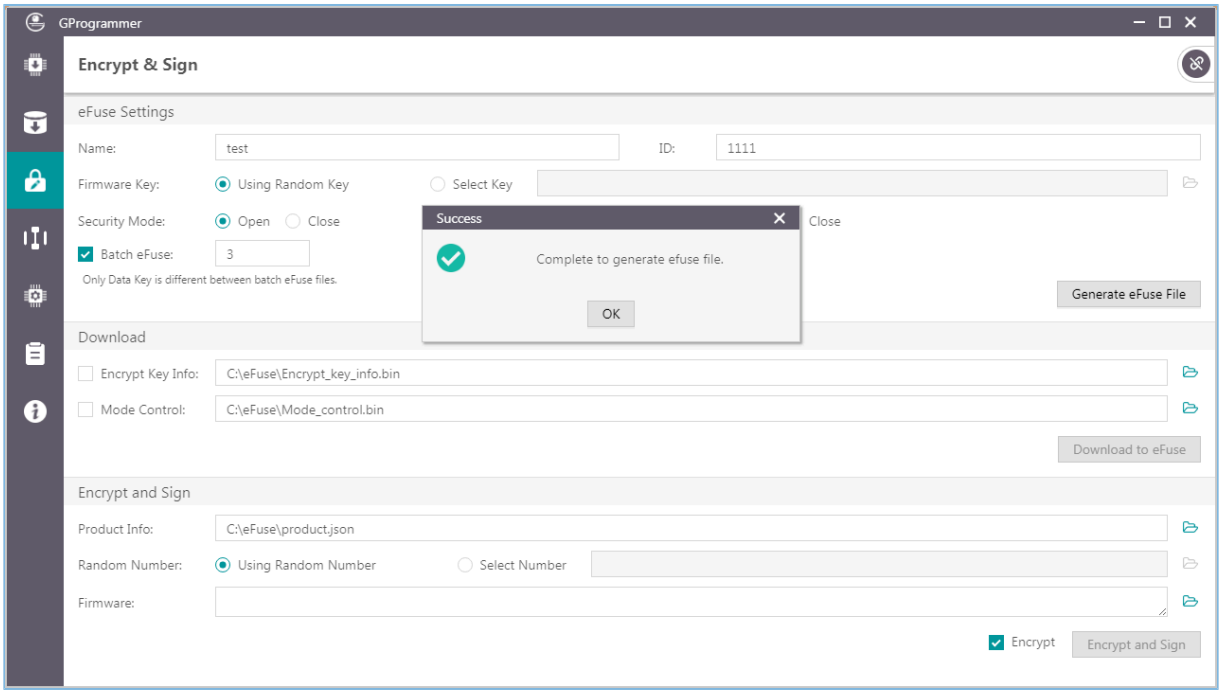

Figure 3-26 Paths for automatically loaded files

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.

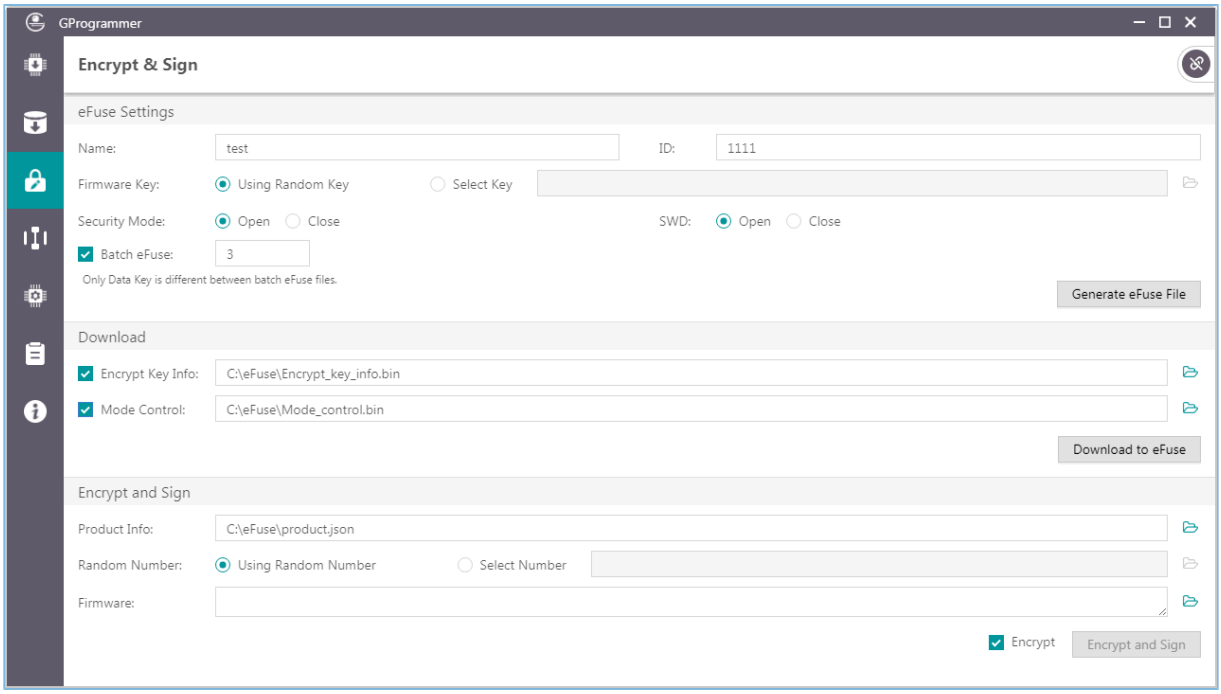

Figure 3-27 Downloading files to eFuse

Note:

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 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.

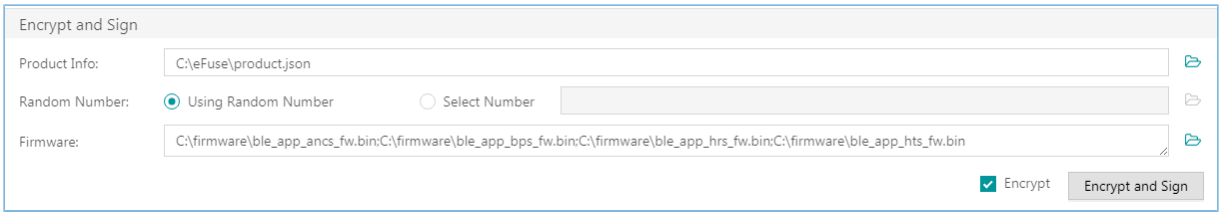

Figure 3-28 Adding more than one firmware file

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, uncheck 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 listed below:

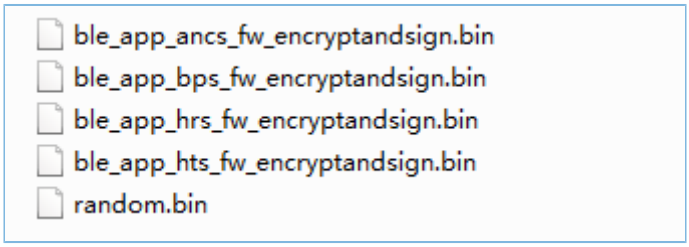

Figure 3-29 GProgrammer-generated files after encryption and signing

Files after being signed are listed below:

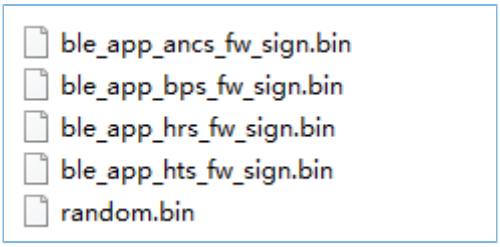

Figure 3-30 GProgrammer-generated files after signing

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.

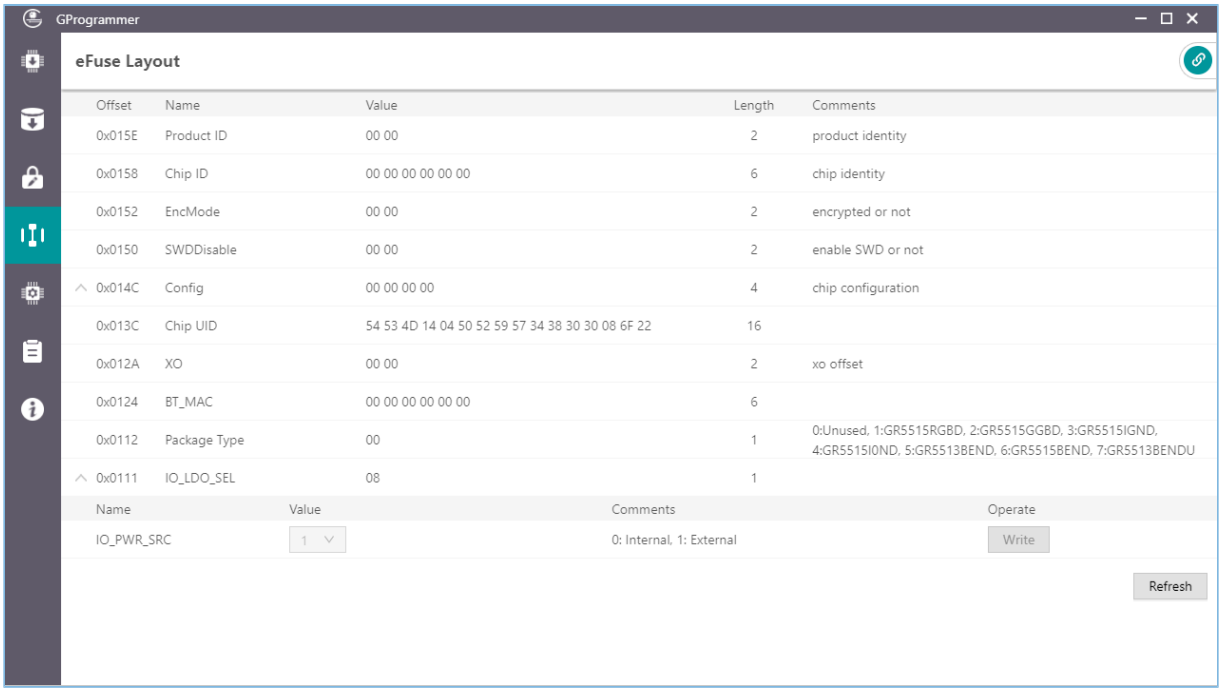

Figure 3-31 **eFuse Layout** interface

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.

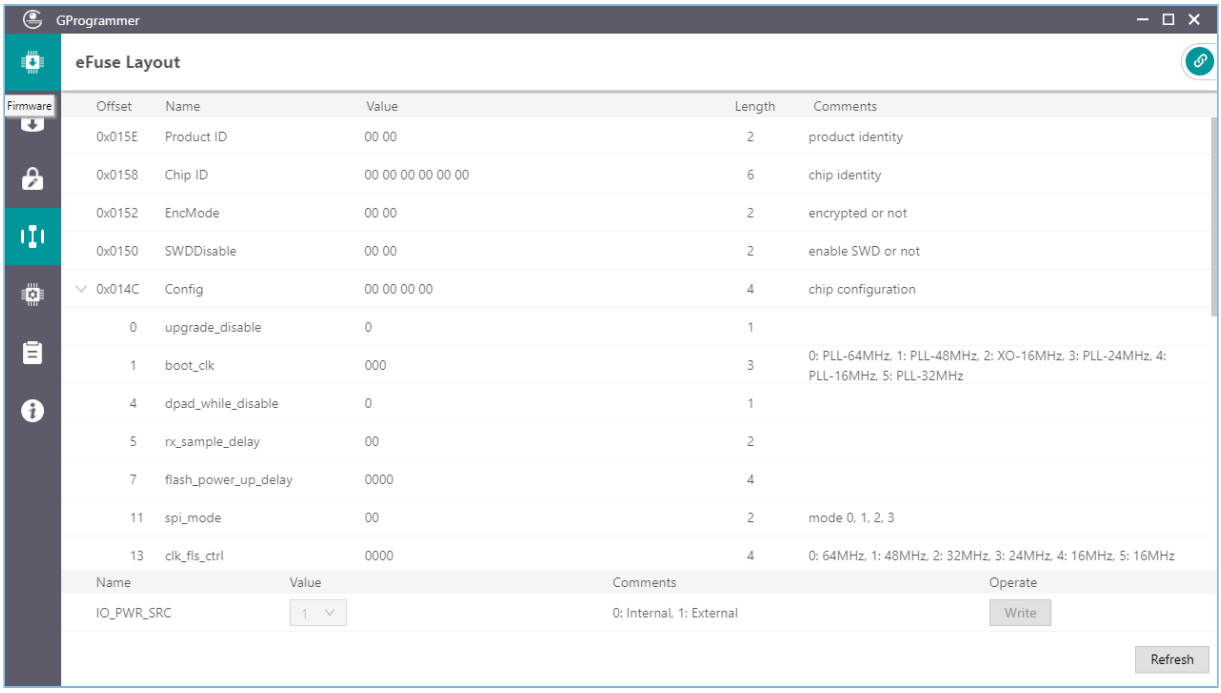

Figure 3-32 Expanded **Offset**

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 Chip Configuration

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

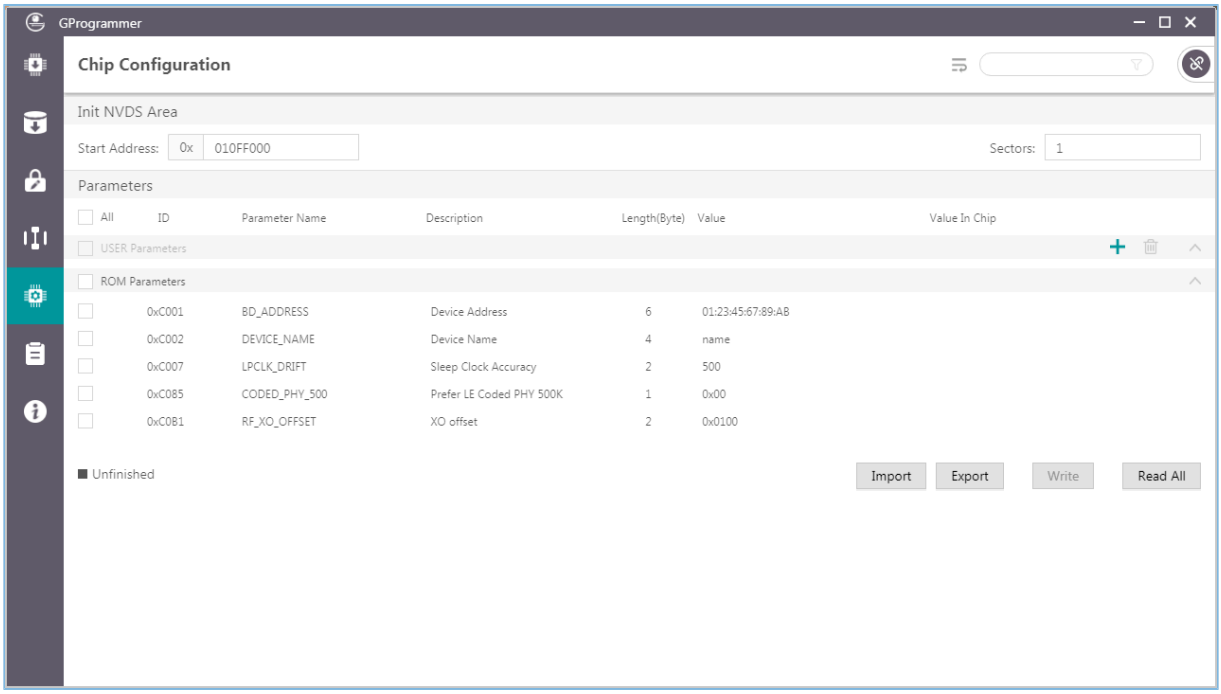

Figure 3-33 GProgrammer **Chip Configuration** interface

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.

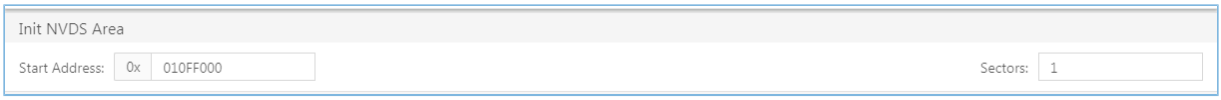

Figure 3-34 Setting the starting address and sector quantity in the NVDS area

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

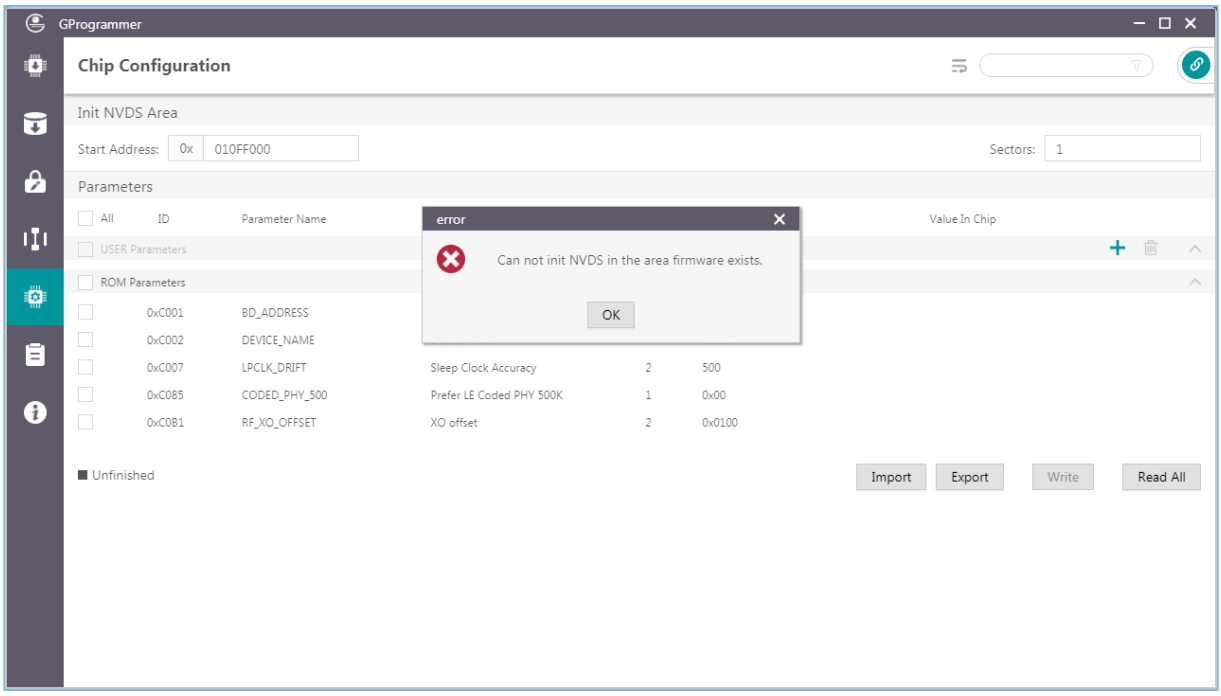

Figure 3-35 NVDS initialization failure

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)

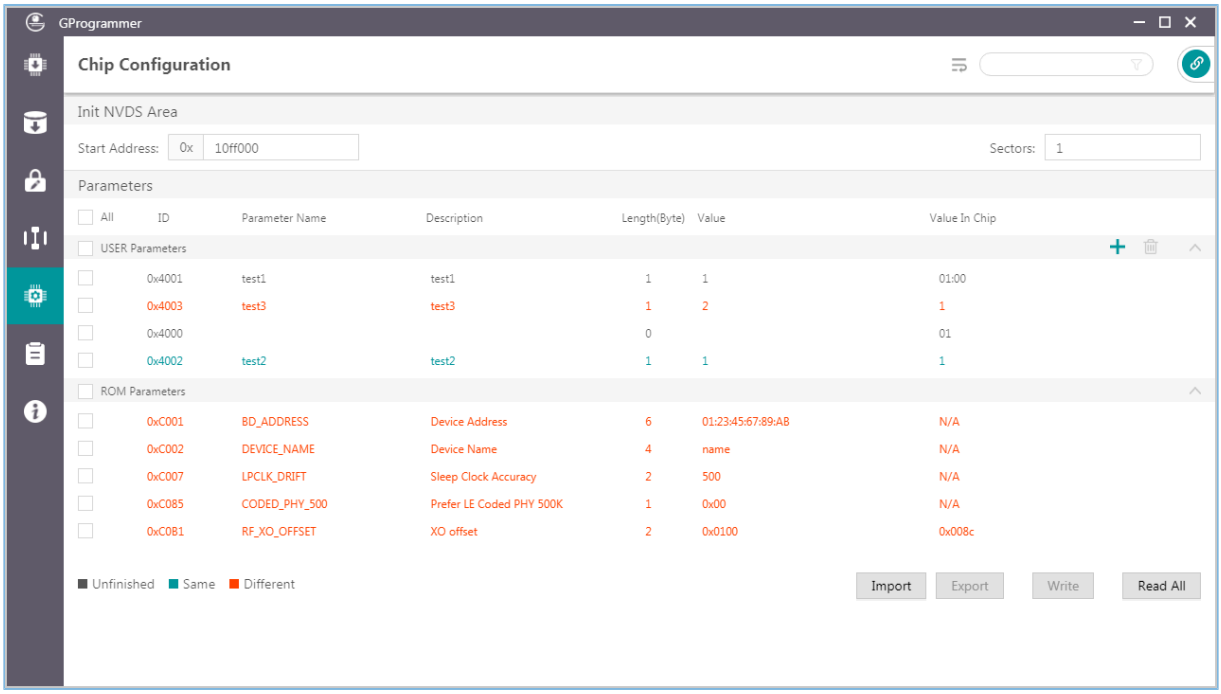

Figure 3-36 **Read All** interface

3.9.3 Write

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

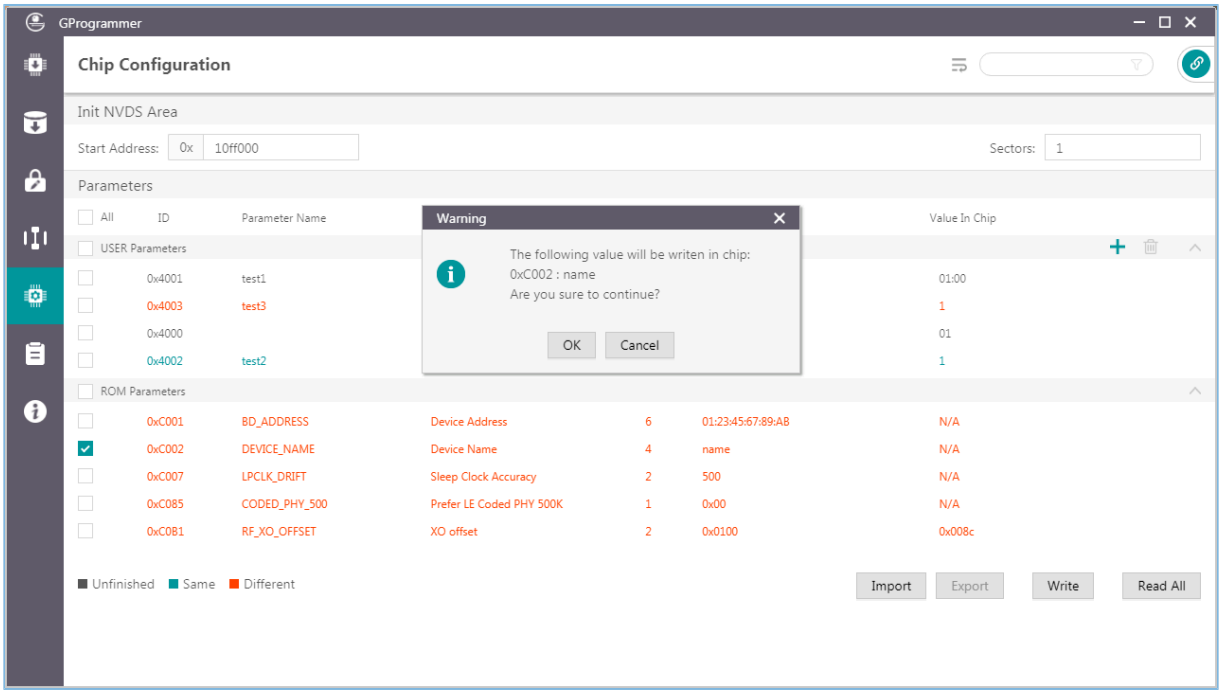

Figure 3-37 **Write** parameters to NVDS

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**).

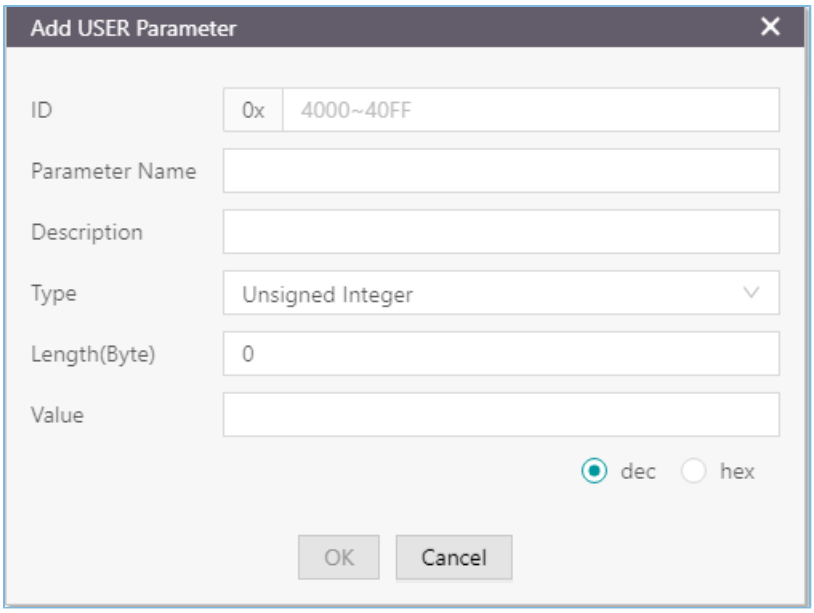

Figure 3-38 Adding a user parameter to NVDS

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.

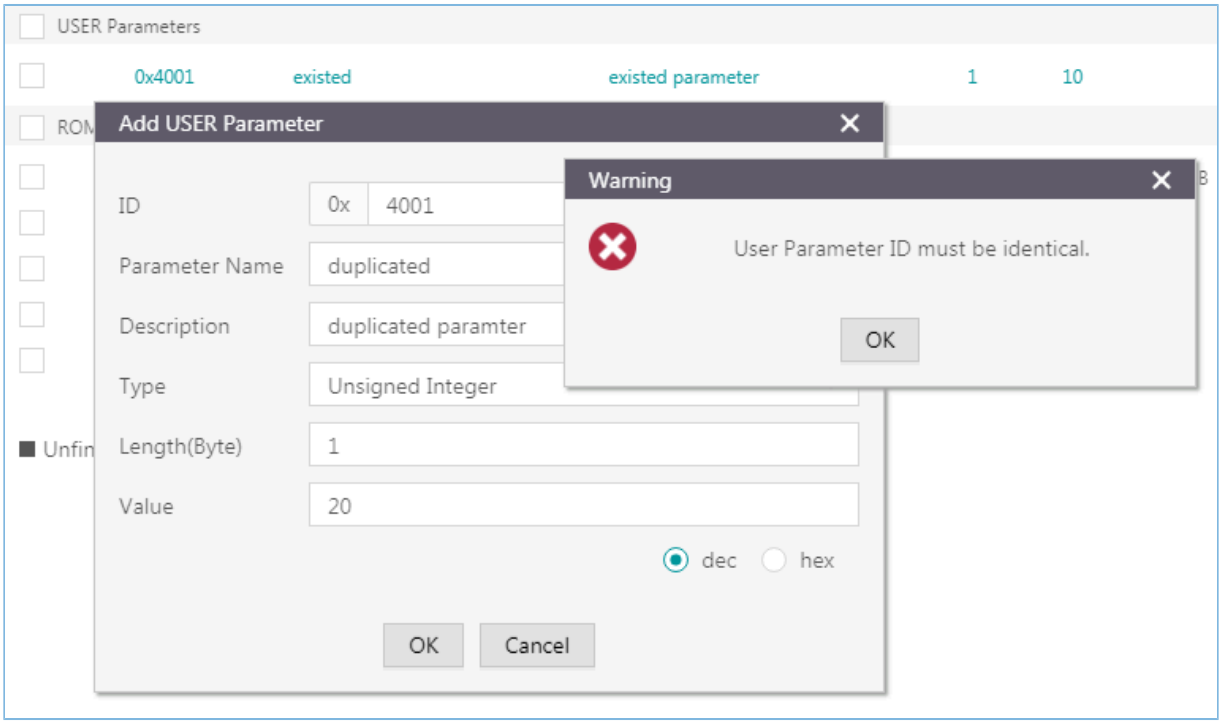

Figure 3-39 Failure to add a user parameter due to an identical parameter ID

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.

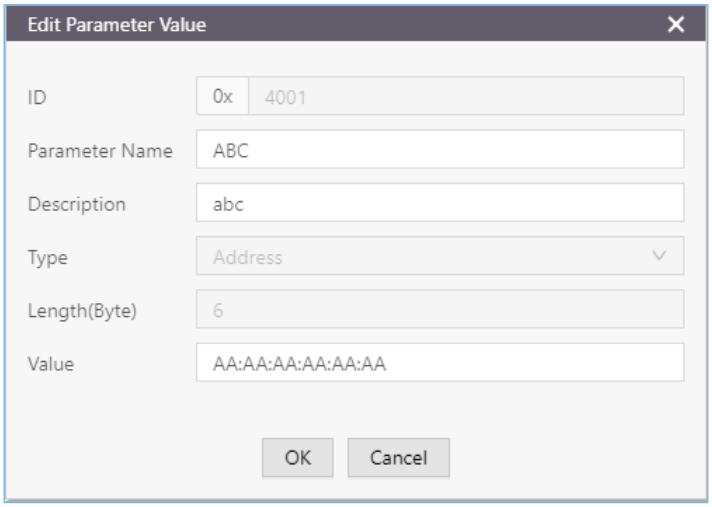

Figure 3-40 **Edit Parameter Value** window

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.

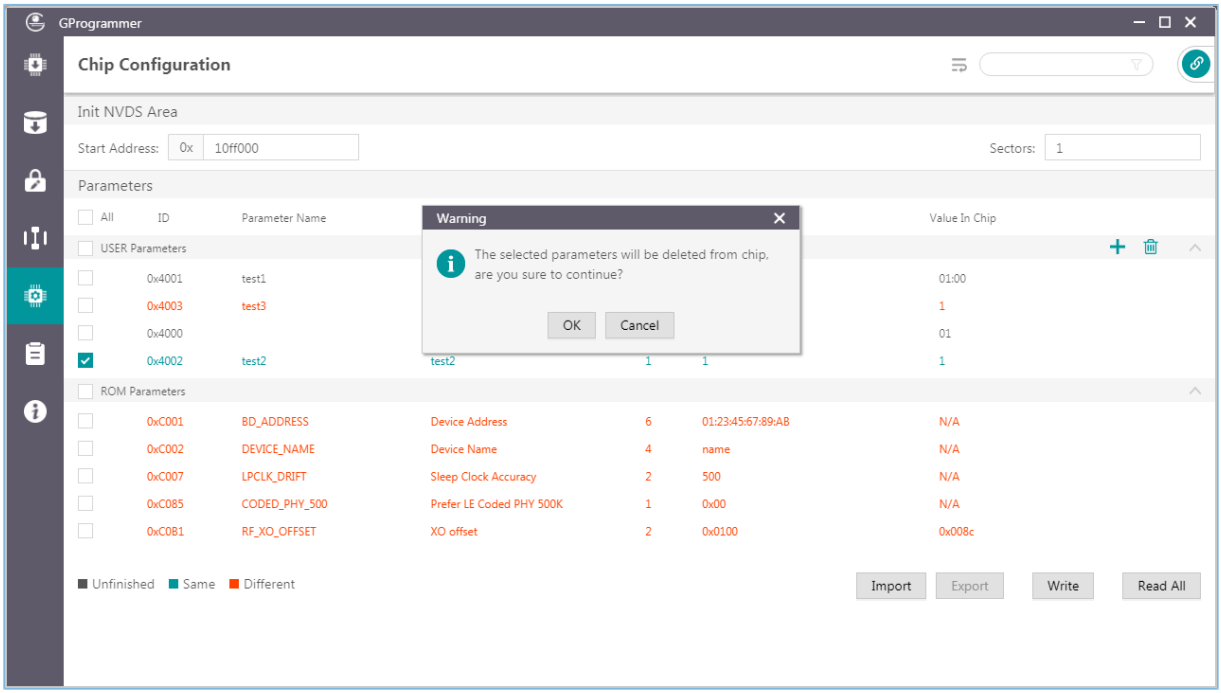

Figure 3-41 Removing a parameter

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.

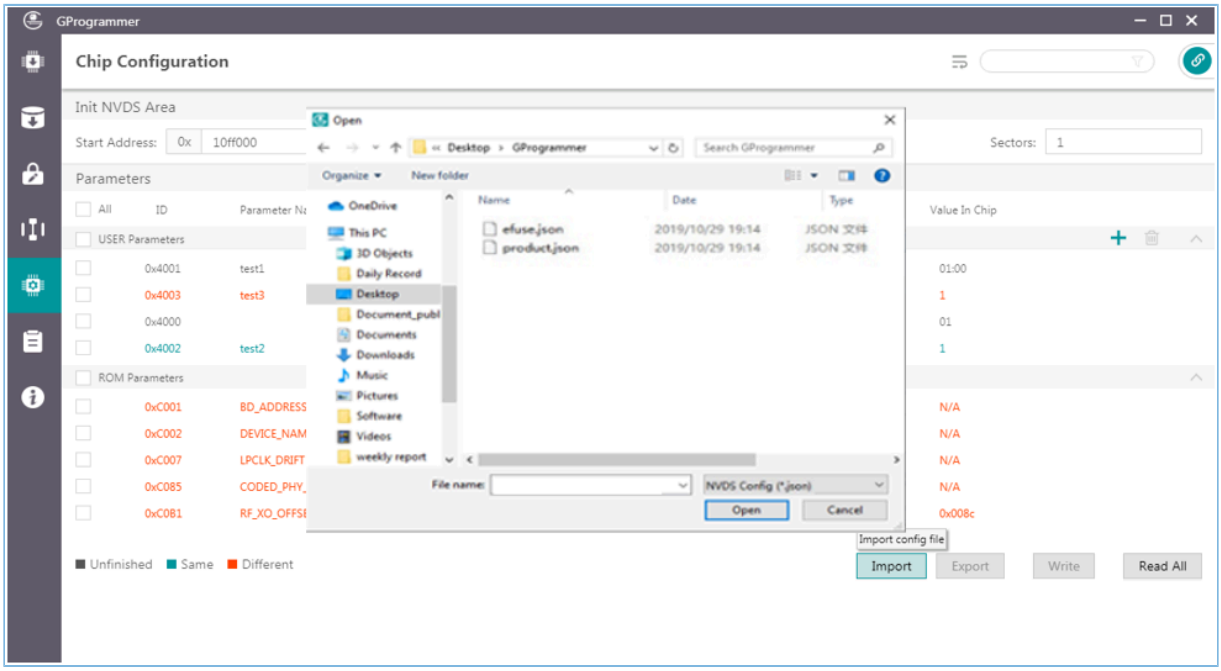

Figure 3-42 Importing local JSON configuration files to GProgrammer

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.

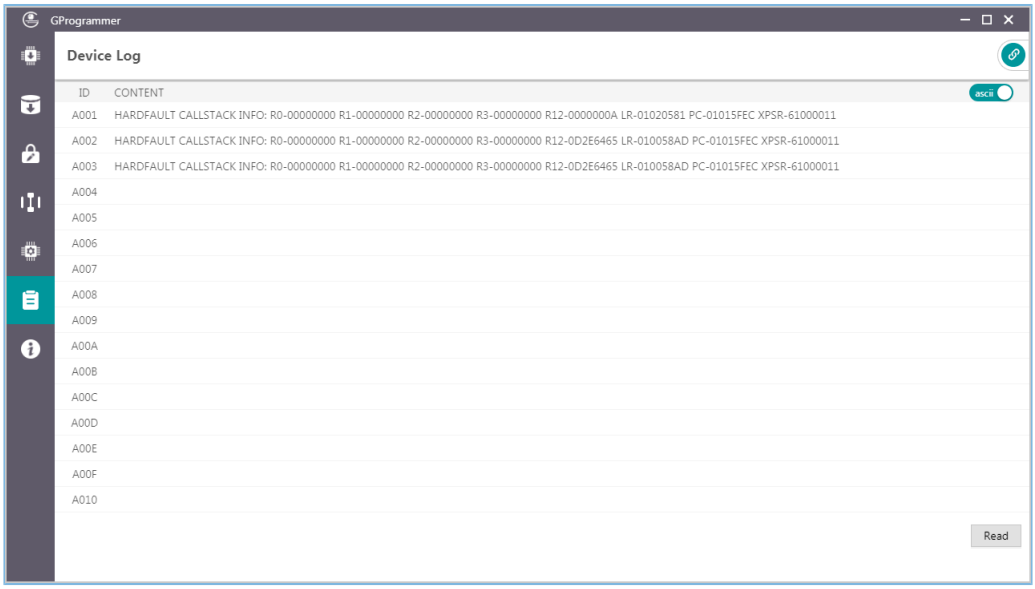

Figure 3-43 **Device Log** interface

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.

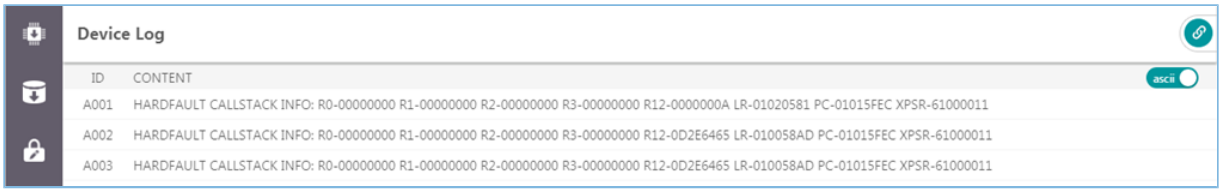

Figure 3-44 Device logs in ASCII characters

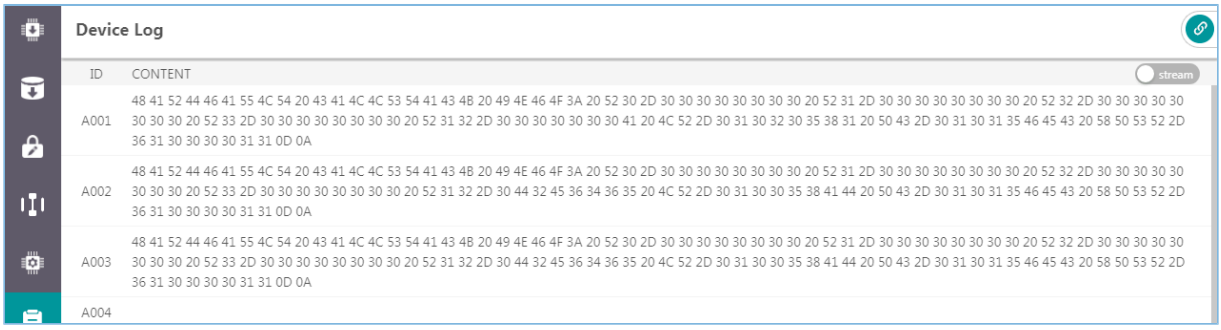

Figure 3-45 Device logs in byte streams

3.11 Command-line Programs

Goodix provides two command-line programs in the GProgrammer installation directory: *GR551x_console.exe* and *GR551x_encrypt_signature.exe*.

- *GR551x_console.exe* supports firmware download and flash programming in GR551x SoCs in a command-line interface.
- *GR551x_encrypt_signature.exe* supports firmware encryption and signing or firmware signing in a command-line interface.

3.11.1 *GR551x_console.exe*

Follow the steps below to run *GR551x_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 GR551x_console.exe command to complete corresponding operations. The details about the command are shown in Table 3-6.

Table 3-6 GR551x_console supported commands

Command	Functional Description	Command Format and Parameter Description
program	Programs firmware files to internal SoC flash memories.	program <firmware file path> <run immediately:y | n> 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.
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> Parameter description: • <start address<(hex)>>: It represents the start address of the storage area to be erased (in hexadecimal).

Command	Functional Description	Command Format and Parameter Description
		• <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.
eraseall	Erases all flash memory data within an SoC.	eraseall
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> 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> 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
help	Displays all help information.	help

The example below shows how to use the program command to download a firmware file to internal SoC flash memories and run the firmware immediately after downloading.

```
GR551x_console.exe program “C:\ble_app_hrs_fw.bin” y
```

The downloading progress is displayed in real time during executing the program command, as shown in Figure 3-46.

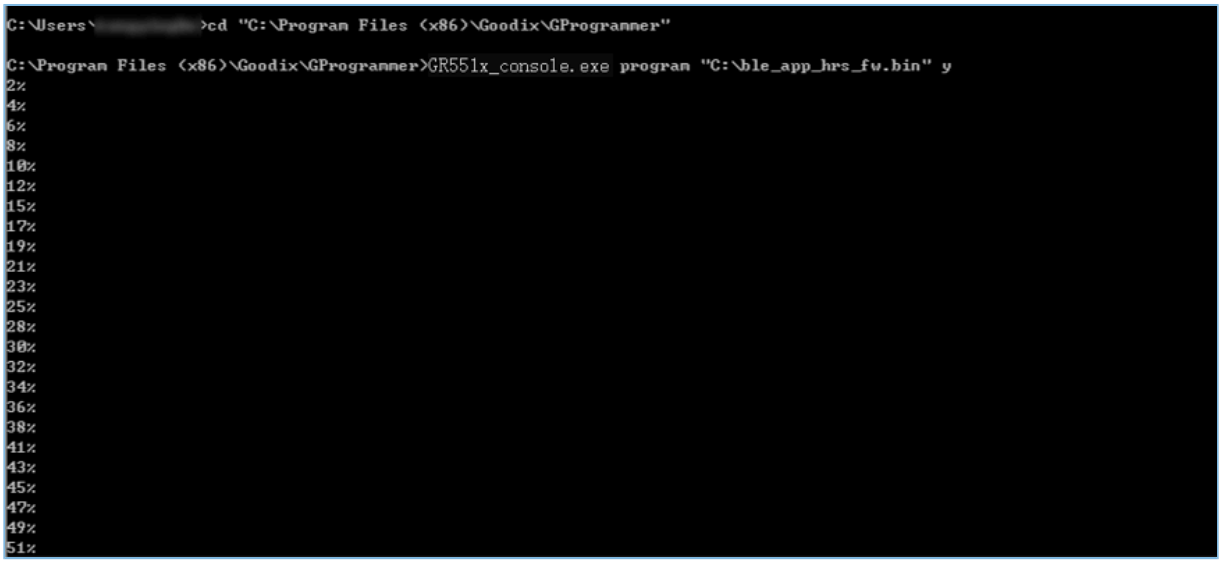

Figure 3-46 Executing the program command

Note:

You cannot operate *GR551x_console.exe* while GProgrammer is running.

3.11.2 *GR551x_encrypt_signature.exe*

Follow the steps below to run *GR551x_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 `GR551x_encrypt_signature.exe --parameter` to complete corresponding operations. The details about “parameter” are shown in Table 3-7. (Only the most frequently used parameters are listed in the table. To view all parameters, enter `GR551x_encrypt_signature.exe --help`.)

Table 3-7 Frequently used parameters for *GR551x_encrypt_signature.exe*

Parameter	Description
operation	Encrypts and signs firmware or signs firmware only. Options: encryptandsign and sign.
firmware_key	Shows the directory of *firmware.key*, which is used for encryption and signing, or signing only.
signature_key	Shows the directory of *sign.key*, which is used for encryption and signing, or signing only.
signature_pub_key	Shows the directory of *sign_pub.key*, which is used for encryption and signing, or signing only.
product_json_path	Shows the directory of *product.json*, which is used for encryption and signing, or signing only.
rand_number	Shows the directory of *random.bin*, which is used for 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.

Parameter	Description
random_output	Shows the directory that saves the random numbers used in encryption and signing, or signing only.
help	Displays help information.

The example below shows how to use the program command to encrypt and sign firmware:

```
GR551x_encrypt_signature.exe
--operation="encryptandsign"
--firmware_key="C:/eFuse/firmware.key"
--signature_key="C:/eFuse/sign.key"
--signature_pub_key="C:/eFuse/sign_pub.key"
--product_json_path="C:/eFuse/product.json"
--ori_firmware="C:/firmware/ble_app_hrs_fw.bin"
--output="C:/firmware_encryptAndSign/ble_app_hrs_fw_encryptAndSign.bin"
--random_output="C:/firmware_encryptAndSign/random.bin"
```

In the code snippet above, the “C:/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”.

- --ori_firmware=“C:/firmware/ble_app_hrs_fw.bin”: the directory of the firmware before any operation
- --output=“C:/firmware_encryptAndSign/ble_app_hrs_fw_encryptAndSign.bin”: the directory of the encrypted and signed firmware

A sample of executing the encryption and signing command is shown in Figure 3-47:

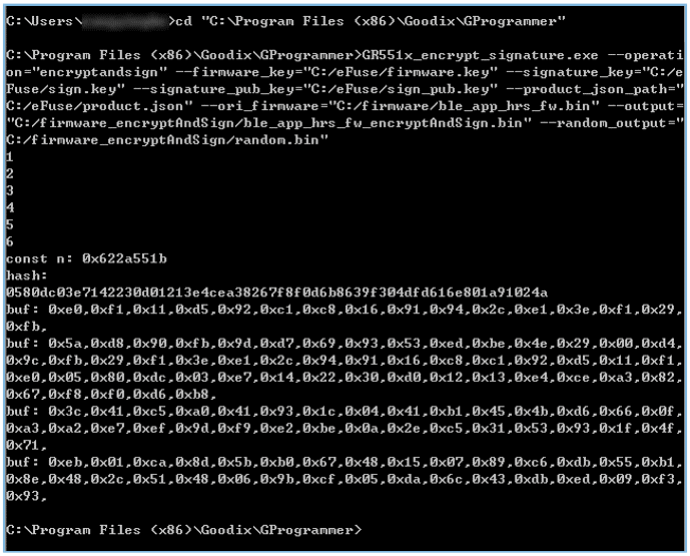

Figure 3-47 Executing the encryption and signing command

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 GR551x_script file in the GProgrammer installation directory.

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.