



Flash Cube

使用说明

Version: 1.1

Copyright @ 2022

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	8
3	烧录配置文件介绍	10
4	新增烧写项	12
4.1	修改分区表文件	12
4.2	修改烧录配置文件	13
4.3	烧写程序	14
5	命令行下载	15
6	Flash 调试助手	20
6.1	配置通信方式	21
6.2	读 Flash 内容	21
6.3	擦除 Flash 内容	22
6.4	读写寄存器内容	23
7	高级功能	24
7.1	支持固件路径模糊匹配	24
7.2	支持 ISP 烧写模式	25
7.3	支持压缩烧写	26
7.4	支持 eFuse 校验选择	27
7.5	支持修改烧录时擦除方式	29
7.6	支持擦写的 skip 功能	30
7.7	生成量产烧录文件	32

8	注意事项	34
8.1	自定义的功能配置以用户导入为准	34
8.2	烧录界面每个烧录选项名称最大支持 10 个字符	34
8.3	晶振类型默认设定	34
8.4	固件超出分配的地址大小时会提示错误	34
9	修改记录	36

Flash Cube 是博流提供的芯片烧写工具，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。

Flash Cube 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。

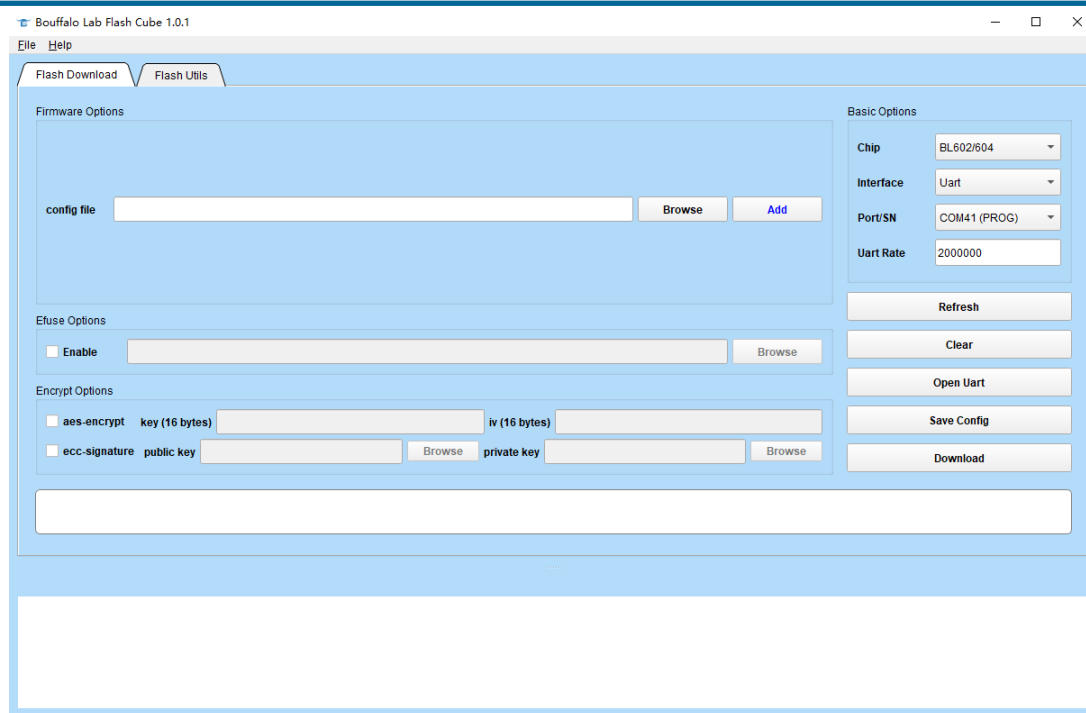


图 1.1: 烧写主界面

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** 区域用于 **BL602 / BL702** 加密或签名模式下的烧写使用
 - **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 bouffalo sdk 的 `examples/wifi/sta` 为例，介绍如何烧录。

2.1 编译应用固件

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

编译完成后在 `build/build_out` 路径下，生成了对应的固件 `sta_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 Flash Cube 后默认进入到 Flash Download 页面，点击 Firmware Options 区域 Browse 按钮选择 `examples/wifi/sta` 目录下的 `flash_prog_cfg.ini` 文件，更新后的界面如下图所示：

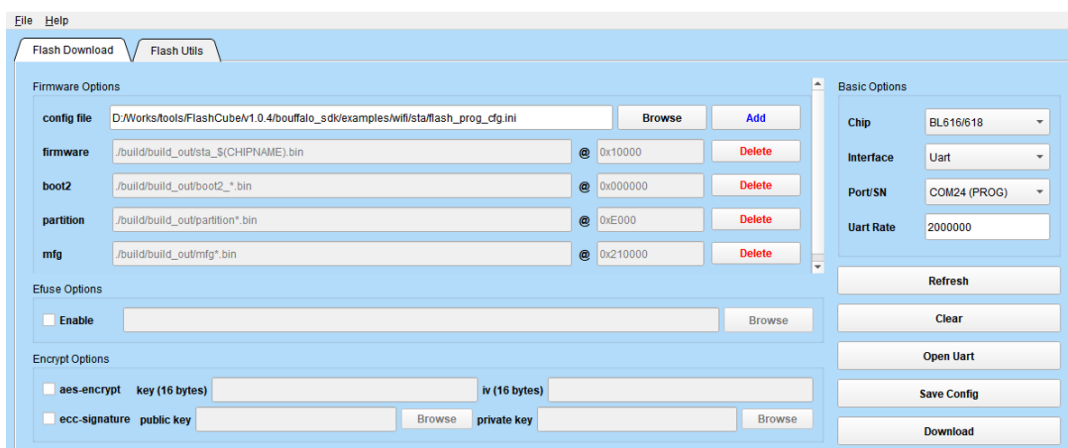


图 2.1: 导入配置文件

2.3 烧写程序

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。当出现如图所示 **100%** 的绿色进度条时，则表示程序下载成功。

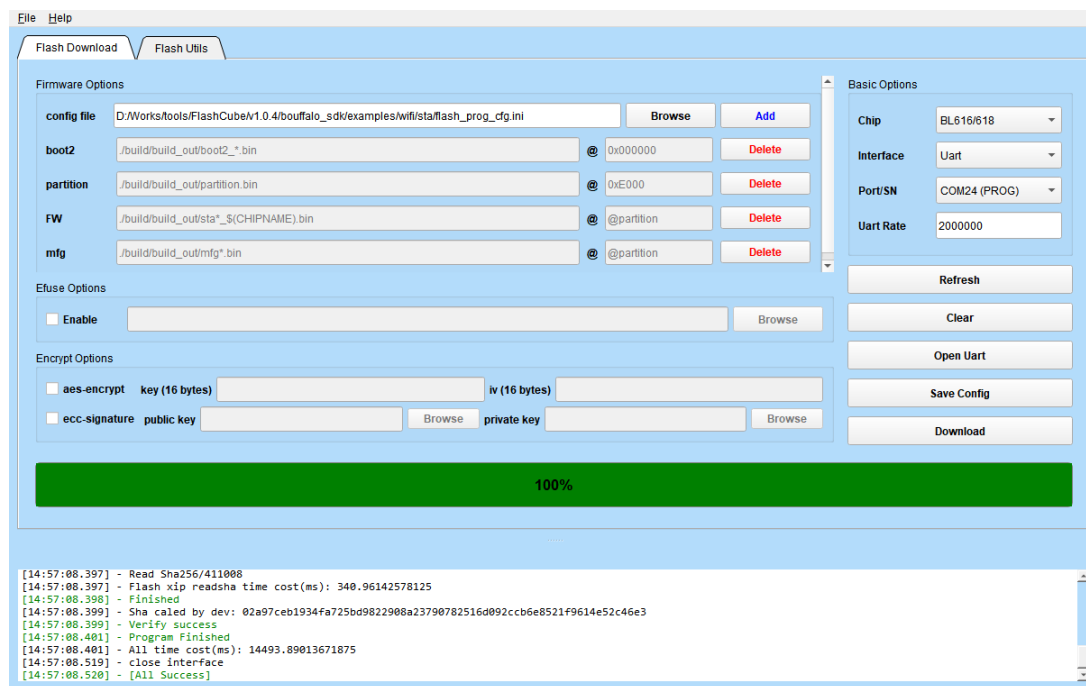


图 2.2: 成功下载程序

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** 程序运行起来的效果。

图 2.3: wifi sta 程序结果

烧录配置文件介绍

根据 `examples/wifi/sta` 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/sta*_$(CHIPNAME).bin
address = @partition

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

cfg 表示烧录时的一些配置，正常不需要改动

- **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 “sta” 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 中检测获取地址。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**: “@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，增加对应文件的分区信息
- 在 `flash_prog_cfg.ini` 文件中增加对应的烧写项配置

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，`0` 表示 CPU0 启动的镜像，`1` 表示 CPU1 启动的镜像，`Boot2` 是根据这个 `type` 来启动镜像的，因此，客户在自定义分区表时候，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`。

如果提示下面的 log 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：“[Warning] No partiton file found in ./../bsp/board/bl616dk/config,go on next steps”

4.2 修改烧录配置文件

打开 `examples/wifi/sta` 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所示：

```
[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/sta*_${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，建议使用相对路径（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` 指定烧写地址，建议使用 `@partition` 从分区表中自动获取烧写地址，也可以直接指定烧录地址 `0x1FF000`，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 Flash Cube 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：

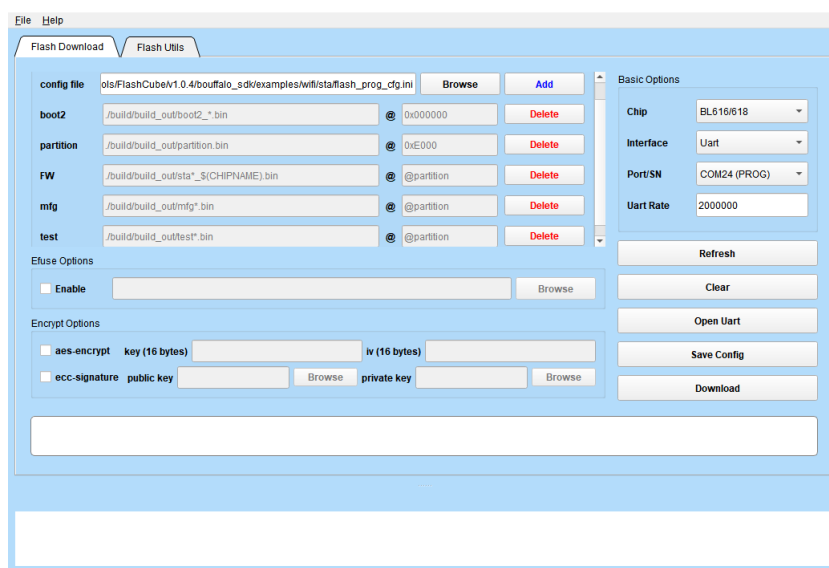


图 4.1: test 烧写项成功导入到界面

烧写成功的 log:

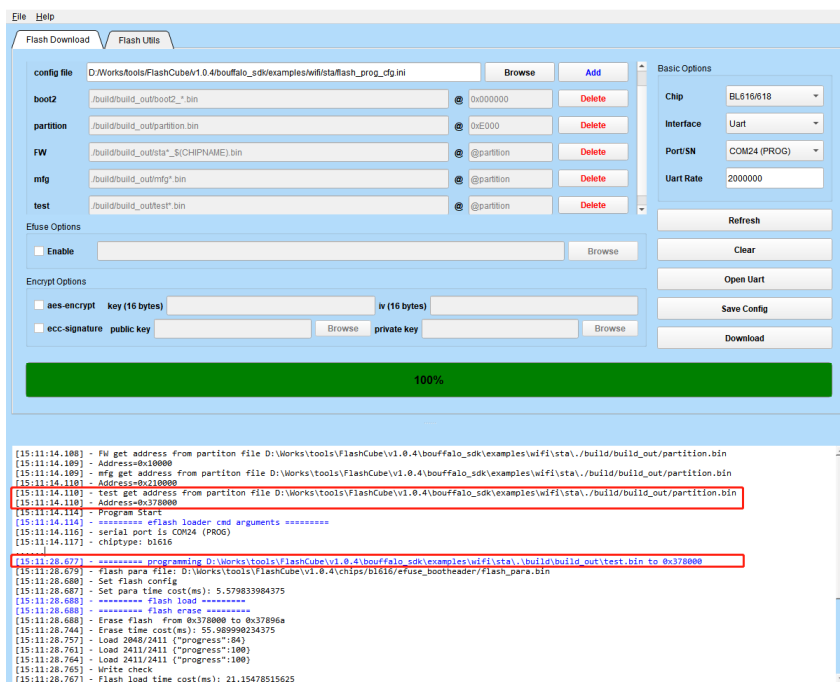


图 4.2: 成功下载程序

Flash Cube 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。

具体使用说明如下：

```
PS D:\Works\bouffalo_flash_cube> .\BLFlashCommand.exe --help
usage: BLFlashCommand.exe [-h] [--interface INTERFACE] [--port PORT] [--chipname CHIPNAME] [--baudrate BAUDRATE]
                        [--config CONFIG] [--cpu_id CPU_ID] [--efuse EFUSE] [--key KEY] [--iv IV] [--pk PK]
                        [--sk SK]

flash-command

optional arguments:
-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
--cpu_id CPU_ID       cpu id
--efuse EFUSE         efuse options
--key KEY             aes key
--iv IV              aes iv
--pk PK              ecc public key
--sk SK              ecc private key
```

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。

以 Windows 下烧写为例，使用 `cmd` 或 `pwsh` 进入到工具目录下，按照下面方式执行烧录命令：

```
PS D:\Works\bouffalo_flash_cube> .\BLFlashCommand.exe --interface=uart --chipname=bl616 --port=COM24 --
-baudrate=2000000 --config=bouffalo_sdk/examples/wifi/sta/flash_prog_cfg.ini
['D:\\Works\\tools\\FlashCube\\v1.0.4\\BLFlashCommand.exe', '--interface=uart', '--chipname=bl616', '--
-port=COM24', '--baudrate=2000000', '--config=bouffalo_sdk/examples/wifi/sta/flash_prog_cfg.ini']
[16:13:43.969] - Serial port is COM24
[16:13:43.970] - =====
[16:13:43.973] - FW get address from partiton file D:\Works\tools\FlashCube\v1.0.4\bouffalo_sdk\examples\
-wifi\sta\./build/build_out/partition.bin
[16:13:43.973] - Address=0x10000
[16:13:43.974] - mfg get address from partiton file D:\Works\tools\FlashCube\v1.0.4\bouffalo_sdk\examples\
-wifi\sta\./build/build_out/partition.bin
[16:13:43.974] - Address=0x210000
[16:13:43.976] - test get address from partiton file D:\Works\tools\FlashCube\v1.0.4\bouffalo_sdk\examples\
-wifi\sta\./build/build_out/partition.bin
[16:13:43.976] - Address=0x378000
[16:13:43.977] - Program Start
[16:13:43.977] - ===== eflash loader cmd arguments =====
[16:13:43.978] - serial port is COM24
[16:13:43.978] - chiptype: bl616
[16:13:43.978] - cpu_reset=False
[16:13:44.140] - ===== Interface is uart =====
[16:13:44.140] - Bootrom load
[16:13:44.140] - ===== get_boot_info =====
[16:13:44.141] - ===== image get bootinfo =====
[16:13:44.145] - default set DTR high
[16:13:44.258] - usb serial port
[16:13:44.319] - clean buf
[16:13:44.367] - send sync
[16:13:44.585] - ack is b'4f4b464c0201'
[16:13:44.630] - shake hand success
[16:13:45.143] - data read is b'01001606000001002516891034ddd52c720c19008f758018'
[16:13:45.143] - ===== ChipID: 0c722cd5dd34 =====
[16:13:45.146] - Get bootinfo time cost(ms): 1004.73828125
[16:13:45.146] - change bdrate: 2000000
[16:13:45.146] - Clock PLL set
[16:13:45.147] - Set clock time cost(ms): 1.01318359375
[16:13:45.281] - Read mac addr
[16:13:45.282] - flash set para
[16:13:45.282] - get flash pin cfg from bootinfo: 0x24
[16:13:45.282] - set flash cfg: 14124
[16:13:45.282] - Set flash config
[16:13:45.284] - Set para time cost(ms): 1.998046875
[16:13:45.284] - ===== flash read jedec ID =====
```

(continues on next page)

(continued from previous page)

```
[16:13:45.284] - Read flash jedec ID
[16:13:45.285] - readdata:
[16:13:45.285] - b'ef401700'
[16:13:45.285] - Finished
[16:13:45.287] - Program operation
[16:13:45.287] - Dealing Index 0
[16:13:45.287] - ===== programming D:\Works\tools\FlashCube\v1.0.4\buffalo_sdk\examples\wifi\sta\.\
->build\build_out\boot2_bl616_release_v8.0.7.bin to 0x000000
[16:13:45.289] - flash para file: D:\Works\tools\FlashCube\v1.0.4\chips/bl616/efuse_bootheader/flash_para.
->.bin
[16:13:45.290] - Set flash config
[16:13:45.296] - Set para time cost(ms): 6.062255859375
[16:13:45.298] - ===== flash load =====
[16:13:45.298] - ===== flash erase =====
[16:13:45.300] - Erase flash from 0x0 to 0xa93f
[16:13:45.656] - Erase time cost(ms): 355.3857421875
[16:13:45.670] - Load 43328/43328 {"progress":100}
[16:15:39.658] - Write check
[16:15:39.660] - Flash load time cost(ms): 250.399169921875
[16:15:39.660] - Finished
[16:15:39.661] - Sha calcd by host: 81bdd6bd9e028b2d1fa5da8d12aa4438353842d3f2a0b85e61a4efb00dd50fd0
[16:15:39.661] - xip mode Verify
[16:15:39.699] - Read Sha256/43328
[16:15:39.699] - Flash xip readsha time cost(ms): 37.889404296875
[16:15:39.702] - Finished
[16:15:39.704] - Sha calcd by dev: 81bdd6bd9e028b2d1fa5da8d12aa4438353842d3f2a0b85e61a4efb00dd50fd0
[16:15:39.704] - Verify success
[16:15:39.705] - Dealing Index 1
[16:15:39.793] - ===== programming D:\Works\tools\FlashCube\v1.0.4\buffalo_sdk\examples\wifi\sta\.\
->build\build_out\sta_bl616.bin to 0x10000
[16:15:39.797] - flash para file: D:\Works\tools\FlashCube\v1.0.4\chips/bl616/efuse_bootheader/flash_para.
->.bin
[16:15:39.797] - Set flash config
[16:15:39.803] - Set para time cost(ms): 5.73193359375
[16:15:39.803] - ===== flash load =====
[16:15:39.804] - ===== flash erase =====
[16:15:39.804] - Erase flash from 0x10000 to 0xda61f
[16:15:42.502] - Erase time cost(ms): 2697.24658203125
[16:15:47.222] - Load 828960/828960 {"progress":100}
[16:15:47.222] - Write check
[16:15:47.226] - Flash load time cost(ms): 4721.73046875
[16:15:47.226] - Finished
[16:15:47.230] - Sha calcd by host: 6167624bb39d78d164eada22e9802520fb2bea0b526a5563fc4ea6568d557747
```

(continues on next page)

```
[16:15:47.230] - xip mode Verify
[16:15:47.918] - Read Sha256/828960
[16:15:47.918] - Flash xip readsha time cost(ms): 686.973388671875
[16:15:47.920] - Finished
[16:15:47.922] - Sha caled by dev: 6167624bb39d78d164eada22e9802520fb2bea0b526a5563fc4ea6568d557747
[16:15:47.922] - Verify success
[16:15:47.925] - Dealing Index 3
[16:15:47.925] - ===== programming D:\Works\tools\FlashCube\v1.0.4\bouffalo_sdk\examples\wifi\sta\.\
-build\build_out\mfg_bl616_gu_af8b0946f_v2.26.bin to 0x210000
[16:15:47.927] - flash para file: D:\Works\tools\FlashCube\v1.0.4\chips/bl616/efuse_bootheader/flash_para.
-bin
[16:15:47.927] - Set flash config
[16:15:47.934] - Set para time cost(ms): 6.00146484375
[16:15:47.934] - ===== flash load =====
[16:15:47.935] - ===== flash erase =====
[16:15:47.936] - Erase flash from 0x210000 to 0x27457f
[16:15:49.320] - Erase time cost(ms): 1383.6416015625
[16:15:51.672] - Load 411008/411008 {"progress":100}
[16:15:51.672] - Write check
[16:15:51.676] - Flash load time cost(ms): 2355.2607421875
[16:15:51.676] - Finished
[16:15:51.678] - Sha caled by host: 02a97ceb1934fa725bd9822908a23790782516d092ccb6e8521f9614e52c46e3
[16:15:51.679] - xip mode Verify
[16:15:52.021] - Read Sha256/411008
[16:15:52.022] - Flash xip readsha time cost(ms): 342.109619140625
[16:15:52.023] - Finished
[16:15:52.025] - Sha caled by dev: 02a97ceb1934fa725bd9822908a23790782516d092ccb6e8521f9614e52c46e3
[16:15:52.025] - Verify success
[16:15:52.027] - Program Finished
[16:15:52.028] - All time cost(ms): 14265.697021484375
[16:15:52.147] - close interface
[16:15:52.147] - [All Success]
```

使用串口工具即可以看到启动 log:

图 5.1: 启动 log

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**、读取和擦除 **Flash** 中指定地址的内容、读取和写入对应寄存器的值。

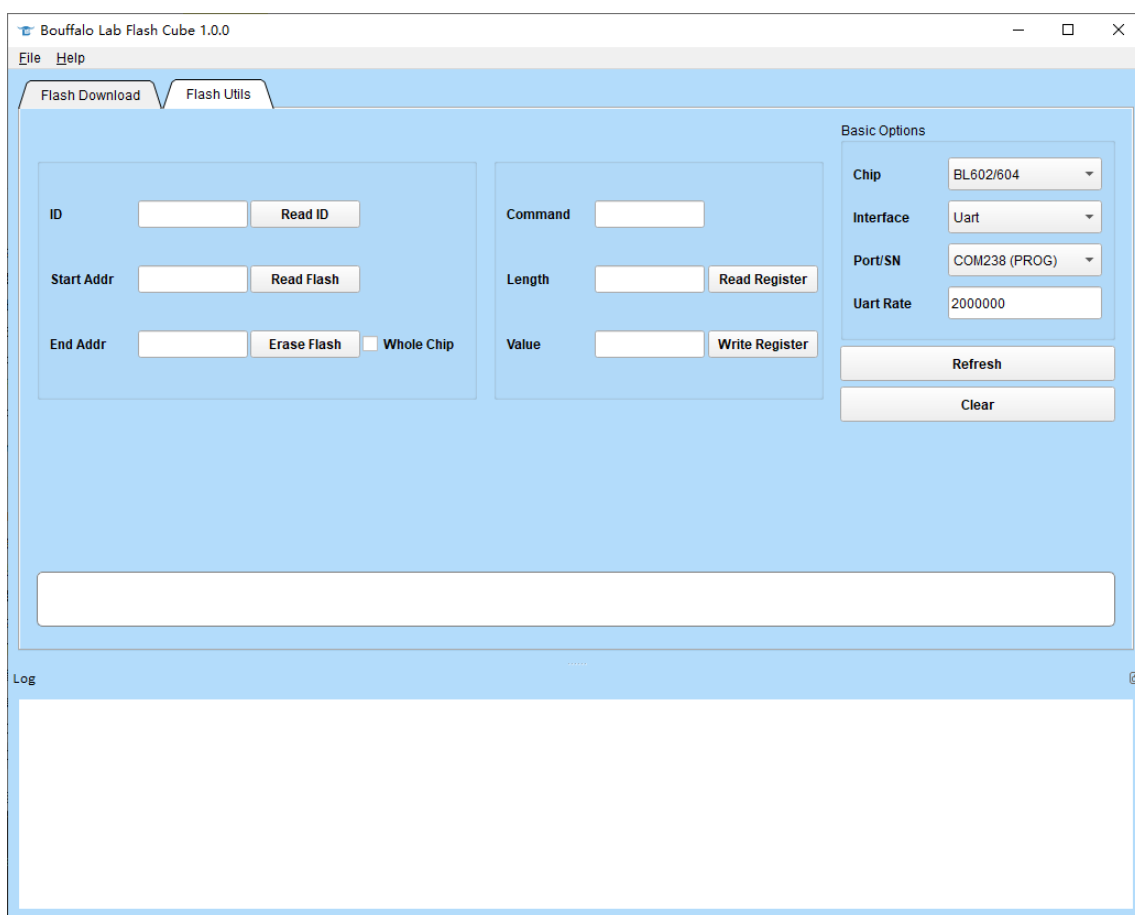


图 6.1: Flash 调试助手界面

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，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 内容

- 读取 Flash 的 ID: 点击 Read ID

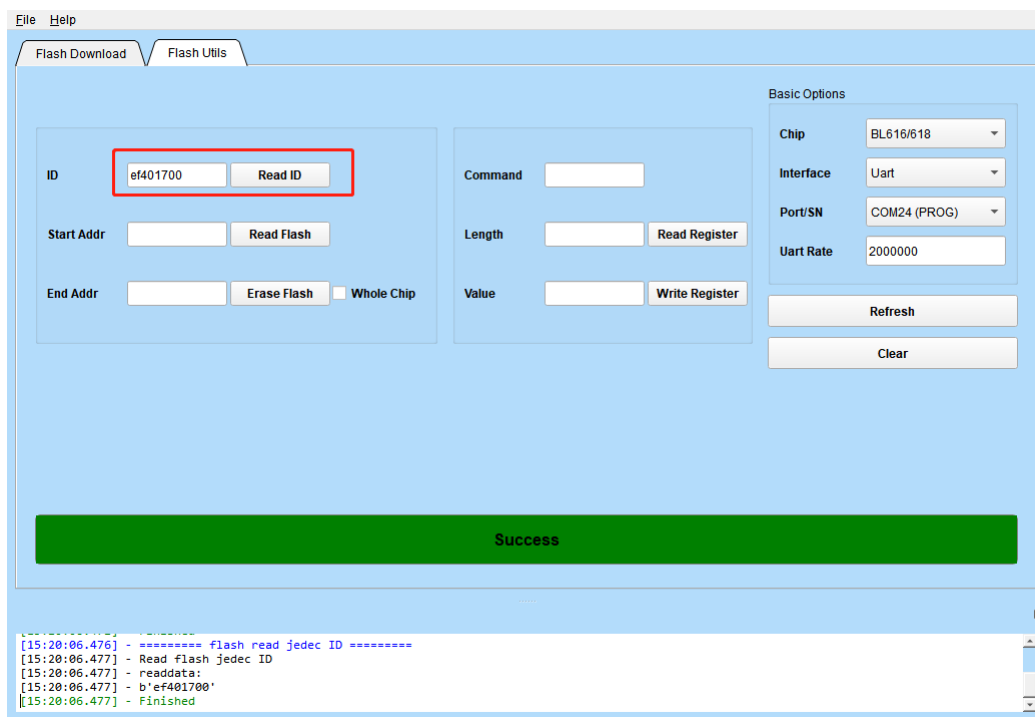


图 6.2: 读取 flash ID

- 读取 Flash 固定长度的值: 在 Start Addr 中设置需要读取数据的开始地址，在 End Addr 中设置需要读取数据的结束地址，点击 Read Flash，读取的内容会存放在工具的根目录下 flash.bin 文件中

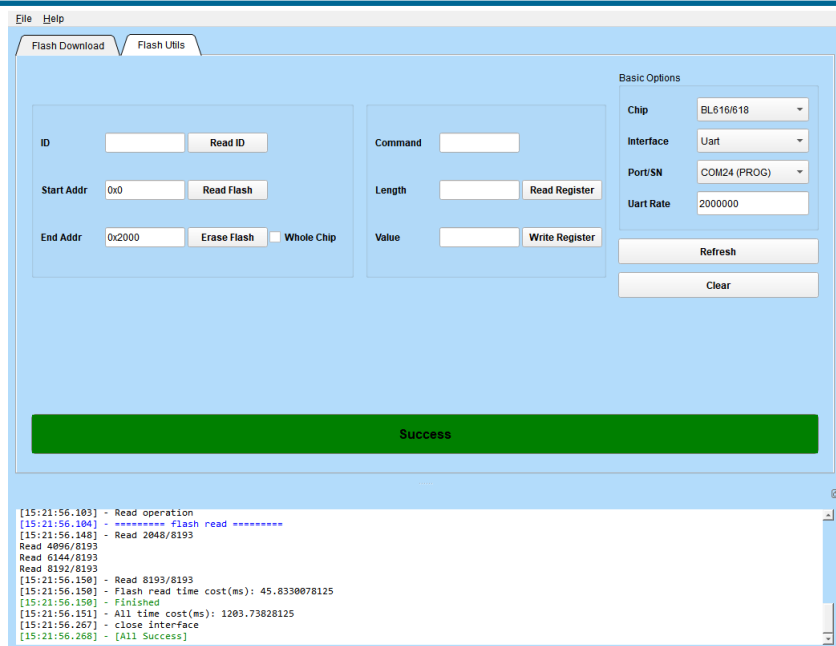


图 6.3: 读取 flash 0x0 ~ 0x2000 地址的数据

6.3 擦除 Flash 内容

- 擦除 Flash 固定长度的值：在 Start Addr 中设置需要擦除数据的开始地址，在 End Addr 中设置需要擦除数据的结束地址，点击 Erase Flash(若需要擦除整块芯片的值，需要勾选 Whole Chip)

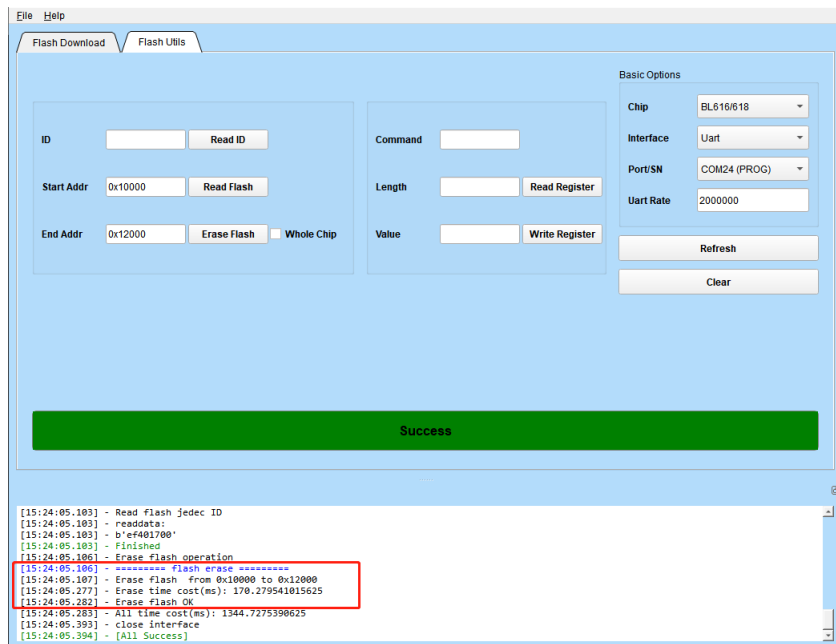


图 6.4: 擦除 Flash 界面

6.4 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

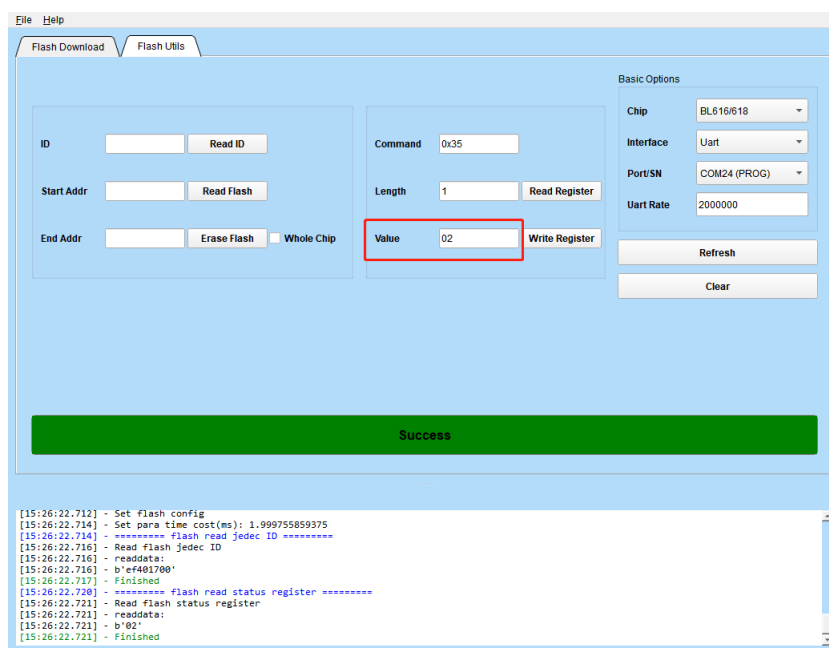


图 6.5: 读寄存器界面

- 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

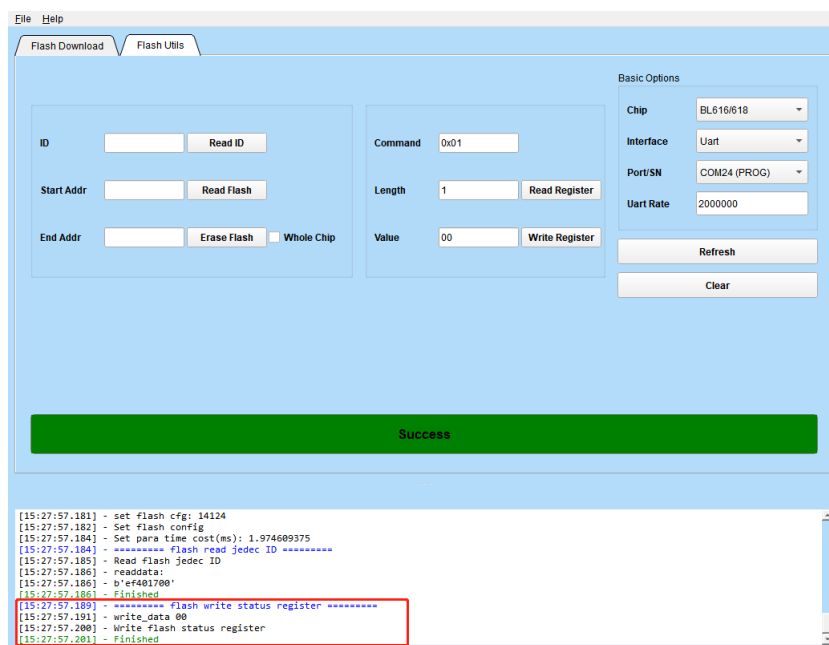


图 6.6: 写寄存器界面

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` 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/sta*_$(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/sta*_\$(CHIPNAME).bin”模糊匹配可以查找到 build/build_out 目录下的 sta_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!”

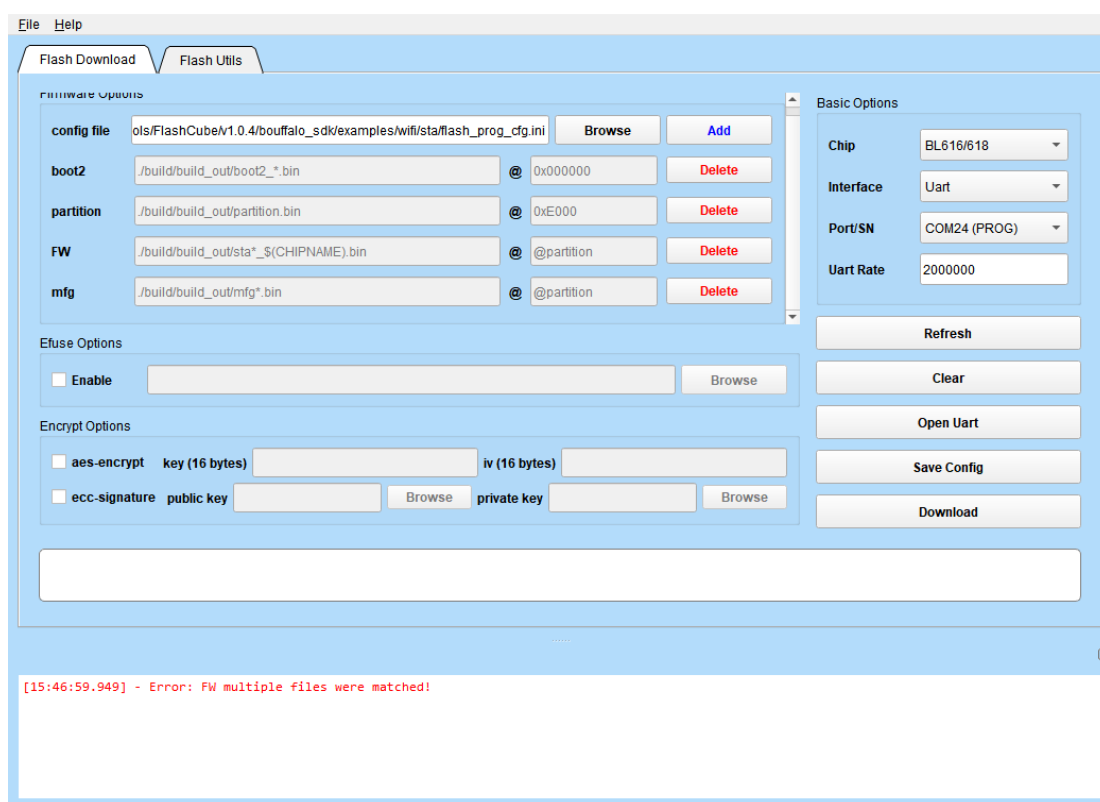


图 7.1: 查找固件错误

7.2 支持 ISP 烧写模式

ISP 即 In-System Programming，Flash Cube 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。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 秒钟以内复位一下芯片，在握手成功后会提示”read ready”或”isp ready”，然后成功烧写。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，然后握手并执行烧写操作。

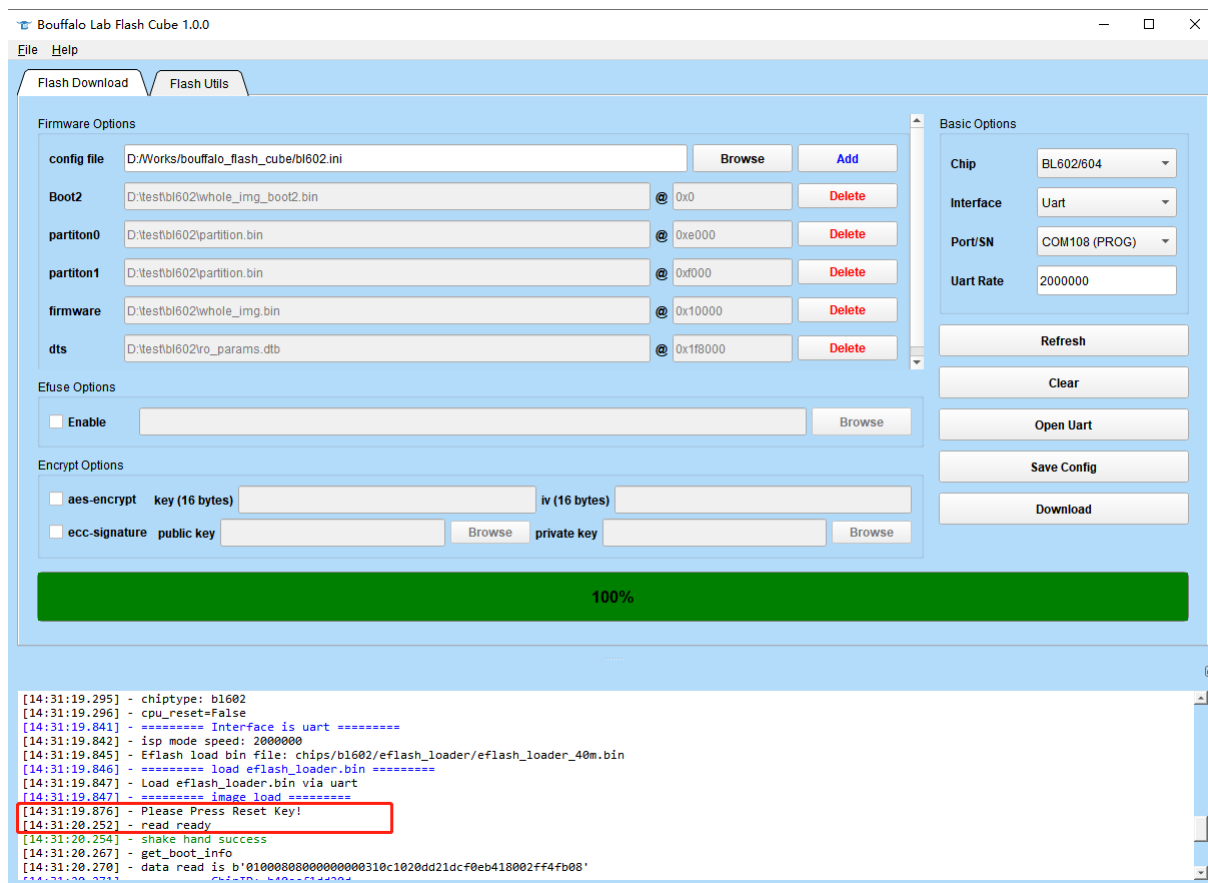


图 7.2: ISP 烧写模式

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，芯片会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。

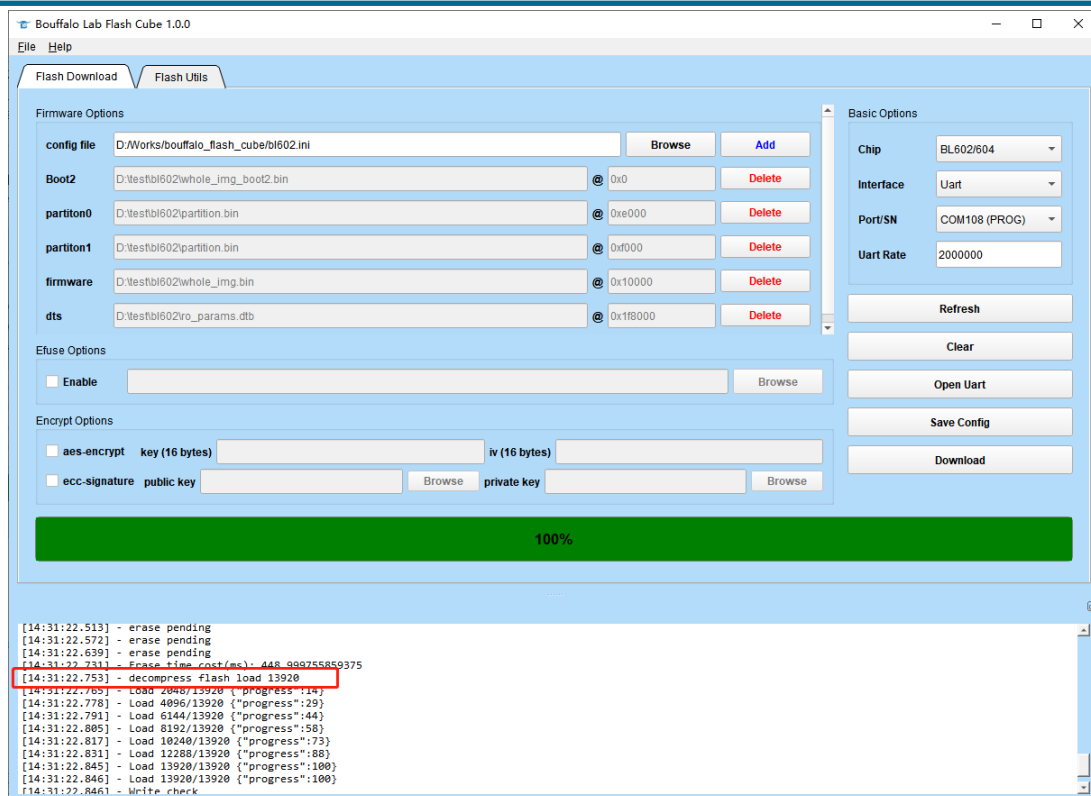


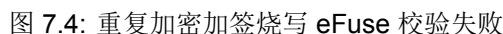
图 7.3: 压缩烧写模式

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.ini 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。



File

Help

Flash Download

Flash Utils

Firmware Options

config file

D:\test\buffalo_flash_cube\flash_prog_cfg_mcu.ini

Browse

Add

Basic Options

Chip

BL602/604

Interface

Uart

Port/5N

COM41 (PROG)

Uart Rate

2000000

Refresh

Clear

Open Uart

Save Config

Download

Efuse Options

☒ Enable

D:\test\efusedata.bin

Browse

Encrypt Options

☒ aes-encrypt

key (16 bytes)

12345678123456781234567812345678

iv (16 bytes)

12345678123456781234567800000000

☒ ecc-signature

public key

be_v1.0.2\utils\pem\publickey_uecc.pem

Browse

private key

e_v1.0.2\utils\pem\privatekey_uecc.pem

Browse

100%

```

[13:36:35.332] - b"eF401580"
[13:36:35.332] - Finished
[13:36:35.335] - Program operation
[13:36:35.335] - Warning: No input file to program to flash
[13:36:35.335] - ===== efuse load =====
[13:36:35.335] - Load file: D:/test/efusedata.bin
[13:36:35.335] - Decrypt efuse data
[13:36:35.335] - Load efuse 0
[13:36:35.338] - Finished
[13:36:35.339] - All time cost(ms): 3200.465576171875
[13:36:35.562] - close interface
[13:36:35.564] - [All Success]

```

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：

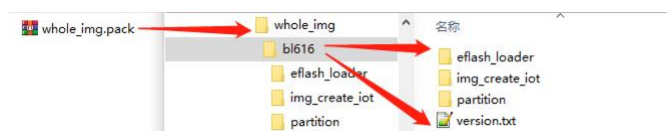


图 7.11: whole_img.pack 构成

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

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.ini) 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.ini 文件中, 生成的 whole_img.pack 中使用的也是更新后的 eflash_loader_cfg.ini 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

Flash Cube 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!。

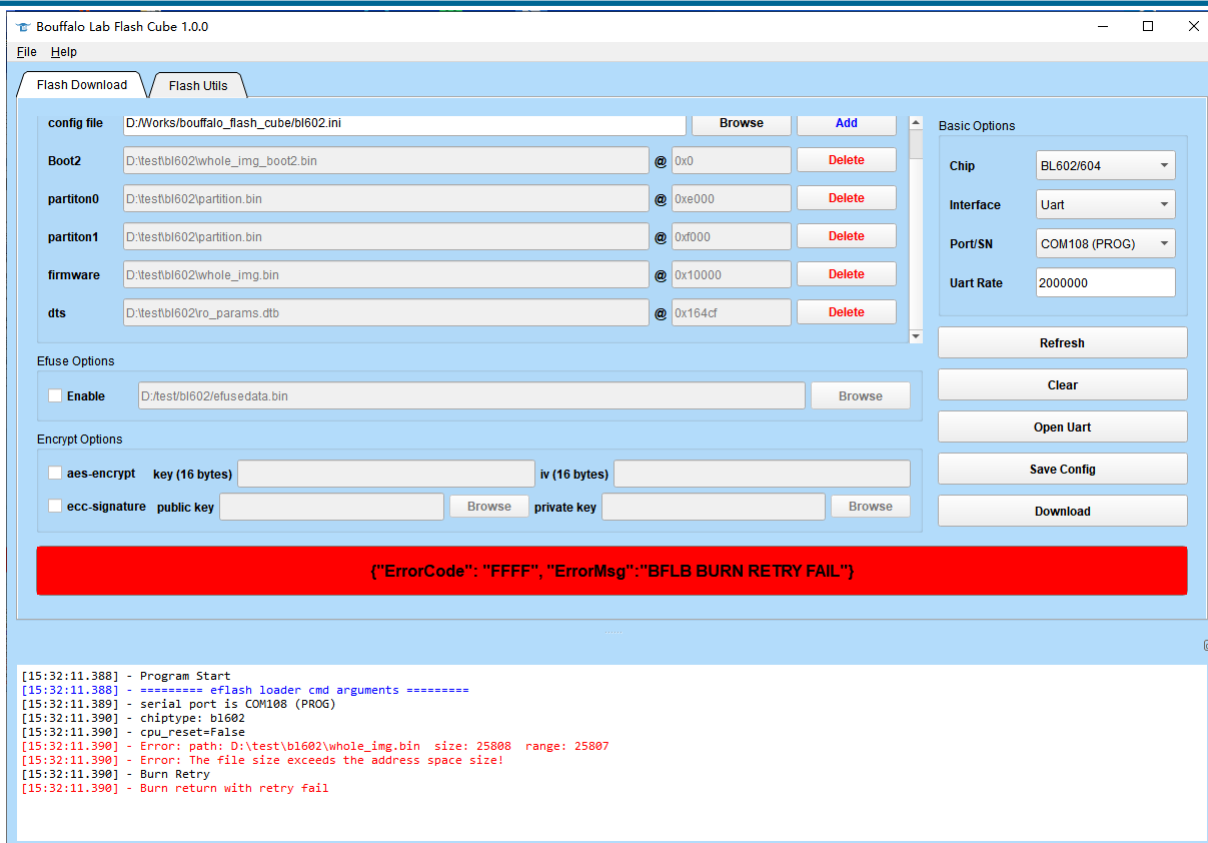


图 8.1: 固件大小超出错误

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