



Flash Cube

使用说明

Version: 1.1

Copyright @ 2022

www.bouffalolab.com

1	Flash Cube 简介	4
2	程序下载	5
2.1	更新配置文件	5
2.2	进入烧写页面	7
2.3	配置程序下载方式	8
2.4	导入配置文件	8
2.5	下载程序	10
2.6	启动程序	11
2.7	修改界面烧录选项	11
2.8	保存烧录页面配置	13
3	命令行下载	14
4	Flash 调试助手	18
4.1	配置程序下载方式	19
4.2	读擦 Flash 内容	19
4.3	读写寄存器内容	20
5	高级功能	21
5.1	自定义烧录界面	21
5.2	支持固件路径模糊匹配	21
5.3	支持 ISP 烧写模式	22
5.4	支持压缩烧写	23
5.5	支持重复加密/加签烧写	24
5.6	支持 efuse 校验选择	26
5.7	支持修改烧录时擦除方式	28
5.8	支持擦写的 skip 功能	29
5.9	生成完整的镜像文件	31

6	注意事项	32
6.1	自定义的功能配置以用户导入为准	32
6.2	烧录界面每个烧录选项名称最大支持 10 个字符	32
6.3	晶振类型默认设定	32
6.4	固件超出分配的地址大小时会提示错误	32
7	修改记录	34

Flash Cube 是博流提供的芯片集成开发工具，主要提供程序的下载功能。本文档主要介绍程序下载的方法及相关配置。

Flash Cube 提供用户下载应用程序的功能，通过导入配置文件的方式实现烧录功能。自定义的烧录界面可以支持多分区多地址烧写，能够同时烧录用户资源文件、分区表、boot2 等。工具还提供 **efuse** 烧写功能，支持烧写加密/签名固件。

Flash Cube 的主要功能如下：

1. 支持应用程序的下载
2. 支持多种型号 **Flash** 的擦、写、读
3. 支持各类文件下载到 **Flash** 并验证
4. 下载通讯接口支持 **UART**、**JLink**、**CKLink** 和 **OpenOCD**，下载速度可配
5. 支持烧写 **AES** 加密以及签名功能的程序镜像

用户可以通过 [Bouffalo Lab Flash Cube](#)，获取最新版本的 **Flash Cube**。双击解压后文件夹中的 `BLFlashCube.exe` 即可进入 **Flash Cube** 主界面。

2.1 更新配置文件

烧录工具 Flash Cube 使用的是 ini 类型配置文件。

SDK 的测试 case 目录下提供了默认的配置文​​件 `flash_prog_cfg.ini`，下面主要介绍一下 `Flash prog cfg.ini` 的语法。

2.1.1 常规 MCU 使用（不使用无线功能）

```
[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

[firmware]
filedir = ./build/build_out/xxx*_${CHIPNAME}.bin
address = 0x0000
```

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。

firmware 表示要烧录的应用固件

- **filedir**: 表示应用固件所在相对路径，SDK 编译完成后会放在 `build/build_out` 目录。其中“xxx”表示应用固件名

称, `_${CHIPNAME}.bin` 是必须的后缀, 用于区分不同芯片类型, 路径支持正则匹配, 可使用 `*` 进行字符串模糊匹配。

- **address:** 必须使用 0 地址

2.1.2 常规 IOT 使用（使用无线功能）

```
[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/xxx*_${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, SDK 编译完成后会放在 `build/build_out` 目录。
- **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，SDK 编译完成后会放在 build/build_out 目录。
- **address**: 由 SDK 编译时选择的分区表文件 '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，SDK 编译完成后会放在 build/build_out 目录。其中 "xxx" 表示应用固件名称，_\${CHIPNAME}.bin 是必须的后缀，用于区分不同芯片类型，路径支持正则匹配，可使用 * 进行字符串模糊匹配。
- **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，SDK 编译完成后会放在 build/build_out 目录。
- **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 = 0x17000。

用户可以根据实际需要调整参数和烧写项配置，或者按照上述格式作为模板自定义专属配置文件。

2.2 进入烧写页面

配置文件准备好后，双击解压后文件夹中的 BLFlashCube.exe 进入 Flash Cube 主界面。在首行菜单中选择 Flash Download 选项，会进入程序下载主界面，分为固件程序烧录和 efuse 烧录。

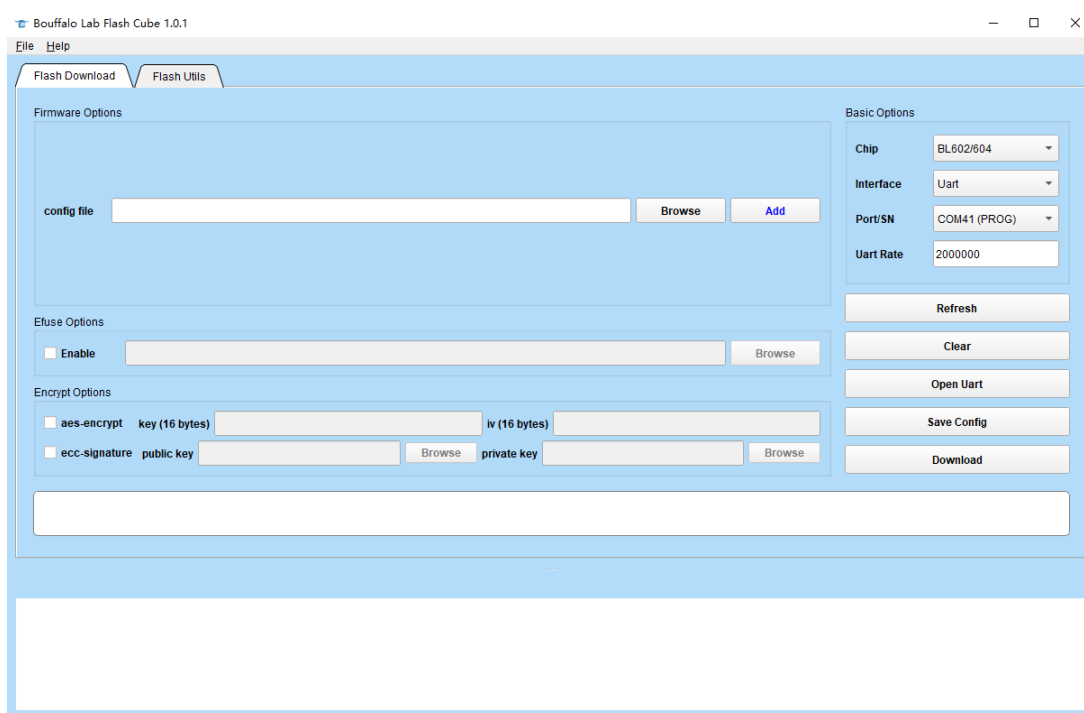


图 2.1: 烧写主界面

2.3 配置程序下载方式

在主界面的右侧 **Basic Options** 区域为程序下载的相关配置，如芯片类型、烧写方式、串口号等等。

用户需要手动修改这些配置选项。

- 配置参数包括：
 - **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

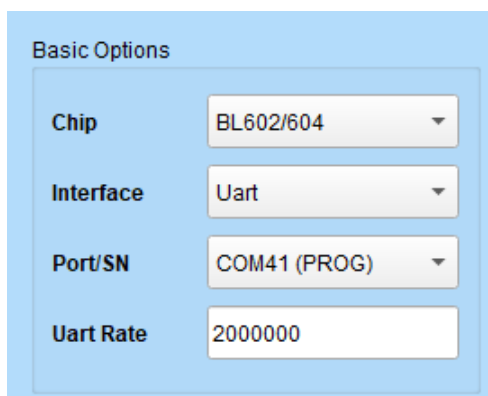


图 2.2: 程序下载方式选择界面

以上图为例,当前烧写的是 BL602,连接到 PC 端的串口为 COM41,使用 Uart 烧写方式,所以 Chip 类型选择 BL602/604, Interface 选择 Uart, Port/SN 选择 COM41, Uart Rate 选择 2000000。

2.4 导入配置文件

Flash Cube 主页面无默认的烧写项，需要用户导入配置文件或者使用 **Add** 按钮新增烧写项。

在 **Firmware Options** 区域，**config file** 显示所选择的配置文件，**Browse** 按钮选择配置文件，**Add** 按钮新增烧写项。

点击 **Browse** 按钮选择刚自定义的配置文件（本例中为 `flash_prog_cfg.ini`）。

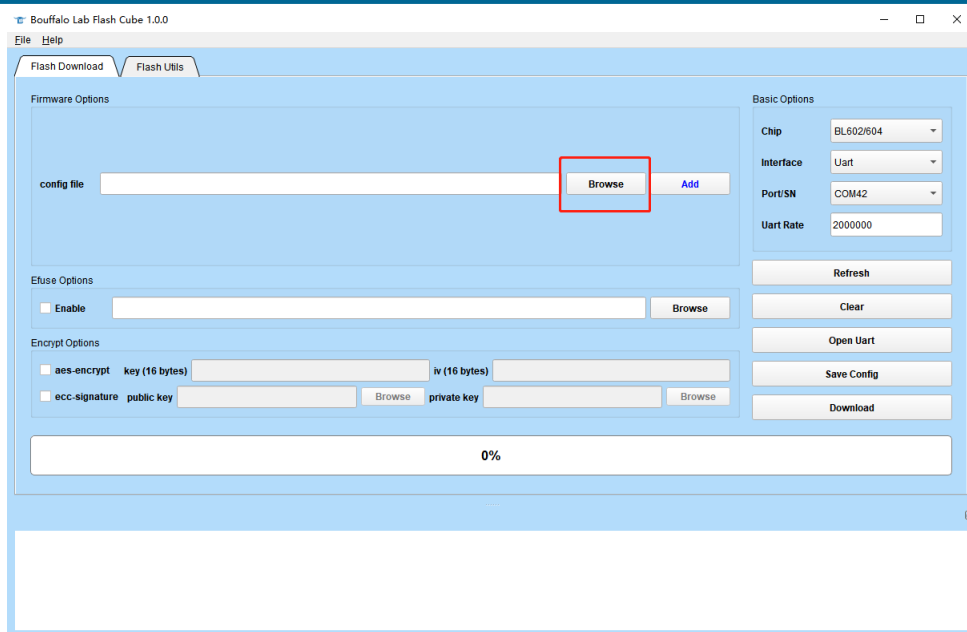


图 2.3: 选择自定义的配置文件

如果导入的是常规 MCU 使用（不使用无线功能）的配置文件，则更新后的界面如下图所示：

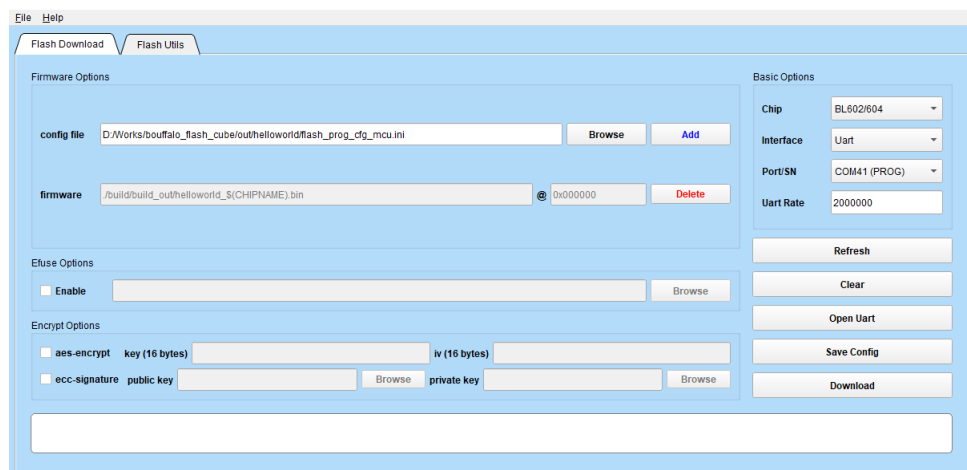


图 2.4: 导入常规 MCU 使用的配置文件

如果导入的是常规 IOT 使用（使用无线功能）的配置文件，则更新后的界面如下图所示：

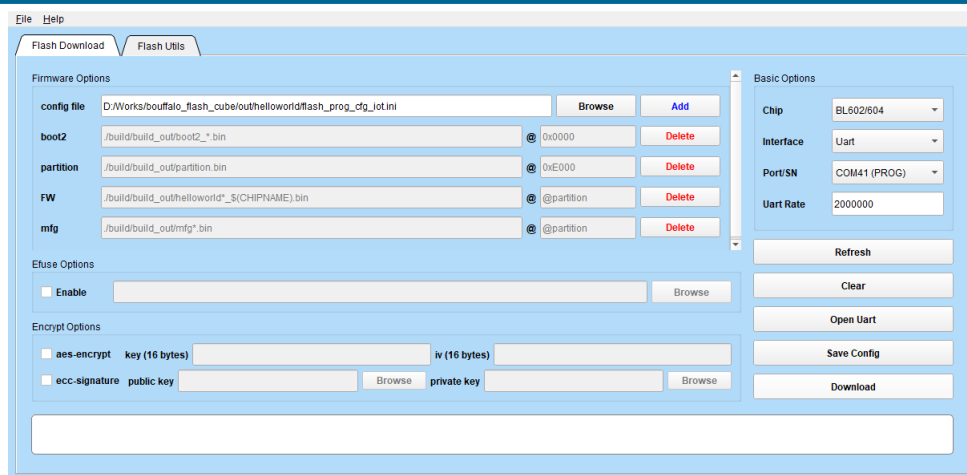


图 2.5: 导入常规 IOT 使用的配置文件

2.5 下载程序

当选择 UART 方式烧录程序，需要将板子的 BOOT 设置为高电平，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 Boot 引脚和 Reset 芯片)。

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。

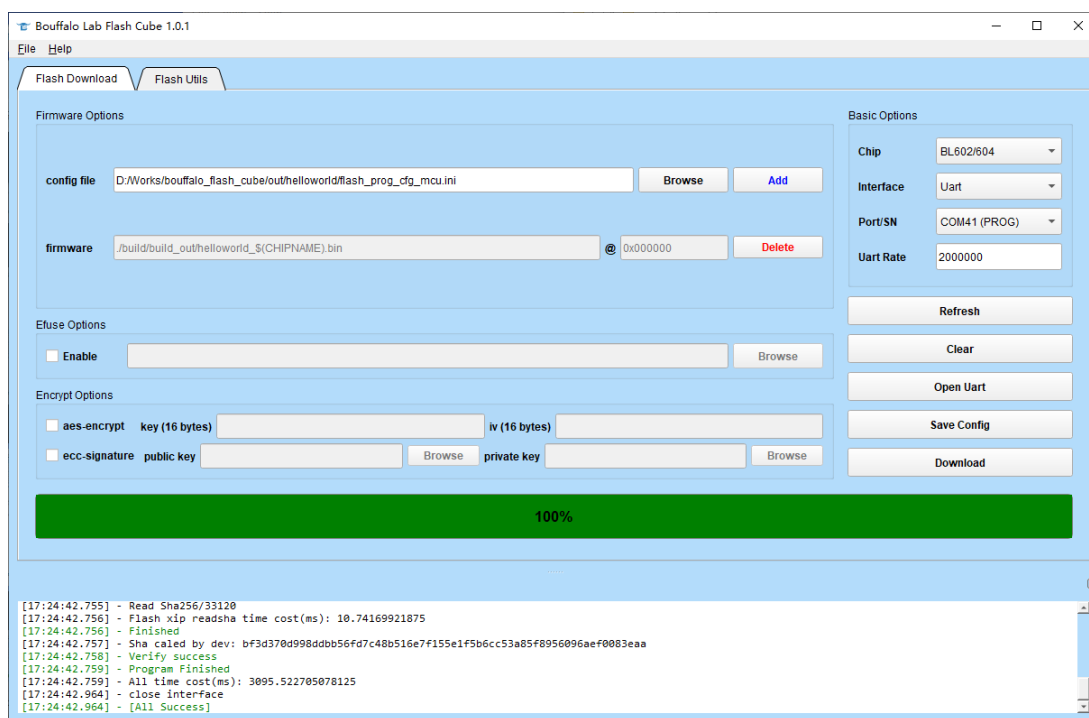


图 2.6: 成功下载程序

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

2.6 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 hello world 程序运行起来的效果。



图 2.7: hello world 程序结果

2.7 修改界面烧录选项

Flash Cube 工具支持在界面中更改烧录选项，支持新增和删除烧录项。

点击页面中的 Add 按钮可以新增一个烧写项，点击之后会弹出一个窗口，需要依次填入 name，烧录地址以及烧录文件。

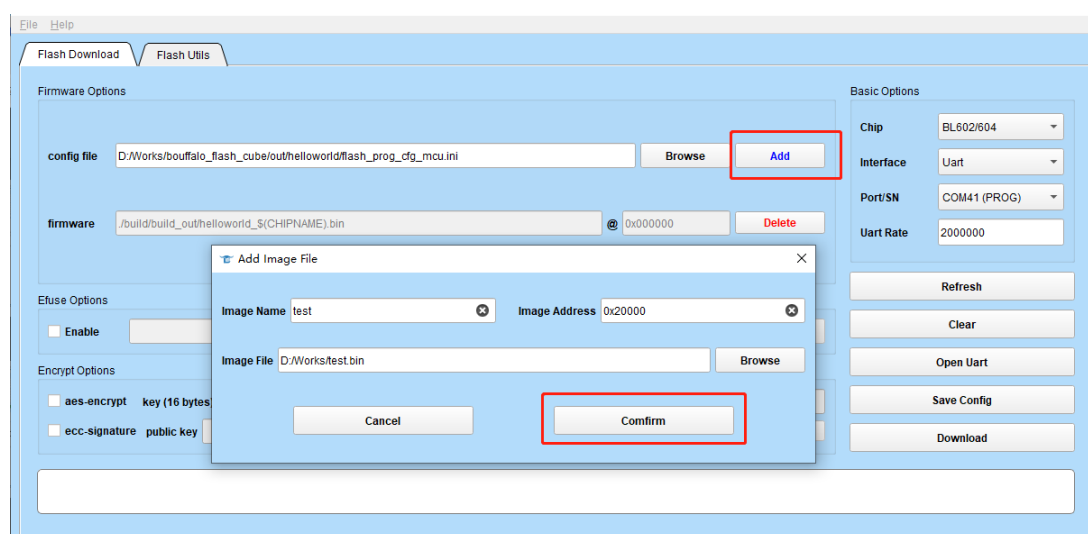


图 2.8: 添加烧录镜像文件

其中 Image Name 限制为不超过 10 个字符，Image Address 限制为十六进制，Image File 必须是已经存在的文件，否则会出现错误提示：“Image file is not existed, add image failed”。

以烧写 test.bin 到 0x20000 地址为例，Image Name 中填写镜像名称 test，Image Address 中填写烧录地址 0x20000，Image File 可点击 Browse 按钮选择烧录文件 test.bin，也可手动填写 test.bin 的绝对路径或相对路径，点击 Confirm 按钮后添加成功（Add Image File 页面参数皆不可为空，否则无法添加成功），添加后的界面如下所示：

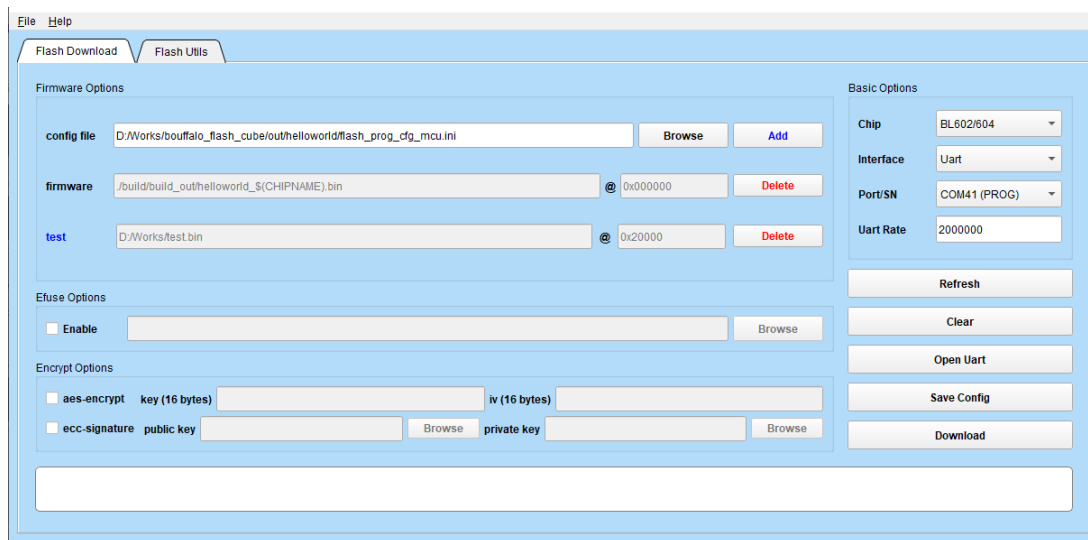


图 2.9: 添加 test 镜像文件成功

在每个烧录项的末尾都有一个 Delete 按钮，对于不需要的选项，通过点击 Delete 即可实现删除功能。以删除 test 为例，点击 Delete 按钮之后，页面会自动更新。

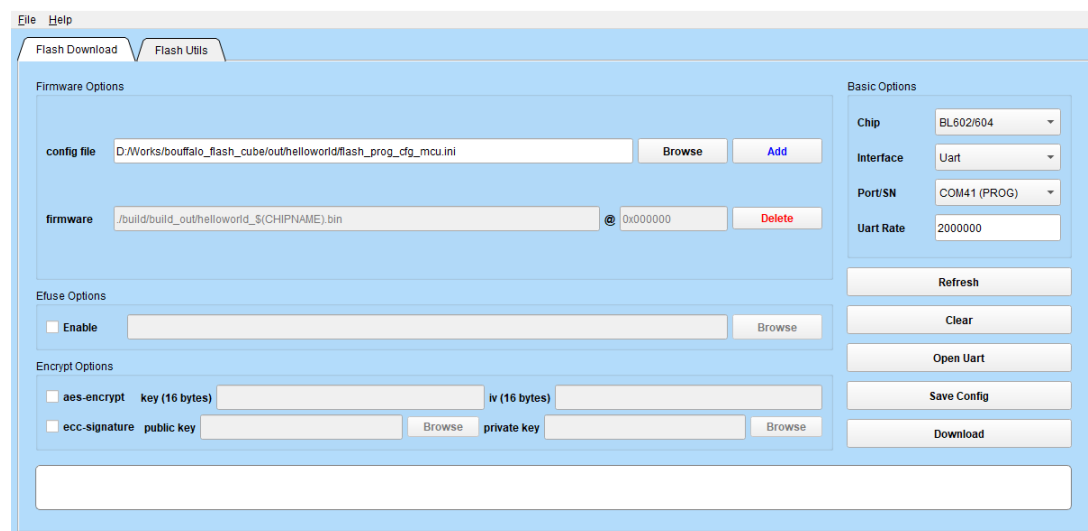


图 2.10: 删除 test 选项界面

2.8 保存烧录页面配置

点击界面右侧的 **Save Config** 按钮，即可保存或者更新当前的配置。如果需要将配置保存到其他位置，只需修改 **config file** 的路径，下图中保存了配置到 **flash_prog_cfg_bak.ini** 文件中。

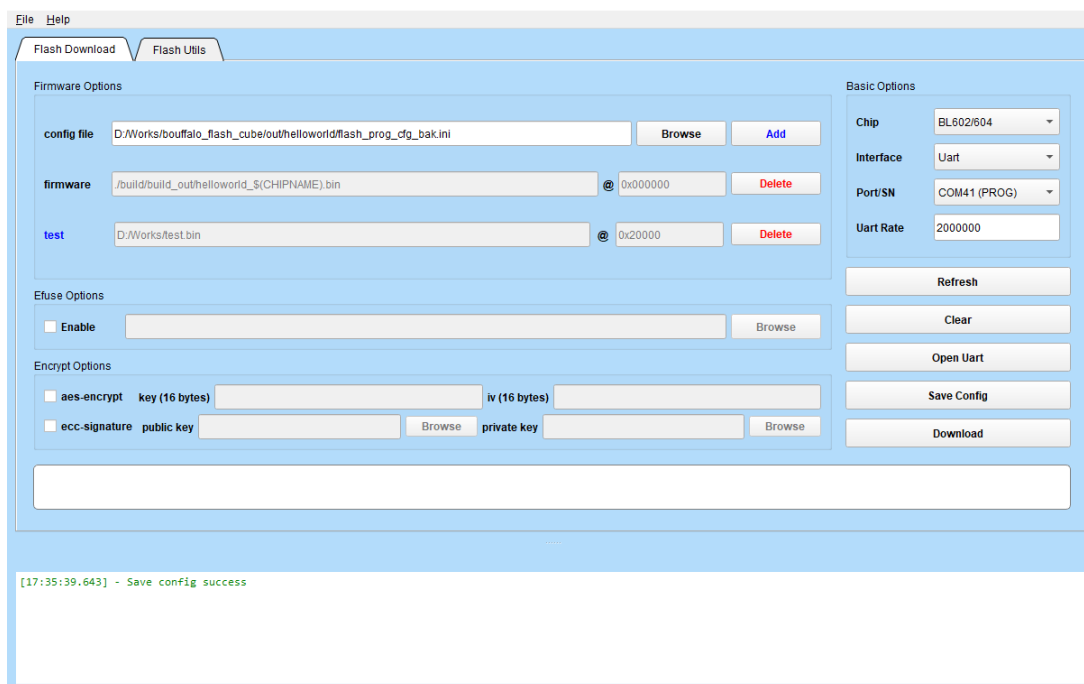


图 2.11: 保存配置

Flash Cube 中默认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=bl602 --port=COM41 --
baudrate=2000000 --config=flash_prog_cfg.ini
['D:\Works\bouffalo_flash_cube\BLFlashCommand.exe', '--interface=uart', '--chipname=bl602', '--port=COM41
', '--baudrate=2000000', '--config=flash_prog_cfg.ini']
[19:45:09.568] - Serial port is COM41
[19:45:09.568] - Baudrate is 2000000
[19:45:09.568] - =====
[19:45:09.572] - Program Start
[19:45:09.572] - ===== eflash loader cmd arguments =====
[19:45:09.573] - serial port is COM41
[19:45:09.573] - chiptype: bl602
[19:45:09.573] - cpu_reset=False
[19:45:09.587] - ===== Interface is uart =====
[19:45:09.587] - Eflash load bin file: D:\Works\bouffalo_flash_cube\chips/bl602/eflash_loader/eflash_loader_
40m.bin
[19:45:09.587] - ===== load eflash_loader.bin =====
[19:45:09.587] - Load eflash_loader.bin via uart
[19:45:09.588] - ===== image load =====
[19:45:09.886] - tx rx and power off, press the machine!
[19:45:09.886] - cutoff time is 0.1
[19:45:09.996] - power on tx and rx
[19:45:10.803] - reset cnt: 0, reset hold: 0.005, shake hand delay: 0.1
[19:45:10.806] - clean buf
[19:45:10.810] - send sync
[19:45:11.019] - ack is b'4f4b464c0201'
[19:45:11.067] - shake hand success
[19:45:11.083] - get_boot_info
[19:45:11.091] - data read is b'01000000000000003000400a5f9e9fdd7c49100'
[19:45:11.091] - ===== ChipID: c4d7fde9f9a5 =====
[19:45:11.092] - last boot info: None
[19:45:11.092] - sign is 0 encrypt is 0
[19:45:11.118] - segcnt is 1
[19:45:11.139] - segdata_len is 37184
[19:45:11.235] - 4080/37184
[19:45:11.331] - 8160/37184
[19:45:11.428] - 12240/37184
[19:45:11.524] - 16320/37184
[19:45:11.619] - 20400/37184
[19:45:11.716] - 24480/37184
[19:45:11.812] - 28560/37184
[19:45:11.907] - 32640/37184
[19:45:12.003] - 36720/37184
```

(continues on next page)

(continued from previous page)

```
[19:45:12.019] - 37184/37184
[19:45:12.036] - Run img
[19:45:12.158] - Load helper bin time cost(ms): 2570.591064453125
[19:45:12.267] - Flash load shake hand
[19:45:12.272] - default set DTR high
[19:45:12.379] - clean buf
[19:45:12.381] - send sync
[19:45:12.598] - ack is b'4f4b'
[19:45:12.645] - Read mac addr
[19:45:12.660] - flash set para
[19:45:12.660] - ===== flash read jedec ID =====
[19:45:12.676] - Read flash jedec ID
[19:45:12.677] - readdata:
[19:45:12.679] - b'c8401580'
[19:45:12.679] - Finished
[19:45:12.734] - Program operation
[19:45:12.734] - Dealing Index 0
[19:45:12.735] - ===== programming D:\Works\bouffalo_flash_cube\helloworld_bl602.bin to 0x000000
[19:45:12.738] - ===== flash load =====
[19:45:12.738] - ===== flash erase =====
[19:45:12.741] - Erase flash from 0x0 to 0x815f
[19:45:12.755] - erase pending
[19:45:12.867] - erase pending
[19:45:12.924] - Erase time cost(ms): 182.915283203125
[19:45:12.938] - decompress flash load 17732
[19:45:12.964] - Load 2048/17732 {"progress":11}
[19:45:12.979] - Load 4096/17732 {"progress":23}
[19:45:12.996] - Load 6144/17732 {"progress":34}
[19:45:13.011] - Load 8192/17732 {"progress":46}
[19:45:13.027] - Load 10240/17732 {"progress":57}
[19:45:13.044] - Load 12288/17732 {"progress":69}
[19:45:13.059] - Load 14336/17732 {"progress":80}
[19:45:13.075] - Load 16384/17732 {"progress":92}
[19:45:13.091] - Load 17732/17732 {"progress":100}
[19:45:13.091] - Load 17732/17732 {"progress":100}
[19:45:13.092] - Write check
[19:45:13.109] - Flash load time cost(ms): 183.462158203125
[19:45:13.109] - Finished
[19:45:13.115] - Sha calcd by host: 60cf293df703ac6dd53cc962c67e93bb23940c3ebb8a03deabc37dfb1a02d18e
[19:45:13.115] - xip mode Verify
[19:45:13.140] - Read Sha256/33120
[19:45:13.140] - Flash xip readsha time cost(ms): 16.60791015625
[19:45:13.144] - Finished
```

(continues on next page)

(continued from previous page)

```
[19:45:13.155] - Sha caled by dev: 60cf293df703ac6dd53cc962c67e93bb23940c3ebb8a03deabc37dfb1a02d18e
[19:45:13.155] - Verify success
[19:45:13.156] - Program Finished
[19:45:13.160] - All time cost(ms): 3588.349365234375
[19:45:13.374] - close interface
[19:45:13.376] - [All Success]
```

使用串口工具即可以看到启动 log:



```
Log
[20:48:37.502] - 
[20:48:37.503] - 
[20:48:37.505] - 
[20:48:37.506] - BouffaloLab
[20:48:37.508] - 
[20:48:37.509] - Build:14:28:35, Apr 21 2022
[20:48:37.510] - Copyright (c) 2021 Bouffalolab team
[20:48:37.510] - dynamic memory init success,heap size = 174 Kbyte
[20:48:37.511] - show flash cfg:
[20:48:37.511] - jedec id 0xEF4015
[20:48:37.514] - mid 0xEF
[20:48:37.514] - iomode 0x04
[20:48:37.514] - clk delay 0x01
[20:48:37.515] - clk invert 0x01
[20:48:37.515] - read reg cmd0 0x05
[20:48:37.515] - read reg cmd1 0x35
[20:48:37.517] - write reg cmd0 0x01
[20:48:37.517] - write reg cmd1 0x31
[20:48:37.518] - qe write len 0x01
[20:48:37.519] - cread support 0x01
[20:48:37.519] - cread code 0x20
[20:48:37.520] - burst wrap cmd 0x77
[20:48:37.520] - -----
[20:48:37.520] - bouffalolab />hello world!
[20:48:37.688] - hello world!
[20:48:37.888] - hello world!
[20:48:38.088] - hello world!
[20:48:38.288] - hello world!
[20:48:38.488] - hello world!
[20:48:38.688] - hello world!
[20:48:38.888] - hello world!
[20:48:39.088] - hello world!
[20:48:39.288] - hello world!
[20:48:39.488] - case success
```

图 3.1: 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。

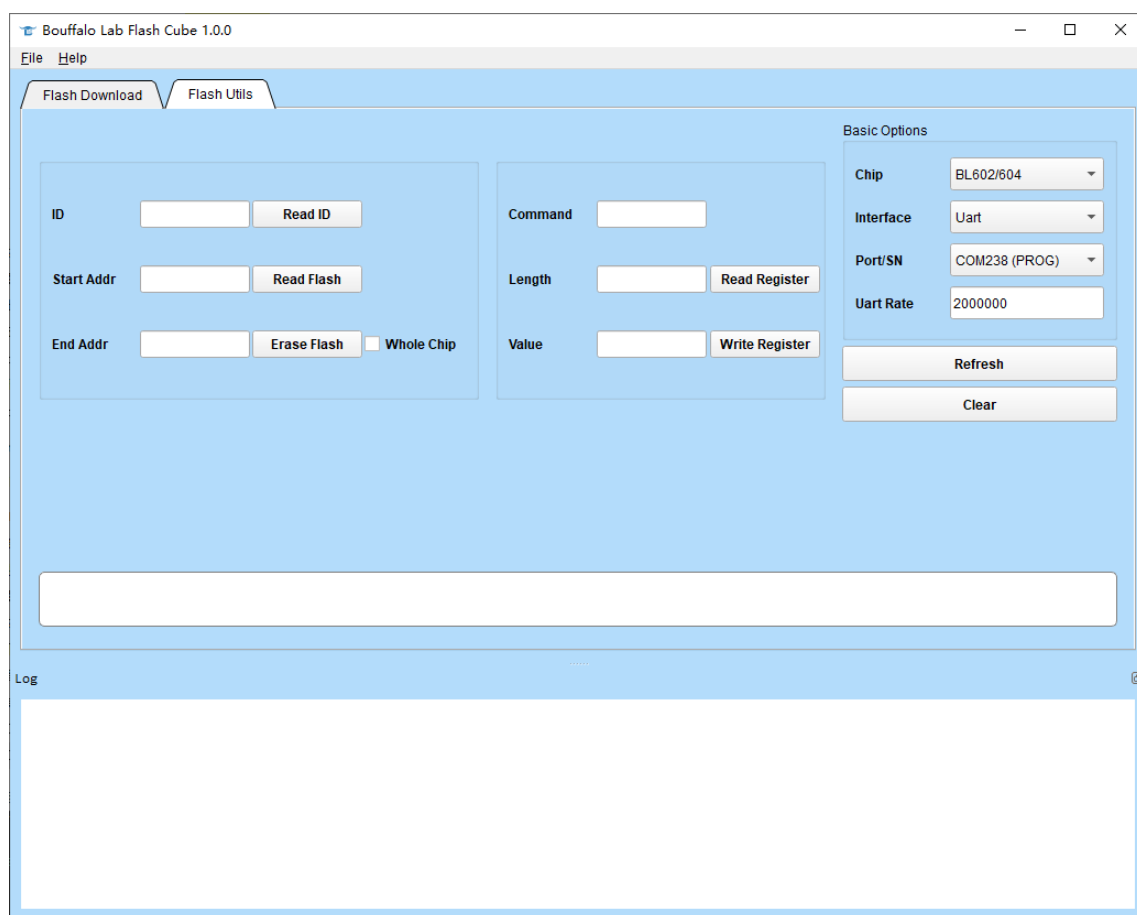
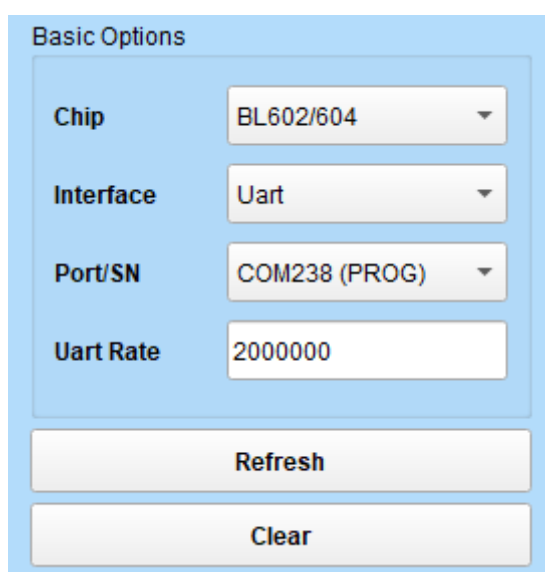


图 4.1: Flash 调试助手界面

4.1 配置程序下载方式

- 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

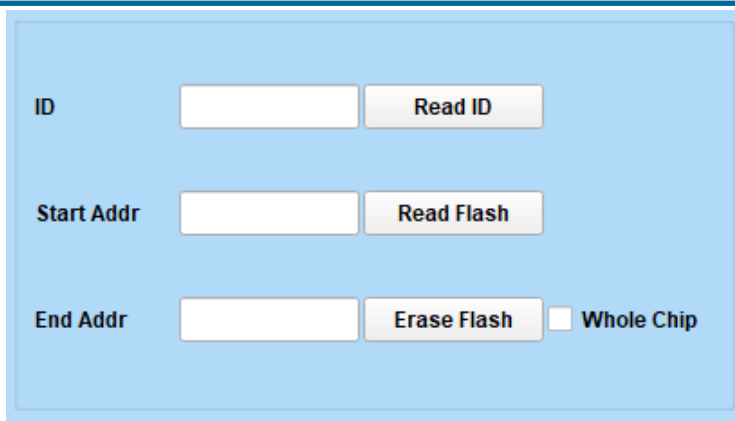


The image shows a 'Basic Options' configuration window. It contains four dropdown menus: 'Chip' set to 'BL602/604', 'Interface' set to 'Uart', 'Port/SN' set to 'COM238 (PROG)', and 'Uart Rate' set to '2000000'. Below these are two buttons: 'Refresh' and 'Clear'.

图 4.2: 下载方式界面

4.2 读擦 Flash 内容

- 读取 Flash 的 ID: 点击 Read 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
- 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，只需勾选 Whole Chip)



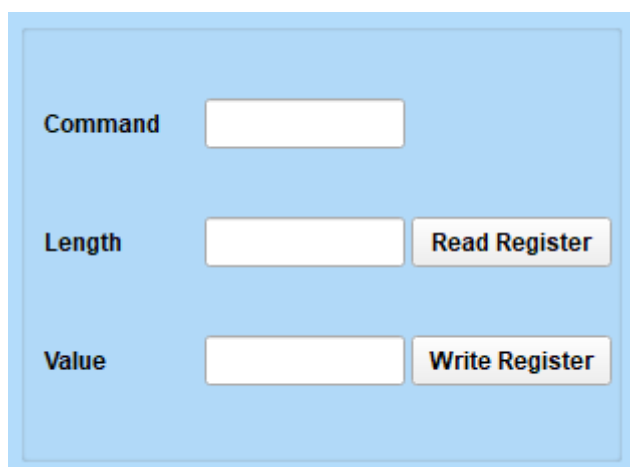
The interface for the Flash Cube tool, showing fields for ID, Start Addr, and End Addr, along with buttons for Read ID, Read Flash, and Erase Flash, and a checkbox for Whole Chip.

ID	<input type="text"/>	Read ID
Start Addr	<input type="text"/>	Read Flash
End Addr	<input type="text"/>	Erase Flash <input type="checkbox"/> Whole Chip

图 4.3: 读擦 Flash 界面

4.3 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
- 写入寄存器的内容: 在 Command 中输入写命令 0x01, 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



The interface for the Flash Cube tool, showing fields for Command, Length, and Value, along with buttons for Read Register and Write Register.

Command	<input type="text"/>	
Length	<input type="text"/>	Read Register
Value	<input type="text"/>	Write Register

图 4.4: 读写寄存器界面

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。

5.1 自定义烧录界面

Flash Cube 使用配置文件的方式自定义烧录界面。每个烧写项和烧写地址都可以自行设定，按照实际需求定制烧录界面。详细参考”修改界面烧录选项”章节。

5.2 支持固件路径模糊匹配

用户导入的配置文件中，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。

如果烧录的是 BL602,且对应目录下存在匹配的文件,则工具能够查找到相对路径”./build/build_out”目录下的 helloworld_bl602.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!”

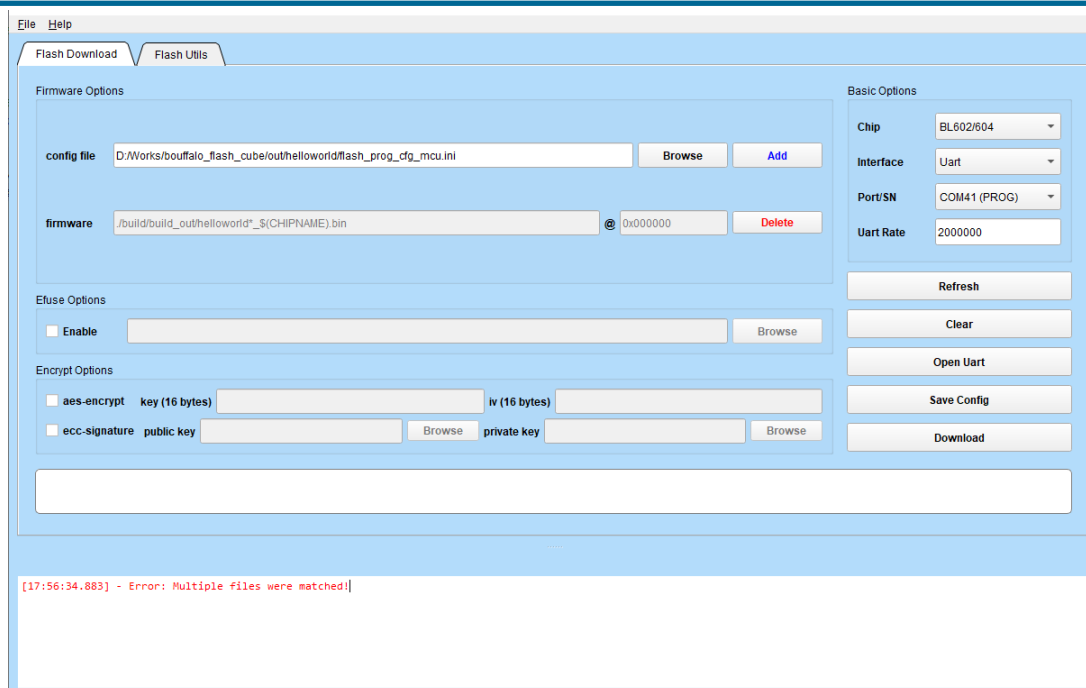


图 5.1: 查找固件错误

5.3 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。

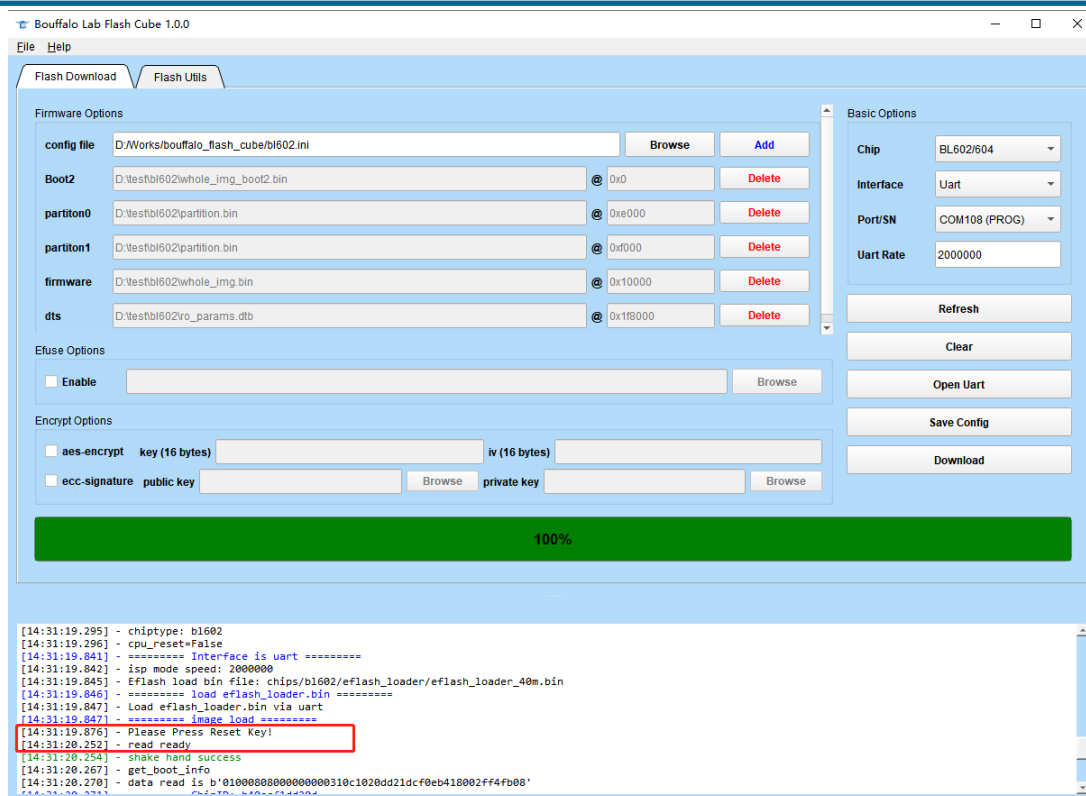


图 5.2: ISP 烧写模式

5.4 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。

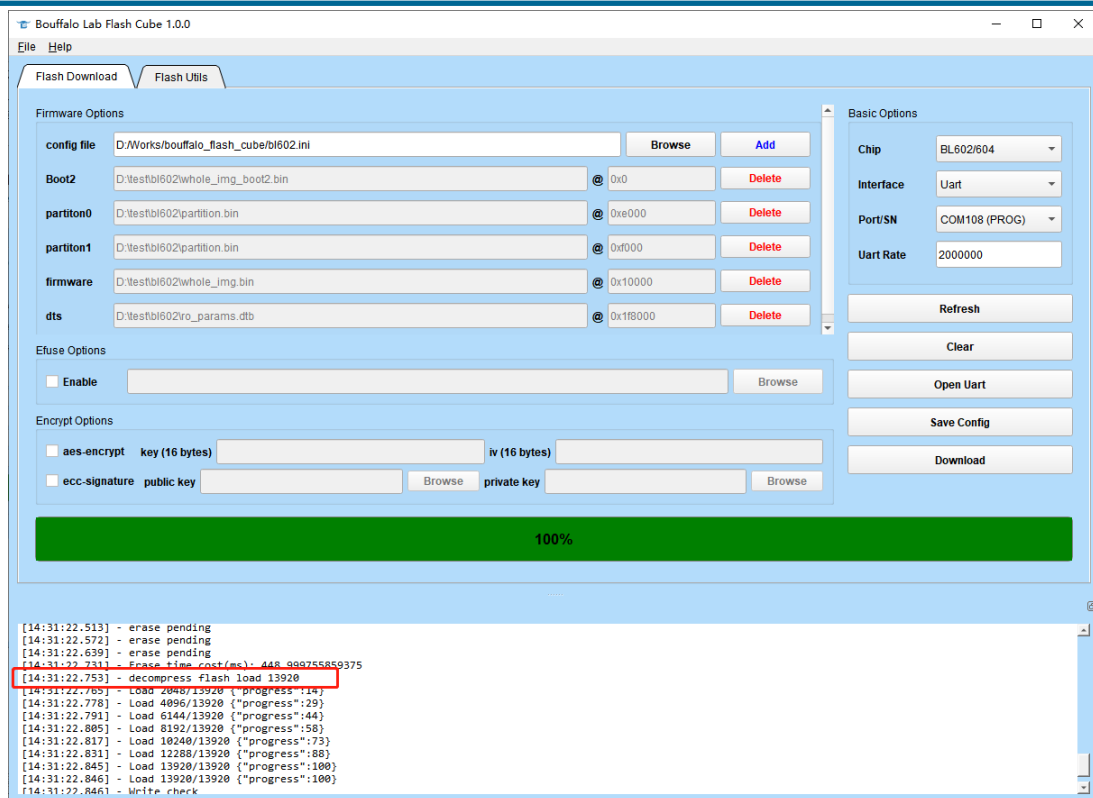


图 5.3: 压缩烧写模式

5.5 支持重复加密/加签烧写

工具支持重复加密加签烧写，在主界面中勾选加密或加签，并将密钥和签名填写进去即可。Flash 重复加密/加签烧写必须使用加密/加签固件才能启动成功，密钥和签名仅用于烧写使用。

注解: aes-encrypt 和 ecc-signature 仅用于重复加密/加签烧写时使用

- **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，烧写完成后启动时会自动做签名。

对于 BL602/BL702，烧写已经加密加签的板子，如果不勾选 **aes-encrypt** 和 **ecc-signature**，烧录 log 会提示: sign is 1 encrypt is 1 且不进行烧录操作。对于 BL808/BL606P/BL616，烧写已经加密加签的板子，如果使用加密加签固件可以直接烧写启动成功。如果烧写非加密加签的固件，启动失败。

以下为 BL602 烧写已经加密加签的芯片，但不勾选 **aes-encrypt** 和 **ecc-signature** 的情况:

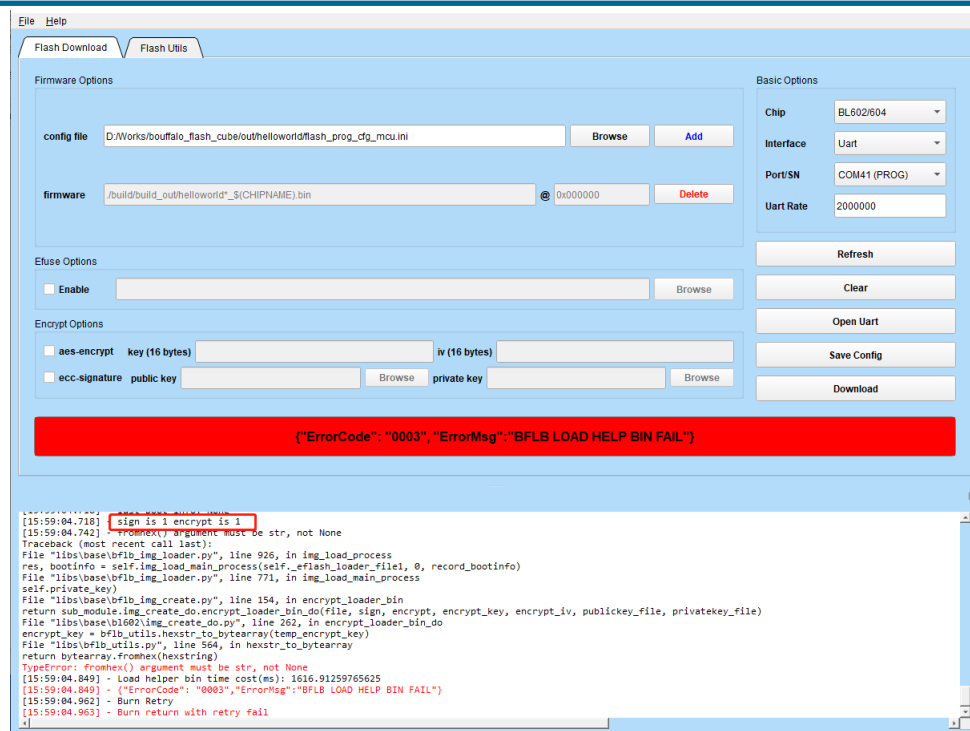


图 5.4: BL602 重复加密加签烧写失败

对于 BL602/BL702，如果勾选上 **aes-encrypt** 和 **ecc-signature**，再次烧写已经加密加签的板子，能够烧写成功

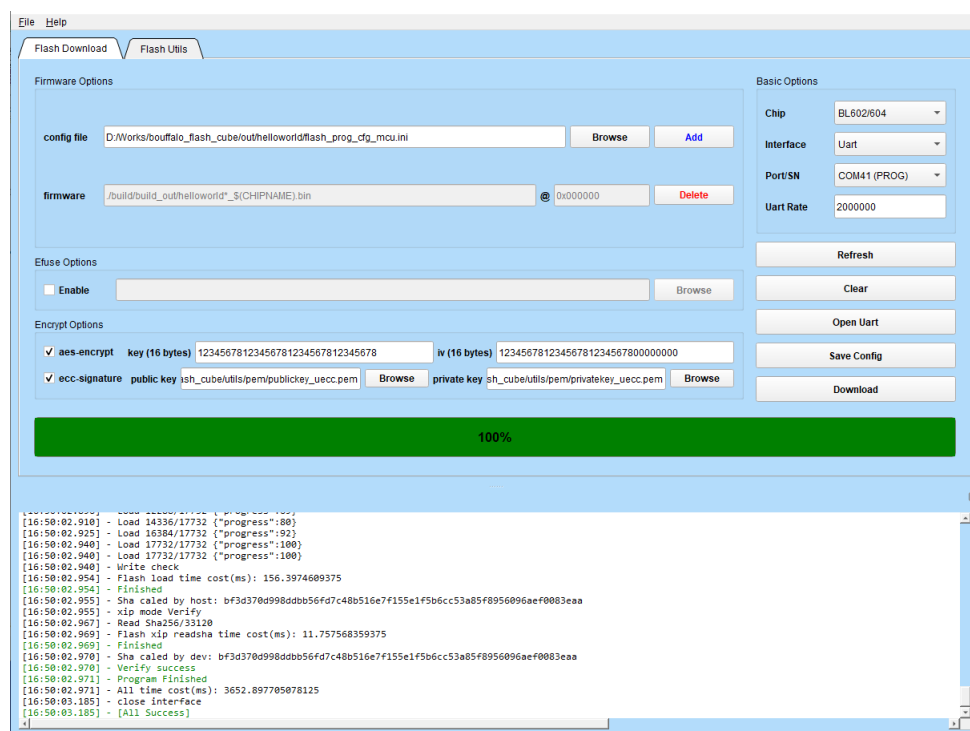


图 5.5: 加密加签烧写成功

如果同时烧写 **flash** 和 **efuse**，则烧写界面如下表示烧写成功（此时 **eflash_loader_cfg.ini** 文件中的 **factory_mode** 为

false, 默认不进行 efuse 校验)

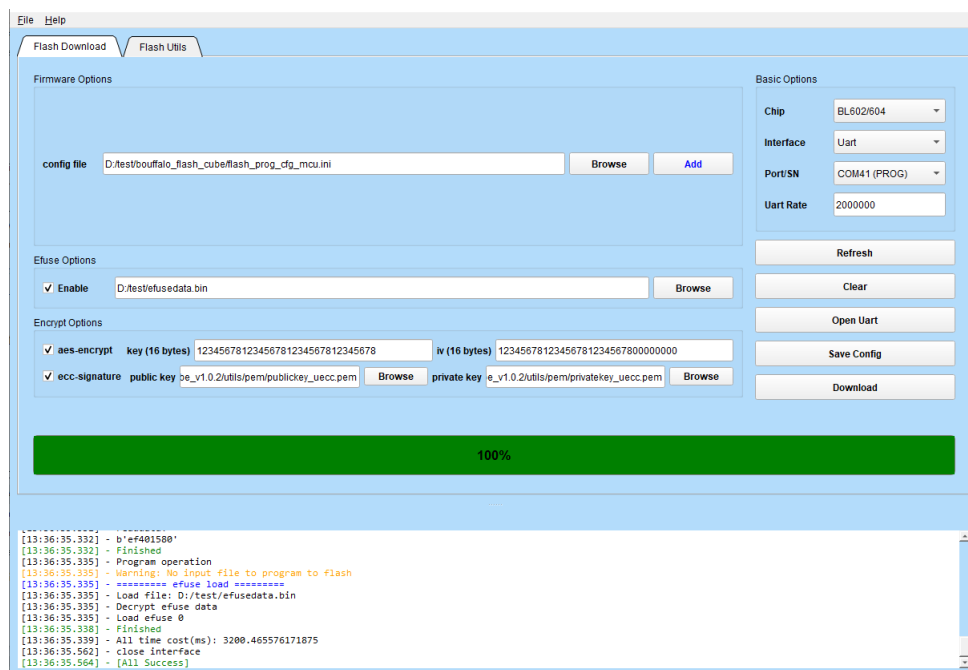


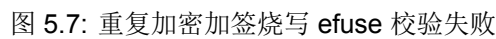
图 5.6: 加密加签烧写成功

5.6 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, 当第二次写 efuse 同时 factory_mode = true 的时候, 会显示 efuse 校验失败, 但实际烧写成功。



再次烧写重复加密加签的芯片，修改 `factory_mode = false` 不进行校验烧写，则会直接显示烧写成功。



5.7 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中 `erase` 参数控制。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 calcd 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 calcd 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 calcd 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 calcd 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 calcd by host: fd6af18fc4aaf2807277cac767ca19d12af7b55f5ecbb8902ef28bc2430524aa
[14:31:22.264] - xip mode Verify
```

图 5.9: 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 5.10: 烧写前全擦除

5.8 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。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，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 5.11: 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
```

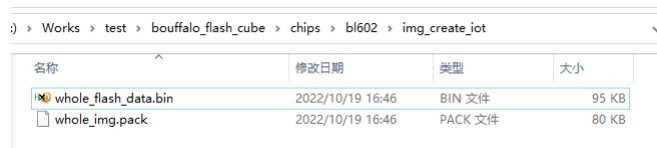
图 5.12: IOT 页面的 skip 功能

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。

5.9 生成完整的镜像文件

在每次烧写时，工具会按地址将烧录文件排序并拼接成一个完整的 `whole_flash_data.bin` 文件，方便下次直接使用。同时会将烧录文件和配置文件打包压缩成 `whole_img.pack`，在量产工具中可以直接导入使用。

以 BL602 为例，生成的 `whole_flash_data.bin` 和 `whole_img.pack` 存放于“chips/bl602/img_create_iot”目录下。





名称	修改日期	类型	大小
 whole_flash_data.bin	2022/10/19 16:46	BIN 文件	95 KB
 whole_img.pack	2022/10/19 16:46	PACK 文件	80 KB

图 5.13: 生成的 Whole_img 镜像

6.1 自定义的功能配置以用户导入为准

用户导入的配置文件包含多种功能配置，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` 功能以用户导入的配置文件中的定义为准，在烧写时会根据用户的配置更新到 `eflash_loader_cfg.ini` 文件中，用于 `whole_img.pack` 的生成。实际使用的仍是用户导入的配置选项值。

6.2 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

6.3 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。

6.4 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!。

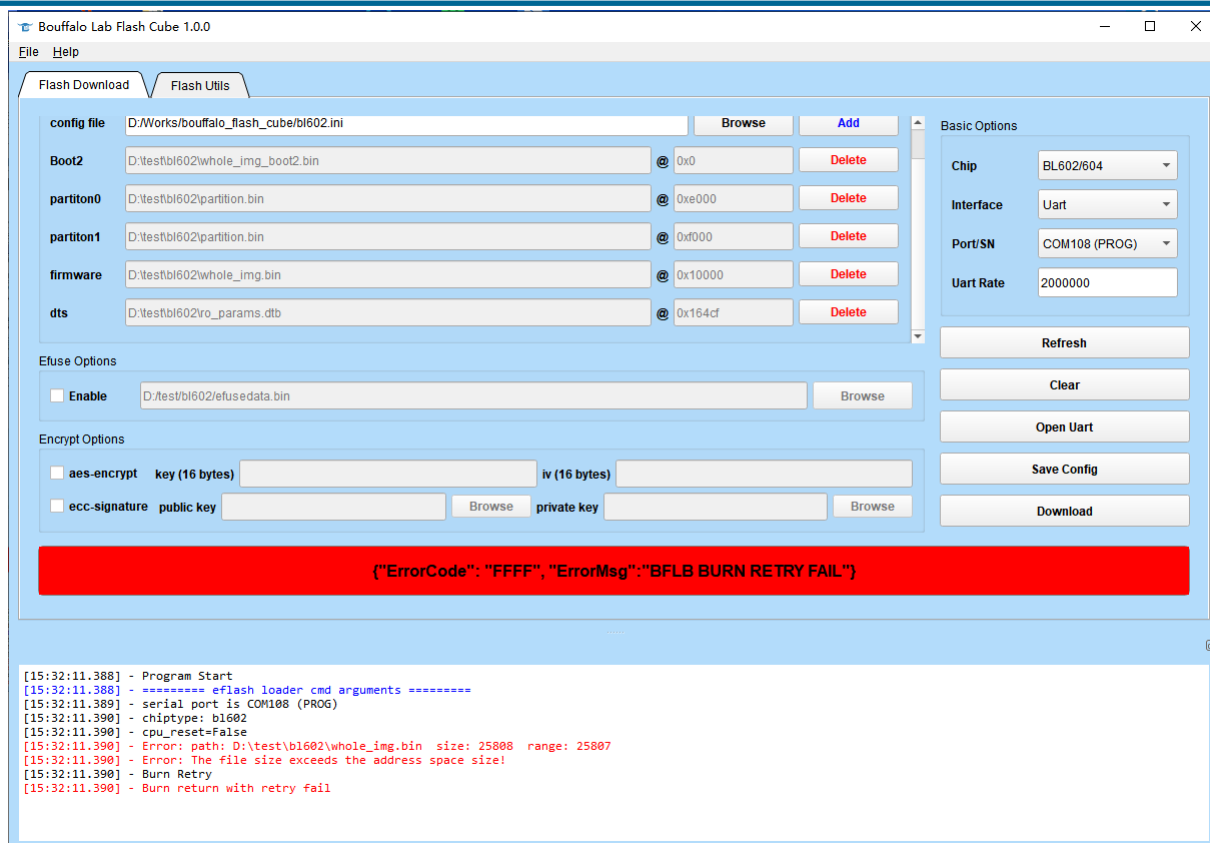


图 6.1: 固件大小超出错误

表 7.1: 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