



# Flash Cube

## 使用说明

*Version: 1.5*

*Copyright @ 2024*

*[www.bouffalolab.com](http://www.bouffalolab.com)*

1	Flash Cube 简介	4
1.1	烧写界面介绍	4
2	烧写方法	7
2.1	编译应用固件	7
2.2	导入配置文件	7
2.3	烧写程序	8
2.4	启动程序	9
3	烧录配置文件介绍	11
4	新增烧写项	13
4.1	修改分区表文件	13
4.2	修改烧录配置文件	14
4.3	烧写程序	15
5	命令行下载	16
5.1	单个固件程序烧写	17
5.2	IOT 多固件程序烧写	19
5.3	RAM 烧写	23
5.4	Flash/Efuse 读写操作	24
5.5	Flash Otp 读写操作 (仅支持 BL616D/BL616L)	25
5.6	自动烧写的板子支持默认启动	26
5.7	支持 RSA 公钥加密	27
6	Flash 调试助手	28
6.1	配置通信方式	29
6.2	读 Flash ID	29
6.3	读 Flash 内容	30
6.4	擦除 Flash 内容	30
6.5	写 Flash 数据	32

6.6	读写寄存器内容 . . . . .	32
7	高级功能 . . . . .	34
7.1	支持固件路径模糊匹配 . . . . .	34
7.2	支持 ISP 烧写模式 . . . . .	36
7.3	支持压缩烧写 . . . . .	37
7.4	支持 eFuse 校验选择 . . . . .	37
7.5	支持修改烧录时擦除方式 . . . . .	39
7.6	支持擦写的 skip 功能 . . . . .	40
7.7	生成量产烧录文件 . . . . .	42
7.8	BL602/BL702 支持不填写密钥签名烧写已经加密加签的板子 . . . . .	43
7.9	新增预处理功能 . . . . .	45
7.10	支持用户自定义的 efusedata.bin 的加密密钥 . . . . .	46
8	注意事项 . . . . .	47
8.1	自定义的功能配置以用户导入为准 . . . . .	47
8.2	烧录界面每个烧录选项名称最大支持 10 个字符 . . . . .	47
8.3	晶振类型默认设定 . . . . .	47
8.4	固件超出分配的地址大小时会提示错误 . . . . .	47
8.5	固件 Flash size 时会提示错误 . . . . .	48
9	修改记录 . . . . .	49

BLFlashCube 是芯片烧写工具，支持将用户程序、分区表、boot2、用户资源等文件烧写到芯片的 Flash 中，同时工具还提供 eFuse 烧写功能，本文档主要介绍程序烧录的方法及相关配置。

BLFlashCube 的主要功能如下：

1. 支持应用程序代码等各类文件的 Flash 烧录和验证
2. 支持芯片 eFuse 烧录和验证
3. 支持多种型号 Flash 的擦、写、读
4. 下载通讯接口支持 UART、JLink、CKLink 和 OpenOCD，下载速度可配
5. 支持芯片加密或签名模式下的 Flash 烧录

用户可以通过 [Bouffalo Lab Flash Cube](#)，获取最新版本的 Flash Cube。

## 1.1 烧写界面介绍

双击解压后文件夹中的 BLFlashCube.exe，即可进入 Flash Download 程序下载主页面。

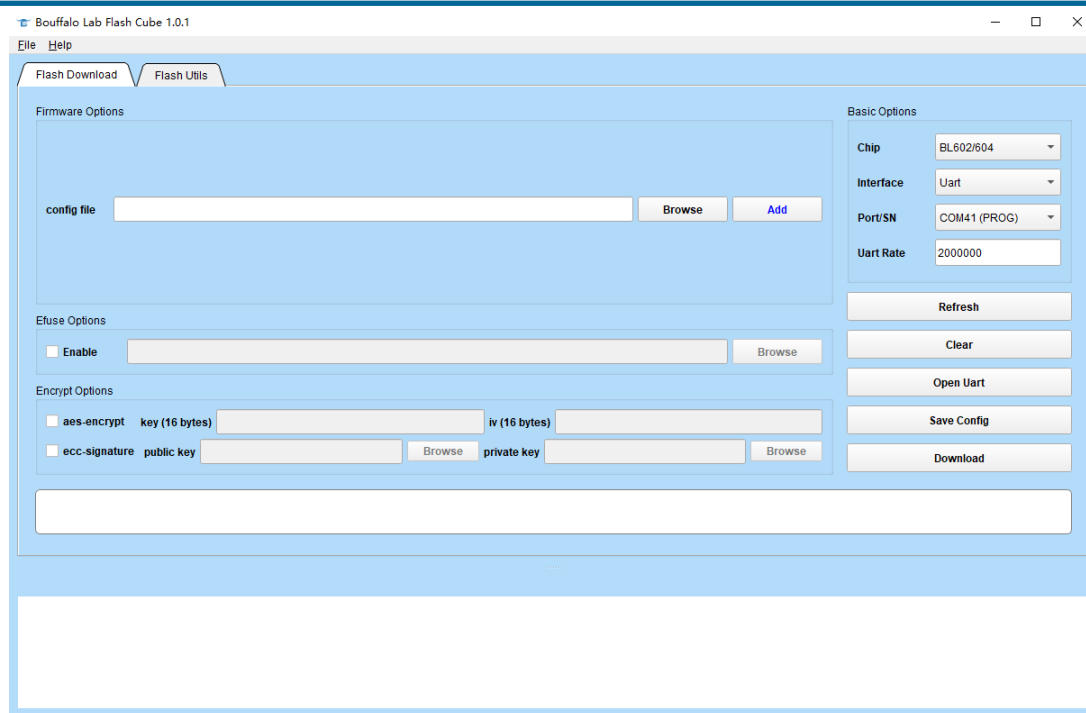


图 1.1: 烧写主界面

烧写主界面由以下几部分组成:

- **Firmware Options** 区域用于选择烧录配置文件，通过 **Brower** 按钮选择烧写使用的配置文件后，可显示具体的烧写项目和烧写地址。
- **Efuse Options** 区域用于 **eFuse** 的烧写。在勾选了 **Enable** 之后，通过 **Brower** 按钮选择相应的 **efusedata.bin** 文件。其相同目录下要存在 **efusedata\_mask.bin** 文件，用于 **eFuse** 烧录验证，否则烧写会出错。
- **Encrypt Options** 区域仅在 **BL602 / BL702** 的界面中存在，用于加密或签名模式下的烧写
  - **aes-encrypt**: 如果使用加密功能，需要将 **aes-encrypt** 选项选中，并在旁边的文本框中输入加密所使用的 **Key** 和 **IV**。输入的是十六进制对应的“0”~“F”，一个 **Byte** 由两个字符构成，所以 **Key** 和 **IV** 分别要求输入 32 个字符。需要注意的是 **IV** 的最后 8 个字符（即 4Bytes）必须全为 0
  - **ecc-signature**: 如果使用签名功能，需要将 **ecc-signature** 选项选中，并在旁边的 **public key** 选择公钥文件，**private key** 选择私钥文件，工具会生成 **pk hash** 并写入 **eFuse** 中
- **Basic Options** 区域是烧录的相关配置，如芯片类型、烧写方式、串口号等等。
  - **Chip**: 用于选择当前需要烧录的芯片类型，工具支持 **BL602/604**, **BL702/704/706**, **BL702L**, **BL808**, **BL606P** 和 **BL616/BL618** 等多种类型芯片烧写功能
  - **Interface**: 用于选择下载烧录的通信接口，可选的接口有 **UART**、**Jlink**、**CKLink** 和 **Openocd**，用户可根据实际物理连接进行选择
  - **Port/SN**: 当选择 **UART** 进行下载的时候这里选择与芯片连接的 **COM** 口号，当选择 **Jlink/CKLink/Openocd** 的时候，这里显示的是设备的端口号。可以点击 **Refresh** 按钮进行 **COM** 号或者端口号的刷新

- Uart Rate: 当选择 UART 进行下载的时候, 烧录使用的波特率, 推荐下载频率 2M
- JLink Rate: 当选择 JLink 进行下载的时候, 烧写速度的配置, 默认值是 1000
- 右侧按钮区域是相关的功能按键
  - Refresh: 用于刷新串口使用。当连接新的串口时需要点击 Refresh 按钮更新一下当前串口
  - Clear: 用于清除进度条的状态和 LOG 区域显示
  - Open Uart: 当 Interface 为 Uart 时, 打开 Port/SN 选择的串口
  - Save Config: 将当前 Firmware Options 区域的烧录项和地址保存到 config file 选择的配置文件中
  - Download: 将对应页面选择的烧写项下载到 Flash 中。如果勾选了 eFuse, 则勾选的 eFuse 数据也会写到芯片中
- 底部区域显示烧写的进度和烧写 LOG
  - 进度条: 显示当前的烧写进度。如果烧写出错, 错误类型也会显示在进度条中
  - LOG 区域: 使用 Download 按钮时显示烧写过程中的 log。使用 Open Uart 按钮时会显示启动的 log。

本文以 sdk 的 `examples/wifi/sta/wifi_ota` 为例，介绍 bl616 的烧录流程。

## 2.1 编译应用固件

```
$ cd examples/wifi/sta/wifi_ota
$ make
```

编译完成后在 `build/build_out` 路径下，生成了对应的固件 `wifi_ota_bl616.bin`，同时该目录下也包含了 `boot2`，`mfg` 等固件。

## 2.2 导入配置文件

打开 BLFlashCube 后默认进入到 Flash Download 页面，点击 Firmware Options 区域 Browse 按钮选择 `examples/wifi/sta/wifi_ota` 目录下的 `flash_prog_cfg.ini` 文件，更新后的界面如下图所示：

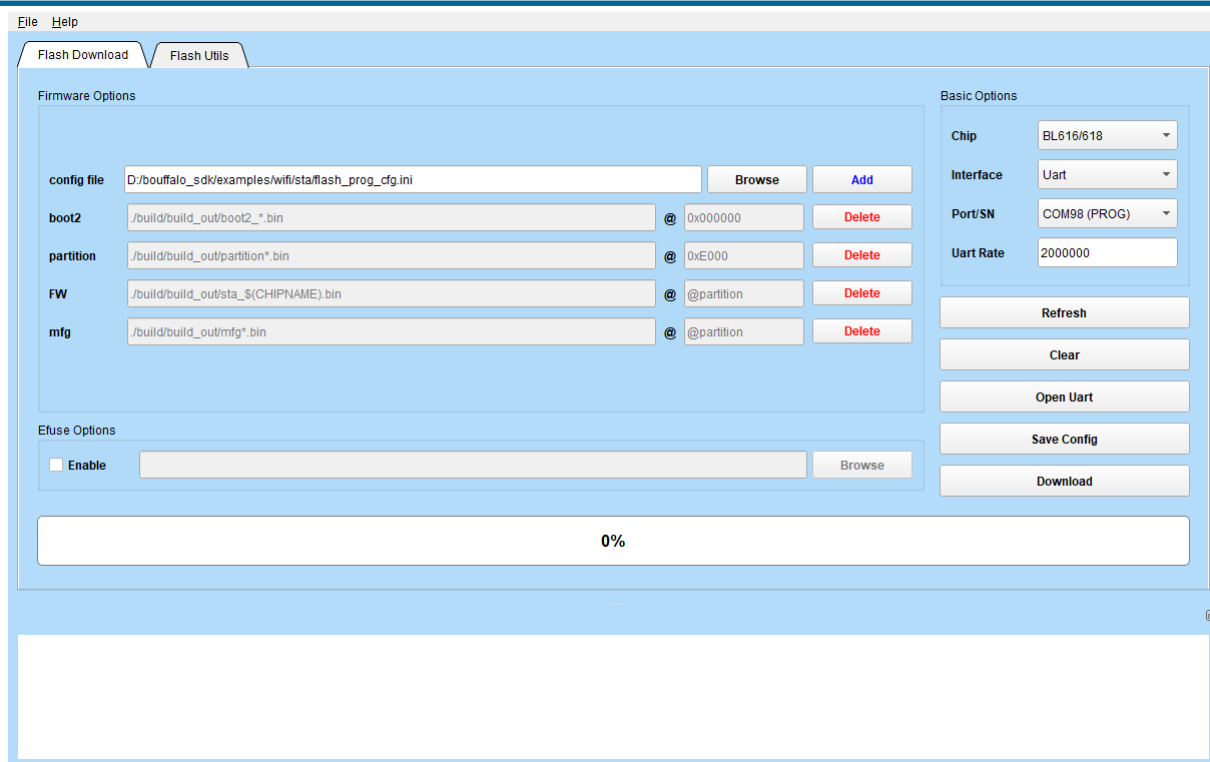


图 2.1: 导入配置文件

## 2.3 烧写程序

烧写前先确认烧写选项是否正确。将 **Basic Options** 区域的 **Chip** 选择为 BL616/618，**Interface** 选择对应的烧写方式，然后点击 **Refresh** 按钮更新串口号/序列号。

- 当选择 **UART** 方式烧写程序时，需要将板子的 **BOOT** 引脚设置为高电平，然后复位芯片，使其处于 **UART** 引导下载的状态（如果用户板子的 **Boot** 引脚和 **Reset** 引脚都与 **USB** 转串口的 **DTR** 和 **RTS** 连接，则无需手动设置，下载程序会自动设置引脚，使其处于 **UART** 引导下载的状态）。
- 当选择 **Jlink** 方式烧录时，可以一直将 **Boot** 引脚设置为低电平，让其处于从 **Flash** 启动的状态。

点击 **Download**，工具会根据页面配置向指定的地址烧录文件。



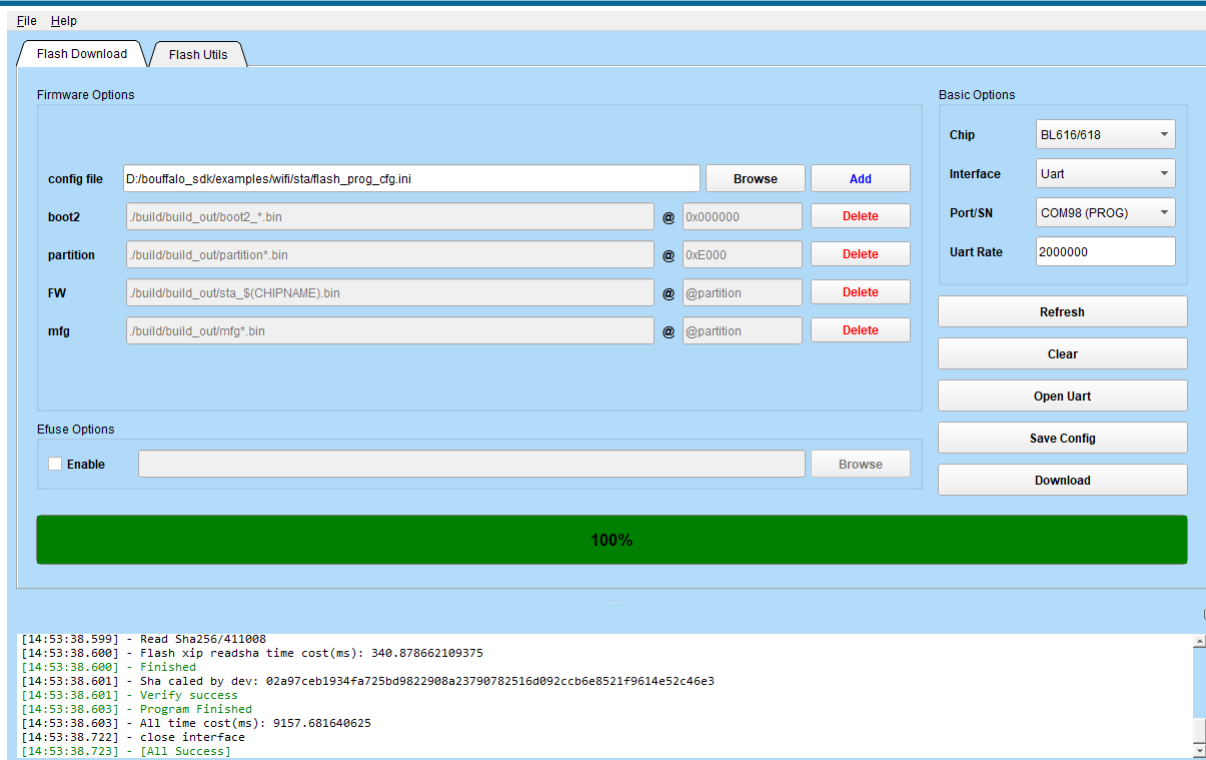


图 2.2: 成功下载程序

当出现如图所示 100% 的绿色进度条时，则表示程序下载成功。

注解：若没有连接板子，只需生成完整的镜像文件，亦可点击 Download 按钮生成

## 2.4 启动程序

下载成功后，将板子的 BOOT 引脚设置为低电平，复位芯片，使其从 Flash 启动，此时应用程序即可运行。

下图是 wifi/sta/wifi\_ota 程序运行起来的效果。

```
[14:39:48.564] - [14:39:48.566] - [14:39:48.568] - [14:39:48.570] - [14:39:48.572] - [14:39:48.573] - [14:39:48.574] - Build:14:26:08,Mar 17 2023 [14:39:48.576] - Copyright (c) 2022 BouffaloLab team [14:39:48.577] - ***** flash cfg ***** [14:39:48.578] - jedec id 0xEF4017 [14:39:48.580] - mid 0xEF [14:39:48.581] - lmode 0x04 [14:39:48.581] - clk delay 0x01 [14:39:48.581] - clk invert 0x01 [14:39:48.582] - read reg cmd0 0x05 [14:39:48.582] - read reg cmd1 0x35 [14:39:48.583] - write reg cmd0 0x01 [14:39:48.584] - write reg cmd1 0x31 [14:39:48.585] - qe write len 0x01 [14:39:48.586] - cread support 0x00 [14:39:48.586] - cread code 0xFF [14:39:48.587] - burst wrap cmd 0x77 [14:39:48.588] - ***** [14:39:48.590] - dynamic memory init success, ocran heap size = 150 Kbyte, psram heap size = 4096 Kbyte [14:39:48.591] - sig1:ffffff [14:39:48.592] - sig2:0000f32f [14:39:48.592] - cgen1:9f7ffffd [14:39:48.593] - @BouffaloLab />@[0mlwp init done [14:39:48.925] - sys_mbox_new done! [14:39:48.926] - sys_mutex_new done! [14:39:48.927] - tcpip thread init done! [14:39:48.927] - [I][MAIN] Starting wifi ... [14:39:48.929] - [I][rparam] rparam>>xtal value 40000000 [14:39:48.929] - [I][rparam] rparam>>pw_mode is bf [14:39:48.930] - Empty slot:1 [14:39:48.931] - [I][rparam] rparam>>efuse wlan pwr_offset[14]: 1,1,1,1,1,0,0,0,0,0,0,0,0,0, [14:39:48.932] - [I][rparam] rparam>>tlv wlan pwr_offset[14]: 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, [14:39:48.934] - [I][rparam] rparam>>wlan pwr_offset[14]: 1,1,1,1,1,0,0,0,0,0,0,0,0,0,0, [14:39:48.934] - Empty slot:0 [14:39:48.935] - No written slot found [14:39:48.938] - [I][rparam] rparam>>no lp pwr_offset in efuse [14:39:48.939] - [I][rparam] rparam>>tlv wlan lp pwr_offset[14]: 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, [14:39:48.941] - [I][rparam] rparam>>wlan lp pwr_offset[14]: 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, [14:39:48.943] - Empty slot:0 [14:39:48.944] - No written slot found [14:39:48.946] - [I][rparam] rparam>>no bz pwr_offset in efuse [14:39:48.946] - [I][rparam] rparam>>tlv bz pwr_offset[5]: 0,0,0,0,0, [14:39:48.947] - [I][rparam] rparam>>bz pwr_offset[5]: 0,0,0,0,0, [14:39:48.949] - [I][rparam] rparam>>pw_11b[4]: 20,20,20,20, [14:39:48.950] - [I][rparam] rparam>>pw_11g[8]: 18,18,18,18,18,18,16,16, [14:39:48.951] - [I][rparam] rparam>>pw_11n_ht40[8]: 18,18,18,18,18,16,15,15, [14:39:48.953] - [I][rparam] rparam>>pw_11n_ht40[8]: 18,18,18,18,18,16,15,14, [14:39:48.955] - [I][rparam] rparam>>pw_11ac[16]: 18,18,18,18,18,18,18,18,18,18,18,18,18,18,18,18,
```

图 2.3: wifi\_ota 程序结果

## 烧录配置文件介绍

根据 `examples/wifi/sta/wifi_ota` 的烧录需求，配置文件 `flash_prog_cfg.ini` 中包含了分区表、Boot2、Firmware、mfg 等固件烧录信息。本章节介绍一下烧录配置文件的组成。

```
[cfg]
# 0: no erase, 1:programmed section erase, 2: chip erase
erase = 1
# skip mode set first para is skip addr, second para is skip len, multi-segment region with ; separated
skip_mode = 0x0, 0x0
# 0: not use isp mode, #1: isp mode
boot2_isp_mode = 0

[boot2]
filedir = ./build/build_out/boot2_*.bin
address = 0x000000

[partition]
filedir = ./build/build_out/partition.bin
address = 0xE000

[FW]
filedir = ./build/build_out/wifi_ota*_${CHIPNAME}.bin
address = @partition

[mfg]
filedir = ./build/build_out/mfg*.bin
address = @partition
```

**cfg** 表示烧录时的一些配置，正常使用默认值即可

- **erase**: 设置烧写时的擦除方式。默认的 **erase = 1**，表示下载时按照烧录地址和内容大小进行擦除。**erase = 2** 表示程序烧录之前会将 Flash 全部擦除。**erase = 0** 表示烧写前不进行擦除操作，一般不使用。

- **skip\_mode**: 设置擦写时不操作的区域。第一个参数为起始地址，第二个参数为长度。**skip\_mode** 支持同时配置多个区域，中间以“;”分隔。
- **boot2\_isp\_mode**: 控制是否选择 isp 烧写模式。**boot2\_isp\_mode** = 1 表示使用 isp 烧写模式。

**boot2** 表示要烧录的 boot2 固件

- **filedir**: boot2 固件所在相对路径
- **address**: 必须使用 0 地址

**partition** 表示要烧录的 partition 固件，必须使用 partition 名称。

- **filedir**: partition 固件所在相对路径
- **address**: 由 SDK 提供的分区表文件 ‘partition\_xxx.toml’ 指定

**FW** 表示要烧录的应用固件，使用“FW” 可以从分区表中获取。

- **filedir**: 应用固件所在相对路径。其中 “wifi\_ota” 表示应用固件名称，\$(CHIPNAME) 表示芯片类型。
- **address** 使用“@partition” 表示自动从 partition.bin 中获取烧录地址。也可以直接指定烧录地址,如 **address** = 0x10000

**mfg** 表示要烧录的 mfg 固件，使用“mfg” 可以从分区表中获取。

- **filedir**: mfg 固件所在相对路径
- **address**: 使用“@partition” 表示自动从 partition.bin 中检测 mfg 地址。也可以直接指定烧录地址，如 **address** = 0x210000。

对于 IOT 多固件程序烧写，如果要增加新的烧写项，需要修改配置信息。

- 修改分区表文件，增加新烧写项的分区信息
- 在 `flash_prog_cfg.ini` 文件中增加新烧写项的配置

下面以 bl616 增加 `test` 烧写项为例，介绍添加的流程。

## 4.1 修改分区表文件

在 SDK 的“`bsp/board/bl616dk/config`”目录下有命名格式为“`partition_xxx.toml`”的分区表文件，增加新的烧写项只需要修改这个分区表文件就可以。

例如：将提前准备好的 `test.bin` 烧写到 `flash` 的 `0x378000` 位置，固件大小为 `0x1000`。

在分区表中增加新的 `name` 为“`test`”的分区信息，其中 `address0` 为 `0x378000`，`size` 为 `0x1000`，其他配置使用默认值。

`type` 表示分区类型，`Boot2` 会根据这个 `type` 来启动镜像。当 `type` 等于 `0` 时表示 `CPU0` 启动的镜像，等于 `1` 时表示 `CPU1` 启动的镜像。因此，客户在自定义分区表的时候，要避开 `0` 和 `1`，否则会被 `Boot2` 当成可运行的镜像启动运行。详细的介绍可以参考“分区表说明文档”

```
[[pt_entry]]
type = 8
name = "test"
device = 0
address0 = 0x378000
size0 = 0x1000
address1 = 0
size1 = 0
# compressed image must set len, normal image can left it to 0
len = 0
```

修改之后重新编译应用固件，在编译 LOG 中看到“`Create partition using partition_xxx.toml`”即表示在 `build/build_out`

目录下成功生成了分区表文件 `partition.bin`。

如果提示 log: "[Warning] No partiton file found in ./../bsp/board/bl616dk/config,go on next steps", 则表示没有找到分区表文件, 用户需要检查"bsp/board/bl616dk/config" 目录下的分区表文件是否存在。

## 4.2 修改烧录配置文件

打开 `examples/wifi/sta/wifi_ota` 目录下的 `flash_prog_cfg.ini`, 增加 `test` 烧写项, 添加 `test` 烧写项后的配置文件内容如下:

```
[cfg]
# 0: no erase, 1:programmed section erase, 2: chip erase
erase = 1
# skip mode set first para is skip addr, second para is skip len, multi-segment region with ; separated
skip_mode = 0x0, 0x0
# 0: not use isp mode, #1: isp mode
boot2_isp_mode = 0

[boot2]
filedir = ./build/build_out/boot2_*.bin
address = 0x000000

[partition]
filedir = ./build/build_out/partition.bin
address = 0xE000

[FW]
filedir = ./build/build_out/wifi_ota*_${CHIPNAME}.bin
address = @partition

[mfg]
filedir = ./build/build_out/mfg*.bin
address = @partition

[test]
filedir = ./build/build_out/test*.bin
address = @partition
```

在新增的 `test` 烧录项中:

- `filedir` 指定烧录的 `bin` 文件位置, 示例中使用的是相对路径(相对于配置文件的路径)。建议将新增的烧录固件也拷贝到 `FW` 同目录下。
- `address` 指定烧录地址, 示例中使用 `@partition` 表示从分区表中自动获取烧写地址, 也可以直接指定烧录地址 `0x378000`。使用 `@partition` 的好处是: 如果该分区要更换地址, 只要修改分区表文件即可。

## 4.3 烧写程序

修改成功后使用 **BLFlashCube** 工具导入新的烧录配置文件然后烧写即可，导入后的烧写界面如下：

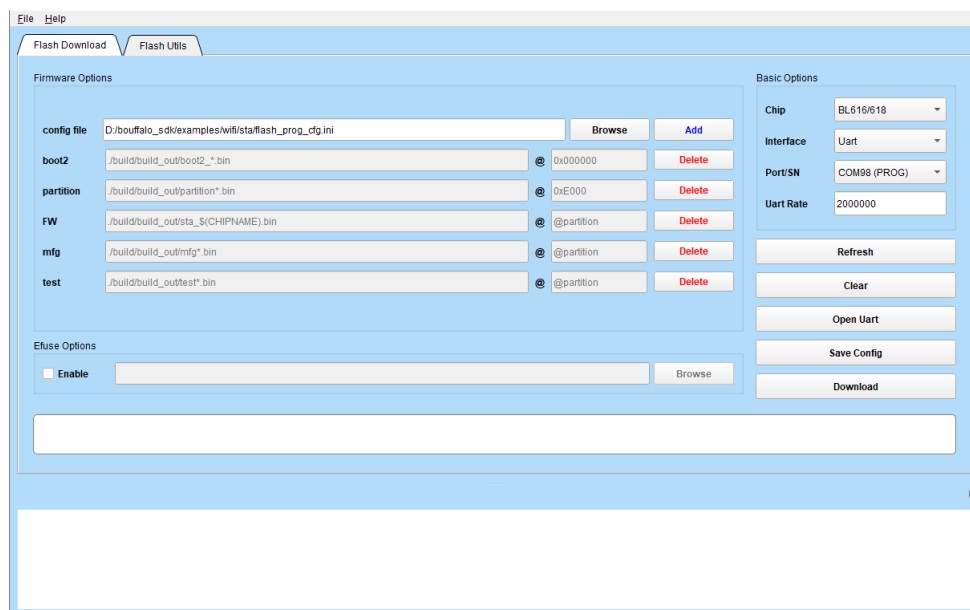


图 4.1: test 烧写项成功导入到界面

烧写成功界面和烧写 log 如下图：

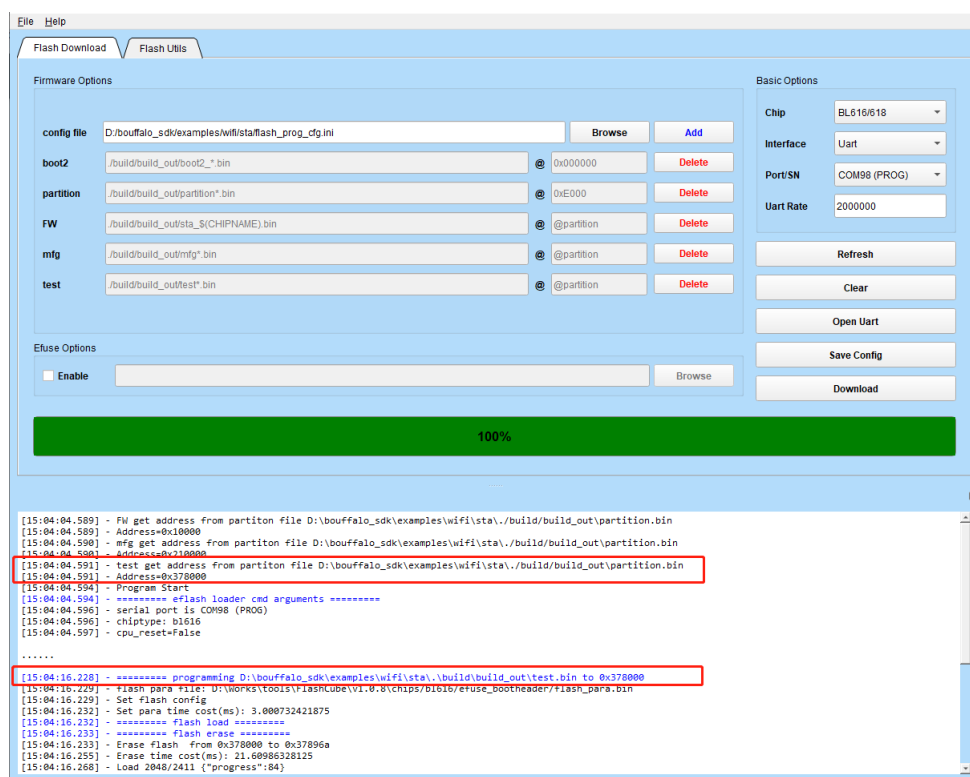


图 4.2: 成功下载程序

BLFlashCube 提供了命令行的烧写方式，Windows 环境下的可执行文件为 BLFlashCommand.exe，Linux 下的可执行文件为 BLFlashCommand-ubuntu。

具体使用说明如下：

```
PS D:\Works\BLFlashCube> .\BLFlashCommand.exe --help
usage: BLFlashCommand.py [-h] [--interface INTERFACE] [--port PORT]
                        [--chipname CHIPNAME] [--baudrate BAUDRATE]
                        [--config CONFIG] [--firmware FIRMWARE]
                        [--efusefile EFUSEFILE] [--cpu_id CPU_ID]
                        [--key KEY] [--iv IV] [--pk PK] [--sk SK]
                        [--pk_str PK_STR] [--sk_str SK_STR]
                        [--lockidx LOCKIDX] [--lockstart LOCKSTART]
                        [--lockend LOCKEND] [--flash] [--flash_otp]
                        [--efuse] [--erase] [--build] [--read] [--write]
                        [--start START] [--end END] [--len LEN]
                        [--file FILE] [--ram] [--reset]
                        [--efuse_encrypted EFUSE_ENCRYPTED] [--addr ADDR]
                        [--whole_chip]

flash-command

options:
  -h, --help            show this help message and exit
  --interface INTERFACE
                        interface to use
  --port PORT            serial port to use
  --chipname CHIPNAME    chip name
  --baudrate BAUDRATE    the speed at which to communicate
  --config CONFIG        run config
```

(continues on next page)



```
--firmware FIRMWARE    image to write
--efusefile             efuse write file
--cpu_id CPU_ID        cpu id
--key KEY              aes key
--iv IV                aes iv
--pk PK                ecc public key
--sk SK                ecc private key
--pk_str PK_STR        ecc public key string
--sk_str SK_STR        ecc private key string
--lockidx LOCKIDX      flash otp lock index
--lockstart LOCKSTART

                        otp lock start address (hex, e.g., 0x1A)
--lockend LOCKEND      otp lock end address (hex, e.g., 0x2C)
--flash                Indicate flash operation
--flash_otp            Indicate flash otp operation
--efuse                Read or write efuse
--erase                Erase flash memory
--build                build pack
--read                 Read from flash memory
--write                Write to flash memory
--start START          Start address (hex, e.g., 0x1A2B)
--end END              End address (hex, e.g., 0x1A2C)
--len LEN              Length (hex, e.g., 0x100)
--file FILE            File for reading or writing
--ram                  Download image to RAM
--reset                Reset CPU after download
--efuse_encrypted EFUSE_ENCRYPTED

                        encrypted data to write

--whole_chip           Erase whole flash
```

用户烧写的时候需要使用参数**--interface**指定烧写接口，**--port**指定烧写串口号，**--chipname**指定芯片类型，**--baudrate**指定烧写波特率，**--config**指定使用的烧录配置文件。

## 5.1 单个固件程序烧写

命令行工具可以使用 **--firmware** 参数直接指定要烧录的固件，默认烧录到 Flash 的 0 地址起始位置。如需在烧录之前将 flash 全部擦除，可在命令中增加**--whole\_chip**选项。

以 BLFlashCommand.exe 为例，烧写 wifi\_ota\_bl616.bin 到 Flash 的命令如下：

```
D:\Works\tools\BLFlashCube\v1.0.8> .\BLFlashCommand.exe --interface=uart --chipname=bl616 --port=COM98 --
-baudrate=2000000 --firmware=D:\bouffalo_sdk\examples\wifi\sta\build\build_out\sta_bl616.bin
```

(continues on next page)

```
[15:43:00.454] - Serial port is COM98
[15:43:00.454] - =====
[15:43:00.455] - Program Start
[15:43:00.456] - ===== eflash loader cmd arguments =====
[15:43:00.457] - serial port is COM98
[15:43:00.457] - chiptype: bl616
[15:43:00.457] - cpu_reset=False
[15:43:00.480] - com speed: 2000000
[15:43:00.480] - ===== Interface is uart =====
[15:43:00.480] - Bootrom load
[15:43:00.480] - ===== get_boot_info =====
[15:43:00.480] - ===== image get bootinfo =====
[15:43:00.485] - default set DTR high
[15:43:00.587] - usb serial port
[15:43:00.649] - clean buf
[15:43:00.653] - send sync
[15:43:00.867] - ack is b'4f4b'
[15:43:00.869] - shake hand success
[15:43:01.382] - data read is b'0100160600000100279280015e64de05b91819000f758010'
[15:43:01.382] - ===== ChipID: 18b905de645e =====
[15:43:01.382] - Get bootinfo time cost(ms): 902.612060546875
[15:43:01.382] - change bdrate: 2000000
[15:43:01.383] - Clock PLL set
[15:43:01.383] - Set clock time cost(ms): 0.0
[15:43:01.504] - Read mac addr
[15:43:01.506] - MACADDR: 18b905de645e
[15:43:01.506] - flash set para
[15:43:01.507] - get flash pin cfg from bootinfo: 0x02
[15:43:01.507] - set flash cfg: 1014102
[15:43:01.508] - Set flash config
[15:43:01.510] - Set para time cost(ms): 1.98876953125
[15:43:01.510] - ===== flash read jedec ID =====
[15:43:01.511] - Read flash jedec ID
[15:43:01.511] - readdata:
[15:43:01.511] - b'c8401600'
[15:43:01.511] - Finished
[15:43:01.511] - flash config Not found,use default
[15:43:01.511] - jedec_id:c84016
[15:43:01.511] - capacity_id:22
[15:43:01.511] - capacity:4.0M
[15:43:01.511] - get flash size: 0x00400000
[15:43:01.511] - Program operation
```

(continues on next page)

(continued from previous page)

```
[15:43:01.511] - Dealing Index 0
[15:43:01.511] - ===== programming D:\bouffalo_sdk\examples\wifi\sta\build\build_out\sta_bl616.bin to
->0x0
[15:43:01.511] - flash para file: D:\Works\tools\FlashCube\v1.0.8\chips/bl616/efuse_bootheader/flash_para.
->.bin
[15:43:01.512] - Set flash config
[15:43:01.515] - Set para time cost(ms): 2.417236328125
[15:43:01.515] - ===== flash load =====
[15:43:01.515] - ===== flash erase =====
[15:43:01.515] - Erase flash from 0x0 to 0xca61f
[15:43:02.226] - Erase time cost(ms): 711.35107421875
[15:43:02.496] - decompress flash load 494024
[15:43:02.507] - Load 2048/494024 {"progress":0}
[15:43:06.088] - Load 494024/494024 {"progress":100}
[15:43:06.088] - Write check
[15:43:06.102] - Flash load time cost(ms): 3874.525634765625
[15:43:06.102] - Finished
[15:43:06.107] - Sha calcd by host: 6167624bb39d78d164eada22e9802520fb2bea0b526a5563fc4ea6568d557747
[15:43:06.107] - xip mode Verify
[15:43:06.793] - Read Sha256/828960
[15:43:06.793] - Flash xip readsha time cost(ms): 686.288818359375
[15:43:06.793] - Finished
[15:43:06.794] - Sha calcd by dev: 6167624bb39d78d164eada22e9802520fb2bea0b526a5563fc4ea6568d557747
[15:43:06.794] - Verify success
[15:43:06.795] - Program Finished
[15:43:06.795] - All time cost(ms): 6340.654296875
[15:43:06.914] - close interface
[15:43:06.914] - [All Success]
```

## 5.2 IOT 多固件程序烧写

参照“新增烧写项”的章节，修改分区表文件和烧录配置文件。

以 BLFlashCommand.exe 为例，烧录 flash\_prog\_cfg.ini 的配置项到 Flash 的命令如下：

```
PS D:\Works\tools\FlashCube\v1.0.8> .\BLFlashCommand.exe --interface=uart --chipname=bl616 --port=COM98 --
->baudrate=2000000 --config=D:/bouffalo_sdk/examples/wifi/sta/flash_prog_cfg.ini
[15:47:11.112] - Serial port is COM98
[15:47:11.113] - =====
[15:47:11.116] - FW get address from partiton file D:\bouffalo_sdk\examples\wifi\sta\./build/build_out\
->partition.bin
[15:47:11.116] - Address=0x10000
[15:47:11.117] - mfg get address from partiton file D:\bouffalo_sdk\examples\wifi\sta\./build/build_out\
->partition.bin
```

(continues on next page)

(continued from previous page)

```
[15:47:11.118] - Address=0x210000
[15:47:11.119] - Program Start
[15:47:11.119] - ===== eflash loader cmd arguments =====
[15:47:11.120] - serial port is COM98
[15:47:11.120] - chiptype: bl616
[15:47:11.120] - cpu_reset=False
[15:47:11.257] - com speed: 2000000
[15:47:11.257] - ===== Interface is uart =====
[15:47:11.257] - Bootrom load
[15:47:11.257] - ===== get_boot_info =====
[15:47:11.257] - ===== image get bootinfo =====
[15:47:11.262] - default set DTR high
[15:47:11.366] - usb serial port
[15:47:11.428] - clean buf
[15:47:11.431] - send sync
[15:47:11.645] - ack is b'4f4b'
[15:47:11.648] - shake hand success
[15:47:12.160] - data read is b'0100160600000100279280015e64de05b91819000f758010'
[15:47:12.160] - ===== ChipID: 18b905de645e =====
[15:47:12.160] - Get bootinfo time cost(ms): 903.37841796875
[15:47:12.160] - change bdrate: 2000000
[15:47:12.160] - Clock PLL set
[15:47:12.161] - Set clock time cost(ms): 0.26513671875
[15:47:12.282] - Read mac addr
[15:47:12.283] - MACADDR: 18b905de645e
[15:47:12.284] - flash set para
[15:47:12.284] - get flash pin cfg from bootinfo: 0x02
[15:47:12.285] - set flash cfg: 1014102
[15:47:12.285] - Set flash config
[15:47:12.287] - Set para time cost(ms): 1.990966796875
[15:47:12.287] - ===== flash read jedec ID =====
[15:47:12.287] - Read flash jedec ID
[15:47:12.287] - readdata:
[15:47:12.287] - b'c8401600'
[15:47:12.287] - Finished
[15:47:12.287] - flash config Not found,use default
[15:47:12.287] - jedec_id:c84016
[15:47:12.287] - capacity_id:22
[15:47:12.287] - capacity:4.0M
[15:47:12.287] - get flash size: 0x00400000
[15:47:12.287] - Program operation
[15:47:12.288] - Dealing Index 0
[15:47:12.288] - ===== programming D:\bouffalo_sdk\examples\wifi\sta\.\build\build_out\boot2_bl616_
```

```
->release_v8.0.7.bin to 0x000000
```

(continues on next page)

(continued from previous page)

```
[15:47:12.288] - flash para file: D:\Works\tools\FlashCube\v1.0.8\chips/bl616/efuse_bootheader/flash_para.
- bin
[15:47:12.288] - Set flash config
[15:47:12.292] - Set para time cost(ms): 3.126953125
[15:47:12.292] - ===== flash load =====
[15:47:12.292] - ===== flash erase =====
[15:47:12.292] - Erase flash from 0x0 to 0xa93f
[15:47:12.599] - Erase time cost(ms): 307.765380859375
[15:47:12.615] - decompress flash load 21796
[15:47:12.788] - Load 21796/21796 {"progress":100}
[15:47:12.789] - Write check
[15:47:12.806] - Flash load time cost(ms): 204.3623046875
[15:47:12.806] - Finished
[15:47:12.807] - Sha caled by host: 81bdd6bd9e028b2d1fa5da8d12aa4438353842d3f2a0b85e61a4efb00dd50fd0
[15:47:12.807] - xip mode Verify
[15:47:12.844] - Read Sha256/43328
[15:47:12.844] - Flash xip readsha time cost(ms): 36.54638671875
[15:47:12.844] - Finished
[15:47:12.845] - Sha caled by dev: 81bdd6bd9e028b2d1fa5da8d12aa4438353842d3f2a0b85e61a4efb00dd50fd0
[15:47:12.845] - Verify success
[15:47:12.846] - Dealing Index 1
[15:47:12.846] - ===== programming D:\bouffalo_sdk\examples\wifi\sta\.\build\build_out\partition.bin to
- 0xE000
[15:47:12.846] - flash para file: D:\Works\tools\FlashCube\v1.0.8\chips/bl616/efuse_bootheader/flash_para.
- bin
[15:47:12.846] - Set flash config
[15:47:12.849] - Set para time cost(ms): 2.998779296875
[15:47:12.849] - ===== flash load =====
[15:47:12.849] - ===== flash erase =====
[15:47:12.849] - Erase flash from 0xe000 to 0xe133
[15:47:12.888] - Erase time cost(ms): 39.04541015625
[15:47:12.893] - Load 308/308 {"progress":100}
[15:47:12.893] - Write check
[15:47:12.894] - Flash load time cost(ms): 3.004150390625
[15:47:12.894] - Finished
[15:47:12.895] - Sha caled by host: 6c50e14f2776fb705aaffb46d9022f2c062a296303d27c9545715585ddf93625
[15:47:12.895] - xip mode Verify
[15:47:12.896] - Read Sha256/308
[15:47:12.896] - Flash xip readsha time cost(ms): 1.000732421875
[15:47:12.896] - Finished
[15:47:12.896] - Sha caled by dev: 6c50e14f2776fb705aaffb46d9022f2c062a296303d27c9545715585ddf93625
[15:47:12.896] - Verify success
[15:47:12.898] - Dealing Index 2
```

(continues on next page)

(continued from previous page)

```
[15:47:12.898] - ===== programming D:\bouffalo_sdk\examples\wifi\sta\.\build\build_out\sta_bl616.bin to 0x10000
[15:47:12.899] - flash para file: D:\Works\tools\FlashCube\v1.0.8\chips/bl616/efuse_bootheader/flash_para.bin
[15:47:12.899] - Set flash config
[15:47:12.901] - Set para time cost(ms): 1.998046875
[15:47:12.901] - ===== flash load =====
[15:47:12.902] - ===== flash erase =====
[15:47:12.902] - Erase flash from 0x10000 to 0xda61f
[15:47:14.840] - Erase time cost(ms): 1938.134033203125
[15:47:15.112] - decompress flash load 494024
[15:47:18.693] - Load 494024/494024 {"progress":100}
[15:47:18.694] - Write check
[15:47:18.706] - Flash load time cost(ms): 3865.414794921875
[15:47:18.707] - Finished
[15:47:18.710] - Sha calcd by host: 6167624bb39d78d164eada22e9802520fb2bea0b526a5563fc4ea6568d557747
[15:47:18.710] - xip mode Verify
[15:47:19.396] - Read Sha256/828960
[15:47:19.396] - Flash xip readsha time cost(ms): 685.072998046875
[15:47:19.397] - Finished
[15:47:19.397] - Sha calcd by dev: 6167624bb39d78d164eada22e9802520fb2bea0b526a5563fc4ea6568d557747
[15:47:19.397] - Verify success
[15:47:19.399] - Dealing Index 3
[15:47:19.399] - ===== programming D:\bouffalo_sdk\examples\wifi\sta\.\build\build_out\mfg_bl616_gu_af8b0946f_v2.26.bin to 0x210000
[15:47:19.400] - flash para file: D:\Works\tools\FlashCube\v1.0.8\chips/bl616/efuse_bootheader/flash_para.bin
[15:47:19.400] - Set flash config
[15:47:19.403] - Set para time cost(ms): 2.56005859375
[15:47:19.403] - ===== flash load =====
[15:47:19.403] - ===== flash erase =====
[15:47:19.403] - Erase flash from 0x210000 to 0x27457f
[15:47:20.325] - Erase time cost(ms): 922.190673828125
[15:47:20.461] - decompress flash load 222180
[15:47:22.110] - Load 222180/222180 {"progress":100}
[15:47:22.110] - Write check
[15:47:22.124] - Flash load time cost(ms): 1797.292236328125
[15:47:22.124] - Finished
[15:47:22.126] - Sha calcd by host: 02a97ceb1934fa725bd9822908a23790782516d092ccb6e8521f9614e52c46e3
[15:47:22.126] - xip mode Verify
[15:47:22.467] - Read Sha256/411008
[15:47:22.467] - Flash xip readsha time cost(ms): 339.635986328125
[15:47:22.468] - Finished
```

(continues on next page)

```
[14:39:48.564] -
[14:39:48.566] -
[14:39:48.568] -
[14:39:48.570] -
[14:39:48.572] -
[14:39:48.573] -
[14:39:48.574] Build:14:26:08,Mar 17 2023
[14:39:48.576] Copyright (c) 2022 Bouffalolab team
[14:39:48.577] ----- Flash cfg -----
[14:39:48.578] jedec id 0x6F4017
[14:39:48.580] mid 0x0F
[14:39:48.581] lomode 0x04
[14:39:48.581] clk delay 0x01
[14:39:48.581] clk invert 0x01
[14:39:48.582] read reg cmd0 0x05
[14:39:48.582] read reg cmd1 0x35
[14:39:48.583] write reg cmd0 0x01
[14:39:48.584] write reg cmd1 0x31
[14:39:48.585] oe write len 0x01
[14:39:48.586] cread support 0x00
[14:39:48.586] cread code 0xFF
[14:39:48.587] burst wrap cmd 0x77
[14:39:48.588] -----
[14:39:48.590] dynamic memory init success, ocran heap size = 150 Kbyte, psram heap size = 4096 Kbyte
[14:39:48.591] sig1:0000fffff
[14:39:48.592] sig2:0000f32ff
[14:39:48.592] cgen1:9f7ffffd
[14:39:48.593] 0@0bouffalolab /S@0mlwip init done
[14:39:48.925] sys_mbox_new done!
[14:39:48.926] sys_mutex_new done!
[14:39:48.927] tcpip thread init done!
[14:39:48.927] [I][MAIN] Starting wifi ...
[14:39:48.929] [I][rparam] rparam>xtal value 40000000
[14:39:48.929] [I][rparam] rparam>pur_mode is bf
[14:39:48.930] Empty slot:1
[14:39:48.931] [I][rparam] rparam>efuse wlan pur_offset[14]: 1,1,1,1,1,0,0,0,0,0,0,0,0,
[14:39:48.932] [I][rparam] rparam>tiv wlan pur_offset[14]: 0,0,0,0,0,0,0,0,0,0,0,0,0,
[14:39:48.934] [I][rparam] rparam>wlan pur_offset[14]: 1,1,1,1,1,0,0,0,0,0,0,0,0,
[14:39:48.934] Empty slot:0
[14:39:48.935] No written slot found
[14:39:48.936] [I][rparam] rparam>no lp pur_offset in efuse
[14:39:48.939] [I][rparam] rparam>tiv wlan lp pur_offset[14]: 0,0,0,0,0,0,0,0,0,0,0,0,0,
[14:39:48.941] [I][rparam] rparam>wlan lp pur_offset[14]: 0,0,0,0,0,0,0,0,0,0,0,0,0,
[14:39:48.943] Empty slot:0
[14:39:48.944] No written slot found
[14:39:48.946] [I][rparam] rparam>no bz pur_offset in efuse
[14:39:48.946] [I][rparam] rparam>tiv bz pur_offset[5]: 0,0,0,0,0,
[14:39:48.947] [I][rparam] rparam>bz pur_offset[5]: 0,0,0,0,0,
[14:39:48.949] [I][rparam] rparam>pur_l1b[4]: 20,20,20,20,
[14:39:48.950] [I][rparam] rparam>pur_l1b[8]: 18,18,18,18,18,18,16,16,
[14:39:48.951] [I][rparam] rparam>pur_l1n_H20[8]: 18,18,18,18,18,16,15,15,
[14:39:48.953] [I][rparam] rparam>pur_l1n_H240[8]: 18,18,18,18,18,16,15,14,
```

图 5.1: 启动 log

### 5.3 RAM 烧写

以 BLFlashCommand.exe 为例，烧写 flash forbidden cmd test bl616.bin 到 Flash 的命令如下：

```
D:\Works\tools\BLFlashCube\vl.1.2> .\BLFlashCommand.exe --interface=uart --chipname=bl616 --port=COM98 --
baudrate=2000000 --firmware=D:\bouffalo_sdk\chiptest\bl616\flash\flash_forbidden_cmd_test\build\build_out\
flash_forbidden_cmd_test_bl616.bin --ram

[15:47:57.661] - Serial port is COM98
[15:47:57.661] - =====
[15:47:57.681] - ===== image load =====
[15:47:57.688] - default set DTR high
[15:47:57.797] - usb serial port
[15:47:57.860] - clean buf
[15:47:57.905] - send sync
```

(continues on next page)

(continued from previous page)

```
[15:47:58.128] - ack is b'4f4b'
[15:47:58.130] - shake hand success
[15:47:58.144] - get_boot_info
[15:47:58.146] - data read is b'0200160600000100679680013bf22c5042f41a000f758010c42a6dad'
[15:47:58.146] - ===== ChipID: f442502cf23b =====
[15:47:58.146] - last boot info: None
[15:47:58.146] - sign is 0 encrypt is 0
[15:47:58.167] - Download D:\bouffalo_sdk\chiptest\bl616\flash\flash_forbidden_cmd_test\build\build_out\
-flash_forbidden_cmd_test_bl616.bin
[15:47:58.167] - segcnt is 1
[15:47:58.175] - segdata_len is 34496
[15:47:58.258] - 4080/34496
[15:47:58.341] - 8160/34496
[15:47:58.425] - 12240/34496
[15:47:58.507] - 16320/34496
[15:47:58.590] - 20400/34496
[15:47:58.673] - 24480/34496
[15:47:58.756] - 28560/34496
[15:47:58.839] - 32640/34496
[15:47:58.877] - 34496/34496
[15:47:58.878] - Run img
[15:47:58.879] - Img run success
[15:47:58.879] - All time cost(ms): 1198.422607421875
[15:47:58.879] - True
```

## 5.4 Flash/Efuse 读写操作

```
# flash chip erase
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --flash --erase --whole_chip
```

```
# flash section erase using start and end address
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --flash --erase --start=0x00 --end=0xFFFF
```

```
# flash section erase using start and len
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --flash --erase --start=0x00 --len=0x1000
```

```
# program file data to flash
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --flash --write --start=0x1000 --
-file=write_file.bin
```



```
# erase whole flash chip before program file data to flash
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --flash --write --start=0x1000 --
--file=write_file.bin --whole_chip
```

```
# read data from flash and save to read_file.bin
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --flash --read --start=0x00 --end=0xFF
--file=read_file.bin
```

```
# read data from flash and save to read_file.bin
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --flash --read --start=0x00 --len=0x1000 --
--file=read_file.bin
```

```
# efuse write
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --efusefile=efusedata.bin
```

```
# read data from efuse and save to read_file.bin
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --efuse --read --start=0x00 --end=0xFF --
--file=read_file.bin
```

```
# read data from efuse and save to read_file.bin
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --efuse --read --start=0x00 --len=0x200 --
--file=read_file.bin
```

```
# program file data to flash as well as efuse
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --flash --write --start=0x1000 --
--file=write_file.bin --efusefile=efusedata.bin
```

```
# erase whole flash chip before program file data to flash then burn efuse
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --flash --write --start=0x1000 --
--file=write_file.bin --efusefile=efusedata.bin --whole_chip
```

## 5.5 Flash Otp 读写操作 (仅支持 BL616D/BL616L)

本工具提供 **flash otp** 功能，可以对其进行擦除和读写操作，适合需要高安全性和数据保护的应用场合，数据写入后一旦 **lock** 住就不能被修改或擦除。通常用于存储固定的、重要的数据，如设备的序列号、加密密钥或配置参数。由于其不可更改的特性，OTP 存储器在安全性和防篡改方面具有优势。

```
# read flash_otp data from flash using start and end address
--interface=uart --chipname=bl616d --port=COM24 --flash_otp --read --start=0x0 --end=0xff --baudrate=115200
```

```
# read flash_otp data from flash using start and len
--interface=uart --chipname=bl616d --port=COM24 --flash_otp --read --start=0x0 --len=0x100 --baudrate=115200
```

```
# flash_otp section erase using start and end address
--interface=uart --chipname=bl616d --port=COM24 --flash_otp --erase --start=0x0 --end=0xff --baudrate=115200
```

```
# flash_otp section erase using start and len
--interface=uart --chipname=bl616d --port=COM24 --flash_otp --erase --start=0x1 --len=0x105 --
-baudrate=115200
```

```
# read flash_otp data from flash and save to read_file.bin
--interface=uart --chipname=bl616d --port=COM24 --flash_otp --read --start=0x0 --end=0xff --baudrate=115200
--file=read_file.bin
```

```
# program file flash_otp data to flash
--interface=uart --chipname=bl616d --port=COM24 --flash_otp --write --start=0x0 --file=write_file.bin --
-baudrate=115200
```

```
# lock flash_otp data by index
--interface=uart --chipname=bl616d --port=COM24 --flash_otp --baudrate=115200 --lockidx=1
```

```
# lock flash_otp data from address
--interface=uart --chipname=bl616d --port=COM24 --flash_otp --baudrate=115200 --lockstart=0x0 --lockend=0x30
```

```
# program file flash_otp data to flash and lock flash_otp data by index
--interface=uart --chipname=bl616d --port=COM24 --flash_otp --write --start=0x0 --file=test.bin --
-baudrate=115200 --lockidx=1
```

```
# program file flash_otp data to flash and lock flash_otp data from address
--interface=uart --chipname=bl616d --port=COM24 --flash_otp --write --start=0x0 --file=test.bin --
-baudrate=115200 --lockstart=0x0 --lockend=0x10
```

## 5.6 自动烧写的板子支持默认启动

如果板子的 **Boot** 引脚和 **Reset** 引脚都与 USB 转串口的 DTR 和 RTS 连接，下载程序会自动设置引脚，使其处于 UART 引导下载的状态或者启动的状态。用户使用了 **--reset** 参数，在固件烧录完成后，下载程序会自动复位板子进入启动状态。

```
# flash reset after being downloaded
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --firmware=D:\buffalo_sdk\examples\wifi\
-sta\build\build_out\sta_bl616.bin --reset
```

## 5.7 支持 RSA 公钥加密

工具支持将数据写到 efuse 的指定位置。使用 `--data` 传入的是非加密的数据，使用 `--efuse_encrypted` 可以传入加密的 efuse 数据（使用 RSA 公钥加密），安全性能更高。命令行工具使用 `--efuse_encrypted` 传入加密的 efuse 数据 `--addr` 传入指定位置，以下传入的 `--efuse_encrypted` 为示例。

```
# flash reset after being downloaded
--interface=uart --chipname=bl1616 --port=COM24 --baudrate=2000000 --efuse_
-encrypted=3cfbc0fc209b1192bc52208f03ae5b2d880e3fb0ee26577d993d57fcb5bd8531979d022a68b6f41c7081e776ded777
bf23fb0a832e27c9bbab8b7d938171c6fb48a18bd0f21d8ba1a91c8e55509c1552d5343cd8d49e74c4fd12ba4c0734a0422bfc48a
719f02f3c24d54f530f63a74341bdc94f5a5f96287bb4c02b3b90d288501bca4cfeeab413e391ffdec9c5216e0591ee72cd7bacc94
796cb0ecbd94e985bea160b8778849372e29f9471bbf3ea824f3a097d89c2d77e2834b61643c5c7f7aaf88970217cab069f51b6658
2c6a7d0494c0579f299535d6e2f3d65905fe23964b4eb865e285f6cd715973645511cee6825c7f52d8a77f6a482ecff9e62d9f32b4
d3c8c66e9db9cecf8ee5909c7bbe165f0fa06a8366434fdbcff031c1a6f0d742b13556ef8a5ede64f0be904ceb35180f7b8cd97262
c217ecfec2055d4d20636e6f67cb6b462f75169e37ab96c323d92ea8a7ca1c116c080be3ad8003479d666fbf06efaa111b073bf815
9ed2a3066ef1a3371a9eadfc4e8de1734f34f003a5989ca3a1b8de22298a79f8ac3791da7787a7f921e0c0f7dfee4dfc4f75dcea13
9db544ba77b43682bdeef1ae8b33cdd5a9783cf4502c75e163a73a23f69596de41aa2af35722fe857a82a48ff376b37f61928a4f4c
c20b7e108e248eddb90dc608c657045bd46ca5151087892bfc31dec2cf04a8087b66e97e082ae02b2834751b179155bc11401a31a0
ef0eb9cfc7c1ed5af2de7e534d6144212177b9f4746e9a126c8c10b60515c0b4158738ebb961a10fc628b551fed7cd17553f351dba
7bbbe19007af7895bd800f1d2a0d6eb244f2695881077fdd87f969fafc67f390a072907f6ad654ee6557c0a590f0536135558dc37a
ad4d62c5f0c6ee460de94821b18146daa96938f716bd327211abe7febc1ceb195d96d468b9802680cae90f6dd1b0b6814ea7d90629
be0f6c61567156b04fe8c79362844ddfa2a15641550fe202c78ae0053725f826488e0b05cbcca16a39f4b45f69d9235ec3f37e0945
14727ae4a80443bfc669f953f1802e550d8af479dd8d4b68afe2fea5e0b62b03 --addr=0x0
```

在首行菜单中选择 **Flash Utils** 选项，会进入 **Flash 调试助手** 界面。**Flash 调试助手** 用来获取 **Flash ID**、读取、擦除和写入 **Flash** 指定地址的内容、读取和写入寄存器的值。

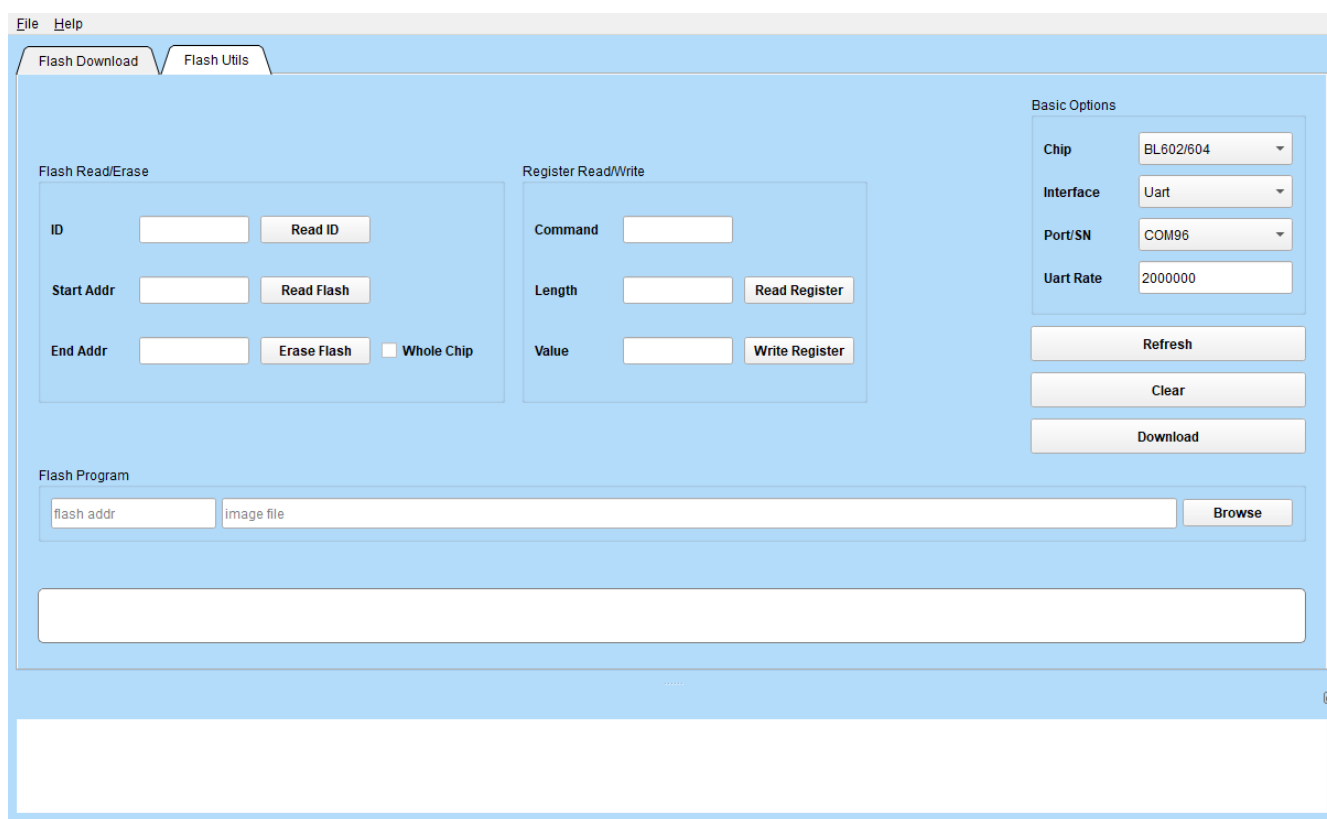


图 6.1: Flash 调试助手界面

## 6.1 配置通信方式

- Basic Options 区域配置参数包括：
  - Chip: 用于选择当前需要烧录的芯片类型，支持 BL602/604, BL702/704/706, BL808, BL606P 和 BL616/BL618 等多种类型芯片烧写功能。
  - Interface: 用于选择下载烧录的通信接口，可选的接口有 Jlink、UART、CKLink 和 Openocd，用户根据实际物理连接进行选择
  - Port/SN: 当选择 UART 进行下载的时候这里选择与芯片连接的 COM 口号，当选择 Jlink/CKLink/Openocd 的时候，这里显示的是设备的端口号。可以点击 Refresh 按钮进行 COM 号或者端口号的刷新
  - Uart Rate: 当选择 UART 进行下载的时候，需要填写波特率，推荐下载频率 2M
  - JLink Rate: 当选择 JLink 进行下载的时候，烧写速度的配置，默认值是 1000

## 6.2 读 Flash ID

- 读取 Flash 的 ID: 点击 Read ID

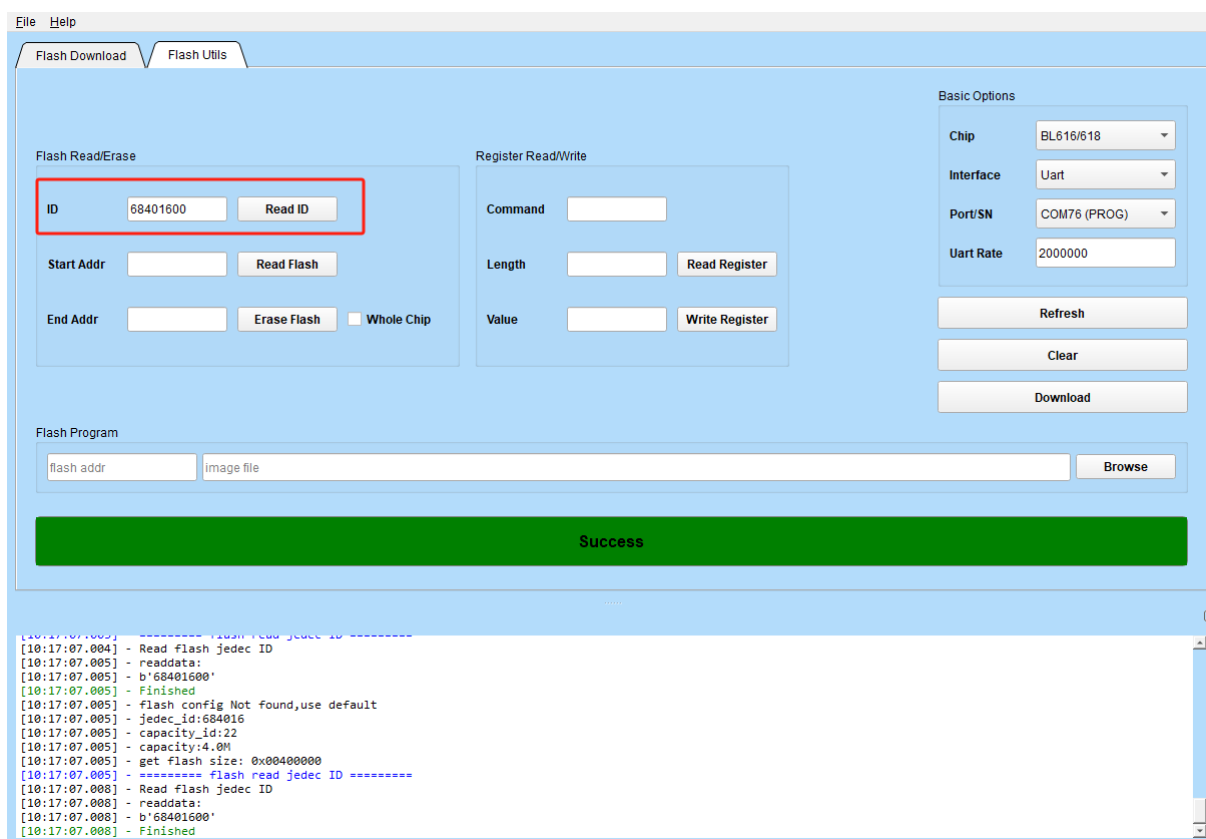


图 6.2: 读取 flash ID

## 6.3 读 Flash 内容

- 读取 Flash 指定地址段的数据：在 **Start Addr** 中填写需要读取数据的开始地址，在 **End Addr** 中填写需要读取数据的结束地址，点击 **Read Flash**，读取的内容会更新在工具根目录下 **flash.bin** 文件中

以下示例为读取 Flash 0x0 ~ 0x2000 地址的数据：

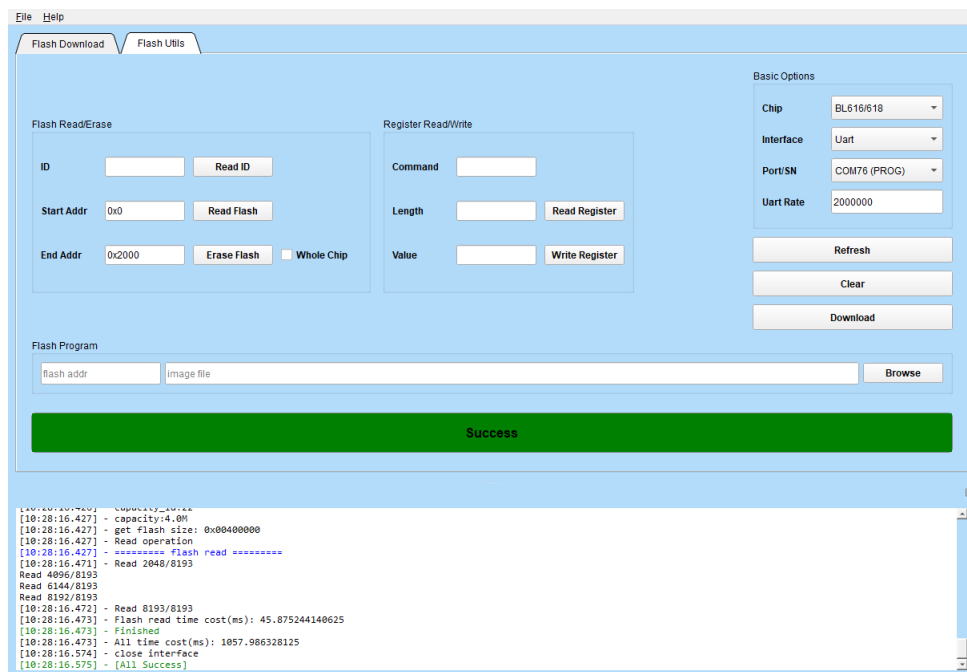


图 6.3: 读取 flash 0x0 ~ 0x2000 地址的数据

## 6.4 擦除 Flash 内容

- 擦除 Flash 指定地址段的数据：在 **Start Addr** 中填写需要擦除数据的开始地址，在 **End Addr** 中填写需要擦除数据的结束地址。点击 **Erase Flash** 后，工具就会进行擦除操作。

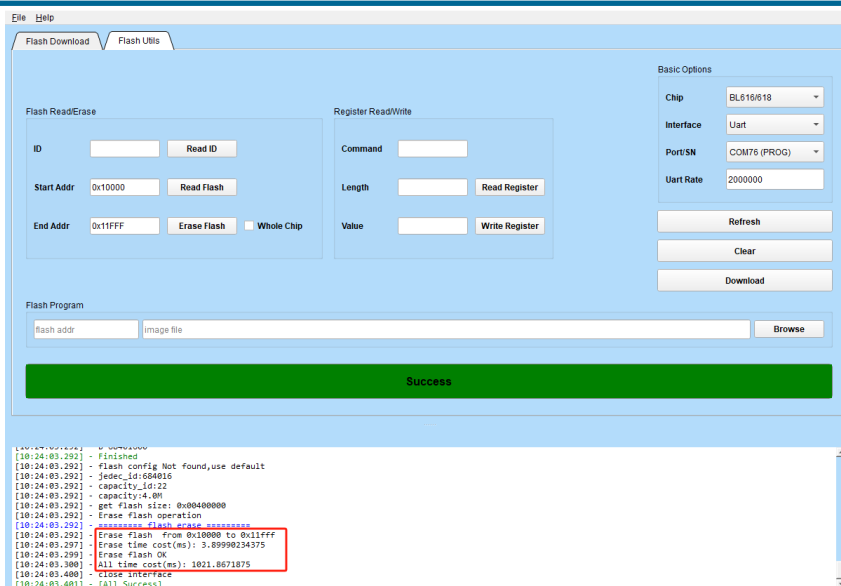


图 6.4: 擦除 Flash 界面

注解: Flash 擦除的单位为 4KB(0x1000), 需要注意 End Addr 不要超出

- 擦除整块芯片的 Flash 区域, 需要勾选 Whole Chip, 然后点击 Erase Flash 即可。

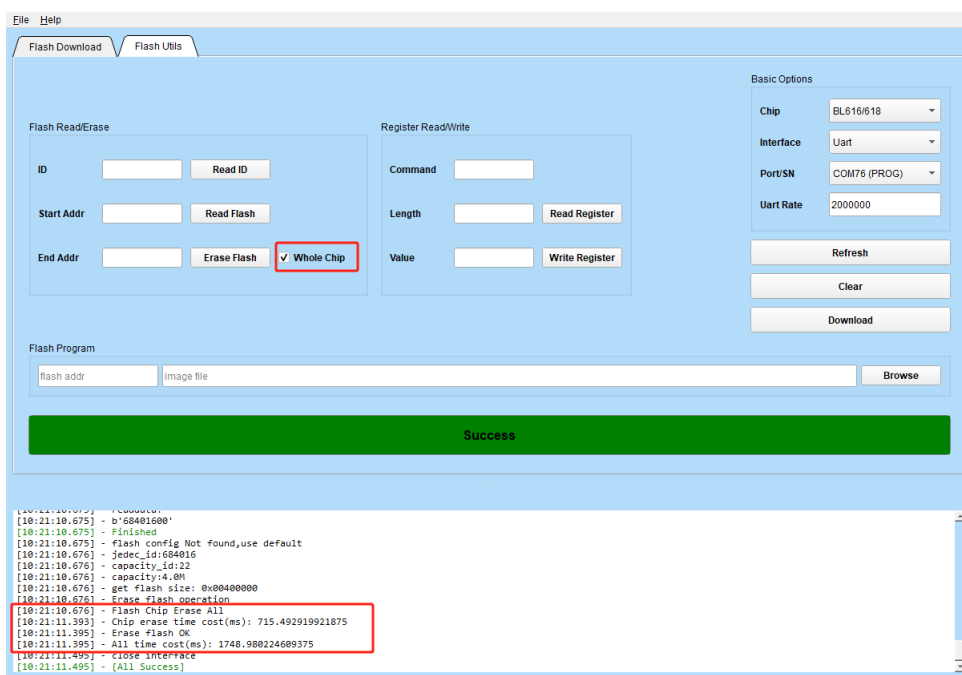


图 6.5: 擦除 Flash 界面

## 6.5 写 Flash 数据

- 写 Flash 数据至指定地址：可下载单个 Raw 文件到指定的 Flash 地址，在左侧文本框填写下载的起始地址，以 0x 打头，在右侧文本框填写 Raw 文件路径，点击 download 下载

以下示例为将 bl616\_demo\_event\_mfg\_nopsram.bin 下载到 0x0 地址：

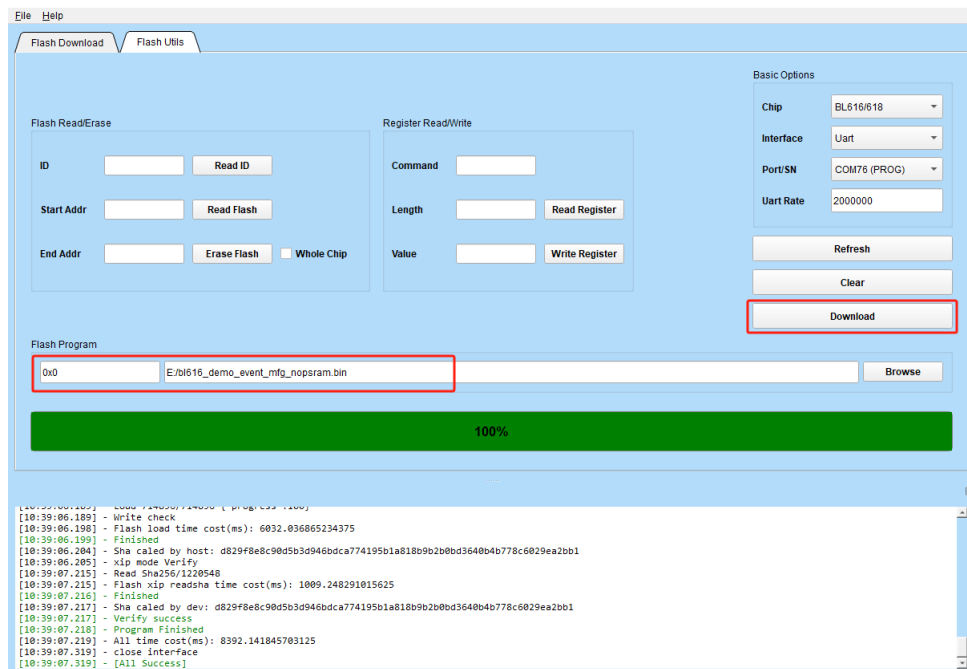


图 6.6: 将 bl616\_demo\_event\_mfg\_nopsram.bin 下载到 0x0 地址

## 6.6 读写寄存器内容

- 读取寄存器的内容：在 Command 中输入读取命令 0x05/0x35, Length 中填写需要读取的位数，点击 Read Register，读取的数据会显示在 Value 中



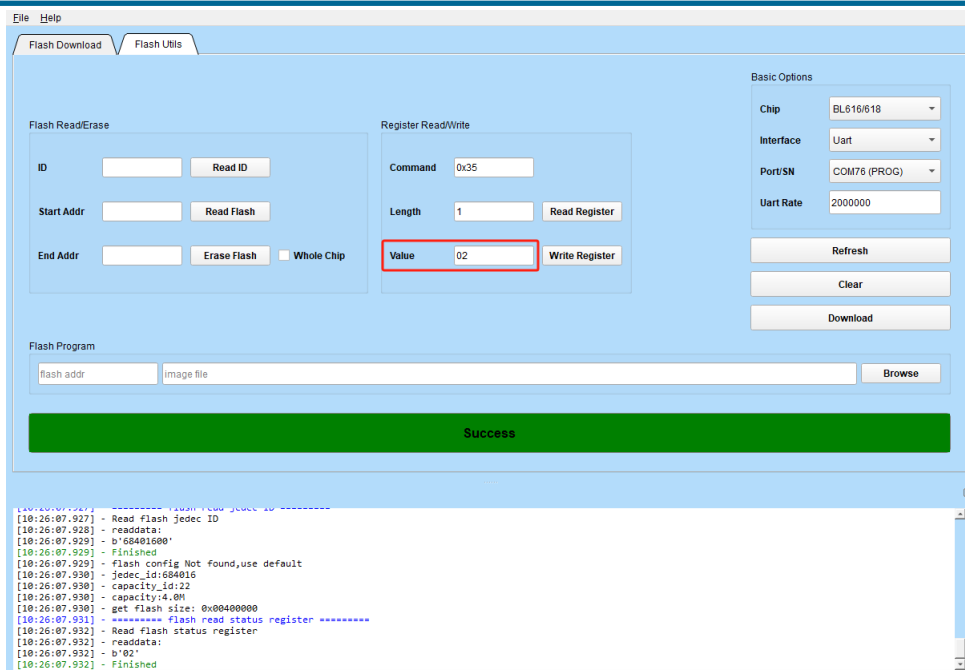


图 6.7: 读寄存器界面

- 写入寄存器内容：在 Command 中输入写命令 0x01/0x31，Length 中填写需要写入的位数，将写入的数据填写在 Value 中，点击 Write Register

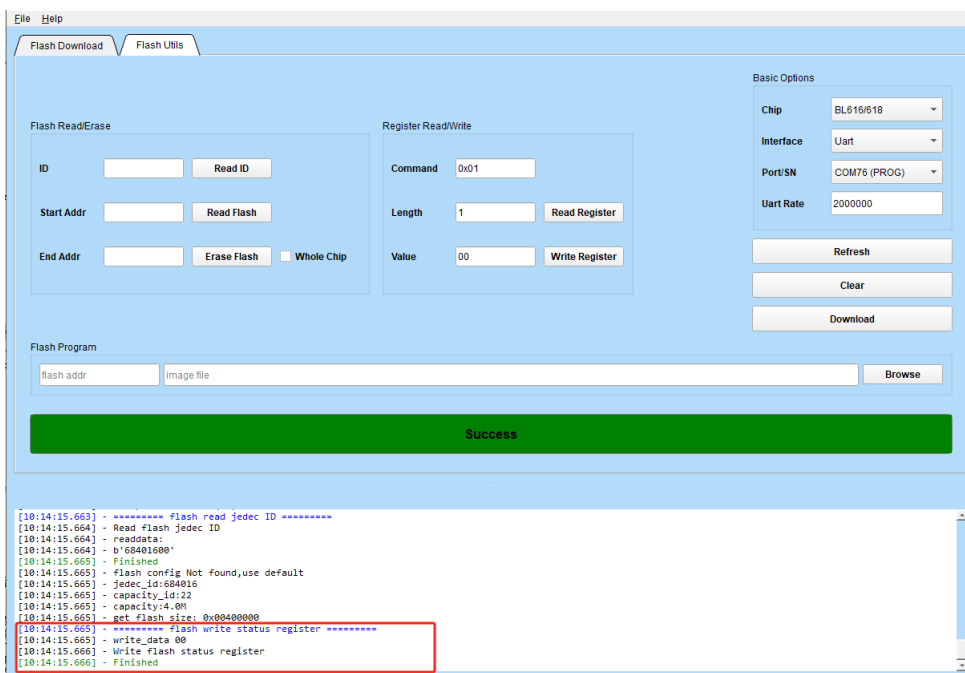


图 6.8: 写寄存器界面

Flash Cube 还提供一些高级烧写功能，通过修改配置文件的方式实现。

## 7.1 支持固件路径模糊匹配

用户导入的配置文件 `flash_prog_cfg.ini` 中，固件路径可以使用类似于“`./build/build_out/helloworld*_$(CHIPNAME).bin`”的方式，由工具去匹配需要烧写的测试固件。

SDK 的 `examples/wifi/sta/wifi_ota` 目录下的 `flash_prog_cfg.ini` 配置文件中使用的模糊匹配的方式。

```
[cfg]
# 0: no erase, 1:programmed section erase, 2: chip erase
erase = 1
# skip mode set first para is skip addr, second para is skip len, multi-segment region with ; separated
skip_mode = 0x0, 0x0
# 0: not use isp mode, #1: isp mode
boot2_isp_mode = 0

[boot2]
filedir = ./build/build_out/boot2_*.bin
address = 0x000000

[partition]
filedir = ./build/build_out/partition.bin
address = 0xE000

[FW]
filedir = ./build/build_out/wifi_ota*_$(CHIPNAME).bin
address = @partition

[mfg]
```

(continues on next page)

```
filedir = ./build/build_out/mfg*.bin
address = @partition
```

- [boot2] 的 filedir 使用“./build/build\_out/boot2\_\*.bin”模糊匹配可以查找到 build/build\_out 目录下的 boot2\_bl616\_release\_v8.0.7.bin 文件。
- [FW] 的 filedir 使用“./build/build\_out/wifi\_ota\*\$(CHIPNAME).bin”模糊匹配可以查找到 build/build\_out 目录下的 wifi\_ota\_bl616.bin 文件，其中 \$(CHIPNAME) 取决于烧写界面的 Chip 选择的芯片类型。
- [mfg] 的 filedir 使用“./build/build\_out/mfg\*.bin”模糊匹配可以查找到 build/build\_out 目录下的 mfg\_bl616\_gu\_af8b0946f\_v2.26.bin 文件。

如果匹配到的文件不止一个，工具会提示错误：“Error: Multiple files were matched!”

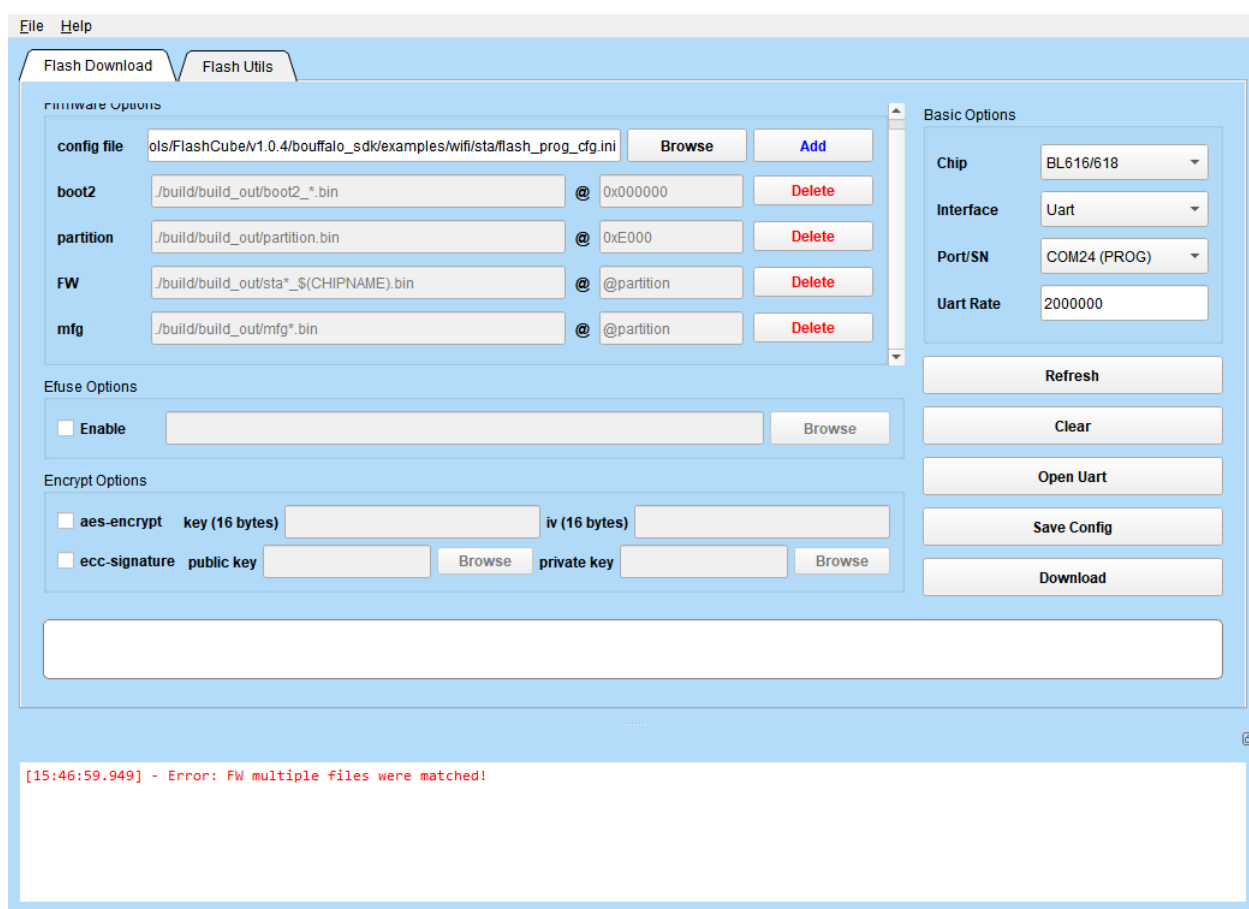


图 7.1: 查找固件错误

## 7.2 支持 ISP 烧写模式

ISP 即 In-System Programming，BLFlashCube 支持 ISP 模式烧写，详细请参考文档“ISP\_下载使用说明”。

以 BL602 为例，boot2\_isp\_mode 控制是否选择 isp 烧写模式，isp\_mode\_speed 控制和 boot2 通信触发 isp 烧写的波特率配置。其中 boot2\_isp\_mode 在用户自定义的烧录配置文件中设定，isp\_mode\_speed 在工具的“chips/bl602/eflash\_loader/eflash\_loader\_cfg.ini”中定义。修改自定义配置文件中的“boot2\_isp\_mode = 0”为“boot2\_isp\_mode = 1”，然后保存文件即可以使用 ISP 烧写模式。

操作步骤如下：

首先确保芯片中已经烧录并启动了 Boot2 程序，然后修改配置文件中“boot2\_isp\_mode = 1”并保存文件。点击工具页面中的 Download，如下图所示，烧录过程中会提示“Please Press Reset Key!”，此时用户需要在 5 秒钟以内复位芯片，看到“read ready”或“isp ready”的 log 表示工具握手成功，然后等待烧录完成即可。

如果是自动烧写的板子，在提示“Please Press Reset Key!”之后，工具会控制 Reset 引脚自动复位芯片执行 ISP 的烧录流程，而无需手动操作。

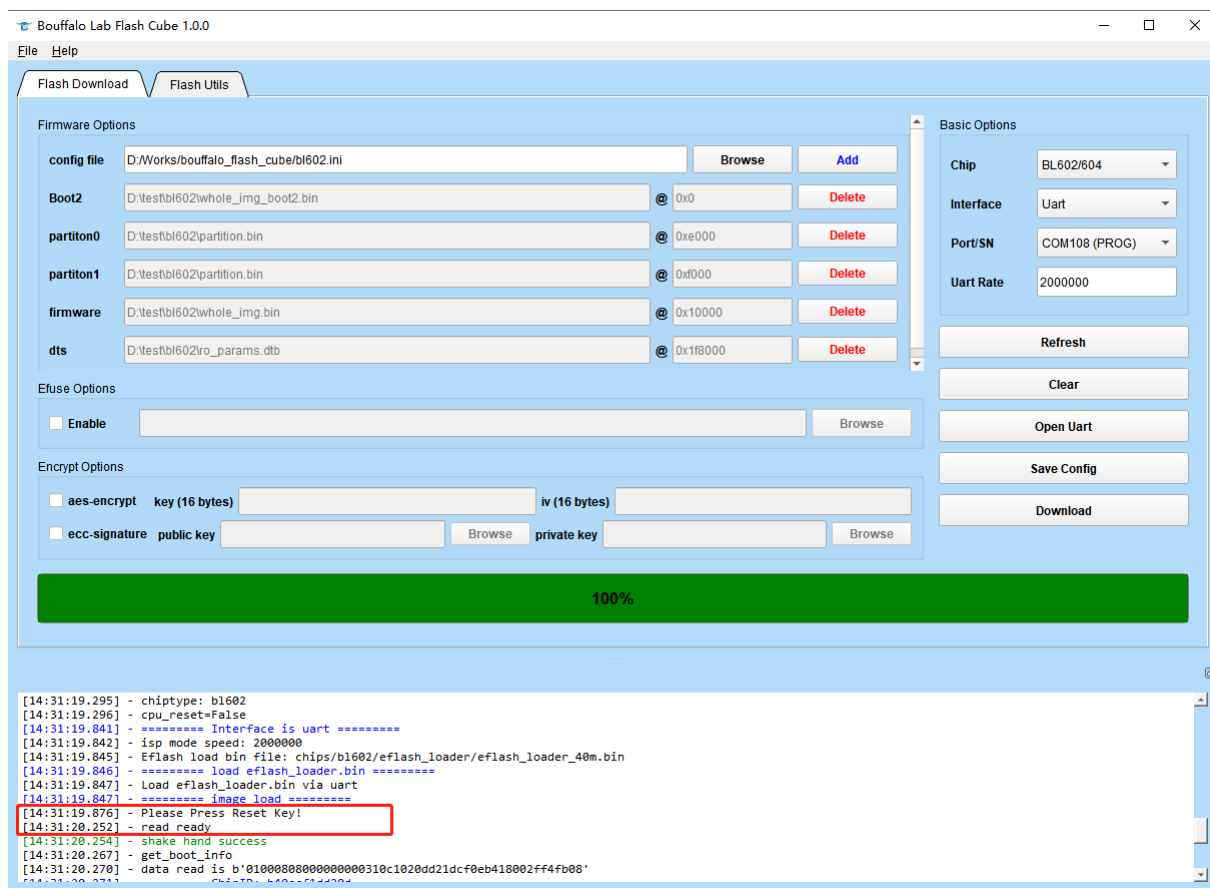


图 7.2: ISP 烧写模式

## 7.3 支持压缩烧写

压缩烧写模式下，工具会对烧写的每个文件进行压缩。通过串口传输到芯片时，芯片进行解压操作并将解压后的文件烧写到 Flash 中，压缩烧写可以极大的提升烧写速度。其中 BL702 不支持压缩烧写方式。

以 BL602 为例，打开工具目录下的“chips/bl602/eflash\_loader/eflash\_loader\_cfg.ini”文件，修改其中的“decompress\_write = false”为“decompress\_write = true”，然后保存文件。在烧写的时候会出现如下图所示的 log，即成功使用了压缩烧写方式。

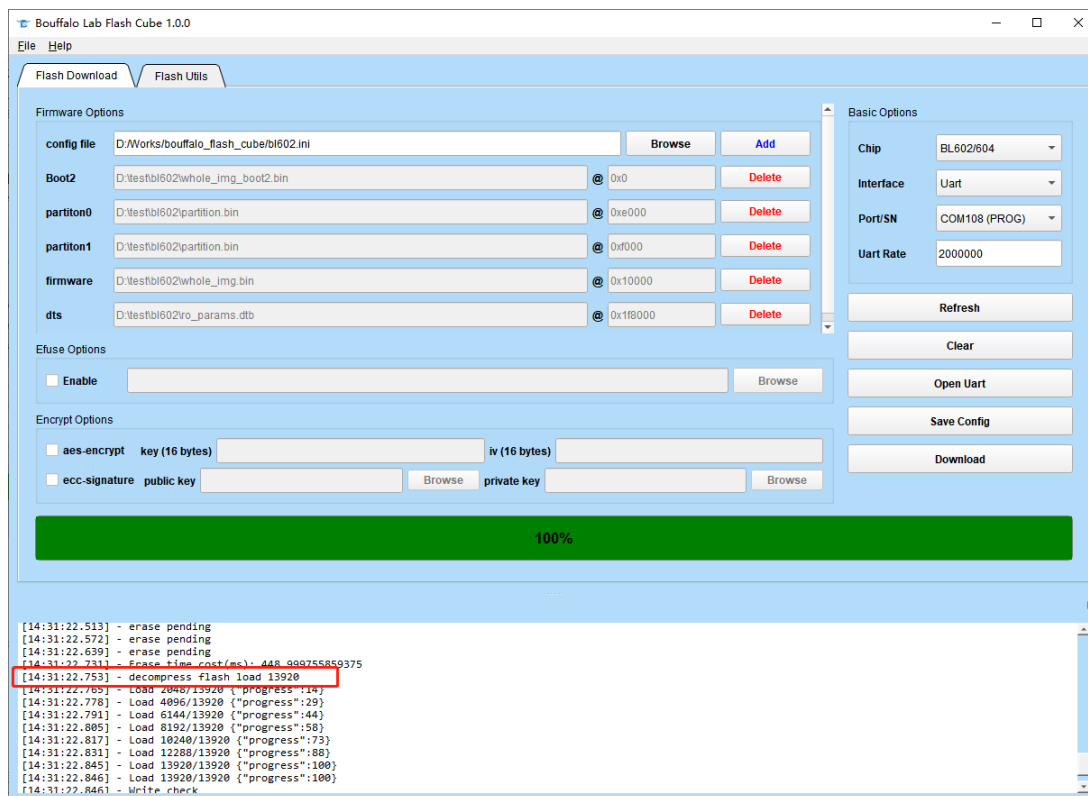


图 7.3: 压缩烧写模式

## 7.4 支持 eFuse 校验选择

Flash Cube 工具支持 eFuse 烧写，在 SDK 编译完成的“build/build\_out”目录下会生成 efusedata.bin 和 efusedata\_mask.bin。其中 efusedata.bin 是 eFuse 烧写时选择的 bin 文件，efusedata\_mask.bin 用于 eFuse 的校验。

是否做 eFuse 校验可配，通过对应芯片类型下 eflash\_loader\_cfg.conf 文件中的 factory\_mode 参数修改（以 BL602 为例，则文件路径为 chips/bl602/eflash\_loader/eflash\_loader\_cfg.ini）。

默认 factory\_mode 为 false，表示不进行 eFuse 校验。当修改 factory\_mode = true 的时候，表示进行 eFuse 校验。

以芯片重复加密加签为例，当芯片再次烧录时，由于加密的密钥或签名的 Hash 已经被烧录且被读写保护，如果 factory\_mode = true，则会显示 eFuse 烧写校验失败，而此时属于正常现象，但会给客户造成疑惑。

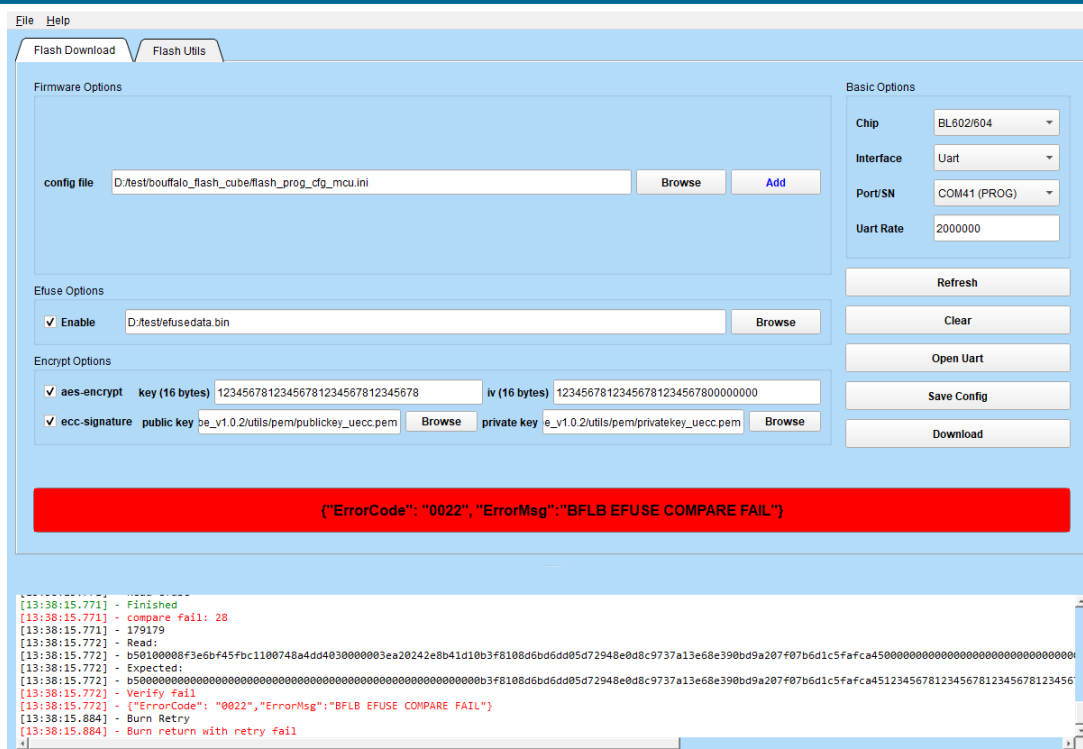


图 7.4: 重复加密加签烧写 eFuse 校验失败

如果修改 `factory_mode = false` 不进行校验烧写，则会直接显示烧写成功。

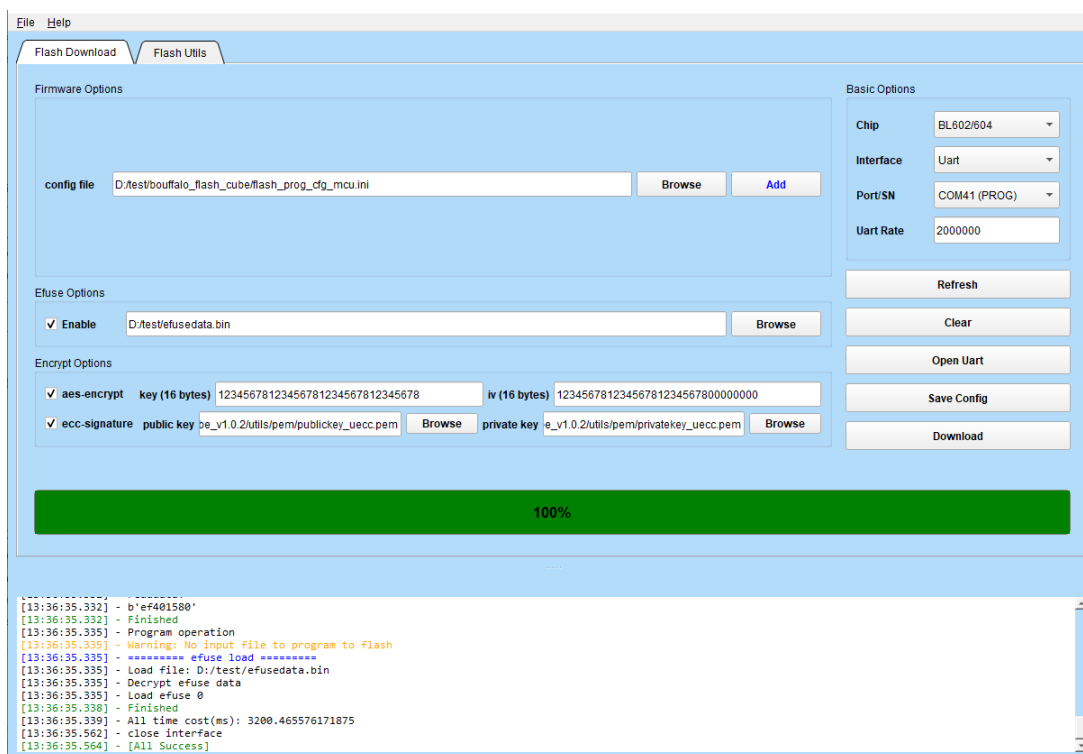


图 7.5: 重复加密加签烧写 eFuse 不校验

## 7.5 支持修改烧录时擦除方式

工具支持 Flash 全擦和分段擦除的方式，通过用户导入的配置文件（flash\_prog\_cfg.ini）中 erase 参数控制。

配置文件中 [cfg] 下的 erase 参数用于配置工具的擦除方式。当 erase = 0 时表示不进行擦除直接烧写，当 erase = 1 时表示下载时按照烧录地址和内容大小进行擦除，当 erase = 2 时表示程序烧录之前会将 Flash 全部擦除。工具中默认的烧写模式是 erase = 1 按照烧录地址和内容大小进行擦除，在烧写每个文件之前进行擦除操作。

```
[14:31:21.098] - ===== programming D:\test\bl602\whole_img_boot2.bin
[14:31:21.105] - ===== flash load =====
[14:31:21.134] - ===== flash erase =====
[14:31:21.135] - Erase flash from 0x0 to 0xb42f
[14:31:21.142] - erase pending
[14:31:21.439] - erase pending
[14:31:21.496] - erase pending
[14:31:21.550] - erase pending
[14:31:21.614] - erase pending
[14:31:21.710] - Erase time cost(ms): 574.00439453125
[14:31:21.745] - decompress flash load 24504
[14:31:21.995] - Load 24504/24504 {"progress":100}
[14:31:21.998] - Load 24504/24504 {"progress":100}
[14:31:21.998] - Write check
[14:31:22.013] - Flash load time cost(ms): 299.993896484375
[14:31:22.015] - Finished
[14:31:22.017] - Sha called by host: a8761e5a3e5a0884ae7f2f6bf2bc82601eac1bda49b5c302e59e562bae5afd6e
[14:31:22.020] - xip mode Verify
[14:31:22.034] - Read Sha256/46128
[14:31:22.035] - Flash xip readsha time cost(ms): 14.999267578125
[14:31:22.036] - Finished
[14:31:22.036] - Sha called by dev: a8761e5a3e5a0884ae7f2f6bf2bc82601eac1bda49b5c302e59e562bae5afd6e
[14:31:22.039] - Verify success
[14:31:22.042] - Dealing Index 1
[14:31:22.042] - ===== programming D:\test\bl602\partition.bin
[14:31:22.047] - ===== flash load =====
[14:31:22.048] - ===== flash erase =====
[14:31:22.049] - Erase flash from 0xe000 to 0xe10f
[14:31:22.052] - erase pending
[14:31:22.129] - Erase time cost(ms): 79.999267578125
[14:31:22.133] - Load 272/272 {"progress":100}
[14:31:22.134] - Load 272/272 {"progress":100}
[14:31:22.134] - Write check
[14:31:22.141] - Flash load time cost(ms): 9.996826171875
[14:31:22.142] - Finished
[14:31:22.143] - Sha called by host: fd6af18fc4aaf2807277cac767ca19d12af7b55f5ecbb8902ef28bc2430524aa
[14:31:22.143] - xip mode Verify
[14:31:22.146] - Read Sha256/272
[14:31:22.146] - Flash xip readsha time cost(ms): 1.000244140625
[14:31:22.146] - Finished
[14:31:22.147] - Sha called by dev: fd6af18fc4aaf2807277cac767ca19d12af7b55f5ecbb8902ef28bc2430524aa
[14:31:22.147] - Verify success
[14:31:22.148] - Dealing Index 2
[14:31:22.148] - ===== programming D:\test\bl602\partition.bin
[14:31:22.157] - ===== flash load =====
[14:31:22.158] - ===== flash erase =====
[14:31:22.158] - Erase flash from 0xf000 to 0xf10f
[14:31:22.160] - erase pending
[14:31:22.253] - Erase time cost(ms): 94.0048828125
[14:31:22.257] - Load 272/272 {"progress":100}
[14:31:22.258] - Load 272/272 {"progress":100}
[14:31:22.258] - Write check
[14:31:22.259] - Flash load time cost(ms): 4.00390625
[14:31:22.259] - Finished
[14:31:22.263] - Sha called by host: fd6af18fc4aaf2807277cac767ca19d12af7b55f5ecbb8902ef28bc2430524aa
[14:31:22.264] - xip mode Verify
```

图 7.6: 按照烧录地址和内容大小擦除

当修改烧写模式为 erase = 2 时，工具在烧录前会将 Flash 全部擦除。

```
[14:42:57.971] - ===== flash read jedec ID =====
[14:42:57.973] - Read flash jedec ID
[14:42:57.973] - readdata:
[14:42:57.973] - b'ef401580'
[14:42:57.973] - Finished
[14:42:57.975] - Program operation
[14:42:57.975] - Flash Chip Erase All
[14:42:58.986] - erase pending
[14:42:59.995] - erase pending
[14:43:01.003] - erase pending
[14:43:02.012] - erase pending
[14:43:03.021] - erase pending
[14:43:03.506] - Chip erase time cost(ms): 5531.059326171875
[14:43:03.508] - Dealing Index 0
[14:43:03.508] - ===== programming D:\test\bl602\whole_img_boot2.bin to 0x0
[14:43:03.511] - ===== flash load =====
[14:43:03.527] - decompress flash load 24504
[14:43:03.699] - Load 24504/24504 {"progress":100}
[14:43:03.699] - Load 24504/24504 {"progress":100}
[14:43:03.699] - Write check
[14:43:03.717] - Flash load time cost(ms): 205.750244140625
[14:43:03.717] - Finished
[14:43:03.718] - Sha caled by host: a8761e5a3e5a0884ae7f2f6bf2bc82601eac1bda49b5c302e59e562bae5afd6e
[14:43:03.718] - xip mode Verify
[14:43:03.732] - Read Sha256/46128
[14:43:03.733] - Flash xip readsha time cost(ms): 14.01171875
[14:43:03.733] - Finished
[14:43:03.734] - Sha caled by dev: a8761e5a3e5a0884ae7f2f6bf2bc82601eac1bda49b5c302e59e562bae5afd6e
[14:43:03.734] - Verify success
[14:43:03.738] - Dealing Index 1
[14:43:03.738] - ===== programming D:\test\bl602\partition.bin to 0xe000
[14:43:03.741] - ===== flash load =====
[14:43:03.743] - Load 272/272 {"progress":100}
[14:43:03.743] - Load 272/272 {"progress":100}
[14:43:03.743] - Write check
[14:43:03.745] - Flash load time cost(ms): 3.998779296875
[14:43:03.745] - Finished
[14:43:03.745] - Sha caled by host: fd6af18fc4aaf2807277cac767ca19d12af7b55f5ecbb8902ef28bc2430524aa
[14:43:03.746] - xip mode Verify
[14:43:03.747] - Read Sha256/272
[14:43:03.747] - Flash xip readsha time cost(ms): 1.006103515625
[14:43:03.748] - Finished
[14:43:03.752] - Sha caled by dev: fd6af18fc4aaf2807277cac767ca19d12af7b55f5ecbb8902ef28bc2430524aa
[14:43:03.753] - Verify success
[14:43:03.754] - Dealing Index 2
[14:43:03.754] - ===== programming D:\test\bl602\partition.bin to 0xf000
[14:43:03.757] - ===== flash load =====
[14:43:03.758] - Load 272/272 {"progress":100}
[14:43:03.759] - Load 272/272 {"progress":100}
[14:43:03.759] - Write check
[14:43:03.760] - Flash load time cost(ms): 3.360107421875
[14:43:03.761] - Finished
[14:43:03.761] - Sha caled by host: fd6af18fc4aaf2807277cac767ca19d12af7b55f5ecbb8902ef28bc2430524aa
[14:43:03.761] - xip mode Verify
```

图 7.7: 烧写前全擦除

## 7.6 支持擦写的 skip 功能

当 flash 烧写时不希望指定区域被擦除或者写入时，通过 skip 功能可以跳过此区域进行烧写。配置文件（flash\_prog\_cfg.ini）中 [cfg] 下的 skip 参数用于设置工具擦写的 skip 功能。以 BL602 为例，烧写过程中不希望 0x11000 ~ 0x12000 地址内容被改变，可以通过修改 skip\_mode 的值来实现，第一个参数为起始地址，第二个参数为长度。

操作步骤:

首先打开用户自定义的配置文件 flash\_prog\_cfg.ini，修改其中的“skip\_mode = 0x0, 0x0”为“skip\_mode = 0x11000, 0x1000”，然后保存文件。点击 Download 按钮之后的烧录 log 如下图所示：



```
[16:06:14.252] - ===== programming D:\test\bl602\whole_img.bin to 0x10000
[16:06:14.255] - skip flash file, skip addr 0x00011000, skip len 0x00001000
[16:06:14.257] - ===== flash load =====
[16:06:14.257] - ===== flash erase =====
[16:06:14.257] - Erase flash from 0x10000 to 0x10fff
[16:06:14.259] - erase pending
[16:06:14.337] - Erase time cost(ms): 79.93701171875
[16:06:14.348] - Load 2048/4096 {"progress":50}
[16:06:14.360] - Load 4096/4096 {"progress":100}
[16:06:14.361] - Load 4096/4096 {"progress":100}
[16:06:14.361] - Write check
[16:06:14.363] - Flash load time cost(ms): 25.01123046875
[16:06:14.364] - Finished
[16:06:14.365] - Sha caled by host: c1f100500c5a07ceb87c3379f8a74a48c115c2c5dd454162471e1417681f5a56
[16:06:14.365] - xip mode Verify
[16:06:14.367] - Read Sha256/4096
[16:06:14.367] - Flash xip readsha time cost(ms): 1.857177734375
[16:06:14.367] - Finished
[16:06:14.368] - Sha caled by dev: c1f100500c5a07ceb87c3379f8a74a48c115c2c5dd454162471e1417681f5a56
[16:06:14.368] - Verify success
[16:06:14.370] - ===== flash load =====
[16:06:14.371] - ===== flash erase =====
[16:06:14.371] - Erase flash from 0x12000 to 0x164cf
[16:06:14.376] - erase pending
[16:06:14.438] - erase pending
[16:06:14.491] - erase pending
[16:06:14.546] - erase pending
[16:06:14.617] - erase pending
[16:06:14.710] - Erase time cost(ms): 339.65283203125
[16:06:14.718] - decompress flash load 11124
[16:06:14.730] - Load 2048/11124 {"progress":18}
[16:06:14.741] - Load 4096/11124 {"progress":36}
[16:06:14.755] - Load 6144/11124 {"progress":55}
[16:06:14.770] - Load 8192/11124 {"progress":73}
[16:06:14.782] - Load 10240/11124 {"progress":92}
[16:06:14.797] - Load 11124/11124 {"progress":100}
[16:06:14.798] - Load 11124/11124 {"progress":100}
[16:06:14.798] - Write check
[16:06:14.811] - Flash load time cost(ms): 99.43994140625
[16:06:14.812] - Finished
```

图 7.8: IOT 页面的 skip 功能

skip\_mode 支持同时配置多个区域，中间以“;”分隔。

以 BL602 为例，烧写过程中不希望 0x11000 ~ 0x12000, 0x13000 ~ 0x15000 地址内容被改变，则需要修改配置文件 flash\_prog\_cfg.ini 中 skip\_mode 的值为“skip\_mode = 0x11000, 0x1000; 0x13000, 0x2000”，然后保存文件。

```
[16:00:20.419] - ===== programming D:\test\bl602\whole_img.bin to 0x10000
[16:00:20.423] - skip flash file, skip addr 0x00011000, skip len 0x00001000
[16:00:20.433] - ===== flash load =====
[16:00:20.434] - ===== flash erase =====
[16:00:20.434] - Erase flash from 0x10000 to 0x10fff
[16:00:20.435] - erase pending
[16:00:20.519] - Erase time cost(ms): 84.828369140625
[16:00:20.531] - Load 2048/4096 {"progress":50}
[16:00:20.542] - Load 4096/4096 {"progress":100}
[16:00:20.543] - Load 4096/4096 {"progress":100}
[16:00:20.543] - Write check
[16:00:20.544] - Flash load time cost(ms): 23.99951171875
[16:00:20.545] - Finished
[16:00:20.546] - Sha caled by host: c1f100500c5a07ceb87c3379f8a74a48c115c2c5dd454162471e1417681f5a56
[16:00:20.546] - xip mode Verify
[16:00:20.548] - Read Sha256/4096
[16:00:20.549] - Flash xip readsha time cost(ms): 2.00048828125
[16:00:20.549] - Finished
[16:00:20.549] - Sha caled by dev: c1f100500c5a07ceb87c3379f8a74a48c115c2c5dd454162471e1417681f5a56
[16:00:20.550] - Verify success
[16:00:20.551] - skip flash file, skip addr 0x00013000, skip len 0x00002000
[16:00:20.552] - ===== flash load =====
[16:00:20.552] - ===== flash erase =====
[16:00:20.553] - Erase flash from 0x12000 to 0x12fff
[16:00:20.554] - erase pending
[16:00:20.646] - Erase time cost(ms): 92.510498046875
[16:00:20.658] - Load 2048/4096 {"progress":50}
[16:00:20.670] - Load 4096/4096 {"progress":100}
[16:00:20.671] - Load 4096/4096 {"progress":100}
[16:00:20.671] - Write check
[16:00:20.675] - Flash load time cost(ms): 28.147216796875
[16:00:20.675] - Finished
[16:00:20.675] - Sha caled by host: 0232b58065e8de52132e944a41101b49094b642132294658c773a395b047a177
[16:00:20.676] - xip mode Verify
[16:00:20.680] - Read Sha256/4096
[16:00:20.680] - Flash xip readsha time cost(ms): 3.9560546875
[16:00:20.680] - Finished
[16:00:20.680] - Sha caled by dev: 0232b58065e8de52132e944a41101b49094b642132294658c773a395b047a177
[16:00:20.680] - Verify success
[16:00:20.682] - ===== flash load =====
[16:00:20.682] - ===== flash erase =====
[16:00:20.682] - Erase flash from 0x15000 to 0x164cf
[16:00:20.683] - erase pending
[16:00:20.753] - erase pending
[16:00:20.848] - Erase time cost(ms): 165.797607421875
[16:00:20.855] - decompress flash load 2848
[16:00:20.865] - Load 2048/2848 {"progress":71}
[16:00:20.872] - Load 2848/2848 {"progress":100}
[16:00:20.873] - Load 2848/2848 {"progress":100}
[16:00:20.873] - Write check
[16:00:20.885] - Flash load time cost(ms): 34.90966796875
[16:00:20.886] - Finished
```

图 7.9: IOT 页面的 skip 功能

从烧写 log 中可以看到，烧写过程中会跳过 0x11000 ~ 0x12000, 0x13000 ~ 0x15000 区域，对其他区域内容单独做擦写操作。

## 7.7 生成量产烧录文件

在每次烧写时，工具会生成两个文件用于量产烧录：

- whole\_flash\_data.bin
- whole\_img.pack

1. whole\_flash\_data.bin 文件是二进制文件，按照分区表，排布了所有要烧录的镜像相关文件，

其内容和 Flash 中数据布局完全一致，whole\_flash\_data.bin 的构成如图所示：

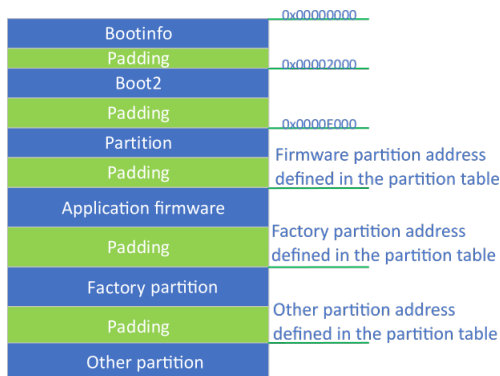


图 7.10: whole\_flash\_data.bin 构成

从图中可以看出，whole\_flash\_data.bin 包含了所有要烧录的 bin 文件，并且已经按照分区表位置排放好。该文件可以直接使用 Flash 编程器或者批量烧写工具的单文件模式烧录到 Flash 的 0 地址起始位置。

此文件包含了不同固件之间的 padding，导致文件较大，其缺点就是烧录时间长。

2. whole\_img.pack 是一个压缩文件，它不仅包含了各种要烧录的文件，还包含了要烧录文件的配置信息，压缩包构成如下图所示：

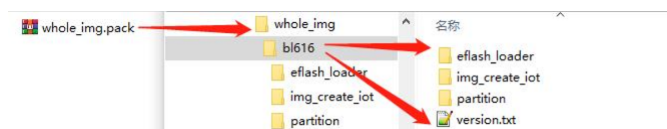


图 7.11: whole\_img.pack 构成

在批量生产的时候，可以从量产烧录工具界面导入该开发包，量产工具会自行完成解压，并根据配置文件，分别烧录要烧写的文件，烧写速度快。

生成的 whole\_flash\_data.bin 和 whole\_img.pack 存放于“chips/bl616/img\_create”目录下。用户可以根据自身情况，选择量产文件。



> Works > tools > FlashCube > v1.0.4 > chips > bl616 > img_create			
名称	修改日期	类型	大小
 whole_flash_data.bin	2023/3/17 15:46	BIN 文件	2,514 KB
 whole_img.pack	2023/3/17 15:46	PACK 文件	1,260 KB

图 7.12: 生成的 Whole\_img 镜像

## 7.8 BL602/BL702 支持不填写密钥签名烧写已经加密加签的板子

Flash Cube v1.0.8 及其之后的版本，用户可以重复烧写已经加密加签的板子，而不用提供密钥和签名，只需要提前准备好 eflash\_loader\_xx\_encrypt.bin，放到工具的 eflash\_loader 目录下即可。

以 BL602 为例，首先将加密后的 eflash\_loader\_40m\_encrypt.bin 拷贝到 chips/bl602/eflash\_loader 目录下：













> v1.0.8 > chips > bl602 > eflash_loader		
名称	修改日期	类型
 eflash_loader.elf	2023/6/16 10:36	ELF 文件
 eflash_loader.map	2023/6/16 10:36	MAP 文件
 eflash_loader_24m.bin	2023/6/16 10:36	BIN 文件
 eflash_loader_26m.bin	2023/6/16 10:36	BIN 文件
 eflash_loader_32m.bin	2023/6/16 10:36	BIN 文件
 eflash_loader_38p4m.bin	2023/6/16 10:36	BIN 文件
 eflash_loader_40m.bin	2023/6/16 10:36	BIN 文件
 eflash_loader_40m_encrypt.bin	2023/9/8 15:12	BIN 文件
 eflash_loader_cfg.conf	2022/10/17 9:12	CONF 文件
 eflash_loader_cfg.ini	2023/11/22 16:41	INI 文件
 eflash_loader_none.bin	2023/6/16 10:36	BIN 文件
 eflash_loader_rc32m.bin	2023/6/16 10:36	BIN 文件

图 7.13: 加密 eflash\_loader\_encrypt.bin 文件

打开 Flash Cube 的烧写界面，按照下面的烧录方式不填写密钥和签名，可以成功烧写已经加密加签的板子。

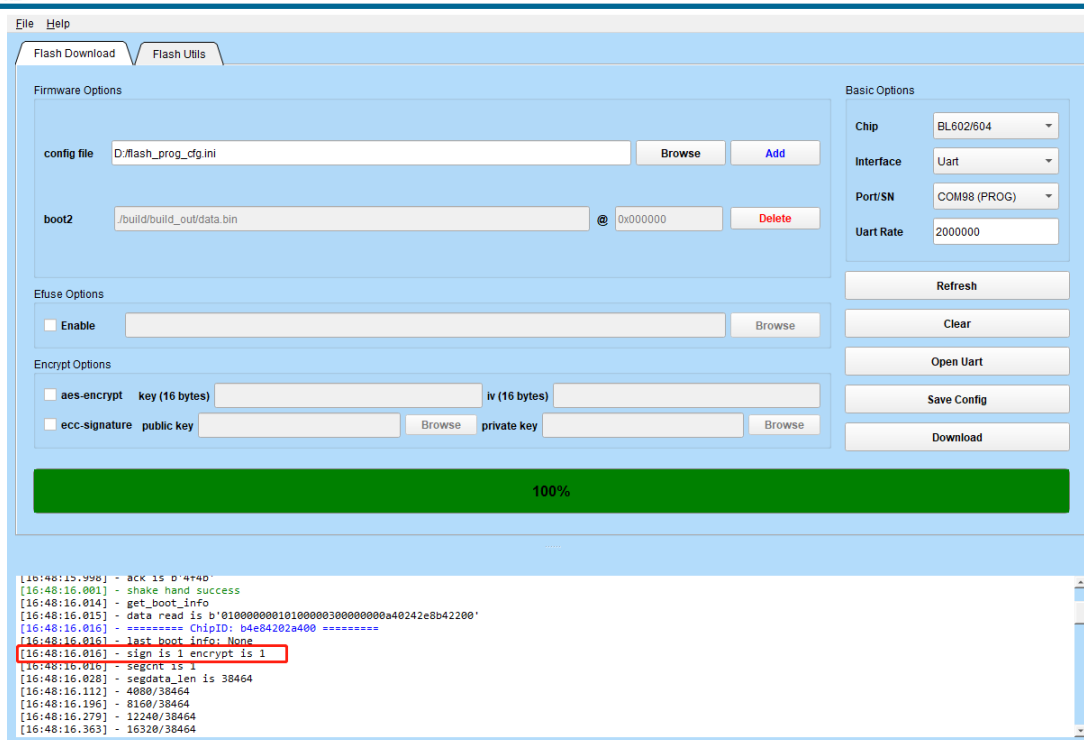


图 7.14: 烧写已经加密加签的板子

如果提供的eflash\_loader\_xx\_encrypt.bin 密钥和签名不匹配,则会提示错误:”BFLB\_IMG\_SECTIONHEADER\_CRC\_ERROR”。

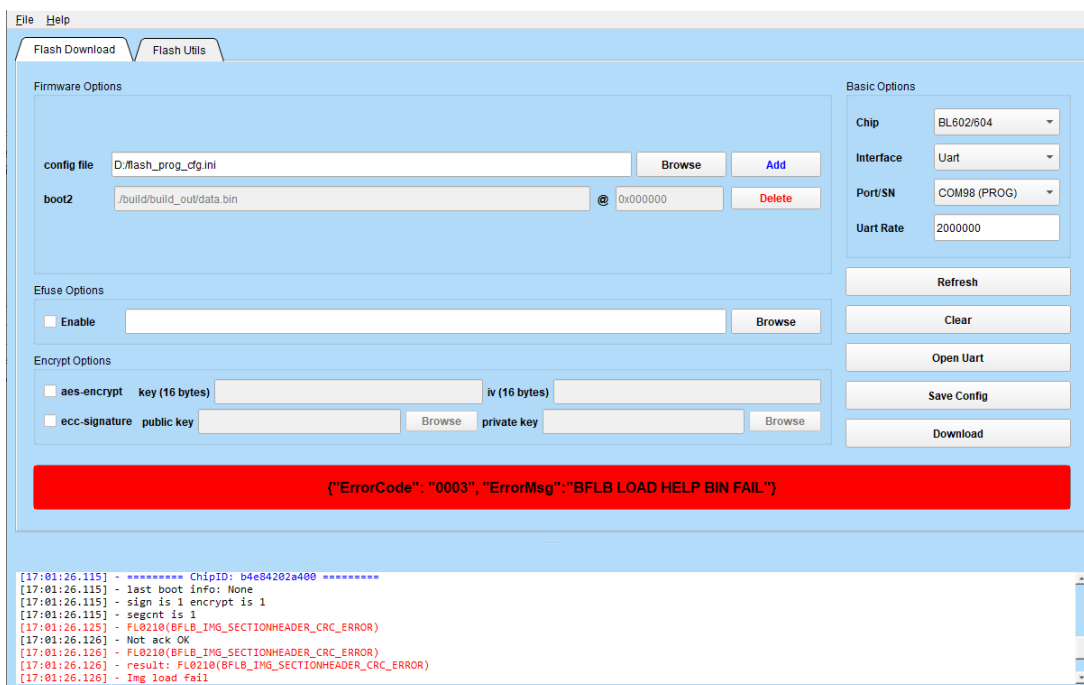


图 7.15: 密钥不匹配烧录已经加密加签的板子

## 7.9 新增预处理功能

若需要 Flash Cube 在执行烧写或者 build 量产固件之前先执行一个预处理程序，则可在自定义的烧录配置文件中新增 pre\_program, pre\_program\_args 这两个参数。

- pre\_program 填写路径：以用户自定义的烧录配置文件为相对路径的预处理程序所在路径。
- pre\_program\_args 填写参数：多个参数中间用空格分割；若传入参数为路径，支持模糊匹配（如：“./build\_out/wifi\_ota\*\_\$(CHIPNAME).bin” 的方式，由工具去匹配需要烧写的测试固件）。

注解：pre\_program 配置项自适应平台，故无需增加后缀（如：.exe）

windows 适配以.exe 结尾的可执行程序

Linux 适配以-ubuntu 结尾的可执行程序

Darwin 适配以-macos 结尾的可执行程序

以预处理程序是 bflb\_fw\_post\_proc 可执行程序为例，操作步骤如下：

在 flash\_prog\_cfg.ini 配置文件中添加 pre\_program 和 pre\_program\_args（因 bflb\_fw\_post\_proc.exe 与 flash\_prog\_cfg.ini 在同一层目录，所以 pre\_program=./bflb\_fw\_post\_proc）。

```
[cfg]
# 0: no erase, 1:programmed section erase, 2: chip erase
erase = 1
# skip mode set first para is skip addr, second para is skip len, multi-segment region with ; separated
skip_mode = 0x0, 0x0
# 0: not use isp mode, #1: isp mode
boot2_isp_mode = 0
pre_program = ./bflb_fw_post_proc
pre_program_args = --chipname=$(CHIPNAME) --imgfile=./build_out/wifi_ota*_$(CHIPNAME).bin
```

点击工具页面中的 Download，如下图所示，烧录过程中会先执行 bflb\_fw\_post\_proc 生成普通镜像，后续会继续按 flash\_prog\_cfg.ini 的配置项烧录。

```
# 实际执行的预处理命令：
bflb_fw_post_proc.exe --chipname=bl616 --imgfile=./build out/wifi_ota_b1616.bin
```

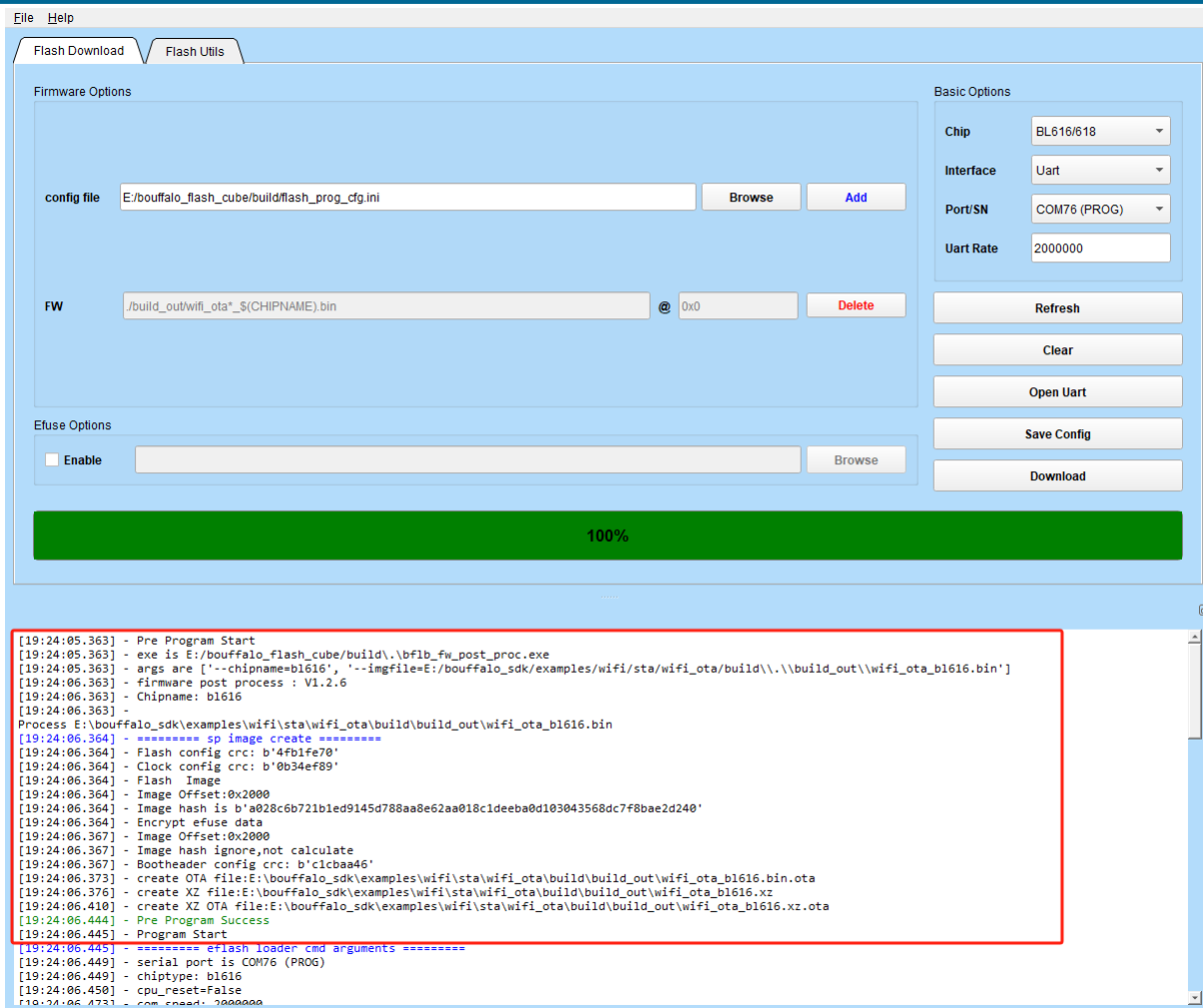


图 7.16: 烧录前调用 pre\_program

## 7.10 支持用户自定义的 efusedata.bin 的加密密钥

工具提供更高的安全性，可以自定义 efusedata.bin 的加密密钥，该功能的使用需要借助 bflb\_img\_encryption\_tool.exe 工具，使用方法：传入用户自定义的加密密钥，使用 AES 加密方式生成 cfg.bin 和加密后的 efusedata.bin。

当 FlashCube 工具的根目录存在 cfg.bin 时，烧录用户自定义加密后的 efusedata.bin 才可以烧录成功，否则烧录失败并提示错误：“Efuse crc check fail”

## 8.1 自定义的功能配置以用户导入为准

用户导入的烧录配置文件 (flash\_prog\_cfg.ini) 中包含多种功能配置, 如 erase, skip\_mode, boot2\_isp\_mode 等功能。这些功能在对应芯片类型的目录 (eflash\_loader/eflash\_loader\_cfg.conf) 中也有相应的配置。

其中 erase, skip\_mode, boot2\_isp\_mode 功能以用户导入的烧录配置文件中的定义为准, 并且在烧写时会将这些配置更新到 eflash\_loader\_cfg.conf 文件中, 生成的 whole\_img.pack 中使用的也是更新后的 eflash\_loader\_cfg.conf 文件。

## 8.2 烧录界面每个烧录选项名称最大支持 10 个字符

分区表中每个分区的 name 字段长度不能超过 10 个字符

## 8.3 晶振类型默认设定

BLFlashCube 工具的晶振类型无法修改, 当前是设置的默认值。其中 BL602 为 40M, BL702 为 32M, BL808/BL606P/BL616 为 auto 自动获取晶振类型。

## 8.4 固件超出分配的地址大小时会提示错误

工具会检测用户填写的烧录地址和烧录文件的大小, 当地址重复或者烧录的固件超出了分配的地址时会提示错误。

以 BL602 烧写为例, 如果按下图方式配置烧写地址和烧写文件, 因 firmware 的地址位置烧写 whole\_img.bin 会超出 1 个字节, 工具提示错误: Error: The file size exceeds the address space size!。

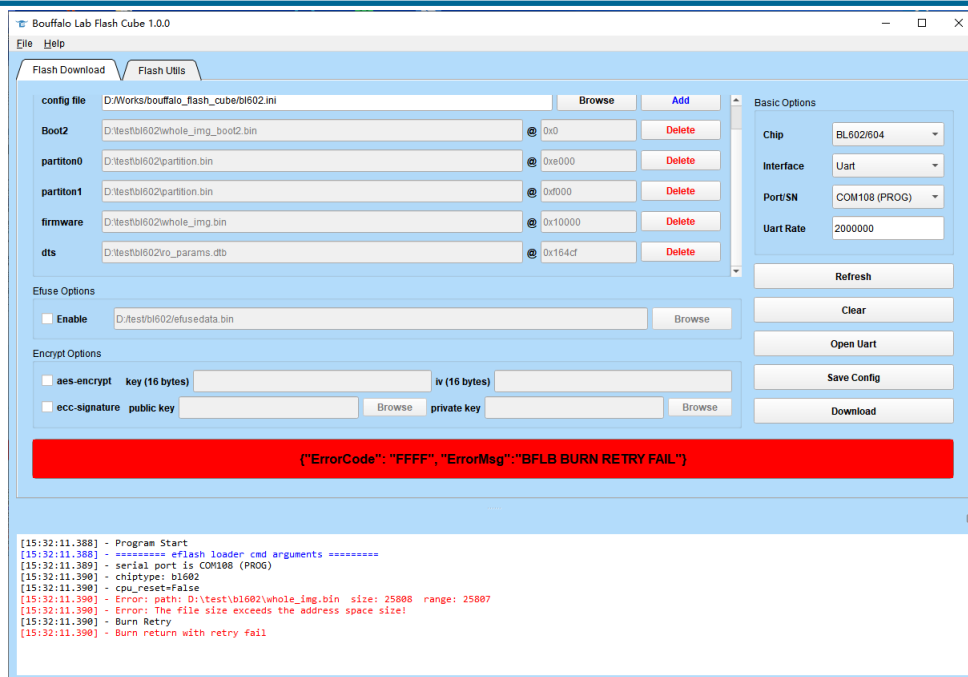


图 8.1: 固件大小超出错误

## 8.5 固件 Flash size 时会提示错误

Flash Cube v1.0.6 及其之后的版本，增加了烧录固件超出 flash size 的检查。如下图所示，如果烧录的程序为 2.1MB，实际的 flash 大小仅为 2M，则工具会提示错误: "ErrorMsg": "QCC74x FLASH SIZE OVER FLOW"。

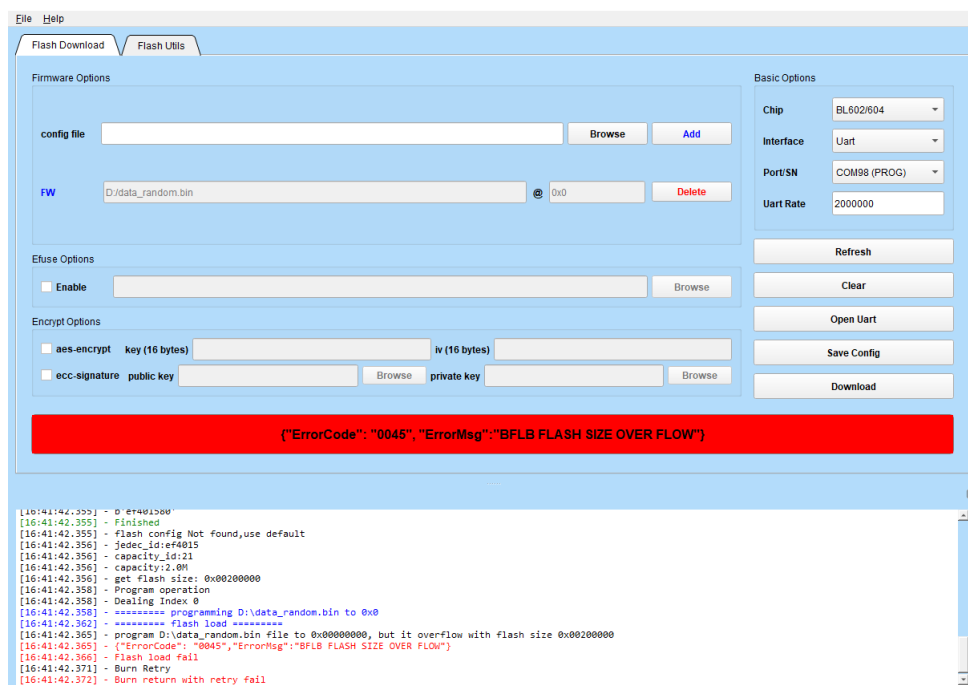


图 8.2: 烧录固件超出 flash 大小



表 9.1: 修改记录

版本	描述	日期
1.0	初版	2022-10-18
1.1	增加命令行工具使用说明	2022-12-28
1.2	更新命令行--firmware 参数和加密烧写说明	2023-11-22
1.3	增加 eFuse 读写操作	2024-3-8
1.4	更新 ram reset efuse_encrypted 命令行使用说明	2024-10-28
1.5	更新 Flash Otp 命令行使用说明	2024-11-8