GR5xx固件升级开发指南

版本： **1.6**

发布日期： **2024-09-24**

深圳市汇顶科技股份有限公司

商标声明

GOODiX 和其他汇顶商标均为深圳市汇顶科技股份有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人持有。

免责声明

本文档中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。

深圳市汇顶科技股份有限公司（以下简称“GOODIX”）对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。GOODIX对因这些信息及使用这些信息而引起的后果不承担任何责任。

未经GOODIX书面批准，不得将GOODIX的产品用作生命维持系统中的关键组件。在GOODIX知识产权保护下，不得暗中或以其他方式转让任何许可证。

深圳市汇顶科技股份有限公司

总部地址：深圳市福田保税区腾飞工业大厦B座13层

电话：+86-755-33338828　　邮编：518000

网址：www.goodix.com

前言

编写目的

本文档介绍设备固件升级（Device Firmware Upgrade，DFU）原理、GR5xx DFU方案设计、App bootloader应用，以及通过GRToolbox（Android）APP和DFU Master方式进行固件升级的方法，帮助用户快速理解使用GR5xx DFU方案。

读者对象

本文适用于以下读者：

- 芯片用户
- 开发人员
- 测试人员
- 技术支持人员

版本说明

本文档为第7次发布，对应的产品系列为低功耗蓝牙GR5xx系列。

修订记录

版本	日期	修订内容
1.0	2023-04-20	首次发布。
1.1	2023-05-10	更新“支持平台”、“DFU Master跨平台移植”章节。
1.2	2023-08-22	更新“支持平台”、“ 非加密非加签固件升级 ”、“ 外部Flash资源升级”和“ UART方式 ”章节。
1.3	2023-09-24	更新“支持平台”。
1.4	2023-11-06	更新GProgrammer、GRUart、GRToolbox获取方式。
1.5	2024-02-27	更新“支持平台”。
1.6	2024-09-24	• 更新“支持平台”。 • 更新UART升级方式的硬件配置。 • 更新App bootloader工程结构描述。 • 更新“Operate System Info”、“Program Start”及“Program End”命令的发送数据格式。

目录

前言 I
1 简介 1
1.1 DFU通讯方式 1
1.2 DFU工作模式 1
2 DFU方案设计 4
2.1 后台双区升级模式 4
2.1.1 Flash布局 4
2.1.2 固件下载流程 5
2.1.3 App bootloader启动流程 6
2.2 非后台单区升级模式 7
2.2.1 Flash布局 7
2.2.2 固件下载流程 7
2.2.3 App bootloader启动流程 8
2.3 升级速率比较 9
2.4 固件格式 11
3 App bootloader工程介绍 12
4 使用GRToolbox升级 16
4.1 支持平台 16
4.2 准备工作 16
4.3 非加密非加签固件升级 17
4.3.1 固件配置 17
4.3.2 固件烧录 18
4.3.3 创建目标升级固件 19
4.3.4 进入GRToolbox升级界面 21
4.3.5 固件升级 22
4.3.5.1 后台双区升级模式 22
4.3.5.2 非后台单区升级模式 23
4.4 加密加签固件升级 24
4.4.1 eFuse设置 25
4.4.2 eFuse下载 27
4.4.3 固件配置 28
4.4.4 生成加密加签固件 28
4.4.5 固件升级 28
4.5 仅加签非加密固件升级 29
4.5.1 固件配置 29
4.5.2 生成仅加签非加密固件 29

4.5.3 固件升级.....29
4.6 资源升级.....30
4.6.1 内部Flash资源升级.....30
4.6.2 外部Flash资源升级.....31
5 DFU移植方式详解.....36
6 使用DFU Master升级.....45
6.1 DFU Master介绍.....45
6.2 DFU Master跨平台移植.....45
6.3 DFU Master升级操作.....50
6.3.1 准备工作.....50
6.3.2 UART方式.....50
6.3.3 低功耗蓝牙方式.....53
7 注意事项.....55
7.1 在App bootloader跳转至App firmware之前，需将App bootloader中使用的外设反初始化.....55
7.2 不同系列芯片使用RTOS时，应用固件DFU任务栈大小设置不同.....55
8 附录：DFU通信协议.....56
8.1 基础帧定义.....56
8.1.1 帧结构定义.....56
8.1.2 字节顺序定义.....56
8.2 附录：DFU命令集.....56
8.2.1 Get Info命令.....57
8.2.1.1 主机端发送数据.....57
8.2.1.2 设备端回应数据.....57
8.2.2 Operate System Info命令.....58
8.2.2.1 主机端发送数据.....58
8.2.2.2 设备端回应数据.....58
8.2.3 DFU Mode Set命令.....59
8.2.3.1 主机端发送数据.....59
8.2.3.2 设备端回应数据.....59
8.2.4 DFU Firmware Info Get命令.....59
8.2.4.1 主机端发送数据.....59
8.2.4.2 设备端回应数据.....60
8.2.5 Program Start命令.....60
8.2.5.1 主机端发送数据.....60
8.2.5.2 设备端回应数据.....61
8.2.6 Program Flash命令.....61
8.2.6.1 主机端发送数据.....62
8.2.6.2 设备端回应数据.....62
8.2.7 快速模式写入固件.....62
8.2.7.1 主机端发送数据.....62

8.2.7.2 设备端回应数据 63
8.2.8 Program End命令 63
8.2.8.1 主机端发送数据 63
8.2.8.2 设备端回应数据 63
8.2.9 配置外部Flash命令 64
8.2.9.1 主机端发送数据 64
8.2.9.2 设备端回应数据 65
8.2.10 获取Flash信息命令 65
8.2.10.1 主机端发送数据 65
8.2.10.2 设备端回应数据 66

1 简介

设备固件升级（Device Firmware Upgrade，DFU）是一种引导加载机制，用于更新目标设备的固件，以修复产品缺陷，丰富产品功能。

本文档主要介绍DFU的工作原理、GR5xx DFU方案设计及固件升级步骤。

进行操作前，可参考以下文档。

表 1-1 文档参考

名称	描述
对应芯片开发者指南	介绍对应芯片的SDK以及基于SDK的应用开发和调试
GProgrammer用户手册	GProgrammer软件的操作说明，包括固件下载、固件加密等
J-Link用户指南	J-Link使用说明：http://www.segger.com/downloads/jlink/UM08001_JLink.pdf
Keil用户指南	Keil详细操作说明：http://www.keil.com/support/man/docs/uv4/
Bluetooth Core Spec	Bluetooth官方标准核心规范

1.1 DFU通讯方式

DFU的通讯方式支持无线和有线两种：

- 无线方式：又称OTA（Over The Air） DFU，即以空中升级的方式实现固件升级，通常简称为OTA。2G、3G、4G、WiFi及蓝牙等无线方式都可用于设备固件升级。
- 有线方式：主要包括UART、USB或SPI等。

1.2 DFU工作模式

- 无论采用OTA或有线升级方式，DFU方案可分两种类型：后台式DFU和非后台式DFU。
  - 后台模式：由应用程序接收下发的固件，且应用程序在执行接收固件任务时，还可执行其他任务。
  - 非后台模式：由Bootloader程序接收下发的固件，且当前只能执行下载固件这一个任务。
- 从占用的存储区域看，DFU又可分为两种类型：双区DFU和单区DFU。
  - 双区DFU：将接收到的固件先缓存到一个区域，待固件校验通过后，再拷贝至目标区域。
  - 单区DFU：将接收到的固件写入到目标区域，直接覆盖原始固件。

后台式DFU必须使用双区模式存放新旧固件，非后台式DFU既可使用单区也可使用双区模式来存放新旧固件。因此，常用的DFU工作模式主要分为三种：

- 后台双区升级模式
- 非后台双区升级模式
- 非后台单区升级模式

三种模式详细介绍如下：

- 后台双区升级模式的升级示意图如图 1-1所示。其升级过程主要包含以下步骤：

  1. 在应用程序部分接收主机端下发的固件。
  2. 由应用程序将接收到的固件写入至Bank1区域。
  3. 固件写入完毕后，对Bank1区域的固件进行校验，校验通过后，程序跳转至Bootloader固件运行。
  4. 由Bootloader程序将新固件拷贝至Bank0区域。然后对新固件进行校验，若校验通过，则跳转至新固件运行。

说明:

设备端指固件端，如果使用手机APP升级，主机端即指手机APP。

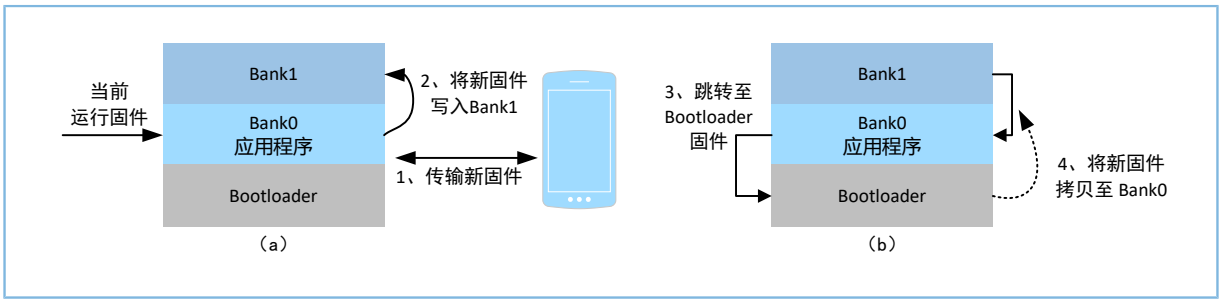

图 1-1 后台双区升级模式示意图

- 非后台双区升级模式的升级示意图如图 1-2所示。其升级过程主要包含以下步骤：

  1. 在Bootloader接收主机端下发的固件。
  2. 由Bootloader将接收的固件写入Bank1区域。
  3. 固件写入完毕后，先对Bank1区域的新固件进行校验，校验通过后，由Bootloader将新固件拷贝至Bank0区域，然后再对Bank0区域的新固件进行校验，若校验通过，则跳转至新固件运行。

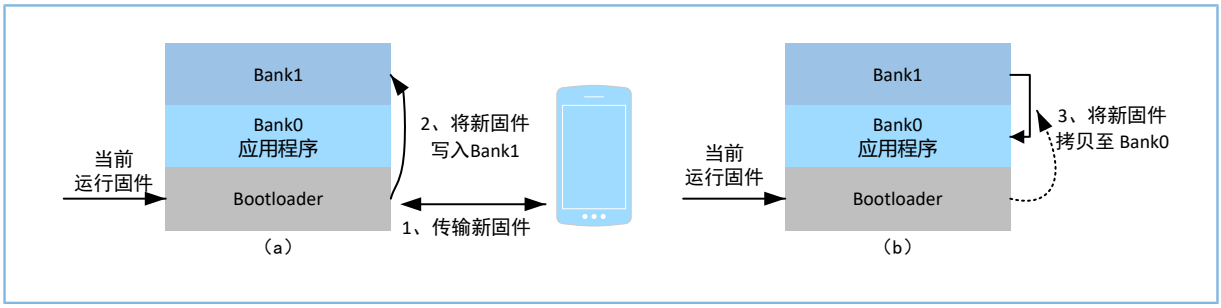

图 1-2 非后台双区升级模式示意图

- 非后台单区升级模式的升级示意图如图 1-3所示。由Bootloader端固件接收手机APP下发的固件，在接收固件的同时直接将新固件写入Bank0，当写入固件完毕后，对Bank0区域的固件进行校验，当校验通过后跳转至Bank0应用程序运行。

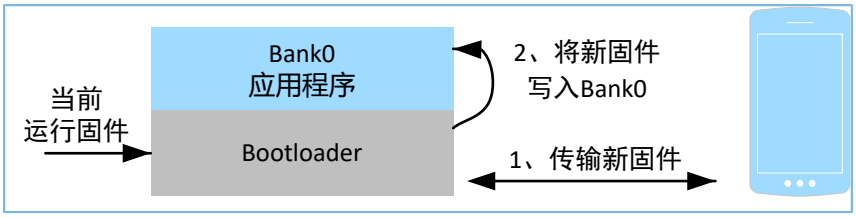

图 1-3 非后台单区升级模式示意图

三种不同的DFU模式比较如下表所示。

表 1-2 不同DFU模式比较

DFU模式	升级相同固件时Flash占用	备份机制	Bootloader固件大小	升级过程中执行其他任务
后台双区升级	大	支持	小	支持
非后台单区升级	小	不支持	大	不支持
非后台双区升级	大	支持	大	不支持

说明:

- 后台双区升级模式和非后台双区升级模式在升级相同固件时，Flash占用大是因为需要两块Flash区域存放新旧固件。
- 备份机制指设备端接收的新固件损坏时，系统可继续运行旧固件。
- 非后台单区升级的Bootloader固件比其他两种模式小是因为其只需要基本的拷贝和跳转功能即可，不需要具备与主机端进行数据交互的功能。
- 在非后台单区升级和非后台双区升级时，设备需要跳转至Bootloader固件，升级过程中无法执行应用程序的其他任务。相比之下，后台双区升级模式可以在升级过程中继续执行其他应用任务，提高用户体验。

由表 1-2 可知，非后台双区升级模式具备的优点后台双区升级模式也有，非后台单区升级模式具备的优点是其他两种模式没有的。因此，GR5xx提供的DFU升级模式，只包含后台双区升级模式和非后台单区升级模式。

在实际升级过程中，使用何种升级模式需要根据Flash存储空间和待升级固件大小来决定，具体原则为，如果（全部Flash空间 - Bootloader固件大小 - 参数存储空间）/ 2 >= 待升级固件大小，则推荐使用后台双区升级模式；否则，推荐使用非后台单区升级模式。

说明:

“参数存储空间”指在Flash区域划分的用于存放非易失性数据的区域。

2 DFU方案设计

本章介绍GR5xx提供的后台双区升级模式和非后台单区升级模式的设计。

2.1 后台双区升级模式

2.1.1 Flash布局

后台双区升级模式的Flash布局设计如下所示。

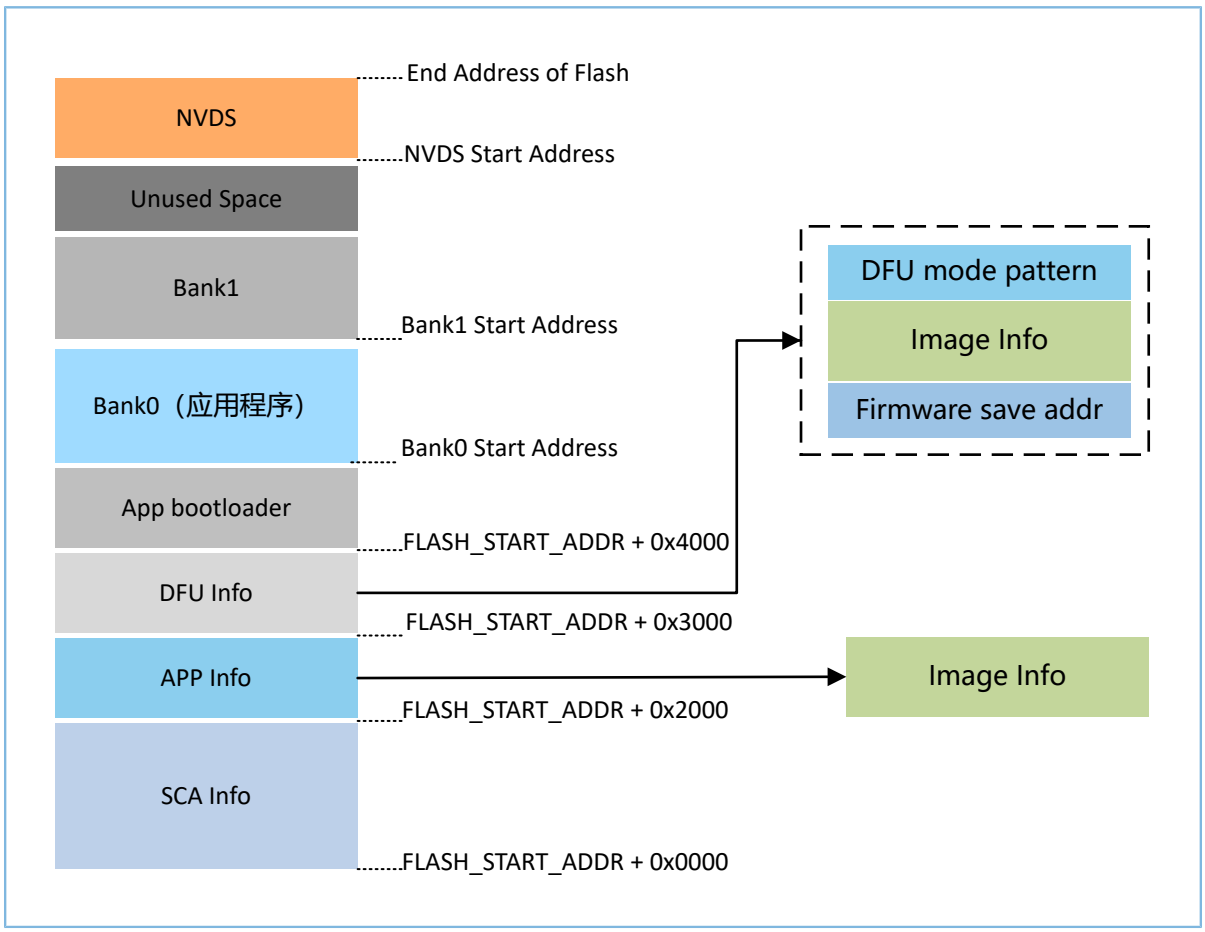

图 2-1 Flash布局设计

- SCA Info：SCA（System Configuration Area）系统配置区，主要用于存储系统信息和App bootloader的启动参数配置信息。
- APP Info：应用固件信息区，用于存放Bank0区域应用固件运行的参数信息。
- DFU Info：DFU固件信息区，存放Bank1区域新固件的相关信息。
  - Firmware save addr：新固件存放的起始地址。
  - Image Info：新固件的参数信息。
  - DFU mode pattern：用于标识当前运行的DFU模式。
- App bootloader：App bootloader固件存放和运行区域。

- Bank0：应用固件存放和运行区。
- Bank1：新固件缓存区，通过有效性检查的新固件将被拷贝至Bank0。
- NVDS（Non-volatile Data Storage）：非易失性数据存储区域。

2.1.2 固件下载流程

如图 2-2所示，对于后台双区升级模式，在接收下发的固件时，当前运行的是位于Bank0区域的应用程序。

1. 由Bank0应用程序接收主机端下发的固件。
2. 由Bank0应用程序将接收到的固件数据写入至Bank1区域。
3. 当固件全部写入完毕后，将新固件的固件信息（图 2-1中的Image Info）、新固件存放区域的起始地址（图 2-1中的Firmware save addr）、当前执行的DFU模式（图 2-1中的DFU mode pattern）更新至DFU Info区域。
4. DFU Info区域更新完毕后，复位设备。
5. 复位后运行App bootloader固件，然后按图 2-3的启动流程运行。

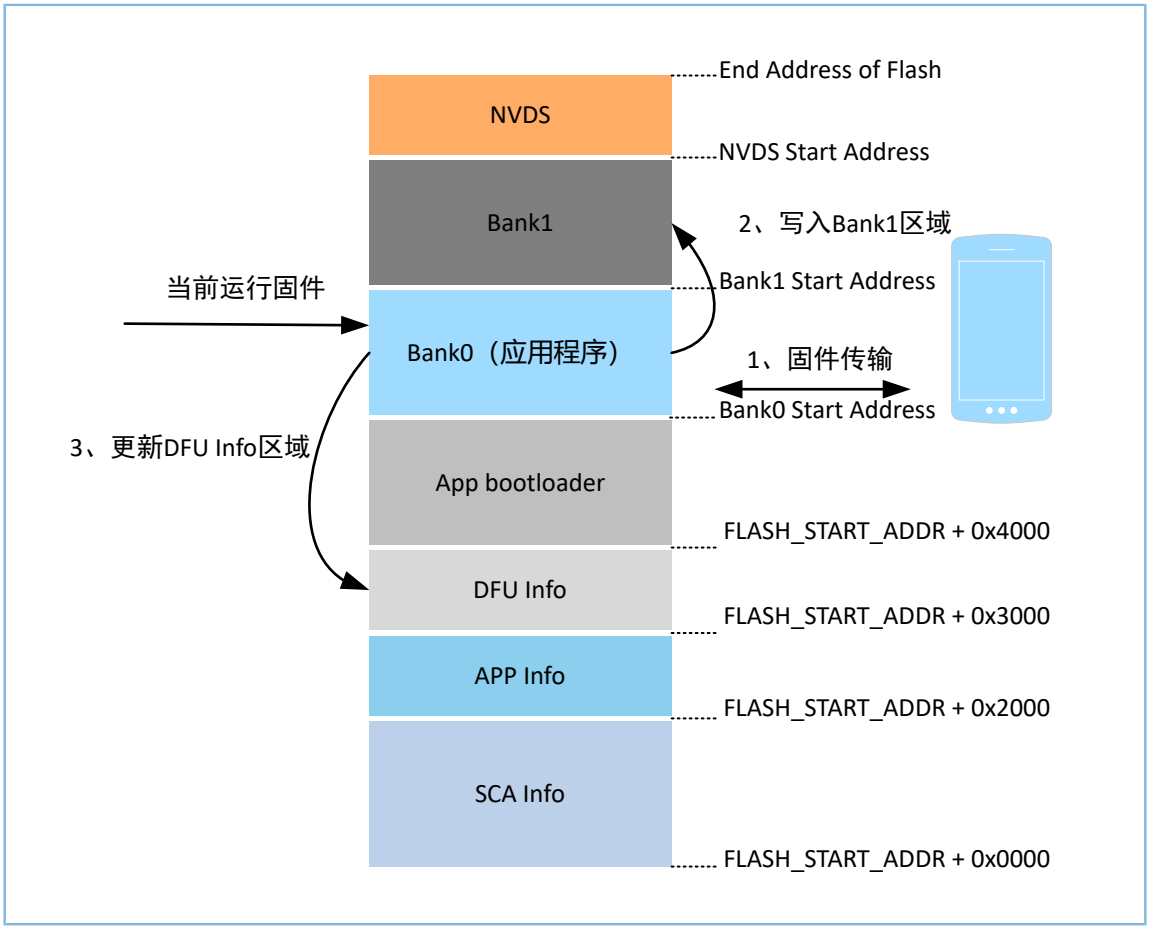

图 2-2 固件下载流程

2.1.3 App bootloader启动流程

设备复位后，App bootloader固件会根据Bank0应用程序更新的DFU Info完成固件拷贝及校验，最后跳转至应用固件运行，详细启动流程如图 2-3所示。

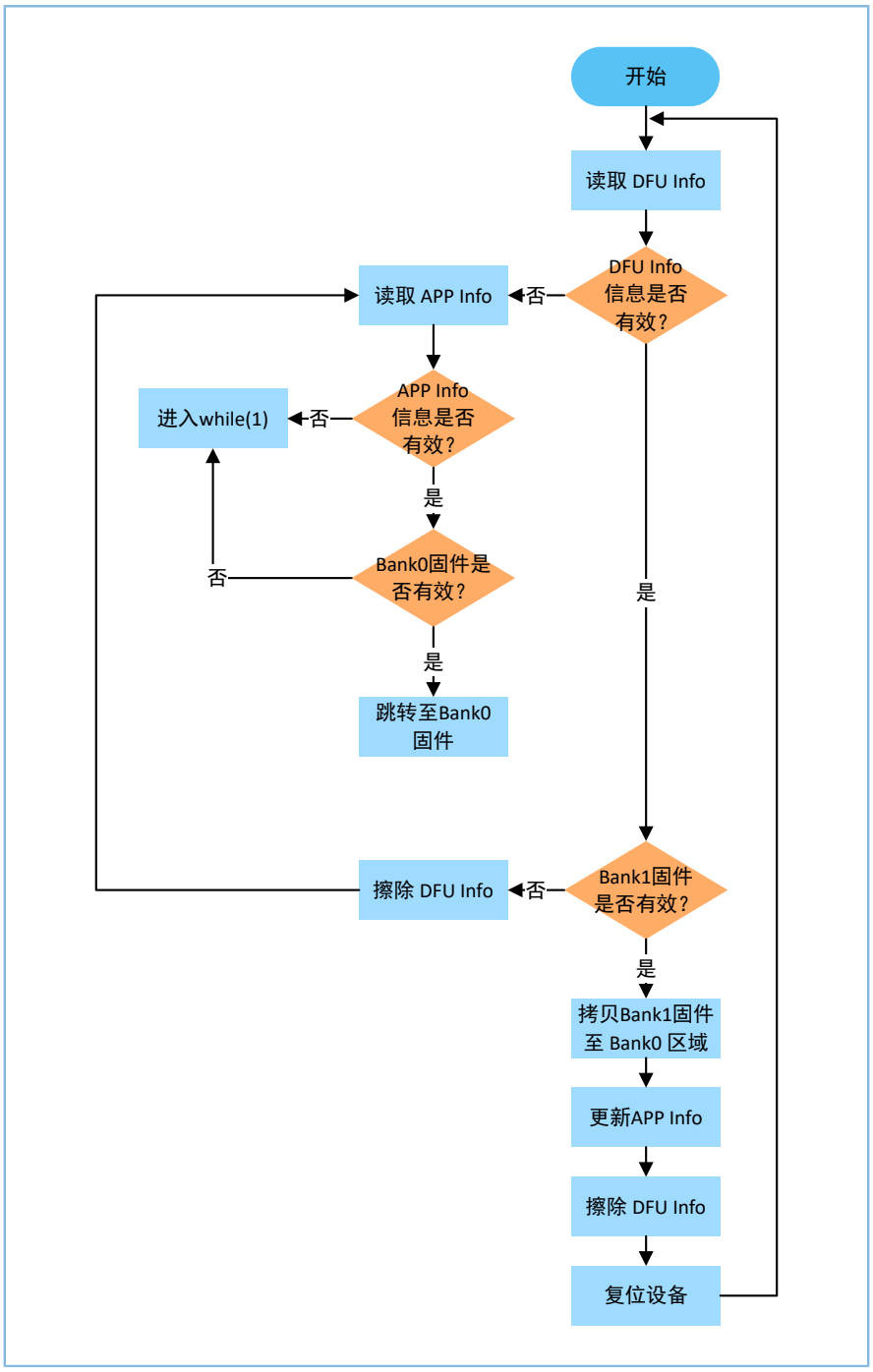

图 2-3 App bootloader启动流程

说明:

- DFU Info信息是否有效：指当前DFU Info区域是否有数据。
- APP Info信息是否有效：指当前APP Info区域是否有数据。

2.2 非后台单区升级模式

2.2.1 Flash布局

非后台单区升级模式的Flash布局如下所示。

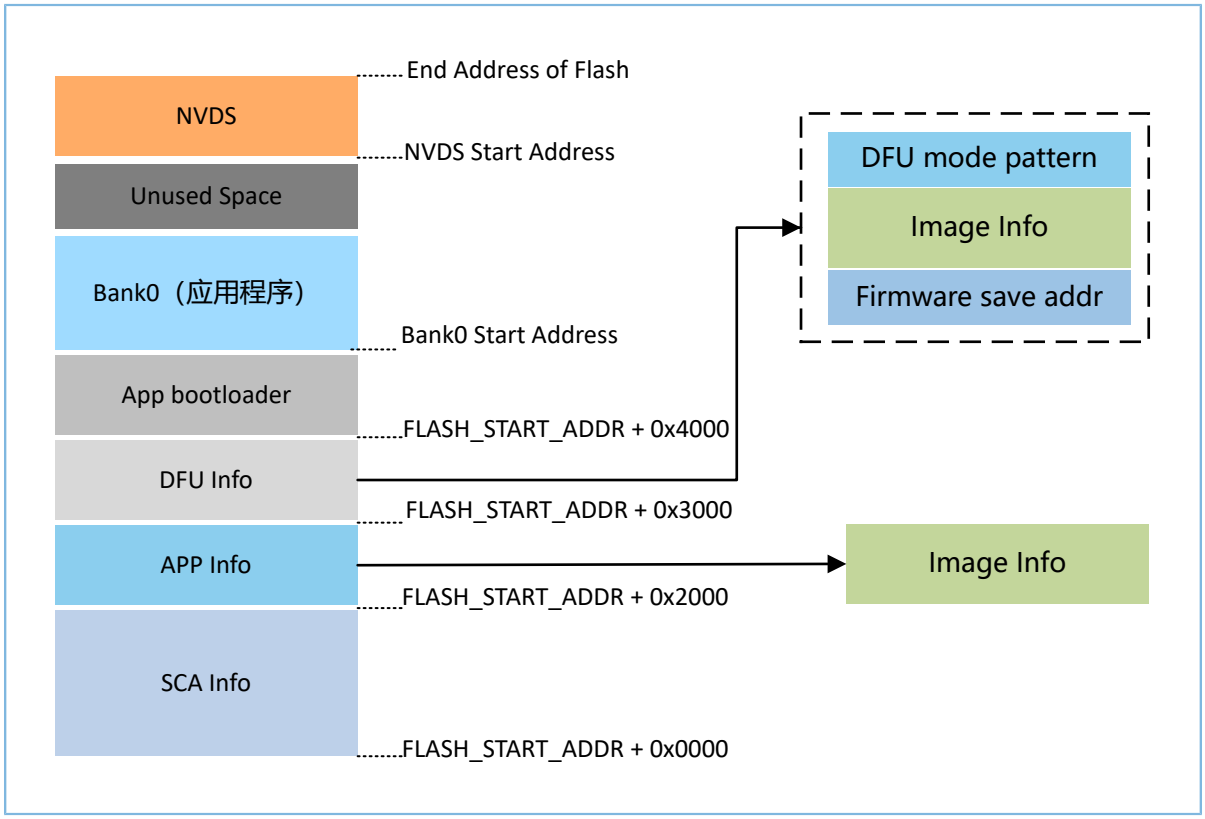

图 2-4 Flash布局

其Flash布局相比于后台双区升级模式，仅没有Bank1区域，而其他区域是一致的。因此各区域的含义请参考2.1.1 Flash布局。

2.2.2 固件下载流程

非后台单区升级模式有两种情况，分别为有Bank0固件和无Bank0固件。

- 如图 2-5所示，有Bank0固件的场景下，固件下载流程为如下步骤：

  1. 当前程序运行在Bank0固件，由Bank0固件接收主机端下发的升级模式命令。
  2. Bank0固件接收到主机端下发的非后台单区升级模式后，将升级模式写入DFU Info区域，然后复位设备。
  3. 复位设备后，当前程序运行于App bootloader固件，在App bootloader固件接收下发的固件。
  4. 由App bootloader固件将下发的新固件写入Bank0区域，然后将新固件信息写入APP Info区域，并擦除DFU Info区域数据，最后复位设备。

  复位后，即按照图 2-6的启动流程运行。

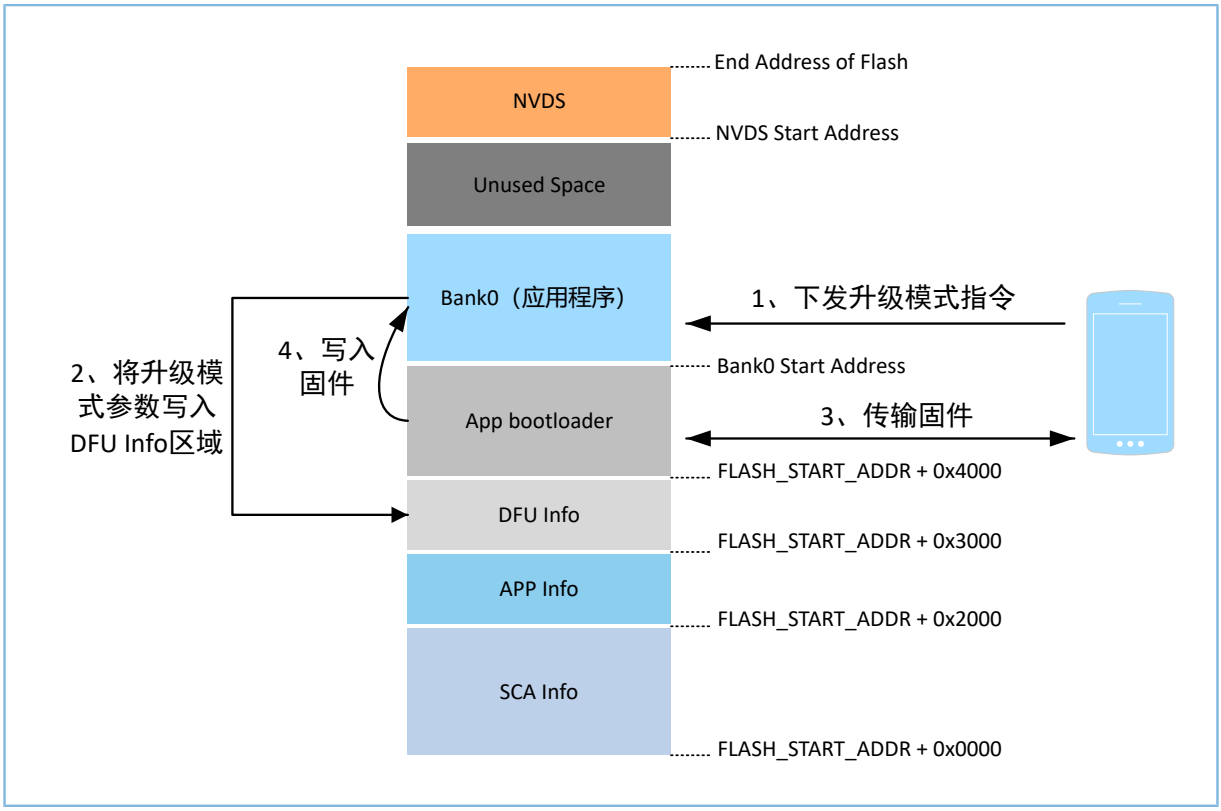

图 2-5 固件下载流程

在上述操作中，主机端最开始连接的是Bank0固件，当下发非后台单区升级指令后，设备会进行复位，此时手机APP需要自动重连至App bootloader固件。为了使手机能够准确连接至App bootloader固件，GR5xx提供的方案是：假设Application的蓝牙设备地址为x，跳转至Bootloader后蓝牙设备地址会变成x+1，这样手机即可通过“地址+1”的方式自动连接至App bootloader固件。

- 无Bank0固件的场景下，由App bootloader接收升级模式命令，将升级模式写入DFU Info区域，整个过程中蓝牙无需重连。其余流程和有Bank0固件的场景一致。

2.2.3 App bootloader启动流程

因非后台单区升级模式没有Bank1区域用于缓存新固件，所以DFU Info区域仅用于存储DFU升级模式参数。App bootloader启动流程如下：

1. 读取到非后台单区升级模式后，开始升级。
2. 固件下载完成后，将新固件的固件参数信息更新至APP Info，然后复位。
3. 复位设备后，读取APP Info区域的数据，然后根据APP Info区域的信息跳转至应用固件运行。

流程如下图所示。

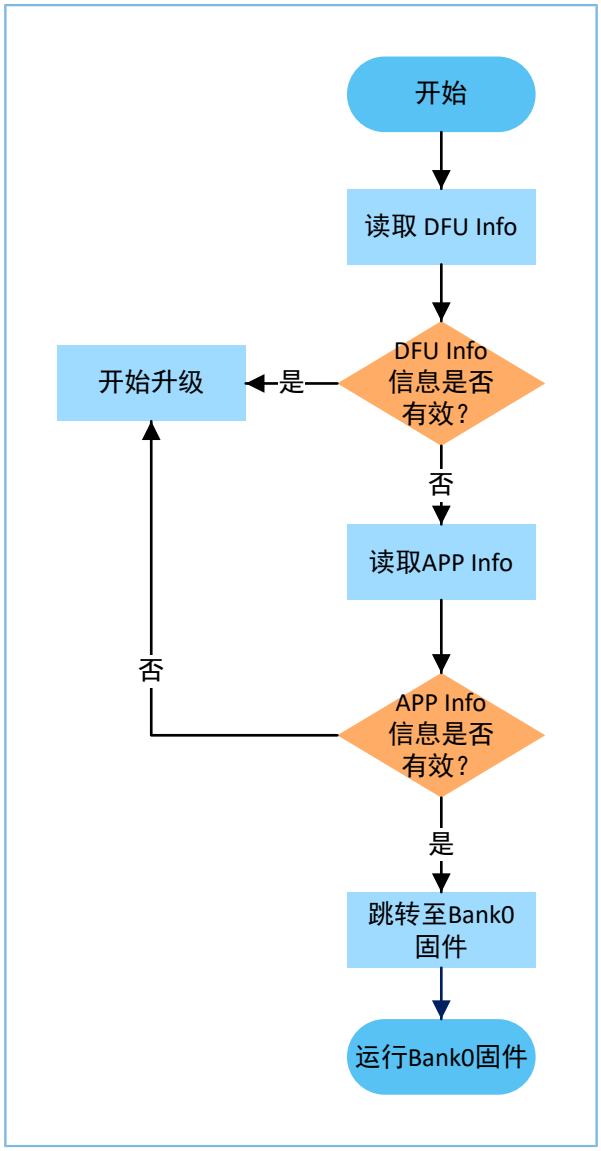

图 2-6 App bootloader启动流程

2.3 升级速率比较

在固件升级速度方面，GR5xx提供普通模式和快速模式两种，快速模式的升级速度高于普通模式。两者的对比如下表所示。

表 2-1 普通模式与快速模式比较

模式	固件传输方式	RAM占用	耗时	说明
普通	每传输一帧数据，设备端都会回复一帧当前数据的校验值。	需要2 KB Buffer接收固件	多	在固件尚未发送完时如果出现一帧数据错误，可立即发现并终止升级流程。
快速	所有固件写入完毕再回复校验值。	最大带宽配置下需要8 KB	较少	升级大固件时可显著缩短升级时间，但不存在每一帧数据的校验，固件传输完成才会进行校验。

模式	固件传输方式	RAM占用	耗时	说明
		Buffer接收固件		

在最大带宽的情况下，快速模式的升级速度可达到普通模式的6倍左右，虽然理论上快速模式在传输过程中可能会出现错误，在最后校验过程中才会被发现，出错后时间成本比较高。但在实际测试过程中，出错的概率较低，因此在升级体积较大的固件时，可使用快速模式，提升效率。

快速模式的引入也导致了RAM占用的增加，因此在某些应用场景下，留给固件升级的RAM空间非常有限，使用快速模式可能导致系统RAM空间不足，只能使用普通模式进行升级。在这种RAM空间紧缩的场景中，固件端需要进行相关配置，以尽量减少RAM占用。配置文件路径位于`SDK_Folder\components\libraries\dfu_port\dfu_port.h`，配置内容如下，基于此配置，可使RAM占用最少。

```
#define ONCE_WRITE_DATA_LEN              1024
#define DFU_BUFFER_SIZE                  2048
```

快速模式和普通模式的数据交互原理如下图所示。

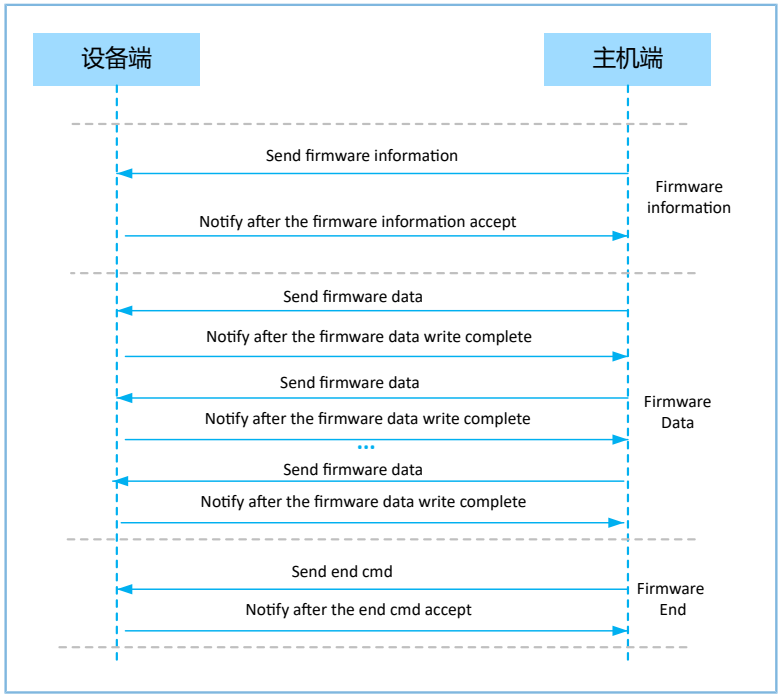

图 2-7 普通模式

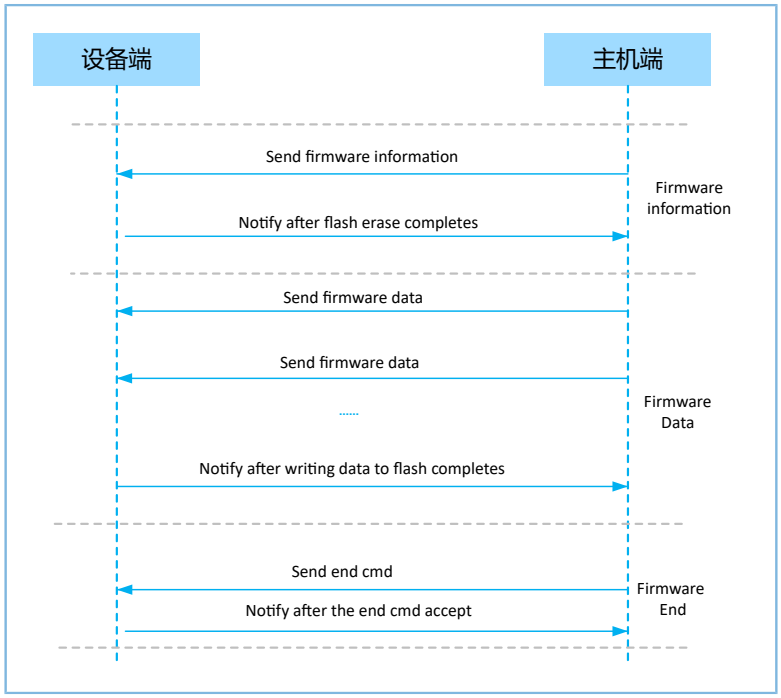

图 2-8 快速模式

普通模式主机端每发送一帧固件数据，设备端就需要回复一帧，而快速模式只有在数据全部发送完成之后才会进行回复，所以快速模式比普通模式速率高。

2.4 固件格式

从安全角度考虑，传输固件时，需对固件进行加签或者加密。因此GR5xx提供三种固件格式：非加密非加签固件、加密加签固件和仅加签固件。三种固件的bin格式如下图所示。

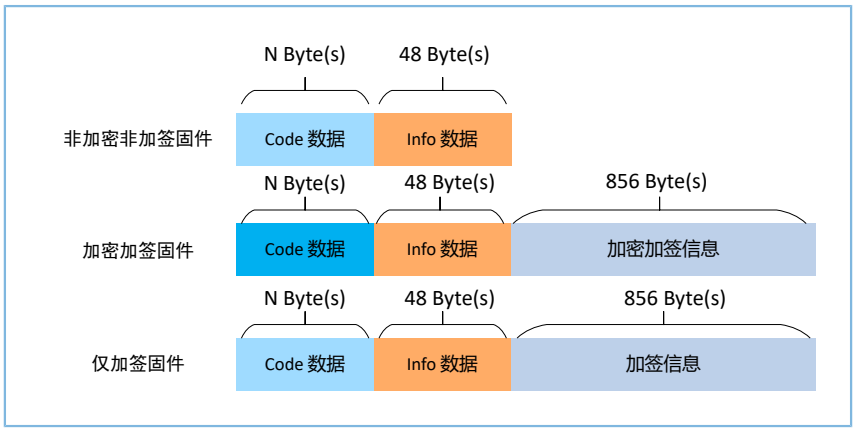

图 2-9 固件格式

数据格式各字段说明如下所示：

- Code数据：固件本身数据，需要16字节对齐，N表示长度可变。
- Info数据：固件描述信息，即2.1.1 Flash布局中的Image Info信息。
- 加密加签信息：将非加签非加密固件变成加密加签固件所需要的信息。
- 仅加签信息：将非加签非加密固件变成仅加签固件所需要的信息。

3 App bootloader工程介绍

App bootloader工程位于SDK_Folder\projects\ble\dfu\app_bootloader，工程目录结构如下图所示。

说明:

SDK_Folder为对应芯片SDK的根目录。

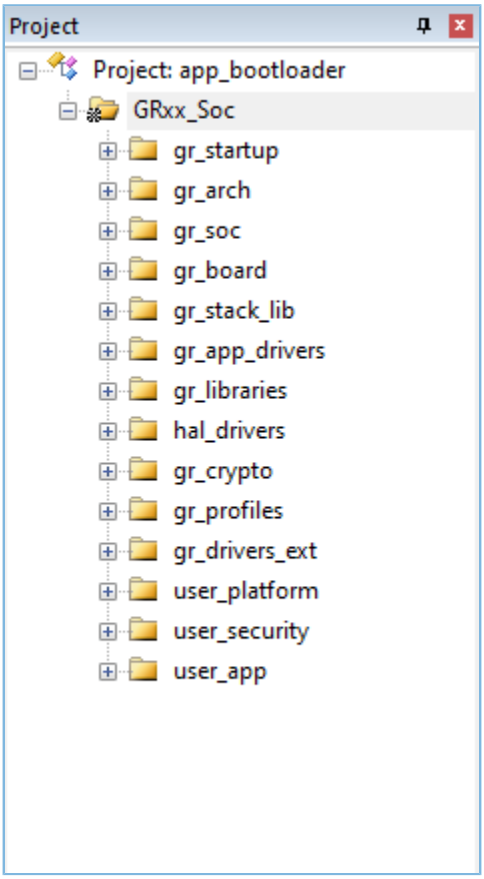

图 3-1 工程结构

说明:

仅GR5405 SDK中的App bootloader工程包含 hal_drivers目录。

各Group下存放的文件及功能描述如下表所示。

表 3-1 App bootloader工程Group描述

Group	描述
gr_startup	汇编启动文件
gr_arch	芯片架构方面的文件
gr_soc	SoC初始化相关文件
gr_board	Starter Kit开发板相关初始化文件

Group	描述
gr_stack_lib	协议栈Lib文件
gr_app_drivers	App driver相关文件
gr_libraries	相关组件文件，包括DFU组件
hal_drivers	HAL层驱动文件
gr_crypto	加密算法文件
gr_profiles	OTA profile相关文件
user_platform	用户初始化平台的相关文件
user_security	加密加签的相关文件
user_app	App bootloader工程功能相关文件

在后台双区升级模式和非后台单区升级模式的整个升级过程中，均需使用App bootloader固件，App bootloader固件的功能主要包含以下四方面：

- 启动应用固件
- 固件验签及Checksum校验
- 固件下载（从主机端接收固件）
- 固件烧录

由于不同的升级模式需要的功能不同，为了避免 App bootloader 功能冗余，提供相关宏以对App bootloader的功能进行裁剪。App bootloader详细配置项如下表所示。

说明:

- *bootloader_config.h*文件路径：`SDK_Folder\projects\ble\dfu\app_bootloader\Src\config`
- *custom_config.h*文件路径：`SDK_Folder\projects\ble\dfu\app_bootloader\Src\config`

表 3-2 App bootloader配置项

文件名	宏	描述
bootloader_config.h	BOOTLOADER_DFU_BLE_ENABLE	• 0：关闭DFU的低功耗蓝牙通讯方式。 后台双区升级模式仅需要App bootloader具备基本跳转和拷贝功能，可将该宏关闭。 • 1：打开DFU的低功耗蓝牙通讯方式。 非后台单区升级模式需要App bootloader接收下发的固件信息，使用蓝牙升级需要将该宏打开。
	BOOTLOADER_DFU_UART_ENABLE	• 0：关闭DFU的UART通讯方式。 后台双区升级模式可关闭该宏。 • 1：打开DFU的UART通讯方式。 非后台单区升级模式在使用UART升级时，需要将该宏打开。

文件名	宏	描述
	BOOTLOADER_DFU_ENABLE	该宏使用默认配置即可，表示App bootloader是否使用DFU功能。该宏的值取决于上述两个宏BOOTLOADER_DFU_BLE_ENABLE和BOOTLOADER_DFU_UART_ENABLE的值。
	BOOTLOADER_WDT_ENABLE	是否使能看门狗。 • 0：关闭看门狗功能 • 1：打开看门狗 在程序奔溃宕机后，可通过喂狗的方式复位系统，使系统重新运行起来。如果系统存在异常宕机的风险，需要开启该宏。
	BOOTLOADER_SIGN_ENABLE	当升级的固件是加密固件或加签固件时，需要打开该宏，使能App bootloader对升级固件的验签功能。 • 0：关闭验签功能 • 1：打开验签功能
	BOOTLOADER_PUBLIC_KEY_HASH	开启验签后，需配置 public_key_hash 值，才能正确完成App bootloader 固件对新固件的验签过程，具体配置方法请参考4.4.3 固件配置。
	DFU_FW_SAVE_ADDR	在App bootloader中，该宏表示Bank0应用固件存储的起始地址，具体取值请参考2.2.1 Flash布局。 在应用固件ble_app_template_dfu中表示Bank1固件存储的起始地址，即在GRToolbox中使用后台双区升级时拷贝地址栏显示的地址，在应用固件中设置时也需要参考2.2.1 Flash布局。
	BOOTLOADER_BOOT_PORT_ENABLE	当用户不使用GR5xx提供的DFU方案时，可通过使能该宏，使用自定义的方案。 • 0：使用厂商默认方案 • 1：使用自定义方案
	APP_FW_COMMENTS	默认设置为"ble_app_temp"，必须与 Bank0区域的应用程序匹配，才能使程序从App bootloader固件跳转至应用固件。 在App bootloader中启动时，当检查到APP Info数据无效时，会进一步查找SCA区域的数据，根据SCA区域数据的固件COMMENTS和APP_FW_COMMENTS宏进行匹配。如果匹配成功，则将SCA区域的数据更新至APP Info，并根据匹配到的SCA区域数据启动固件。 **说明：** • APP_FW_COMMENTS最大只支持12个字符。

文件名	宏	描述
		• 固件文件名的前12个字符需与APP_FW_COMMENTS对应。
custom_config.h	APP_LOG_ENABLE	当处于开发调试阶段时，打开该宏可查看App bootloader输出的Log信息。 • 0：关闭Log信息 • 1：打开Log信息
	APP_CODE_LOAD_ADDR	表示App bootloader固件在Flash存储的起始地址，取值请根据升级模式参考2.1.1 Flash布局或2.2.1 Flash布局。
	APP_CODE_RUN_ADDR	表示App bootloader固件在Flash运行的起始地址，一般与APP_CODE_LOAD_ADDR取值一样。
无	ENABLE_DFU_SPI_FLASH	是否支持外部Flash升级资源数据，打开该宏可在App bootloader固件为外部Flash升级资源数据，将宏添加于Keil工程“Options for Target > C/C++ > Preprocessor Symbols > Define”配置框中，配置操作请参考4.6.2 外部Flash资源升级。

打开*bootloader_config.h*中的BOOTLOADER_BOOT_PORT_ENABLE自定义配置项，在`SDK_Folder\projects\ble\dfu\app_bootloader\Src\user\bootloader_boot_port.c`文件下添加自定义的DFU方案。GR5xx提供的框架如下：

```
#include "bootloader_config.h"
#if BOOTLOADER_BOOT_PORT_ENABLE
#include "bootloader_boot.h"
void bootloader_dfu_task(void)
{
}
void bootloader_verify_task(void)
{
}
void bootloader_jump_task(void)
{
}
#endif
```

说明:

如果要获得最小的App bootloader固件（Code size最小），需要采用后台双区升级模式，关闭App bootloader的低功耗蓝牙（BOOTLOADER_DFU_BLE_ENABLE）、UART升级功能宏（BOOTLOADER_DFU_UART_ENABLE），关闭Log打印宏（APP_LOG_ENABLE），关闭外部Flash宏（ENABLE_DFU_SPI_FLASH）。

4 使用GRToolbox升级

本章详细介绍如何使用GRToolbox升级GR5xx固件。

4.1 支持平台

表 4-1 支持的开发平台

软件开发平台	硬件平台开发板型号
GR551x SDK V2.0.1及以后版本	GR5515-SK-BASIC
GR5526 SDK V1.0.1及以后版本	GR5526-SK-BASIC
GR5525 SDK V0.8.0及以后版本	GR5525-SK-BASIC
GR533x SDK V0.9.0及以后版本	GR5331-SK-BASIC
GR5405 SDK V1.1.6及以后版本	GR5405-SK-BASIC

4.2 准备工作

- 硬件准备

表 4-2 硬件准备

名称	描述
开发板	对应芯片Starter Kit开发板（以下简称“开发板”）
安卓手机	Android 5.0（KitKat）及以上版本
连接线	USB Type-C（GR551x系列使用Micro USB连接线）
杜邦线	3根
J-Link工具	SEGGER公司推出的JTAG仿真器，如需更多了解，请访问：http://www.segger.com/products/debug-probes/j-link/

- 软件准备

表 4-3 软件准备

名称	描述
Windows	Windows 7/Windows 10操作系统
J-Link Driver	J-Link驱动程序，下载网址：http://www.segger.com/downloads/jlink/
Keil MDK5	IDE工具，支持MDK-ARM 5.20 及以上版本，下载网址：http://www.keil.com/download/product/
GProgrammer（Windows）	Programming工具，下载网址：http://www.goodix.com/zh/software_tool/gprogrammer_ble
GRUart（Windows）	串口调试工具，下载网址：http://www.goodix.com/zh/download?objectId=64&objectType=software
GRToolbox（Android）2.16及以后版本	低功耗蓝牙调试工具，下载网址：http://www.goodix.com/zh/software_tool/grtoolbox

说明:

- 本文介绍的DFU方案主机端必须使用2.16及以后版本的GRToolbox，设备端固件需要满足表 4-1 各SDK版本的要求。
- 当用户在应用固件中使用本文介绍的DFU方案时，启动固件依然使用Second Boot固件（GR551x SDK V1.7.0及以前版本，GR5526 SDK V1.0.0及以前版本），此时使用2.16及以后版本的GRToolbox并采用后台双区升级模式可完成升级。

下述章节将以GR551x为例，按非加密非加签、加密加签和仅加签三种固件类型分别介绍升级步骤。

4.3 非加密非加签固件升级

4.3.1 固件配置

使用GRToolbox工具进行升级时，将使用两个工程：

- `SDK_Folder\projects\ble\dfu\app_bootloader\Keil_5`工程
- `SDK_Folder\projects\ble\ble_peripheral\ble_app_template_dfu\Keil_5`工程

1. App bootloader工程配置

   对于非加密非加签固件升级的App bootloader工程配置项如下表所示：

说明:

- *bootloader_config.h*文件路径：`SDK_Folder\projects\ble\dfu\app_bootloader\Src\config`
- *custom_config.h*文件路径：`SDK_Folder\projects\ble\dfu\app_bootloader\Src\config`

表 4-4 App bootloader工程配置项（非加密非加签固件升级）

文件名	宏	值
bootloader_config.h	BOOTLOADER_DFU_BLE_ENABLE	1：打开蓝牙升级
	BOOTLOADER_DFU_UART_ENABLE	0：关闭串口升级
	BOOTLOADER_WDT_ENABLE	1：打开看门狗
	BOOTLOADER_SIGN_ENABLE	0：关闭验签
	BOOTLOADER_BOOT_PORT_ENABLE	0：使用GR5xx DFU方案
	APP_FW_COMMENTS	"ble_app_temp"
	DFU_FW_SAVE_ADDR	(FLASH_START_ADDR + 0x30000) 说明： 不同芯片“FLASH_START_ADDR”可能不同。
custom_config.h	APP_LOG_ENABLE	1：打开Log输出
	APP_CODE_LOAD_ADDR	(Flash_START_ADDR+0x40000)

文件名	宏	值
	APP_CODE_RUN_ADDR	说明： 不同芯片“FLASH_START_ADDR”可能不同。

2. ble_app_template_dfu工程配置

说明:

- *custom_config.h*文件路径：`SDK_Folder\projects\ble\ble_peripheral\ble_app_template_dfu\Src\config`
- *user_periph_setup.c*文件路径：`SDK_Folder\projects\ble\ble_peripheral\ble_app_template_dfu\Src\platform`

表 4-5 ble_app_template_dfu工程配置项（非加密非加签固件升级）

文件名	宏	值
custom_config.h	APP_CODE_LOAD_ADDR	0x01030000
	APP_CODE_RUN_ADDR	说明： 不同芯片地址可能不同。
user_periph_setup.c	DFU_FW_SAVE_ADDR	(FLASH_START_ADDR + 0x60000) 说明： Flash布局中Bank1区域的起始地址，不可与Bank0以及App bootloader区域重叠

4.3.2 固件烧录

编译App bootloader工程和ble_app_template_dfu工程，生成*app_bootloader.bin*固件和*ble_app_template_dfu.bin*固件，将固件导入至GProgrammer烧录，设置app_bootloader固件为启动固件，固件烧录如下图所示。

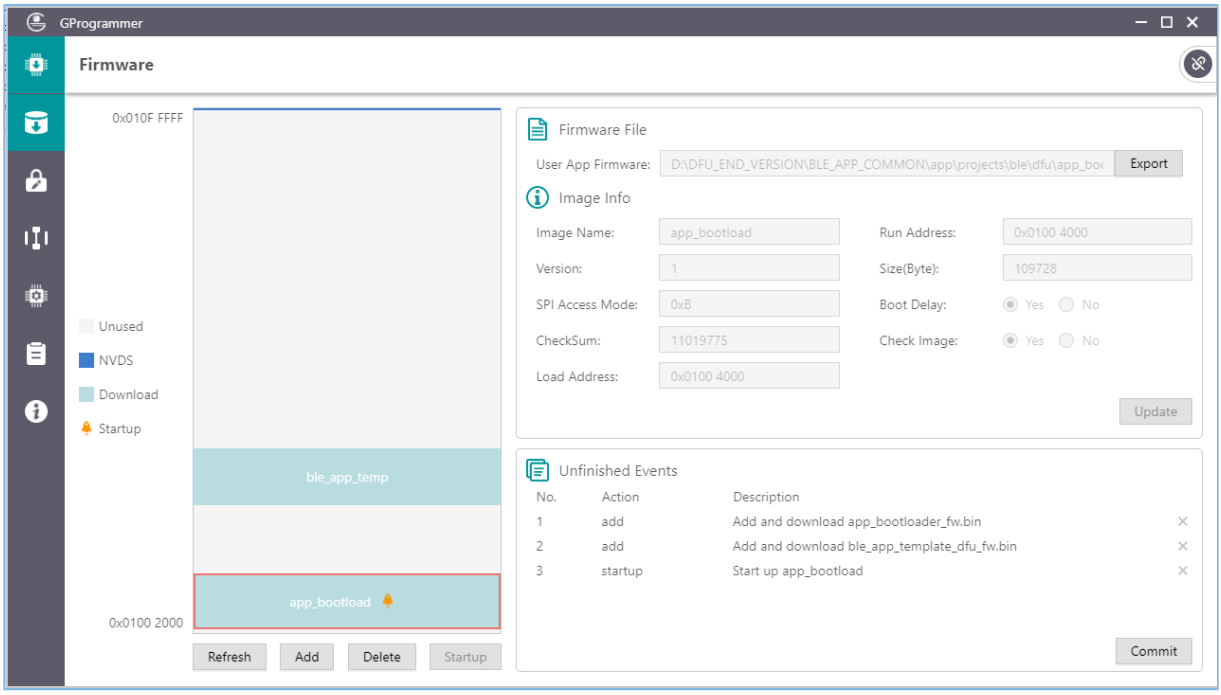

图 4-1 固件烧录

固件烧录完成后，串口输出的Log信息如下图所示。

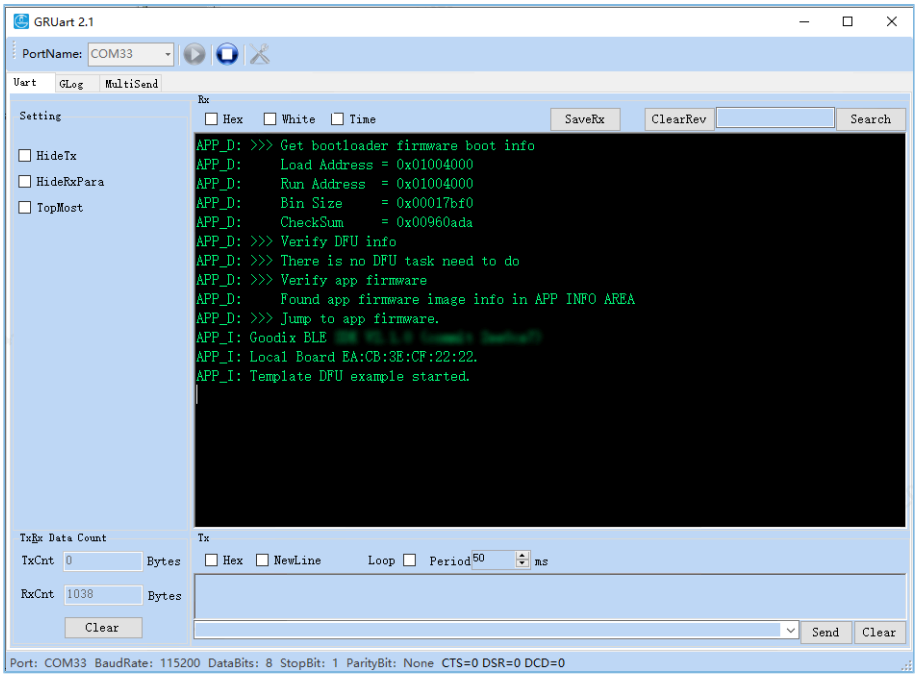

图 4-2 Log信息

4.3.3 创建目标升级固件

进入SDK_Folder\projects\ble\ble_peripheral\ble_app_template_dfu目录，拷贝ble_app_template_dfu，重命名为ble_app_template_dfu_mine，然后在Keil中打开该工程。按照如下步骤创建用于升级的目标固件文件：

1. 命名目标固件。

   将*user_app.c*（路径：SDK_Folder\projects\ble\ble_peripheral\ble_app_template_dfu_mine\Src\user）中的设备广播名修改为“Goodix_Tem_New”，操作如下：

   (1) 修改设备广播名。

```
#define DEVICE_NAME "Goodix_Tem_New"
```

   (2) 修改扫描应答所使用的名称。

```
static const uint8_t s_adv_rsp_data_set[] =
{
 // Complete Name
 0x0f,
 BLE_GAP_AD_TYPE_COMPLETE_NAME,
 'G', 'o', 'o', 'd', 'i', 'x', '_', 'T', 'e', 'm', '_', 'N', 'e', 'w',
 // Manufacturer specific adv data type
 0x05,
 BLE_GAP_AD_TYPE_MANU_SPECIFIC_DATA,
 // Goodix SIG Company Identifier: 0x04F7
 0xF7,
 0x04,
 // Goodix specific adv data
 0x02, 0x03,
};
```

2. 修改Load Address和Run Address，以GR551x为例，修改APP_CODE_RUN_ADDR和APP_CODE_LOAD_ADDR为0x01030000，其他系列芯片请参考2.1.1 Flash布局或2.2.1 Flash布局填写，如下所示：

表 4-6 修改Load Address和Run Address

文件名	宏	值
custom_config.h	APP_CODE_LOAD_ADDR	0x01030000
	APP_CODE_RUN_ADDR	

3. 生成bin文件。

   (1) 重新编译工程，编译后在ble_app_template_dfu_mine\keil_5\Listings文件夹中生成*ble_app_template_dfu.bin*文件为最新固件。

   (2) 由Keil生成的固件在固件尾部不存在固件信息，而在升级时，传输的固件需要包含固件信息。因此，在Keil生成bin文件后，还需将*ble_app_template_dfu.bin*文件导入GProgrammer。

   (3) 导入GProgrammer后，在ble_app_template_dfu_mine\keil_5\Listings路径下生成*ble_app_template_dfu_fw.bin*文件，如下图所示。

图 4-3 目标固件生成界面

(4) 将生成的目标升级固件*ble_app_template_dfu_fw.bin*存入手机根目录下。

4.3.4 进入GRToolbox升级界面

进入GRToolbox升级界面有两种方式：

- 通过点击页面上方的 进入
- 通过DFU应用进入

说明:

GRToolbox界面图片中的参数均以GR551x为例。

下面分别介绍上述两种方式。

- 通过点击页面上方的 进入

1. 打开手机端GRToolbox APP，设备列表中选择“Goodix_Tem_DFU”并连接（如图 4-4）。
2. 点击页面上方的 升级按钮（如图 4-5），此时会跳转进入DFU升级界面（如图 4-6）。

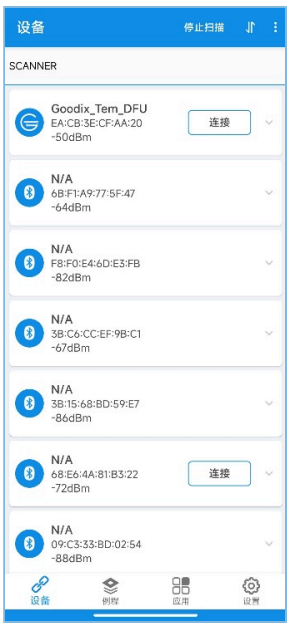

图 4-4 选择“Goodix_Tem_DFU”

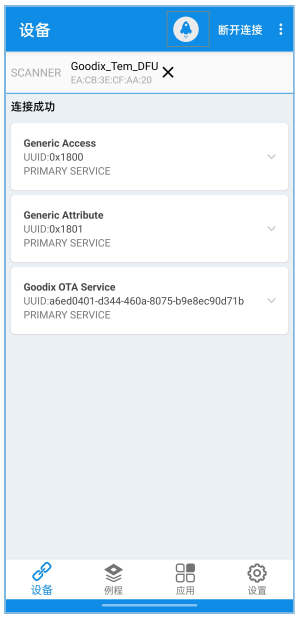

图 4-5 点击升级按钮

图 4-6 DFU升级界面

- 通过DFU应用进入

1. 打开手机端GRToolbox，在界面下方选项卡中选择应用，然后点击“DFU”应用，如图 4-7所示。

2. 进入DFU升级界面（如图 4-8），此时显示未连接设备，点击界面下方“连接”按钮，进入连接设备界面，如图 4-9所示。

3. 在设备列表中选择“Goodix_Tem_DFU”，进入升级模式，如图 4-10所示。

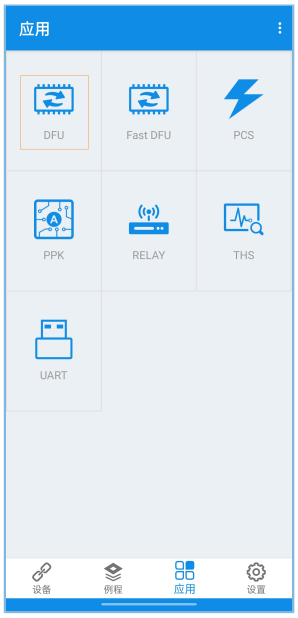

图 4-7 点击“DFU”应用

图 4-8 DFU升级界面

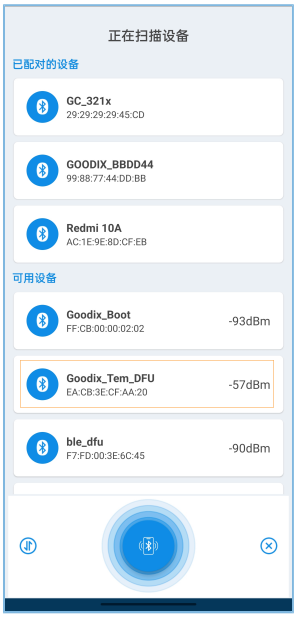

图 4-9 扫描设备

图 4-10 进入升级模式

4.3.5 固件升级

本节主要介绍后台双区升级模式及非后台单区升级模式的详细操作。

4.3.5.1 后台双区升级模式

1. 进入GRToolbox界面后，点击“选择文件”按钮，选择存入手机根目录的*ble_app_template_dfu_fw.bin*固件，勾选“双区升级模式”。

2. 勾选“双区升级模式”后，下方出现“拷贝地址”（如图 4-11所示）。

   默认情况下，该地址置灰，不可修改。若发现该地址有误，可点击界面右上角 后，勾选“自定义拷贝地址”，即可修改“拷贝地址”，如图 4-13所示。

   图 4-12中“快速模式”和“写控制点”选项介绍如下：

   - “快速模式”：选择当前升级方式是否使能快速模式，如果取消勾选“快速模式”，当前升级过程中将采用普通DFU模式，速度较慢。
   - “写控制点”：下发开始设备DFU任务的命令。实际应用中为减小功耗，DFU任务并非始终运行。当手机端连接的设备此时的DFU任务尚未运行，并且需要在开始升级前，通过GRToolbox获取固件信息，可通过点击“写控制点”，下发启动DFU任务命令的方式唤醒DFU任务运行。关于DFU任务停止以及启动的机制请参考5 DFU移植方式详解。

图 4-11 勾选“双区升级模式”

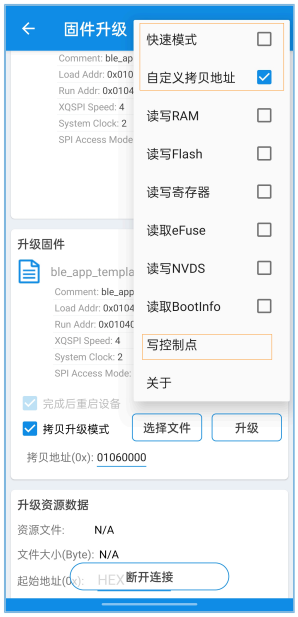

图 4-12 自定义拷贝地址

图 4-13 修改拷贝地址

3. 点击“升级”按钮进行升级，如图 4-14所示，升级过程会有进度条标明升级进度。升级完成后，在界面底部会提示升级完成。

   在创建目标升级固件时，已将广播名更改为“Goodix_Tem_New”，因此，为了验证当前升级是否完成，可搜索有无广播名为“Goodix_Tem_New”的固件。如图 4-15所示，可搜索到该设备，说明升级成功。

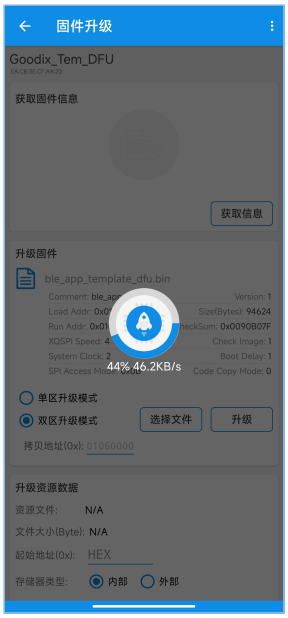

图 4-14 升级进度

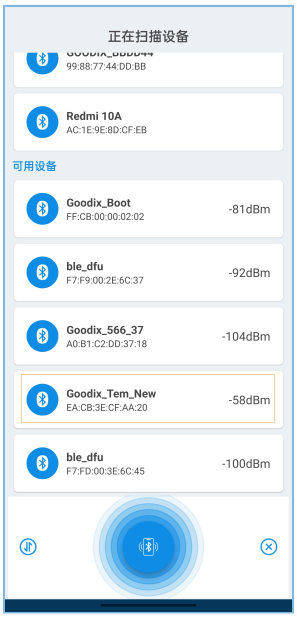

图 4-15 搜索“Goodix_Tem_New”

4.3.5.2 非后台单区升级模式

非后台单区升级模式有两种场景：

- 当前固件运行在ble_app_template_dfu。

- 应用固件损坏，当前固件运行在App bootloader。

下面将分别介绍两种场景的操作步骤。

- 当前固件运行在ble_app_template_dfu。

  非后台单区升级模式的步骤与后台双区升级模式的基本一致。

  1. 进入手机端GRToolbox时，同样搜索“Goodix_Tem_DFU”，连接后进入升级界面，然后勾选“单区升级模式”，如图 4-16所示。
  2. 如需选择“快速模式”，则点击界面右上角，勾选“快速模式”（如图 4-17所示）。
  3. 设置完成后，点击“升级”。

     此模式下进入升级进度条之前所耗时间要比后台双区升级模式多，因为当前需要从ble_app_template_dfu固件跳转至app_bootloader固件并进行重连，重连后便进入升级进度条（如图 4-18所示）。升级完成后，会在界面底部提示。

图 4-16 勾选“单区升级模式”

图 4-17 勾选“快速模式”

图 4-18 升级进度

- 当前固件运行在App bootloader

  当应用固件损坏或当前没有应用固件时，则需要在app_bootloader中进行固件升级。其操作与在ble_app_template_dfu中的基本一致，区别在于广播名称不同，需要连接的广播名称为Bootloader_OTA，连接后的操作同在ble_app_template_dfu中升级是一致的，此处不再赘述，可参考当前固件运行在ble_app_template_dfu的升级步骤。

4.4 加密加签固件升级

将系统设置成加密模式，需要设置eFuse，eFuse中存放着产品配置信息、安全模式控制信息以及用于加密加签的各种密钥信息，因此如果尚未下载过设置信息至eFuse中，需要先进行eFuse设置，然后将设置信息下载至eFuse。用户可点击GProgrammer工具左侧工具栏中的“Encrypt & Sign”进入芯片加密操作界面，如下图所示。

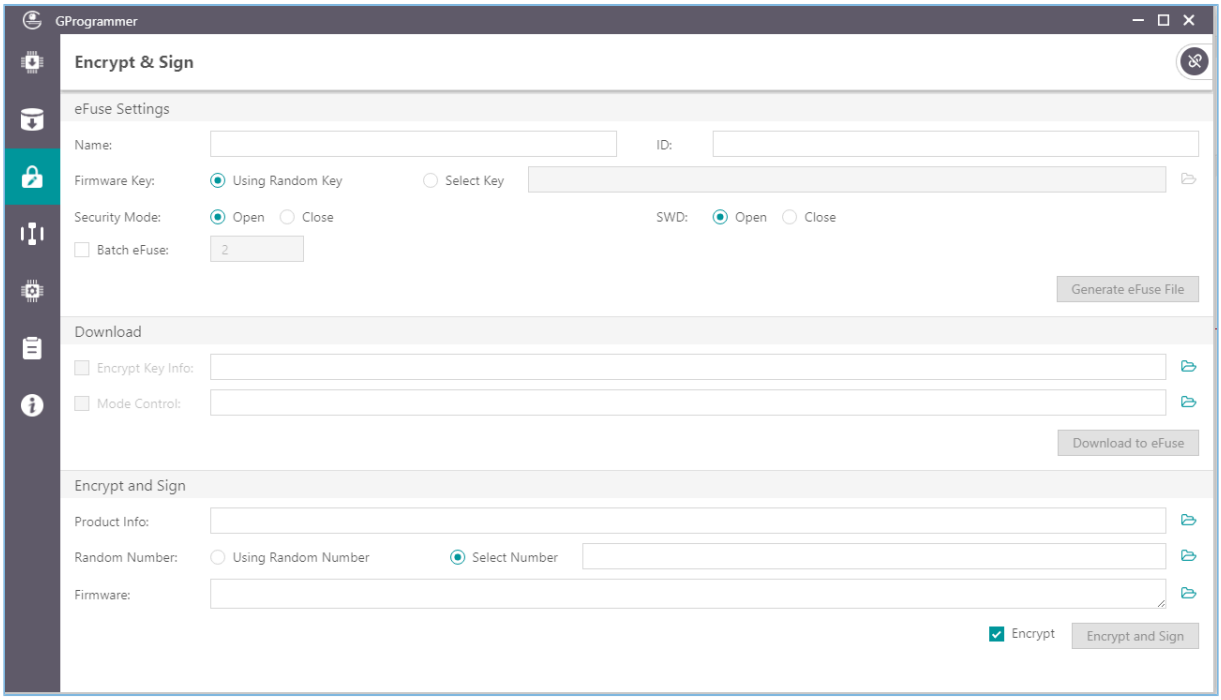

图 4-19 Encrypt & Sign界面

说明:

GR533x系列芯片不支持加密加签设置。

4.4.1 eFuse设置

用户可通过GProgrammer中的“Encrypt & Sign”指定产品的Name、ID和Firmware Key，并配置Security Mode和SWD接口，生成存储在eFuse中的相关文件。

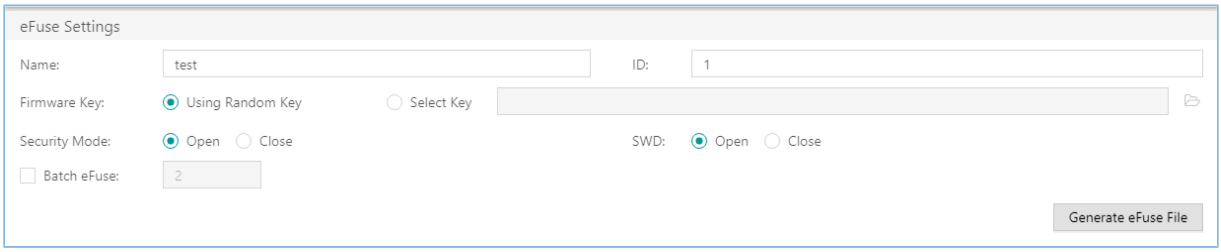

图 4-20 eFuse设置栏界面

eFuse是芯片内部的一个具有随机访问接口的一次性可编程存储器。图 4-20中参数说明如下：

- Firmware Key：可使用软件自动生成的随机Key，也可使用用户自行添加的Key文件。
- Security Mode：选择“Open”将启用芯片安全模式，该模式开启后不能被关闭。
- SWD：选择“Close”将关闭SWD功能，开发者依然可以通过DFU对固件进行升级。
- Batch eFuse：可指定生成相应数量的数据密钥文件，以支持一机一密的使用场景。若不勾选则只会生成一个数据密钥文件。

生成的文件包括：

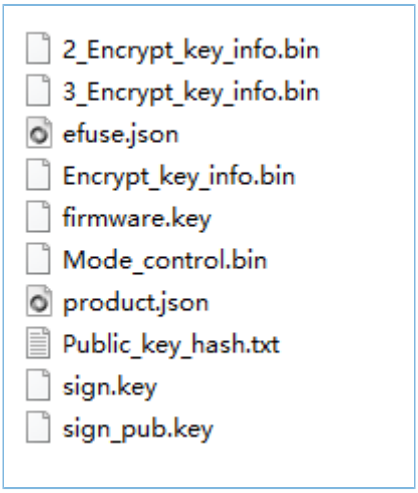

图 4-21 生成的文件

- efuse.json：临时文件。
- Encrypt_key_info.bin、2_Encrypt_key_info.bin、3_Encrypt_key_info.bin：eFuse下载文件，包含产品及加密加签信息。此文件需下载至eFuse。
- firmware.key：用于加密固件的私钥。
- Mode_control.bin：eFuse下载文件，包含Security Mode和SWD信息。此文件需下载至eFuse。
- product.json：产品信息文件。当固件加密加签时，需导入此文件。
- sign.key：用于生成签名的私钥。
- sign_pub.key：用于验证签名的公钥。
- Public_key_hash.txt：用于验证签名的公钥Hash。

为了方便用户下载文件至eFuse或加密加签固件，生成的*Encrypt_key_info.bin*和*Mode_control.bin*文件的路径将被默认添加至“Download”区域，产品信息文件*product.json*的路径被默认添加至“Encrypt and Sign”的“Product Info”中。

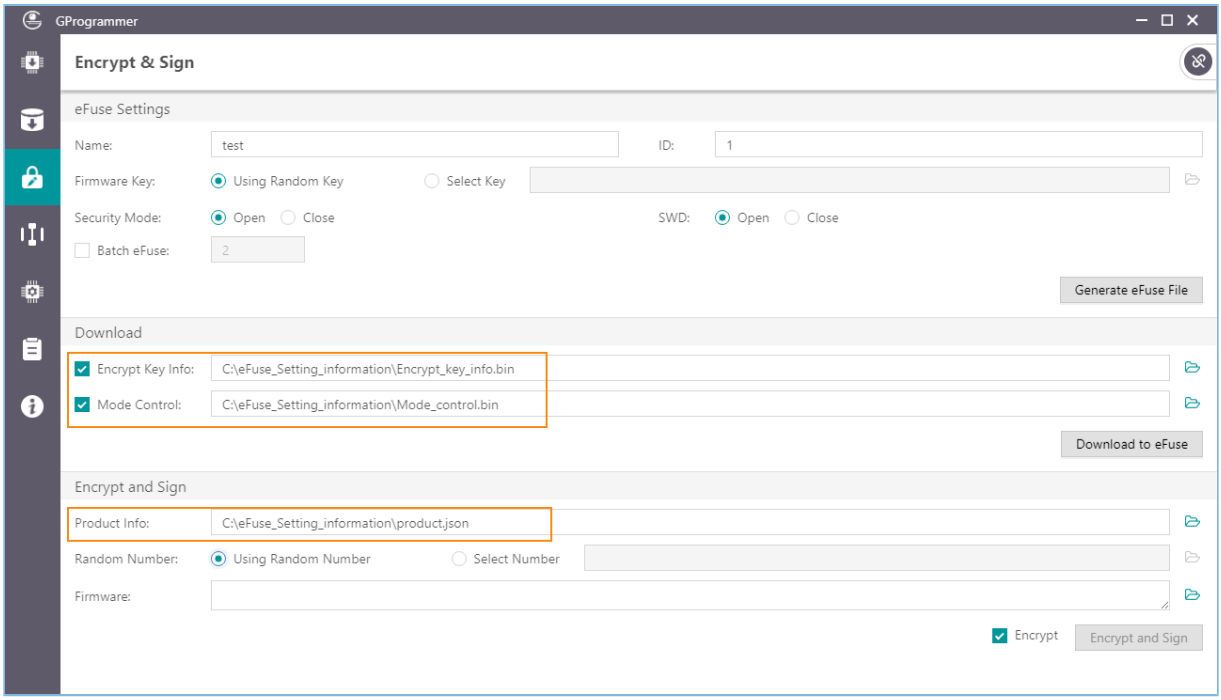

图 4-22 GProgrammer加密加签界面

4.4.2 eFuse下载

若用户已执行eFuse设置操作，可直接点击“Download to eFuse”将文件下载至eFuse中。否则，需先手动选择添加*Encrypt_key_info.bin*和*Mode_control.bin*文件，然后才能下载。

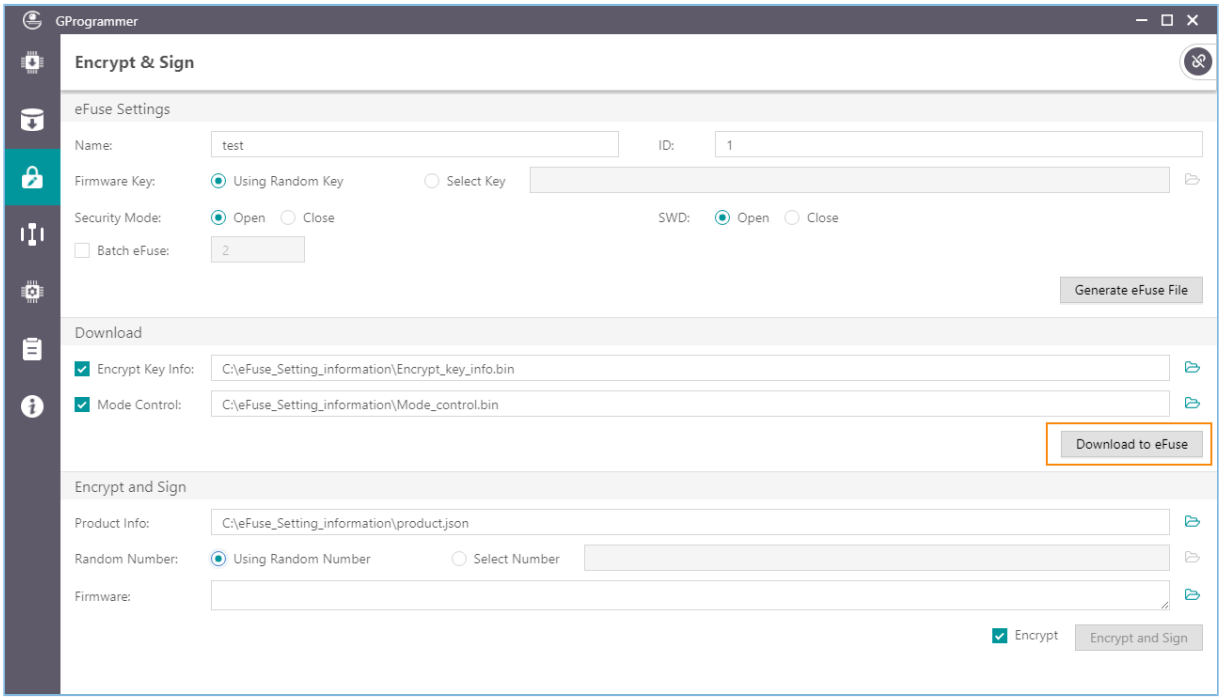

图 4-23 下载文件至eFuse

4.4.3 固件配置

对于加密加签固件的升级，需要配置App bootloader相关参数，才能在跳转应用固件之前对应用固件进行正确校验。

说明:

*bootloader_config.h*文件路径：SDK_Folder\projects\ble\dfu\app_bootloader\Src\config

表 4-7 bootloader_config.h配置（加密加签）

文件名	宏	值
bootloader_config.h	BOOTLOADER_SIGN_ENABLE	1：打开验签
	BOOTLOADER_PUBLIC_KEY_HASH	*Public_key_hash.txt*文件内的值

BOOTLOADER_PUBLIC_KEY_HASH是一组值，该值存放在图 4-21所示的*Public_key_hash.txt*文件中，将该文件中的数值复制到*bootloader_config.h*文件中的BOOTLOADER_PUBLIC_KEY_HASH宏定义中。

4.4.4 生成加密加签固件

在芯片加密模式下，固件需要加密加签后才可下载至Flash中运行。GProgrammer允许用户使用同一套产品信息和同一个Random Number加密加签多个固件文件。在 “Encrypt & Sign > Encrypt and Sign > Firmware”中添加多个固件时，各固件的路径之间用分号隔开，如图 4-24所示：

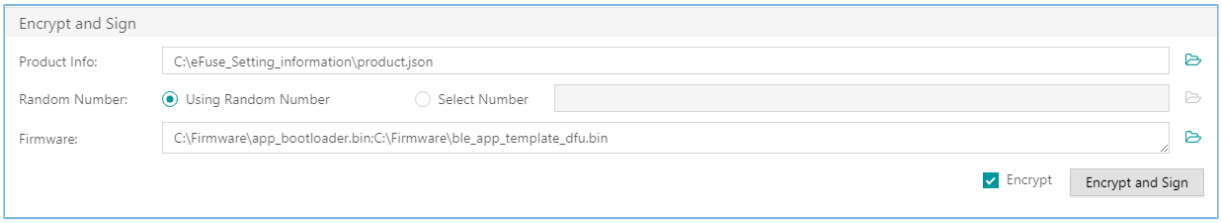

图 4-24 添加多个固件

在加密芯片中，需要对app_bootloader和ble_app_template_dfu固件同时进行加密，程序才能正确运行，加密加签后生成的文件如下图所示：

图 4-25 加密加签生成的文件

4.4.5 固件升级

加密加签固件在GRToolbox的升级操作与非加密非加签固件的操作基本一致，区别仅在于存放入手机的固件是加密加签固件，其他操作可参考4.3.5 固件升级。

4.5 仅加签非加密固件升级

对固件加签是为了在固件传输过程中，保证固件不被第三方篡改。因此，在固件写入完毕后，跳转到应用固件运行前，需要对应用固件进行验签。

4.5.1 固件配置

仅加签非加密固件需要在app_bootloader固件进行相关配置，相关配置项如下表所示：

说明:

*bootloader_config.h*文件路径：`SDK_Folder\projects\ble\dfu\app_bootloader\Src\config`

表 4-8 bootloader_config.h配置（仅加签）

文件名	宏	值
bootloader_config.h	BOOTLOADER_SIGN_ENABLE	1：打开验签
	BOOTLOADER_PUBLIC_KEY_HASH	*Public_key_hash.txt*文件内的值

4.5.2 生成仅加签非加密固件

使用GProgrammer工具对固件进行加签，如图 4-26所示，由2.4 固件格式可知，加密固件和加签固件在固件尾部增加的长度是相同的，在生成时的步骤也基本一致。区别在于生成仅加签非加密固件时，在GProgrammer的“Encrypt & Sign > Encrypt and Sign”区域中去勾选“Encrypt”，生成的加签文件如图 4-27所示。

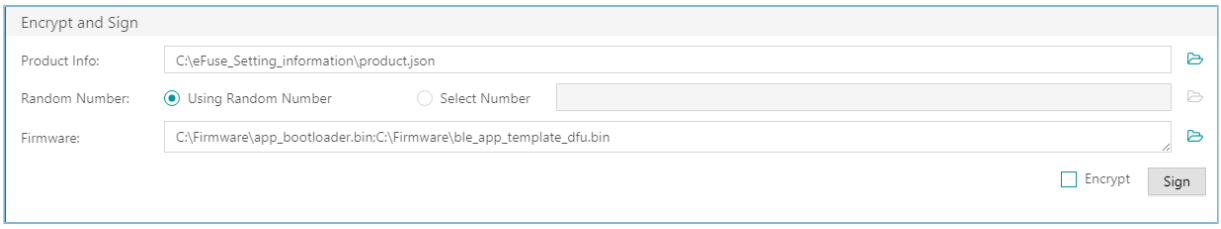

图 4-26 GProgrammer固件加签界面

图 4-27 仅加签固件

4.5.3 固件升级

升级仅加签非加密固件时，其操作步骤与非加密非加签固件是相同的，可参考4.3.5 固件升级。

4.6 资源升级

资源升级指对图片、字体、音频等不作为代码的数据进行升级。GR5xx支持内部Flash及外部Flash资源升级两种方式。

- 内部资源升级：只需加入DFU相关组件，无需其他配置，即可在app_bootloader固件或者应用固件中进行升级。
- 外部资源升级：在加入DFU相关组件的情况下，需要在Keil工程里进行宏配置。

4.6.1 内部Flash资源升级

采用内部Flash进行资源升级的步骤如下：

1. 设置“起始地址”和“存储器类型”，如图 4-28所示。

图 4-28 内部Flash资源升级示意图

- 起始地址：数据存放的开始地址，需开发人员提前规划，避免下载资源数据时覆盖其他有用的数据。对于GR551x，考虑app_bootloader及ble_app_template_dfu在Flash区域的位置，将该起始地址的值设置为0x01060000，其他芯片的地址设置请参考2.1.1 Flash布局或2.2.1 Flash布局。
- 存储器类型：“内部”指芯片内部的Flash存储器，“外部”指芯片外接的Flash。

2. 点击“升级”，升级进度如下图所示。

图 4-29 内部资源升级进度

3. 资源下载完成后，当前运行的固件会对资源数据进行校验。若校验成功，GRToolbox APP界面底部会提示升级成功。

4.6.2 外部Flash资源升级

app_bootloader和ble_app_template_dfu固件均可进行外部Flash资源升级：

- 常见应用场景是在应用固件ble_app_template_dfu中进行外部Flash资源升级，只需在ble_app_template_dfu工程里配置“ENABLE_DFU_SPI_FLASH”，如图 4-30所示，使能当前工程采用外部Flash资源升级。

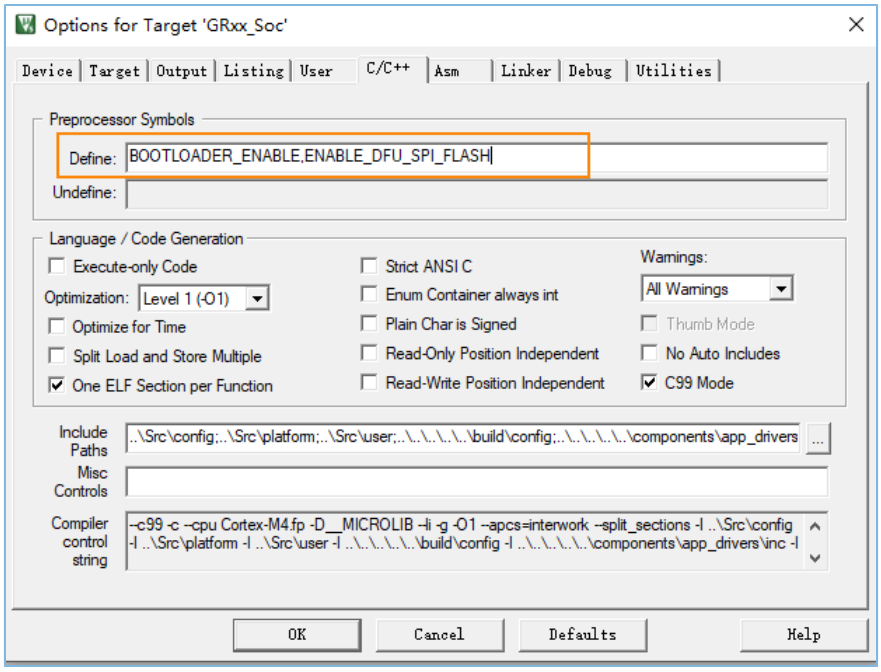

图 4-30 使能外部Flash资源升级

- 在app_bootloader固件中进行外部Flash资源升级，需要如图 4-30中添加“ENABLE_DFU_SPI_FLASH”，还需要在app_bootloader工程里添加使能低功耗蓝牙的宏，具体配置如下表所示。

表 4-9 bootloader_config.h配置

文件名	宏	值
bootloader_config.h	BOOTLOADER_DFU_BLE_ENABLE	1：打开蓝牙升级

根据实际情况完成上述配置后，其他升级步骤无论是在app_bootloader中还是ble_app_template_dfu固件中进行外部Flash资源升级都是相同的，具体步骤如下所示：

1. 配置存储器I/O，如图 4-31所示。

图 4-31 配置存储器I/O

配置I/O时，根据外部Flash芯片与GR5xx的通信方式，可选择SPI、QSPI0及QSPI1等方式。示例中均采用各系列芯片SK板的板载外部Flash进行升级，各系列芯片SK板的板载 Flash 接口如下表所示：

表 4-10 SK板的板载Flash接口

开发板型号	接口类型	CS	CLK	IO0	IO1	IO2	IO3
GR5515-SK-BASIC	QSPI1	GPIO_15	GPIO_9	GPIO_8	GPIO_14	GPIO_13	GPIO_12
GR5525-SK-BASIC	QSPI0	GPIO_15	GPIO_18	GPIO_19	GPIO_14	GPIO_13	GPIO_12
GR5526-SK_BASIC	QSPI0	GPIO_26	GPIO_21	GPIO_22	GPIO_23	GPIO_24	GPIO_25

说明:

- GR5331-SK-BASIC 没有板载外部 Flash，也没有 QSPI 接口，如果要使用外部Flash进行资源升级，只能使用SPI接口驱动外部 Flash，在实际操作时，需要参考《GR533x Datasheet》，使用杜邦线连接GR533x的SPI引脚和外部Flash芯片引脚。
- GR5405-SK-BASIC有板载外部 Flash，但无QSPI，使用SPI接口驱动外部Flash，用户可根据实际需求实现Flash读/写/擦除控制。

以GR5515-SK-BASIC开发板为例，在手机APP界面选择“配置存储器I/O”后，按照图 4-32所示进行配置。

图 4-32 外部Flash引脚配置界面

2. 完成配置后，点击“升级”按钮，升级进度如下图所示。

图 4-33 升级进度

3. 升级完成后，当前运行的固件会对外部Flash的数据进行校验。若校验通过，则会在GRToolbox APP界面底部提示升级完成。

5 DFU移植方式详解

为方便用户将GR5xx提供的DFU方案应用到自定义的工程中，本章以ble_app_template_freertos示例工程为例（工程位于SDK_Folder\projects\ble\ble_peripheral\ble_app_template_freertos\Keil_5），介绍为该工程移植DFU功能的操作。移植过程主要包括以下步骤。

1. DFU初始化
2. 添加DFU调度器
3. DFU服务初始化（使用低功耗蓝牙通讯方式）及数据接收处理
4. 添加DFU组件
5. 添加OTA Profile

具体操作步骤如下：

1. 初始化DFU，代码位于*user_periph_setup.c*文件的app_periph_init函数中，如下所示：

```
#include “dfu_port.h”
void app_periph_init(void)
{
    dfu_uart_init();
    dfu_uart_ctrl_pin_init();
    dfu_port_init(uart_send_data, DFU_FW_SAVE_ADDR, &dfu_pro_call);
}
```

dfu_uart_init()是串口初始化函数，当使用串口通讯方式时，需要初始化串口，以GR551x为例，使用UART1进行数据传输，初始化代码如下所示：

```
static app_uart_params_t dfu_uart_param;
#define DFU_UART_RX_BUFF_SIZE  0x400
#define DFU_UART_TX_BUFF_SIZE  0x400
static uint8_t s_dfu_uart_rx_buffer[DFU_UART_RX_BUFF_SIZE];
static uint8_t s_dfu_uart_tx_buffer[DFU_UART_TX_BUFF_SIZE];

static void dfu_uart_init(void)
{
    app_uart_tx_buf_t uart_buffer;

    uart_buffer.tx_buf       = s_dfu_uart_tx_buffer;
    uart_buffer.tx_buf_size  = DFU_UART_TX_BUFF_SIZE;

    dfu_uart_param.id                   = APP_UART1_ID;
    dfu_uart_param.init.baud_rate       = APP_UART_BAUDRATE;
    dfu_uart_param.init.data_bits       = UART_DATABITS_8;
    dfu_uart_param.init.stop_bits       = UART_STOPBITS_1;
    dfu_uart_param.init.parity          = UART_PARITY_NONE;
    dfu_uart_param.init.hw_flow_ctrl    = UART_HWCONTROL_NONE;
    dfu_uart_param.init.rx_timeout_mode = UART_RECEIVER_TIMEOUT_ENABLE;
```

```
    dfu_uart_param.pin_cfg.rx.type      = APP_UART1_RX_IO_TYPE;
    dfu_uart_param.pin_cfg.rx.pin       = APP_UART1_RX_PIN;
    dfu_uart_param.pin_cfg.rx.mux       = APP_UART1_RX_PINMUX;
    dfu_uart_param.pin_cfg.rx.pull      = APP_UART_RX_PULL;
    dfu_uart_param.pin_cfg.tx.type      = APP_UART1_TX_IO_TYPE;
    dfu_uart_param.pin_cfg.tx.pin       = APP_UART1_TX_PIN;
    dfu_uart_param.pin_cfg.tx.mux       = APP_UART1_TX_PINMUX;
    dfu_uart_param.pin_cfg.tx.pull      = APP_UART_TX_PULL;

    app_uart_init(&dfu_uart_param, dfu_uart_evt_handler, &uart_buffer);
    app_uart_receive_async(APP_UART1_ID, s_dfu_uart_rx_buffer,
                           sizeof(s_dfu_uart_rx_buffer));
}
```

在使用UART升级时，除了初始化串口的TX和RX引脚以外，还需初始化一个控制引脚，该控制引脚的作用：唤醒正在睡眠的设备；启动已停止的DFU任务。因此，如果用户的设备采用了睡眠机制，必须选择AON类型的引脚作为控制引脚。以GR551x为例，选择AON_GPIO_PIN_1作为控制引脚，初始化代码如下所示：

```
#define DFU_UART_CTRL_PIN    AON_GPIO_PIN_1

void dfu_uart_ctrl_pin_init(void)
{
    app_io_init_t io_init = APP_IO_DEFAULT_CONFIG;

    io_init.pull = APP_IO_PULLUP;
    io_init.mode = APP_IO_MODE_IT_FALLING;
    io_init.pin  = DFU_UART_CTRL_PIN;
    io_init.mux  = APP_IO_MUX;
    app_io_init(APP_IO_TYPE_AON, &io_init);

    app_io_event_register_cb(APP_IO_TYPE_AON, &io_init, app_io_event_handler,
                             "DFU uart ctrl pin interrupt");
}
```

在应用固件中，DFU_FW_SAVE_ADDR表示当进行后台双区升级时，固件拷贝的起始地址，其值设置为：

```
#define DFU_FW_SAVE_ADDR (FLASH_START_ADDR + 0x60000)
```

说明:

该值不是固定的，设置时请注意不能和App bootloader固件以及Bank0固件的地址冲突。

uart_send_data是串口发送数据函数，使用串口通讯方式升级时，需要注册该接口，该函数定义如下所示：

```
static void uart_send_data(uint8_t *data, uint16_t size)
{
    app_uart_transmit_async(APP_UART1_ID, data, size);
```

```
}
```

dfu_pro_call是升级过程中升级进度打印的回调函数，详细代码如下所示：

```
static void dfu_program_start_callback(void);
static void dfu_programing_callback(uint8_t pro);
static void dfu_program_end_callback(uint8_t status);

static dfu_pro_callback_t dfu_pro_call =
{
    .dfu_program_start_callback = dfu_program_start_callback,
    .dfu_programing_callback = dfu_programing_callback,
    .dfu_program_end_callback = dfu_program_end_callback,
};

static void dfu_program_start_callback(void)
{
    APP_LOG_DEBUG("Dfu start program");
}

static void dfu_programing_callback(uint8_t pro)
{
    APP_LOG_DEBUG("Dfu programing---%d%%", pro);
}

static void dfu_program_end_callback(uint8_t status)
{
    APP_LOG_DEBUG("Dfu program end");
    if (0x01 == status)
    {
        APP_LOG_DEBUG("status: successful");
    }
    else
    {
        APP_LOG_DEBUG("status: error");
    }
}
```

2. 创建DFU信号量及DFU调度任务。

```
#include “dfu_port.h”

#define DFU_TASK_STACK_SIZE             ( 1024 * 2 )
TaskHandle_t         dfu_task_handle;
SemaphoreHandle_t    xDfuSemaphore;
uint8_t              start_dfu_task_flag = 0;


int main(void)
{
```

```
    ......
    xDfuSemaphore = xSemaphoreCreateBinary();
    xTaskCreate(vStartTasks, "create_task", 512, NULL, 0, NULL);
    vTaskStartScheduler();
    for(;;);
}

static void vStartTask(void *arg)
{
    ......
    xTaskCreate(dfu_schedule_task, "dfu_schedule_task", DFU_TASK_STACK_SIZE, NULL,
                configMAX_PRIORITIES - 2,  &dfu_task_handle);
    ......
    vTaskDelete(NULL);
}



static void dfu_schedule_task(void *p_arg)
{
    while (1)
    {
        if (!start_dfu_task_flag)
        {
            xSemaphoreTake(xDfuSemaphore, portMAX_DELAY);
        }
        dfu_schedule();
    }
}
```

实际应用中，在主机端与设备端不进行数据交互时，需要暂停DFU任务的运行，以降低功耗。GR5xx提供的方案是为DFU任务添加超时机制及信号量。如上述代码所示，DFU任务创建后，即在等待信号量，获取到信号量后，开始运行DFU任务。DFU信号量在不同通讯方式下释放方式不同，低功耗蓝牙通过Control Point特性下发DFU Enter命令释放信号量；UART通过控制引脚触发外部中断的形式释放信号量。

当设备端超过一定时间没有接收到主机下发的数据时，则判定为当前DFU任务已结束，在超时回调函数里将DFU任务开始标志位复位，停止DFU任务的运行，继续等待DFU信号量。超时机制通过app_timer软件定时器组件实现。在使用app_timer时需要包含#include “app_timer.h”，定时器初始化函数定义如下，该函数需要在app_periph_init()函数内进行调用。

```
static app_timer_id_t  s_cmd_wait_timer_id;

static void dfu_timer_init(void)
{
    app_timer_create(&s_cmd_wait_timer_id, ATIMER_REPEAT, cmd_wait_timeout_handler);
}
```

超时回调函数的定义如下所示：

```
static uint16_t        s_dfu_last_count;
static uint16_t        s_dfu_curr_count;
extern uint8_t         start_dfu_task_flag;

static void cmd_wait_timeout_handler(void* p_arg)
{
    if (s_dfu_last_count < s_dfu_curr_count)
    {
        s_dfu_last_count = s_dfu_curr_count;
    }
    else
    {
        s_dfu_curr_count = 0;
        s_dfu_last_count = 0;
        dfu_cmd_parse_state_reset();
        fast_dfu_state_machine_reset();
        start_dfu_task_flag = 0;
        dfu_timer_stop();
    }
}
```

由上述代码可知，当主机端超过一定时间没有发送数据至设备端后，会触发超时函数，超时函数会调用dfu_cmd_parse_state_reset函数，该函数由SDK内部实现，可直接调用，具体作用是复位DFU命令解析的状态机，以免影响下次升级时数据解析。fast_dfu_state_machine_reset函数也需要在超时函数里进行调用，作用是复位快速模式的状态机，以免影响下次升级时主机端与设备端的数据传输。GR551x SDK V2.0.1和GR5525 SDK V0.8.0的*dfu_port.c*里目前没有实现fast_dfu_state_machine_reset接口，需要自行实现，实现代码如下所示：

```
void fast_dfu_state_machine_reset(void)
{
    s_fast_dfu_mode = 0;
    s_program_end_flag = 0;
    s_fast_dfu_state = FAST_DFU_INIT_STATE;
}
```

然后将DFU任务开始标志位复位，最后停止DFU软件定时器。软件定时器停止函数定义如下：

```
static void dfu_timer_stop(void)
{
    app_timer_stop(s_cmd_wait_timer_id);
}
```

3. 服务初始化及数据接收处理。当使用低功耗蓝牙通讯方式进行DFU时，在*user_app.c*文件的service_init函数中初始化DFU相关服务，并添加头文件#include “dfu_port.h”，如下所示：

```
static void services_init(void)
{
    dfu_service_init(dfu_enter);
```

```
}
```

dfu_enter函数定义默认为空，用户可在函数内部添加DFU任务触发代码，另一方面，为了实现DFU的超时机制，在dfu_enter函数里调用定时器开始函数。当设备端收到DFU Enter指令后，释放DFU信号量、将DFU任务开始标志位置位，开启DFU超时机制的软件定时器。添加的代码如下所示：

```
extern SemaphoreHandle_t    xDfuSemaphore;
extern uint8_t          start_dfu_task_flag;

static void dfu_enter(void)
{
    if (!start_dfu_task_flag)
    {
        start_dfu_task_flag = 1;
        xSemaphoreGive(xDfuSemaphore);
        dfu_timer_start();
    }
}
```

dfu_timer_start的代码实现如下所示：

```
#define DFU_CMD_WAIT_TIMEOUT            4000

void dfu_timer_start(void)
{
    app_timer_start(s_cmd_wait_timer_id, DFU_CMD_WAIT_TIMEOUT, NULL);
}
```

如果是以UART方式进行通讯，则没有dfu_enter命令，也没有对应的dfu_enter回调函数。UART通讯方式的DFU任务唤醒方法使用在步骤1初始化的控制引脚的中断服务函数里完成，中断服务函数定义如下：

```
extern SemaphoreHandle_t    xDfuSemaphore;
extern uint8_t              start_dfu_task_flag;

void app_io_event_handler(app_io_evt_t *p_evt)
{
    app_io_pin_state_t pin_level = APP_IO_PIN_RESET;

    if (p_evt->pin == DFU_UART_CTRL_PIN)
    {
        pin_level = app_io_read_pin(APP_IO_TYPE_AON, DFU_UART_CTRL_PIN);
        if (pin_level == APP_IO_PIN_RESET)
        {
            do
            {
                pin_level = app_io_read_pin(APP_IO_TYPE_AON, DFU_UART_CTRL_PIN);
            } while(pin_level == APP_IO_PIN_SET);

            if (!start_dfu_task_flag)
            {
```

```
                start_dfu_task_flag = 1;
                xSemaphoreGive(xDfuSemaphore);
                dfu_timer_start();
            }
        }
    }
}
```

开启DFU软件定时器后，需要在数据接收位置调用count累加的函数，count累加函数定义如下所示：

```
void dfu_rev_cmd_count(void)
{
    s_dfu_curr_count++;
}
```

当以低功耗蓝牙通讯方式进行DFU时，在低功耗蓝牙接收数据处调用dfu_rev_cmd_count函数，具体位置为*dfu_port.c*文件的otas_evt_process(otas_evt_t *p_evt)函数内，dfu的数据接收函数也在此添加，具体代码如下所示：

```
static void otas_evt_process(otas_evt_t *p_evt)
{
    switch(p_evt->evt_type)
    {
        ......
        case OTAS_EVT_RX_RECEIVE_DATA:
        dfu_rev_cmd_count();
        if (!s_fast_dfu_mode || s_program_end_flag)
        {
            dfu_ble_receive_data_process(p_evt->p_data, p_evt->length);
        }
        else
        {
            s_fast_dfu_state = FAST_DFU_PROGRAM_FLASH_STATE;
            fast_dfu_write_data_to_buffer(p_evt->p_data, p_evt->length);
        }
        break;
        ......
    }
}
```

当以UART方式进行DFU时，在UART接收数据处调用dfu_rev_cmd_count函数，即在dfu_uart_evt_handler函数内进行调用，dfu的数据接收函数也在此进行添加，具体代码如下所示：

```
void dfu_uart_evt_handler(app_uart_evt_t * p_evt)
{
    switch(p_evt->evt_type)
    {
    ......
        case APP_UART_EVT_RX_DATA:
        dfu_rev_cmd_count();
```

```
            dfu_uart_receive_data_process(s_dfu_uart_rx_buffer, p_evt->data.size);
            app_uart_receive_async(APP_UART1_ID, s_dfu_uart_rx_buffer,
                                   DFU_UART_RX_BUFF_SIZE);
            break;
        ......
        }
}
```

说明:

在手机端当前连接设备的DFU任务尚处于停止状态时，点击“升级”按钮前，若需通过 GRToolbox 获取固件信息，可通过写控制点的方式手动下发唤醒DFU任务的命令，具体操作请参考4.3.5 固件升级。

4. 在gr_libraries和gr_profiles工程文件夹下分别添加*dfu_port.c*和*otas.c*文件，如下图所示。

说明:

*dfu_port.c*和*otas.c*文件分别位于SDK_Folder\components\libraries\dfu_port和SDK_Folder\components\profiles\otas。

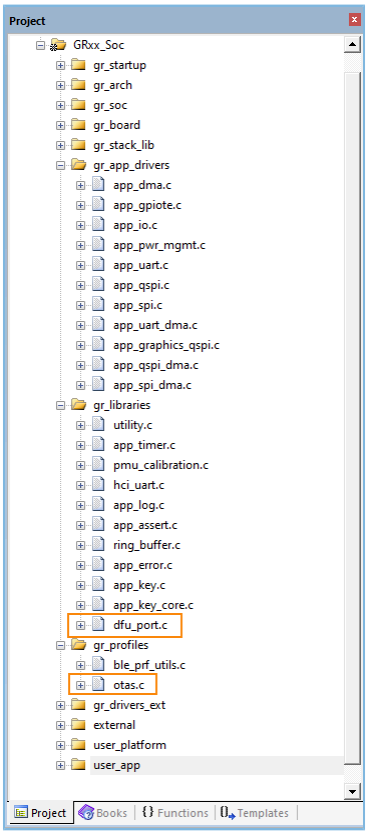

图 5-1 Keil工程列表

5. 在*custom.config.h*中设置APP_CODE_LOAD_ADDR和APP_CODE_RUN_ADDR为FLASH_START_ADDR + 0x30000，使其和boot固件不会产生覆盖的情况。因此，以GR551x为例，APP_CODE_RUN_ADDR和APP_CODE_LOAD_ADDR设置为0x01030000。

```
// <o> Code load address
```

```
// <i> Default:  0x01002000
#define APP_CODE_LOAD_ADDR      0x01030000

// <o> Code run address
// <i> Default:  0x01002000
#define APP_CODE_RUN_ADDR       0x01030000
```

6. 为使App bootloader固件能正确跳转至应用固件，需匹配两者的COMMENTS。App bootloader固件默认COMMENTS为"ble_app_temp"，因此在*custom.config.h*中设置ble_app_template_freertos固件的COMMENTS如下：

```
#define APP_FW_COMMENTS                         "ble_app_temp"
```

6 使用DFU Master升级

本章介绍以DFU Master方式进行升级的方法。

6.1 DFU Master介绍

除使用GRToolbox与设备通信完成固件升级的方式以外，在某些场景还会使用DFU Master的方式实现固件升级。GR5xx提供的SDK中关于dfu_master的内容包含dfu_master组件及基于dfu_master组件实现的dfu_master示例工程，dfu_master组件实现的固件升级方式包含UART和低功耗蓝牙两种，升级原理如下示意图所示：

- UART方式的示意图如下：

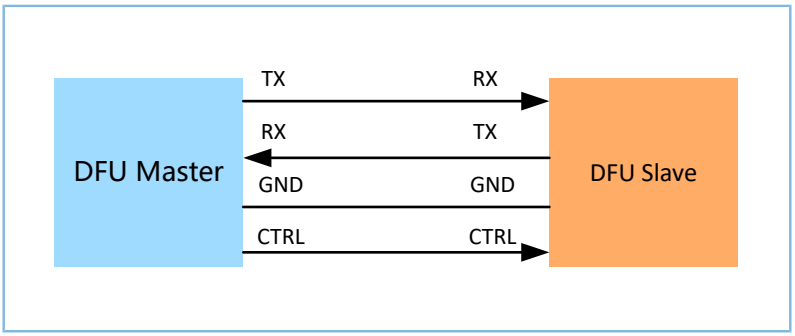

图 6-1 UART方式

- 低功耗蓝牙方式的示意图如下：

图 6-2 低功耗蓝牙方式

说明:

GR5xx的dfu_master方案只提供后台双区升级模式。UART通讯方式只支持普通模式升级，低功耗蓝牙通讯方式支持普通模式和快速模式。

6.2 DFU Master跨平台移植

GR5xx提供的dfu_master组件路径为`SDK_Folder\components\libraries\dfu_master`，将dfu_master组件移植到其他平台，主要分为如下步骤：

1. dfu_master函数接口注册，注册代码如下。

```
static dfu_m_func_cfg_t s_dfu_m_func_cfg =
{
    .dfu_m_get_img_info = dfu_m_get_img_info,
```

```
    .dfu_m_get_img_data = dfu_m_get_img_data,
    .dfu_m_send_data    = dfu_uart_data_send,
    .dfu_m_fw_read      = hal_flash_read,
    .dfu_m_event_handler = dfu_m_event_handler,
};
```

说明:

GR551x SDK V2.0.1和GR5525 SDK V0.8.0无dfu_m_fw_read接口。

dfu_m_get_img_info的函数定义如下，函数功能是获取待传输固件的image_info信息保存至img_info变量中，函数内部具体的实现需要用户结合待传输固件image_info所在的位置去实现。

```
static void dfu_m_get_img_info(dfu_img_info_t *img_info)
{

}
```

dfu_m_get_img_data的函数定义如下，函数功能是获取待传输固件起始地址为addr、长度为length的数据存入p_data 中。

```
static void dfu_m_get_img_data(uint32_t addr, uint8_t *p_data, uint16_t length)
{

}
```

dfu_uart_data_send的函数定义如下，函数功能是通过串口发送长度为length的p_data数据至串口。此处若使用低功耗蓝牙方式进行传输数据，则注册的接口需要是低功耗蓝牙数据发送函数。

```
void dfu_uart_data_send(uint8_t *p_data, uint16_t length)
{

}
```

dfu_m_fw_read的功能是读取待传输固件的固件信息里的标志位信息，在dfu_master示例工程中，因待传输固件存于芯片Flash，所以注册的接口为hal_flash_read，用户需要根据所使用的平台读取固件数据的方式来自行注册该接口。

dfu_m_event_handler是升级命令执行完毕后的事件回调函数，可参考如下方式进行实现：

```
static void dfu_m_event_handler(dfu_m_event_t event, uint8_t pre)
{
    switch(event)
    {
        case PRO_START_SUCCESS:
            APP_LOG_DEBUG("Upgrade Start");
            break;

        case PRO_FLASH_SUCCESS:
            APP_LOG_DEBUG("Upgrade Progress %d%%", pre);
            break;
```

```
        case PRO_END_SUCCESS:
            APP_LOG_DEBUG("Upgrade End");
            user_master_status_set(MASTER_IDLE);
            break;
    ......
    }
}
```

2. 升级前升级参数确定，UART方式和低功耗蓝牙方式升级前需要确定的参数不同，UART方式在升级前需要获取传输固件的固件信息及固件数据。低功耗蓝牙方式除了这两项数据以外，还需要在升级前确认是否需要使能快速模式。GR5xx在dfu_master示例工程中按如下原理确定升级前参数。

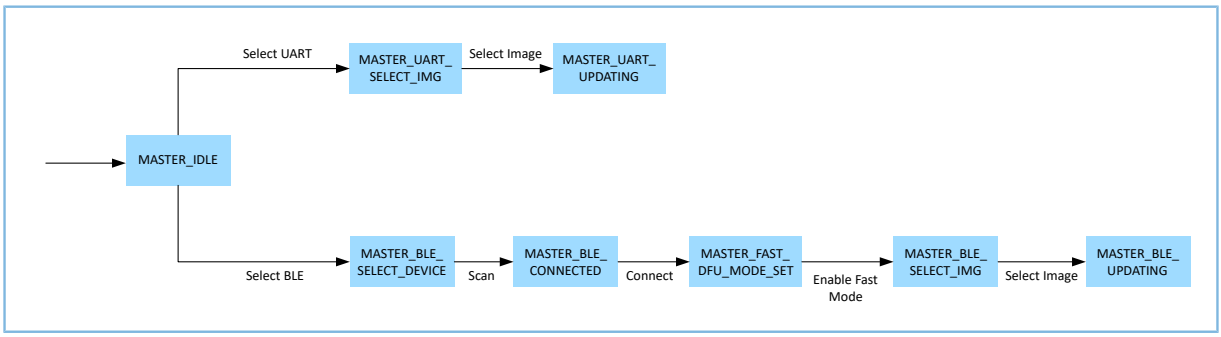

图 6-3 dfu_master升级前状态

- MASTER_IDLE：空闲状态
- MASTER_UART_SELECT_IMG：UART方式下选择固件状态
- MASTER_UART_UPDATING：UART方式下正在升级状态
- MASTER_BLE_SELECT_DEVICE：低功耗蓝牙方式下选择设备状态
- MASTER_BLE_CONNECTED：低功耗蓝牙方式下连接成功状态
- MASTER_FAST_DFU_MODE_SET：快速模式使能状态
- MASTER_BLE_SELECT_IMG：低功耗蓝牙方式下选择固件状态
- MASTER_BLE_UPDATING：低功耗蓝牙方式下正在升级状态

用户可依据上述状态图在自己的平台实现升级前的参数确定。

3. dfu_master组件移植。在将dfu_master组件移植到其他平台时，在*dfu_master.c*中需确保所包含的头文件与平台无关，保留如下两个头文件：

```
#include "dfu_master.h"
#include <string.h>
```

*dfu_master.c*的dfu_m_system_info_get函数需要目标升级设备的起始地址，GR5xx芯片的Flash起始地址如下表所示：

表 6-1 GR5xx芯片Flash起始地址

芯片型号	FLASH_START_ADDR
GR551x	0x01000000
GR5526	0x00200000
GR5525	0x00200000
GR533x	0x00200000
GR5405	0x00200000

以待升级设备为GR551x为例，在*dfu_master.c*中加入如下宏定义：

```
#define FLASH_START_ADDR    0x01000000
```

在使用低功耗蓝牙通讯方式进行升级时，需选择是否使能快速模式。使能快速模式后，Master端发送固件数据后将不会等待升级设备回复之后再发送第二帧数据。当第一帧数据发送完成后，则会开始发送第二帧，依此类推。dfu_master示例工程提供的方法是在使用快速模式发送固件数据时，在低功耗蓝牙发送完成事件回调里，每发送完一帧后将ble_send_cplt_flag置位，具体代码如下所示：

```
static void otas_c_evt_process(otas_c_evt_t *p_evt)
{
        ......
        switch (p_evt->evt_type)
        {
                ......
                case OTAS_C_EVT_TX_CPLT:
                        if (fast_dfu_mode == 0x00)
                        {
                                dfu_m_send_data_cmpl_process();
                        }
                        else if (fast_dfu_mode == 0x02 && s_program_size != 0)
                        {
                                ble_send_cplt_flag = 1;
                        }
                        break;
                ......
        }
        ......
}
```

说明:

上述代码中“fast_dfu_mode”取值说明如下：

- 0x00：表示Disable
- 0x02：表示Enable

在*dfu_master.h*文件中也需要确保头文件与平台无关，保留如下两个头文件：

```
#include <stdbool.h>
#include <stdint.h>
```

在GR551x SDK V2.0.1和GR5525 SDK V0.8.0的*dfu_master.h*文件中需加入如下boot_info_t类型定义：

```
typedef struct
{
    uint32_t bin_size;
    uint32_t check_sum;
    uint32_t load_addr;
    uint32_t run_addr;
    uint32_t xqspi_xip_cmd;
    uint32_t xqspi_speed: 4;
    uint32_t code_copy_mode: 1;
    uint32_t system_clk: 3;
    uint32_t check_image:1;
    uint32_t boot_delay:1;
    uint32_t is_dap_boot:1;
    uint32_t reserved:21;
} boot_info_t;
```

4. 添加DFU命令调度器及DFU命令解析器，DFU命令调度器需要在升级时一直循环调用，示例代码如下所示：

```
while(1)
{
    dfu_m_schedule(app_dfu_rev_cmd_cb);
}
```

app_dfu_rev_cmd_cb是DFU超时机制Count累加函数，超时机制的实现可参考5 DFU移植方式详解。

DFU命令解析器需在Master接收待升级设备回传数据的位置调用，以低功耗蓝牙为例，在如下位置进行调用：

```
static void otas_c_evt_process(otas_c_evt_t *p_evt)
{
        ......
        switch (p_evt->evt_type)
        {
                ......
                case OTAS_C_EVT_PEER_DATA_RECEIVE:
                        dfu_m_cmd_prase(p_evt->p_data, p_evt->length);
                        break;
                ......
        }
        ......
}
```

6.3 DFU Master升级操作

6.3.1 准备工作

请参考4.2 准备工作。

6.3.2 UART方式

1. 硬件配置

搭建DFU Master UART方式升级环境，GR5xx系列芯片均使用UART1进行数据传输，不同系列芯片对应的TX、RX引脚如下表所示：

表 6-2 硬件配置

硬件平台	TX引脚	RX引脚
GR5515-SK-BASIC	GPIO_30	GPIO_26
GR5526-SK-BASIC	GPIO_32	GPIO_33
GR5525-SK-BASIC	GPIO_7	GPIO_6
GR5331-SK-BASIC	GPIO_5	GPIO_6
GR5405-SK-BASIC	AON_GPIO_3	AON_GPIO_2

下面以GR5515-SK-BASIC开发板为例介绍具体升级步骤，GPIO_30是开发板UART1的TX引脚，GPIO_26是开发板UART1的RX引脚。两个开发板之间通过各自的UART1引脚进行通信，示意图如下所示。

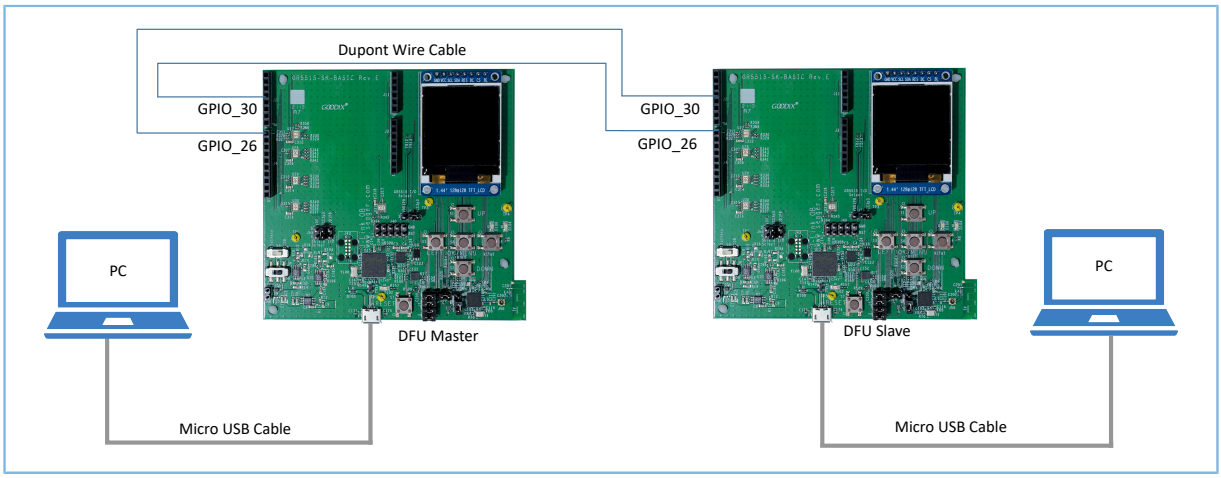

图 6-4 UART升级环境

2. 操作步骤

(1) 使用UART方式进行升级时，若当前使用RTOS环境，则DFU任务可能不是始终运行的，且当前系统可能处于睡眠状态，则需要使用图 6-1所示的CTRL控制线。

DFU Master需使用DFU功能时，通过CTRL控制线发送信号通知DFU Slave当前需要运行DFU任务。信号类型可以是一个IO电平翻转。

(2) dfu_master固件默认的Load Address和Run Address为FLASH_START_ADDR + 0x2000，使用GProgrammer工具将dfu_master固件下载至设备端，将待升级的固件也存放至设备端。

(3) 请参考4.3.3 创建目标升级固件的方式创建目标升级固件，目标升级固件的Load Address和Run Address为FLASH_START_ADDR + 0x30000。以GR551x为例，将dfu_master固件和待升级的固件同时下载至设备端，如图 6-5所示。

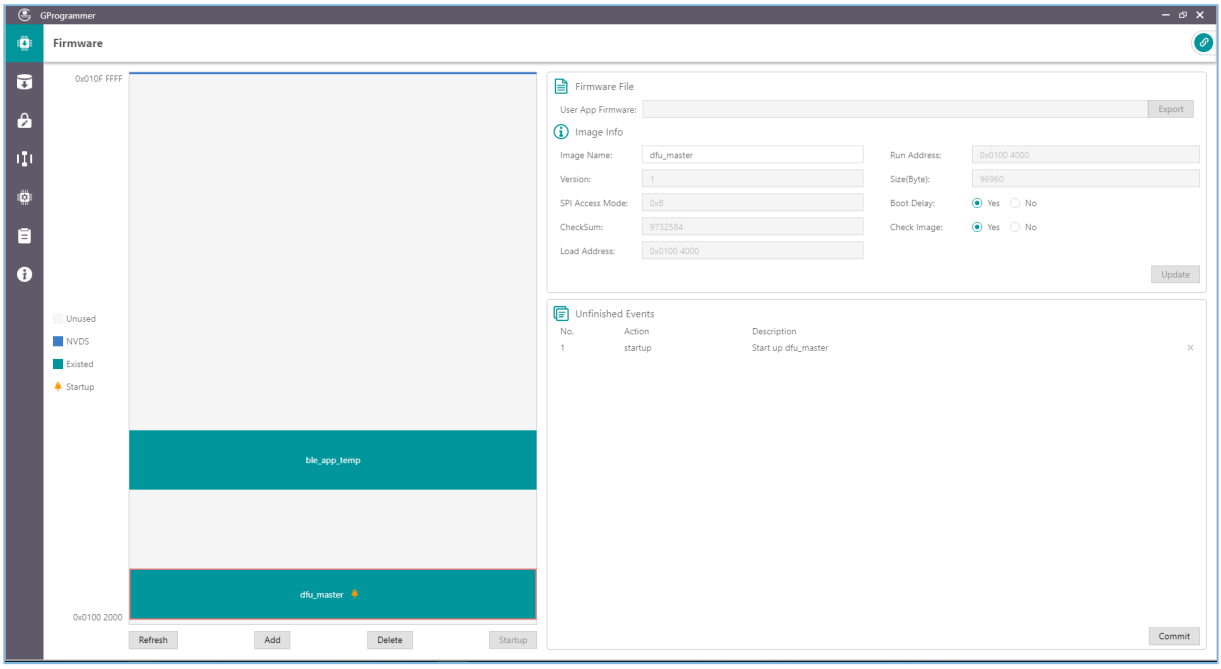

图 6-5 dfu_master和待升级固件下载

(4) DFU Slave端需要下载app_bootloader固件和ble_app_template_dfu固件，如图 6-6所示，具体下载方式请参考《GProgrammer用户手册》。

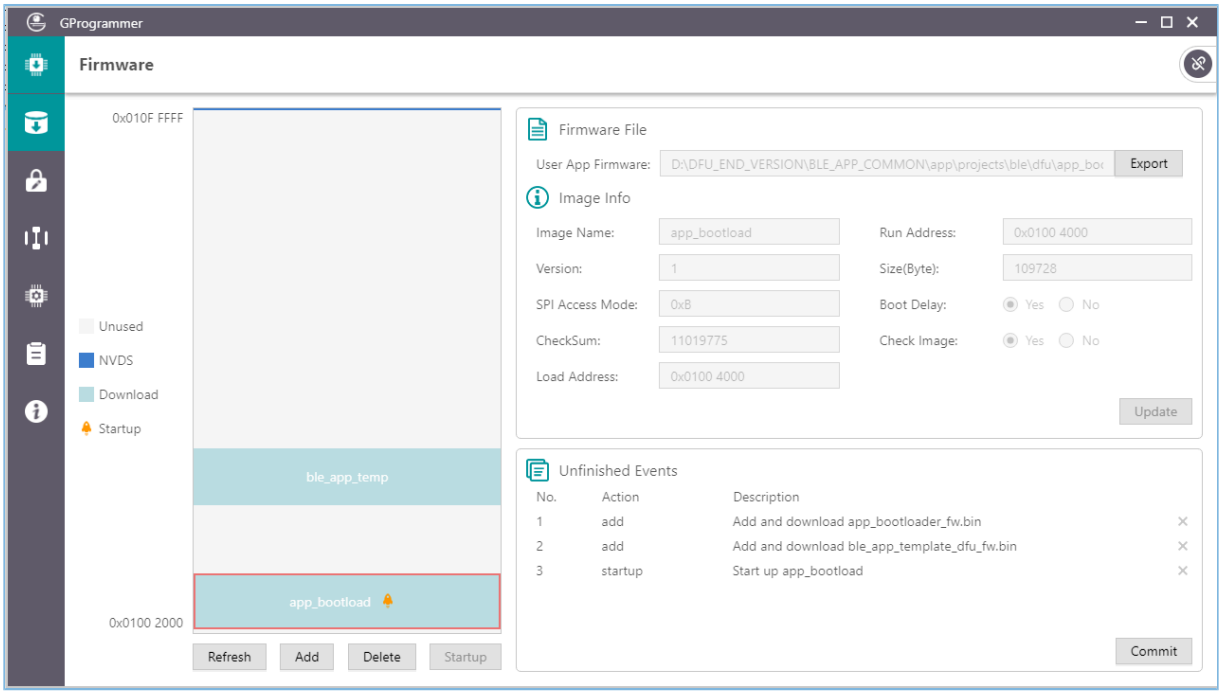

图 6-6 app_bootloader和ble_app_template_dfu固件下载

(5) 下载完成后，按表 6-3 使用杜邦线连接DFU Master和DFU Slave，DFU Master和DFU Slave的UART0通过Micro USB连接至PC端。

表 6-3 连接DFU Master和DFU Slave

DFU Master端	DFU Slave端
UART1	UART1
GPIO_30	GPIO_26
GPIO_26	GPIO_30

(6) 打开GRUart观察DFU Master Log打印情况如下图所示，可选择UART或BLE（低功耗蓝牙）两种方式进行升级，在“Tx”窗口输入“1”表示采用UART方式。

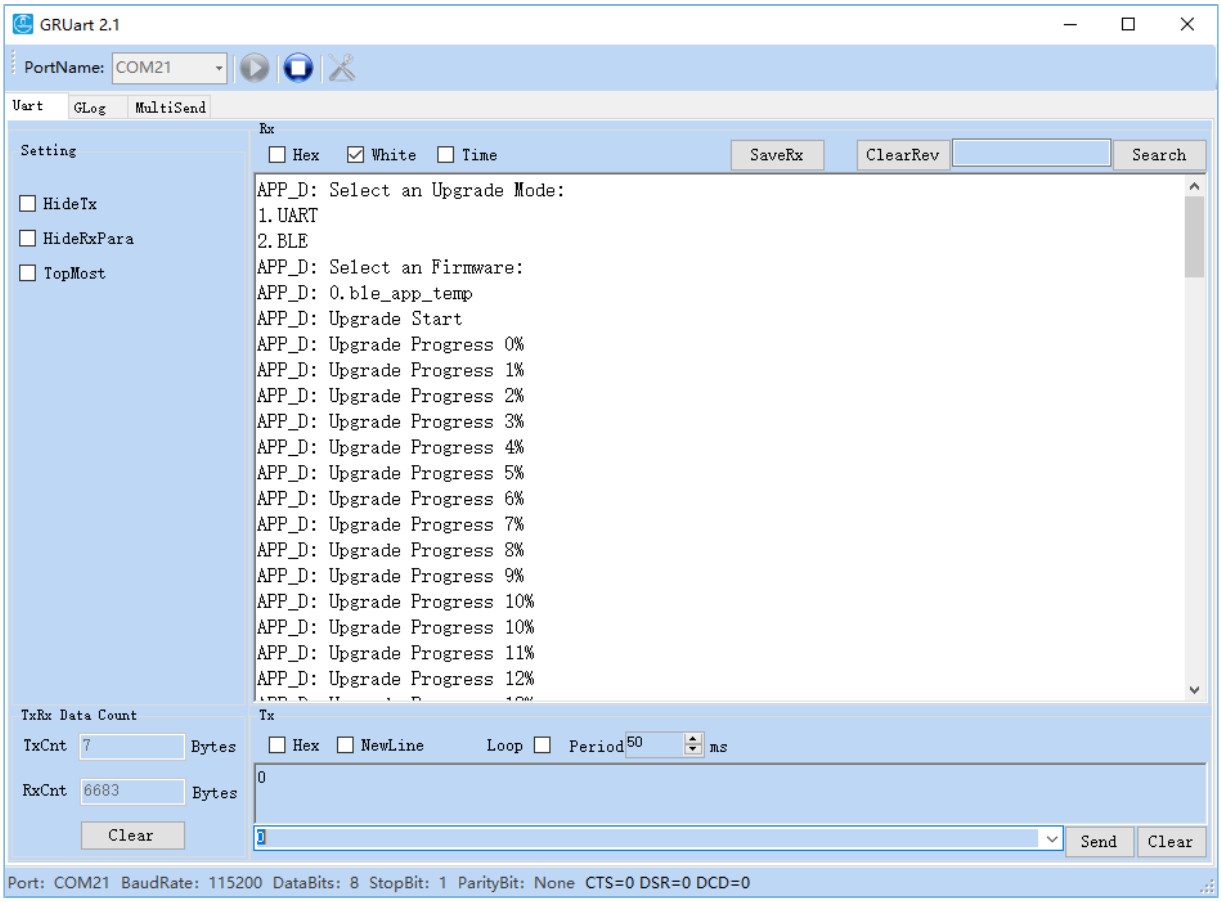

图 6-7 UART方式Log打印

(7) 选择升级的固件序号后，进入升级模式，打印升级进度。

6.3.3 低功耗蓝牙方式

若当前使用RTOS方式，DFU任务没有运行，并且系统处于休眠模式，将采用Control Point形式唤醒系统，唤醒系统后便可执行DFU任务。

低功耗蓝牙升级方式的步骤仅比UART升级方式增加一步，其余均相同，具体步骤如下：

1. 选择“BLE”升级方式。
2. 选择“Normal DFU”或“Fast DFU”。
3. 选择升级的固件序号后，进入升级模式，打印升级进度。

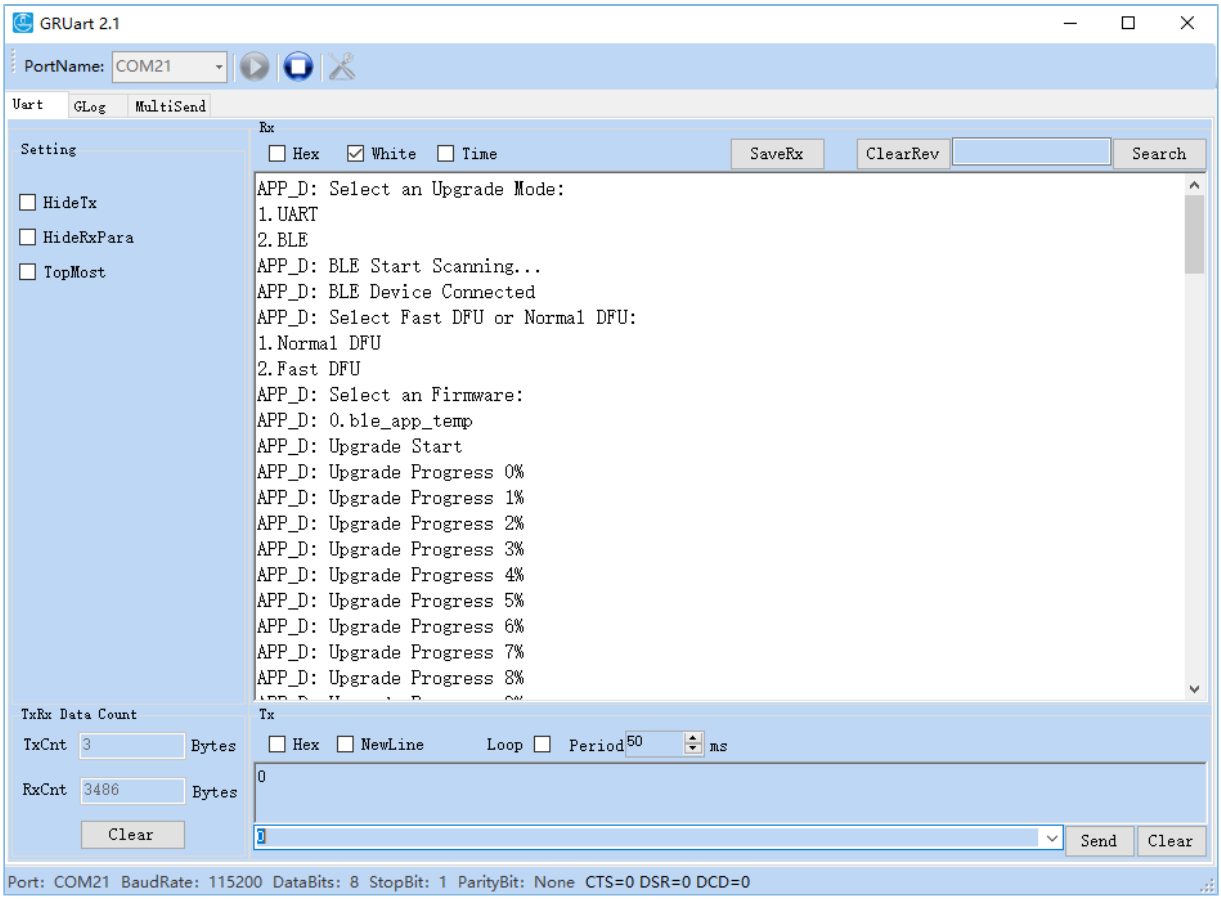

图 6-8 低功耗蓝牙方式Log打印

7 注意事项

本章介绍DFU过程中需要注意的事项。

7.1 在App bootloader跳转至App firmware之前，需将App bootloader中使用的外设反初始化

- 原因

  App bootloader跳转到App firmware之前，必须经历如下操作：

  1. 关中断
  2. 清Pending位
  3. 反初始化App bootloader用到的外设

- 处理方法

  SDK层面已执行1和2两个操作，因此为了避免固件跳转失败或者应用固件功耗异常，需要在跳转前反初始化App bootloader用到的外设。

7.2 不同系列芯片使用RTOS时，应用固件DFU任务栈大小设置不同

- 原因

  GR551x系列芯片与其他系列芯片在内部实现的DFU有差异，任务栈大小要求与其他系列芯片要求不同。

- 处理方法

  GR551x DFU任务栈至少需要分配6 KB，其他系列芯片需要分配至少1 KB的DFU任务栈。

8 附录：DFU通信协议

主机端和设备端基于DFU通信协议进行固件升级。

8.1 基础帧定义

基础帧定义了通信中最底层的数据包结构。应用数据包协议建立在基础帧之上，位于基础帧的“数据”域。如果基础帧长度超过链路通信的最大载荷，主机端需要将其分成片段发送。设备端在收到正确的帧头与数据长度后，开始处理数据。

8.1.1 帧结构定义

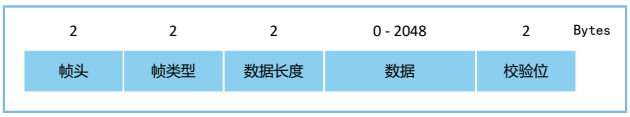

图 8-1 帧结构

- 帧头：标识帧的开始，以字符‘G’和‘D’的ASCII码值0x47和0x44表示。
- 帧类型：用于区别“数据”域中的数据类型。
- 数据长度：“数据”域的长度值。
- 数据：数据长度可变，最长为2048字节。
- 校验位：帧类型、数据长度、数据的16位校验和。

8.1.2 字节顺序定义

基础帧的数据域采用小端模式，即低字节在前、高字节在后。

8.2 附录：DFU命令集

DFU命令由主机端发送，设备端接收。DFU命令集如表 8-1 所示。

表 8-1 DFU命令说明

命令	命令代码	描述
Get Info	0x01	主机端获取芯片系统信息以及DFU版本号
Operate system info	0x27	主机端操作设备端System Configuration区域，包括读取、更新数据
DFU mode set	0x41	主机端设置设备端DFU升级模式
DFU firmware info get	0x42	主机端获取设备端固件信息
Program Start	0x23	在向设备端下载程序时，主机端发送Image Info信息以及是否使能快速模式信息
Program Flash	0x24	主机端向设备端写入固件数据
Program End	0x25	主机端使用此命令告知设备端编程数据已经发送完成
Config external flash	0x2A	配置外部Flash

命令	命令代码	描述
Get flash information	0x2B	获取Flash信息

8.2.1 Get Info命令

Get Info命令用于获取芯片的系统信息，设备端接收到此命令后，会将芯片ID、Flash、RAM及Stack版本信息发送给主机端。

8.2.1.1 主机端发送数据

表 8-2 Get Info发送数据格式

字节序号	描述	有效值	说明
0-1	帧头	0x4744	以字符’G’和’D’的ASCII码值0x47和0x44表示
2-3	帧类型	0x01	Get Info命令
4-5	数据长度	0	无内容数据
6-7	校验和	0x00-0xFF	帧类型、数据长度的16位校验和

8.2.1.2 设备端回应数据

表 8-3 Get Info的回应数据格式

字节序号	描述		有效值	说明
0-1	帧头		0x4744	以字符’G’和’D’的ASCII码值 0x47和0x44表示
2-3	帧类型		0x01	Get Info命令
4-5	数据长度		1或9	如果获取信息失败，数据域只包含应答
6	数据内容	应答	0x01或0x02	• 0x01：获取系统信息成功 • 0x02：获取系统信息失败
7		Stack Major	0x00-0xff	主版本号
8		Stack Minor	0x00-0xff	副版本号
9-10		Stack Build	0x00-0xff	Build Number
11-14		Stack SVN	0x00-0xff	SVN Number
15		SDK Major	0x00-0xff	主版本号
16		SDK Minor	0x00-0xff	副版本号
17-18		SDK Build	0x00-0xff	Build Number
19-22		SDK SVN	0x00-0xff	SVN Number
23		DFU Version	0x02或数据位无效	DFU版本号获取 • 0x02：本文介绍的DFU方案 • 数据位无效：旧版本DFU方案
24-25	校验和		0x00-0xFF	帧类型、数据长度、数据内容的16位校验和

8.2.2 Operate System Info命令

主机端使用此命令操作设备端System Configuration区域数据，包括读取、更新数据。

8.2.2.1 主机端发送数据

表 8-4 Operate System Info发送数据格式

字节序号	描述		有效值	说明
0-1	帧头		0x4744	以字符’G’和’D’的ASCII码值0x47和0x44表示
2-3	帧类型		0x27	Operate System Info命令
4-5	数据长度		N	数据域内容长度
6	数据内容	操作命令	0x00或0x01	• 0x00：获取System Configuration区域数据 • 0x01：更新System Configuration区域数据
7-10		起始地址	0x00-0xFF	地址范围必须是System Configuration区域有效地址
11-12		数据长度	0x00-0xFF	从起始地址开始读取的数据长度，不能超过System Configuration区域
13-N		更新内容	0x00-0xFF	如果是获取数据命令，不需要更新内容段
N+1-N+2	校验和		0x00-0xFF	帧类型、数据长度、数据内容的16位校验和

说明:

上表中的N表示数据域的长度可变：

- 若是更新数据命令，则N的取值为： 14 ～ 1036字节
- 若是读取数据命令，则数据域为固定长度：7字节

8.2.2.2 设备端回应数据

表 8-5 Operate System Info回应数据格式

字节序号	描述		有效值	说明
0-1	帧头		0x4744	以字符’G’和’D’的ASCII码值 0x47和0x44表示
2-3	帧类型		0x27	Operate System Info命令
4-5	数据长度		N	数据域内容长度
6	数据内容	应答	0x01或0x02	• 0x01：读取成功 • 0x02：读取失败
7		操作命令	0x00、0x10、0x01、0x11	• 0x00：读取System Configuration，芯片为非加密 • 0x10：读取System Configuration，芯片为加密 • 0x01：更新System Configuration，芯片为非加密 • 0x11：更新System Configuration，芯片为加密
8-11		起始地址	0x00-0xFF	如果是更新数据指令，此数据段无效
12-13		数据长度	0x00-0xFF	

字节序号	描述		有效值	说明
14-N		数据	0x00-0xFF	
N+1-N+2	校验和		0x00-0xFF	帧类型、数据长度、数据内容的16位校验和

8.2.3 DFU Mode Set命令

该命令用于手机APP设置升级模式，有两种升级模式可供选择：后台双区升级模式和非后台单区升级模式。

8.2.3.1 主机端发送数据

表 8-6 DFU Mode Set的发送数据格式

字节序号	描述	有效值	说明
0-1	帧头	0x4744	以字符’G’和’D’的ASCII码值 0x47和0x44表示
2-3	帧类型	0x41	DFU Mode设置
4-5	数据长度	0x01	1个字节的数据内容
6	数据内容	0x01 0x02	• 0x01：后台双区升级模式 • 0x02：非后台单区升级模式
7-8	校验和	0x00-0xFF	帧类型、数据长度、数据内容的16位校验和

8.2.3.2 设备端回应数据

当前命令设备端不进行回应。

8.2.4 DFU Firmware Info Get命令

该指令用于主机端获取固件端APP Info区域存储的固件信息。

8.2.4.1 主机端发送数据

表 8-7 DFU Firmware Info Get发送数据格式

字节序号	描述	有效值	说明
0-1	帧头	0x4744	以字符’G’和’D’的ASCII码值 0x47和0x44表示
2-3	帧类型	0x42	DFU Firmware Info获取
4-5	数据长度	0	无内容数据
6-7	校验和	0x00-0xFF	帧类型、数据长度的16位校验和

8.2.4.2 设备端回应数据

表 8-8 DFU Firmware Info Get回应数据格式

字节序号	描述		有效值	说明
0-1	帧头		0x4744	以字符’G’和’D’的ASCII码值 0x47和0x44表示
2-3	帧类型		0x42	DFU Firmware Info获取
4-5	数据长度			回应的数据长度
6	数据内容	应答	0x01 0x02	• 0x01：获取信息成功 • 0x02：获取信息失败
7-10		Copy_load_addr		升级固件的存储起始地址
11		Run_position	0x00 0x01	• 0x00：表示当前运行在App bootloader • 0x01：表示当前运行在App firmware
12-59		Image_Info		APP Info区域存放的固件信息
59-61	校验和		0x00-0xFF	帧类型、数据长度、数据内容的16位校验和

8.2.5 Program Start命令

Program Start指令用于下发固件信息以及是否使能快速模式。

8.2.5.1 主机端发送数据

表 8-9 Program Start的发送数据格式

字节序号	描述	有效值	说明
0-1	帧头	0x4744	以字符’G’和’D’的ASCII码值 0x47和0x44表示
2-3	帧类型	0x23	Program Start命令
4-5	数据长度	N	• 若Program的对象为固件，则数据域为41个字节，包括1个字节的Flash类型和长度为40个字节的Image Info • 若Program的对象为数据，则数据域为9个字节，包括1个字节的Flash类型、4个字节起始地址和4个字节数据内容
6	升级类型	0x00 0x01 0x02 0x03 0x10 0x20 0x12 0x22	• 0x00：在内部Flash使用普通模式升级非加密非加签固件 • 0x10：在内部Flash使用普通模式升级仅加签非加密固件 • 0x20：在内部Flash使用普通模式升级加密加签固件 • 0x02：在内部Flash使用快速模式升级非加密非加签固件 • 0x12：在内部Flash使用快速模式升级非加密仅加签固件 • 0x22：在内部Flash使用快速模式升级加密加签固件 • 0x01：在外部Flash使用普通模式升级资源 • 0x03：在外部Flash使用快速模式升级资源
7-N	数据内容	0x00-0xFF	写入设备端的数据内容

字节序号	描述	有效值	说明
N+1-N+2	校验和	0x00-0xFF	帧类型、数据长度、数据内容的16位校验和

8.2.5.2 设备端回应数据

设备端回应数据会根据具体升级模式，回复的数据不同。

- 普通模式设备端回应的数据帧如表 8-10 所示。

表 8-10 Program Start普通模式的回应数据格式

字节序号	描述	有效值	说明
0-1	帧头	0x4744	以字符’G’和’D’的ASCII码值 0x47和0x44表示
2-3	帧类型	0x23	Program Start命令
4-5	数据长度	0x01	数据域内容长度
6	应答	0x01或0x02	◦ 0x01：成功 ◦ 0x02：失败
7-8	校验和	0x00-0xFF	帧类型、应答、数据长度的16位校验和

- 快速模式设备端回应的数据帧如表 8-11 所示。

表 8-11 Program Start快速模式的回应数据格式

字节序号	描述	有效值	说明
0-1	帧头	0x4744	以字符’G’和’D’的ASCII码值 0x47和0x44表示
2-3	帧类型	0x23	Program Start命令
4-5	数据长度	0x04	数据域内容长度
6	应答	0x01或0x02	◦ 0x01：成功 ◦ 0x02：失败
7	擦除状态	0x00 0x01 0x02 0x03 0x04 0x05 0x06	0x00：待擦除的Flash起始地址不是4 KB对齐 0x01：开始擦除 0x02：正常擦除 0x03：擦除完成，可以下发数据 0x04：擦除区域与当前运行区域有重叠 0x05：擦除失败 0x06：擦除区域不存在
8-9	已擦除页数	0x00-0xFF	已经擦除的页数
10-11	校验和	0x00-0xFF	帧类型、应答、数据长度、擦除状态、已擦除页数的16位校验和

8.2.6 Program Flash命令

只有普通DFU会使用到 Program Flash命令。对于快速模式DFU，在写入固件时，采用直接下发数据的形式，没有数据帧格式。

8.2.6.1 主机端发送数据

表 8-12 Program Flash的发送数据格式

字节序号	描述		有效值	说明
0-1	帧头		0x4744	以字符’G’和’D’的ASCII码值 0x47和0x44表示
2-3	帧类型		0x24	Program Flash命令
4-5	数据长度		N	最长2 KB
6	数据内容	Program类型	0x00 0x01 0x02 0x10 0x11 0x12	0x00：对内部Flash指定地址的页进行擦除后存储。 0x01：对内部Flash按照Program Start命令下发的Image Info信息进行Flash存储。 0x02：对内部Flash直接调用Flash write接口写入数据。 0x10：对外部Flash指定地址的页进行擦除后存储。 0x11：对外部Flash按照Program Start命令下发的Image Info信息进行Flash存储。 0x12：对外部Flash直接调用Flash write接口写入数据。
7-10		起始地址	0x00-0xFF	设备端Flash有效地址
11-12		数据长度	0x00-0xFF	建议数据长度最长1024 bytes
13-N		数据	0x00-0xFF	主机端发送至设备端的数据
N+1-N+2	校验和		0x00-0xFF	帧类型、数据长度、数据内容的16位校验和

8.2.6.2 设备端回应数据

表 8-13 Program Flash的回应数据格式

字节序号	描述	有效值	说明
0-1	帧头	0x4744	以字符’G’和’D’的ASCII码值 0x47和0x44表示
2-3	帧类型	0x24	Program Flash命令
4-5	数据长度	0x01	应答1字节
6	数据内容	0x01或0x02	• 0x01：成功 • 0x02：失败
7-8	校验和	0x00-0xFF	帧类型、数据长度、数据内容的16位校验和

8.2.7 快速模式写入固件

8.2.7.1 主机端发送数据

快速模式下，DFU在写入每一帧数据时，直接以蓝牙最大所能传输的大小进行传输的（MTU），直接下发的是固件数据，因此，不存在数据帧格式。

8.2.7.2 设备端回应数据

表 8-14 快速模式写入固件的回应数据格式

字节序号	描述	有效值	说明
0-1	帧头	0x4744	以字符’G’和’D’的ASCII码值 0x47和0x44表示
2-3	帧类型	0xFF	Fast DFU写入Flash完毕命令
4-5	数据长度	0x01	数据域内容长度
6	应答	0x01或0x02	• 0x01：成功 • 0x02：失败
7-8	校验和	0x00-0xFF	帧类型、数据长度、应答的16位校验和

8.2.8 Program End命令

主机端使用此命令告知设备端编程数据已发送完成。

8.2.8.1 主机端发送数据

表 8-15 Program End的发送数据格式

字节序号	描述		有效值	说明
0-1	帧头		0x4744	以字符’G’和’D’的ASCII码值 0x47和0x44表示
2-3	帧类型		0x25	Program End命令
4-5	数据长度		0x05	数据域内容长度
6	数据内容	复位类型标志	0x00 0x01 0x02 0x12	• 0x00：仅将固件Image Info存储到SCA区域的Img Info区域，复位后不会运行该固件 • 0x01：将固件Image Info存储到Flash APP Info区域，复位后会立即运行该固件 • 0x02：下载数据到内部Flash时，不会对APP Info和SCA区域进行操作 • 0x12：下载数据到外部Flash
7-10		编程文件的校验和	0x00-0xFF	下发文件的校验和值
11-12	校验和		0x00-0xFF	帧类型、数据长度、数据内容的16位校验和

8.2.8.2 设备端回应数据

设备端回应的数据会根据是否为快速模式而不同，如果是快速模式，多回复一个固件数据校验和，用于手机端判断当前下发的固件是否正确。具体的数据帧格式如下所示：

表 8-16 Program End的回应数据格式

字节序号	描述	有效值	说明
0-1	帧头	0x4744	以字符’G’和’D’的ASCII码值0x47和0x44表示
2-3	帧类型	0x25	Program End命令
4-5	数据长度	0x04	数据域内容长度
6	应答	0x01或0x02	• 0x01：成功 • 0x02：失败
7-10	固件数据校验和（快速模式DFU）	0x00-0xFF	返回固件端计算的校验和
11-12	校验和	0x00-0xFF	帧类型、应答、数据长度、数据内容的16位校验和

8.2.9 配置外部Flash命令

主机端使用此命令配置设备端外部Flash SPI接口。

8.2.9.1 主机端发送数据

表 8-17 配置外部Flash发送数据格式

字节序号	描述		有效值	说明
0-1	帧头		0x4744	以字符’G’和’D’的ASCII码值0x47和0x44表示
2-3	帧类型		0x2A	配置外部Flash命令
4-5	数据长度		N	数据域内容长度
6	外部flash类型		0x01或0x02	• 0x01：SPI Flash • 0x02：QSPI Flash
7-9	数据内容	CS IO TYPE	00-04	GPIO类型选择
		CS PIN	00-15	GPIO PIN选择
		CS IO MUX	00-09	PIN MUX选择
10-12		CLK IO TYPE	00-03	GPIO类型选择
		CLK PIN	00-31	GPIO PIN选择
		CLK IO MUX	00-09	PIN MUX选择
13-15		MOSI(IO0) IO TYPE	00-03	GPIO 类型选择
		MOSI(IO0) PIN	00-31	GPIO PIN选择
		MOSI(IO0)IO MUX	00-09	PIN MUX选择

字节序号	描述		有效值	说明
16-18		MISO(IO1)IO TYPE	00-03	GPIO 类型选择
		MISO(IO1) PIN	00-31	GPIO PIN选择
		MISO(IO1)IO MUX	00-09	PIN MUX选择
19-21		IO2IO TYPE	00-03	GPIO 类型选择，仅QSPI有效
		IO2 PIN	00-31	GPIO PIN选择，仅QSPI有效
		IO2 IO MUX	00-09	PIN MUX选择，仅QSPI有效
22-24		IO3IO TYPE	00-03	GPIO 类型选择，仅QSPI有效
		IO3 PIN	00-31	GPIO PIN选择，仅QSPI有效
		IO3 IO MUX	00-09	PIN MUX选择，仅QSPI有效
25		QSPI ID	00-02	QSPI Module ID，仅QSPI有效
26-27	校验和		0x00-0xFF	帧类型、数据长度、外部Flash类型、数据内容的16位校验和

8.2.9.2 设备端回应数据

表 8-18 配置外部Flash回应数据格式

字节序号	描述	有效值	说明
0-1	帧头	0x4744	以字符’G’和’D’的ASCII码值0x47和0x44表示
2-3	帧类型	0x2A	初始化外部Flash命令
4-5	数据长度	0x01	应答1字节
6	数据内容	0x01、0x02	• 0x01：初始化成功 • 0x02：初始化失败
7-8	校验和	0x00-0xFF	帧类型、数据长度、数据内容的16位校验和

8.2.10 获取Flash信息命令

主机端使用此命令获取内外部Flash信息，包括Flash ID和Flash Size。外部Flash Size通过SFDP（Serial Flash Discoverable Parameters）协议获取。凡是支持SFDP协议的Flash芯片，都可通过此命令获取Flash Size。

8.2.10.1 主机端发送数据

表 8-19 获取Flash命令发送数据格式

字节序号	描述	有效值	说明
0-1	帧头	0x4744	以字符’G’和’D’的ASCII码值0x47和0x44表示
2-3	帧类型	0x2B	获取外部Flash ID值
4-5	数据长度	0x01	长度为1

字节序号	描述	有效值	说明
6	Flash类型	0x00或0x01	• 0x00：内部Flash • 0x01：外部Flash
7-8	校验和	0x00-0xFF	帧类型、数据长度、Flash类型的16位校验和

8.2.10.2 设备端回应数据

表 8-20 获取Flash命令回应数据格式

字节序号	描述		有效值	说明
0-1	帧头		0x4744	以字符’G’和’D’的ASCII码值 0x47和0x44表示
2-3	帧类型		0x2B	获取外部Flash ID值
4-5	数据长度		0x09	应答1字节
6	应答		0x01、0x02	• 0x01：读取成功 • 0x02：读取失败
7-14	内容	Flash ID	0x00-0xFF	Flash ID值
		Flash Size	0x00-0xFF	Flash Size值
14-15	校验和		0x00-0xFF	帧类型、数据长度、应答、数据内容的16位校验和