GigaDevice MCU

GD32 SBSFU 使用指南

版本 1.0

简介

安全启动和安全固件更新确保设备在启动过程中的完整性和真实性，同时保证了只有受信任的固件才能被加载和执行。本文档描述了基于 GD32 MCU 的安全启动与安全固件更新功能实现，以及使用方法。

目录

图

表

1 术语与缩写

本文档中使用的专业术语与缩写如*表 1.1. 常用缩略语对照表*所示。

表 1.1. 常用缩略语对照表

缩略语	全称	中文术语
API	Application programming interface	应用程序接口
IBL	Immutable bootloader	不可变的引导加载程序
IDE	Integrated Development Environment	集成开发环境
ISP	In system programming	在系统可编程
MBL	Main bootloader	主要的引导加载程序
OTP	One time programmable	一次性可编程器件
ROTPK	Root of trust public key	信任根公钥
SBSFU	Secure boot and secure firmware updates	安全启动与安全固件更新

2 SBSFU 简介

2.1 简介

当下设备运行环境的复杂性与多变性不断增加，这使得预防和避免设备被攻击和破坏变得尤为重要。鉴于此，设备的安全设计和实施就变得不可或缺。其中安全启动和安全固件更新作为设备安全运行的关键组成部分，共同确保设备在启动过程中的完整性、真实性和稳定性。

2.2 特性

- 固定且不可变更的启动入口（唯一的启动入口）；
- 仅允许执行经过数字签名验证通过完整性和真实性的软件代码；
- 安全引导代码尽可能简单、可靠和通用；
- 受控的代码更新（更新方式和更新内容）；
- 仅允许对通过完整性和真实性检查的固件进行更新。

2.3 安全启动

安全启动是一种在设备引导过程中实施的安全机制，旨在确保设备只能启动经过数字签名验证通过完整性和真实性的软件。安全启动的特性如下：

- SPC 或者 EFUSE 保证的唯一启动入口；
- 使用 SHA256 和非对称加密验证完整性和真实性（目前使用 ECDSA-Secp256r1-SHA256）；
- 随机的启动时间；
- FWDG、侵入检测等支持；
- 下一阶段代码版本的检查；
- 状态信息的交付；
- mbedtls 加解密库的支持；
- 导出 API；
- 硬件加速引擎的支持（HAU、CAU、PKCAU 和 TRNG）；
- SPC 或 EFUSE 禁止调试和 SRAM 启动执行代码。

2.4 安全固件更新

安全固件更新指的是仅允许对通过完整性和真实性检查的固件进行更新。安全固件更新的特性如下：

- 提供串口支持的 Ymodem 协议的代码更新；
- 更新内容的完整性和真实性检查，只有检查通过的固件才能被更新；
- 防回滚和回退的支持。

3 SBSFU 工程简介

3.1 简介

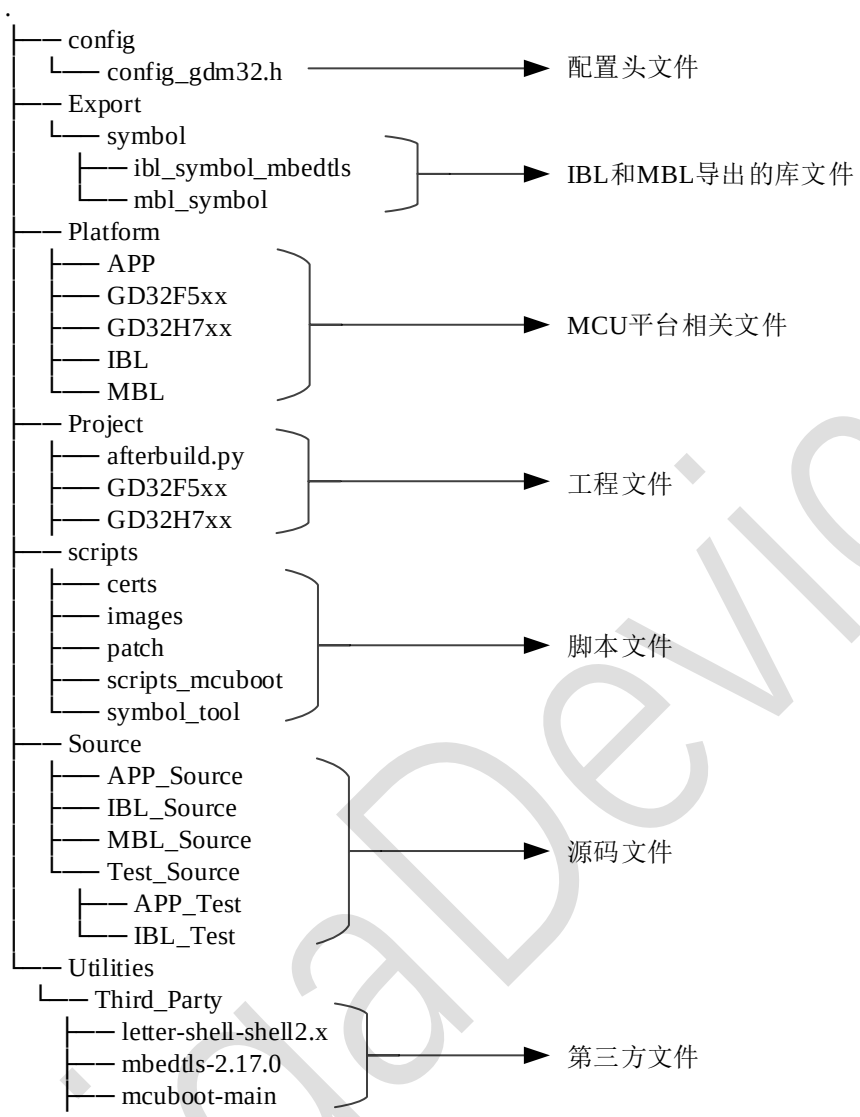
SBSFU 由三部分代码构成：IBL（固定的引导代码）、MBL（主要的引导代码）和 APP（用户代码）。设备复位后首先运行 IBL 代码，然后由 IBL 验证和启动 MBL，之后再由 MBL 验证和启动用户代码，完成启动流程。

3.2 工程结构

SBSFU 工程目录结构可以参考*图 3.1. SBSFU 工程目录结构*。其中主要部分功能如下：

- Config 目录：包含了全局配置头文件，该文件会在程序编译和脚本程序中使用；
- Export 目录：包含了 IBL 和 MBL 的导出库文件，IBL 库文件可以在 MBL 和 APP 中调用，MBL 导出库文件可以在 APP 中使用；
- Platform 目录：包含了 MCU 平台关联的源码文件；
- Project 目录：包含了工程文件，目前只支持使用 KEIL IDE 进行工程管理；
- Scripts 目录：主要包含了工程中会使用到的脚本程序；
- Source 目录：包含了主要的实现代码和测试代码文件；
- Utilities 目录：第三方源码存放的目录，如 mbedtls、shell 和 MCUBOOT 源代码。

图 3.1. SBSFU 工程目录结构

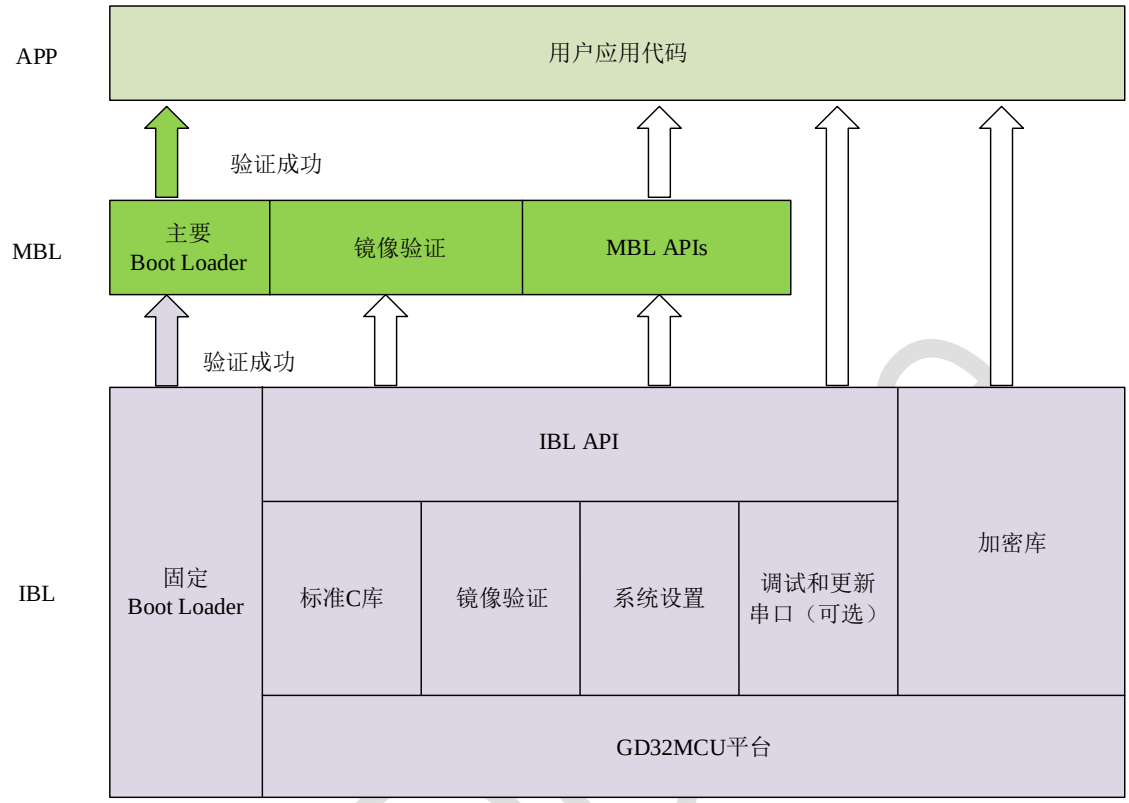


3.3 软件框架

SBSFU 工程的软件框架，如*图 3.2. 软件框架*所示，工程主要由三个部分组成：

- IBL：包含了标准的 C 库、加解密库、IBL 的 API、串口 Ymodem 更新代码，验证并执行 MBL
- MBL：包含主要的引导启动功能，由 IBL 验证和执行，MBL 也会对 APP 镜像进行验证，验证通过后会执行 APP；
- APP：用户代码，由 MBL 验证和执行。

图 3.2. 软件框架

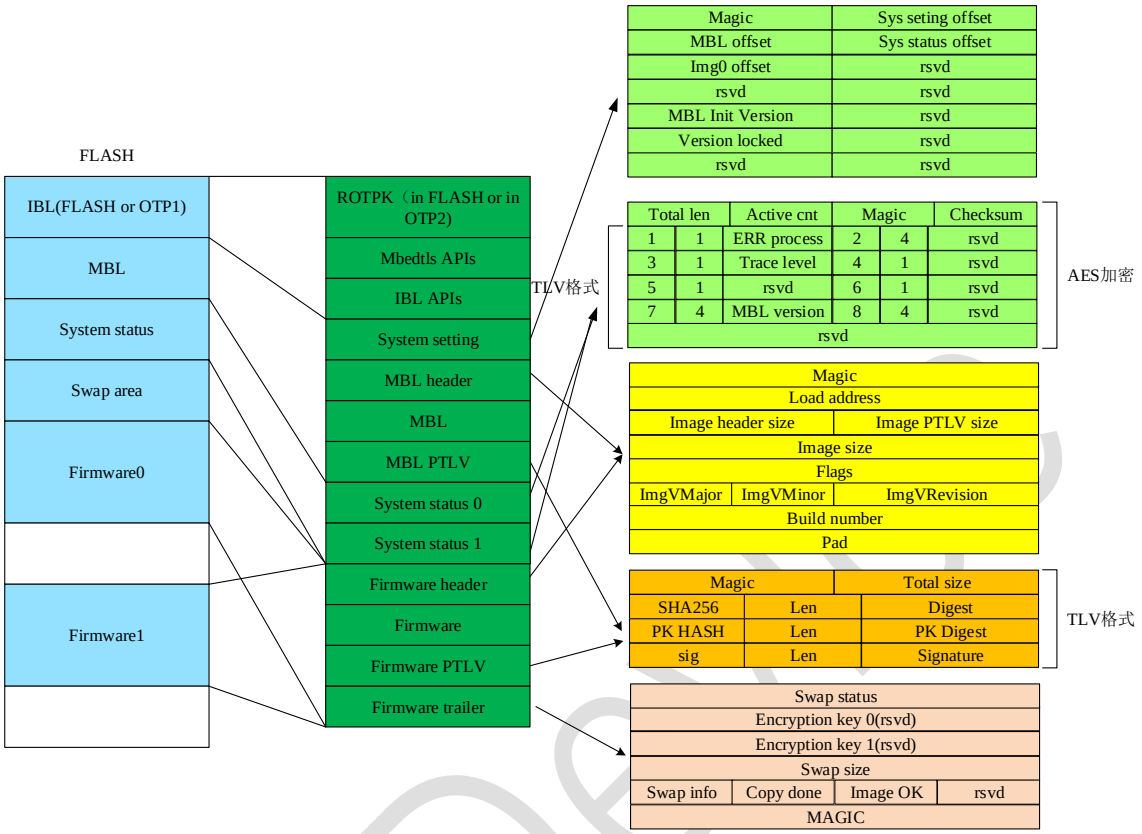


3.4 FLASH 的布局

SBSFU 工程的 FLASH 布局，如*图 3.3. FLASH 布局*所示，其中：

- IBL 在芯片上可以执行在 FLASH 或者 OTP1 区域中（如 GD32F5xx 支持将 IBL 加载到 OTP1 区域执行），默认是存放在 FLASH 中的；ROTPK 也是默认存放在 FLASH 中，但在 GD32F5xx 上支持设置到 OTP2 区域中。

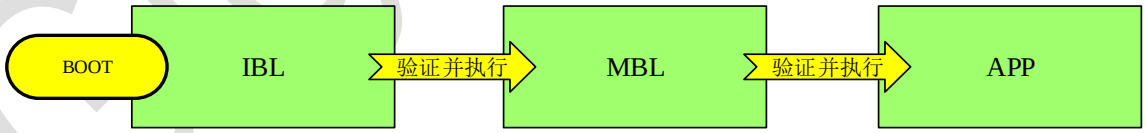
图 3.3. FLASH 布局



3.5 信任链和信任链的实施

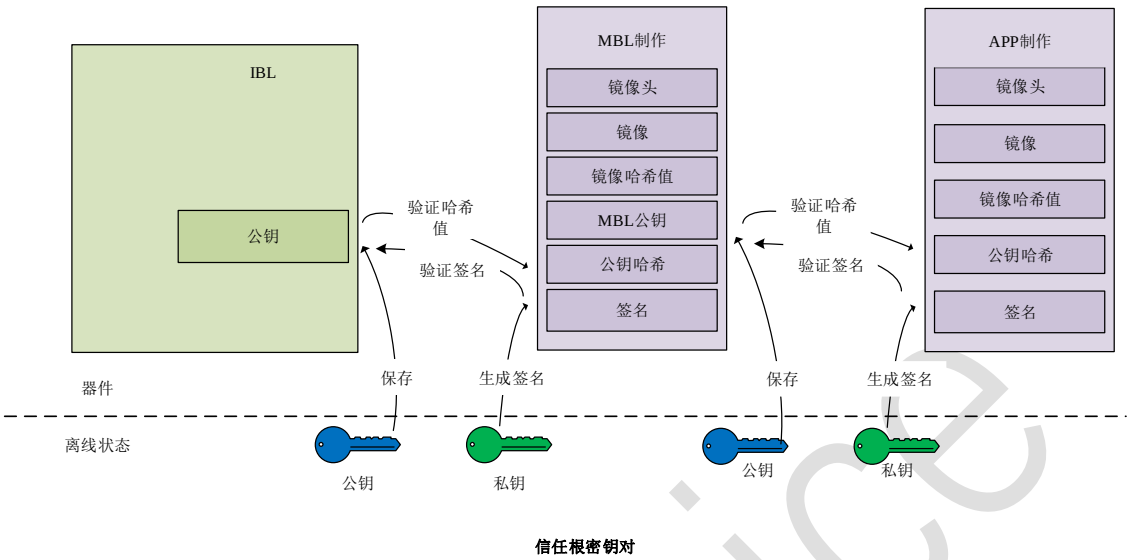
SBSFU 工程的信任链如*图 3.4. 信任链*所示，系统启动后会最先运行 IBL 代码，之后会有 IBL 验证 MBL 代码，验证通过后会去执行 MBL 代码，MBL 代码会去验证 APP 代码，验证通过之后再去执行 APP 代码。

图 3.4. 信任链



SBSFU 工程的信任链实施如*图 3.5. 信任链的实施*所示。以 IBL 为例，使用非对称的 SECP256R1 进行签名和验签，公钥存放于 FLASH 或者 OTP2 中，公钥的 SHA256 的哈希值会附加到 MBL 镜像的后面，一起附加的信息还有 MBL 的 SHA256 哈希值和使用私钥对 MBL 镜像的签名值。IBL 会通过验证公钥的哈希来判断公钥是否有效，通过镜像的哈希来判断镜像是否有效，最后通过签名信息来完成镜像完整性和真实性的验证。MBL 验证 APP 的流程和 IBL 验证的流程类似，目前 IBL 和 MBL 验证时使用的是相同的公钥。

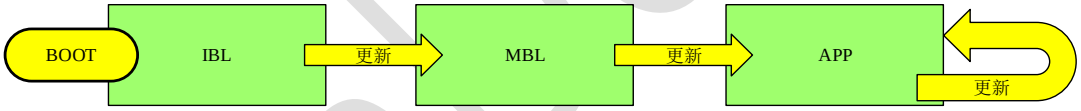
图 3.5. 信任链的实施



3.6 固件更新的实施

SBSFU 工程支持使用串口 YMODEM 协议的固件更新，固件更新的实施如*图 3.6. 固件更新*所示。

图 3.6. 固件更新



系统启动后会最先运行 IBL 代码，如果有更新需求 IBL 代码可以通过串口更新 MBL 固件区域的代码。APP 的固件会分为两个区域“Firmware0”和“Firmware1”区域，其中“Firmware0”区域为执行区域，“Firmware1”区域为固件的更新区域。如果 APP 有固件更新的需求 MBL 可以通过串口将 APP 的固件更新至更新区域，同时 APP 也支持将 APP 固件更新至 APP 固件更新区域。在每次完成固件更新后系统会由软件复位重新启动。

4 IBL 工程

IBL 工程是 SBSFU 工程的信任根之一，系统启动后应该首先开始执行的 IBL 工程，之后由 IBL 工程执行通过完整性和真实性检查的软件代码，完成系统的启动流程。同时 IBL 集成了 mbedtls 加解密库、芯片驱动和固件更新等基础安全代码。其主要功能和特点如下：

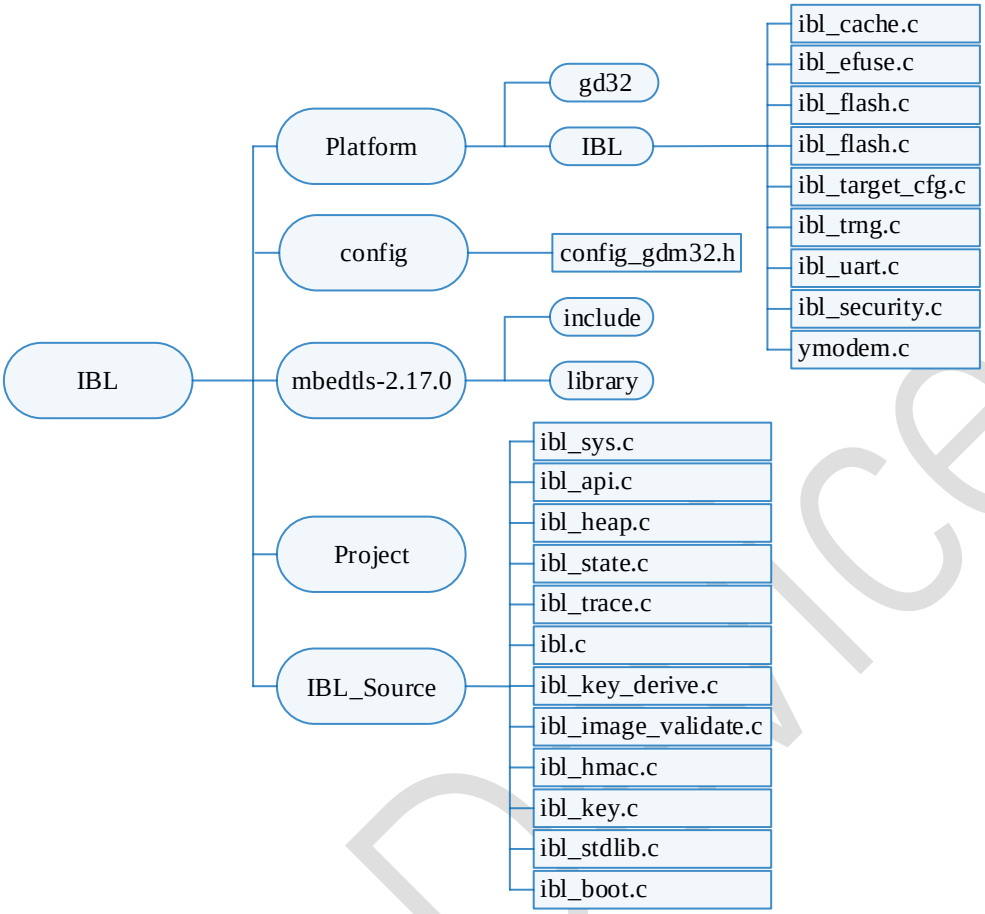
复位后应当首先启动并被执行的代码；

- 支持 mbedtls 加解密库，支持加解密库的导出；
- 串口 Ymodem 协议实现的固件更新；
- 直接获取 ROTPK；
- 支持执行 MBL 之前对 MBL 的完整性和真实性的验证；
- 支持对 MBL 版本的管理；
- 实现设备级的安全功能配置和检测；
- 导出的 IBL API 和 IBL 初始化状态信息的交付。

4.1 IBL 工程代码文件结

IBL 工程代码的文件结构如*图 4.1. IBL 工程代码文件结构*所示。其中主要包含了芯片底层的驱动库、mbedtls 加解密库、芯片驱动层代码、标准的 C 库、API 代码、镜像的验签和镜像更新跳转代码。

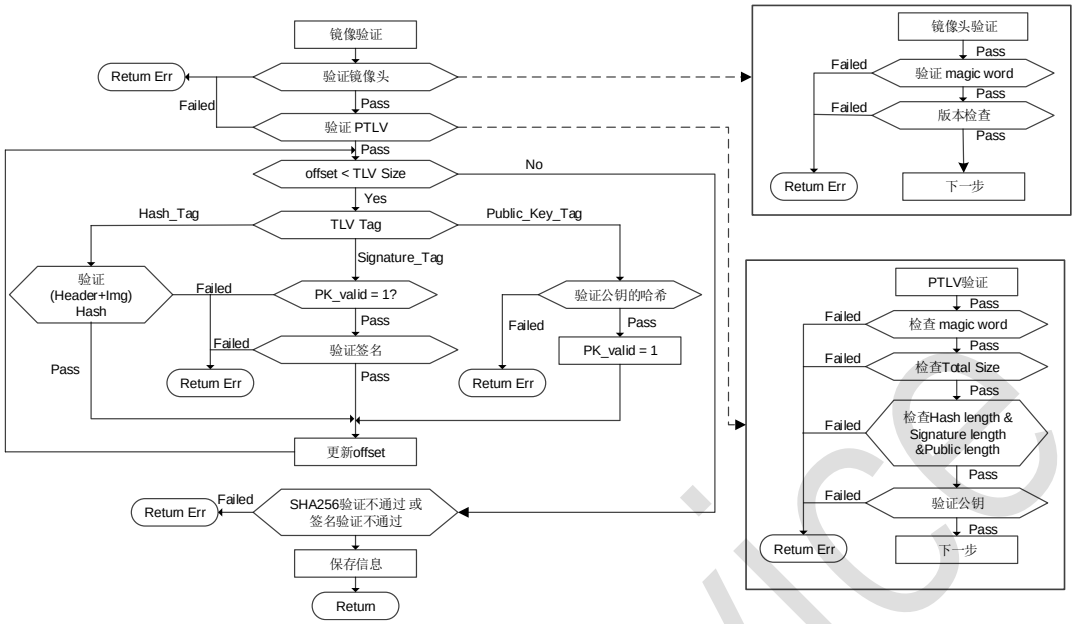
图 4.1. IBL 工程代码文件结构



4.2 镜像的验签流程

固件验签的流程如*图 4.2. 验签的流程*所示。首先验证镜像头是否正确，分别检查 Magic word 和版本号。然后进行 TLV 验证，分别检查 Magic word、Total Size、哈希/签名/公钥长度、公钥。根据系统设置的偏移值判断各部分是否校验完成，当完成后保存信息后进行下一步。

图 4.2. 验签的流程



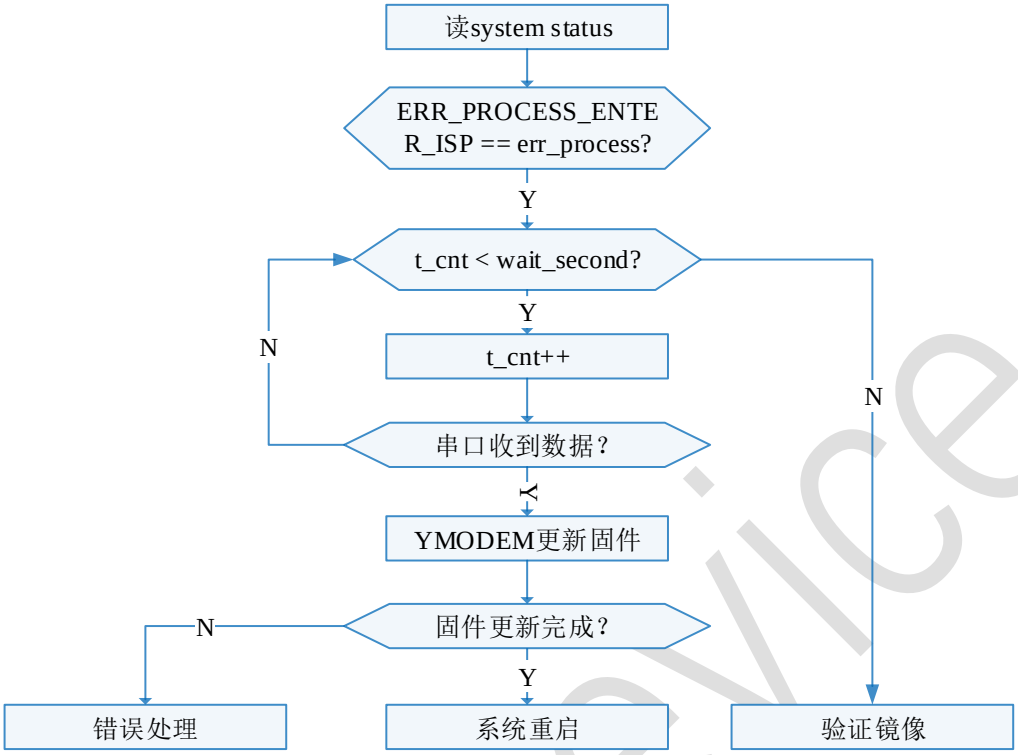
4.3 IBL 更新 MBL

在 IBL 中提供一个通过串口实现的 YMODEM 协议来对 MBL 的固件进行更新的功能，其实现流程如*图 4.3. IBL 更新 MBL* 所示。

1. 首先会判断系统状态中的错误处理状态是否为“ERR_PROCESS_ENTER_ISP”，如果是表明允许通过串口来更新MBL固件；
2. 在wait_second查找串口是否有数据输入，如果有则进行YMODEM协议升级固件，如果没有则进行验证镜像的步骤；
3. 如果固件更新成功系统会重启，如果更新失败系统会进入错误处理状态。

注意：
错误处理状态可以通过 IBL 的 API ibl_sys_status_set、ibl_sys_status_get 进行设置和获取。

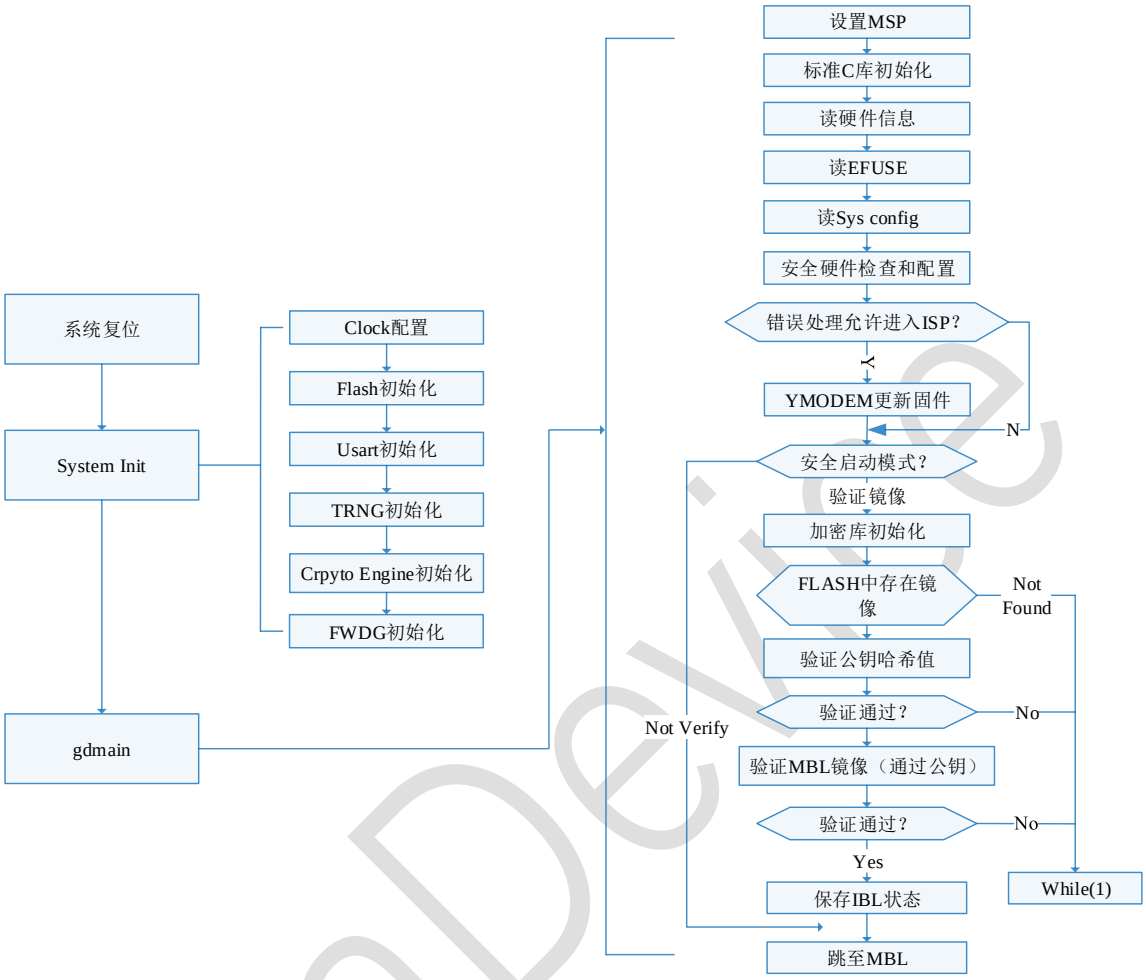
图 4.3. IBL 更新 MBL



4.4 IBL 的启动流程

IBL 的启动流程如*图 4.4. IBL 启动流程*所示，系统复位后，IBL 将配置相应的外设，用于后续的签名认证过程。之后会判断是否需要更新固件，如果需要更新完成固件更新并重启。之后会在一段时间延迟后开始验证固件如果验证通过，则跳转到 MBL，否则，它将进入无限循环并等待下一次重置。

图 4.4. IBL 启动流程



4.5 IBL 的导出库与 API 接口

为了提高代码的使用率，IBL 会将适配的 mbedtls 加解密库和部分 API 接口导出，这样在 MBL 和 APP 工程中就可以使用这些导出的功能库的功能。IBL 的导出库主要分为两个部分 mbedtls 的加解密库和 API 接口。Mbedtls 的导出库主要为常用的加解密接口，这里不作过多描述。IBL 导出的 API 接口可以参考*表 4.1. API 接口*。

表 4.1. API 接口

函数名称	描述
ibl_log_uart_init	串口初始化
ibl_printf	串口格式化打印
ibl_uart_putc	串口打印字符
ibl_trace_ex	串口追溯打印
ibl_rand	随机数获取
ibl_set_mutex_func	互斥量添加
ibl_memset	字符串赋值
ibl_memcmp	字符串拷贝

函数名称	描述
ibl_strlen	字符串长度计算
ibl_strncmp	字符串比较
ibl_uart_rx_to	串口接收处理
ibl_strtoul	字符串转十进制数
ibl_cal_checksum	校验和计算
ibl_img_verify_sign	签名验证
ibl_img_verify_hash	哈希值验证
ibl_img_verify_hdr	镜像头验证
ibl_img_verify_pkhash	公钥哈希验证
ibl_img_validate	镜像验证
ibl_sys_setting_get	获取系统配置
ibl_sys_status_set	设置系统状态
ibl_sys_status_get	获取系统状态
ibl_sys_set_trace_level	设置系统追溯级别
ibl_sys_set_err_process	设置系统错误处理
ibl_sys_set_img_flag	设置镜像标志（未启用）
ibl_sys_reset_img_flag	重置镜像标志（未启用）
ibl_sys_set_running_img	设置运行的镜像（未启用）
ibl_sys_set_fw_ver	设置固件的版本（未启用）
ibl_sys_set_pk_ver	设置公钥的版本
ibl_sys_set_trng_seed	设置随机数的种子
ibl_jump_to_img	跳转执行镜像
ibl_ymodem_download_check	Ymodem 更新检查
ibl_is_valid_flash_offset	FLASH 偏移合法性判断
ibl_is_valid_flash_addr	FLASH 地址合法性判断
ibl_flash_total_size	获取 FLASH 的总大小
ibl_flash_erase_size	获取 FLASH 的擦除大小
ibl_flash_init	FLASH 初始化
ibl_flash_read	读 FLASH
ibl_flash_write	写 FLASH
ibl_flash_write_fast	速写 FLASH（未启用）
ibl_flash_erase	FLASH 擦除
ibl_fmc_unlock	FLASH 解锁
ibl_fmc_lock	FLASH 上锁
ibl_fmc_flag_clear	FLASH 标志位清除
ibl_fmc_page_erase	FLASH 页擦除
ibl_fmc_mass_erase	FLASH 全片擦除
ibl_fmc_word_program	FLASH 字编程
ibl_fwdg_reload	FWDG 喂狗

4.6 IBL 状态信息

在 IBL 完成启动流程并跳转至 MBL 之前会有部分状态信息通过 SRAM 递交给 MBL，这些状态信息称为初始引导状态，在 MBL 中可以使用初始引导状态信息获取需要的启动信息如 MBL 公钥等。*表 4.2. 初始引导状态信息*显示了初始引导状态信息的内容。

表 4.2. 初始引导状态信息

信息名称	描述
reset_flag	复位标志
boot_status	引导的状态
ibl_ver	IBL 的版本
ibl_opt	IBL 的配置信息
impl_id	实施的 ID（未启用）
mbl_pk	MBL 的公钥
mbl_info	MBL 的信息（未启用）

5 MBL 工程

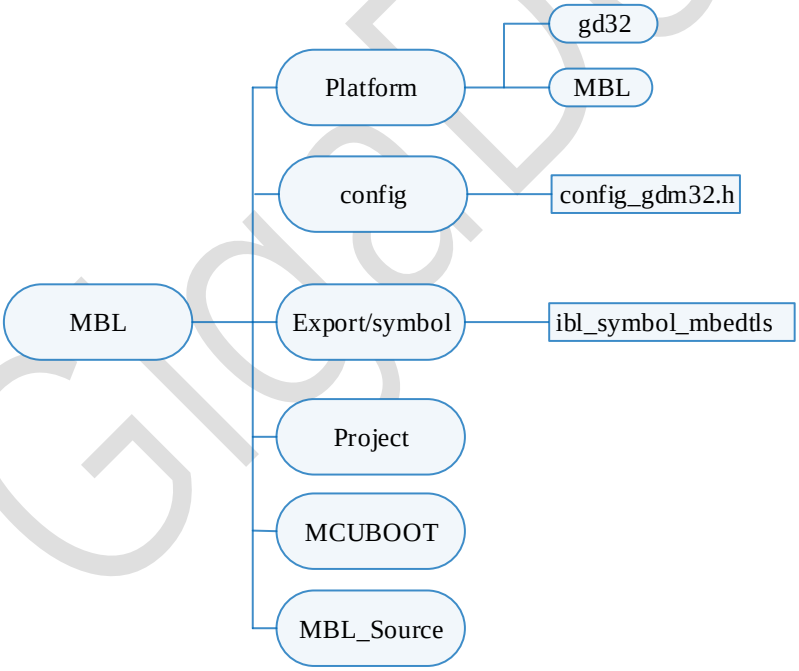
MBL 工程是 SBSFU 工程中第二阶段的引导代码，MBL 本身由 IBL 完成验签和加载，其主要功能是完成对应用代码的合理引导和加载。为了满足对应用代码的引导和加载功能，MBL 适配了 MCUBOOT，目前支持和验证的功能如下：

- 目前MBL支持单镜像双插槽的APP镜像管理，插槽0（主插槽）为APP代码的执行区，插槽1（副插槽、第二插槽（second slot）或者临时插槽（scratch slot））为APP更新区域；
- 支持串口实现YMODEM协议来更新APP代码至更新区域；
- 支持更新和执行APP代码之前对APP代码完整性和真实性的验证；
- 支持双插槽APP代码的交换和回退功能，交换期间支持掉电恢复的功能；
- 支持APP代码的版本管理。

5.1 MBL 工程代码文件结

MBL 工程代码的文件结构如*图 5.1. MBL 工程代码文件结构*所示。其中主要包含了 MCUBOOT 代码、IBL API 和 IBL mbedtls 的导出库来实现对 APP 固件的更新、验签和执行的管理。

图 5.1. MBL 工程代码文件结构



5.2 APP 镜像的交换类型

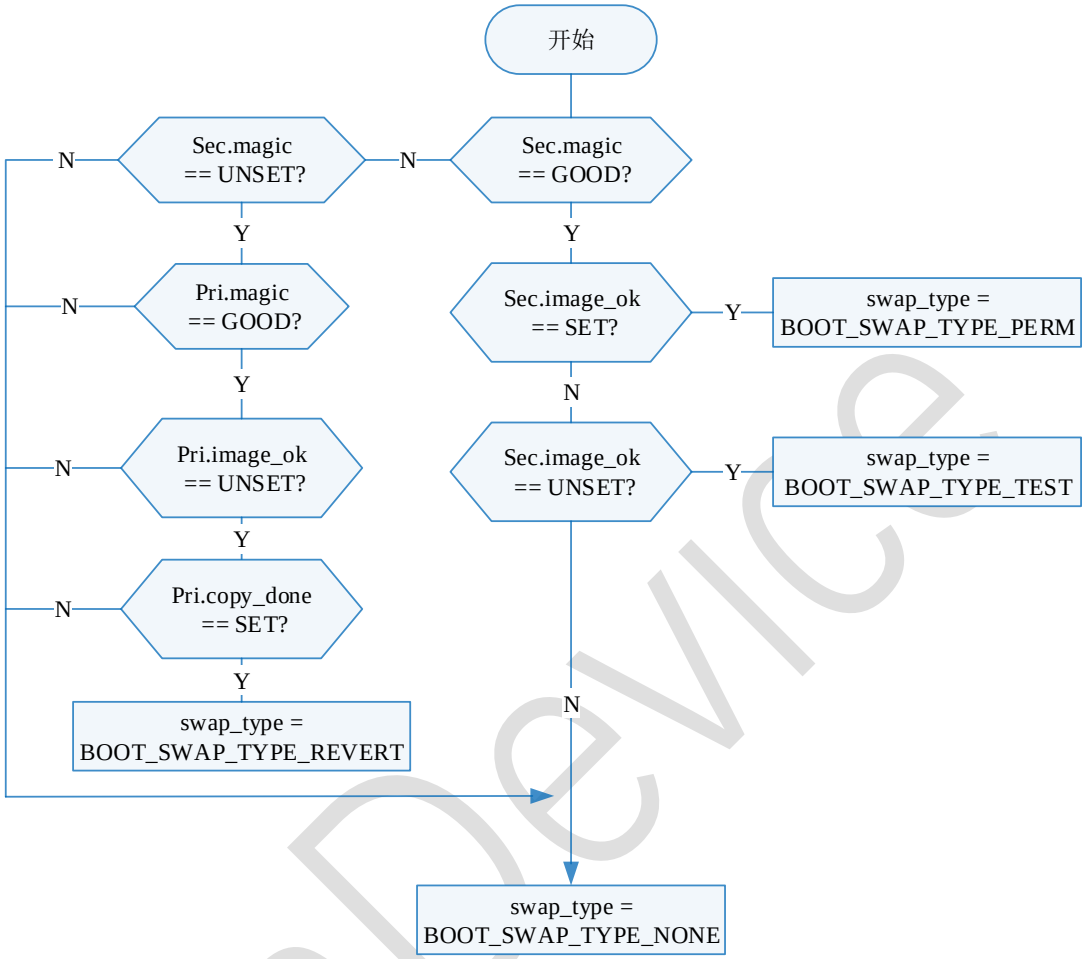
MBL 在进行 APP 固件的交换之前，会先根据 APP 插槽中的尾部（trailer）信息，来判断固件交换的类型。trailer 信息的内容可以参考 FLASH 的布局。

表 5.1. trailer 信息与交换类型

信息名称	取值	描述
交换类型（swap type）	BOOT_SWAP_TYPE_NONE	尝试启动主插槽的内容
	BOOT_SWAP_TYPE_TEST	交换到副槽，如果未确认，下次启动时恢复。
	BOOT_SWAP_TYPE_PERM	永久交换到副槽
	BOOT_SWAP_TYPE_REVERT	换回副槽位。确认后将此状态更改为 NONE
	BOOT_SWAP_TYPE_FAIL	交换失败
	BOOT_SWAP_TYPE_PANIC	交换遇到一个不可恢复的错误
拷贝完成标志（copy done）	BOOT_FLAG_SET	flag 已设置
	BOOT_FLAG_BAD	flag 非法
	BOOT_FLAG_UNSET	flag 未设置
	BOOT_FLAG_ANY	flag 无关（仅用作控制）
镜像有效标志（image ok）	BOOT_FLAG_SET	flag 已设置
	BOOT_FLAG_BAD	flag 非法
	BOOT_FLAG_UNSET	flag 未设置
	BOOT_FLAG_ANY	flag 无关（仅用作控制）
魔数（magic）	BOOT_MAGIC_GOOD	magic 正常
	BOOT_MAGIC_BAD	magic 非法
	BOOT_MAGIC_UNSET	magic 未设置
	BOOT_MAGIC_ANY	magic 无关（仅用作控制）
	BOOT_MAGIC_NOTGOOD	magic 不正常（仅用作控制）

trailer 信息与交换类型的取值的描述可以参考*表 5.1. trailer 信息与交换类型*。*图 5.2. 交换类型的确认*显示了交换类型的确认过程。其中“BOOT_SWAP_TYPE_TEST”交换类型，如果在交换完成后不进行确认（设置交换后的槽 image ok 信息为 BOOT_FLAG_SET），在下次重启后会进行“BOOT_SWAP_TYPE_REVERT”交换，交换回副槽中的固件。

图 5.2. 交换类型的确认



5.3 trailer 信息的变更

不同的交换类型会导致插槽的 trailer 信息产生不同的变更，基本的交换类型和 trailer 信息的变更可以参考*图 5.3. trailer 信息的变更*。其中“BOOT_SWAP_TYPE_NONE”交换不会导致 trailer 信息的变更，在“BOOT_SWAP_TYPE_TEST”交换后可以手动将 image ok 置位，这样下次复位后就不会产生“BOOT_SWAP_TYPE_REVERT”交换。“BOOT_SWAP_TYPE_PERM”交换是永久性的不会再进行回退交换。

图 5.3. trailer 信息的变更

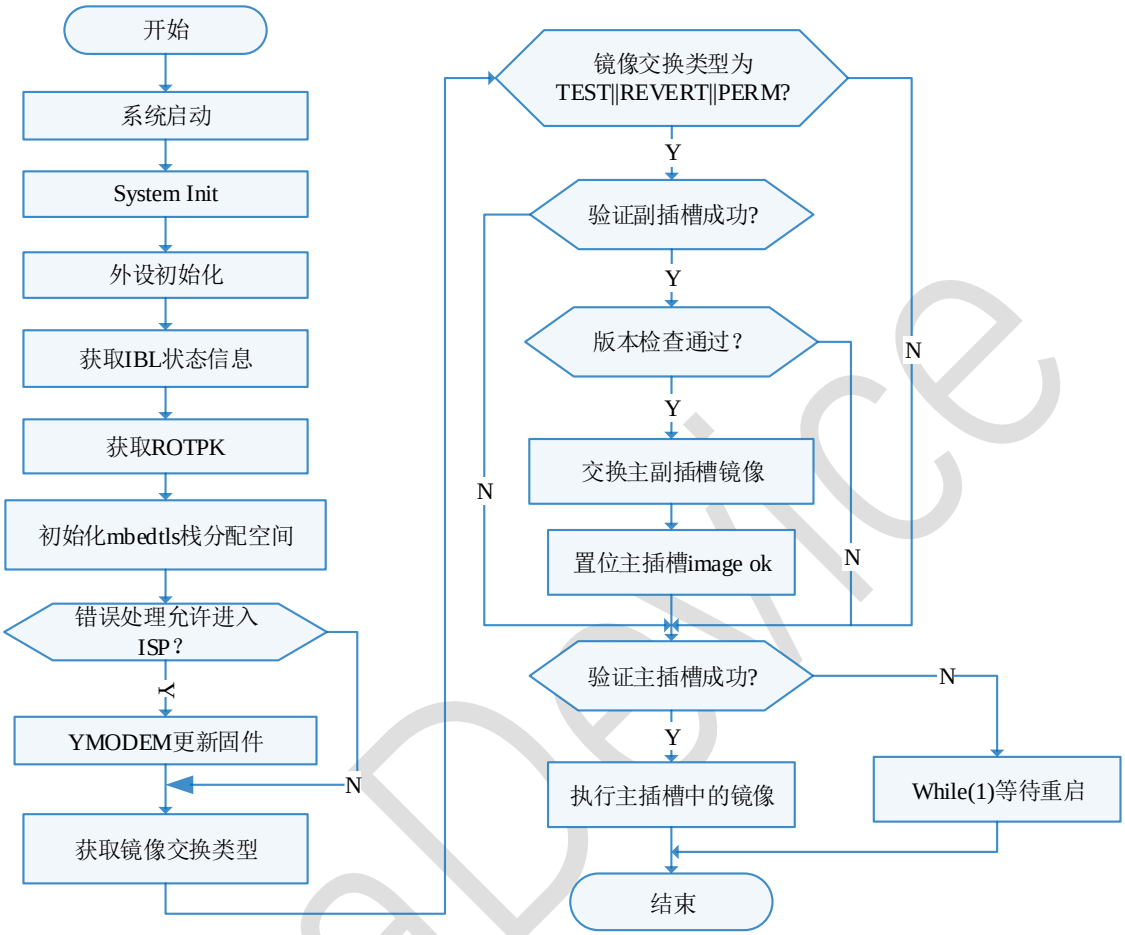
5.4 MBL 的启动流程

MBL 整体的启动流程如*图 5.4. MBL 的启动流程*所示。

1. 首先系统启动之后由于不再使用内部晶振，所以会重新初始化系统时钟，以及部分外设（如串口）；
2. 之后会获取IBL的状态信息，并从IBL状态信息中获取用于固件验签的ROTPK;
3. 然后会为mbedtls分配堆栈空间，在完成该工作后从IBL导出的mbedtls库就能够在MBL工程中使用了；
4. 之后会判断是否需要更新固件，如果需要更新完成固件更新并重启，固件更新的流程与*IBL 更新MBL*类似，不同的是MBL更新的固件是APP固件，并且会将固件更新至副插槽中；
5. 接下来会从主副插槽的尾部信息中获取并判断镜像的交换类型；
6. 如果交换类型为需要进行镜像交换则运行步骤7，否则执行步骤9；
7. 验证副插槽并，并检查版本信息，如果验证不通过执行步骤9；
8. 副插槽有效，交换主副插槽镜像，并置位主插槽镜像有效标志位；

9. 验证主插槽中的镜像，如果验证通过，尝试执行主插槽，验证不通过会循环，并等待系统重启。

图 5.4. MBL 的启动流程



5.5 MBL 的导出库

如前文所述，在完成进行交换，以及固件更新后，APP 代码会设置尾部信息的魔数和镜像有效标志位，为此 MBL 会导出部分接口函数供 APP 调用。MBL 导出的库接口可以参考*表 5.2. MBL 导出库接口*。

表 5.2. MBL 导出库接口

函数名称	描述
flash_area_open	打开 FLASH 区域
boot_write_magic	写魔数
boot_write_image_ok	写镜像有效标志

6 APP 工程

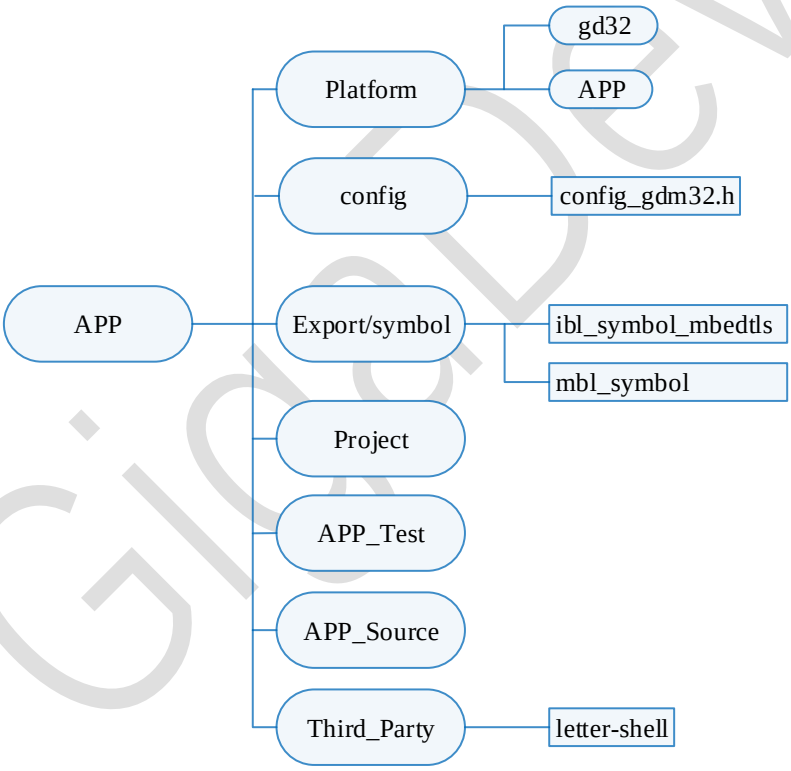
APP工程是SBSFU工程中最终完成引导并执行的代码，此时的代码已经通过了完整性和真实性验证，可以认为是可信的固件，目前APP的实现较为简单，主要实现的功能如下：

- 集成了letter-shell作为串口中断工具；
- 支持APP主插槽更新固件至副插槽；
- 支持IBL导出的API和导出mbedtls库；
- 支持部分测试功能。

6.1 APP 工程代码文件结

APP工程代码的文件结构如*图6.1. APP工程代码文件结构*所示。其中主要包含了letter-shell代码、测试代码、IBL API和IBL mbedtls的导出库来实现对APP固件的更新。

图 6.1. APP 工程代码文件结构



6.2 APP shell 命令

在串口终端中可以通过help命令列出目前支持的命令，*图6.2. APP shell命令*显示了目前全部支持的命令。

图 6.2. APP shell 命令

```
COMMAND LIST:

help                    --command help
cls                     --clear command line
reboot                  --reboot system
ymodem_update           --ymodem update image
app_test                --run app testsuit

#>>
```

Shell命令的介绍可以参考*表6.1. Shell命令与描述*。

表 6.1. Shell 命令与描述

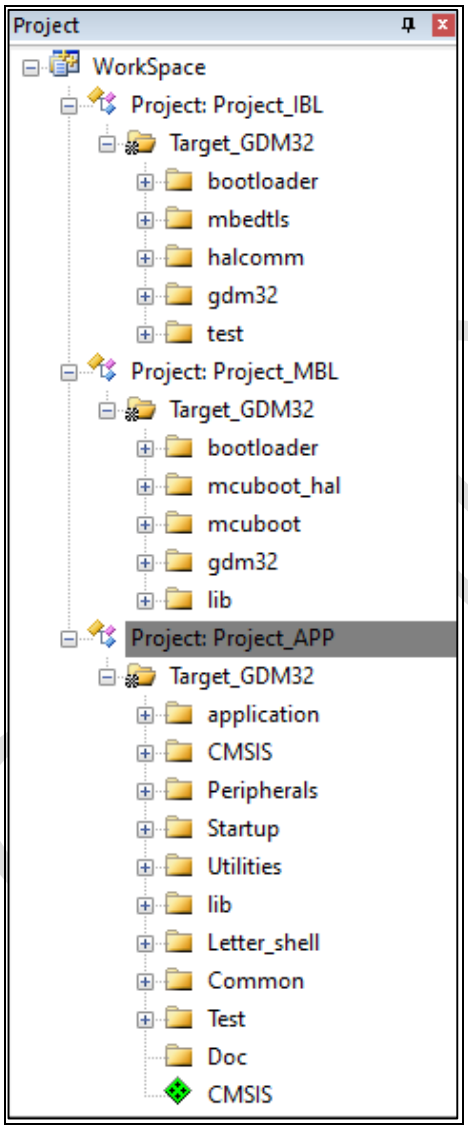
命令名称	描述
help	帮助命令，能够列出所有支持的命令
cls	清屏命令，清除终端屏幕
reboot	软件复位重启
ymodem_update	Ymodem 协议下载 APP 固件到副插槽
app_test	运行测试代码（目前仅用于验证 IBL 的 mbedtls 导出库）

7 SBSFU 工程在 GD32F5xx 上的实施

7.1 工程编译

GD32F5xx 的 SBSFU 工程使用 keil IDE 进行管理，可以通过“Project/GD32F5xx/MDK-ARM/SBSFU.uvmpw”打开SBSFU工程组，工程组打开后的工程结构如*图7.1. SBSFU工程组*所示。其中总共包含了三个独立的工程IBL、MBL和APP工程。使用图中所示的“Batch Build”按钮能够一次性编译所有的工程，在编译完成后会在“scripts/images”目录下生成可用于下载到芯片中的镜像。

图 7.1. **SBSFU 工程组**



生成的镜像可以参考*表7.1. 生成的镜像*。其中mbl-sys.hex会复制到“MBL/mbl/Objects/mbl.hex”，之后可以直接使用MBL工程下载。app-sign.hex和app-sign1.hex文件会分别复制到“APP/app/Objects/app.hex”和“APP/app/Objects/app1.hex”，之后可以使用APP工程下载该文件。

注意：

"mbl-sys.hex"是包含了系统配置信息的，在首次下载时必须下载包含配置信息的文件。

表 7.1. 生成的镜像

镜像名称	描述	用途
ibl.bin	IBL 镜像 bin 格式文件	可直接用于下载
ibl.hex	IBL 镜像 hex 格式文件	可直接用于下载
mbl.bin	原始的 MBL 镜像 bin 格式文件	临时文件
mbl-sign.bin	添加镜像头和签名信息的 MBL 镜像 bin 格式文件	可用于 MBL 更新
mbl-sys.bin	添加镜像头、签名信息和系统设计信息的 MBL 镜像 bin 格式文件	首次下载文件
mbl-sys.hex	添加镜像头、签名信息和系统设计信息的 MBL 镜像 hex 格式文件	首次下载文件
app.bin	原始的 APP 镜像 bin 格式文件	临时文件
app-sign.bin	添加镜像头和签名信息的 APP 镜像 bin 格式文件	可用于 APP 更新
app-sign.hex	添加镜像头和签名信息的 APP 镜像 hex 格式文件 1	可直接用于下载
app-sign1.hex	添加镜像头和签名信息的 APP 镜像 hex 格式文件 2	可直接用于下载

7.2 工程下载和验证

- 下载IBL工程

IBL代码是最开始执行的代码，不需要做特殊处理，所以和普通工程一样下载和调试即可。

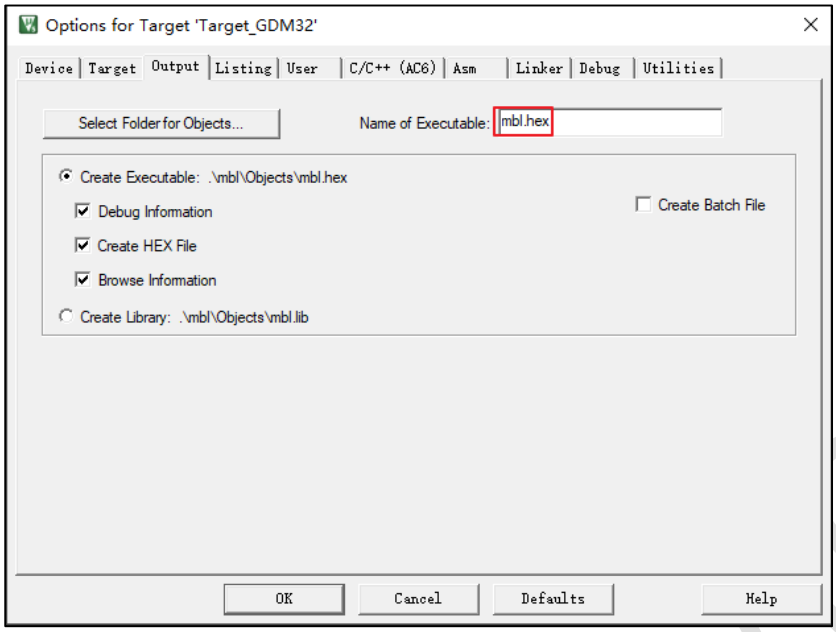
- 下载MBL工程

MBL工程编译后会通过工具程序，在MBL可执行代码的基础上添加包含固件头、固件签名和系统设置信息，然后在"scripts/images"目录下会生成"mbl-sys.hex"，同时会使用"mbl-sys.hex"替换原工程中的"mbl.hex"。之后这两个文件都可用于下载，这里可以直接使用keil下载"mbl.hex"文件到芯片中。参考*图7.2. MBL工程下载*修改可执行文件名为"mbl.hex"，不需要编译工程直接下载即可。

注意：

- 安照上述方式下载代码，如果想重新编译需要，需要先重新修改可执行文件名为"mbl"；
- "mbl-sys.hex"镜像可用于初次下载，更新的固件应当使用"mbl-sign.bin"。

图 7.2. MBL 工程下载



- 下载APP工程

APP工程的下载可以参考MBL工程，其中app.hex可以将镜像下载到主插槽，app1.hex可以将镜像下载到副插槽。

- SBSFU工程启动验证

在所有工程都正确下载完成后，如果串口输出如*图7.3. SBSFU工程串口输出*所示，则表明工程正确运行。

注意：

安全启动工程默认使用的是USART0，波特率为115200。

图 7.3. SBSFU 工程串口输出

```
ALW: GIGA DEVICE
INF: Get ROTPK form flash.
ALW: IBL version 1.1
ALW: Reset by pin.
INF: Sys status checked OK.
ALW: IBL wait for ymodem download OS
INF: MBL version: 1.0.0, Local: 1.0.0
ALW: Validate MBL Image OK.
INF: local version: 1.0.0
ALW: Update MBL version to 1.0.0
ALW: Jump to MBL.
ALW: MBL version is 1.0.0
ALW: MBL wait for ymodem download OS
INF: Primary image: magic=good, swap_type=0x2, copy_done=0x1, image_ok=0x1
INF: Scratch: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
INF: Boot source: none
INF: Image index: 0, Swap type: none
INF: validate primary solt succeed.
ALW: Jump to APP
ALW: APP image version is 1.0.0
DBG: writing image_ok; fa_id=0 off=0xx (0xx)


+=================================================+
|                  Letter shell v2.0.8            |
|                   SBSFU for GD32MCU             |
|               Build: Jun  6 2024 17:40:13       |
+=================================================+

#>>
```

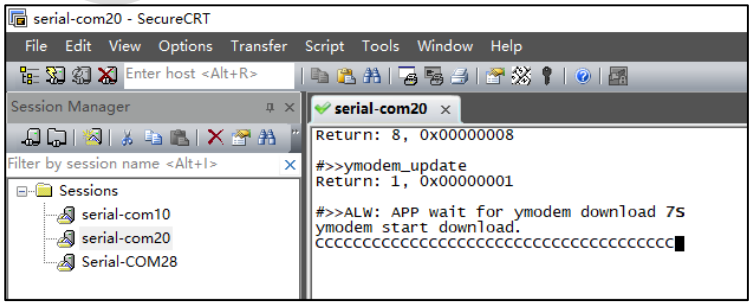
7.3 固件的更新

固件更新的实施可以参考*固件更新的实施*，本节将主要介绍如何更新MBL和APP固件，终端软件使用的SecureCRT7.3.3。

7.3.1 APP 固件的更新

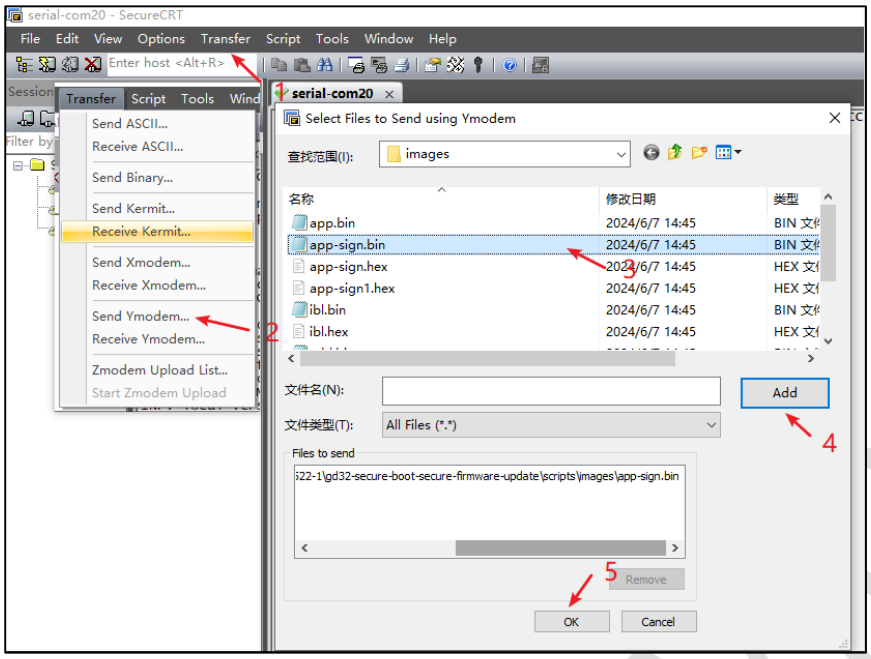
APP固件的更新可以遵循如下步骤，如*图7.4. 终端发送更新命令*所示，在终端中输入“ymodem_update”命令，之后设备会等待更新确认（用户可以在终端中任意输入一个字符，如按下空格键确认更新），在倒计时结束之前如果确认需要更新设备会一直输出'C'等待进行更新。

图 7.4. 终端发送更新命令



在设备进入等待更新后可以参考*图7.5. 终端发送更新固件*，从终端发送更新的固件给设备。在更新成功后设备会有由软件复位重启。

图 7.5. 终端发送更新固件



更新过程和更新成功后设备重启的日志打印可以参考*图7.6. 终端更新日志打印*。从日志中能够发现在重启之后会由MBL程序进行测试交换，将副插槽中的镜像交换到主插槽中，完成APP固件的更新。

图 7.6. 终端更新日志打印

```
Starting ymodem transfer.  Press Ctrl+C to cancel.
Transferring app-sign.bin...
  100%       11 KB       11 KB/sec    00:00:01       0 Errors


DBG: writing magic; fa_id=1 off=0xx (0xx)
ALW: Ymodem update success, system reboot.
ALW: GIGA DEVICE
INF: Get ROTPK form flash.
ALW: IBL version 1.1
ALW: Reset by sw.
INF: Sys status checked OK.
ALW: IBL wait for ymodem download OS
INF: MBL version: 1.0.0, Local: 1.0.0
ALW: Validate MBL Image OK.
INF: local version: 1.0.0
ALW: Update MBL version to 1.0.0
ALW: Jump to MBL.
ALW: MBL version is 1.0.0
ALW: MBL wait for ymodem download OS
INF: Primary image: magic=good, swap_type=0x2, copy_done=0x1, image_ok=0x1
INF: Scratch: magic=unset, swap_type=0x1, copy_done=0x3, image_ok=0x3
INF: Boot source: none
INF: Image index: 0, Swap type: test
INF: Starting swap using scratch algorithm.
DBG: erasing scratch area
DBG: initializing status; fa_id=2
DBG: writing swap_info; fa_id=2 off=0xx (0xx), swap_type=0x1fd8 image_num=0x83fd8
DBG: writing swap_size; fa_id=2 off=0xx (0xx)
DBG: writing magic; fa_id=2 off=0xx (0xx)
DBG: erasing trailer; fa_id=0
DBG: initializing status; fa_id=0
DBG: writing swap_info; fa_id=0 off=0xx (0xx), swap_type=0xfffd8 image_num=0x19ffd8
DBG: writing swap_size; fa_id=0 off=0xx (0xx)
DBG: writing magic; fa_id=0 off=0xx (0xx)
DBG: erasing trailer; fa_id=1
DBG: erasing scratch area
DBG: writing copy_done; fa_id=0 off=0xx (0xx)
INF: validate primary solt succeed.
ALW: Jump to APP
ALW: APP image version is 1.0.0
DBG: writing image_ok; fa_id=0 off=0xx (0xx)


+=============================================+
|                  Letter shell v2.0.8        |
|                   SBSFU for GD32MCU         |
|               Build: Jun  7 2024 14:45:22   |
+=============================================+

#>>
```

7.3.2 MBL 固件的更新

MBL固件可以通过IBL进行更新，在SBSFU工程启动的第一个倒计时如*图7.7. MBL的更新*所示，可以进行更新确认（确认过程与APP更新类似）。之后可以通过终端软件将更新固件发送给设备完成固件更新，成功更新后设备同样会复位重启。

图 7.7. MBL 的更新

```
ALW: GIGA DEVICE
INF: Get ROTPK form flash.
ALW: IBL version 1.1
ALW: Reset by pin.
INF: Sys status checked OK.
ALW: IBL wait for ymodem download 2S
```

7.3.3 MBL 更新 APP

APP固件也可以通过MBL进行更新，在SBSFU工程启动的第二个倒计时可以进行更新确认（确认过程与APP更新类似），并完成固件更新。

7.4 FLASH 布局的实施

SBSFU工程的整体FLASH布局可以参考*图3.3. FLASH布局*，其中IBL、MBL和APP的FLASH布局分别可以通过“Platform/IBL/ibl_region.h”、“Platform/MBL/mbl_region.h”和“Platform/APP/app_region.h”文件进行配置，其中主要使用到的配置宏可以参考*表7.2. FLASH布局配置宏*。

表 7.2. FLASH 布局配置宏

命令名称	描述	默认值
IBL		
FLASH_BASE_IBL	IBL 的基地址	0x08000000
IBL_CODE_START	IBL 代码的起始地址	0x08000000
IBL_CODE_SIZE	IBL 代码的大小	0x4000
FLASH_BASE_LIB	IBL 库的基地址	0x08004000
FLASH_LIB_START	IBL 库的起始地址	0x08004000
FLASH_LIB_SIZE	IBL 库的大小	0x50000
FLASH_OFFSET_SYS_SETTING	系统设置数据的偏移地址	0x60000
MBL		
MBL_BASE_ADDRESS	MBL 的基地址	0x08061200
MBL_CODE_START	MBL 代码的起始地址	0x08061200
MBL_CODE_SIZE	MBL 代码的大小	0x1E00
APP		
APP_CODE_START	APP 代码的起始地址	0x080A0200
APP_CODE_SIZE	APP 代码的大小	0xFFE00

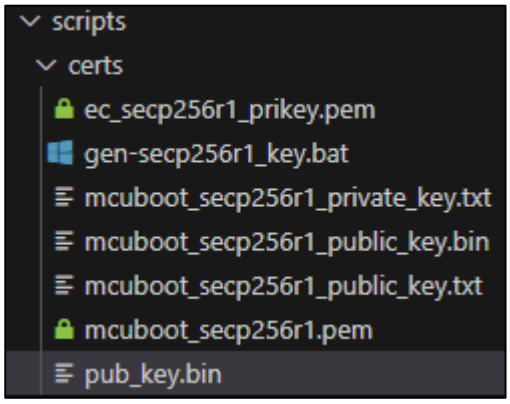
7.5 SBSFU 工程非对称密钥配置

7.5.1 新密钥的生成

目前SBSFU工程使用的是SECP256R1进行签名和验签，在“scripts/certs”目录下提供了一个“gen-

secp256r1_key.bat”，可以用于生成SECP256R1密钥对，双击BAT脚本后会生成如*图7.8. 生成SECP256R1密钥对*。

图 7.8. 生成 SECP256R1 密钥对



“scripts/certs”目录下文件和功能可以参考*表7.3. 密钥和证书文件描述*。

表 7.3. 密钥和证书文件描述

文件名	描述
ec_secp256r1_prikey.pem	默认的 secp256r1 私钥，APP 和 MBL 镜像生成的脚本会使用该文件。
pub_key.bin	默认的 secp256r1 公钥 bin 格式保存的文件。
gen-secp256r1_key.bat	secp256r1 密钥生成脚本，运行后会生成下面的三个文件。
mcuboot_secp256r1_public_key.bin	gen-secp256r1_key.bat 生成的 16 进制存储的公钥，可用于下载到 OTP2 区域。
mcuboot_secp256r1.pem	gen-secp256r1_key.bat 生成的 PEM 格式的私钥，可以用于替换 ec_secp256r1_prikey.pem。
mcuboot_secp256r1_private_key.txt	gen-secp256r1_key.bat 生成的 16 进制存储的私钥，可用于签名
mcuboot_secp256r1_public_key.txt	gen-secp256r1_key.bat 生成的 16 进制存储的公钥，可用于验签。

7.5.2 新密钥的适配

如果需要适配新的密钥需要对如下2个地方做修改：

- MBL和APP生成镜像脚本的私钥适配

生成镜像脚本的私钥适配，以MBL为例需要适配如*图7.9. 适配私钥*所示的位置。APP生成镜像脚本的私钥适配和MBL类似。

图 7.9. 适配私钥

```
afterbuild.py
Project > afterbuild.py > sysset_gen
20  from scripts.scripts_mcuboot.hextool import convert_to_hex
21  from scripts.scripts_mcuboot.imgtool.main import sign
22
23  ROTPK      = ROOT + "/scripts/certs/ec_secp256r1_prikey.pem"
24  CONFIG_FILE = ROOT + "/config/config_gdm32.h"
25  IMGTOOL    = ROOT + "/scripts/scripts_mcuboot/imgtool.py"
26  SREC_CAT   = ROOT + "/scripts/scripts_mcuboot/srec_cat.exe"
```

- IBL工程中的公钥适配

公钥适配时如果是使用存储在FLASH中的公钥（默认情况下是存储在FLASH中），需要将公钥的16进制数据写入到如*图7.10. 适配公钥*所示的位置。

图 7.10. 适配公钥

```
ibl_key.c 2
Source > IBL_Source > ibl_key.c > ...
22  };
23  #elif SIGN_ALGO_SET == IMG_SIG_ECDSA256
24  static const uint8_t flash_rotpk[IMG_PK_LEN] = {
25      0xce, 0x43, 0xb2, 0xfc, 0x3a, 0x04, 0x15, 0x20,
26      0x82, 0xe7, 0xc5, 0x93, 0xa1, 0x3d, 0x93, 0xb2,
27      0x82, 0x55, 0xc2, 0x19, 0x97, 0x13, 0x32, 0x05,
28      0x5a, 0x01, 0x6f, 0x07, 0x72, 0x47, 0x46, 0xbd,
29      0x77, 0xb7, 0x66, 0xda, 0x7c, 0xe4, 0x07, 0x2d,
30      0xec, 0xa0, 0x23, 0x8e, 0x88, 0x49, 0x35, 0x37,
31      0xc4, 0x62, 0xe7, 0x7b, 0x32, 0x2c, 0xfe, 0x04,
32      0xe7, 0xb2, 0x4c, 0x54, 0xb5, 0xb9, 0xbb, 0x80
33  };
34  #endif /* SIGN_ALGO_SET */
35  #endif /* GD32F5XX_OTP2_ROTPK */
```

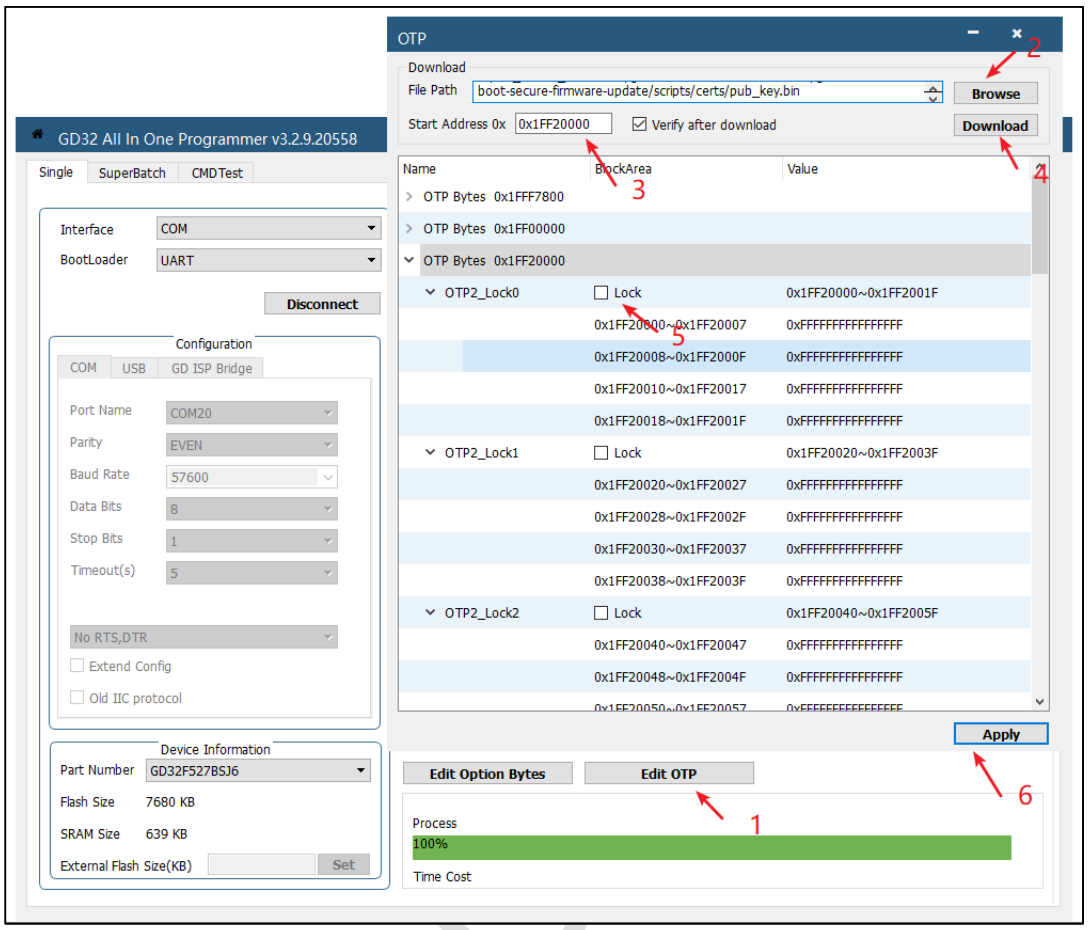
7.6 适配公钥到 OTP2 区域

GD32F5xx的OTP2区域支持读写锁定配置，非常适合用于存储安全性要求较高的公钥，想要将公钥写入OTP2区域并正常使用需要做如下操作和修改：

- 将公钥写入OTP2区域，并设置合适的读写锁定

将公钥写入OTP2区域，可以使用“GD32AllInOneProgrammer.exe”软件，写入OTP2区域可以参考*图7.11. 将公钥写入OTP2区域*。

图 7.11. 将公钥写入 OTP2 区域



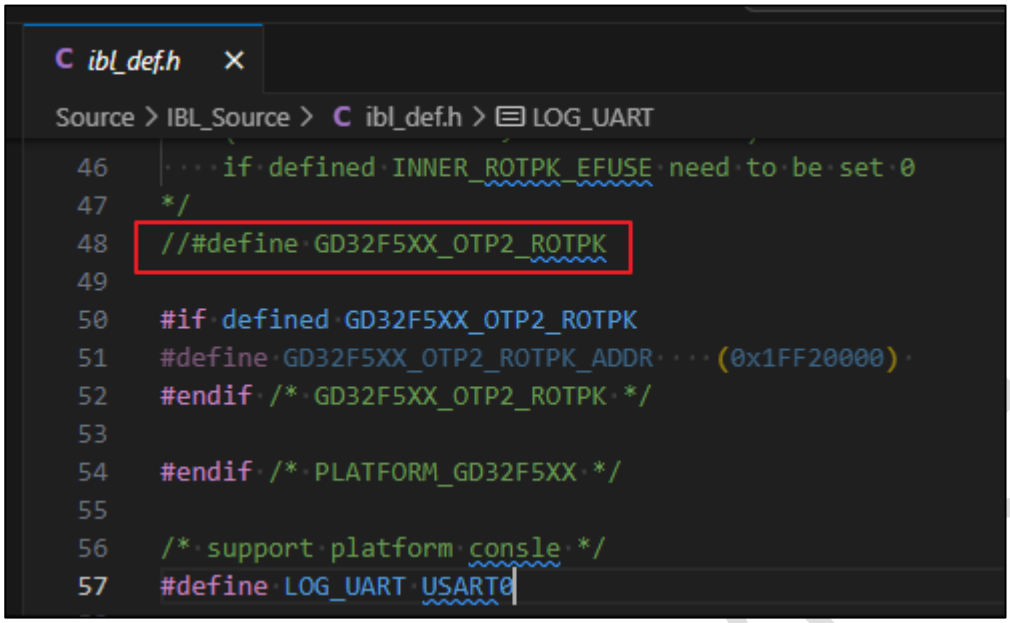
注意：

- 使用GD32AllInOneProgrammer.exe软件需要从BootLoader区域启动；
- 目前代码中默认使用的是OTP2的前64字节存储公钥；
- OTP区域仅支持一次编程，锁定块字节仅可从 0xFF 到 0x00 编程一次。

- IBL工程中的公钥适配。

在IBL工程中需要使能如*图7.12. IBL工程使用OTP2存储公钥*所示的宏“GD32F5XX_OTP2_ROTPK”。OTP2区域在IBL读取到密钥后会由代码设置为读锁定，这样后面的代码将不能再读取OTP2区域，MBL想获取公钥仅能通过IBL的状态信息。

图 7.12. IBL 工程使用 OTP2 存储公钥

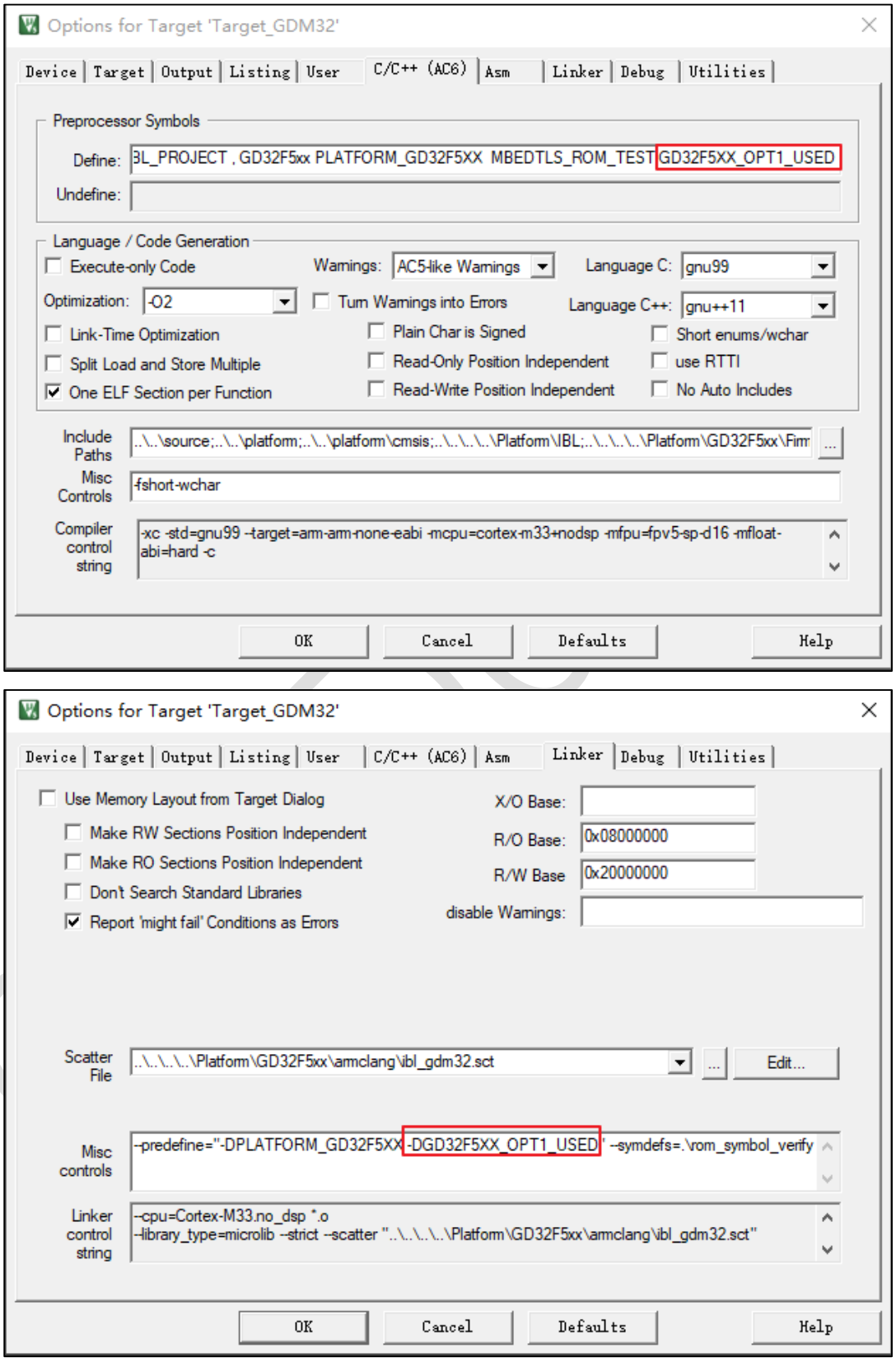


7.7 适配 IBL 到 OTP1 区域

SBSFU的IBL工程程默认是放置在FLASH中的，但在GD32F5xx芯片上存在一个128KB的OTP1区域支持读写锁定配置（关于OTP区域的描述可以参考《GD32F5xx_用户手册》），适合运行IBL工程，SBSFU工程也支持配置IBL到OTP1区域运行，使用时需要作如下修改：

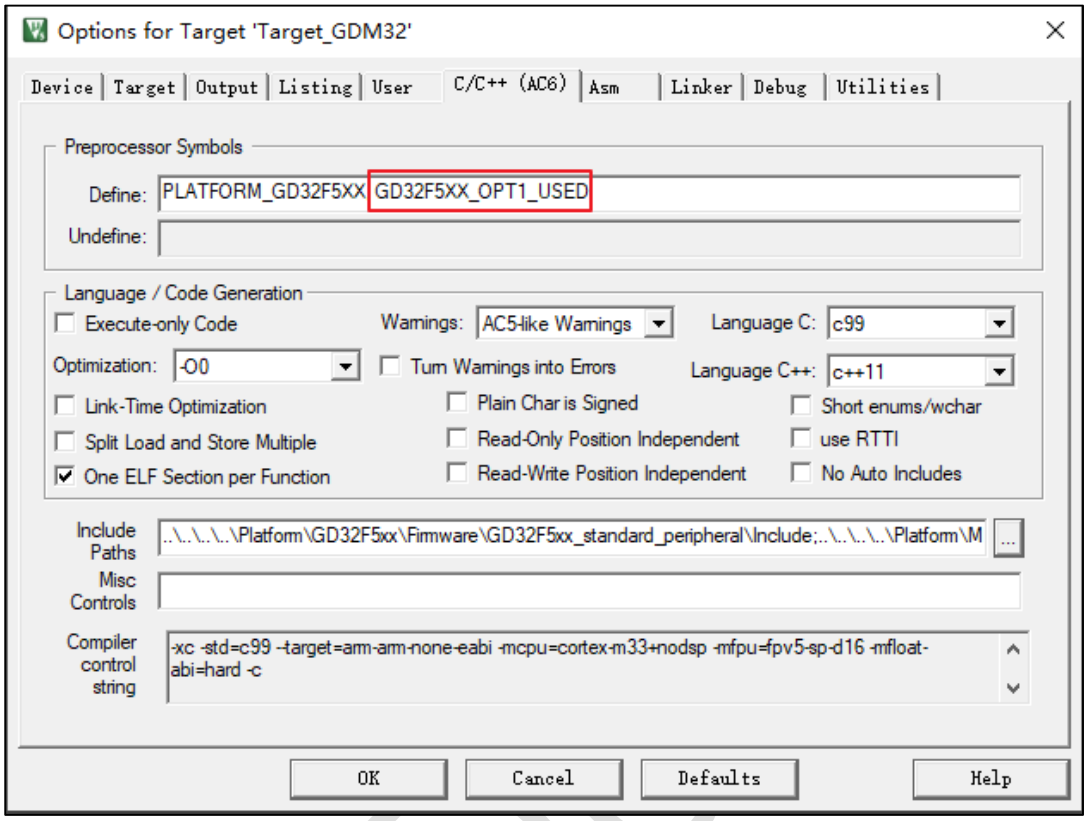
- 修改IBL工程
  - IBL工程中需要使能宏“GD32F5XX_OPT1_USED”，需要修改的地方有两处，可以参考*图 7.13. IBL适配到OTP1区域*。

图 7.13. IBL 适配到 OTP1 区域



- MBL和APP工程中也需要使能宏“GD32F5XX_OPT1_USED”，MBL需要修改的地方可以参考*图7.14. MBL修改*（APP工程的修改可以参考MBL工程）。

图 7.14. MBL 修改

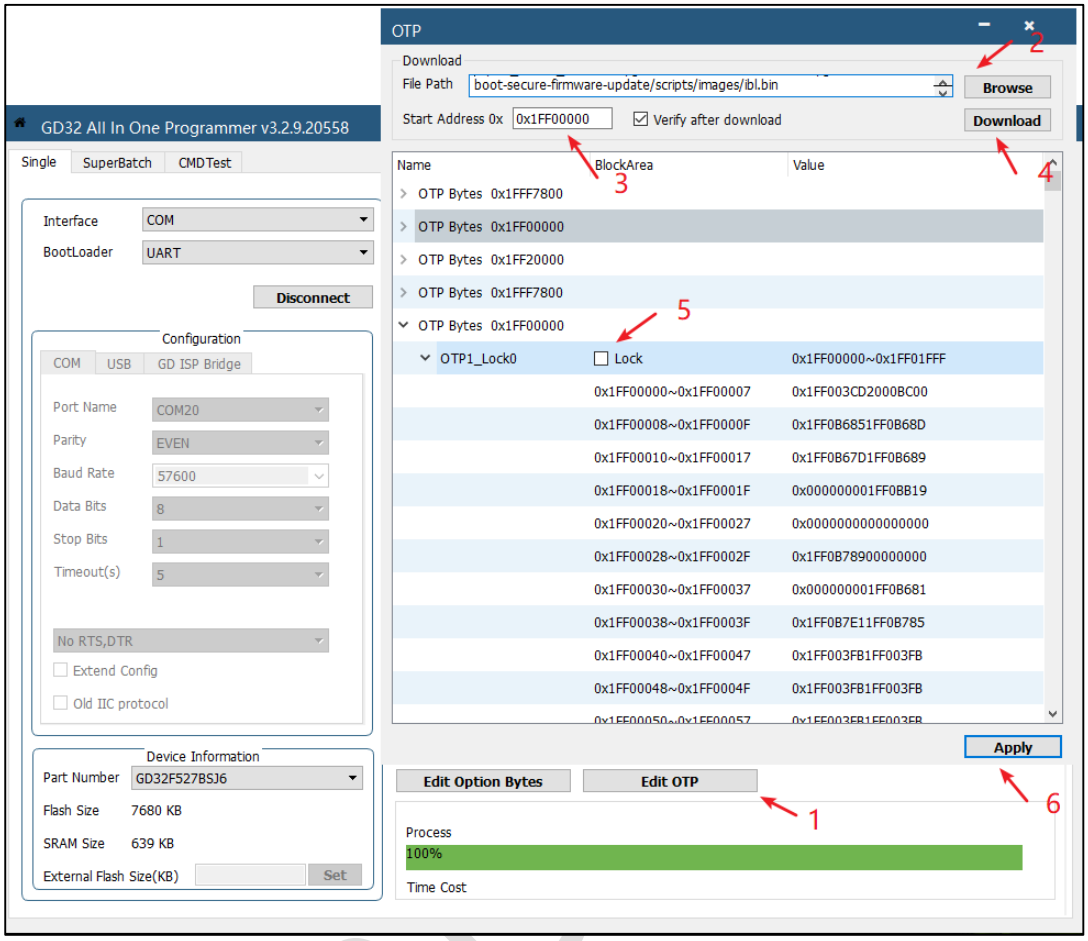


– 修改“config/config_gdm32.h”配置文件，需要将宏RE_FLASH_BASE修改为0x08000000。

■ 使用GD32AllInOneProgrammer.exe适配IBL到OTP1区域

在正确修改工程并编译成功后，会在“scripts/mages”目录下生成ibl.bin文件，ibl.bin文件就是IBL最终可执行文件。可以使用GD32AllInOneProgrammer.exe软件下载代码并配置OTP1为写锁定，具体步骤可以参考*图7.15. 下载IBL到OTP1区域*。

图 7.15. 下载 IBL 到 OTP1 区域



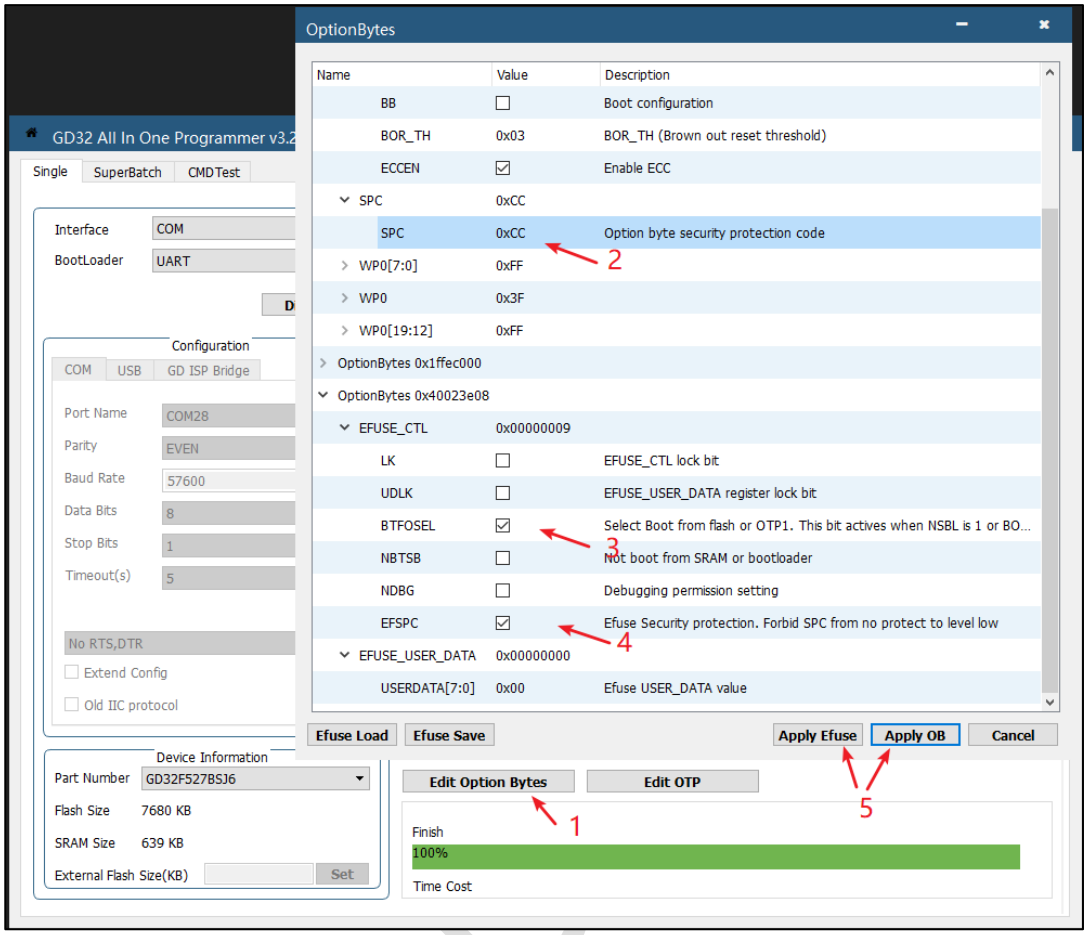
注意：

- 由于OTP1区域仅有128KB，所以IBL的部分功能可能会缺失；
- OTP1的读锁定暂未实施；
- 要想芯片从OTP1区域启动，需要设置BTFOSEL位，该位位于EFUSE中，一旦使能将无法回退。

7.8 启动入口的锁定

安全启动，需要确保唯一的启动入口，复位之后首先运行IBL的代码而不能运行其它代码，在GD32F5xx推荐配置芯片为高等级的安全保护和EFUSE配置来达到这样的目的，推荐使用GD32AllInOneProgrammer.exe软件进行配置，以配置芯片为高等级的安全保护为例，具体配置步骤可以参考*图7.16. 配置高等级的安全保护*。

图 7.16. 配置高等级的安全保护



注意：

- 一旦配置为高安全保护等级，将无法回退，同时无法使用调试器进行调试；
- 配置为高安全保护等级后选项字节将无法修改；
- 配置为高安全保护等级后将无法从RAM和BootLoader启动。

7.9 双 bank 交换功能

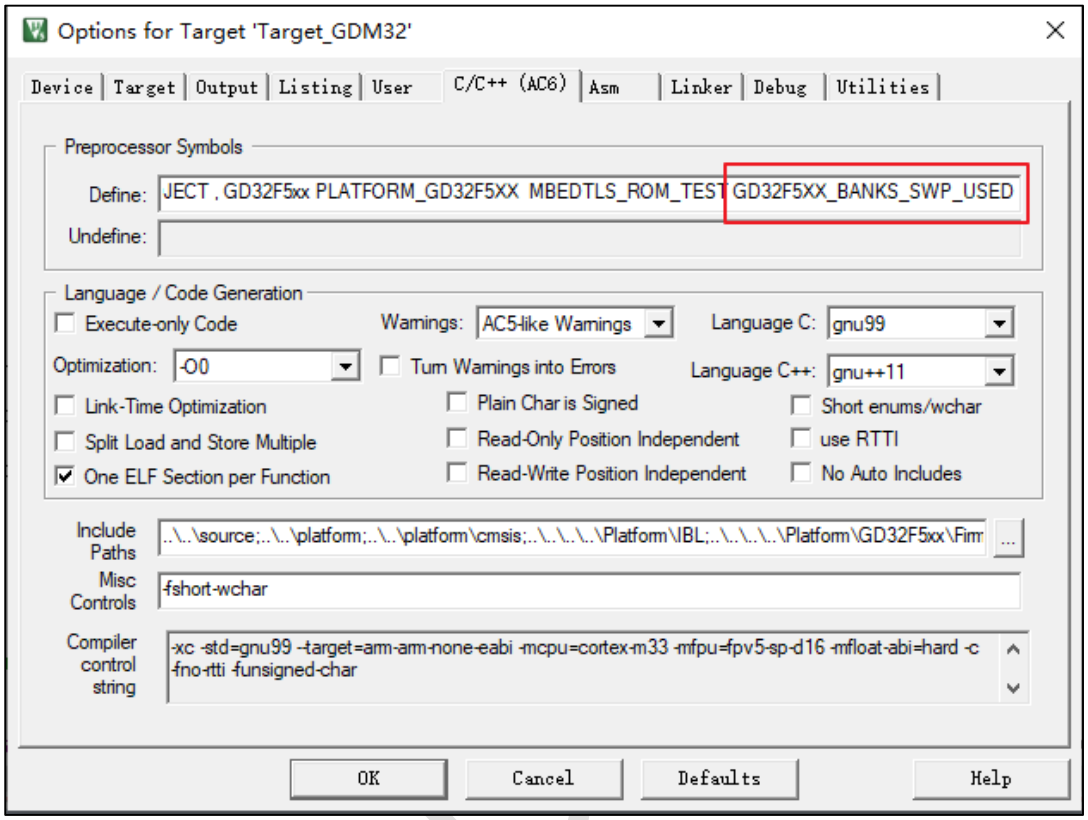
GD32F5xx部分芯片系列支持双块闪存的结构，并能够支持BANK0与BANK1区域的交换，可用于实现镜像的更新，关于双bank的详细内容可以参考《GD32F5xx_用户手册》。GD32 SBSFU工程针对GD32F5xx设备做了双bank交换的适配，用来实现APP镜像的更新（目前仅支持7M片上闪存芯片）。

7.9.1 使能双 bank

在SBSFU工程中提供了“config/config_gdm32_otp_bankswp.h”配置文件，在需要使用双bank功能时，可以使用“config/config_gdm32_otp_bankswp.h”文件中的内容替代“config/config_gdm32.h”配置文件中的内容。之后可以分别在IBL、MBL和APP工程中添加

“GD32F5XX_BANKS_SWP_USED”宏来使能双bank功能。IBL工程使能双bank功能可以参考*图7.17. IBL工程使能双bank功能*，MBL和APP的使能方式与IBL类似。

图 7.17. IBL 工程使能双 bank 功能



注意：

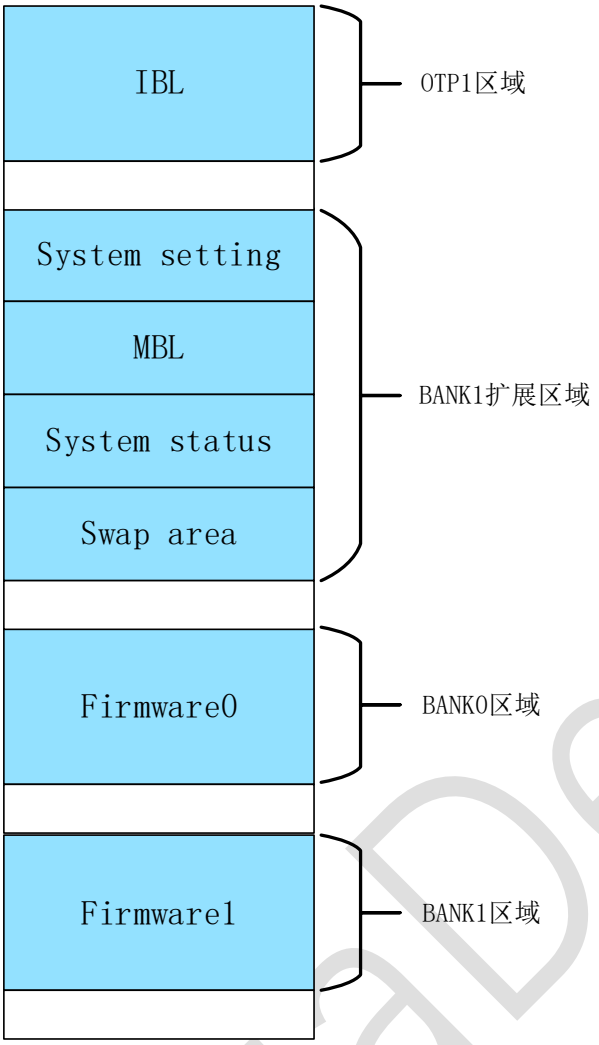
- “GD32F5XX_BANKS_SWP_USED”宏使能后，需要同时使能“GD32F5XX_OPT1_USED”。

7.9.2 使能双 bank 后的镜像分布

在使能双 bank 后镜像分布可以参考*图7.18.使能双 bank 后的镜像分布*。其中：

- IBL 将运行在 OTP1 区域中；
- 系统配置信息、系统状态信息、MBL 和交换区域存放在 bank1 的扩展区域（交换区域不再有效）；
- 运行固件（固件 0）和更新固件（固件 1）分别存放在 bank0 和 bank1 中。

图 7.18.使能双 bank 后的镜像分布



7.10 升级固件加密

加密的固件头会将 APP IMG 标记为 ENCRYPTED（0x04）。固件通过 AES 加密，在创建新 IMG 时，密钥由 imgtool 随机生成。此密钥不应重新使用，并且没有对此进行检查，但使用 TRNG 随机生成 16 字节块使重复发生的可能性极小。AES 密钥被非对称加密后存储在镜像的 TLVs 中，非对称加密的私钥存储在 MCU（公钥用来加密固件）中。固件的头和 TLVs 仍然以明文数据发送。哈希和签名的功能也与以前相同，应用于未加密的数据。

加密固件在更新时为边解密边更新，如*图 3.3. FLASH 布局*所示，当前正在运行的 IMG 存放在 Frimware0 处，是明文。加密固件会通过 ymodem 的方式下载到 Firmware1 处。固件更新前，会将 TLVs 中被加密的 AES 密钥通过非对称解密，使能升级固件加密后，Firmware0 中存放更新固件的明文，被更新的固件则通过 AES 加密存放在 Firmware1 中。

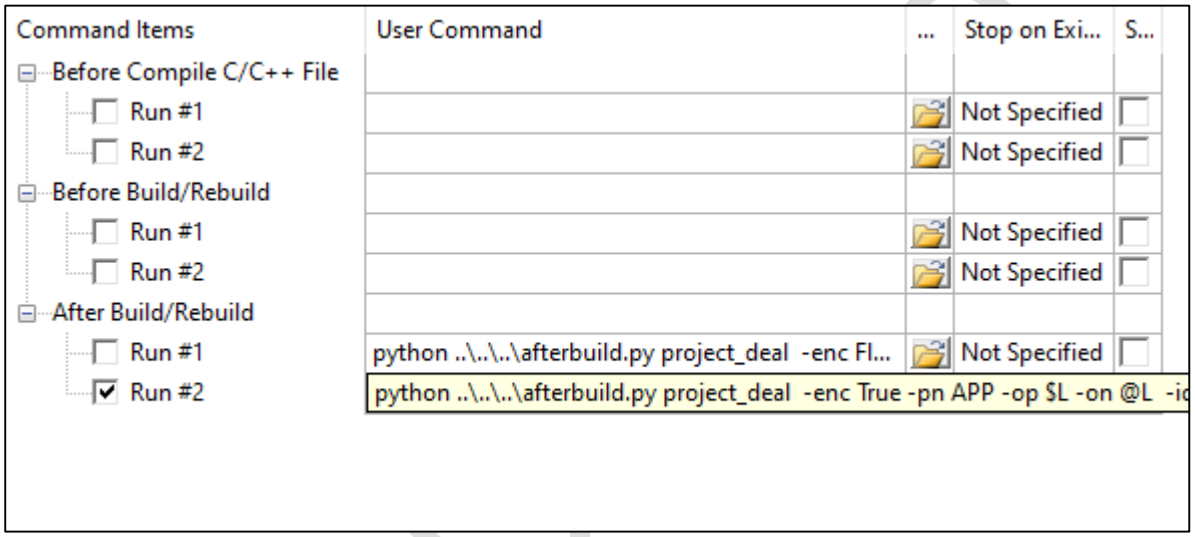
- 如*图7.19.使能升级固件加密*所示，使能MBL中的mcuboot_config.h中的相关宏。

图 7.19.使能升级固件加密

```
#if defined(MCUBOOT_ENC_IMAGES)
#define MCUBOOT_ENCRYPT_RSA
#endif
```

- 如*图7.20.生成加密固件*所示，通过Ymodem更新的APP固件，需要勾选Run#2。直接load的固件则勾选Run#1。

图 7.20.生成加密固件



- 如*图7.21.RSA key*所示，以RSA举例，密钥存放在keys.c文件中。

图 7.21.RSA key

keys.c

```
unsigned char enc_priv_key[] = {
0x30, 0x82, 0x04, 0xA3, 0x02, 0x01, 0x00, 0x02, 0x82, 0x01, 0x01, 0x00, 0xD1, 0x06, 0x08, 0x1A,
0x18, 0x44, 0x2C, 0x18, 0xE8, 0xFB, 0xFD, 0xF7, 0x0D, 0xA3, 0x4F, 0x1F, 0xBB, 0xEE, 0x5E, 0xF9,
0xAA, 0xD2, 0x4B, 0x18, 0xD3, 0x5A, 0xE9, 0x6D, 0x18, 0x80, 0x19, 0xF9, 0xF0, 0x9C, 0x34, 0x1B,
0xCB, 0xF3, 0xBC, 0x74, 0xDB, 0x42, 0xE7, 0x8C, 0x7F, 0x10, 0x53, 0x7E, 0x43, 0x5E, 0x0D, 0x57,
0x2C, 0x44, 0xD1, 0x67, 0x08, 0x0F, 0x0D, 0xBB, 0x5C, 0xEE, 0xEC, 0xB3, 0x99, 0xDF, 0xE0, 0x4D,
0x84, 0x0B, 0xAA, 0x77, 0x41, 0x60, 0xED, 0x15, 0x28, 0x49, 0xA7, 0x01, 0xB4, 0x3C, 0x10, 0xE6,
0x69, 0x8C, 0x2F, 0x5F, 0xAC, 0x41, 0x4D, 0x9E, 0x5C, 0x14, 0xDF, 0xF2, 0xF8, 0xCF, 0x3D, 0x1E,
0x6F, 0xE7, 0x5B, 0xBA, 0xB4, 0xA9, 0xC8, 0x88, 0x7E, 0x47, 0x3C, 0x94, 0xC3, 0x77, 0x67, 0x54,
0x4B, 0xAA, 0x8D, 0x38, 0x35, 0xCA, 0x62, 0x61, 0x7E, 0xB7, 0xE1, 0x15, 0xDB, 0x77, 0x73, 0xD4,
0xBE, 0x7B, 0x72, 0x21, 0x89, 0x69, 0x24, 0xFB, 0xF8, 0x65, 0x6E, 0x64, 0x3E, 0xC8, 0x0E, 0xD7,
0x85, 0xD5, 0x5C, 0x4A, 0xE4, 0x53, 0x0D, 0x2F, 0xFF, 0xB7, 0xFD, 0xF3, 0x13, 0x39, 0x83, 0x3F,
0xA3, 0xAE, 0xD2, 0x0F, 0xA7, 0x6A, 0x9D, 0xF9, 0xFE, 0xB8, 0xCE, 0xFA, 0x2A, 0xBE, 0xAF, 0xB8,
0xE0, 0xFA, 0x82, 0x37, 0x54, 0xF4, 0x3E, 0xE1, 0x2B, 0xD0, 0xD3, 0x08, 0x58, 0x18, 0xF6, 0x5E,
0x4C, 0xC8, 0x88, 0x81, 0x31, 0xAD, 0x5F, 0xB0, 0x82, 0x17, 0xF2, 0x8A, 0x69, 0x27, 0x23, 0xF3,
0xAB, 0x87, 0x3E, 0x93, 0x1A, 0x1D, 0xFE, 0xE8, 0xF8, 0x1A, 0x24, 0x66, 0x59, 0xF8, 0x1C, 0xAB,
0xDC, 0xCE, 0x68, 0x1B, 0x66, 0x64, 0x35, 0xEC, 0xFA, 0x0D, 0x11, 0x9D, 0xAF, 0x5C, 0x3A, 0xA7,
0xD1, 0x67, 0xC6, 0x47, 0xEF, 0xB1, 0x4B, 0x2C, 0x62, 0xE1, 0xD1, 0xC9, 0x02, 0x03, 0x01, 0x00,
0x01, 0x02, 0x82, 0x01, 0x00, 0x46, 0xA1, 0x14, 0x21, 0xC5, 0x2B, 0x5B, 0xFF, 0x3A, 0xD2, 0xD3,
0x79, 0x24, 0x99, 0x97, 0x45, 0xF0, 0xD9, 0xD6, 0x2B, 0xE5, 0x05, 0xD4, 0x2C, 0x5A, 0x56, 0xB0,
0xE3, 0x95, 0x50, 0xCB, 0xF6, 0x41, 0xD0, 0x76, 0x67, 0x22, 0x1E, 0x85, 0x02, 0xB3, 0x88, 0x42,
0xF7, 0x9D, 0x83, 0xE5, 0xC2, 0x97, 0x7E, 0xF3, 0x61, 0x0E, 0x6B, 0x5E, 0x9A, 0xC3, 0x05, 0x5B,
0x2D, 0x81, 0x74, 0x96, 0x75, 0x05, 0xBC, 0xB9, 0x6D, 0x57, 0xFE, 0x1D, 0x26, 0xD8, 0xE7, 0xA8,
0x94, 0xEA, 0x9D, 0x20, 0x9A, 0x99, 0xCD, 0x66, 0x24, 0x85, 0x6B, 0xC2, 0x22, 0x40, 0xF1, 0x7C,
0x09, 0xD3, 0xB1, 0x96, 0x0E, 0xE2, 0xF6, 0x1B, 0xFF, 0xE9, 0xEE, 0x32, 0x77, 0xBF, 0x4E, 0x53,
0x9D, 0x93, 0x95, 0xFC, 0xA9, 0x83, 0xF7, 0x17, 0xEA, 0x4A, 0xFB, 0x21, 0x66, 0xE9, 0xFE, 0x2E,
0x0A, 0x25, 0xA8, 0x7A, 0x9C, 0xAC, 0x06, 0xB5, 0x7F, 0x87, 0x26, 0xB8, 0xFF, 0x42, 0xF6, 0x8A,
0x9E, 0x9D, 0x9A, 0xF2, 0x96, 0x3C, 0x9F, 0x8F, 0xBA, 0x62, 0x68, 0x86, 0x46, 0x23, 0x42, 0x51,
0x3E, 0xB2, 0x49, 0xE6, 0x42, 0x26, 0x60, 0x9A, 0x91, 0x70, 0xC8, 0x0A, 0x4F, 0x21, 0xCF, 0xE0,
0xBF, 0x27, 0xC1, 0x0E, 0x26, 0xA0, 0xC2, 0x6B, 0xEB, 0x36, 0xDF, 0x6D, 0x6C, 0x71, 0x6F, 0x94,
0xC2, 0xEC, 0xA5, 0x59, 0x49, 0x0F, 0x54, 0x29, 0x96, 0x7E, 0x64, 0xBB, 0xB2, 0xFE, 0x99, 0x14,
0x41, 0xEE, 0x06, 0xF1, 0x74, 0x84, 0xD2, 0xBF, 0x91, 0xF6, 0xEE, 0x09, 0x0A, 0x0F, 0x3B, 0x23,
0x5B, 0xE0, 0x1E, 0x19, 0xFC, 0x58, 0x94, 0xB1, 0x55, 0x7D, 0x36, 0xEC, 0xE5, 0x54, 0xAB, 0x91,
0xDF, 0x3B, 0x38, 0x36, 0xB2, 0x44, 0x01, 0x4D, 0xC1, 0x31, 0xA6, 0x1E, 0x55, 0xF4, 0x1D, 0x9B,
0xA2, 0x73, 0x71, 0x77, 0xC5, 0x02, 0x81, 0x81, 0x00, 0xFF, 0x7D, 0x8F, 0x5B, 0xE6, 0x70, 0x45,
0xC7, 0x4A, 0x5C, 0xC8, 0xFE, 0x41, 0x59, 0x31, 0x5D, 0x47, 0x7A, 0xC6, 0x45, 0xF2, 0xEE, 0x64,
```

- 如*图7.22.afterbuild.py 中的密钥配置*所示，ROTPK对应固件中签名的非对称密钥对，ENCK对应给加密AES密钥的非对称密钥对。

图 7.22.afterbuild.py 中的密钥配置

```
# ROTPK        = ROOT + "/scripts/certs/ec_secp256r1_prikey.pem "
ROTPK        = ROOT + "/scripts/certs/ec_rsa2048_prikey.pem "
ENCK         = ROOT + "/scripts/certs/root-rsa-2048.pem "
CONFIG_FILE = ROOT + "/config/config_gdm32.h"
IMGTOOL      = ROOT + "/scripts/scripts_mcuboot/imgtool.py"
SREC_CAT     = ROOT + "/scripts/scripts_mcuboot/srec_cat.exe"
```

8 SBSFU 工程在 GD32H7xx 上的实施

SBSFU工程在GD32H7xx上的实施与在GD32F5xx上的实施遵循相似的架构和使用流程。其中不同需要注意的地方有：

- GD32H7xx的工程组位于“Project/GD32H7xx/MDK-ARM/SBSFU.uvmpw”；
- 如果使用O2优化等级编译IBL工程，需要使能宏“MEM_CMP_NOT_INV_ICACHE”。

9 SBSFU 工程在 GD32G5x3 上的实施

SBSFU工程在GD32G5x3上的实施与在GD32F5xx上的实施遵循相似的架构和使用流程。其中不同需要注意的地方有：

- GD32G5x3的工程组位于“Project/GD32G553/MDK-ARM/SBSFU.uvmpw”；使能了bank交换的工程组位于“Project/GD32G553_bankswap/MDK-ARM/SBSFU.uvmpw”；
- G5X3使能双bank交换后，APP代码不能获取IBL接口函数的准确地址，故APP没有适配安全更新固件的功能。

10 附录 A 功能宏说明

表 10.1. 功能宏说明

名称.	描述	值
LOG_UART	定义使用的串口	USART0：PA9、PA10
		USART1：PB15、PA8
		USART2：PB10、PB11
PLATFORM_GD32F5XX	使用 GD32F5XX 系列芯片	
INNER_ROTPK_EFUSE	ROTPK 是否存放于 EFUSE 中	0：不存放于 EFUSE 中
		1：存放于 EFUSE 中
INNER_HUK_EFUSE	HUK 是否存放于 EFUSE 中	0：不存放于 EFUSE 中
		1：存放于 EFUSE 中
GD32F5XX_OTP2_ROTPK	GD32F5XX ROTPK 是否存放于 OPT2 中	/
IMG_VERIFY_OPT	验证镜像的类型	IBL_VERIFY_CERT_IMG：验证证书（未实施）
		IBL_VERIFY_IMG_ONLY：仅验证镜像
		IBL_VERIFY_NONE：不验证，直接跳转
SPC_PROTECT_ENABLE	是否使能安全保护	/
FWDG_PROTECT_ENABLE	是否使能看门狗保护	/
IBL_TEST_SUIT	使能 IBL 的测试	/
SYS_STATUS_ENCRPTED	是否加密存储系统状态	0：不加密（未验证）
		1：加密
SIGN_ALGO_SET	验签算法	IMG_SIG_ED25519：ED25519
		IMG_SIG_ECDSA256：ECDSA256
GD32F5XX_OPT1_USED	GD32F5XX 系列将 IBL 存放于 OPT1 中	/
RE_FLASH_BASE	系统配置信息的起始地址	/
RE_SRAM_BASE	参考的 RAM 起始地址	/
RE_SHARED_DATA_START	IBL 共享数据的起始地址	如果要使用 IBL 的加密库，后续代码 RAM 地址需要使用，共享数据的起始地址之后的空间
RE_MBL_DATA_START	MBL 数据区域起始地址	
RE_APP_DATA_START	APP 数据区域起始地址	
RE_VTOR_ALIGNMENT	中断向量的对齐地址	默认 0x200
RE_SYS_SET_OFFSET	系统配置数据的偏移地址，相对于 RE_FLASH_BASE 的偏移量	默认为 0
RE_MBL_OFFSET	MBL 代码的起始偏移地址，相对于 RE_FLASH_BASE 的偏移	默认为 0x1000（系统配置数据为 4KB，一个最小的页单位）

名称.	描述	值
	量	
RE_SYS_STATUS_OFFSET	系统状态的偏移地址	默认为 0x20000
RE_IMG_0_APP_OFFSET	APP0 的偏移地址	
RE_IMG_1_APP_OFFSET	APP1 的偏移地址	
RE_MBL_VERSION	MBL 的版本	目前还不支持自动生成版本，需要调用脚本时手动添加
RE_APP_VERSION	APP 的版本	目前还不支持自动生成版本，需要调用脚本时手动添加
FLASH_BASE_IBL	IBL 代码段的基地址	
IBL_CODE_START	IBL 代码段的起始地址	默认为 FLASH_BASE_IBL
IBL_CODE_SIZE	IBL 代码段的大小	
FLASH_BASE_LIB	IBL 库的基地址	
FLASH_LIB_START	IBL 库的起始地址	
FLASH_LIB_SIZE	IBL 库的大小	
FLASH_OFFSET_SYS_SETTING	系统配置数据的偏移地址，相对于 FLASH 的基地址偏移地址	
FLASH_API_ARRAY_BASE	FLASH API 的起始地址信息	默认为 FLASH_LIB_START
FLASH_API_ARRAY_RSVD	FLASH API 的大小信息	默认为 0x800
IBL_DATA_START	IBL 数据段的起始地址	
IBL_DATA_SIZE	IBL 数据段的大小	
IBL_HEAP_SIZE	IBL HEAP 段的大小	
IBL_MSP_STACK_SIZE	IBL STACK 段的大小	
IBL_SHARED_DATA_START	IBL 共享数据的起始地址	
IBL_SHARED_DATA_SIZE	IBL 共享数据的大小	
MBL_BASE_ADDRESS	MBL 代码段的基地址	
MBL_CODE_START	MBL 代码段的起始地址	
MBL_CODE_SIZE	MBL 代码段的大小	
MBL_SHARED_DATA_START	MBL 共享数据的起始地址	
MBL_SHARED_DATA_SIZE	MBL 共享数据的大小	
MBL_DATA_START	MBL 数据的起始地址	
MBL_BUF_SIZE	MBL 分配给 mbedtls 的数据大小	
MBL_MSP_STACK_SIZE	MBL stack 的大小	
BOOTUTIL_HW_DOWNGRADE_PREVENTION	预防固件的降级	
MCUBOOT_DOWNGRADE_PREVENTION_SECURITY_COUNTER	是否使用安全计数器	1：使用安全计数器（未实施） 0：使用版本号作比较（已验证）
MCUBOOT_USE_MBED_TLS	加密库选择 mbedtls	
MCUBOOT_IMAGE_NUMBER	支持的 image 数目	目前只实施了 image 为 1 的情况

名称.	描述	值
MCUBOOT_HAVE_LOGGING	使能 log	
MCUBOOT_LOG_LEVEL	Log 打印的等级	MCUBOOT_LOG_LEVEL_OFF（0）
		MCUBOOT_LOG_LEVEL_ERROR（1）
		MCUBOOT_LOG_LEVEL_WARNING（2）
		MCUBOOT_LOG_LEVEL_INFO（3）
		MCUBOOT_LOG_LEVEL_DEBUG（4）
		MCUBOOT_LOG_LEVEL_SIM（5）
MCUBOOT_SIGN_EC256	使用的验签算法为 ECDSDA256	
MCUBOOT_VALIDATE_PRIMARY_SLOT	每次启动时验证主槽	
MCUBOOT_USE_IBL_VERIFY_SIGN	使用 IBL 的 API 进行验签	
GD32F5XX_BANKS_SWP_USED	使能 GD32F5xx 双 bank 交换功能	

11 版本历史

版本号	描述	日期
1.0	1. 初始发布版本	2024 年 6 月
1.0	1. 添加 GD32F5xx 双 bank 交换的描述	2024 年 7 月

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.