



GProgrammer User Manual

Version: 2.0

Release Date: 2021-01-05

Copyright © 2021 Shenzhen Goodix Technology Co., Ltd. All rights reserved.

Any excerption, backup, modification, translation, transmission or commercial use of this document or any portion of this document, in any form or by any means, without the prior written consent of Shenzhen Goodix Technology Co., Ltd is prohibited.

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 and GMF03x users
- GR551x and GMF03x developers
- GR551x and GMF03x testers

Release Notes

This document is the sixth release of *GProgrammer User Manual*, corresponding to GProgrammer V1.2.14.

Revision History

Version	Date	Description
1.5	2020-05-30	Initial release
1.6	2020-06-30	<ul style="list-style-type: none"> • Updated the version number based on software update. • Updated sector-related description in “Section 3.3.7 Chip Configuration”. • Added “Section 3.3.9.1 GR551x_console.exe”, introducing a command-line program to erase and download commands; added “Section 3.3.9.2 GR551x_encrypt_signature.exe” and “Section 3.3.9.3 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 “Section 3.3.5 Encrypt & Sign”.
1.7	2020-08-30	<ul style="list-style-type: none"> • Updated GProgrammer version number (including a figure), in line with software update (GProgrammer V1.2.11). • Introduced the GR5515I0ND SoC for GR551x SoCs in “Section 3.2 SoC/MCU Selection”. • Changed icons for “Delete” and “Startup” in “Section 3.3.3 Firmware”.
1.8	2020-09-30	Added description on firmware download failure in “Section 3.3.3.1 Download Firmware”.
1.9	2020-11-26	Updated GProgrammer version number (including a figure), in line with software update (GProgrammer V1.2.13).
2.0	2021-01-05	<ul style="list-style-type: none"> • Updated GProgrammer version number (including a figure), in line with software update (GProgrammer V1.2.14). • Updated software UI figures for SoC/MCU selection and firmware operations.

Contents

Preface	I
1 Introduction	1
2 Installation Instructions	3
2.1 Installation Requirements.....	3
2.2 Installation Steps.....	3
3 Programming Flash with GProgrammer	5
3.1 Hardware Connection.....	5
3.2 SoC/MCU Selection.....	6
3.3 GR551x Series.....	8
3.3.1 Main Operational Interface.....	8
3.3.2 Connection Management.....	9
3.3.3 Firmware.....	11
3.3.3.1 Download Firmware.....	11
3.3.3.2 Action Order.....	13
3.3.4 Flash.....	14
3.3.4.1 Internal Flash.....	15
3.3.4.2 External Flash.....	19
3.3.5 Encrypt & Sign.....	22
3.3.5.1 eFuse Settings.....	22
3.3.5.2 Download.....	24
3.3.5.3 Encrypt & Sign.....	25
3.3.6 eFuse Layout.....	26
3.3.7 Chip Configuration.....	28
3.3.7.1 Init NVDS Area.....	29
3.3.7.2 Read All.....	29
3.3.7.3 Write.....	30
3.3.7.4 Add a User Parameter.....	31
3.3.7.5 Modify NVDS Parameters.....	32
3.3.7.6 Remove a User Parameter.....	33
3.3.7.7 Import and Export.....	34
3.3.8 Device Log.....	35
3.3.9 Command-line Programs.....	36
3.3.9.1 GR551x_console.exe.....	36
3.3.9.2 GR551x_encrypt_signature.exe.....	38
3.3.9.3 User-defined Windows Scripts.....	40
3.4 GMF03x Series.....	40
3.4.1 Main Operational Interface.....	40
3.4.2 Connection Management.....	41

3.4.3 Memory & Flash.....	44
3.4.3.1 Read/Write to Memory.....	44
3.4.3.2 Open File.....	45
3.4.4 Programming & Erasing.....	46
3.4.4.1 Flash Programming.....	47
3.4.4.2 Flash Erasing.....	48
3.4.4.3 Read/Write Protection for Flash Memory.....	49
3.4.5 User Option Bytes.....	49
3.5 Help.....	50

1 Introduction

GProgrammer supports programming of flash memories on Goodix SoCs and MCUs. It runs on Windows only.

GProgrammer provides diverse features:

- Flexible SoC/MCU selections:
 - GR551x series
 - GMF03x series
- Connection via SWD and UART
- Automatic GUI adaptation specific to SoC/MCU type
- Functionalities for GR551x series:
 - 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
- Functionalities for GMF03x series:
 - Reading from/Writing to SRAM and flash memory
 - Reading out BIN and HEX files
 - Programming and erasing of flash memory
 - Read/Write protection for flash memory
 - Configuring User Option Bytes

[Figure 1-1](#) shows the Graphical User Interface (GUI) of GProgrammer (for GR551x series).

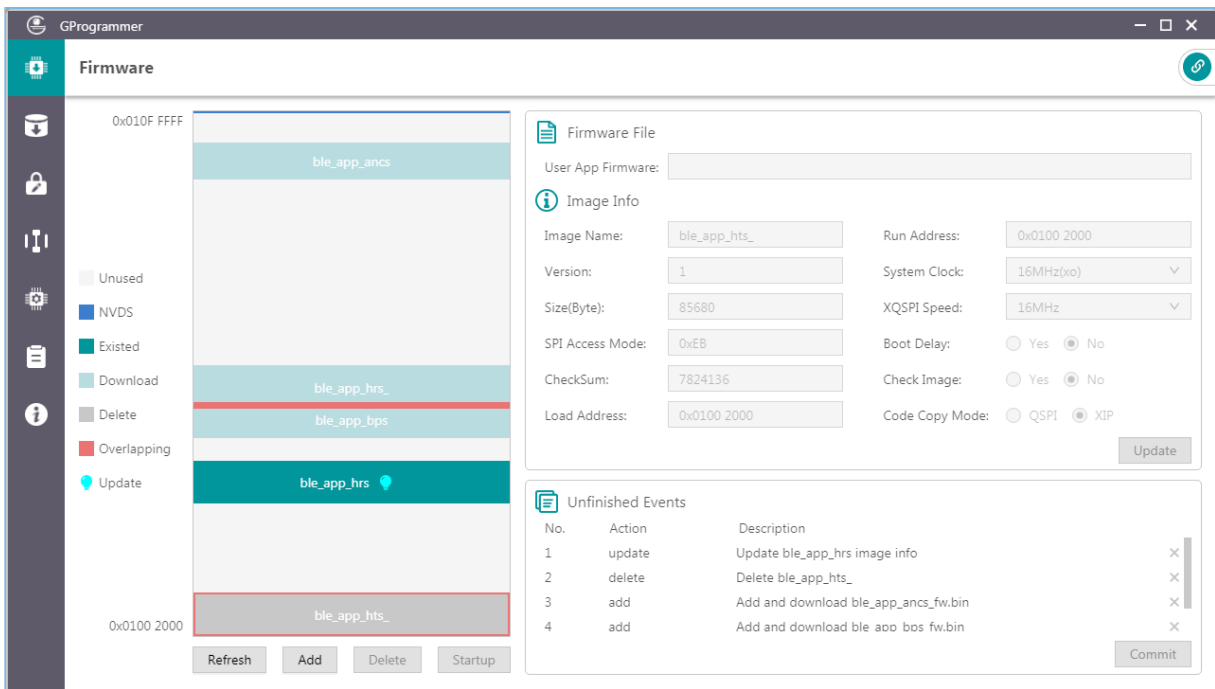


Figure 1-1 GProgrammer GUI (for GR551x series)

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 1.2.14.exe*.

Users can follow the steps below when installing GProgrammer:

1. Double-click *GProgrammer Setup 1.2.14.exe*, and follow the steps in the **GProgrammer Setup** wizard (see [Figure 2-1](#)).

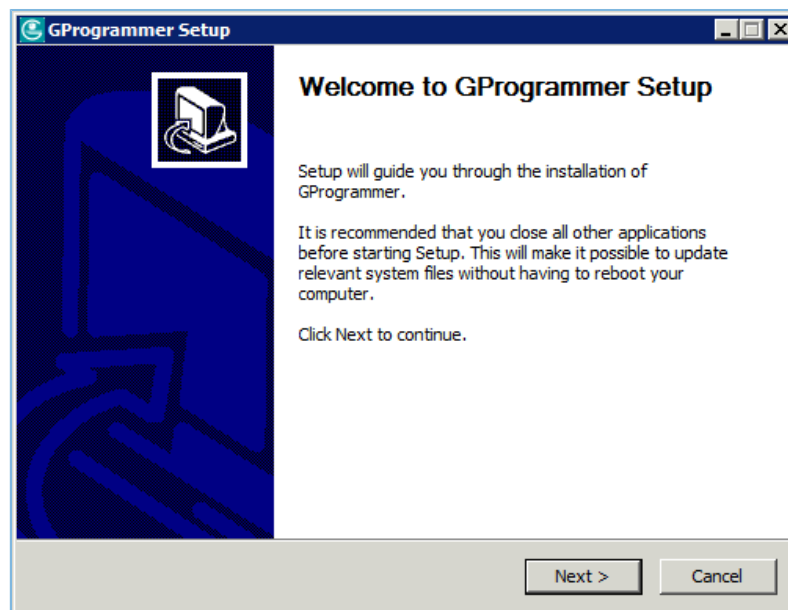


Figure 2-1 GProgrammer Setup installation wizard

2. After installing GProgrammer, you are prompted to install J-Link on demand. See [Figure 2-2](#).

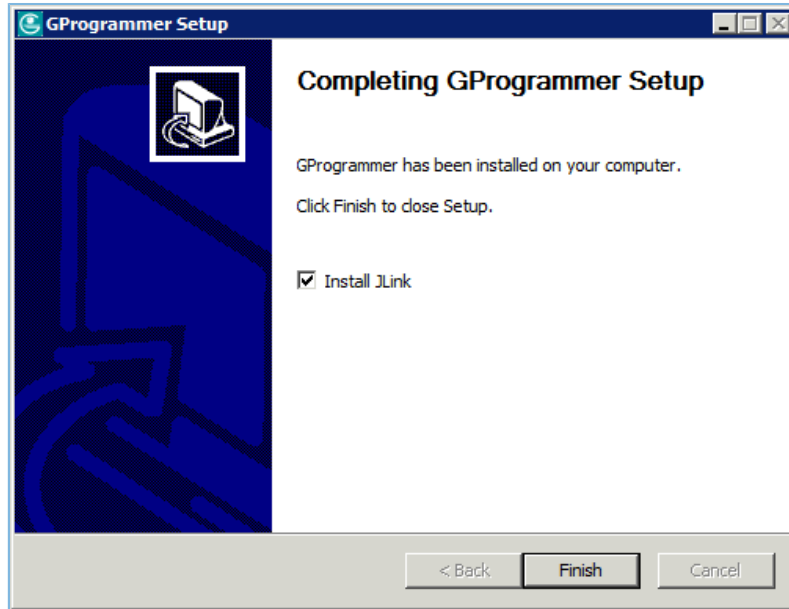


Figure 2-2 Prompt to install J-Link

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/MCU pins of the target board through Dupont wire cables.

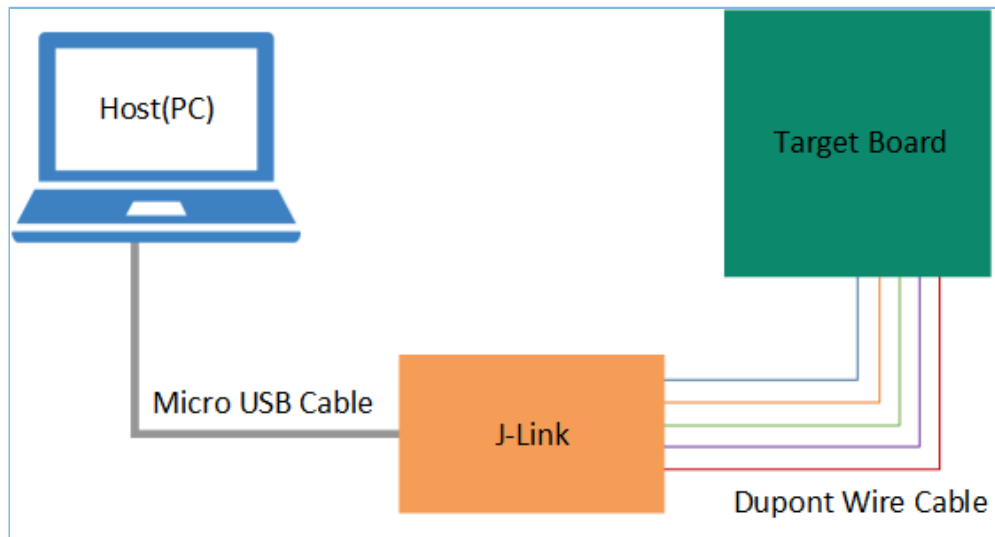


Figure 3-1 Host-target-board connection in SWD mode

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

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

J-Link Emulator Pin	GR551x SoC Pin	GMF03x MCU Pin
VCC	VCC	VCC
GND	GND	GND
SWDIO	GPIO_1	PA13
SWCLK	GPIO_0	PA14

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/MCU pins of the target board through Dupont wire cables.

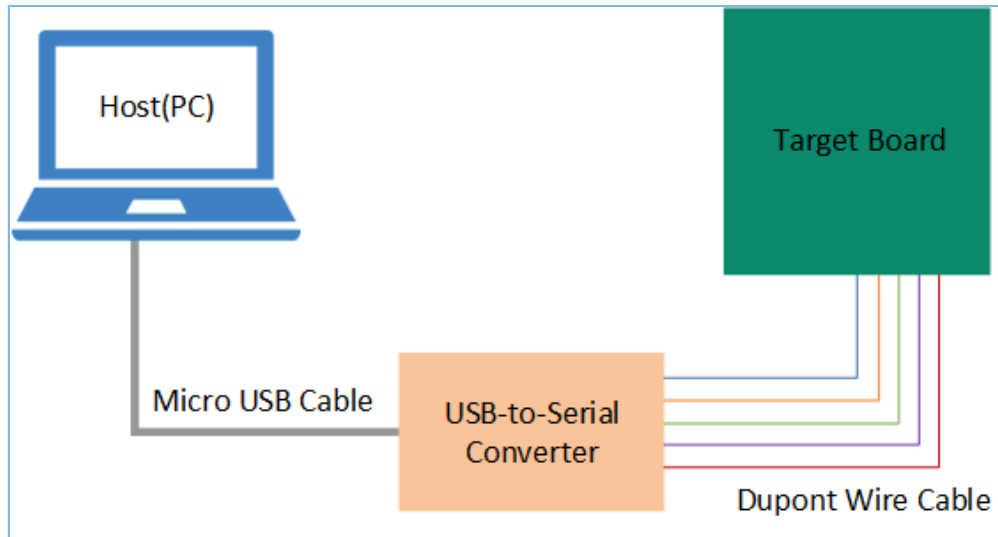


Figure 3-2 Host-target-board connection in UART mode

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

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

USB-to-Serial Converter Pin	GR551x SoC Pin	GMF03x MCU Pin
VCC	VCC	VCC
GND	GND	GND
TX	GPIO_1	PA15
RX	GPIO_0	PA14
RTS	CHIP_EN	N/A

In UART mode, the BOOT0 pin of a GMF03x MCU shall be pulled up to high level. For details, refer to “Chapter 2 Bootloader Activation and Hardware Connection” in *GMF03x Bootloader Description*.

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/MCU Selection

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

Tip:

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

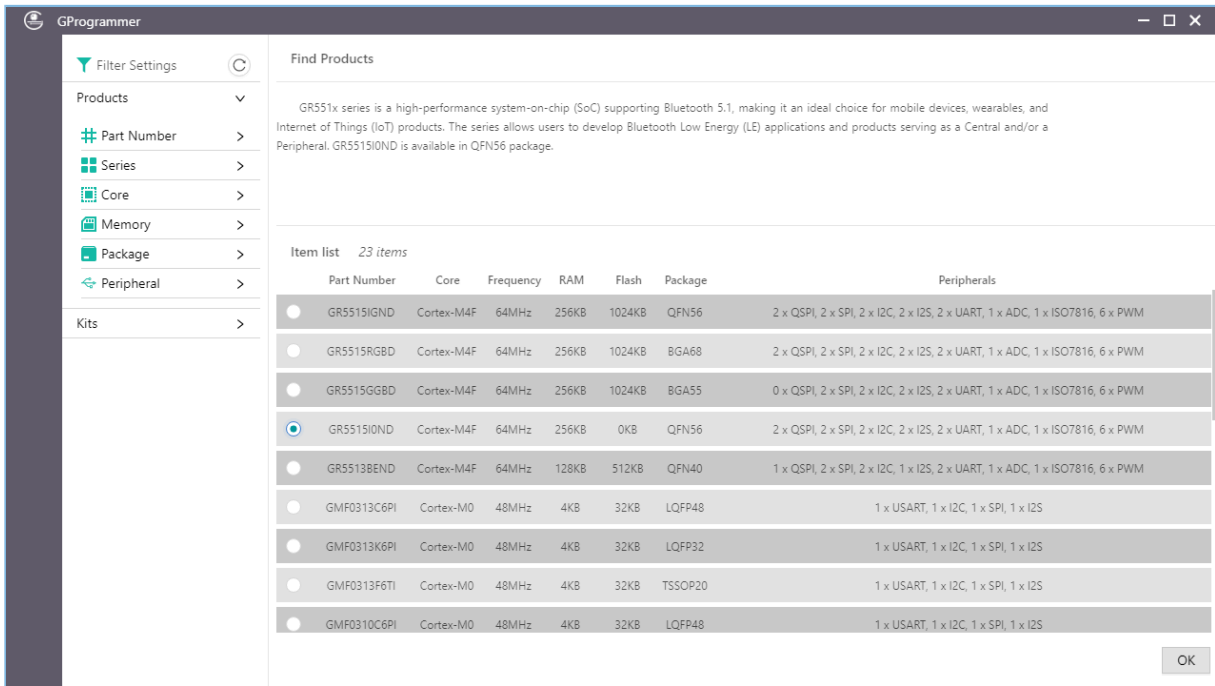


Figure 3-3 SoC/MCU selection interface

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

Under **Series**, you can quickly switch between GR551x SoCs and GMF03x MCUs.

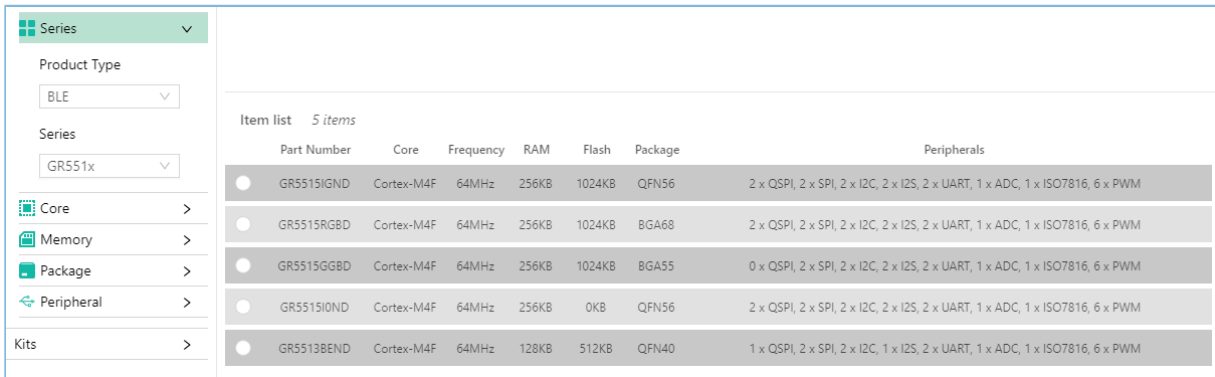


Figure 3-4 Selecting GR551x series

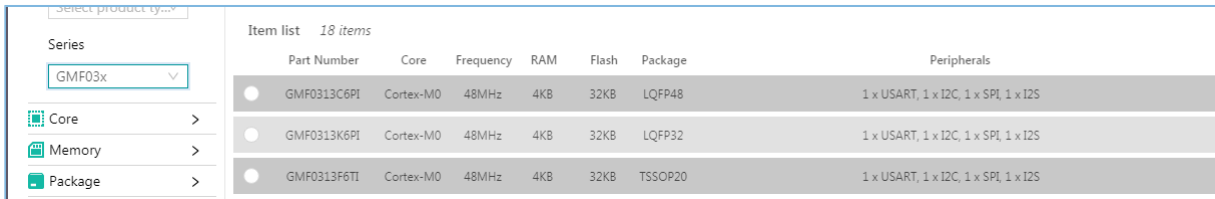


Figure 3-5 Selecting GMF03x series

To date, GProgrammer has supported both GR551x and GMF03x series:

- Five GR551x models such as GR5515IGND and GR5515RGBD
- 18 GMF03x models such as GMF0313K6PI, GMF0313C6PI, and GMF0313F6TI

Tip:

Peripherals listed on the SoC/MCU selection interface are only part of the peripherals of a GR551x SoC or GMF03x MCU. For details of all peripherals, see *GR551x Datasheet* or *GMF03x Datasheet*.

3.3 GR551x Series

This section elaborates on functional modules of GProgrammer for GR551x series.

3.3.1 Main Operational Interface

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

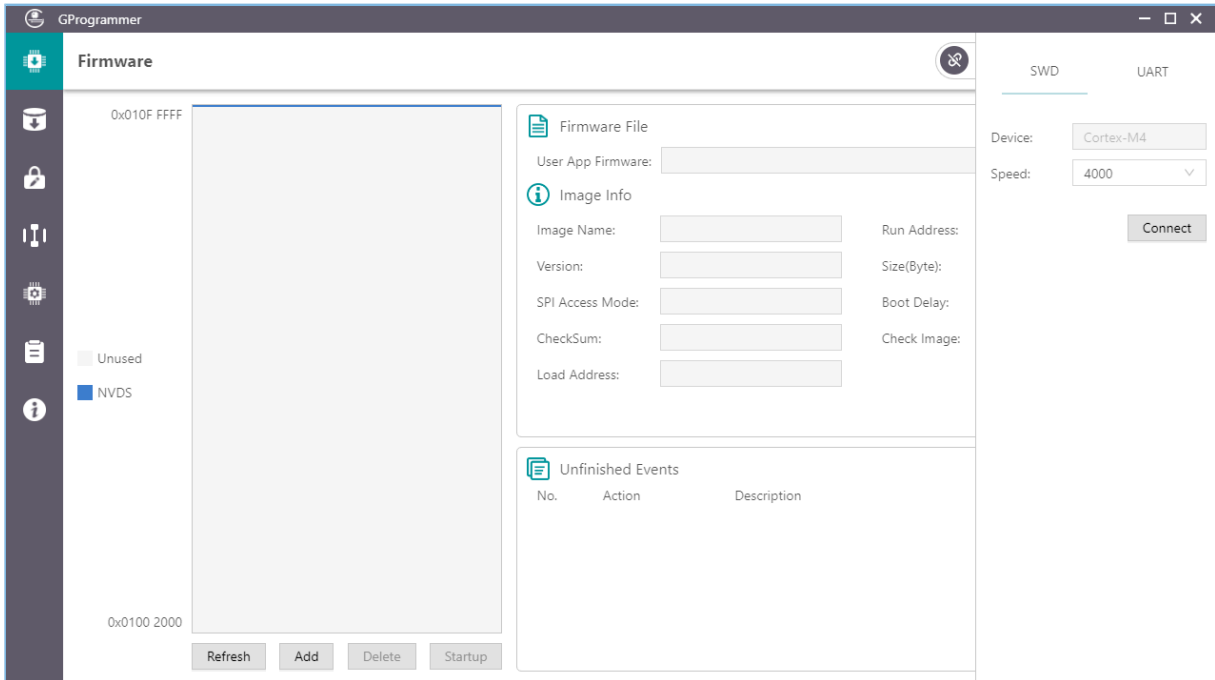









Figure 3-6 GProgrammer GUI (for GR551x series)


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.

3.3.2 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.

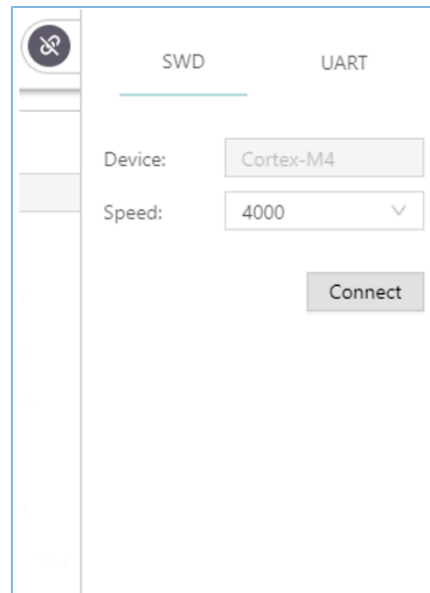


Figure 3-7 GProgrammer SWD connection

- 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.

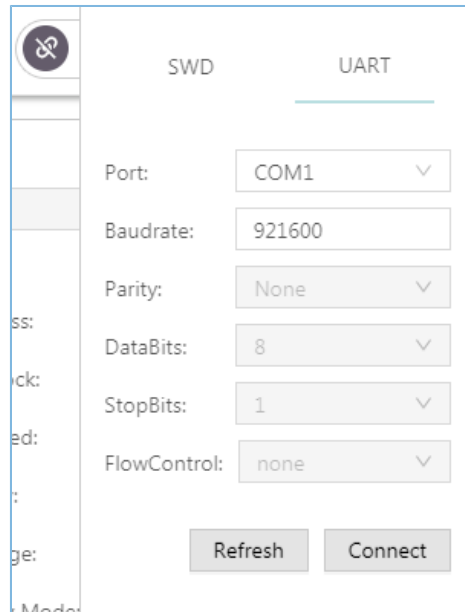





Figure 3-8 GProgrammer UART connection

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**.

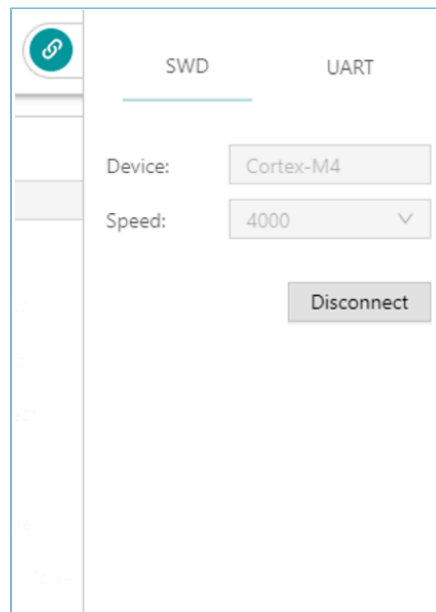



Figure 3-9 Clicking **Disconnect** on GProgrammer

3.3.3 Firmware

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

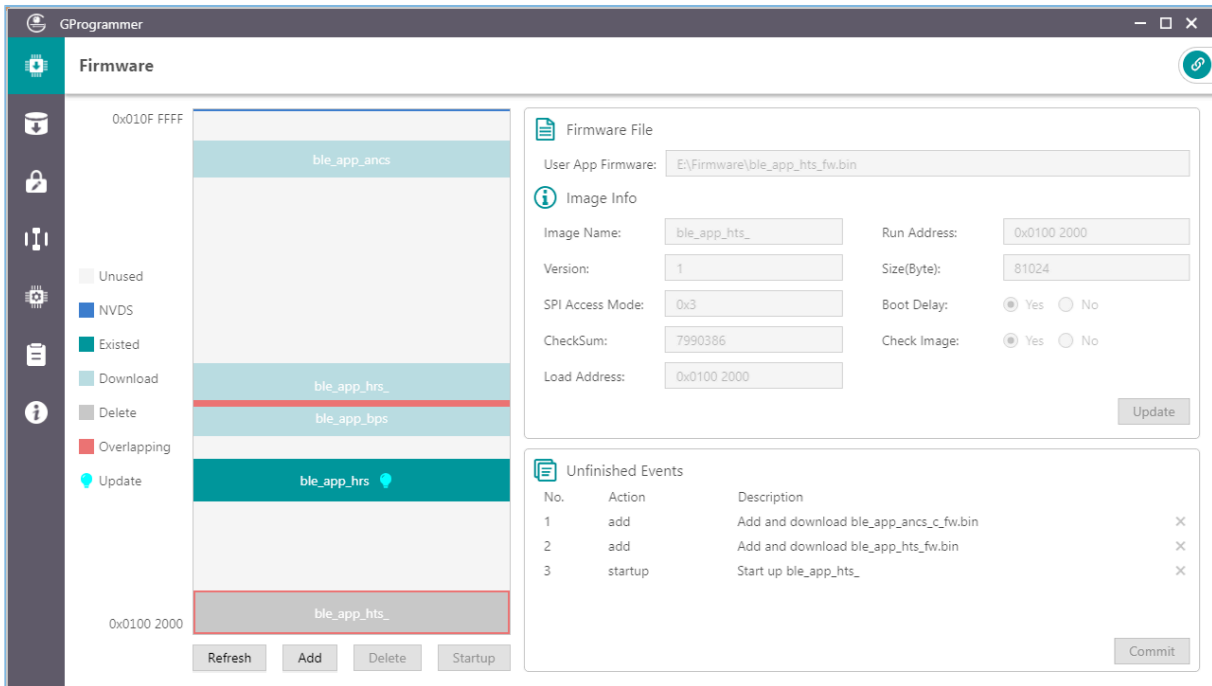


Figure 3-10 GProgrammer **Firmware** interface

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

3.3.3.1 Download Firmware

GProgrammer graphically displays the flash memory space layout occupied by firmware (see [Figure 3-11](#)), which helps you easily learn the flash occupation status.

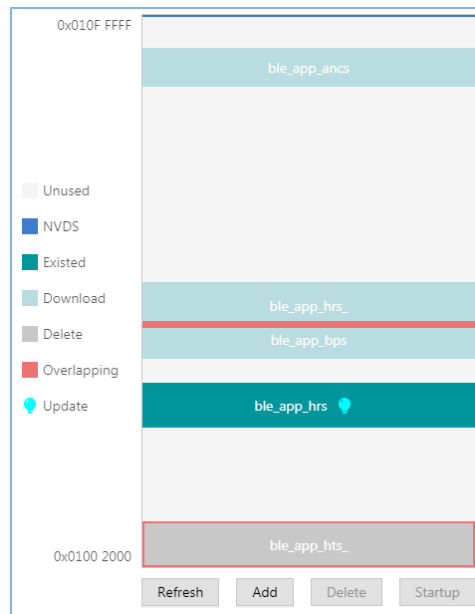










Figure 3-11 Flash firmware layout

- : flash space to which data can be downloaded
- : default NVDS area to which firmware cannot be downloaded
- : space for storing to-be-deleted firmware. Example: ble_app_ancs
- : space for storing to-be-downloaded firmware. Example: ble_app_hrs
- : space for storing downloaded firmware in flash memories. Example: ble_app_bps
- : space overlapped by two pieces of firmware. Examples: ble_app_T3u and ble_app_hrs

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.

3.3.3.2 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 3-12](#).

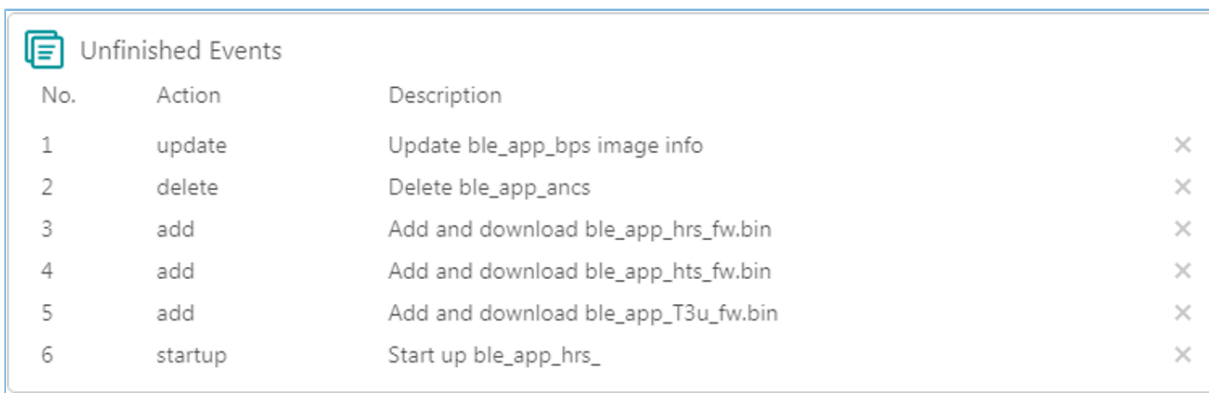
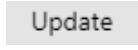




Figure 3-12 Action order


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		Click Add to add a local firmware file to GProgrammer.
Refresh firmware		Click Refresh to obtain the information of firmware downloaded in the flash memories of a target board. Unexecuted actions 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		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 .
Start execution		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		<p>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.</p> <p>Execute update actions, and all parameters involved are locked. No editing is allowed. If modification is required, withdraw the previous update action.</p>

 **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.

3.3.4 Flash

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

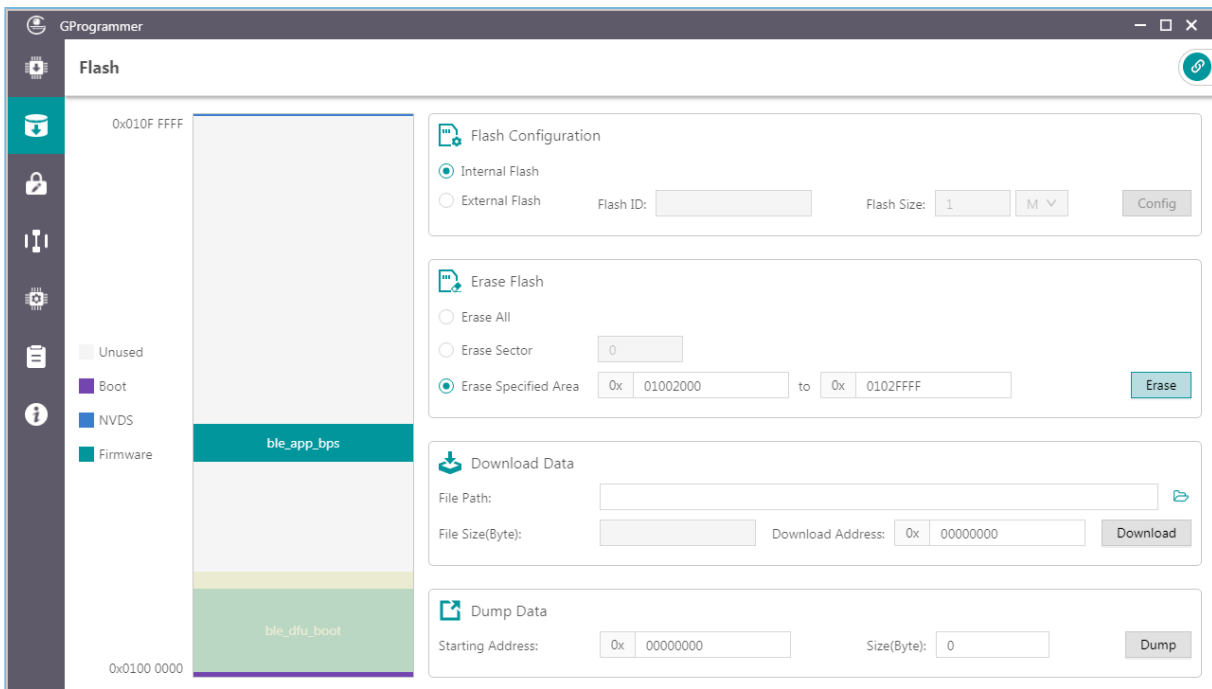







Figure 3-13 GProgrammer **Flash** interface

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

3.3.4.1 Internal Flash

3.3.4.1.1 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.

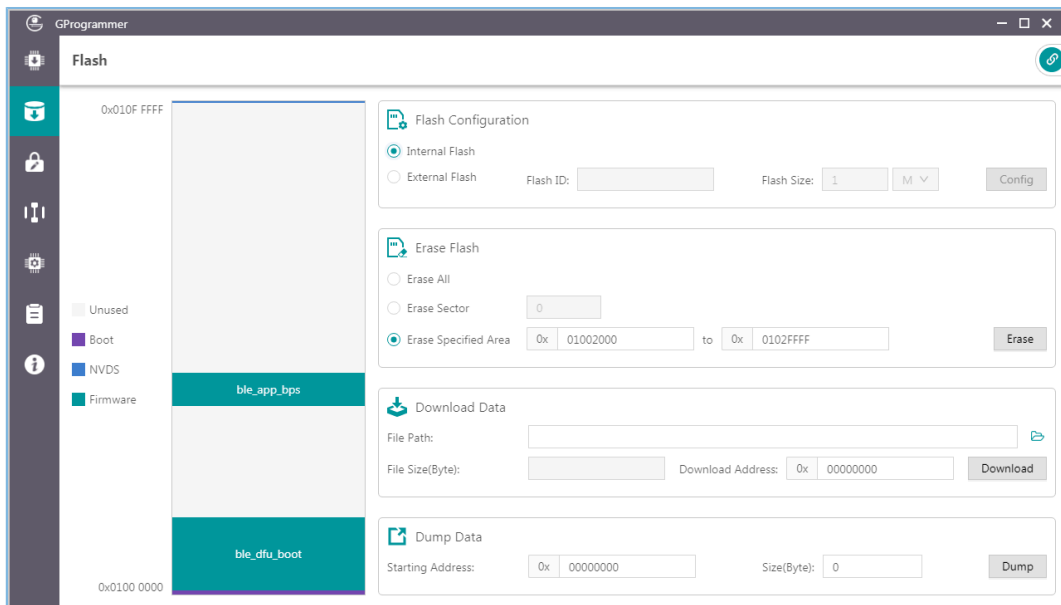


Figure 3-14 Selecting Internal Flash

3.3.4.1.2 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.

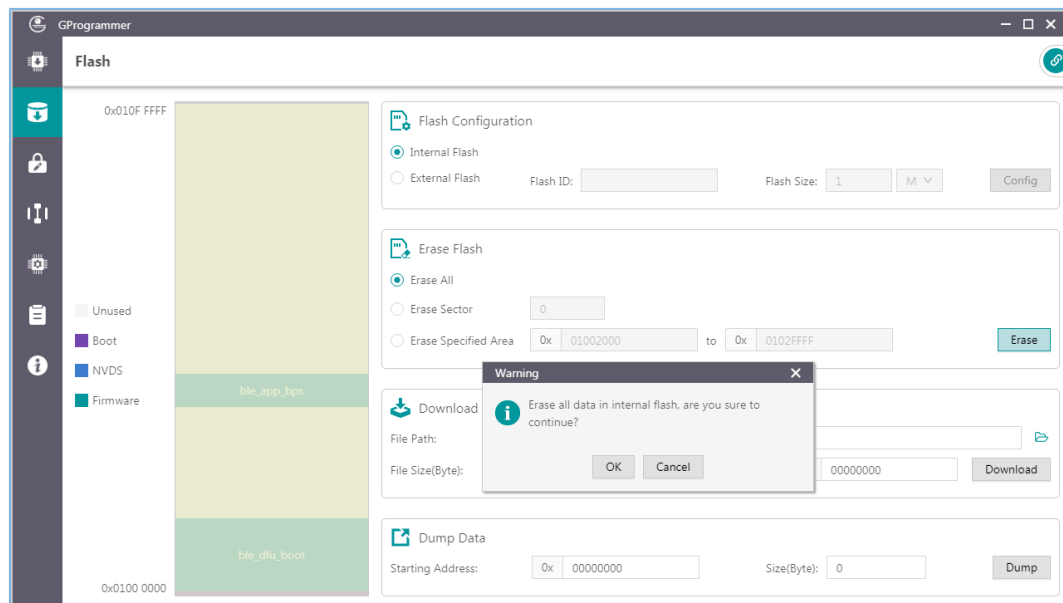


Figure 3-15 Erase All on GProgrammer

- **Erase Sector**

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

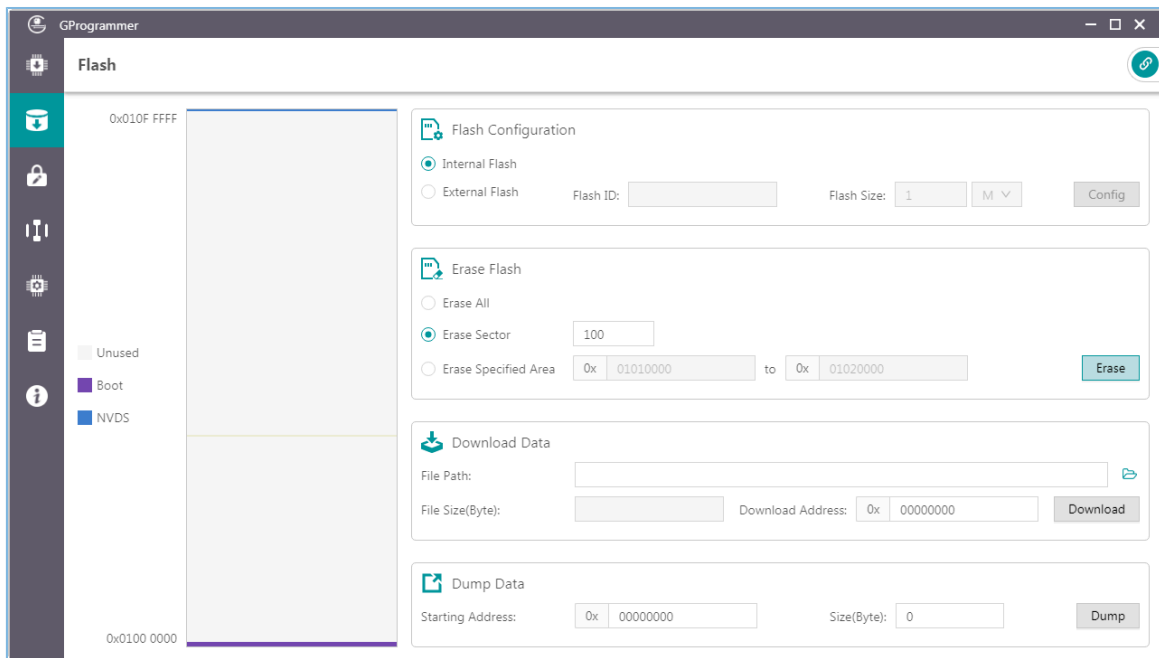


Figure 3-16 Erase Sector on GProgrammer

- **Erase Specified Area**

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

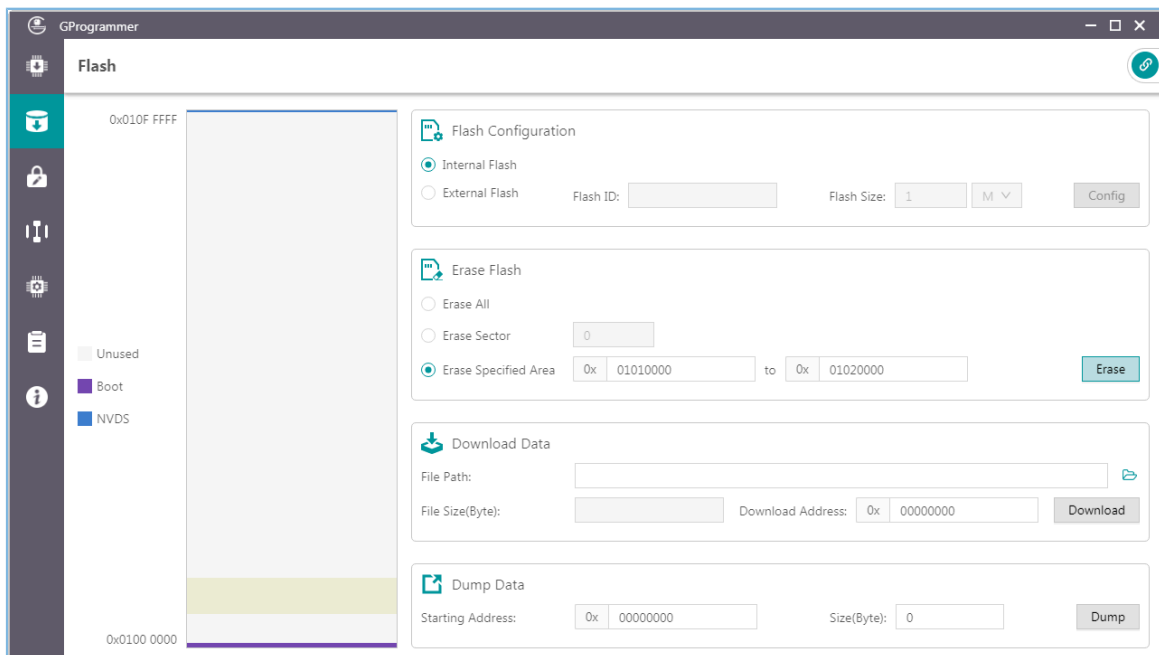


Figure 3-17 Erase Specified Area on GProgrammer

3.3.4.1.3 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**.

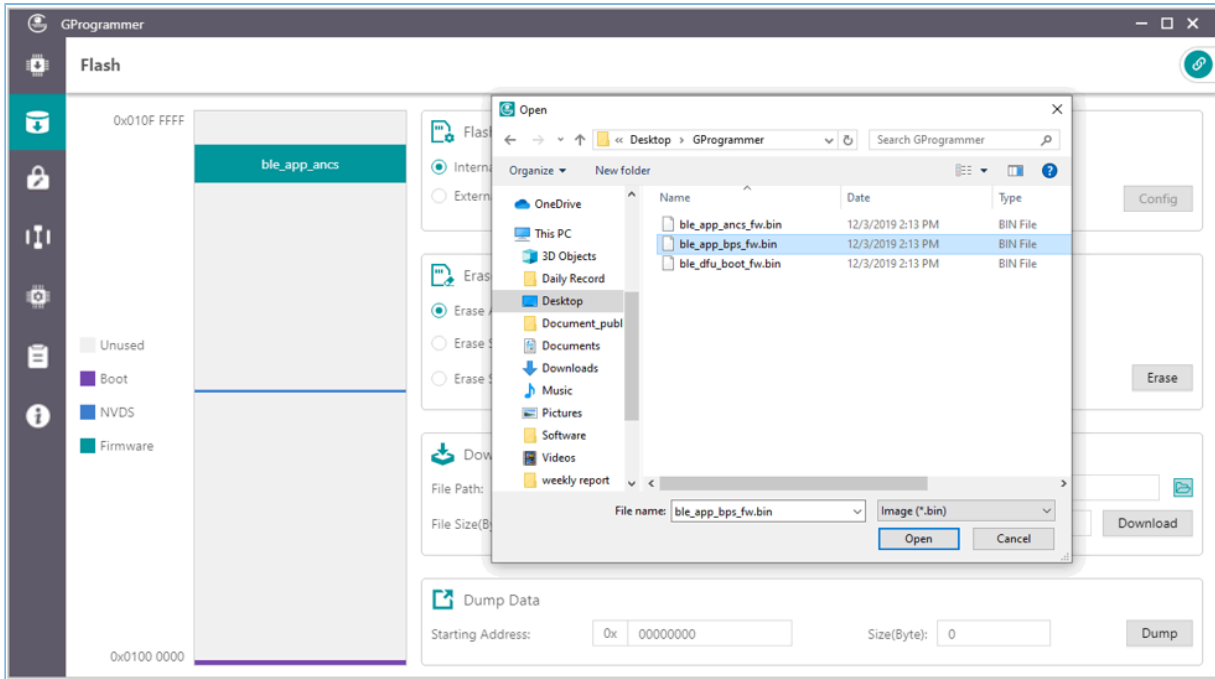


Figure 3-18 Viewing and selecting a data file to be downloaded

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

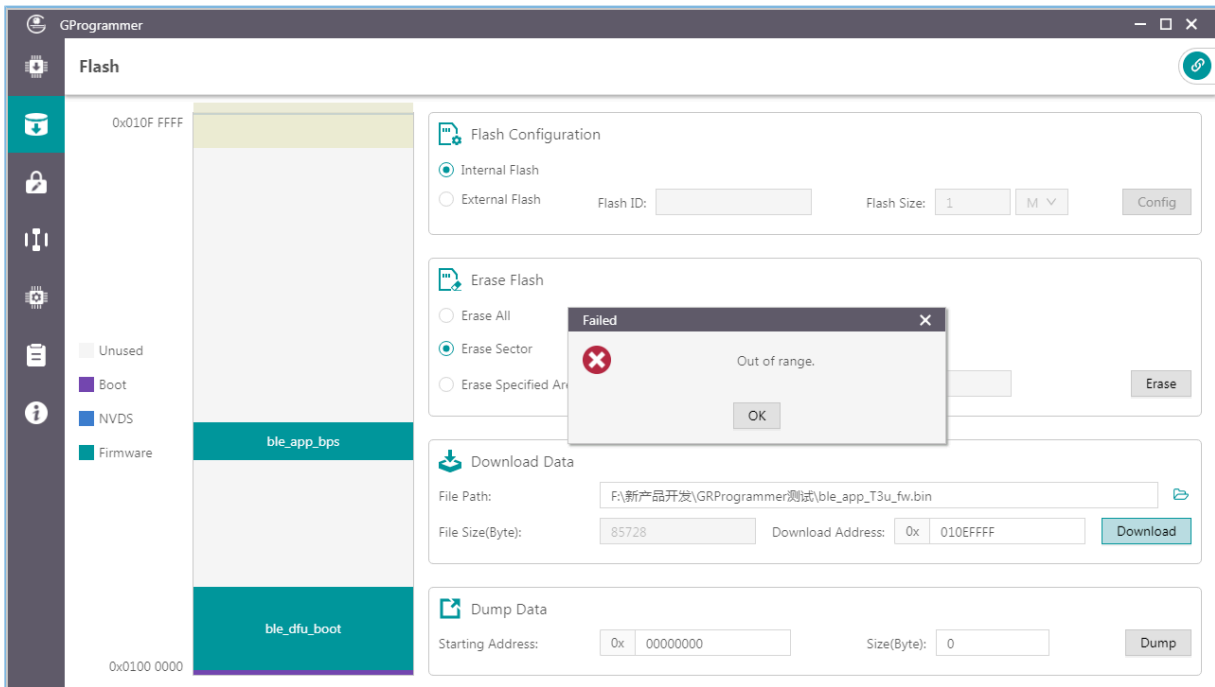


Figure 3-19 Flash overflow error

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.

3.3.4.1.4 Dump Data

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

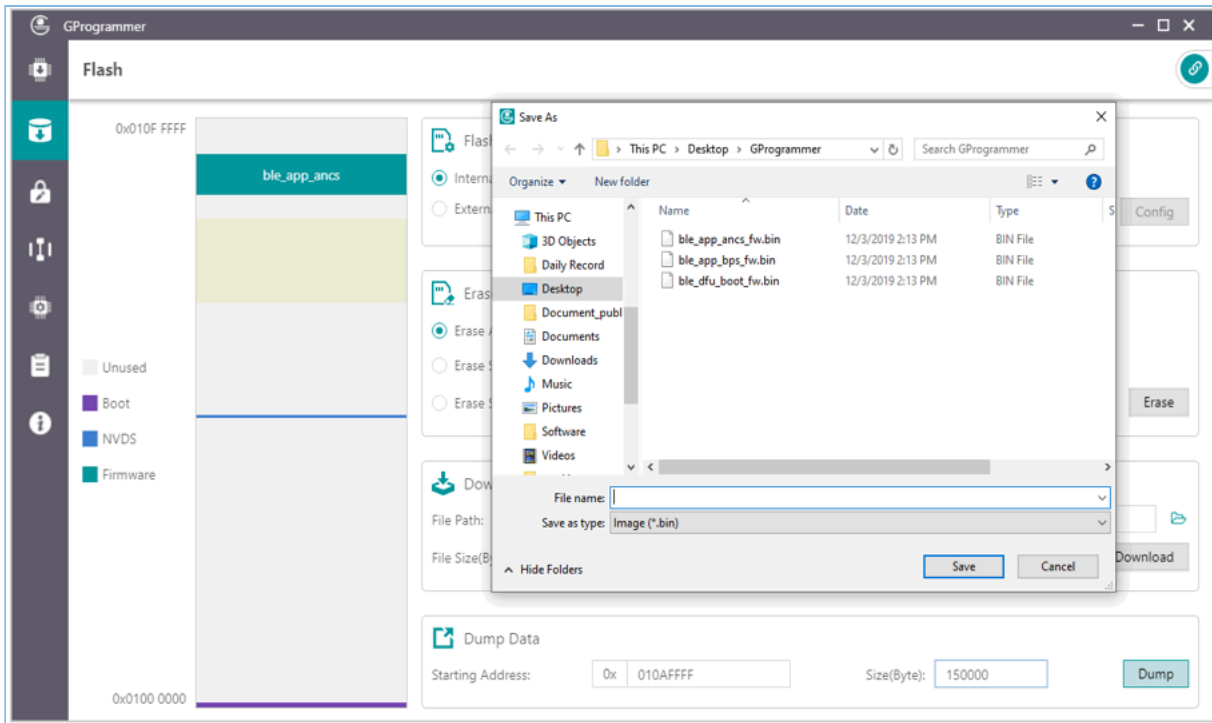


Figure 3-20 Dump Data on GProgrammer

3.3.4.2 External Flash

3.3.4.2.1 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.

The screenshot shows the 'External Flash Configuration' dialog box with the 'SPI' radio button selected. The configuration table is as follows:

	GPIO Type	GPIO PIN	PIN MUX
CS:	NORMAL	GPIO_0	MUX_0
CLK:	NORMAL	GPIO_3	MUX_2
MOSI:	NORMAL	GPIO_4	MUX_2
MISO:	NORMAL	GPIO_5	MUX_2

Buttons: Apply, Cancel

Figure 3-21 SPI configurations

The screenshot shows the 'External Flash Configuration' dialog box with the 'QSPI0' radio button selected. The configuration table is as follows:

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

Buttons: Apply, Cancel

Figure 3-22 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.

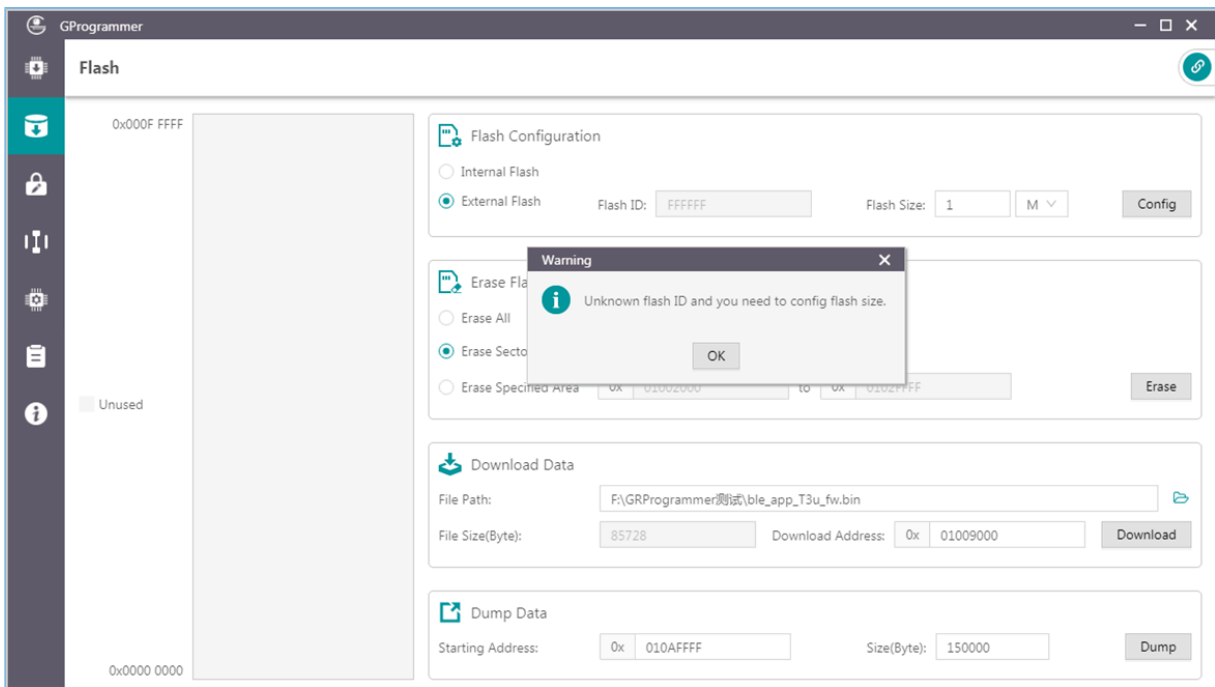


Figure 3-23 Unknown flash ID

3.3.4.2.2 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.

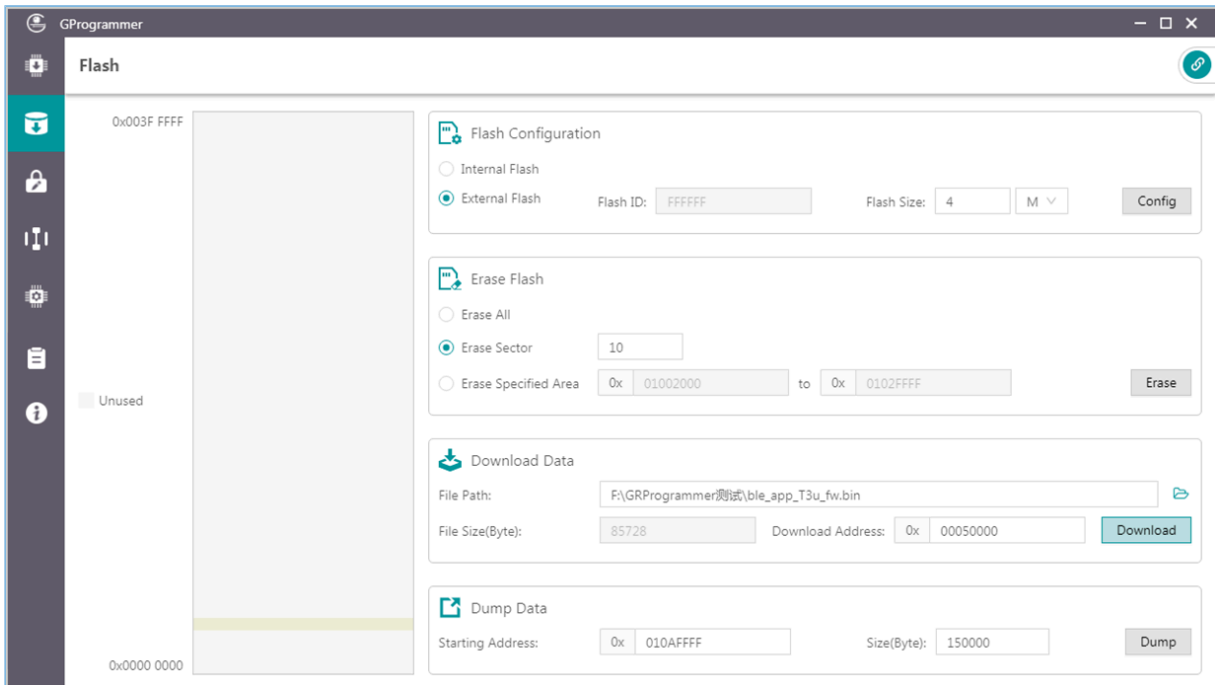


Figure 3-24 Download Data to external flash on GProgrammer

Tip:

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

3.3.5 Encrypt & Sign

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

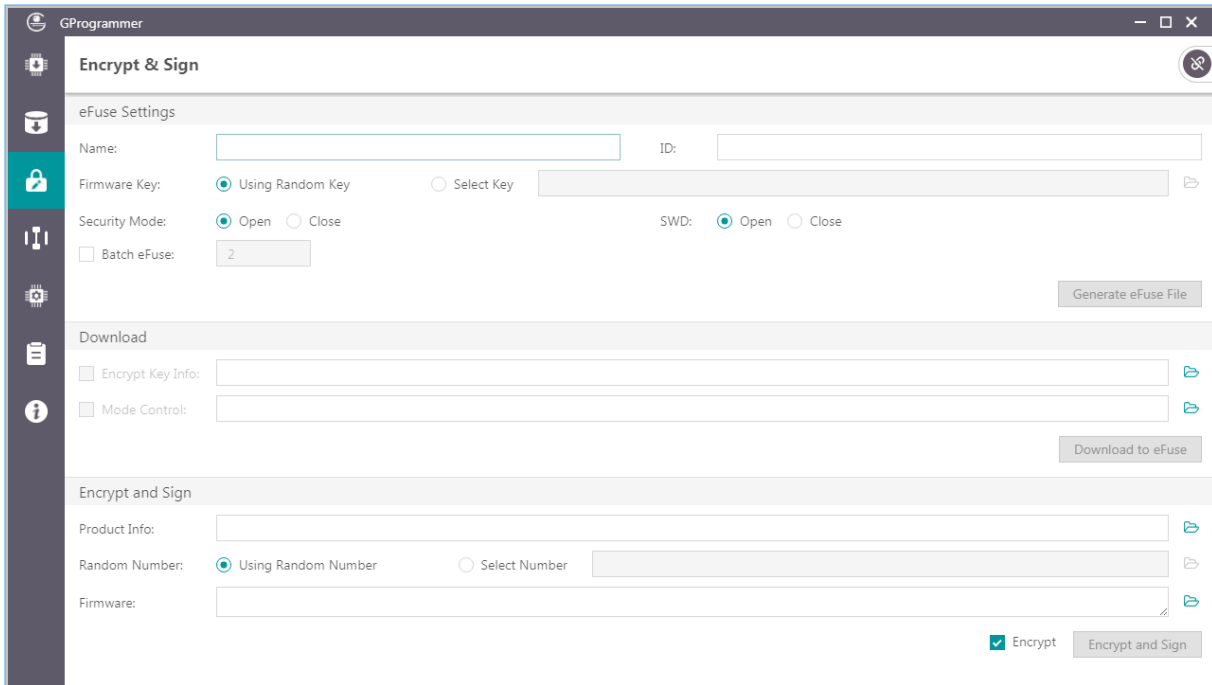


Figure 3-25 GProgrammer **Encrypt & Sign** interface

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.

3.3.5.1 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.

The screenshot shows the 'eFuse Settings' window. It contains the following fields and options:

- Name:** test
- ID:** 1
- Firmware Key:** Using Random Key, Select Key. The path 'F:\GR551X update\tools\GProgrammer\test\firmware.key' is entered in the adjacent text box.
- Security Mode:** Open, Close
- SWD:** Open, Close
- Batch eFuse:** Batch eFuse: 3
- Only Data Key is different between batch eFuse files.
- Generate eFuse File** button

Figure 3-26 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:

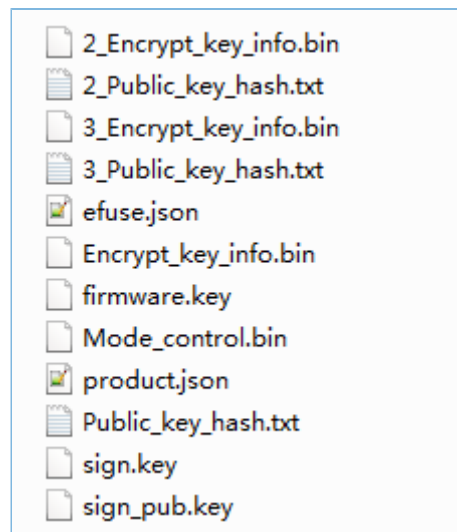


Figure 3-27 Generated files

- *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*, *2_Public_key_hash.txt*, and *3_Public_key_hash.txt*: public key hashes 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.

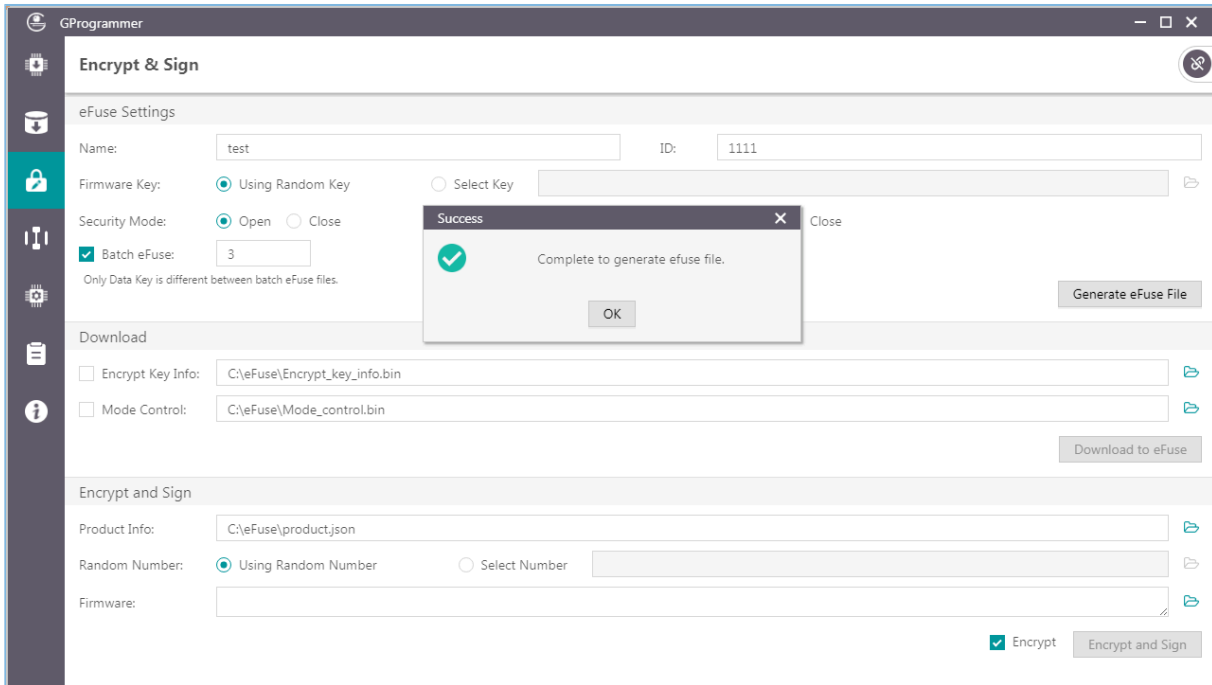


Figure 3-28 Paths for automatically loaded files

3.3.5.2 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.

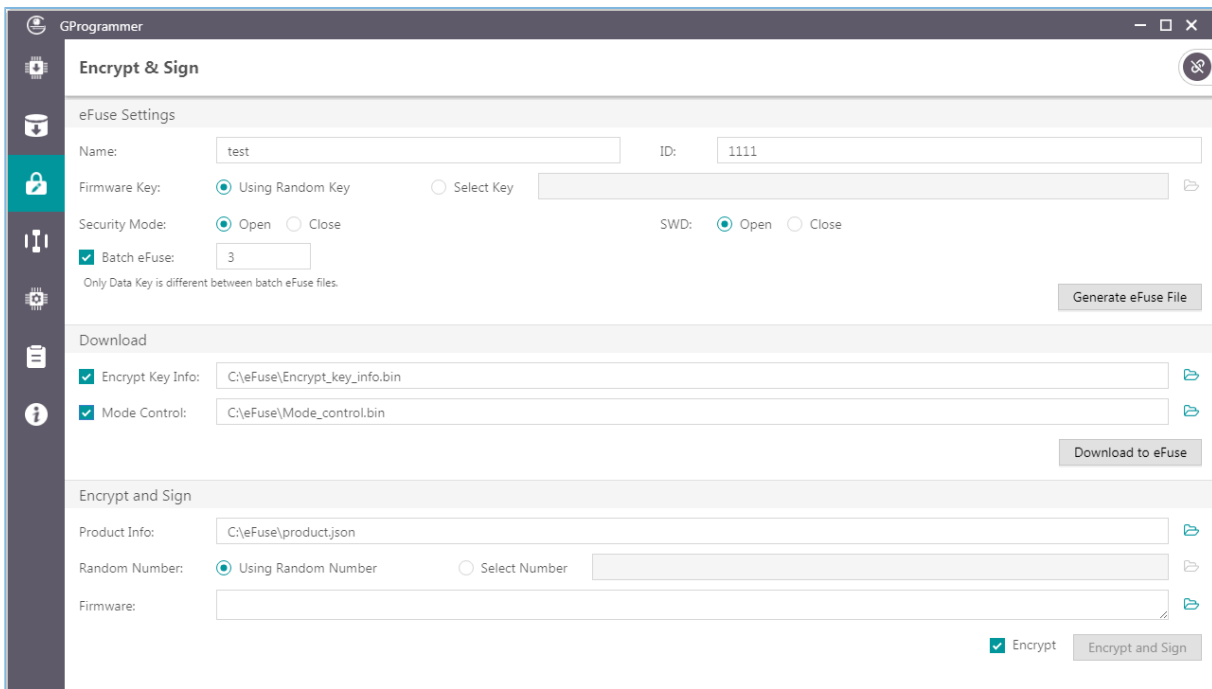


Figure 3-29 Downloading files to eFuse

Note:

eFuse information cannot be repeatedly downloaded to firmware.

3.3.5.3 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 3-30](#).

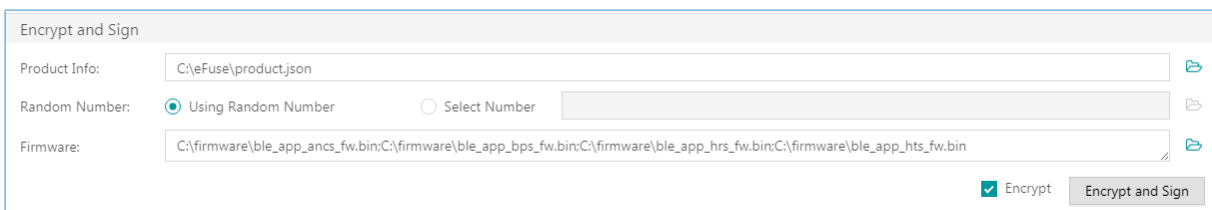


Figure 3-30 Adding more than one firmware file

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:

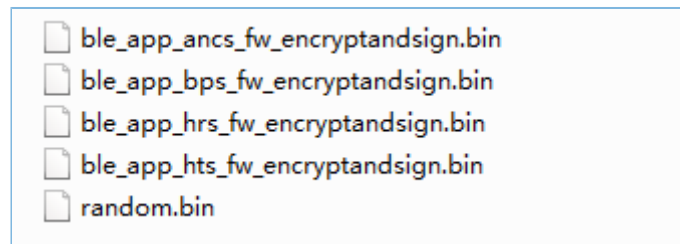


Figure 3-31 GProgrammer-generated files after encryption and signing

Files after being signed are listed below:

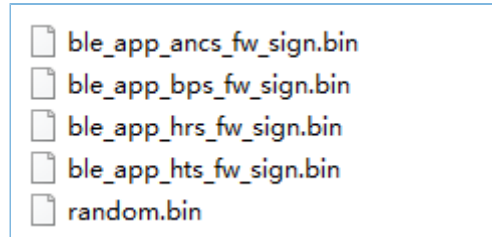


Figure 3-32 GProgrammer-generated files after signing

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.

3.3.6 eFuse Layout

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

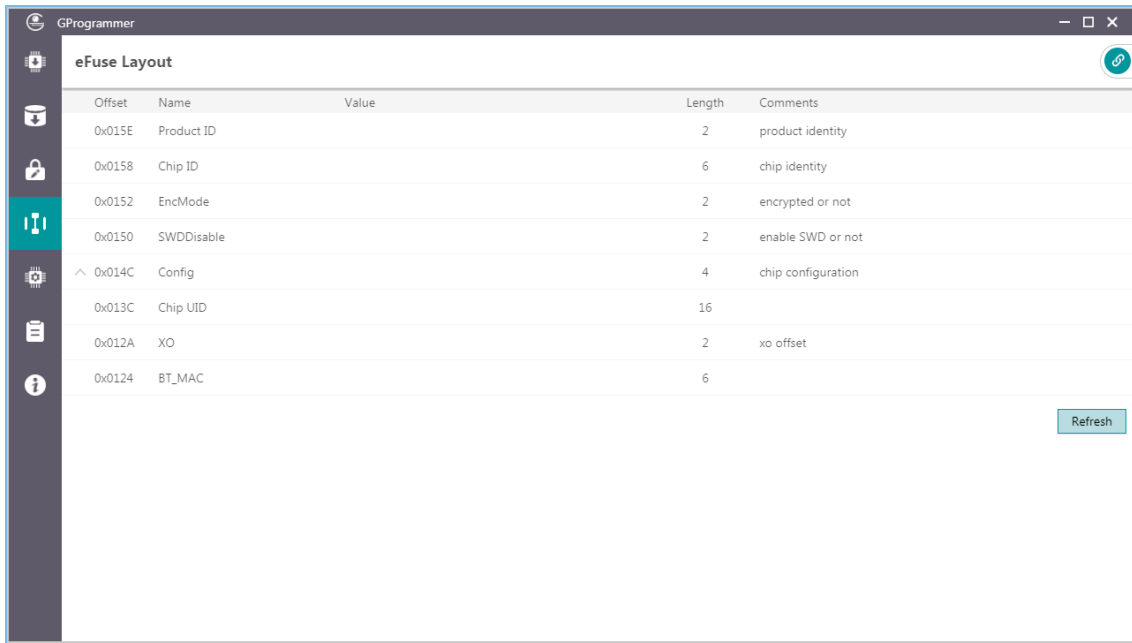


Figure 3-33 eFuse Layout interface

GProgrammer presents users with eFuse layout information: **Offset**, **Value**, **Length**, and **Comments** of fields including **Product ID**, **Chip ID**, **EncMode**, **SWDDisable**, and **Config**. Among them, the **Config** field contains multiple bit fields. Click **Refresh** to obtain the values of all fields or bit fields.

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

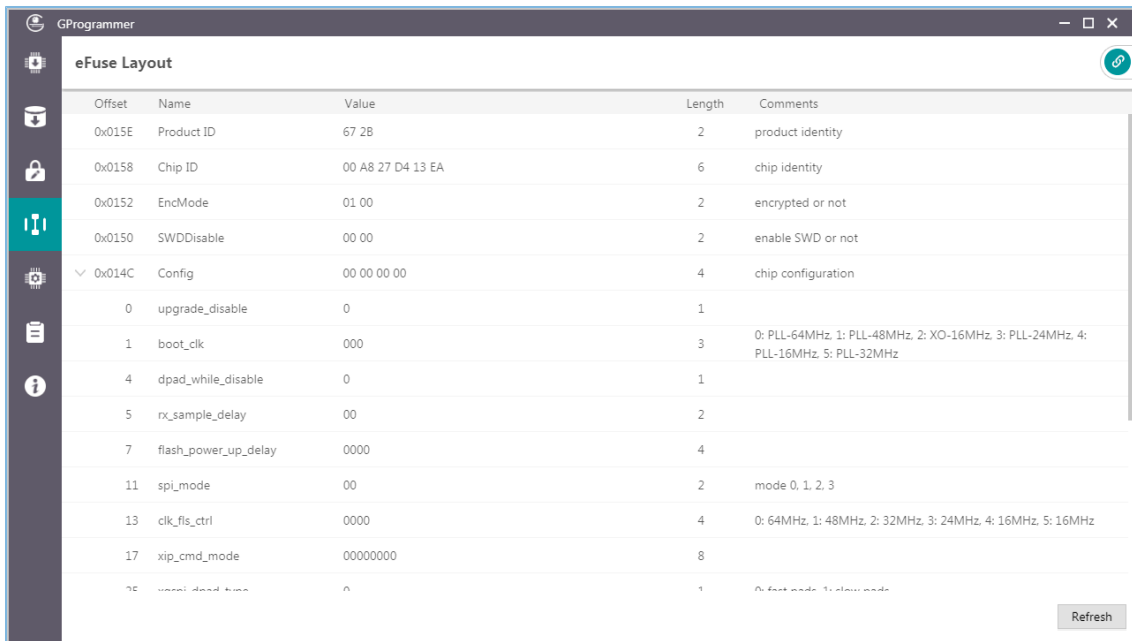


Figure 3-34 Expanded Offset

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.

3.3.7 Chip Configuration

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

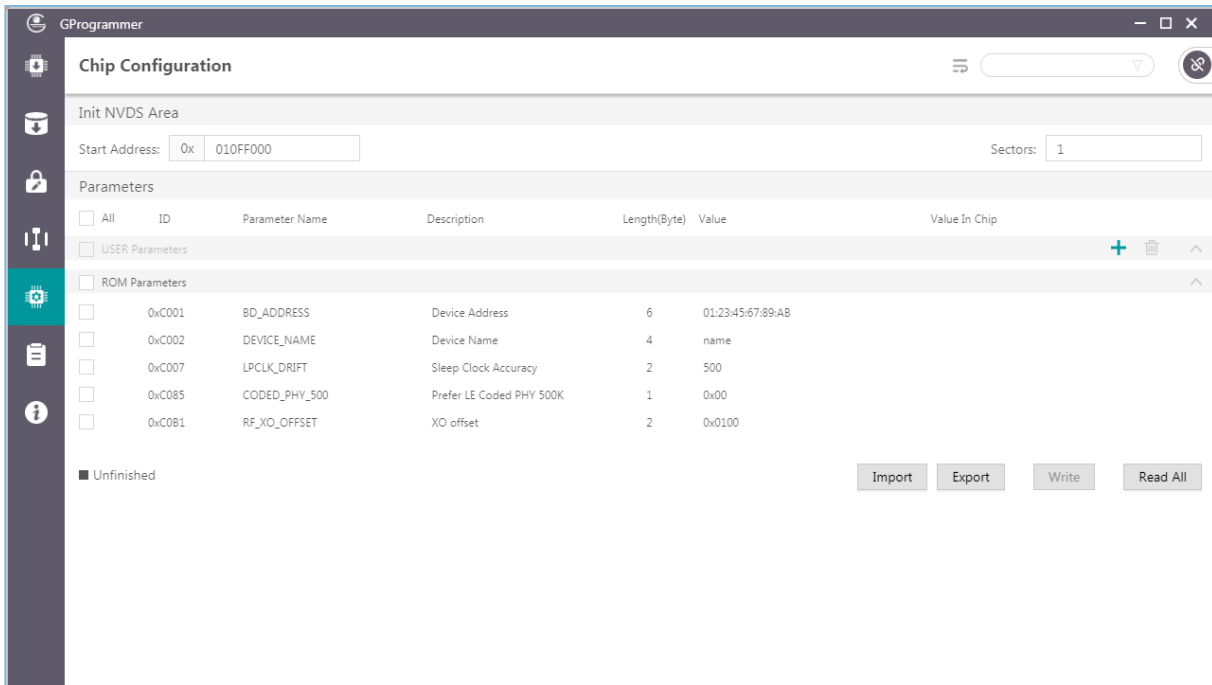


Figure 3-35 GProgrammer **Chip Configuration** interface

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

3.3.7.1 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 3-36 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.

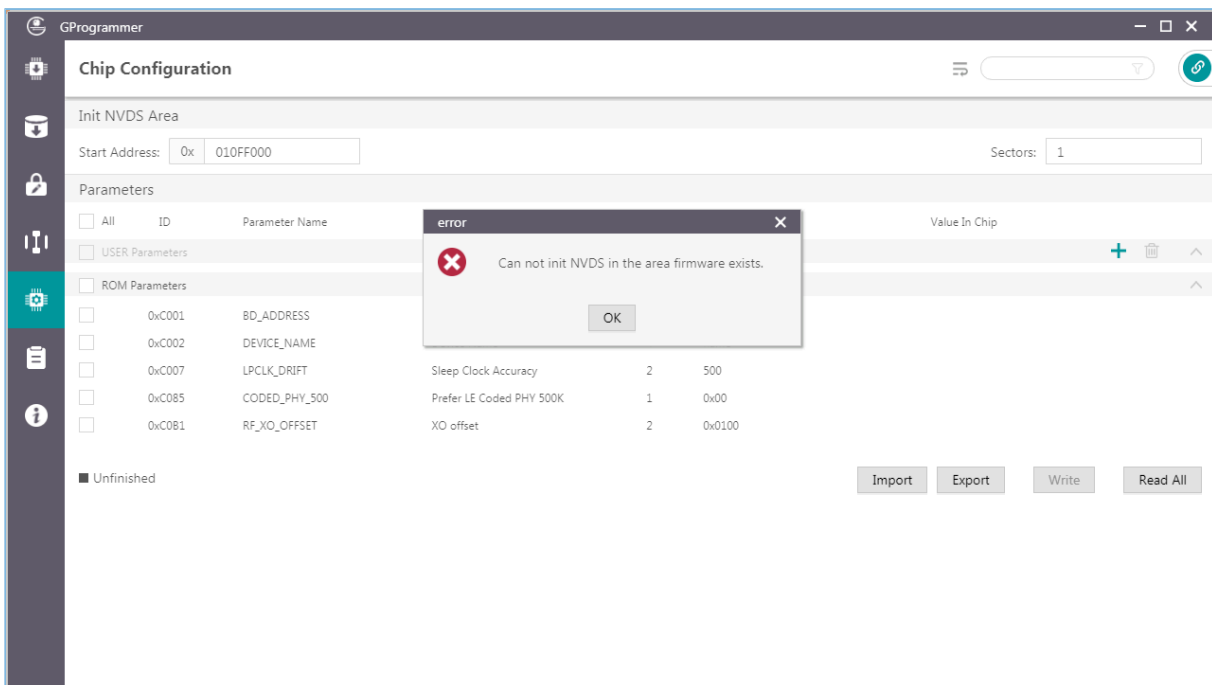


Figure 3-37 NVDS initialization failure

3.3.7.2 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 3-38](#)) or ones that have been listed in the NVDS area but with a different parameter length (example: 0x4001 in [Figure 3-38](#)).
- **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 3-38](#))
- **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 3-38](#))

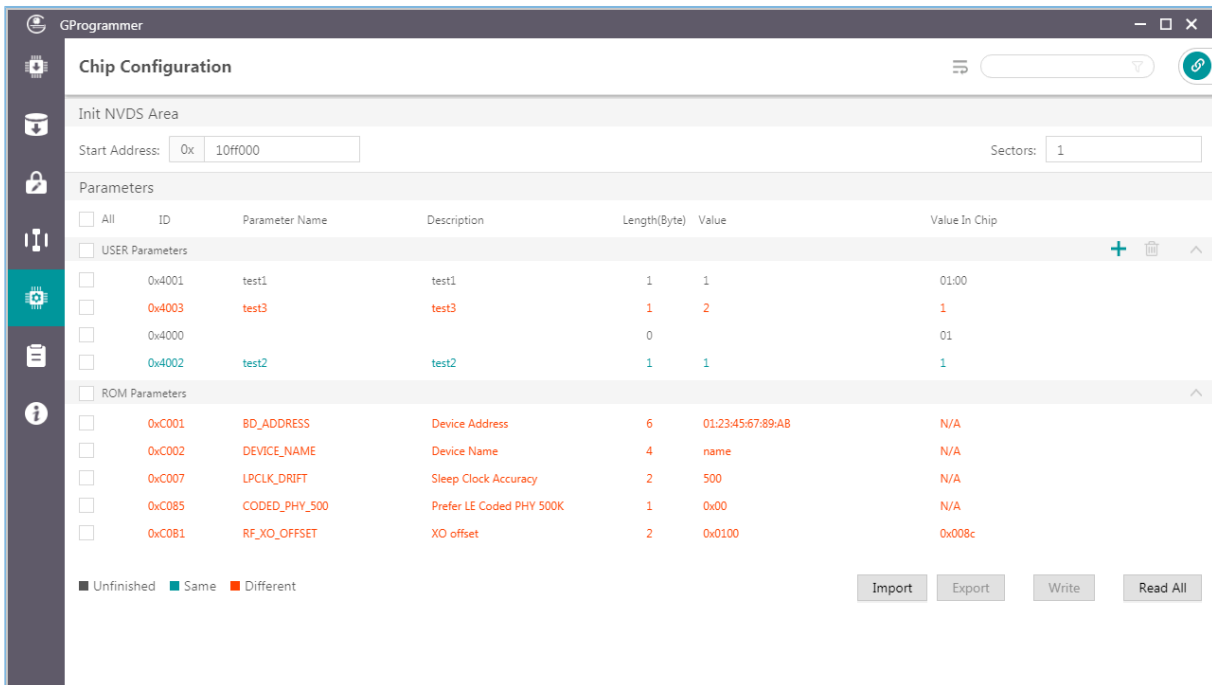


Figure 3-38 Read All interface

3.3.7.3 Write

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

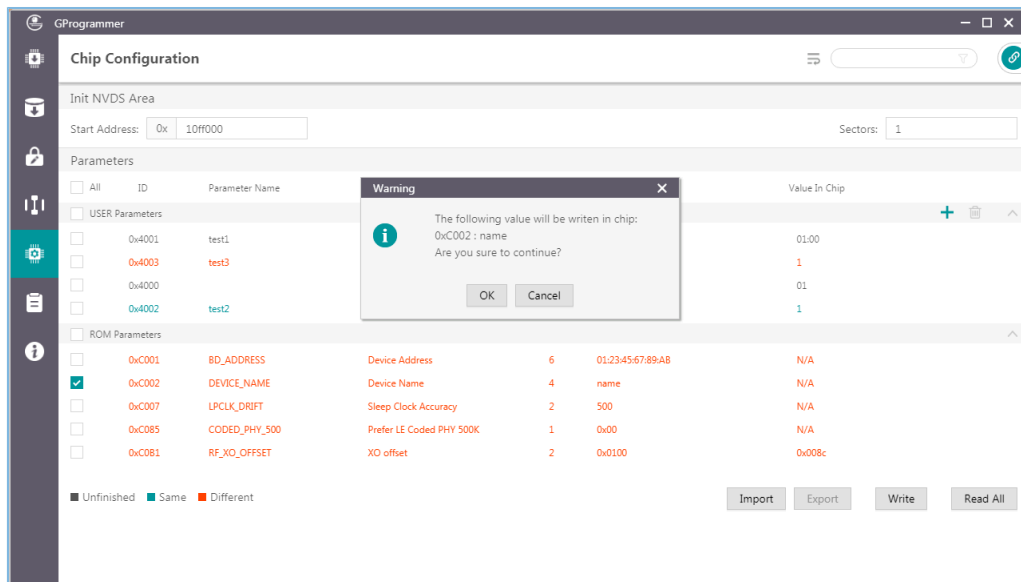


Figure 3-39 Write parameters to NVDS

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.

3.3.7.4 Add a User Parameter

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

- Click **+** to open the **Add USER Parameter** window.
- Specify the **ID**, **Parameter Name**, **Description**, **Type**, **Length(Byte)**, **Value**, and data presentation format (**dec** or **hex**).

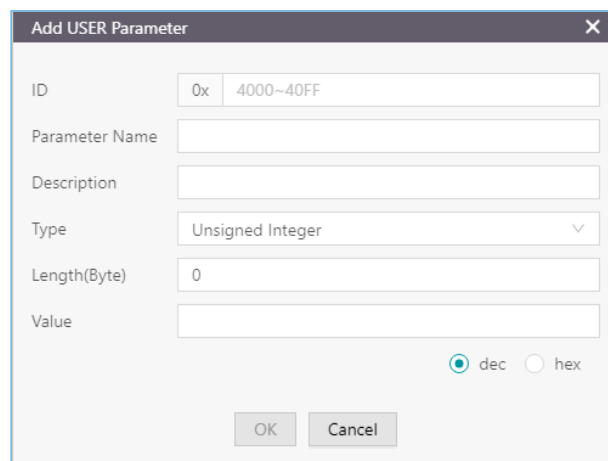


Figure 3-40 Adding a user parameter to NVDS

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 3-41](#).
- 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.

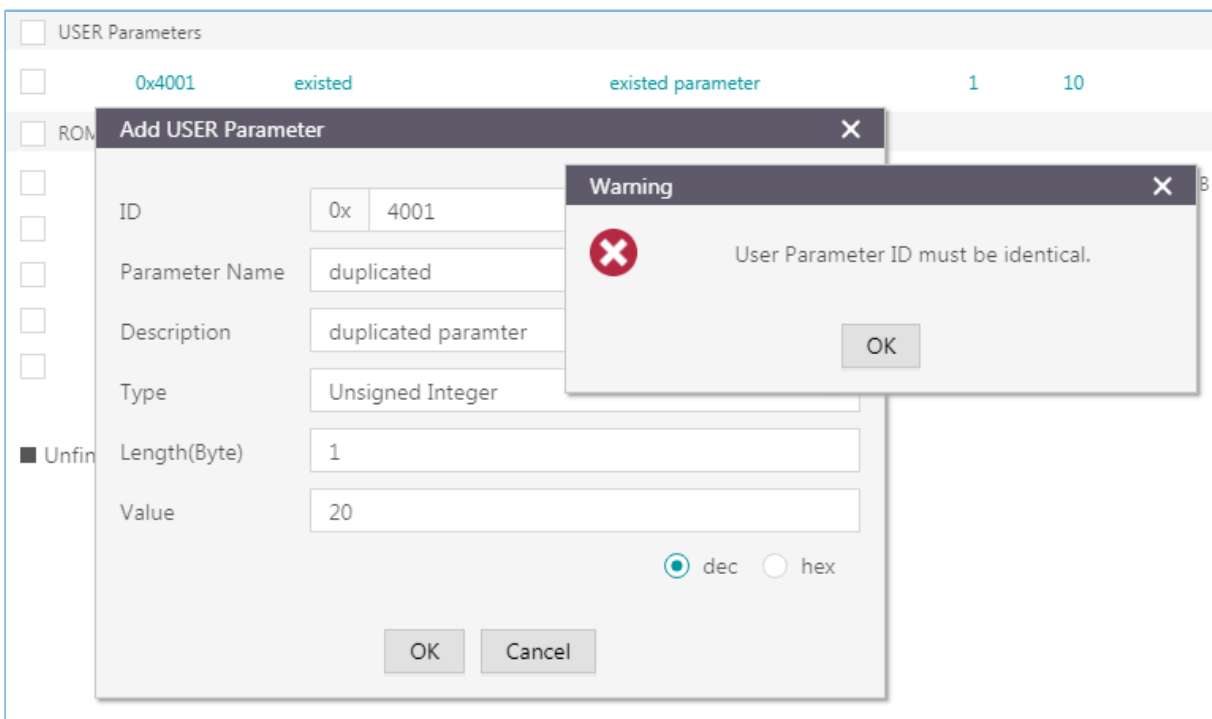


Figure 3-41 Failure to add a user parameter due to an identical parameter ID

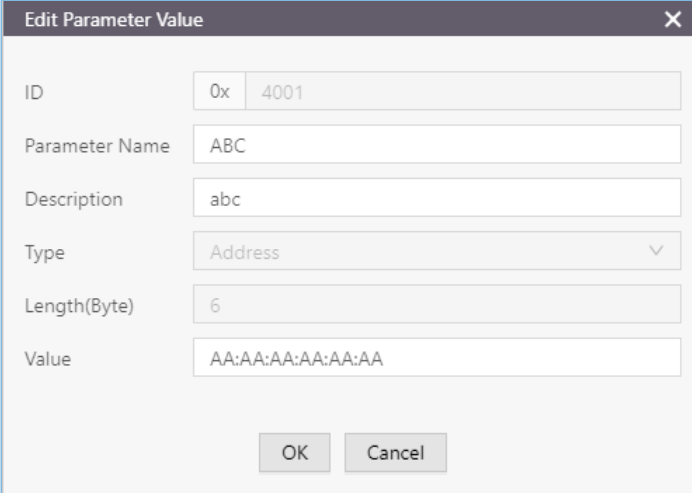
3.3.7.5 Modify NVDS Parameters

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.



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

Figure 3-42 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.

3.3.7.6 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.

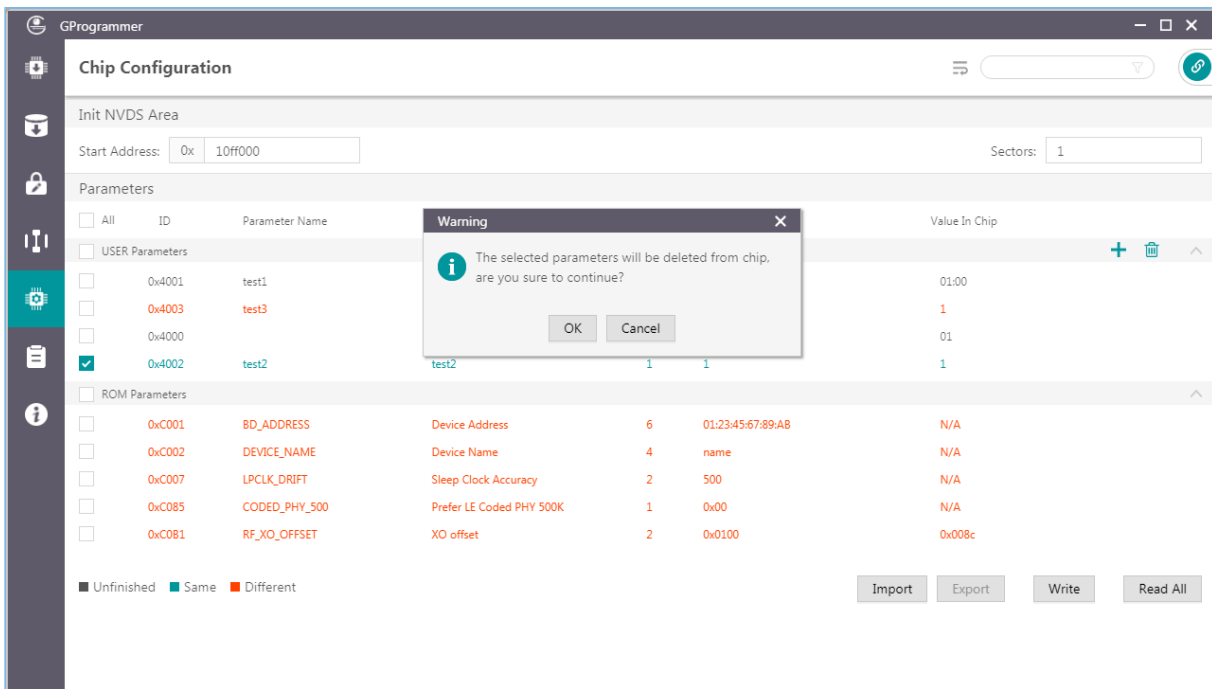


Figure 3-43 Removing a parameter

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).

3.3.7.7 Import and Export

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

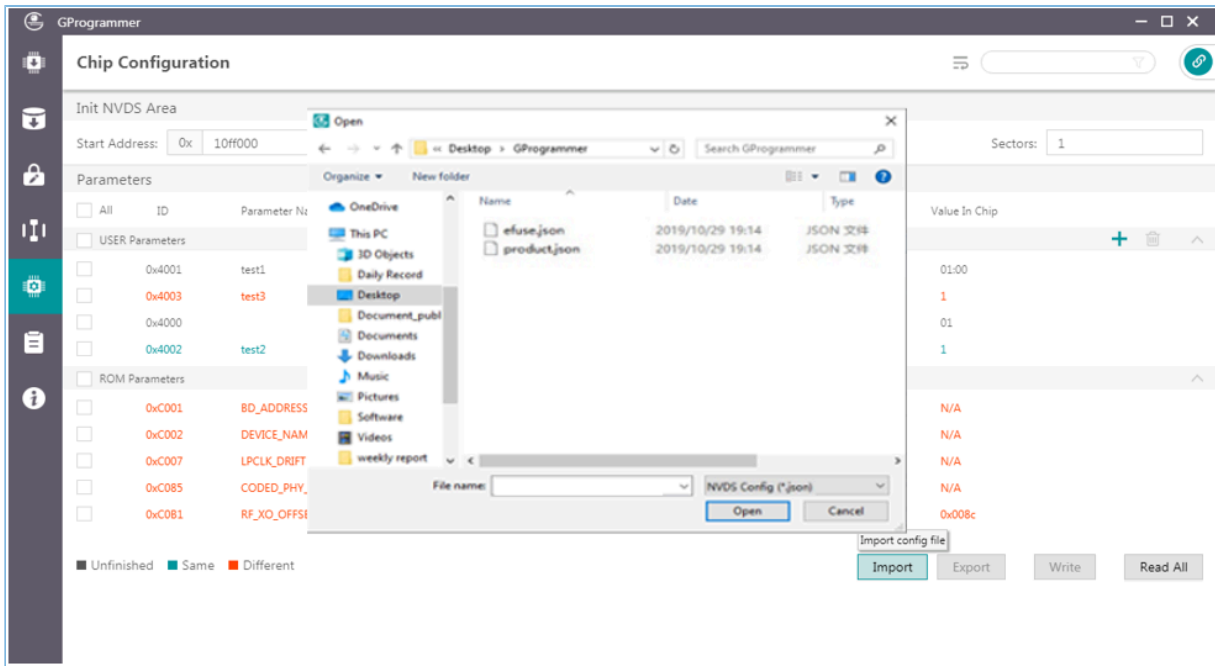



Figure 3-44 Importing local JSON configuration files to GProgrammer

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.

3.3.8 Device Log

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

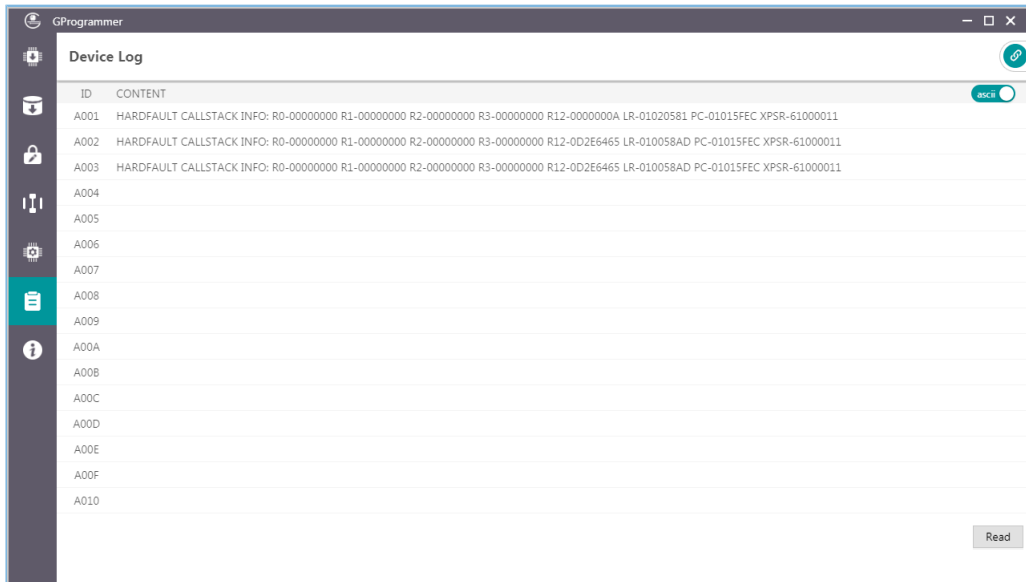


Figure 3-45 Device Log interface

Table 3-6 GR551x_console supported commands

Command	Functional Description	Command Format and Parameter Description
program	Programs firmware files to internal SoC flash memories.	<p>program <firmware file path> <run immediately:y n></p> <p>Parameter description:</p> <ul style="list-style-type: none"> <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.	<p>erase <start address<hex>> <end address<hex>><force erase when conflict with firmware/bootinfo/nvds:y n></p> <p>Parameter description:</p> <ul style="list-style-type: none"> <start address<hex>>: It represents the start address of the storage area to be erased (in hexadecimal). <end address<hex>>: It represents the end address of the storage area to be erased (in hexadecimal). <force erase when conflict with firmware/bootinfo/nvds: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.	<p>download <data file path> <start address<hex>> <force download when conflict with firmware/bootinfo/nvds:y n></p> <p>Parameter description:</p> <ul style="list-style-type: none"> <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/nvds: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, Boot info, or NVDS.
writeefuse	Writes Encrypt Key Info and Mode Control files to eFuse.	<p>writeefuse <Encrypt Key Info file Path> <Mode Control file Path></p> <p>Parameter description:</p> <ul style="list-style-type: none"> <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

Command	Functional Description	Command Format and Parameter Description
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 3-48](#).

```
C:\Users\>cd "C:\Program Files (x86)\Goodix\GProgrammer"
C:\Program Files (x86)\Goodix\GProgrammer>GR551x_console.exe program "C:\ble_app_hrs_fw.bin" y
2%
4%
6%
8%
10%
12%
15%
17%
19%
21%
23%
25%
28%
30%
32%
34%
36%
38%
41%
43%
45%
47%
49%
51%
```

Figure 3-48 Executing the program command

Note:

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

3.3.9.2 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 <i>firmware.key</i> , which is used for encryption and signing, or signing only.

Parameter	Description
signature_key	Shows the directory of <i>sign.key</i> , which is used for encryption and signing, or signing only.
signature_pub_key	Shows the directory of <i>sign_pub.key</i> , which is used for encryption and signing, or signing only.
product_json_path	Shows the directory of <i>product.json</i> , which is used for encryption and signing, or signing only.
rand_number	Shows the directory of <i>random.bin</i> , 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.
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 3.3.5.1 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 3-49](#):

```

C:\Users\ >cd "C:\Program Files (x86)\Goodix\GProgrammer"

C:\Program Files (x86)\Goodix\GProgrammer>GR551x_encrypt_signature.exe --operati
on="encryptandsign" --firmware_key="C:/eFuse/firmware_key" --signature_key="C:/e
Fuse/sign_key" --signature_pub_key="C:/eFuse/sign_pub_key" --product_json_path="
C:/eFuse/product.json" --ori_firmware="C:/firmware/hle_app_hrs_fw.bin" --output=
"C:/firmware_encryptAndSign/hle_app_hrs_fw_encryptAndSign.bin" --random_output="
C:/firmware_encryptAndSign/random.bin"

1
2
3
4
5
6
const n: 0x622a551b
hash:
0580dc03e7142230d01213e4cea38267f8f0d6b8639f304df616e801a91024a
buf: 0xe0,0xf1,0x11,0xd5,0x92,0xc1,0xc8,0x16,0x91,0x94,0x2c,0xe1,0x3e,0xf1,0x29,
0xfb,
buf: 0x5a,0xd8,0x90,0xfb,0x9d,0xd7,0x69,0x93,0x53,0xed,0xbe,0x4e,0x29,0x00,0xd4,
0x9c,0xfb,0x29,0xf1,0x3e,0xe1,0x2c,0x94,0x91,0x16,0xc8,0xc1,0x92,0xd5,0x11,0xf1,
0xe0,0x05,0x80,0xdc,0x03,0xe7,0x14,0x22,0x30,0xd0,0x12,0x13,0xe4,0xce,0xa3,0x82,
0x67,0xf8,0xf0,0xd6,0xb8,
buf: 0x3c,0x41,0xc5,0xa0,0x41,0x93,0x1c,0x04,0x41,0xb1,0x45,0x4b,0xd6,0x66,0x0f,
0xa3,0xa2,0xe7,0xef,0x9d,0xf9,0xe2,0xbe,0x0a,0x2e,0xc5,0x31,0x53,0x93,0xf,0xf,
0x71,
buf: 0xeb,0x01,0xca,0x8d,0x5b,0xb0,0x67,0x48,0x15,0x07,0x89,0xc6,0xdb,0x55,0xb1,
0x8e,0x48,0x2c,0x51,0x48,0x06,0x9b,0xcf,0x05,0xda,0x6c,0x43,0xdb,0xed,0x09,0xf3,
0x93,

C:\Program Files (x86)\Goodix\GProgrammer>

```

Figure 3-49 Executing the encryption and signing command

3.3.9.3 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.

3.4 GMF03x Series

This section elaborates on functional modules of GProgrammer for GMF03x series.

3.4.1 Main Operational Interface

After you choose a GMF03x MCU, the main operational interface opens, as shown in the figure below.

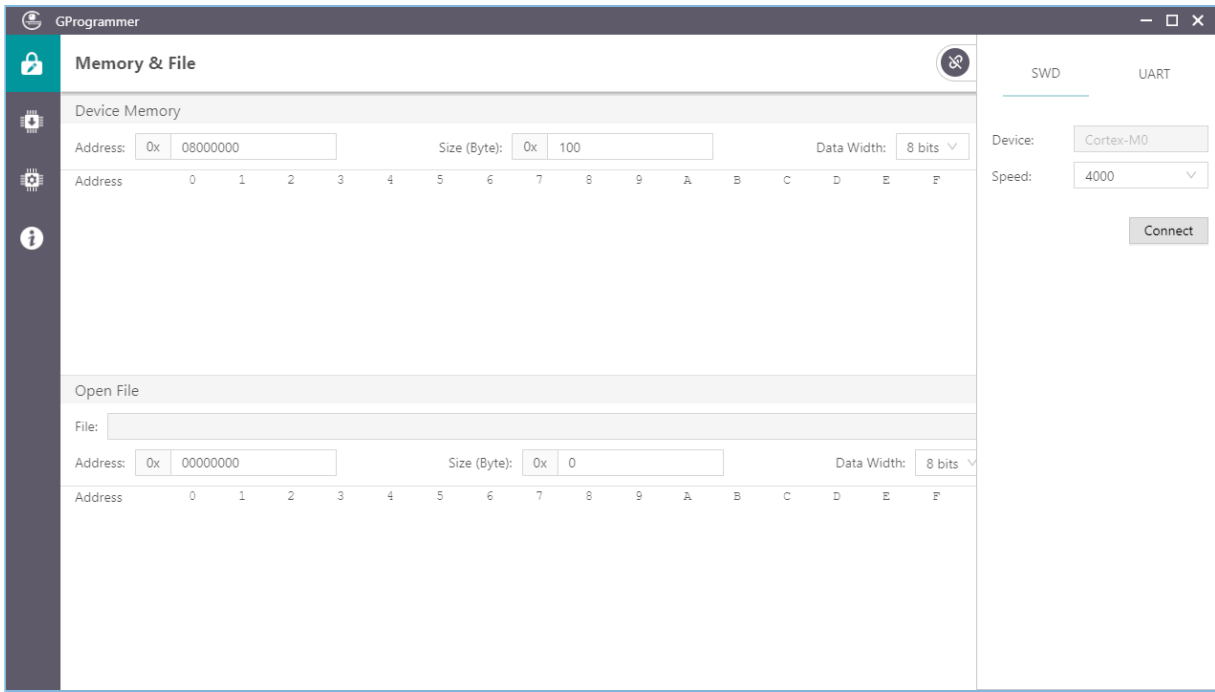


Figure 3-50 GProgrammer GUI (for GMF03x series)

The GUI comprises a functional navigation bar on the left (see Table 3-8) and a function operational zone on the right.

Table 3-8 Options on the functional navigation bar

Icon	Function Name	Description
	Memory & File	Displays operations related to memory and files.
	Programming & Erasing	Displays operations related to flash programming and erasing.
	User Option Bytes	Displays configurations of User Option Bytes.
	Help	Displays help information.

3.4.2 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.

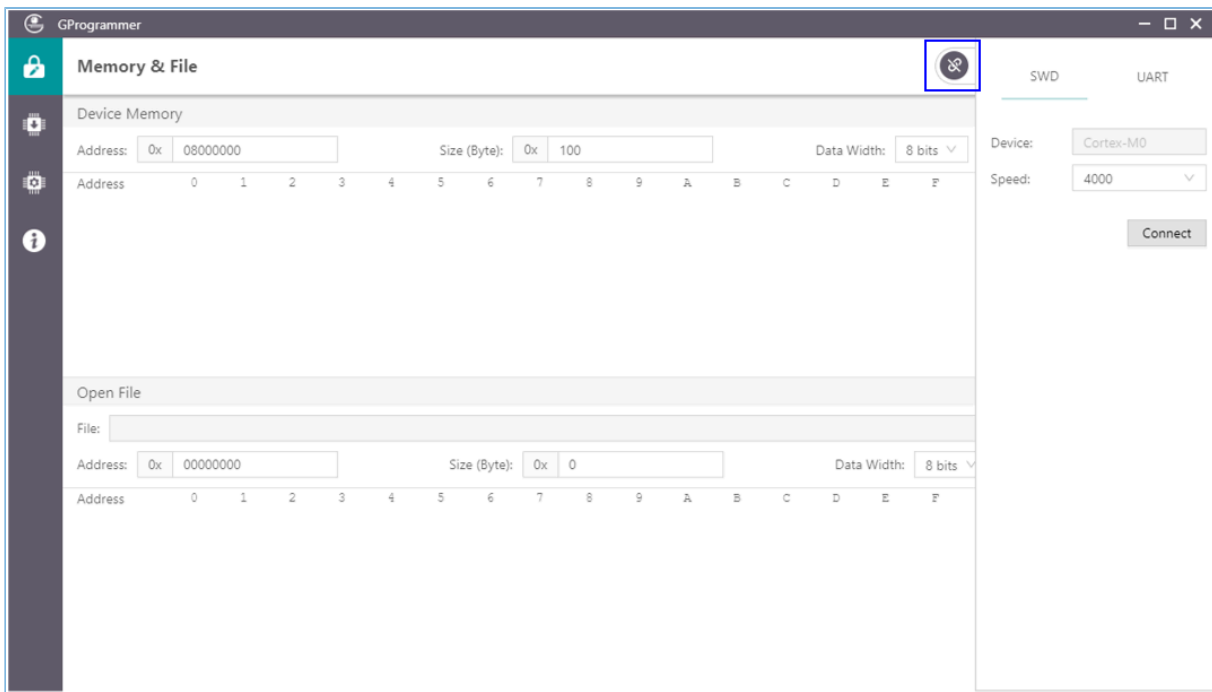


Figure 3-51 GProgrammer connection management window

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.

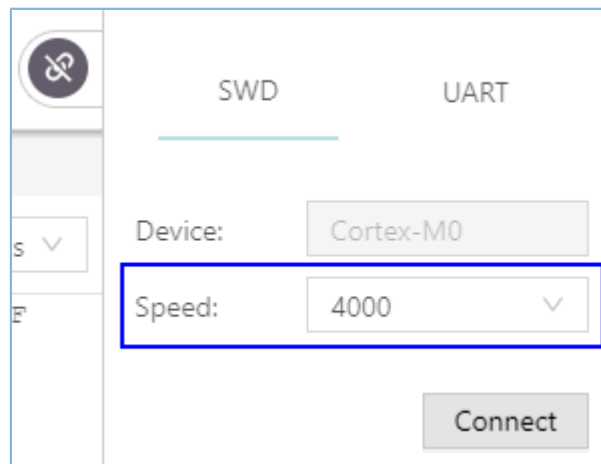


Figure 3-52 GProgrammer SWD connection

- UART

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

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

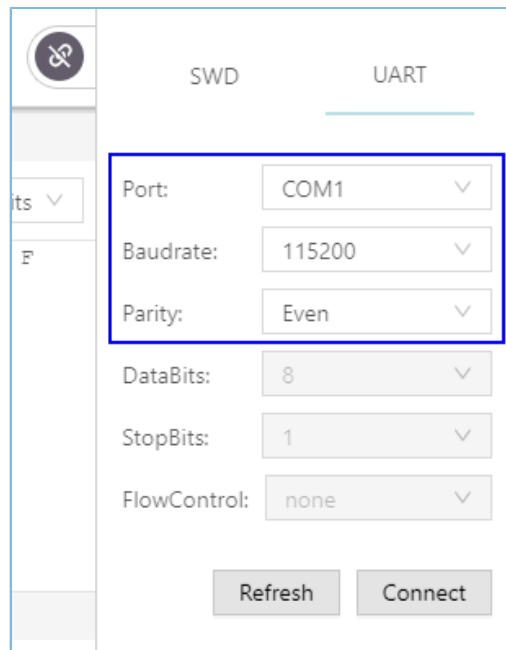




Figure 3-53 GProgrammer UART connection

After the connection is successfully established, the connection management window automatically hides with the  button turning into , which indicates successful connection establishment. Meanwhile, a dialog box pops up, showing the model of the MCU in connection.

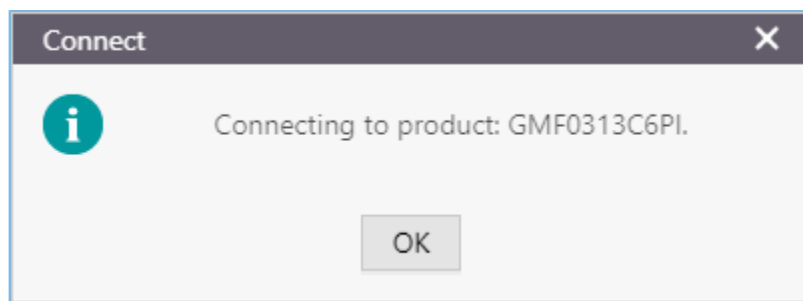

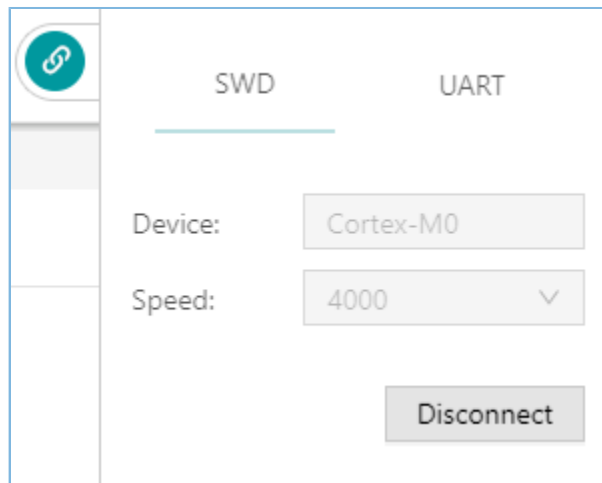


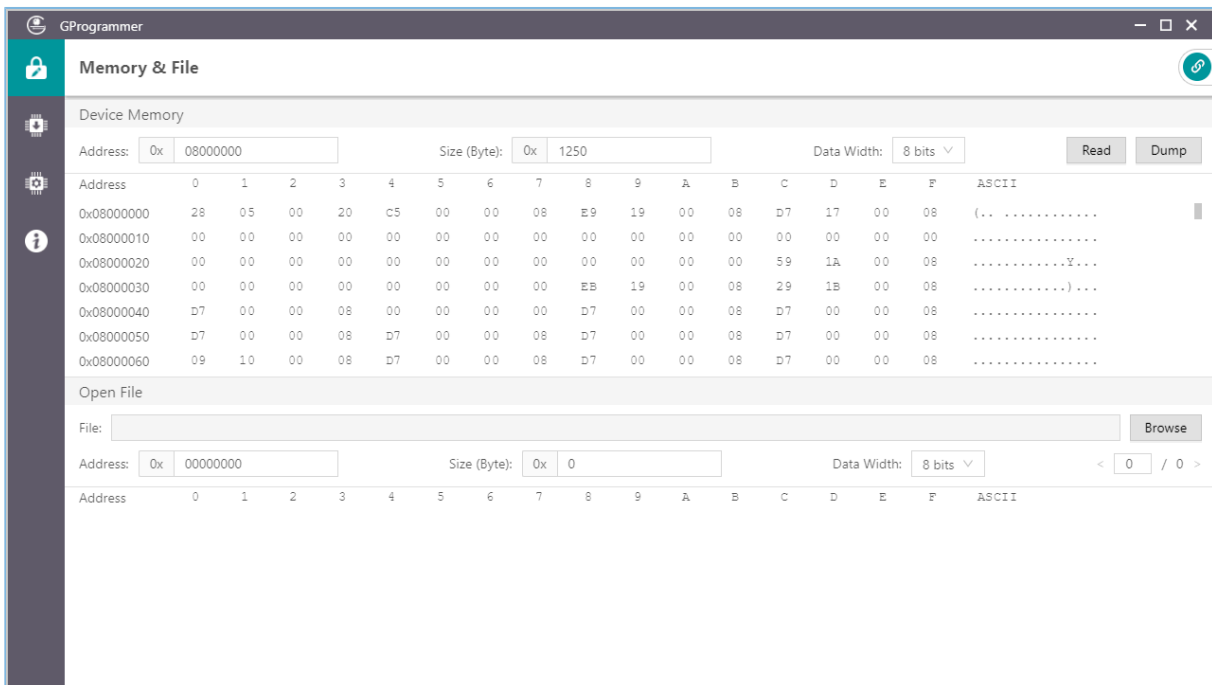
Figure 3-54 MCU model in connection

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

Figure 3-55 Clicking **Disconnect** on GProgrammer

3.4.3 Memory & Flash

Click  on the left side of the main interface of GProgrammer to open the **Memory & File** interface.

Figure 3-56 GProgrammer **Memory & File** interface

GProgrammer allows users to read from and write to memories on GMF03x MCUs as well as view BIN and HEX files.

3.4.3.1 Read/Write to Memory

In the **Device Memory** pane, GProgrammer allows users to read data from and write data to memories on GMF03x MCUs.

- Read memory

When using GProgrammer to read data from memories, set **Address** (starting address) and **Size (Byte)** (data size) before clicking **Read**.

The read data is displayed in the list under the read parameters. You can select **8 bits** (byte), **16 bits** (half-word), and **32 bits** (word) in the **Data Width** list to display the read data.

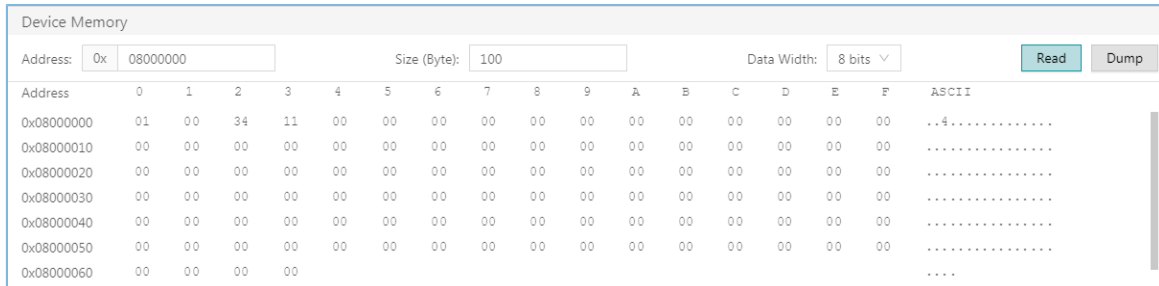


Figure 3-57 Reading data from memories on GProgrammer

You can click **Dump** to store the read data locally into a BIN file.

Tip:

The **Address** should be 4-byte aligned.

- Write to memory

You can write data to memories on GMF03x MCUs using GProgrammer.

In the data display list, double-click the data to be modified. Input a new data value in the text box, and press Enter on your keyboard. The new data is written to the memory.

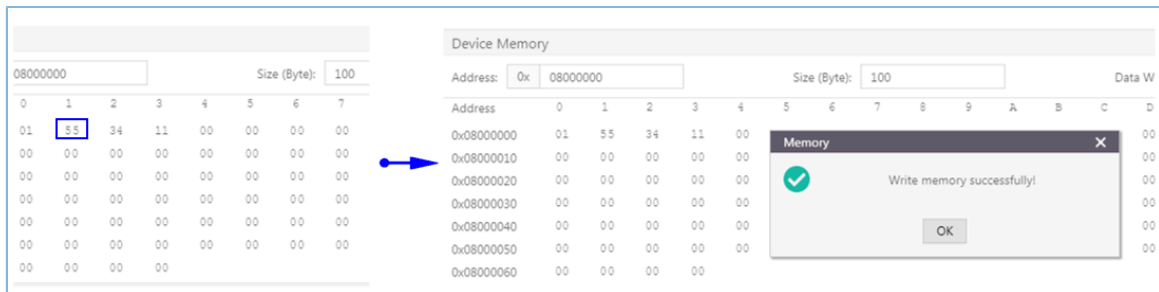


Figure 3-58 Writing data to memories on GProgrammer

Tip:

Double-click the **Size (Byte)** input box to switch between Hexadecimal and Decimal.

3.4.3.2 Open File

In the **Open File** pane, GProgrammer allows users to view data in BIN and HEX files.

Click **Browse** to open a file on your local PC. Select **8 bits**, **16 bits**, or **32 bits** from the **Data Width** list to decide a data display mode. GProgrammer automatically calculates the **Size (Byte)** of the file.

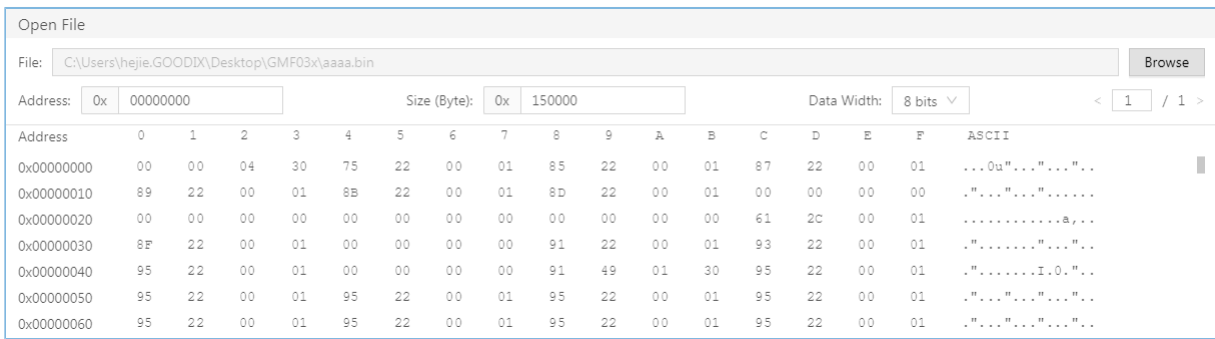



Figure 3-59 Reading out a BIN file on GProgrammer

When reading out a HEX file, click  under the **Browse** button to turn to another page, which enables quick reading of the HEX file.

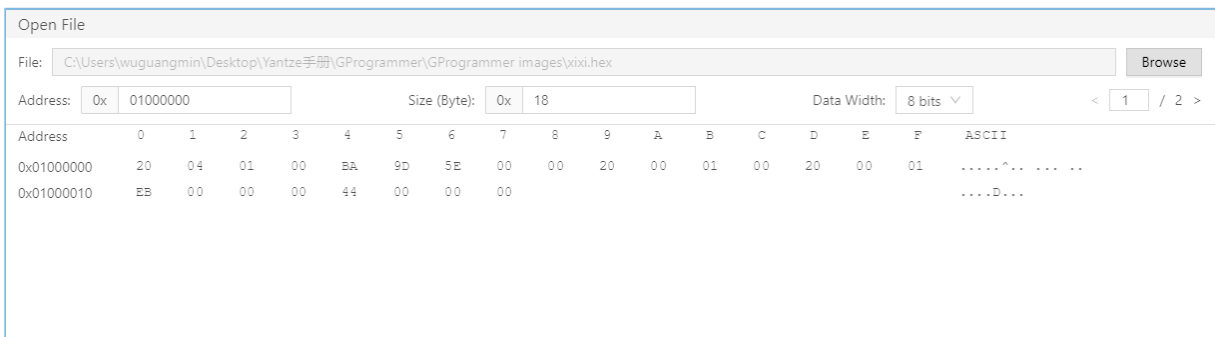



Figure 3-60 Reading out a HEX file in pages on GProgrammer

3.4.4 Programming & Erasing

Click  on the left side of the main interface of GProgrammer to open the **Programming & Erasing** interface.

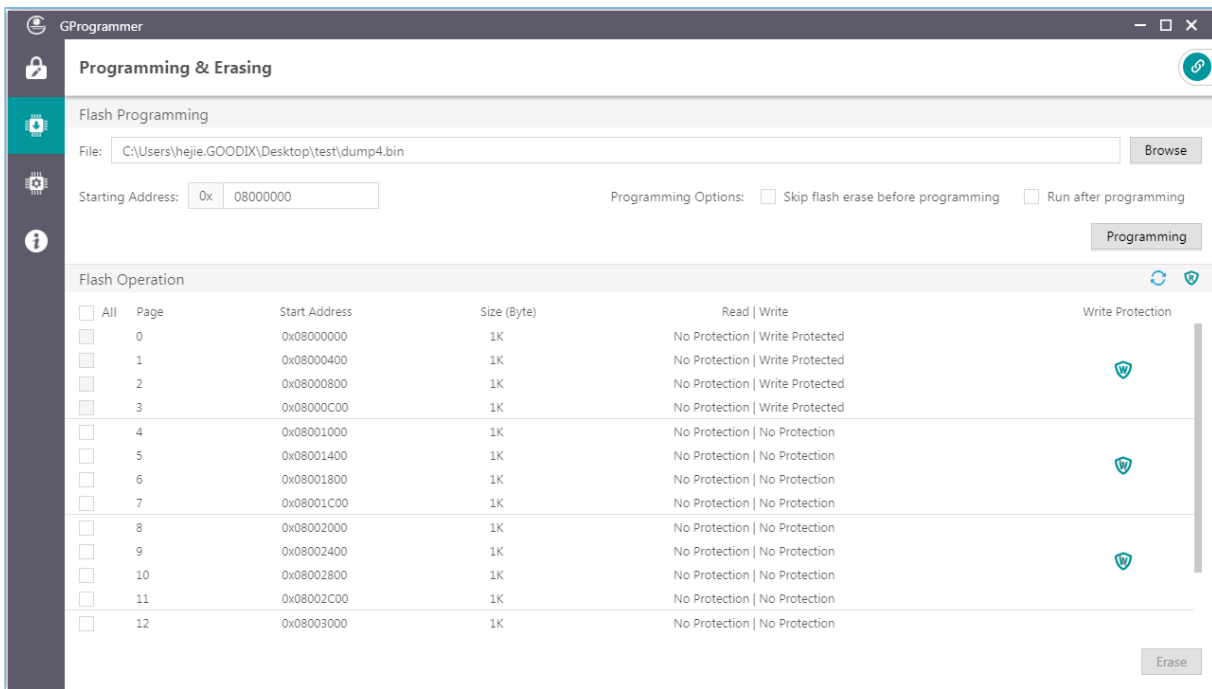


Figure 3-61 GProgrammer Programming & Erasing interface

GProgrammer allows users to program and erase flash memories with read or write protection enabled or disabled.

3.4.4.1 Flash Programming

Follow the steps below to program flash memories on a GMF03x MCU:

1. Click **Browse** to open the file (BIN or HEX) to be programmed.
2. Set the **Starting Address**.
3. Configure **Programming Options** (Select or clear **Skip flash erase before programming** and/or **Run after programming**).
4. Click **Programming**.

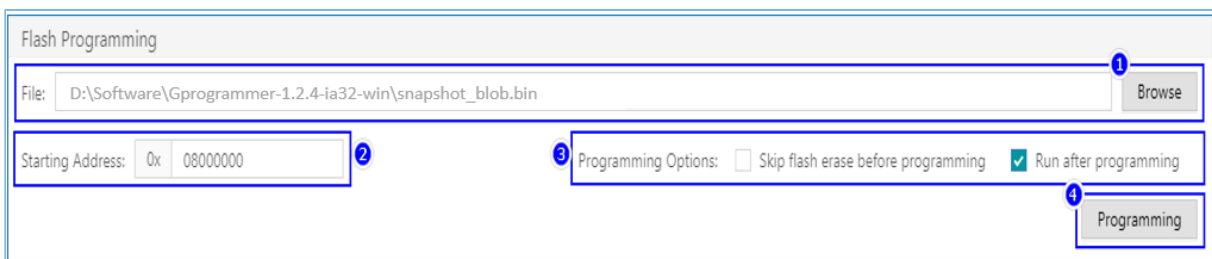


Figure 3-62 Programming flash on GProgrammer

Note:

- When programming a HEX file, you do not need to set the **Starting Address**. **Run after programming** should be cleared.
- The **Starting Address** should be 4-byte aligned.

3.4.4.2 Flash Erasing

GProgrammer supports erasing of main flash memory areas by page. You can select a page or pages to be erased on demand, and click **Erase**.

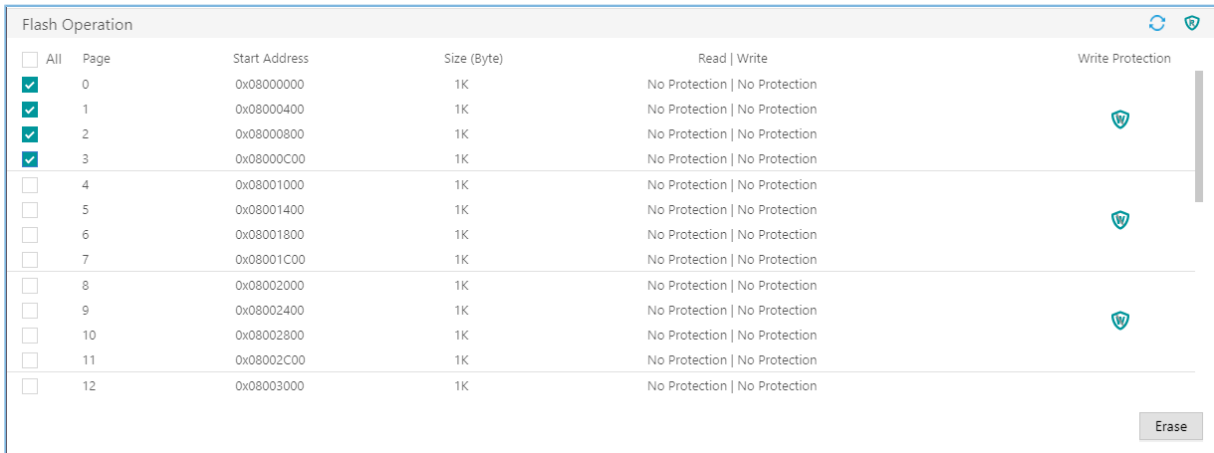


Figure 3-63 Erasing flash memories by page

Note:

Pages under write protection cannot be erased.

You can select **All** to choose all pages (except those under write protection), and click **Erase** to erase all the chosen pages at one time.

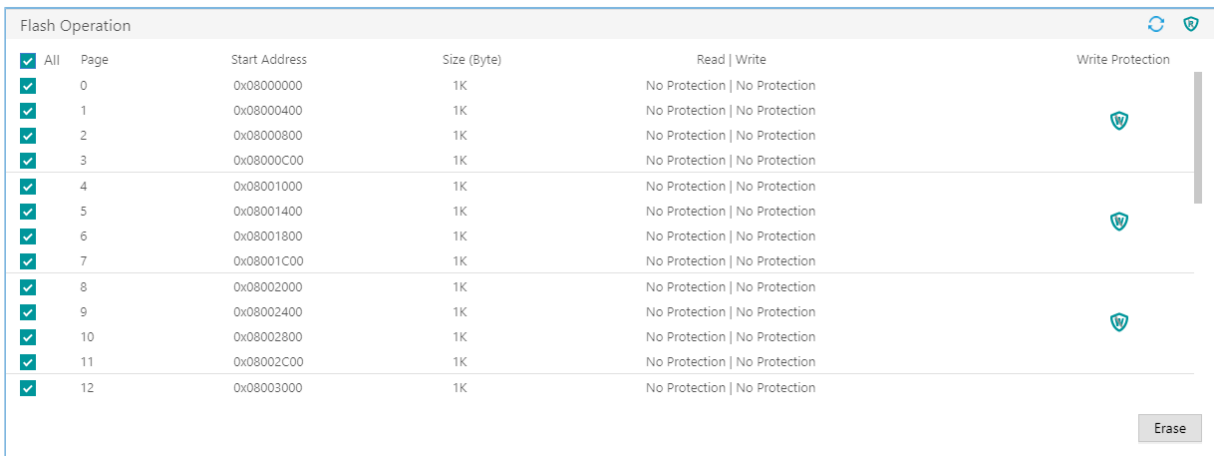


Figure 3-64 Erasing all on GProgrammer

3.4.4.3 Read/Write Protection for Flash Memory

GProgrammer protects data reads and writes on flash memories on GMF03x MCUs.

- Read protection

Flash memory areas under read protection cannot be read. The protected area varies depending on the connection mode. When **Read Protection** is enabled: In SWD mode, the main flash memory area cannot be read. In UART mode, all flash memory areas cannot be read.

Tip:

For MCUs with read protection enabled, UART connection cannot be established.

Click  in the upper right of the **Flash Operation** pane to enable read protection.

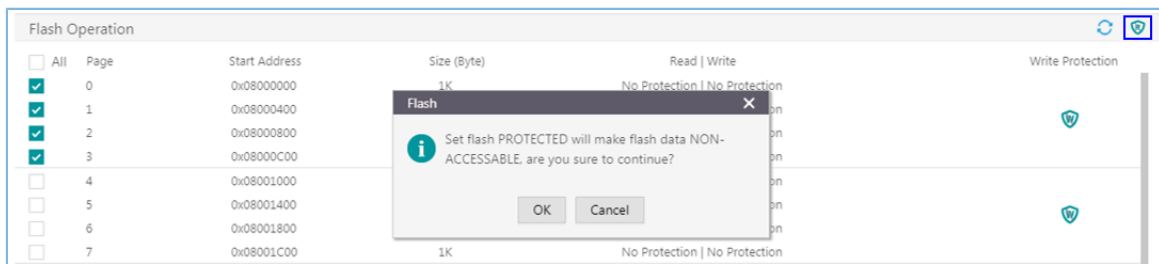





Figure 3-65 Enabling **Read Protection** on GProgrammer

The  icon turns into  after **Read Protection** is enabled. Click  to disable **Read Protection** and erase the main flash memory area.

- Write protection

When **Write Protection** is enabled, you cannot write data to specific flash memory areas under write protection. GProgrammer protects flash memory against being written to by sector (four pages) only.

Click  in the **Write Protection** column to enable write protection.

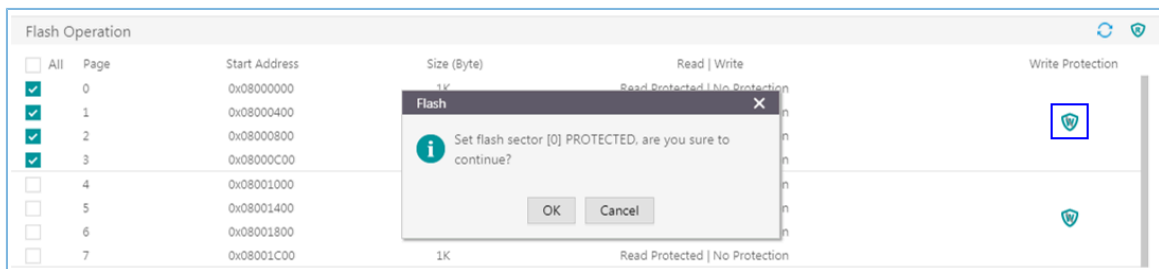





Figure 3-66 Enabling **Write Protection** on GProgrammer

The  icon turns into  after **Write Protection** is enabled. The check boxes of the pages corresponding to protected sectors cannot be selected. Click  to disable **Write Protection**.

3.4.5 User Option Bytes

Click  on the left side of the main interface of GProgrammer to open the **User Option Bytes** interface.

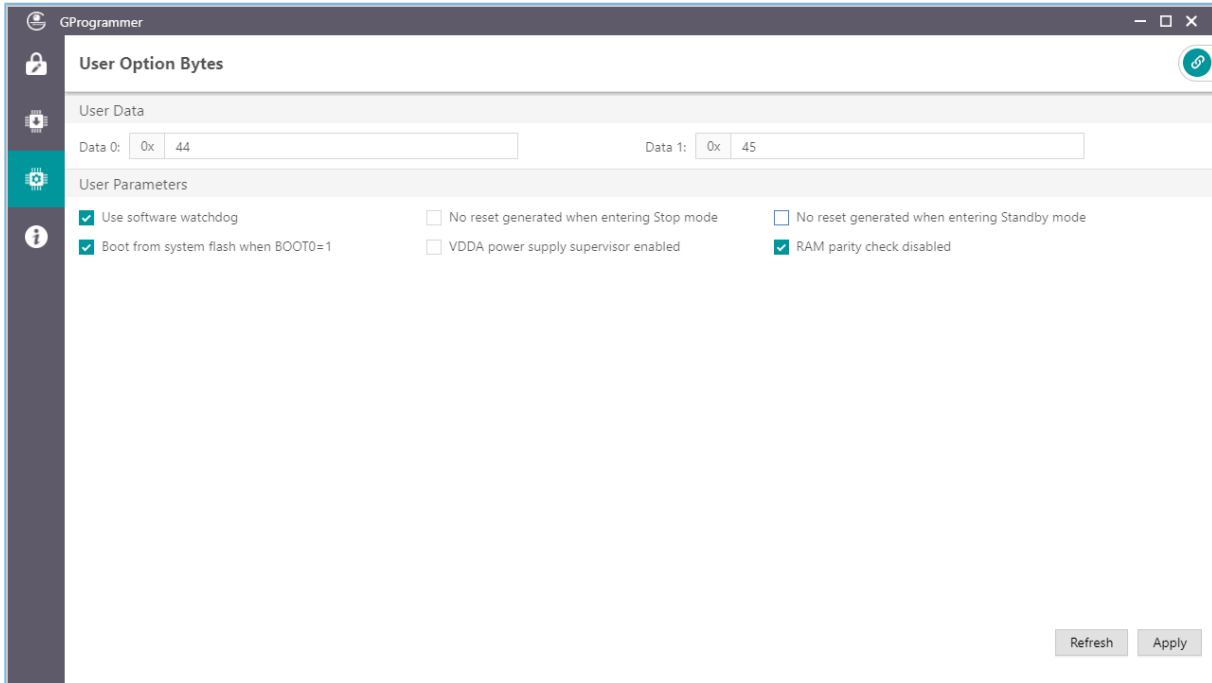



Figure 3-67 GProgrammer **User Option Bytes** interface

GProgrammer allows users to modify User Option Bytes of GMF03x MCUs. Option examples include **User software watchdog**, **Boot from system flash when BOOT0=1**, and **RAM parity check disabled**.

After setting all user option byte parameters in the interface, click **Apply** to program the settings into an MCU.

Click **Refresh** to obtain the user option bytes of the MCU.

3.5 Help

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

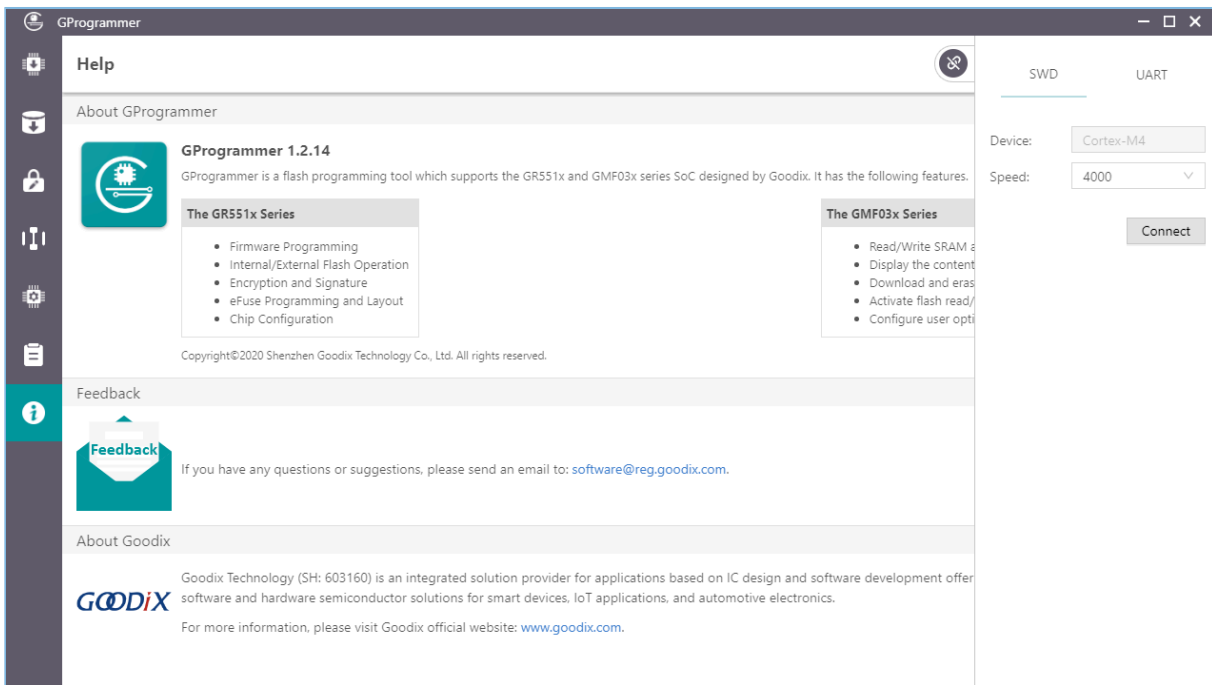


Figure 3-68 GProgrammer Help information

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: <http://www.goodix.com>.