



Dev Cube

使用说明

Version: 1.6

Copyright @ 2023

www.bouffalolab.com

1	Dev Cube 简介	4
2	镜像组成	5
3	IOT 程序下载	6
3.1	配置程序下载方式	6
3.2	配置下载文件	7
3.3	下载程序	8
4	MCU 程序下载	10
4.1	配置固件下载方式	10
4.2	配置镜像参数	11
4.3	配置高级镜像参数	12
4.4	下载程序	13
5	设置硬件安全参数	15
5.1	配置程序下载方式	16
5.2	配置密钥参数	16
6	Flash 调试助手	17
6.1	配置程序下载方式	18
6.2	读擦 Flash 内容	18
6.3	读写寄存器内容	19
7	高级功能	20
7.1	自定义 IOT 烧写界面	20
7.2	支持 ISP 烧写模式	22
7.3	支持压缩烧写	23
7.4	支持 erase skip 功能	23
7.5	界面中的烧录文件是否烧写可灵活配置	25
8	注意事项	27
8.1	分区表中每个分区的名称最大支持 7 个英文字符	27

8.2	BL808/BL606P/BL616 的 IOT 界面不提供 chip erase 选项	28
8.3	BL808/BL606P/BL616 的烧写晶振类型默认为 auto，即自动获取晶振类型	28
8.4	固件超出分区表中定义的 size0 时会出错	28
8.5	压缩镜像超出分区表中定义的 size1 时不会生成 ota 文件	29
8.6	芯片支持的加密方式	30
8.7	Efuse 数据文件支持 CRC 校验	30
8.8	增加 Flash size 超出的检查	31
8.9	BL602/BL702 支持不填写密钥签名烧写已经加密加签的板子	31
9	修改记录	34

Dev Cube 是博流提供的芯片集成开发工具，包含 IOT 程序下载、MCU 程序下载和 RF 性能测试三种功能。本文档主要介绍 IOT 和 MCU 程序下载相关配置，RF 性能测试请参考《射频性能测试使用手册》。

Dev Cube 提供用户下载应用程序的功能，并且支持配置时钟、flash 等参数，可根据用户需求对程序进行加密、添加签名，还具备烧录用户资源文件、分区表等功能。

Dev Cube 的主要功能如下：

1. 支持 IOT 应用程序和 MCU 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 和 JLink, 下载速度可配
5. 支持对程序镜像的 AES 加密以及签名功能

用户可以通过 [Bouffalo Lab Dev Cube](#)，获取最新版本的 Dev Cube。双击解压后文件夹中的 BLDevCube.exe，在 Chip Selection 对话框中选择对应的芯片型号，点击 Finish 进入 Dev Cube 主界面。

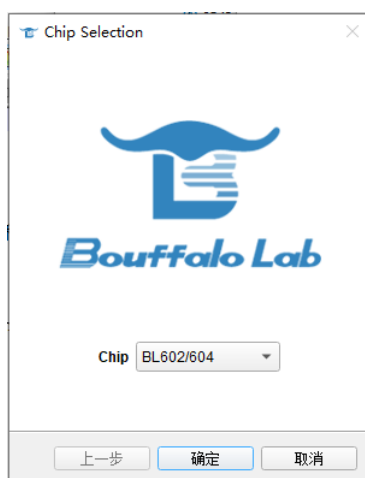


图 1.1: 芯片选择界面

无论是 IOT 程序还是 MCU 程序，它们的镜像组成结构是相同的，均如下图所示：

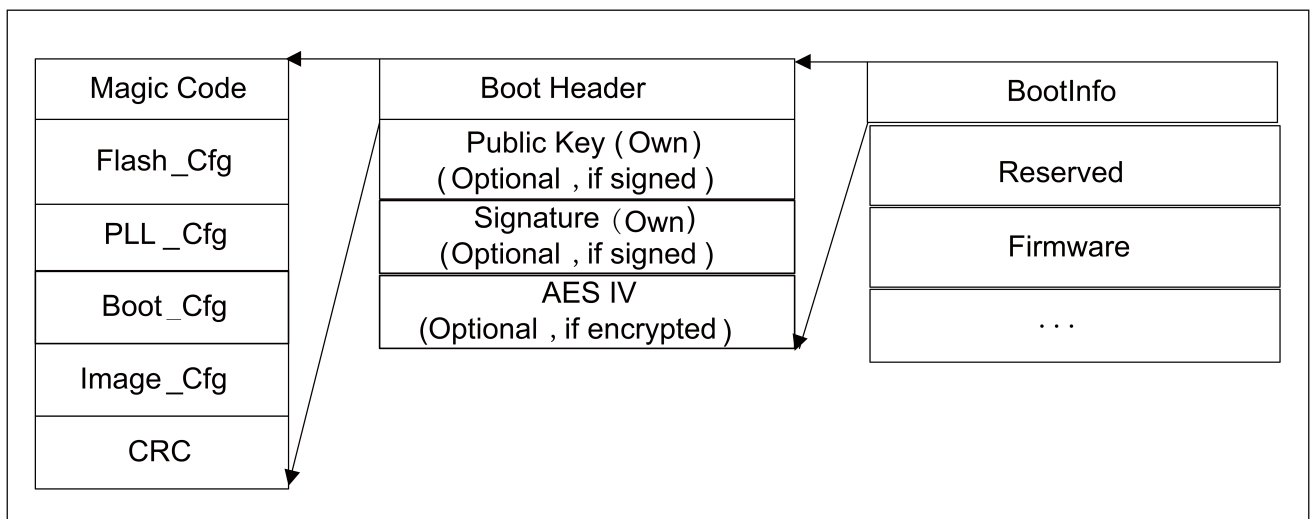


图 2.1: 启动镜像组成结构

启动镜像主要由两部分组成：

- **BootInfo** 主要包含 **BootInfo** 的 **Magic Code**，**Flash** 的配置信息，**PLL** 配置信息，启动参数信息以及镜像配置信息等
- **Firmware** 固件，即应用程序代码

如果只下载应用程序是无法使芯片正常工作的，必须要将启动信息 **BootInfo** 下载到指定位置。以单核下载为例，需要根据硬件电路的实际参数，将 **XTAL**、**PLL**、**Flash** 等配置信息烧录到 **Bootinfo Addr** 对应的地址中，将应用程序编译后的 **bin** 文件烧录到 **Image Addr** 对应的地址中。

在首行菜单中选择 IOT 选项，会进入 IOT 程序下载界面，主要分为程序下载方式配置和烧录文件配置两部分。

3.1 配置程序下载方式

• 配置参数包括：

- **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，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
- **Chip Erase:** 默认设置为 False，下载时按照烧录地址和内容大小进行擦除，选择 True 时，在程序烧录之前会将 Flash 全部擦除
- **Xtal:** 用于选择板子所使用的晶振类型

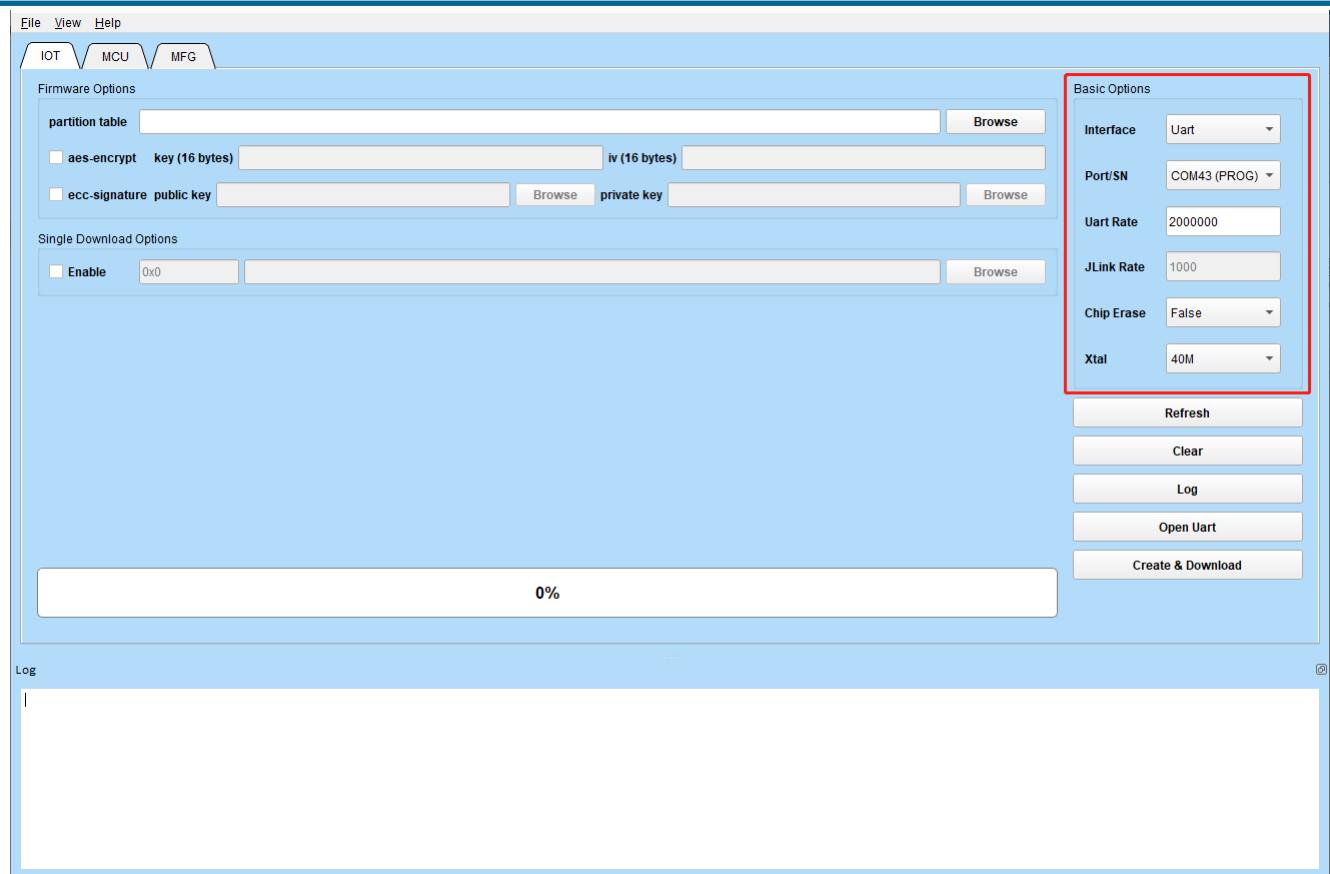


图 3.1: IOT 程序下载方式选择界面

3.2 配置下载文件

IOT 界面默认使用支持动态分区表的方式，同时兼容之前旧的分区表配置。动态分区表为配置文件 V2 版本，支持根据分区配置文件自动生成烧录界面的选择，当通过 PT Table 选项选择一个分区配置文件以后，工具会根据分区配置文件中的分区表自动生成烧写界面。

- 配置参数包括：
 - **partition table**: 使用根目录下对应芯片型号 **partition** 文件夹中的分区表, 分区文件主要是根据 **Flash** 大小确定，默认选择 **2M** 的分区表配置文件
 - **boot2**: 它是系统启动后运行的第一个 **Flash** 程序，负责建立安全环境，并引导主程序运行，使用根目录下对应芯片型号 **builtin_imgs** 文件夹中的 **Boot2 Bin** 文件
 - **firmware**: 用户编译生成的 **bin** 文件，这里选择生成的 **helloworld.bin**
 - **media/romfs**: **Media** 和 **Romfs** 二选一，如果勾选 **Media**，选择的是文件，如果勾选 **Romfs**，则选择的是文件夹
 - **mfg**: 选择 **MFG** 文件，**MFG** 文件是 **RF** 产测时候使用的应用程序，根据晶振类型，选择根目录下对应芯片型号 **builtin_imgs/mfg** 文件夹中的 **mfg bin** 文件

- **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
- **Single Download Options:** 勾选 **Enable** 后可下载单个 **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

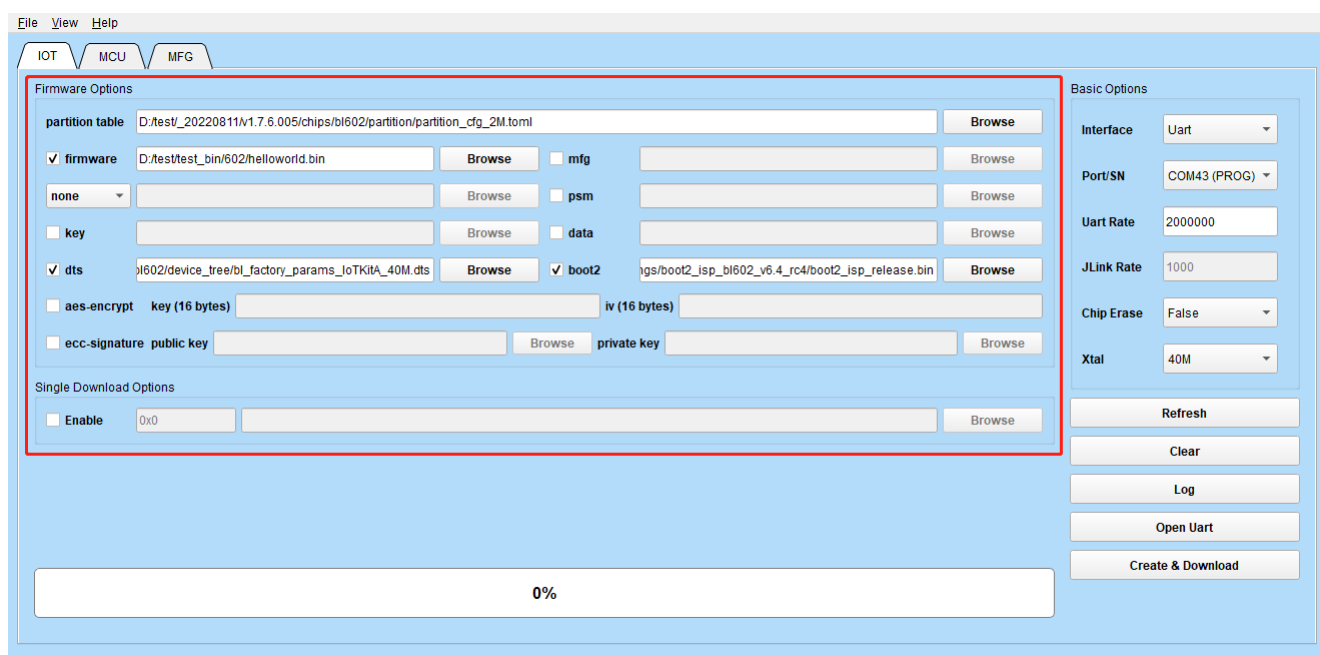


图 3.2: 烧录文件选择界面

3.3 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 **Create&Download**，工具即可自动生成应用程序镜像和启动参数配置文件并开始烧写配置的几个文件。出现下图 **log** 信息，程序下载成功

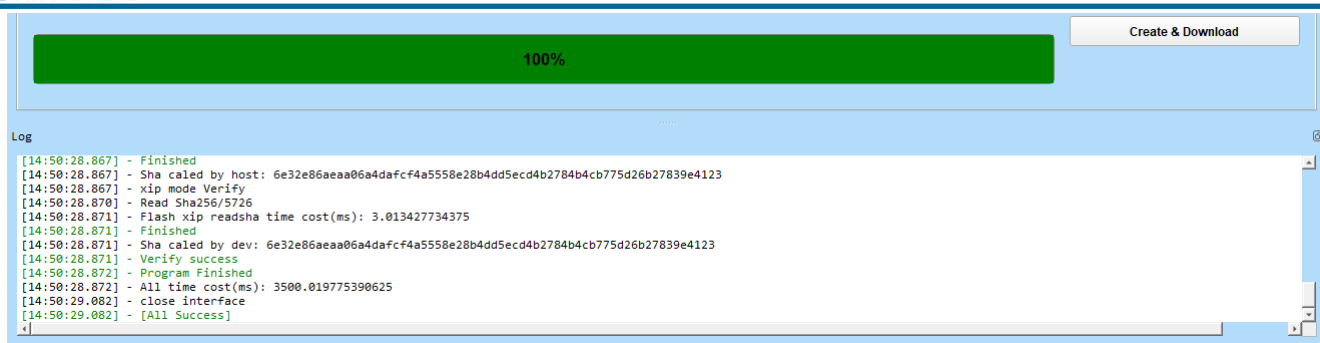


图 3.3: IOT 下载程序

注解：若没有连接板子，只需生成应用程序镜像和启动参数配置文件，也是点击 **Create&Program** 按钮

- 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, 应用程序即可运行起来.

下图是 hello world 程序运行起来的效果.

```
[14:52:10.515] - Open COM43 Success
[14:52:10.516] - 
[14:52:10.518] - 
[14:52:10.519] - BouffaloLab
[14:52:10.521] - 
[14:52:10.522] - 
[14:52:10.523] - 
[14:52:10.524] - Build:16:21:09,Aug 13 2021
[14:52:10.525] - Copyright (c) 2021 Bouffalolab team
[14:52:10.526] - dynamic memory init success,heap size = 176 Kbyte
[14:52:10.526] - hello world!
[14:52:10.713] - hello world!
[14:52:10.913] - hello world!
[14:52:11.112] - hello world!
[14:52:11.312] - hello world!
[14:52:11.512] - hello world!
[14:52:11.712] - hello world!
[14:52:11.912] - hello world!
[14:52:12.112] - hello world!
[14:52:12.312] - hello world!
[14:52:12.513] - case success
```

图 3.4: hello world 程序运行效果

在首行菜单中选择 **MCU** 选项，会进入 **MCU** 程序下载界面，主要分为固件下载方式配置、镜像参数配置和高级镜像参数配置三个主要部分。

4.1 配置固件下载方式

- 配置参数包括：
 - **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**
 - **Chip Erase**: 默认设置为 **False**，下载时按照烧录地址和内容大小进行擦除，选择 **True** 时，在程序烧录之前会将 **Flash** 全部擦除
 - **Xtal**: 用于选择板子所使用的晶振类型

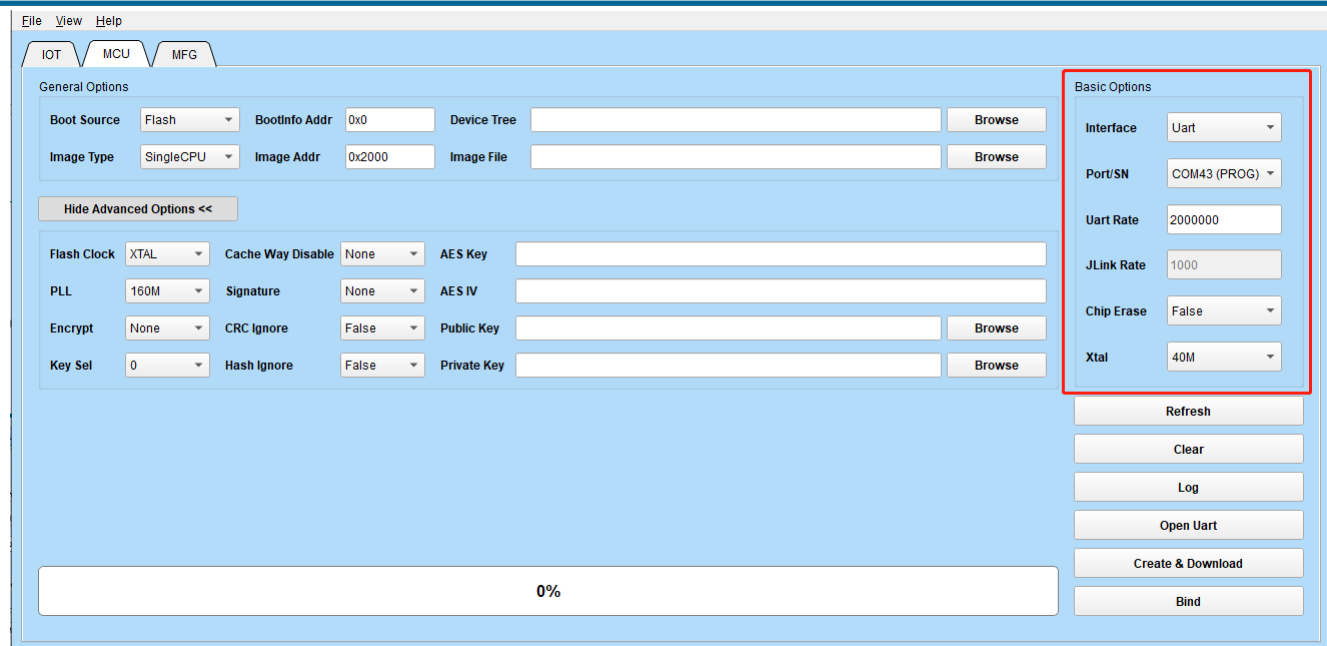


图 4.1: MCU 固件下载方式选择界面

4.2 配置镜像参数

• 配置参数包括:

- **Boot Source:** 默认为 Flash, 只有在需要生成从 UART 或者 SDIO 启动镜像的时候才需要选择 UART/SDIO
- **BootInfo Addr:** Bootinfo 启动参数的存放地址, 对于单核程序, 填写 0x0, 对于双核的 CPU0 镜像, 填写 0x0, 对于双核的 CPU1 镜像, 填写 0x1000
- **Image Type:** SingleCPU 用于生成单核的镜像, CPU0 用于生成双核中 CPU0 的镜像, CPU1 用于生成双核中 CPU1 的镜像, Boot2 用于生成 Boot2 镜像, RAW 用于下载用户自定义的原始资源文件
- **Image Addr:** 应用程序的存放地址, 建议填写 0x2000 或者 0x2000 以后的地址
- **Image File:** 选择应用程序的 Bin 文件或者用户的资源文件

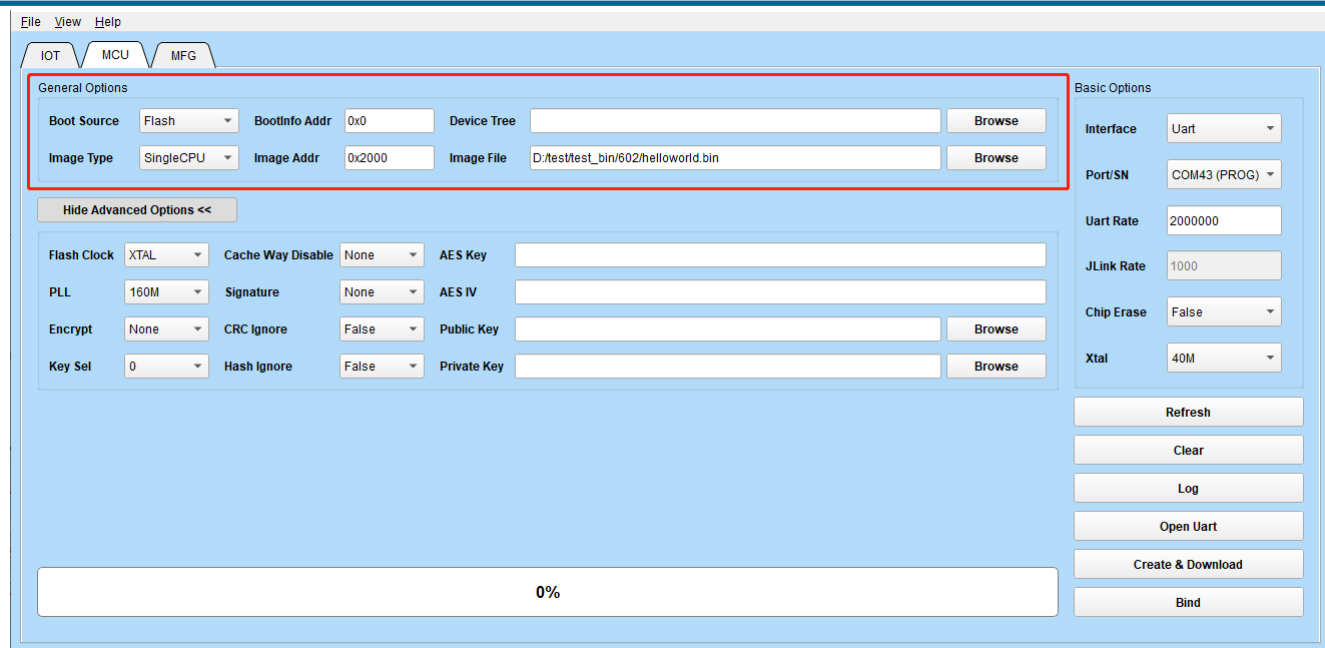


图 4.2: 镜像参数选择界面

4.3 配置高级镜像参数

- 当点击 Show Advanced options 时，会展开高级镜像配置，可配置的参数包括：
 - Flash Clock: 用于设定 Flash 的时钟
 - PLL : PLL 时钟配置，根据不同类型的芯片，选择对应的时钟
 - Cache Way Disable : L1C Cache 的 4 条 way 设定，默认为 none, 即使能全部的 4 条 way
 - Signature : 选择是否对程序镜像进行 ECC 签名
 - Crc Ignore : 是否需要 CRC 校验。当参数选择 False 时，开启 Boot Info 的 CRC 校验；反之, 不设置 Boot Info 的 CRC 校验
 - Hash Ignore : 是否需要做镜像的完整性 Hash 校验。当参数选择 False 时需要做 Hash 校验；反之, 不做镜像的完整性校验
 - Encrypt : 选择加密方式，使能对程序镜像的加密功能。使能加密功能后, 需要根据 AES 加密方式在 AES Key 和 AES IV 中输入对应的数值

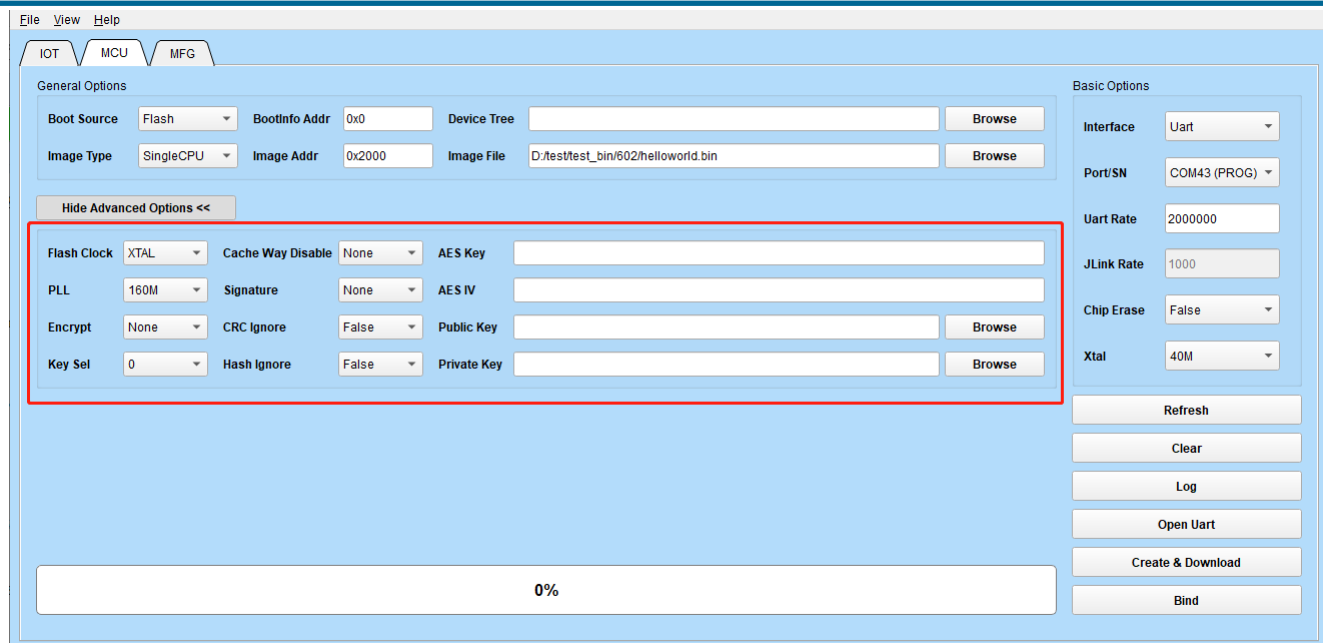


图 4.3: 高级镜像参数选择界面

4.4 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 Create&Program, 工具即可自动生成应用程序镜像和启动参数配置文件并开始烧写。出现下图 log 信息, 程序下载成功

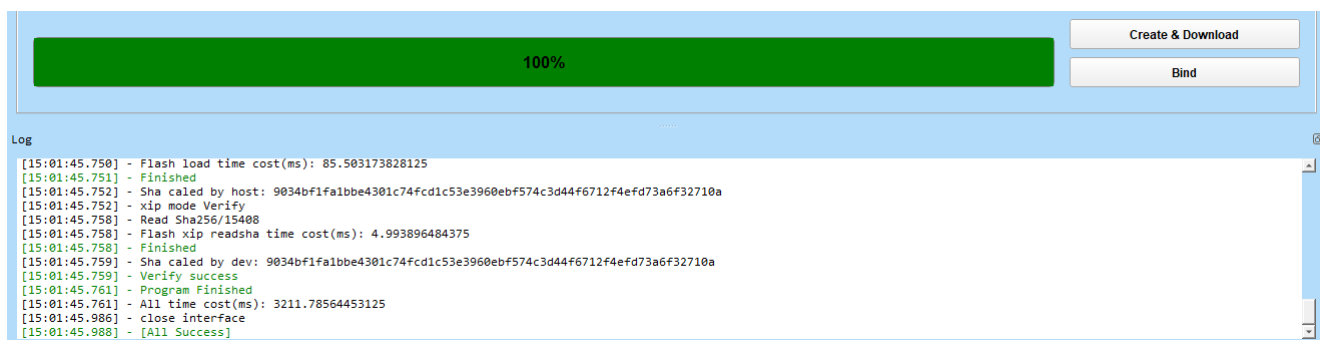


图 4.4: 下载程序

注解: 若没有连接板子, 只需生成应用程序镜像和启动参数配置文件, 也是点击 Create&Program 按钮

- 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, 应用程序即可运行起来

下图是 hello world 程序运行起来的效果.

```
[15:02:27.748] - Open COM43 Success
[15:02:27.749] -
[15:02:27.751] -
[15:02:27.752] - [B]o[u]ffalo[l]ab
[15:02:27.753] - [B]o[u]ffalo[l]ab
[15:02:27.754] - [B]o[u]ffalo[l]ab
[15:02:27.755] - [B]o[u]ffalo[l]ab
[15:02:27.756] - Build:16:21:09,Aug 13 2021
[15:02:27.757] - Copyright (c) 2021 Bouffalolab team
[15:02:27.758] - dynamic memory init success,heap size = 176 Kbyte
[15:02:27.758] - hello world!
[15:02:27.946] - hello world!
[15:02:28.145] - hello world!
[15:02:28.345] - hello world!
[15:02:28.545] - hello world!
[15:02:28.745] - hello world!
[15:02:28.945] - hello world!
[15:02:29.145] - hello world!
[15:02:29.345] - hello world!
[15:02:29.545] - hello world!
[15:02:29.746] - case success
```

图 4.5: hello world 程序运行效果

设置硬件安全参数

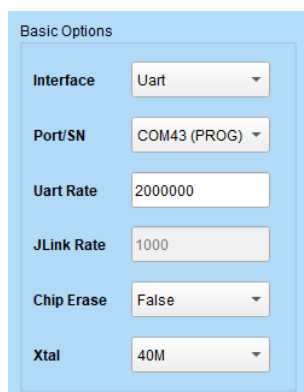
在 View 菜单中选择 Show Advanced Page 选项，点击 Security 选项，进入硬件安全参数配置界面。配置信息包含固件下载方式配置、AES 模式配置和密钥配置。

The screenshot shows the 'Security' configuration page. The 'AES Mode' is set to 'AES128'. The 'Public Key Cfg' field is empty, with a 'Browse' button and a 'Write Lock' checkbox. The 'AES Key' field is empty, with 'Write Lock' and 'Read Lock' checkboxes. The 'Basic Options' section on the right includes settings for 'Interface' (Uart), 'Port/SN' (COM43 (PROG)), 'Uart Rate' (2000000), 'JLink Rate' (1000), 'Chip Erase' (False), and 'Xtal' (40M). Below these are buttons for 'Refresh', 'Clear', 'Program', and 'Create'. At the bottom, there is a 'Log' section with a large text area.

图 5.1: 硬件参数配置界面

5.1 配置程序下载方式

- 配置参数包括：
 - 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
 - Chip Erase:** 默认设置为 False，下载时按照烧录地址和内容大小进行擦除，选择 True 时，在程序烧录之前会将 Flash 全部擦除
 - Xtal:** 用于选择板子所使用的晶振类型



The image shows a 'Basic Options' configuration window with the following settings:

Parameter	Value
Interface	Uart
Port/SN	COM43 (PROG)
Uart Rate	2000000
JLink Rate	1000
Chip Erase	False
Xtal	40M

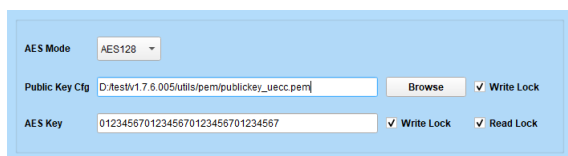
图 5.2: 下载方式界面

5.2 配置密钥参数

若想要给芯片加密，除了在下载程序时，使用 Image 功能中的 AES 对芯片进行软件加密，还需要进行硬件加密。

- 在 AES Mode 中选择对应的加密模式，本例中选择 AES128
- 在 Public Key Cfg 中选择公钥的 PEM 文件，本例选择 utils/pem/publickey_uecc.pem
- AES Key 填写和软件加密相同的值

点击 Create，生成 Efuse 文件，点击 Program，烧录 Efuse 文件。



The image shows a key configuration window with the following settings:

Parameter	Value	Write Lock	Read Lock
AES Mode	AES128		
Public Key Cfg	D:\test\v1.7.6.005\utils\pem\publickey_uecc.pem	<input checked="" type="checkbox"/>	
AES Key	01234567012345670123456701234567	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

图 5.3: 密钥参数配置界面

在 View 菜单中选择 Show Advanced Page 选项，点击 Flash 选项，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。

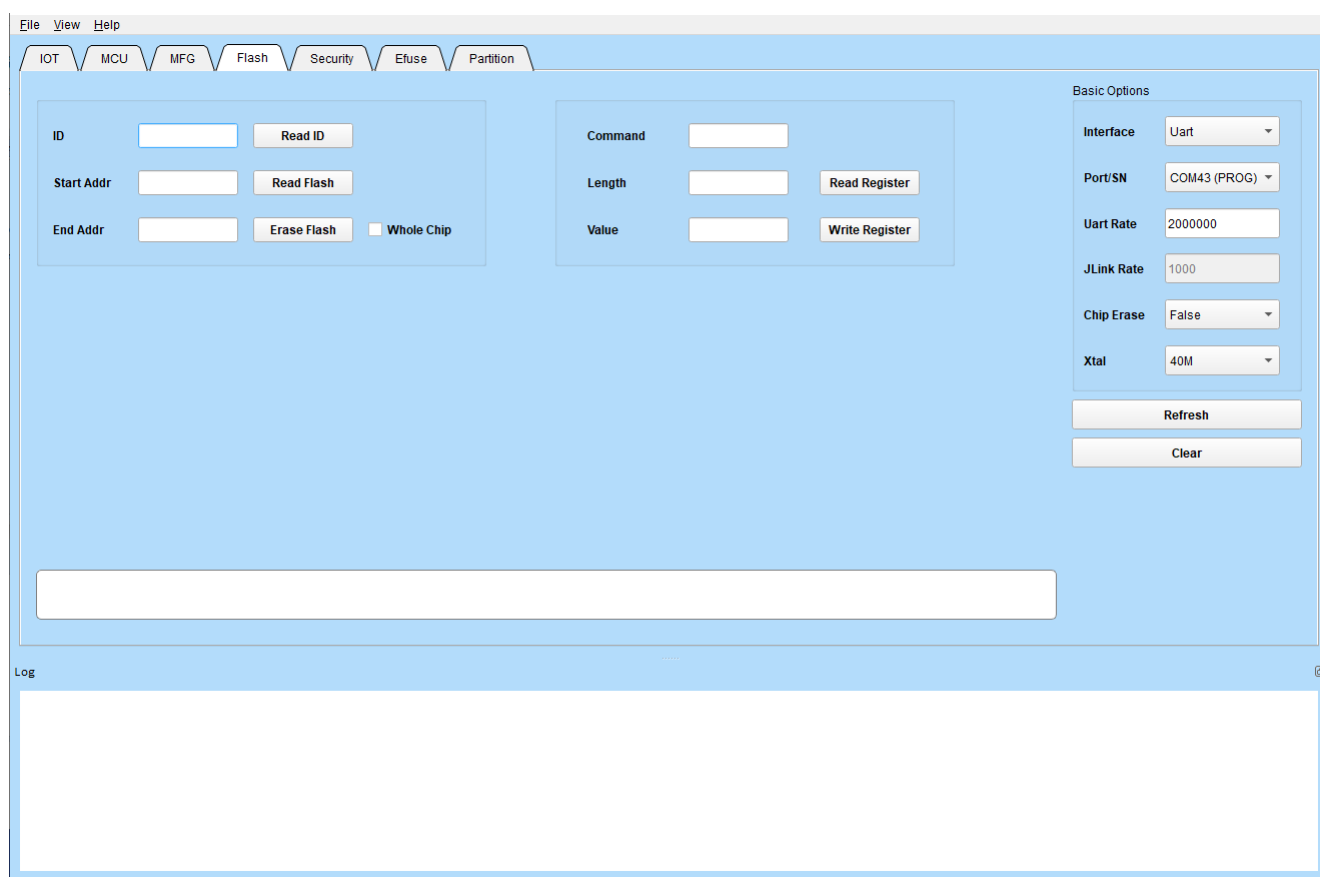
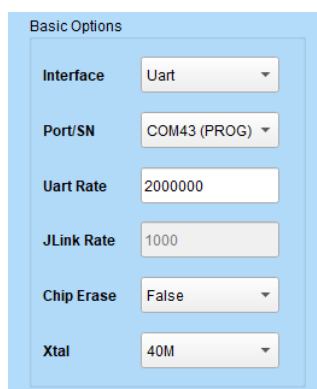


图 6.1: Flash 调试助手界面

6.1 配置程序下载方式

- 配置参数包括：
 - 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
 - Chip Erase:** 默认设置为 False，下载时按照烧录地址和内容大小进行擦除，选择 True 时，在程序烧录之前会将 Flash 全部擦除
 - Xtal:** 用于选择板子所使用的晶振类型

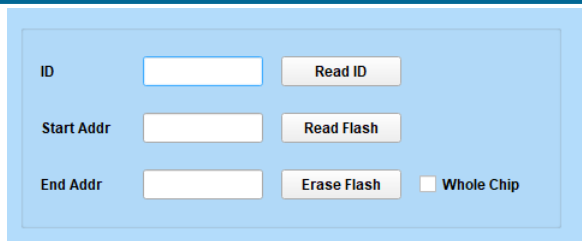


Basic Options	
Interface	Uart
Port/SN	COM43 (PROG)
Uart Rate	2000000
JLink Rate	1000
Chip Erase	False
Xtal	40M

图 6.2: 下载方式界面

6.2 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 flash.bin 文件中，在工具的根目录下
- 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 reading and erasing Flash memory. It contains three input fields for ID, Start Addr, and End Addr. Next to each field is a button: Read ID, Read Flash, and Erase Flash. There is also a checkbox labeled 'Whole Chip'.

图 6.3: 读擦 Flash 界面

6.3 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 reading and writing registers. It contains three input fields: Command, Length, and Value. Next to the Length field is a button labeled 'Read Register'. Next to the Value field is a button labeled 'Write Register'.

图 6.4: 读写寄存器界面

工具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 自定义 IOT 烧写界面

工具支持动态分区表自定义 IOT 烧录界面。即动态分区表中每个分区和烧写地址可以自行设定，按照需求定制 IOT 烧录界面。本例中使用 BL808 的 8M 分区表，在 IOT 界面增加 **test1 ~ test4** 的烧写选项。首先修改分区表文件，打开工具目录下的“chips/bl808/partition/partition_cfg_8M.toml”文件，自定义 **test1 ~ test4** 字段及其参数。

```
[[pt_entry]]
type = 3
name = "test1"
device = 0
address0 = 0x610000
size0 = 0x2000
address1 = 0
size1 = 0
# compressed image must set len,normal image can left it to 0
len = 0
# If header is 1, it will add the header.
header = 0
# If header is 1 and security is 1, It will be encrypted.
security = 0

[[pt_entry]]
type = 4
name = "test2"
device = 0
address0 = 0x620000
size0 = 0x10000
address1 = 0
size1 = 0
# compressed image must set len,normal image can left it to 0
len = 0
# If header is 1, it will add the header.
header = 0
# If header is 1 and security is 1, It will be encrypted.
security = 0

[[pt_entry]]
type = 5
name = "test3"
device = 0
address0 = 0x630000
size0 = 0x10000
```

图 7.1: 动态分区表自定义参数

修改完成后保存文件，在 IOT 界面选择修改后的分区表文件，IOT 界面会更新为如下图所示：

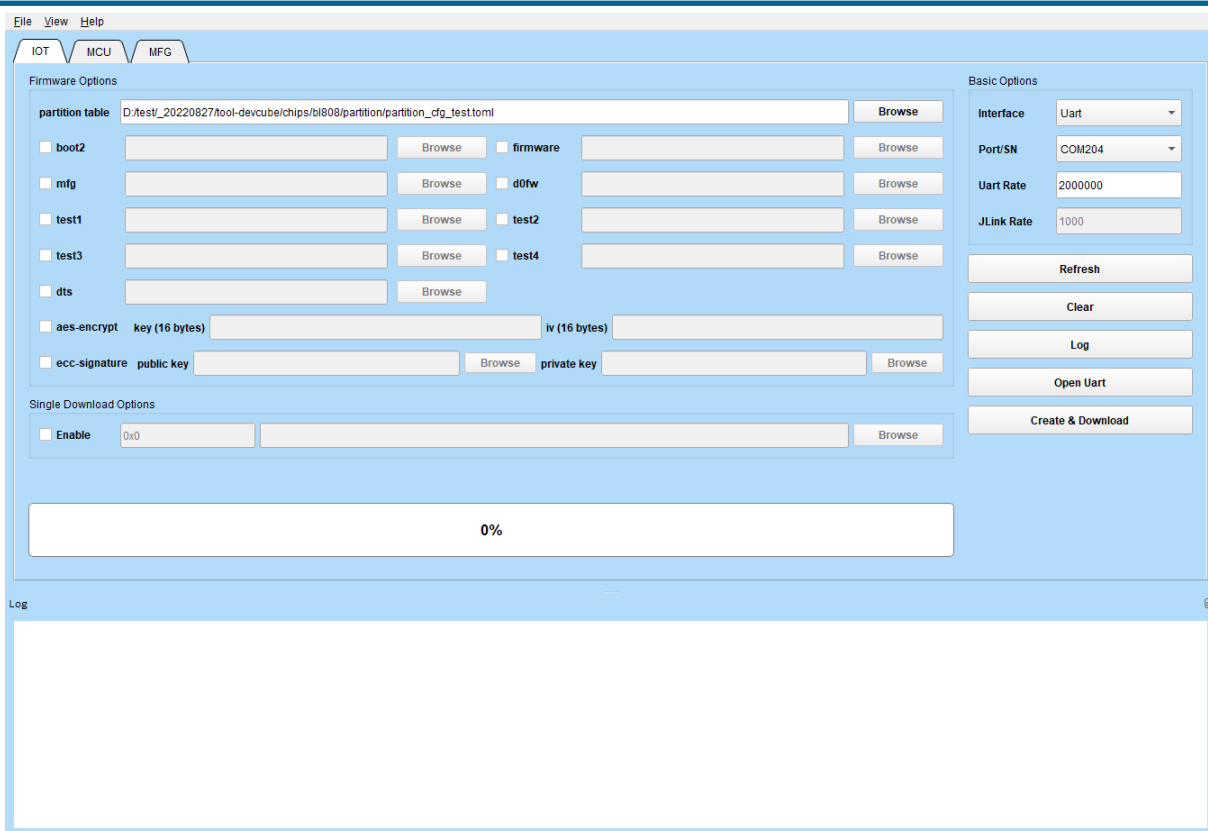


图 7.2: 动态分区表自定义 IOT 烧录界面

选择好烧录文件后，点击 **Create&Download**，工具会根据分区表的配置烧录到相应的 flash 地址中，如下图所示：

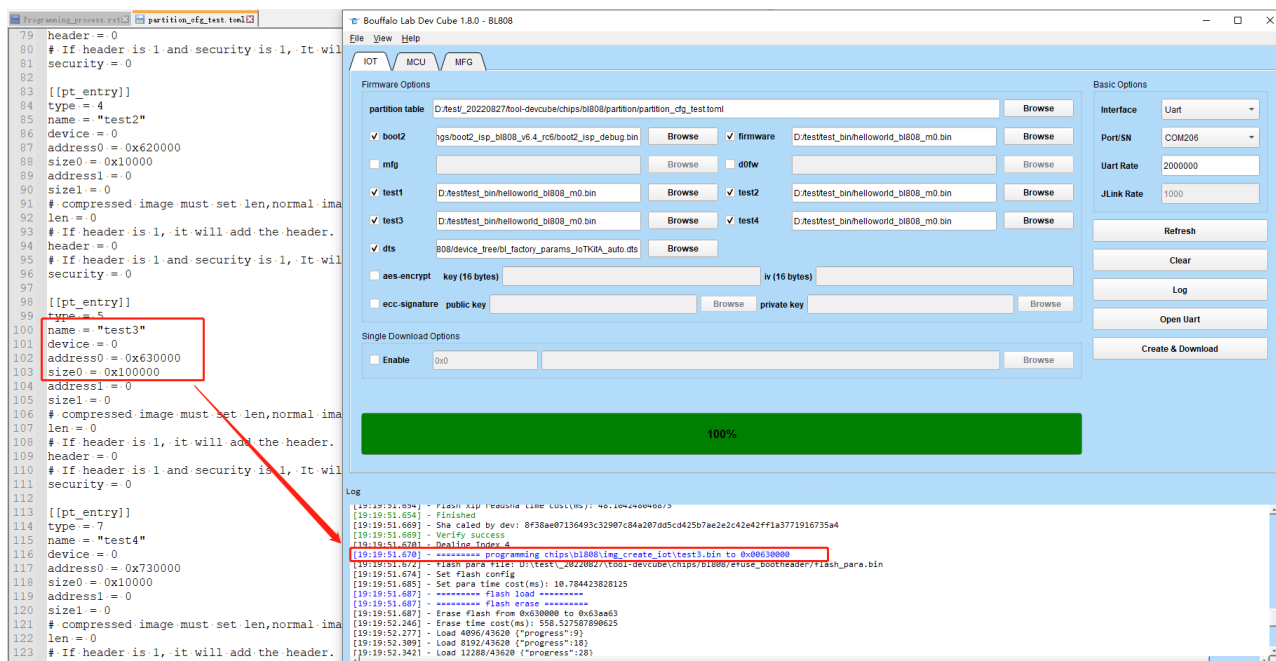


图 7.3: 自定义 IOT 界面的烧写 log

7.2 支持 ISP 烧写模式

ISP 即 In-System Programming，DevCube v1.7.4 及其更新的版本都支持 ISP 模式烧写，详细请参考文档“ISP_下载使用说明”。

打开工具目录下的”chips/xxx/eflash_loader/eflash_loader_cfg.ini”文件, xxx 代指芯片类型，其中 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。修改文件中的”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。

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”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。

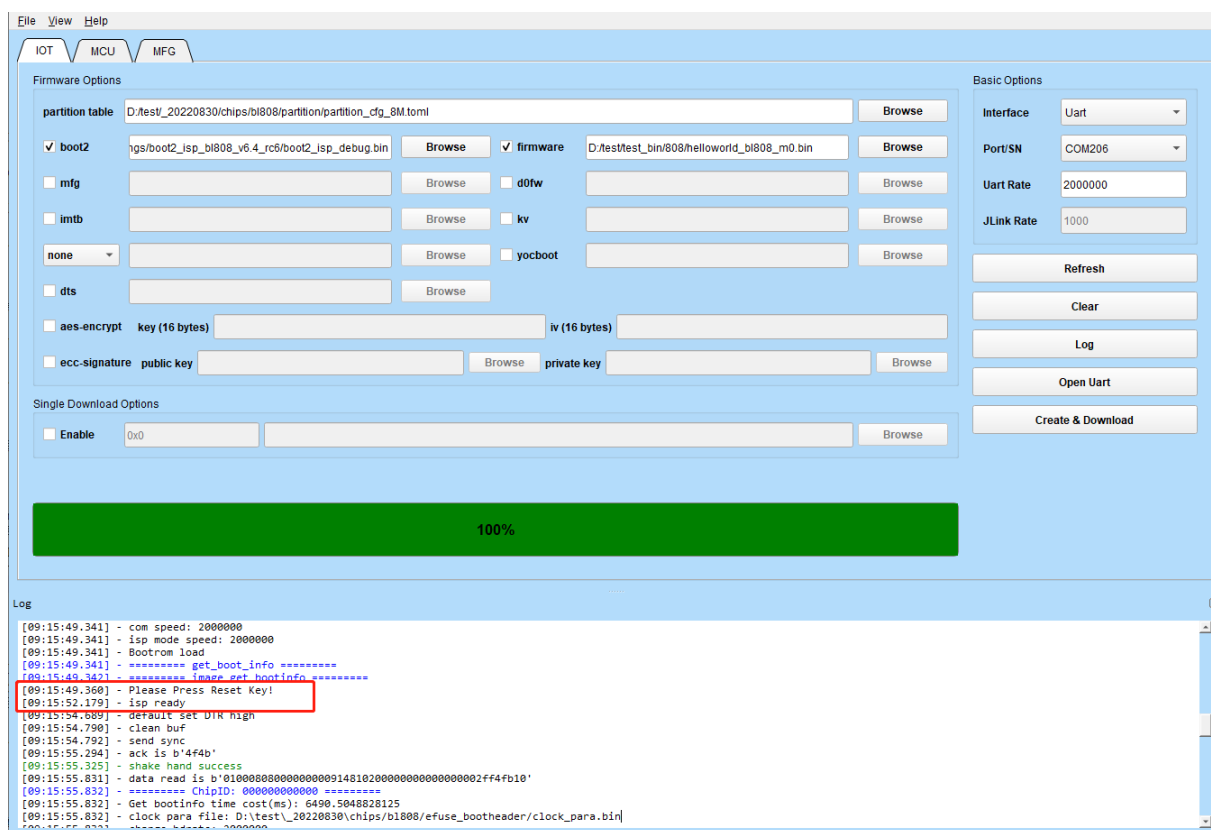


图 7.4: ISP 烧写模式

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。

打开工具目录下的”chips/xxx/eflash_loader/eflash_loader_cfg.ini”文件, xxx 代指芯片类型,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。

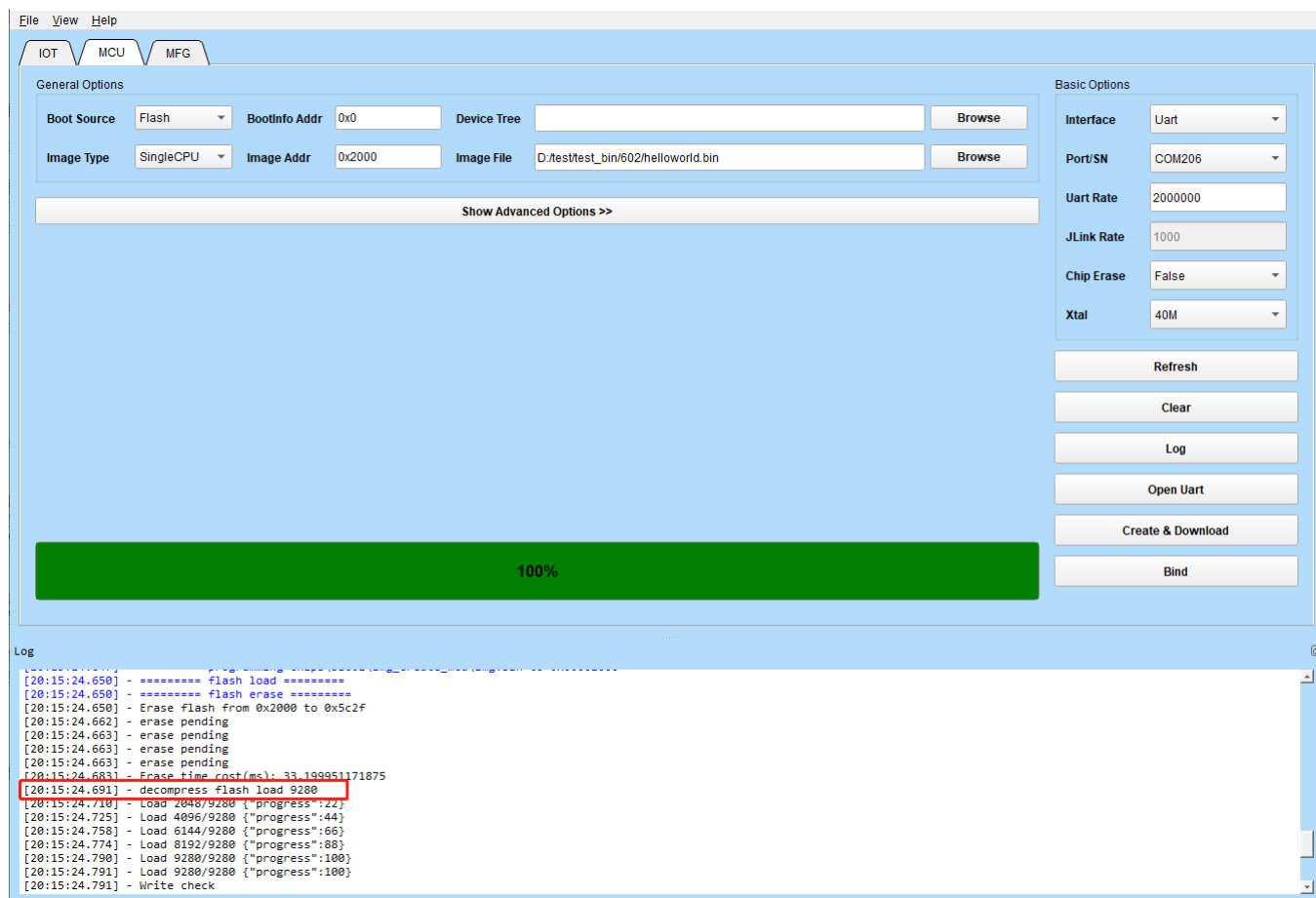


图 7.5: 压缩烧写模式

7.4 支持 erase 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。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: 首先打开工具目录下的”chips/xxx/eflash_loader/eflash_loader_cfg.ini” 文件, xxx 代指芯片类型, 修改其中的”skip_mode = 0x0, 0x0” 为”skip_mode = 0x11000, 0x1000”, 然后保存文件。烧录 log 如下图所示:

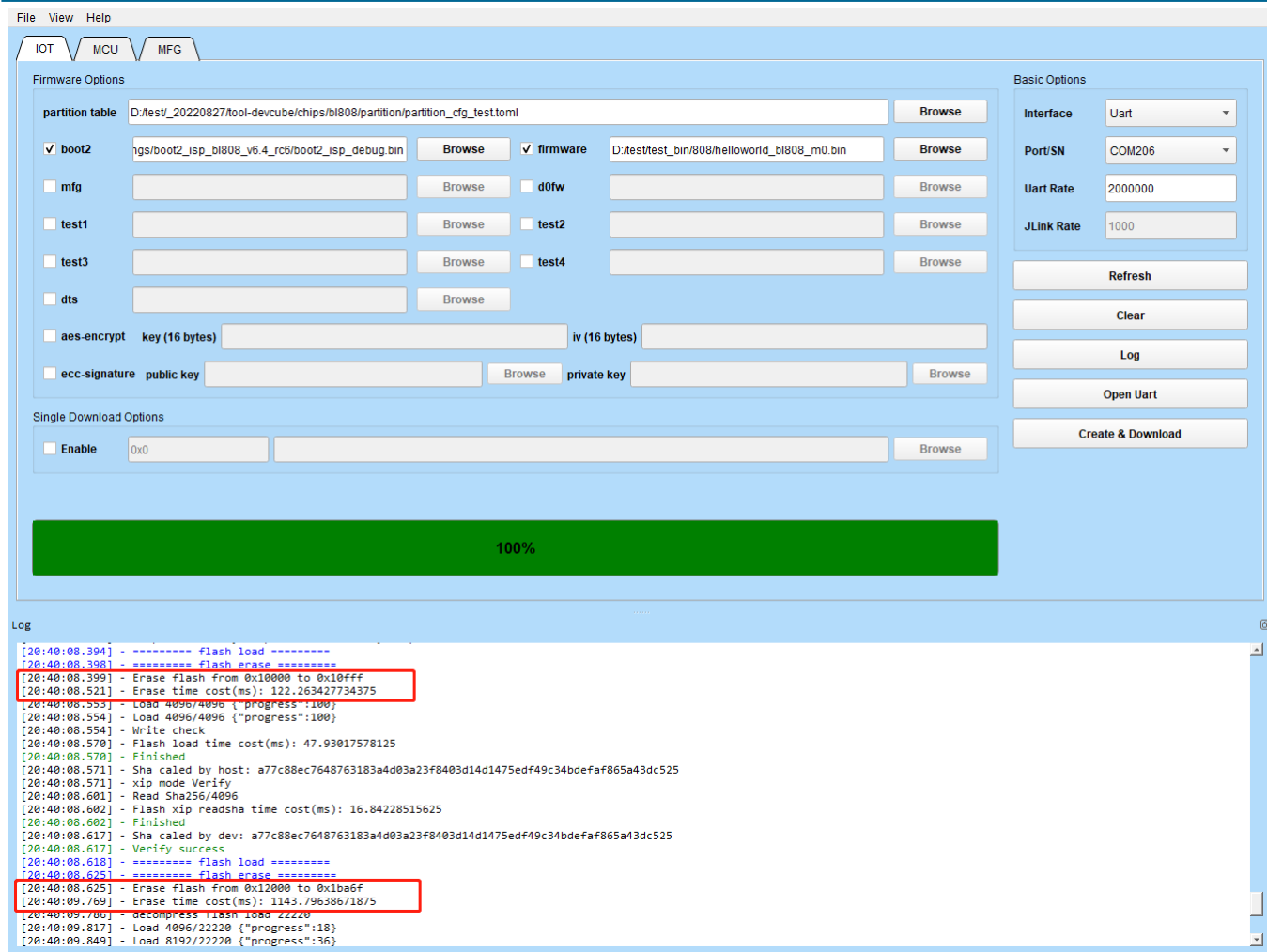


图 7.6: IOT 页面的 skip 功能

注解：skip 功能与 chip erase 功能冲突，所以不能同时配置。在工具上表现为烧写失败并提示 ErrorMsg: BFLB FLASH CHIPERASE CONFLICT WITH SKIP MODE

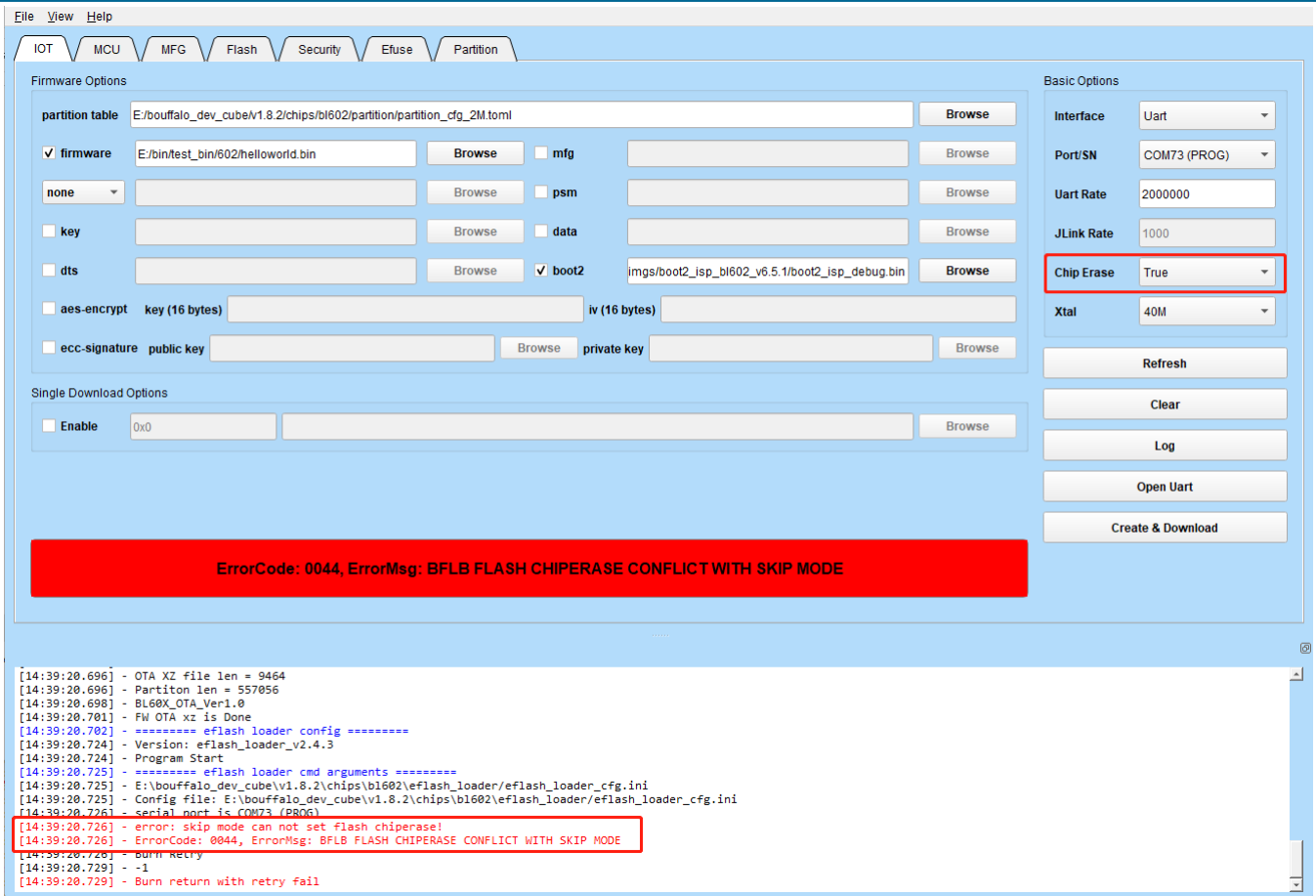


图 7.7: IOT 页面的 skip 功能与 chip erase 功能冲突

7.5 界面中的烧录文件是否烧写可灵活配置

某些情况下,需要界面中显示烧录文件,但不需要进行烧写,可以使用这个功能。打开工具目录下的”chips/xxx/partition/partition_cfg_8M.toml”文件,xxx 代指芯片类型,本例中使用 BL808 的 8M 分区表,Boot2 的配置如下:

```
[[pt_entry]]
type = 16
name = "Boot2"
device = 0
address0 = 0
size0 = 0xE000
address1 = 0
size1 = 0
# compressed image must set len,normal image can left it to 0
len = 0
# If header is 1, it will add the header.
header = 1
# If header is 1 and security is 1, It will be encrypted.
security = 1
```

图 7.8: 分区表的默认配置

修改配置中的”size0 = 0xE000”为”size0 = 0”,在 IOT 界面重新选择修改后的分区表文件,则界面中仍然会显示 boot2 的选项,但实际却不烧写,且烧写 log 中有黄色标记的提示”Warning: boot2 file is not programed to flash”。

File View Help

IOT MCU MFG

Firmware Options

partition table

D:\test\20220830\cube\chips\bl808\partition\partition_cfg_test.toml

Browse

☒ boot2

D:\test\20220830\cube\chips\bl808\partition\partition_cfg_test.toml

Browse

☐ mfg

Browse

☐ test1

Browse

☐ test3

Browse

☐ dts

Browse

☐ aes-encrypt

key (16 bytes)

iv (16 bytes)

☐ ecc-signature

public key

Browse

private key

Browse

firmware

D:\test\20220830\cube\chips\bl808\partition\partition_cfg_test.toml

Browse

☐ d0fw

Browse

☐ test2

Browse

☐ test4

Browse

Basic Options

Interface

Uart

Port/SN

COM204

Uart Rate

2000000

JLink Rate

1000

Refresh

Clear

Log

Open Uart

Create & Download

Single Download Options

☐ Enable

0x0

Browse

100%

Log

```

[17:48:31.485] - ===== Interface is Uart =====
[17:48:31.489] - eFlash loader bin is eflash_loader_auto.bin
[17:48:31.490] - ===== chip flash id: ef4016 =====
[17:48:31.496] - create partition bin, not new is True
[17:48:31.505] - Warning: boot2 file is not programmed to flash
[17:48:31.514] - Warning: boot2 file is not programmed to flash
[17:48:31.536] - Create bootheader using D:\test\20220830\cube\chips\bl808\img_create_iot\efuse_bootheader_cfg.ini
[17:48:31.537] - Updating data according to <D:\test\20220830\cube\chips\bl808\img_create_iot\efuse_bootheader_cfg.ini[BOOTHEADER_GROUP0_CFG]>
[17:48:31.540] - Created file len:352
[17:48:31.543] - Create bootheader using D:\test\20220830\cube\chips\bl808\img_create_iot\efuse_bootheader_cfg.ini
[17:48:31.543] - Updating data according to <D:\test\20220830\cube\chips\bl808\img_create_iot\efuse_bootheader_cfg.ini[BOOTHEADER_GROUP1_CFG]>
[17:48:31.546] - Created file len:352
[17:48:31.548] - Create efuse using D:\test\20220830\cube\chips\bl808\img_create_iot\efuse_bootheader_cfg.ini
[17:48:31.550] - Updating data according to <D:\test\20220830\cube\chips\bl808\img_create_iot\efuse_bootheader_cfg.ini[EFUSE_CFG]>
[17:48:31.553] - Created file len:256
[17:48:31.553] - ef_sw_usage_0 not exist

```

图 7.9: IOT 界面烧录文件不烧写页面

8.1 分区表中每个分区的名称最大支持 7 个英文字符

分区表中每个分区的 **name** 字段长度不应该超过 7 个英文字符，否则不能正确的生成分区表文件。以下图的 boot2mini 为例，因超过 7 个英文字符而烧写出错，有警告提醒：Warning: the length of boot2mini is exceeds the number 7 allowed。且最终因为没有找到分区表文件 partition.bin 而烧写出错，并提示：Please check your partition table file

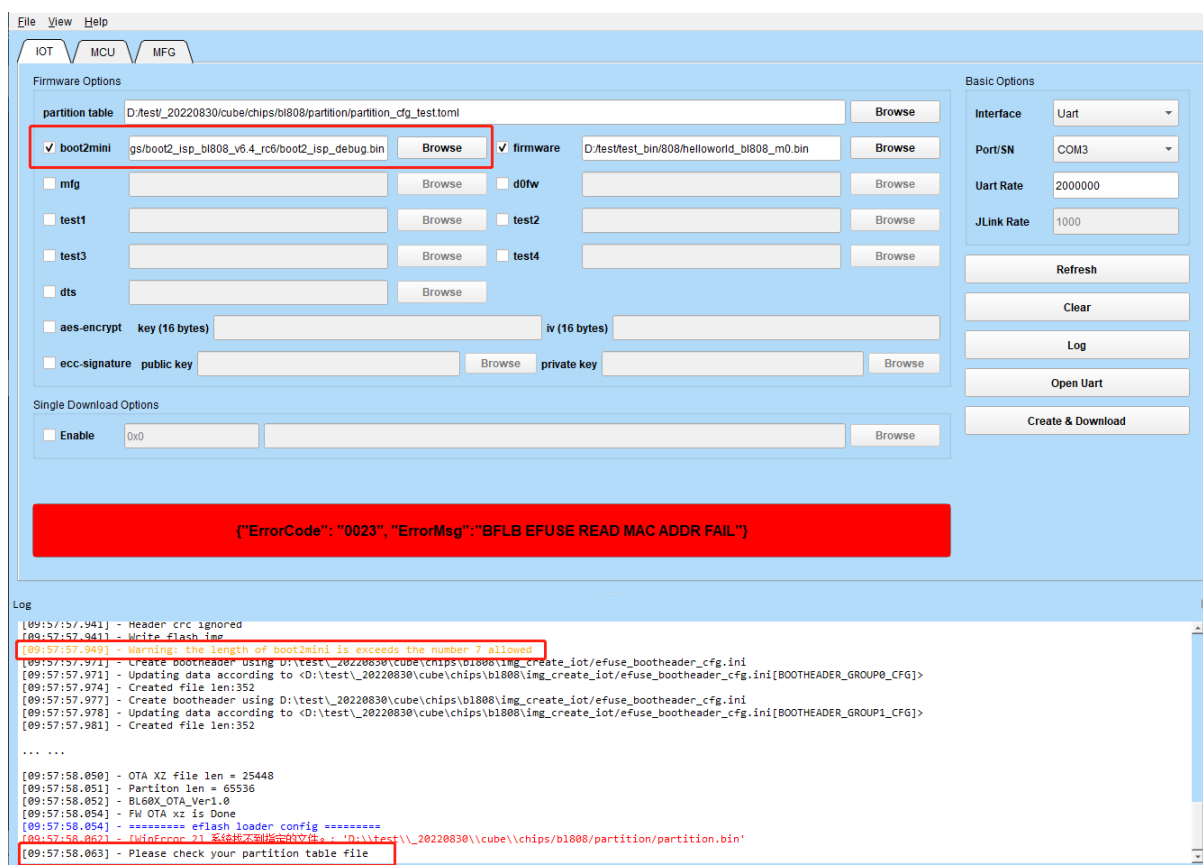


图 8.1: 分区名称超过 7 个英文字符的错误页面

8.2 BL808/BL606P/BL616 的 IOT 界面不提供 chip erase 选项

BL808/BL606P/BL616 的 IOT 界面不提供 chip erase 选项, 即使手动修改配置文件也无效。如果需要做 chip erase 功能, 可以切换到 Flash 页面做 chip erase, 或者切换到 mcu 界面勾选 chip erase 进行烧写。

8.3 BL808/BL606P/BL616 的烧写晶振类型默认为 auto, 即自动获取晶振类型

晶振类型的配置定义在“efuse_bootheader_cfg.ini”文件中, 打开工具中的“chips/xxx/img_create_iot/efuse_bootheader_cfg.ini”文件, xxx 代指芯片类型, 本例中使用 BL616 为例, xtal_type 为烧录时晶振类型的选择。默认值 7 为自动获取晶振类型, 用户也可以手动设定为其他晶振类型。

```
#####clk_cfg#####
clkcfg_magic_code = 0x47464350
#clkcfg_magic_code=0

#0:None,1:24M,2:32M,3:38.4M,4:40M,5:26M,6:RC32M
xtal_type = 7
#mcu_clk
0:RC32M;1:XTAL;2:aupll_div2;3:aupll_div1;4:wifipll_240M;5:wifipll
mcu_clk = 5
mcu_clk_div = 0
mcu_bclk_div = 0
mcu_pbclk_div = 3
```

图 8.2: 晶振类型的选择

8.4 固件超出分区表中定义的 size0 时会出错

IOT 界面的 firmware 选项用于选择需要烧录的固件, 当固件大于分区表中 FW 字段定义的 size0 时, 工具会检测到其固件大小超出, 并提示错误。

首先打开工具目录下的“chips/xxx/partition/partition_cfg_8M.toml”文件, xxx 代指芯片类型, 查看分区表中 name 为“FW”的字段。如下图所示, address0 为第一个分区的地址, size0 为第一个分区的长度, 且固件不应该超过第一个分区定义的长度 size0, 否则不能正常烧写。

```
[[pt_entry]]
type = 0
name = "FW"
device = 0
address0 = 0x10000
size0 = 0x280000
address1 = 0x290000
size1 = 0x200000
# compressed image must set len, normal image can left it to 0
len = 0
# If header is 1, it will add the header.
header = 1
# If header is 1 and security is 1, It will be encrypted.
security = 1
```

图 8.3: 8M 分区表 FW 字段

当固件超出分区表 FW 中定义的 size0 时, 烧写会提示出错: fw bin size is overflow with partition table, don't create ota bin。

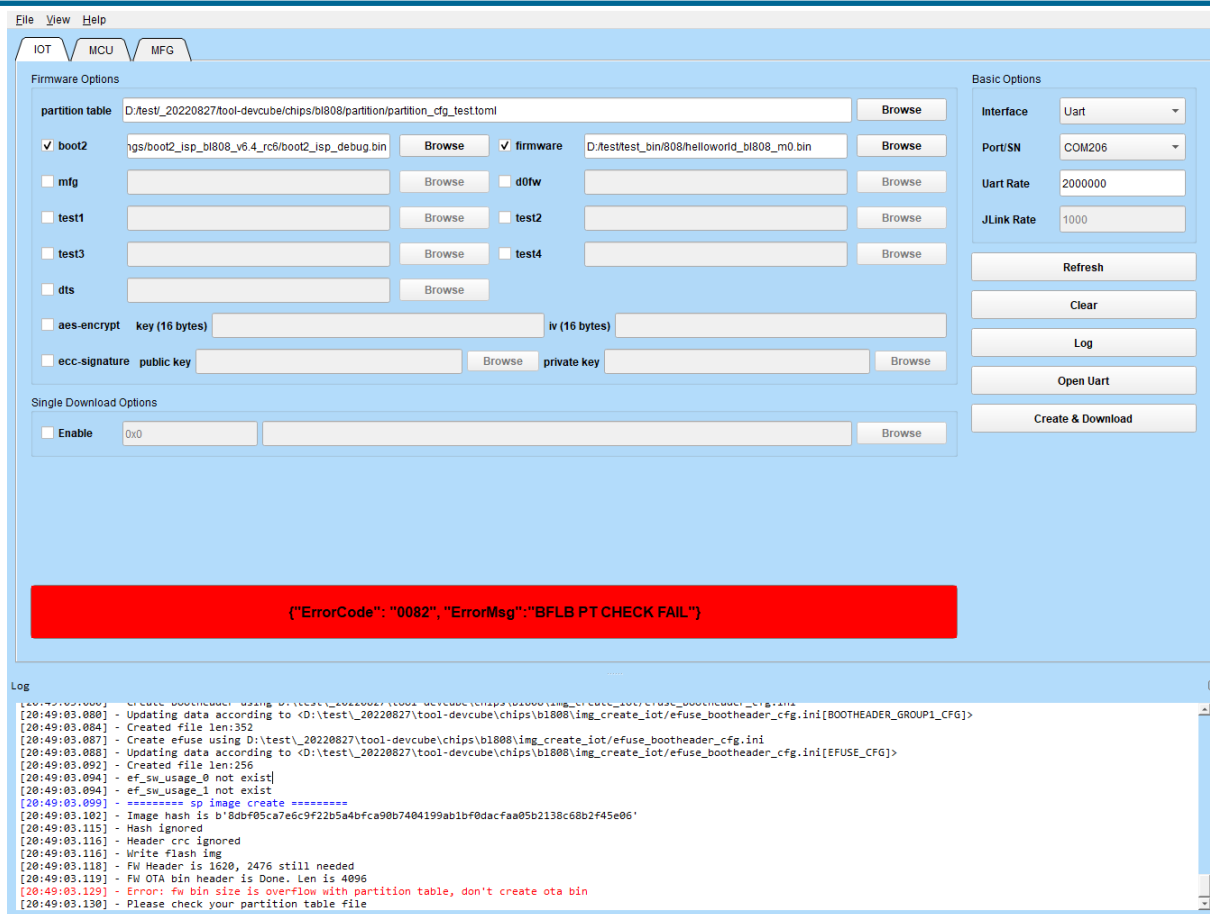


图 8.4: 固件超过分区表定义的大小

8.5 压缩镜像超出分区表中定义的 size1 时不会生成 ota 文件

IOT 界面烧写时默认会生成 ota 文件，当压缩镜像超出分区表中 FW 字段定义的 size1 时则不会生成 ota 文件。首先打开工具目录下的“chips/xxx/partition/partition_cfg_8M.toml”文件，xxx 代指芯片类型，查看分区表中 name 为“FW”的字段。如下图所示，address1 为第二个分区的地址，size1 为第二个分区的长度。且生成的压缩镜像不应该超过第二个分区定义的长度 size1，否则不能正常生成 ota 固件。

```
[[pt_entry]]
type = 0
name = "FW"
device = 0
address0 = 0x10000
size0 = 0x280000
address1 = 0x290000
size1 = 0x200000
# compressed image must set len, normal image can left it to 0
len = 0
# If header is 1, it will add the header.
header = 1
# If header is 1 and security is 1, it will be encrypted.
security = 1
```

图 8.5: 8M 分区表 FW 字段

当固件不超过分区表 FW 中定义的 size0，但压缩镜像的大小超过 size1 时，烧写会提示错误但不影响烧写功能，且不

会生成 ota 文件。

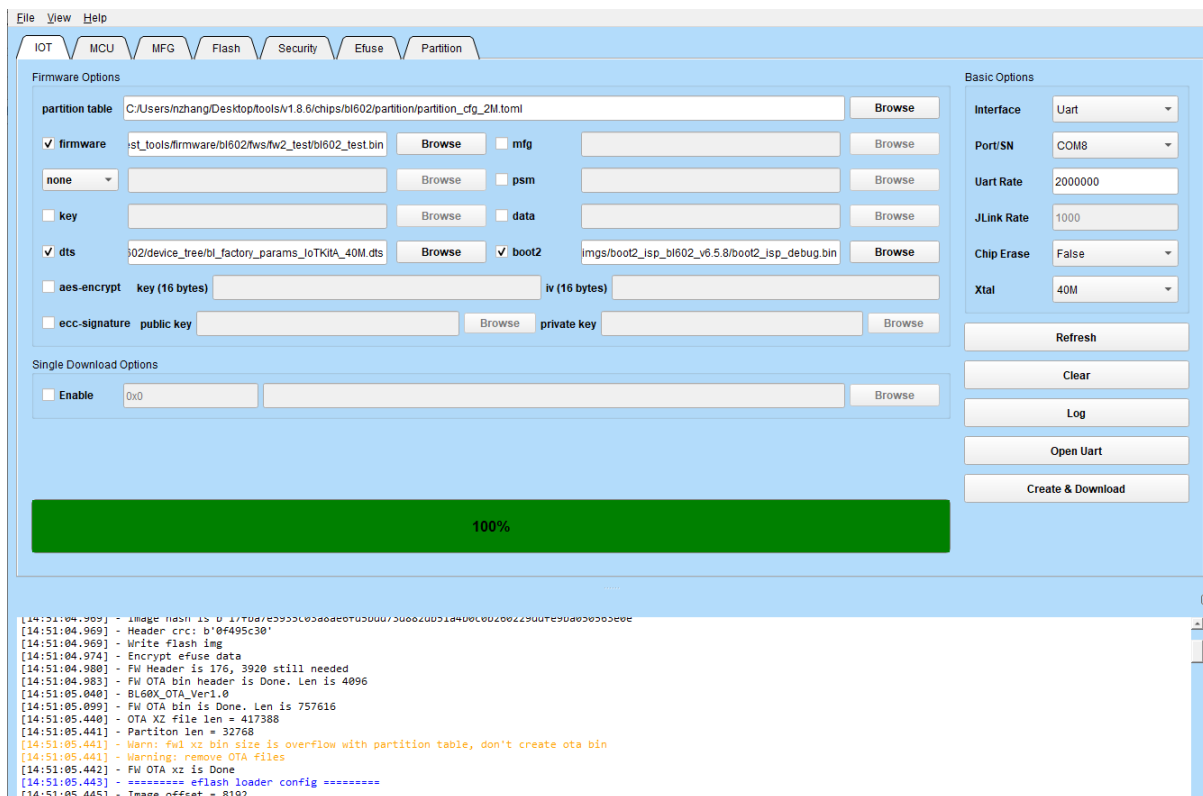


图 8.6: 压缩镜像超过分区表定义的大小

8.6 芯片支持的加密方式

- BL602/BL702/BL702L 仅支持 CTR128 加密方式
- BL616/BL808/BL606P 支持 CTR128/CTR192/CTR256/XTS128/XTS192/XTS256 的加密方式

8.7 Efuse 数据文件支持 CRC 校验

当界面勾选加密/加签或者不勾选的时候,在对应 chip 的 efuse_bootheader 目录下都会生成 efusedata.bin 和 efusedata_mask.bin。其中 efusedata.bin 为加密的固件,对要烧写的 efuse 数据进行了保护。

在 DevCube v1.8.5 及其之后的版本中,对于加密的 efusedata.bin,增加了 CRC 检查的功能,来校验 efuse 数据的正确性。

8.8 增加 Flash size 超出的检查

DevCube v1.8.4 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 4M，实际的 flash 大小仅为 2M，则工具会提示错误：ErrorMsg: BFLB FLASH SIZE OVER FLOW。

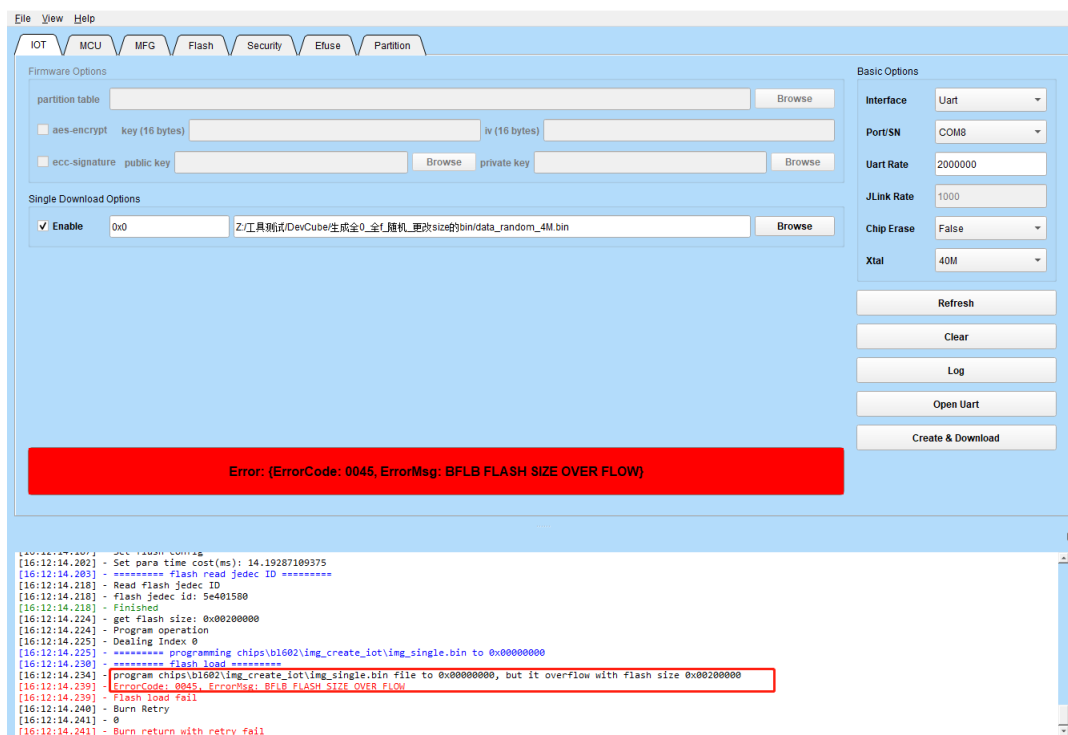


图 8.7: 烧录固件超出 flash 大小

8.9 BL602/BL702 支持不填写密钥签名烧写已经加密加签的板子

DevCube v1.8.6 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，只需要将工具生成的 eflash_loader/eflash_loader_xx_encrypt.bin 拷贝过去即可。

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：

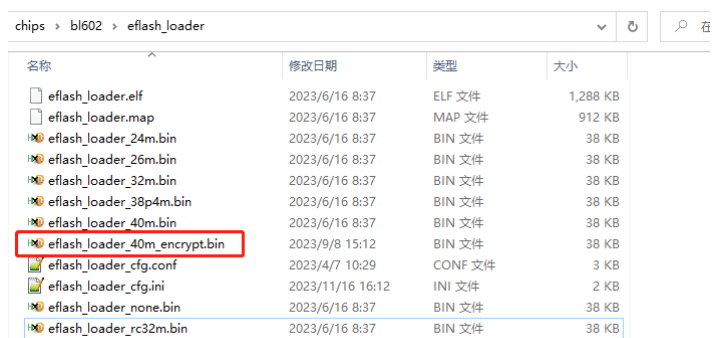


图 8.8: 加密 eflash_loader_encrypt.bin 文件

打开 DevCube 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。

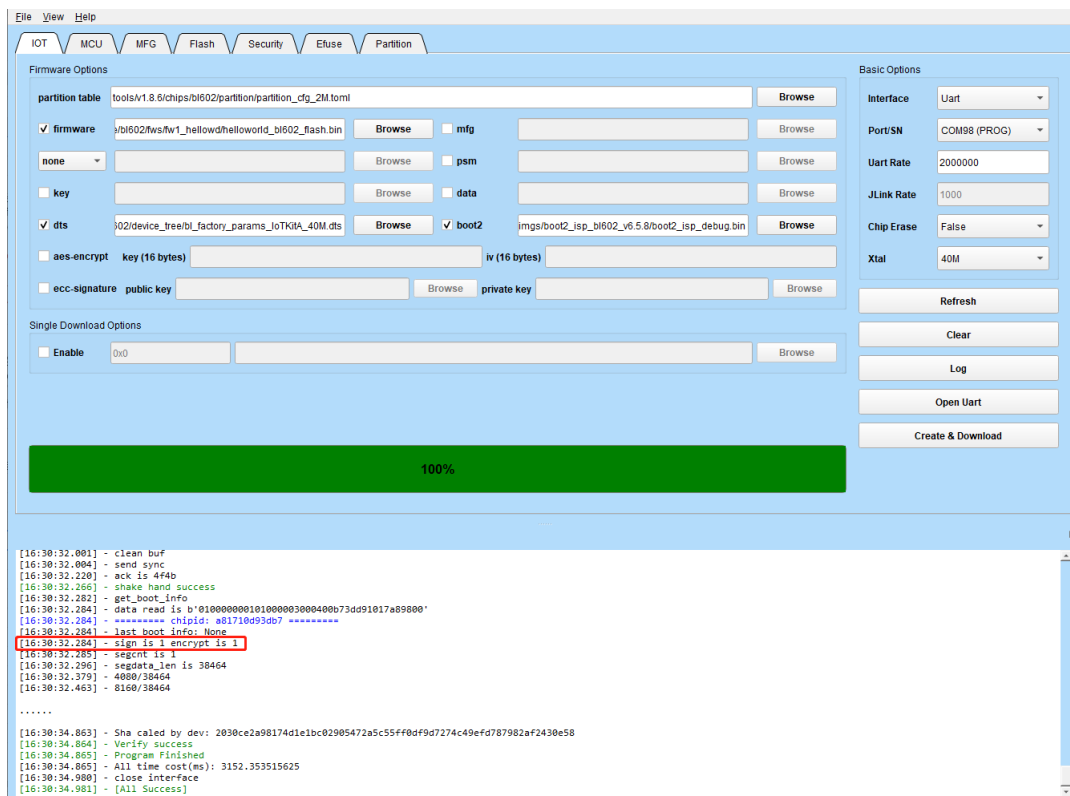


图 8.9: 烧写已经加密加签的板子

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。

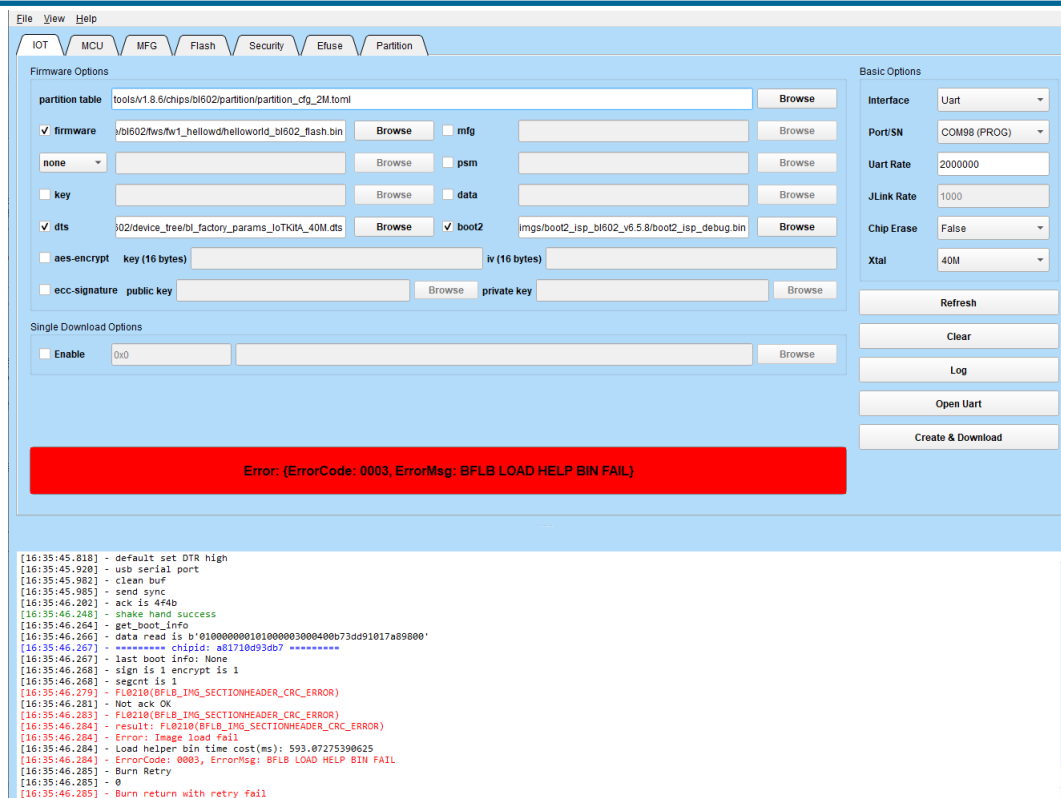


图 8.10: 烧写已经加密加签的板子

表 9.1: 修改记录

版本	描述	日期
1.0	初版	2020-06-14
1.1	增加 Flash 调试助手使用说明	2020-12-22
1.2	增加硬件安全使用说明	2021-03-05
1.3	增加 IAP 程序下载使用说明	2021-06-08
1.4	更新工具界面布局和配置	2022-08-11
1.5	增加高级功能和注意事项	2022-08-29
1.6	增加 Efuse 校验、加密烧写等功能	2023-11-15