

# **GProgrammer User Manual**

Version: 2.3

Release Date: 2021-07-16

Shenzhen Goodix Technology Co., Ltd.

## 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**

**GODIX** 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	<ul> <li>Updated sector-related description in "Chip Configuration".</li> <li>Added "GR551x_console.exe", introducing a command-line program to erase and download commands; added "GR551x_encrypt_signature.exe" and "User-defined Windows Scripts".</li> <li>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 &amp; Sign".</li> </ul>
1.7	2020-08-30	<ul> <li>Introduced the GR5515I0ND SoC for GR551x SoCs in "SoC/MCU Selection".</li> <li>Changed icons for <b>Delete</b> and <b>Startup</b> in "Firmware".</li> </ul>
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	<ul> <li>Added file modification description to "eFuse Settings".</li> <li>Added file export description to "Import and Export".</li> <li>Updated descriptions concerning operations prior to viewing device logs in "Device Log".</li> <li>Added description of IO_LDO_SEL field to "eFuse Layout".</li> <li>Deleted the parameter of nvds in erase and download commands in "GR551x_console.exe".</li> </ul>
2.2	2021-05-13	Deleted functionalities for GMF03x series.
2.3	2021-07-16	Updated software UI figures for SoC selection.

# GODIX

# Contents

1 Introduction	. 1
2 Installation Instructions	.2
2.1 Installation Requirements	2
2.2 Installation Steps	. 2
3 Programming Flash with GProgrammer	5
3.1 Hardware Connection	.5
3.2 SoC Selection	. 6
3.3 Main Operational Interface	. 7
3.4 Connection Management	.8
3.5 Firmware	10
3.5.1 Download Firmware	10
3.5.2 Action Order	12
3.6 Flash	13
3.6.1 Internal Flash	14
3.6.1.1 Flash Configuration	14
3.6.1.2 Erase Flash	15
3.6.1.3 Download Data	17
3.6.1.4 Dump Data	18
3.6.2 External Flash	19
3.6.2.1 Flash Configuration	19
3.6.2.2 External Flash Programming	21
3.7 Encrypt & Sign	22
3.7.1 eFuse Settings	22
3.7.2 Download	24
3.7.3 Encrypt & Sign	25
3.8 eFuse Layout	26
3.9 Chip Configuration	28
3.9.1 Init NVDS Area	30
3.9.2 Read All	30
3.9.3 Write	31
3.9.4 Add a User Parameter	32
3.9.5 Modify NVDS Parameters	34
3.9.6 Remove a User Parameter	35
3.9.7 Import and Export	36
3.10 Device Log	36
3.11 Command-line Programs	38
3.11.1 GR551x_console.exe	38

3.11.2 GR551x_encrypt_signature.exe	40
3.11.3 User-defined Windows Scripts	41
3.12 Help	42

# **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.

9	GProgrammer						- 🗆 🗙
۲	Firmware						Ø
Ŧ	0x010F FFFF		Firmware File				
0			User App Firmware:				
			(i) Image Info				
(I)			Image Name:	ble_app_hts_	Run Address:		
	Unused		Version:	1	System Clock:	16MHz(xo)	$\sim$
٥	NVDS		Size(Byte):		XQSPI Speed:	16MHz	$\sim$
a	Existed		SPI Access Mode:		Boot Delay:	🔵 Yes 🔘 No	
	Download	ble_app_hrs_	CheckSum:	7824136	Check Image:	🔵 Yes 🔘 No	
ð	Delete	ble_app_bps	Load Address:		Code Copy Mode:	🔘 QSPI 💿 XIP	
	Overlapping						Update
	🧧 Update	ble_app_hrs   💡					
			No. Action	Description			
			1 update	Update ble_app_h	rs image info		×
			2 delete	Delete ble_app_ht	s_		×
		ble and bte	3 add	Add and download	d ble_app_ancs_fw.bin		×
	0x0100 2000	ble_app_lits_	4 add	Add and download	d ble app bps fw.bin	-	×
		Refresh Add Delete Startup					Commit

#### 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
СРU	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:

- GODIX
- 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.

# GODIX

## 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 pir	elations between USB-to-serial converter pins	-to-serial converter pins and SoC pins
--	---	--

USB-to-Serial Converter Pin	GR551x SoC Pin
VCC	VCC
GND	GND
тх	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.



Programmer									- (	
<b>T</b> Filter Settings	C	Find	Find Products							
Products	~	GR5	GR551x series is a high-performance system-on-chip (SoC) supporting Bluetooth 5.1, making it an ideal choice for mobile devices, wearables, and							
# Part Number	>	Internet Perinher	nternet of Things (IoT) products. The series allows users to develop Bluetooth Low Energy (LE) applications and products serving as a Central and/or a							
Series	>	i cripitett		o oronooic in o	, no o packag				G@DiX	
Core	>									
🔲 Memory	>									
Package	>	ltem	list 5 items							
< Peripheral	>		Part Number	Core	Frequency	RAM	Flash	Package	Peripherals	
Kits	>	•	GR5515IGND	Cortex-M4F	64MHz	256KB	1024KB	QFN56	2 x QSPI, 2 x SPI, 2 x I2C, 2 x I2S, 2 x UART, 1 x ADC, 1 x ISO/816, 6 x PWM	
		•	GR5515RGBD	Cortex-M4F	64MHz	256KB	1024KB	BGA68	2 x QSPI, 2 x SPI, 2 x I2C, 2 x I2S, 2 x UART, 1 x ADC, 1 x ISO7816, 6 x PWM	
			GR5515GGBD	Cortex-M4F	64MHz	256KB	1024KB	BGA55	0 x QSPI, 2 x SPI, 2 x I2C, 2 x I2S, 2 x UART, 1 x ADC, 1 x ISO7816, 6 x PWM	
			GR551510ND	Cortex-M4F	64MHz	256KB	OKB	QFN56	2 x QSPI, 2 x SPI, 2 x I2C, 2 x I2S, 2 x UART, 1 x ADC, 1 x ISO7816, 6 x PWM	
		•	GR5513BEND	Cortex-M4F	64MHz	128KB	512KB	QFN40	1 x QSPI, 2 x SPI, 2 x I2C, 1 x I2S, 2 x UART, 1 x ADC, 1 x ISO7816, 6 x PWM	

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.



	GProgrammer			- 🗆 ×
۰	Firmware	8	SWD	UART
	0x010F FFFF Unused NVDS	Firmware File         User App Firmware:         Image Info         Image Name:       Run Address:         Version:       Size(Byte):         SPI Access Mode:       Boot Delay:         CheckSum:       Check Image:         Load Address:       Image Name:         Image Description       No.	Device: Speed:	Cortex-M4 4000 ∨ Connect
	0x0100 2000 Refresh Add Delete Startup			

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
-----------	---------	--------	------------	----------------

lcon	Function Name	Description
ø	Firmware	Displays firmware-related operations.
ບົ	Flash	Displays operations related to flash memory.
ê	Encrypt & Sign	Displays operations related to firmware encryption and signing.
40	eFuse Layout	Displays eFuse layout.
ø	Chip Configuration	Displays operations related to chip configurations.
ā	Device Log	Displays device logs.
Ð	Help	Displays help information.

## 3.4 Connection Management

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

Click I 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.

8	SWD		UART
	Port:	COM1	$\sim$
	Baudrate:	921600	
	Parity:	None	$\sim$
ss:	DataBits:	8	$\sim$
ck:	StopBits:	1	$\sim$
	FlowControl:	none	$\sim$
ge:	Re	fresh	Connect
Mada			

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.

( <u></u>	SProgrammer						- 🗆 🗙
۲	Firmware						Ø
¥	0x010F FFFF		Firmware File				
0			User App Firmware:	E:\Firmware\ble_app_hts_fw.b			
			image Info				
ιIι			Image Name:	ble_app_hts_	Run Address:		
	Unused		Version:	1	Size(Byte):	81024	
	NVDS		SPI Access Mode:		Boot Delay:	🖲 Yes 🔘 No	
8	Existed		CheckSum:	7990386	Check Image:	🖲 Yes 🔵 No	
	Download	ble_app_hrs_	Load Address:				
ð	Delete	ble_app_bps					Update
	Overlapping						
	💡 Update	ble_app_hrs	Unfinished Eve	nts			
			No. Action	Add and download	ble ann ancs c fw.bin		×
			2 add	Add and download	ble_app_hts_fw.bin		×
			3 startup	Start up ble_app_ht	:s_		×
	0x0100 2000	ble_app_hts_					
		Refresh Add Delete Startup					Commit

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.

0x010F FFFF	
	ble_app_ancs
Unused	
NVDS	
Existed	
Download	ble_app_hrs_
Delete	ble_app_bps
Overlapping	
💡 Update	ble_app_hrs 🥊
0x0100 2000	ble_app_hts_
	Refresh Add Delete Startup

Figure 3-9 Flash firmware layout

• represents flash space to which data can be downloaded.

GODIX

- Prepresents 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.

# G@DiX

## 🛄 Note:

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

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

#### Figure 3-10 Action order

Executable actions for users are listed in the table below.

#### Table 3-4 Executable actions for users on GProgrammer

Name	Button/Icon	Description
		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
Add firmware	Add	GProgrammer from Windows/File Explorer.
		Note:
		Do not click <b>Open</b> after dragging the file to GProgrammer.
		Click Refresh to obtain the information of firmware downloaded in the flash memories of a target
		board.
Refresh firmware	Refresh	Unexecuted actions of flash firmware on the living target board in the Unfinished Events pane,
		such as those labeled as <b>startup</b> or <b>update</b> are withdrawn with modified parameters being reset
		to values before refresh.



Name	Button/Icon	Description
Delete firmware	Delete	Click the <b>Delete</b> button to delete existing firmware in flash memories. Select firmware to be deleted in the flash firmware layout, and click <b>Delete</b> . The firmware color turns to An action labelled as <b>delete</b> is added to the <b>Unfinished Events</b> . 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 <b>startup</b> to run the firmware immediately. Select firmware in the flash firmware layout, and click the <b>Startup</b> button. displays on the right of the firmware. An action labelled as <b>startup</b> is added to the <b>Unfinished Events</b> . The host automatically disconnects from the target board after running the firmware.
Update firmware information	Update	Click the <b>Update</b> 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 <b>Update</b> , and the icon displays on the right side of the firmware. An action labelled as <b>update</b> is added to the <b>Unfinished Events</b> . 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 imes 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.



<u>e</u> 0	GProgrammer			- 🗆 ×
	Flash			Ø
J	0x010F FFFF		Flash Configuration	
A			● Internal Flash	
			External Flash ID: Flash Size: 1 M V	Config
<b>'</b> "'				
			Erase Flash	
- <b>···</b> ·			C Erase All	
B	Unused		C Erase Sector	
	Boot		Erase Specified Area     Ox     01002000     to     Ox     0102FFFF	Erase
ð	NVDS			
	Firmware	ble_app_bps	🕹 Download Data	
			File Path:	B
			File Size(Byte):         Download Address:         0x         00000000	Download
			🖸 Dump Data	
	0.0100.0000		Starting Address: 0x 00000000 Size(Byte): 0	Dump
	UxU100 0000			

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.



9	GProgrammer			- 🗆 ×
•	Flash			Ø
	Ox010F FFFF		Flash Configuration Internal Flash Flash ID: Flash Size: 1 M V	Config
	Unused Boot NVDS		Erase Flash Erase All Erase Sector O Erase Specified Area O X 01002000 to O X 0102FFFF	Erase
	Firmware	ble_app_bps	Download Data File Path: File Size(Byte): Download Address:	Dwnload
	0x0100 0000	ble_dfu_boot	Dump Data       Starting Address:     0x     00000000     Size(Byte):     0	Dump

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.

9	GProgrammer			- 🗆 ×
•	Flash			Ø
U	0x010F FFFF		Blash Configuration	
2			<ul> <li>● Internal Flash</li> <li>○ External Flash</li> <li>Flash ID:</li> <li>Flash Size: 1</li> <li>M ∨</li> </ul>	Config
.∎ ©			Erase Flash	
B	Unused Boot		Erase Sector         0           Erase Specified Area         0x         01002000         to         0x         0102FFFF	Erase
	Firmware	ble_app_bps	Warning     X       South State     Image: State       Image: State     Image: Sta	Þ
			File Size(Byte): OK Cancel 00000000	Download
	0x0100 0000	ble_dfu_boot	Dump Data       Starting Address:     0x     00000000     Size(Byte):     0	Dump

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

#### Erase Sector

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

، ڪ	GProgrammer		- 🗆 🗙
	Flash		Ø
€	0x010F FFFF	Flash Configuration	
٨		● Internal Flash     ○ External Flash ID: Flash Size: 1 M ✓	Config
١Į١			
0		Erase Flash	
Ē	Unused	Image: Trase Specified Area         Ox         01010000         to         Ox         01020000	Erase
1	NVDS		
		File Path:	B
		File Size(Byte): Download Address: 0x 0000000	Download
		Dump Data	
	0x0100 0000	Starting Address: 0x 00000000 Size(Byte): 0	Dump

Figure 3-14 Erase Sector on GProgrammer



#### • Erase Specified Area

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

٩	GProgrammer		- 🗆 ×
•	Flash		Ø
€	0x010F FFFF	Flash Configuration	
٨		<ul> <li>● Internal Flash</li> <li>&gt; External Flash ID:</li> <li>Flash Size: 1</li> </ul>	Config
UI)			
۵		Erase Hlash	
Ē	Unused	Erase Sector         0                erase Specified Area               0x               0x               0x               0x               01020000	Erase
ð	NVDS		
		Download Data File Path:	B
		File Size(Byte):     Download Address:     0x     00000000     0	Download
		C Dump Data	
	0x0100 0000	Starting Address: 0x 0000000 Size(Byte): 0	Dump

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**.

٩	GProgrammer						- 🗆 ×
	Flash						Ø
₹ 20 11 10 10 10 10 10 10 10 10 10 10 10 10	0x010F FFFF Unused Boot NVDS Firmware	ble_app_ancs	Flas ← → ← ↑ • • • • • • • • • • • • • • • • • •	Desktop > GProgrammer der Name ble_app_ancs_fw.bin ble_app_bps_fw.bin ble_dfu_boot_fw.bin <pre></pre>	<ul> <li>♥</li> <li>■</li> <li>■</li> <li>♥</li> <li>■</li> <li>■</li> <li>■</li> <li>♥</li> <li>■</li> <li>♥</li> <li>■</li> <li>■</li></ul>	× mer Type BIN File BIN File BIN File	Config Erase
	0x0100 0000	Star	Dump Data ting Address: Ox	0000000	Open Size(Byte): 0	Cancel	Dump

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

GODiX

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

9	GProgrammer			- 🗆 ×
	Flash			Ø
	0x010F FFFF		Flash Configuration  Internal Flash Flash ID: Flash Size: 1 M  External Flash Flash Size: 1 M	Config
	Unused Boot NVDS Firmware	ble_app_bps	Erase All  Failed  Failed	Erase
			Download Data     File Path: F\新产品开发\GRProgrammer测试\ble_app_T3u_fw.bin     File Size(Byte): 85728 Download Address: 0x 010EFFFF	Download
	0x0100 0000	ble_dfu_boot	Starting Address: 0x 00000000 Size(Byte): 0	Dump

#### 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.



ي چ	GProgrammer		- 1	ο×
۰	Flash			Ø
Ð	0x010F FFFF	Flas	S Save As X Search GProgrammer A Search GProgrammer A	
A		ble_app_ancs	n Organize ▼ New folder ()33 ▼ (2)	
	Unused Boot NVDS Firmware	<ul> <li>Extern</li> <li>Erase</li> <li>Erase</li> <li>Erase</li> <li>trase</li> </ul>	This PC Name Date Type S Con Type S Con Daily Record Daily Record Daily Record Destop Destop Document, publ Documents Documents Documents Documents Documents Documents Software Videos V C V C V File name	lise
		File Path:	Save as type: Image (".bin)	ø
		File Size(E	B A Hide Folders Cancel Downlo	ad
	0x0100 0000	Starting A	Address: 0x 010AFFFF Size(Byte): 150000 Dun	np

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.

External Flash Configuration								
SPI Type	e: (	SPI O QSPIO	◯ QSPI1					
GPIO Type		GPIO PIN	PIN MUX					
CS:	NORMAL V	GPIO_0 V	MUX_0 V					
CLK:	NORMAL V	GPIO_3 V	MUX_2 V					
MOSI:	NORMAL V	GPIO_4 V	MUX_2 V					
MISO:	NORMAL V	GPIO_5 V	MUX_2 V					
	Ap	oply Cancel						

Figure 3-19 SPI configurations



SPI Typ	e:	🔵 SPI 🛛 🧿 Q	SPIO	🔘 QSPI1
	GPIO Type	GPIO PIN		PIN MUX
CS:	AON V	AON_GPIO_1	$\sim$	MUX_5 V
CLK:	NORMAL V	GPIO_24	$\sim$	MUX_5 V
100:	NORMAL V	GPIO_25	$\sim$	MUX_5 V
101:	NORMAL V	GPIO_16	$\sim$	MUX_5 V
IO2:	NORMAL V	GPIO_17	$\sim$	MUX_5 V
IO3:	NORMAL V	GPIO_31	$\sim$	MUX_5 V

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.



، 2	GProgrammer		- 🗆 X
	Flash		Ø
U	0x000F FFFF	Flash Configuration	
		Internal Flash     External Flash Flash ID: FFFFFF Flash Size: 1 M V	Config
11		Warning ×	
ø		Erase Fla Erase All	
E		Erase Secto     OK	France
ð	Unused	Crase Specified Area	Liase
		🕹 Download Data	
		File Path: F:\GRProgrammer测试\ble_app_T3u_fw.bin	Þ
		File Size(Byte): 0x 01009000	Download
		Dump Data	
	0x0000 0000	Starting Address: 0x 010AFFFF Size(Byte): 150000	Dump

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.

9	GProgrammer		- 🗆 🗙
•	Flash		Ø
€	0x003F FFFF	Rlash Configuration	
		<ul> <li>○ Internal Flash</li> <li>● External Flash</li> <li>FIash ID: FFFFFF</li> <li>Flash Size: 4</li> </ul>	Config
ıI)			
۲		Erase All	
Ē		Erase Sector     10     Erase Specified Area     0x     01002000     to     0x     0102FFFF	Erase
ð	Unused		
		🕹 Download Data	
		File Path: F:\GRProgrammer测试\ble_app_T3u_fw.bin	B
		File Size(Byte):     85728     Download Address:     0x     00050000     I	Download
		🖸 Dump Data	
	0x0000 0000	Starting Address: 0x 010AFFFF Size(Byte): 150000	Dump

Figure 3-22 Download Data to external flash on GProgrammer

# G@DiX

## 🚨 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.

9	SProgrammer							- 0	ı ×
	Encrypt & Sign								8
9	eFuse Settings								
	Name:				ID:				
2	Firmware Key:	<ul> <li>Using Random Key</li> </ul>	🔿 Select Key						в
171	Security Mode:	● Open ○ Close			SWD:	● Open ◯ Close			
1.1	Batch eFuse:	2							
							[	Generate eFuse F	File
	Download								
	Encrypt Key Info:								B
•	Mode Control:								B
								Download to eFu	use
	Encrypt and Sign								
	Product Info:								B
	Random Number:	● Using Random Number	Select Numl	ber					В
	Firmware:							1	Þ
							Encrypt	Encrypt and Si	ign

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.

eFuse Settings						
Name:	test			ID:	1	
Firmware Key:	🔿 Using Random Key	<ul> <li>Select Key</li> </ul>	F:\GR551X u	pdate\tools\GF	Programmer\test\firmware.key	Ø
Security Mode:	● Open ◯ Close			SWD:	● Open ○ Close	
✓ Batch eFuse:	3					
Only Data Key is differe	ent between batch eFuse files.					Generate eFuse File

Figure 3-24 Setting eFuse parameters

# G@DiX

## 🛄 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.



<u>e</u> a	Programmer					- 🗆 X
	Encrypt & Sign					8
9	eFuse Settings					
	Name:	test		ID: 111	11	
₽	Firmware Key:	<ul> <li>Using Random Key</li> </ul>	🔿 Select Key			Þ
	Security Mode:	● Open 🔵 Close	Success		× Close	
171	✓ Batch eFuse:	3	Complete t	o generate efuse file.		
	Only Data Key is different	between batch eFuse files.	_			Generate eFuse File
_	Download			OK		
Ξ	Encrypt Key Info:	C:\eFuse\Encrypt_key_info.bin				Þ
ì	Mode Control:	C:\eFuse\Mode_control.bin				B
						Download to eFuse
	Encrypt and Sign					
	Product Info:	C:\eFuse\product.json				B
	Random Number:	<ul> <li>Using Random Number</li> </ul>	Select Number			B
	Firmware:					B
						<ul> <li>Encrypt Encrypt and Sign</li> </ul>

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.

e e	Programmer								- c	) ×
•	Encrypt & Sign									×
9	eFuse Settings									
	Name:	test			ID:	1111				
ا	Firmware Key:	<ul> <li>Using Random Key</li> </ul>	O Select Key							в
	Security Mode:	● Open ◯ Close			SWD:	● Open   Cl	ose			
'-'	✓ Batch eFuse:	3								
0	Only Data Key is different	between batch eFuse files.							Generate eFuse	File
-	Download									
	<ul> <li>Encrypt Key Info:</li> </ul>	C:\eFuse\Encrypt_key_info.bin								Þ
i	✓ Mode Control:	C:\eFuse\Mode_control.bin								Þ
									Download to eF	use
	Encrypt and Sign									
	Product Info:	C:\eFuse\product.json								Þ
	Random Number:	<ul> <li>Using Random Number</li> </ul>	🔘 Select Num	ber						в
	Firmware:									B
								<ul> <li>Encrypt</li> </ul>	Encrypt and S	Sign

Figure 3-27 Downloading files to eFuse

#### 🛄 Note:

eFuse information cannot be repeatedly downloaded to firmware.

## 3.7.3 Encrypt & Sign

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

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

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

Encrypt and Sign				
Product Info:	C:\eFuse\product.json			Þ
Random Number:	<ul> <li>Using Random Number</li> </ul>	Select Number		
Firmware:	C:\firmware\ble_app_ancs_fw.bin;	:C:\firmware\ble_app_bps_fw.	bin;C:\firmware\ble_app_hrs_fw.bin;C:\firmware\ble_app_hts_fw.bin	/
			Encrypt     Encrypt	t and Sign

Figure 3-28 Adding more than one firmware file

To encrypt and sign the firmware, check the **Encrypt** box, and the button changes from **Sign** to **Encrypt and Sign**; to sign the firmware only, 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:

ble\_app\_ancs\_fw\_encryptandsign.bin
 ble\_app\_bps\_fw\_encryptandsign.bin
 ble\_app\_hrs\_fw\_encryptandsign.bin
 ble\_app\_hts\_fw\_encryptandsign.bin
 random.bin



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.

9	GProgrammer						- 🗆 ×
	eFuse Lay	out					Ø
	Offset	Name		Value		Length	Comments
÷	0x015E	Product ID		00 00		2	product identity
₽	0x0158	Chip ID		00 00 00 00 00 00 00		6	chip identity
	0x0152	EncMode		00 00		2	encrypted or not
171	0x0150	SWDDisable		00 00		2	enable SWD or not
	∧ 0x014C	Config		00 00 00 00		4	chip configuration
_	0x013C	Chip UID		54 53 4D 14 04 50 52 59 57 34 38 30 30	08 6F 22	16	
=	0x012A	ХО		00 00		2	xo offset
1	0x0124	BT_MAC		00 00 00 00 00 00		6	
	0x0112	Package Type		00		1	0:Unused, 1:GR5515RGBD, 2:GR5515GGBD, 3:GR5515IGND, 4:GR5515I0ND, 5:GR5513BEND, 6:GR5515BEND, 7:GR5513BENDU
	∧ 0x0111	IO_LDO_SEL		08		1	
	Name		Value		Comments		Operate
	IO_PWR_S	SRC	1 ~		0: Internal, 1: Externa	al	Write
							Refresh

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  $^{\circ}$  before **Offset** of **Config** or **IO\_LDO\_SEL** to expand the detailed bits, as shown in the figure below. Click  $^{\circ}$  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.



•	GProgrammer					- 🗆 ×
۲	eFuse Lay	out				Ø
Firmware	Offset	Name	Value		Length	Comments
÷	0x015E	Product ID	00 00		2	product identity
۶	0x0158	Chip ID	00 00 00 00 00 00		6	chip identity
	0x0152	EncMode	00 00		2	encrypted or not
171	0x0150	SWDDisable	00 00		2	enable SWD or not
	∨ 0x014C	Config	00 00 00 00		4	chip configuration
	0	upgrade_disable	0		1	
Ε	1	boot_clk	000		3	0: PLL-64MHz, 1: PLL-48MHz, 2: XO-16MHz, 3: PLL-24MHz, 4: PLL-16MHz, 5: PLL-32MHz
•	4	dpad_while_disable	0		1	
	5	rx_sample_delay	00		2	
	7	flash_power_up_delay	0000		4	
	11	spi_mode	00		2	mode 0, 1, 2, 3
	13	clk_fls_ctrl	0000		4	0: 64MHz, 1: 48MHz, 2: 32MHz, 3: 24MHz, 4: 16MHz, 5: 16MHz
	Name	V	alue	Comments		Operate
	IO_PWR_S	SRC	1 ¥	0: Internal, 1: Externa	al	Write
						Refresh

#### 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.



e e	GProgramme	r								- 0	א נ
•	Chip Co	onfiguration	ı					₽			×
Ũ	Init NVD:	S Area	055000					Soctor	. 1		
₽	Paramete	ers	011000					Sectors	. 1		
	All	ID	Parameter Name	Description	Length(Byte)	Value		Value In Chip			
111		arameters								+ 🗇	~
	ROM P	arameters									~
		0xC001	BD_ADDRESS	Device Address	6	01:23:45:67:89:AB					
a		0xC002	DEVICE_NAME	Device Name	4	name					
_		0xC007	LPCLK_DRIFT	Sleep Clock Accuracy	2	500					
A		0xC085	CODED_PHY_500	Prefer LE Coded PHY 500K	1	0x00					
		0xCOB1	RF_XO_OFFSET	XO offset	2	0x0100					
								-			
	Unfinish	ned					Import	Export	Write	Read	All

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  $\square$  screening box in the upper-right corner of the interface.

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

#### Table 3-5 NVDS ROM parameters



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

## 3.9.1 Init NVDS Area

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

Init NVDS Area		
Start Address: 0x 010FF000	Sectors:	1

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

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

Programmer								- 5	ı x
Chip Configuration	on					₽			Ø
Init NVDS Area									
Start Address: 0x	010FF000					Secto	ors: 1		
Parameters									
All ID	Parameter Name	error		×		Value In Chip			
USER Parameters		Can not init NVD	S in the area f	irmware exists.				+ 🛍	~
ROM Parameters									~
0xC001	BD_ADDRESS		ОК						
0xC002	DEVICE_NAME								
0xC007	LPCLK_DRIFT	Sleep Clock Accuracy	2	500					
0xC085	RE XO OFFSET	YO officet	2	0x00					
OXCODI		No onset	6	0.0100					
Unfinished					Imp	ort Export	Write	Read	All
	Programmer Chip Configuration Init NVDS Area Start Address: 0x Parameters All ID USER Parameters 0xC001 0xC002 0xC002 0xC005 0xC085 0xC081 Unfinished	Programmer  Chip Configuration  Init NVDS Area  Start Address: Ox 010FF000  Parameters  All ID Parameter Name USER Parameters  OxC001 BD_ADDRESS OxC002 DEVICE_NAME OxC007 LPCLK_DRIFT OxC085 CODED_PHY_500 OxC081 RF_XO_OFFSET  Unfinished	Programmer         Chip Configuration         Init NVDS Area         Start Address:         Ox       010FF000         Parameters       error         All       ID       Parameter Name         USER Parameters       Can not init NVD         0xC001       BD_ADDRESS       Can not init NVD         0xC002       DEVICE_NANE       Sleep Clock Accuracy         0xC005       CODED_PHY_500       Prefer LE Coded PHY 500K         0xC081       RF_XO_OFFSET       XO offset         Unfinished       Unfinished       Sleep Clock Accuracy	Programmer         Chip Configuration         Init NVDS Area         Start Address:         OX 010FF000         Parameters         error         All ID Parameter Name         USER Parameters         OxC001 BD_ADDRESS         OxC002 DEVICE_NAME         Sleep Clock Accuracy       2         OxC005 CODED_PHY_500         OxC081 RF_XO_OFFSET       XO offset       2         Unfinished	Programmer         Chip Configuration         Init NVDS Area         Start Address: 0x 010FF000         Parameters: 0x 010FF000         Parameters:       0x 010FF000         Parameters:       0x 001         VISER Parameters:       0x C001         0xC001       BD_ADDRESS         0xC002       DEVICE_NAME         0xC005       COBED_PHY_500         0xC005       COBED_PHY_500         0xC001       RF_XO_OFFSET         XO offset       2         0xC011       RF_XO_OFFSET         0xC011       RF_XO_OFFSET         XO offset       2	Programmer         Chip Configuration         Init NVDS Area         Start Address:       Ox 010FF000         Parameters         All ID ParameterName         USER Parameters         OxC001       BD_ADDRESS         OxC002       DEVICE, NAME         OxC003       DEVICE, NAME         OxC004       RD_ADDRESS         OxC005       CODED_PHY_S00         Prefer LE Coded PHY 500K       1         OxC081       RF_XO_OFFSET         XO offset       2         Unfinished       Imprefer	Programmer         Chip Configuration         Init NVDS Area         Start Address:       0x 010FF000         Parameters:       Sect         All       ID         Parameters:       Parameters         OxC001       BD_ADDRESS         OxC002       DEVICE_NAME         OxC0031       RF_X0_OFFSET         X0 offset       2         OxC081       RF_X0_OFFSET         X0 offset       2         Unfinished       Import	Programmer         Chip Configuration         Init NVDS Area         Start Address:       0x 010FF000         Parameters         All       ID         Parameters         OxC001       BD_ADDRESS         OxC002       DEVICE, NAME         OxC002       DEVICE, NAME         OxC002       DEVICE, NAME         OxC002       DEVICE, NAME         OxC003       RE/ND_OFFSET         XD offset       2         OxC003       RE/NO_OFFSET         XD offset       2         Unfinished       Import       Write	Programmer -     Chip Configuration     Init NVDS Area     Start Address:     Mtl   ID   Parameters   All   ID   Value In Chip   USER Parameters   Mc001   BD_ADDRESS   OK   DeVCE_NAME   Okc002   DEVCE_NAME   Steep Clock Accuracy   2   500   Mc003   BJ_ADDRESS   OK   DevCE_NAME   Steep Clock Accuracy   2   Steep Clock Accuracy   2   Mc003   BJ_ADDRESS   OK   DevCE_NAME   Steep Clock Accuracy   2   Mc003   BJ_ADDRESS   OK     Import   Export

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).

# GODIX

- **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)

9	GProgramme	r							- 0	×
•	Chip Co	nfiguration					₽ (			Ø
9	Init NVDS	S Area								
	Start Addr	ess: 0x 10f	f000				Sectors	5: 1		
2	Paramete	ers								
	All	ID	Parameter Name	Description	Length(Byte)	Value	Value In Chip			
'1'	USER P	arameters						+	Ē	^
		0x4001	test1	test1	1	1	01:00			
		0x4003	test3	test3	1	2	1			
-		0x4000			0		01			
		0x4002	test2	test2	1	1	1			
	ROM Pa	arameters								^
i		0xC001	BD_ADDRESS	Device Address	6	01:23:45:67:89:AB	N/A			
		0xC002	DEVICE_NAME	Device Name	4	name	N/A			
		0xC007	LPCLK_DRIFT	Sleep Clock Accuracy	2	500	N/A			
		0xC085	CODED_PHY_500	Prefer LE Coded PHY 500K	1	0x00	N/A			
		0xCOB1	RF_XO_OFFSET	XO offset	2	0x0100	0x008c			
	Unfinish	ed 🔳 Same 📕	Different				Import Export	Write	Read A	AII

Figure 3-36 Read All interface

## 3.9.3 Write

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



- 🗆 ×					ımer	GProgram	9
7)	₽			on	Configurati	Chip	•
					VDS Area	Init N	
	Sectors: 1			10ff000	Address: 0x	Start A	
					neters	Paran	
	Value In Chip	×	Warning	Parameter Name	ID	All	
+ 🛍 🗠		ue will be writen in chip:	The following v		ER Parameters	US US	111
	01:00		0xC002 : name	test1	0x4001		
	1	nunue:	Are you sure to	test3	0x4003		
	01	Cancel	OK		0x4000		-
	1			test2	0x4002		
^					M Parameters	RO	
	N/A	6 01:23:45:67:89:AB	Device Address	BD_ADDRESS	0xC001		U
	N/A	4 name	Device Name	DEVICE_NAME	0xC002	~	
	N/A	2 500	Sleep Clock Accuracy	LPCLK_DRIFT	0xC007		
	N/A	1 0x00	Prefer LE Coded PHY 500K	CODED_PHY_500	0xC085		
	0x008c	2 0x0100	XO offset	RF_XO_OFFSET	0xC0B1		
Read All	Import Export Write			e 📕 Different	nished 🔳 Same	■ Unfi	
	Import Export Write			e Different	inished 🔳 Same	■ Unfi	

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.

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

Add USER Paramete	er	×
ID	0x 4000~40FF	
Parameter Name		
Description		
Туре	Unsigned Integer	,
Length(Byte)	0	
Value		
	● dec ─ hex	
	OK Cancel	

#### Figure 3-38 Adding a user parameter to NVDS

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

#### 🛄 Note:

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



USER Parameters					
	0x4001 e	existed	existed parameter	1	10
RON	Add USER Paramet	ter	×		
	ID	0x 4001	Warning	r ID must be identic	× B
	Parameter Name Description	duplicated duplicated paramter		OSET Parameter 10 must be identica	
	Туре	Unsigned Integer			
Unfin	Length(Byte)	1			
	Value	20			
	● dec ◯ hex				
	OK Cancel				

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.



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

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.

9	GProgram	mer						- 0	×
•	Chip	Configurati	on				<b>F</b>		Ø
9	Init N\	/DS Area							
	Start A	ddress: 0x	10ff000				Sectors: 1		
	Param	eters							
1.71	All	ID	Parameter Name	Warning		×	Value In Chip		
'-'	USE	R Parameters		The selected parameter	ers will be del	eted from chip,		+ 🛍	~
		0x4001	test1	are you sure to contin	ue?		01:00		
		0x4003	test3				1		
_		0x4000		OK	Cancel		01		
	~	0x4002	test2	test2	1	1	1		
	RO	A Parameters							^
i		0xC001	BD_ADDRESS	Device Address	6	01:23:45:67:89:AB	N/A		
		0xC002	DEVICE_NAME	Device Name	4	name	N/A		
		0xC007	LPCLK_DRIFT	Sleep Clock Accuracy	2	500	N/A		
		0xC085	CODED_PHY_500	Prefer LE Coded PHY 500K	1	0x00	N/A		
		0xC0B1	RF_XO_OFFSET	XO offset	2	0x0100	0x008c		
	Unfir Unfir	nished 🔳 Same	e 📕 Different				Import Export Write	Read A	AII



# GODIX

## 🗘 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.

۹	GProgram	mer							- 🗆 ×
۰	Chip	Configurat	ion					₽	7 0
÷	Init N\	/DS Area	-	💽 Open			×		
	Start A	ddress: 0x	10ff000	← → * ↑	Desktop > GProgrammer	✓ ð Search GProgra	mmer P	Sectors:	1
₽	Param	eters		Organize • New fo	lder		01 · 01 0		
		ID	Parameter Na	<ul> <li>OneDrive</li> </ul>	Name	Date	Type	Value In Chip	
III.	USE	R Parameters		This PC	efusejson	2019/10/29 19:14	JSON 文件		+ 前 へ
		0x4001	test1	3D Objects	productjson	2019/10/29 19:14	JSON XII	01:00	
۲		0x4003	test3	Daily Record				1	
		0x4000		Document_publ				01	
Ē		0x4002	test2	Documents Downloads				1	
	ROM	/I Parameters		Music					~
0		0xC001	BD_ADDRESS	Pictures     Software				N/A	
		0xC002	DEVICE_NAM	Videos				N/A	
		0xC007	LPCLK_DRIFT	weekly report	v «		>	N/A	
		0xC085	CODED_PHY_	File	e namei	<ul> <li>NVDS Config (*</li> </ul>	json) 🗸 🗸	N/A	
		0xC0B1	RF_XO_OFFSE			Open	Cancel	0x008c	
	■ Unfir	ished 🔳 Sam	e 📕 Different				Import con	Export V	/rite Read All

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.



٩	GProgram	GProgrammer – 🗆 X			
•	Devic	e Log		Ø	
	ID	CONTENT	as	cii 🔵	
Ŧ	A001	HARDFAULT CALLSTACK INFO: R0-00000000 R1-00000000 R2-00000000 R3-00000000 R12-0000000A LR-01020581 PC-01015FEC XPSR-61000011			
0	A002	HARDFAULT CALLSTACK INFO: R0-00000000 R1-00000000 R2-00000000 R3-00000000 R12-0D2E6465 LR-010058AD PC-01015FEC XPSR-61000011			
2	A003	HARDFAULT CALLSTACK INFO: R0-00000000 R1-00000000 R2-00000000 R3-00000000 R12-0D2E6465 LR-010058AD PC-01015FEC XPSR-61000011			
171	A004				
	A005				
	A006				
-	A007				
B	A008				
	A009				
i	A00A				
	A00B				
	A00C				
	A00D				
	A00E				
	A00F				
	A010				
			1	Read	

#### 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 a or in the upper-right corner to switch the mode in displaying device logs between ASCII and stream.

- Omen: The device logs are displayed by byte stream as shown in Figure 3-45.



Figure 3-44 Device logs in ASCII characters



Devic	e Log
ID	CONTENT Stream
A001	48 41 52 44 46 41 55 4C 54 20 43 41 4C 65 35 44 143 48 20 49 4E 46 4F 3A 20 52 30 2D 30 30 30 30 30 30 30 30 20 52 31 2D 30 30 30 30 30 30 30 20 52 32 2D 30 30 30 30 30 30 30 30 20 52 31 2D 30 30 30 30 30 30 20 52 31 2D 30 30 30 30 30 30 30 30 30 30 30 30 30
A002	48 41 52 44 46 41 55 4C 54 20 43 41 4C 4C 53 54 41 43 48 20 49 4E 46 4F 3A 20 52 30 2D 30 30 30 30 30 30 30 30 20 52 31 2D 30 30 30 30 30 30 30 30 20 52 32 2D 30 30 30 30 30 30 30 30 30 30 30 30 30
A003	48 41 52 44 46 41 55 4C 54 20 43 41 4C 4C 53 54 41 43 48 20 49 4E 46 4F 3A 20 52 30 2D 30 30 30 30 30 30 30 30 20 52 31 2D 30 30 30 30 30 30 30 30 20 52 32 2D 30 30 30 30 30 30 30 30 30 30 30 30 30
A004	



## 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.

Command	Functional Description	Command Format and Parameter Description
program	Programs firmware files to internal SoC flash memories.	<ul> <li>program <firmware file="" path=""> <run immediately:y="" n=""  =""></run></firmware></li> <li>Parameter description:</li> <li><firmware file="" path="">: It sets the path of the to-be-downloaded firmware file.</firmware></li> <li><run immediately:y="" n=""  ="">: It decides on whether to run the firmware immediately after downloading.</run></li> </ul>
erase	Erases flash memory data within an SoC based on a specified address range.	<ul> <li>erase <start address<hex="">&gt; <end address<hex="">&gt;<force erase="" li="" when<=""> <li>conflict with firmware/bootinfo:y   n&gt;</li> <li>Parameter description:</li> <li><start address<(hex)="">&gt;: It represents the start address of the storage area to be erased (in hexadecimal).</start></li> </force></end></start></li></ul>

#### Table 3-6 GR551x\_console supported commands



Command	Functional Description	Command Format and Parameter Description
		<ul> <li><end address<(hex)="">&gt;: It represents the end address of the storage area to be erased (in hexadecimal).</end></li> <li><force bootinfo:y="" conflict="" erase="" firmware="" n="" when="" with=""  ="">: This parameter decides whether to forcibly erase the flash memory data when its address conflicts with that of firmware, Boot info, or NVDS.</force></li> </ul>
eraseall	Erases all flash memory data within an SoC.	eraseall
download	Downloads data files to internal SoC flash memories.	<ul> <li>download <data file="" path=""> <start address<(hex)="">&gt; <force download<="" li=""> <li>when conflict with firmware/bootinfo:y   n&gt;</li> <li>Parameter description: <ul> <li><data file="" path="">: It sets the path of the to-be-downloaded data file.</data></li> <li><start address<(hex)="">&gt;: It represents the start address of the download area (in hexadecimal).</start></li> <li><force bootinfo:y="" conflict="" download="" firmware="" n="" when="" with=""  ="">: 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.</force></li> </ul> </li> </force></start></data></li></ul>
writeefuse	Writes Encrypt Key Info and Mode Control files to eFuse.	<ul> <li>writeefuse <encrypt file="" info="" key="" path=""> <mode control="" file="" path=""></mode></encrypt></li> <li>Parameter description:</li> <li><encrypt file="" info="" key="" path="">: It sets the path of Encrypt Key Info file.</encrypt></li> <li><mode control="" file="" path="">: It sets the path of Mode Control file.</mode></li> </ul>
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.)

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.
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.

Table 3-7 Frequently used parameters for GR551x\_encrypt\_signature.exe

# GODIX

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_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\_encryptAndS ign\firmware\_encryptAndSign.bin.

program\_Firmware\_EncryptAndSign.bat can erase all internal flash memories, and download the firmware firmw are\_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: <u>www.goodix.com</u>.