



# Flash Cube User Manual

*Version: 1.6*

*Copyright @ 2025*

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

|     |   |    |
|-----|---|----|
| 1   | Overview . . . . .  | 4  |
| 1.1 | Programming Interface Description . . . . .   | 4  |
| 2   | Programming Method . . . . .  | 7  |
| 2.1 | Compile Application Firmware . . . . .  | 7  |
| 2.2 | Import Configuration File . . . . .   | 7  |
| 2.3 | Programming . . . . .   | 8  |
| 2.4 | Running the Program . . . . .   | 9  |
| 3   | Introduction to Programming Configuration File . . . . .                            | 11 |
| 4   | Additional Programming Options . . . . .  | 13 |
| 4.1 | Modify the Partition Table File . . . . .   | 13 |
| 4.2 | Modify the Program Configuration File . . . . .                                     | 14 |
| 4.3 | Programming . . . . .   | 15 |
| 5   | Command Line Program . . . . .  | 17 |
| 5.1 | Individual Firmware Programming . . . . .   | 18 |
| 5.2 | IOT multi-firmware Programming . . . . .  | 20 |
| 5.3 | RAM Programming . . . . .   | 24 |
| 5.4 | Flash/Efuse Read Write Operation . . . . .  | 26 |
| 5.5 | Flash Otp Read Write Operation(only support BL616D/BL616L) . . . . .                | 27 |
| 5.6 | Board with Auto-Download Capability Supporting Automatic Operation Post-Programming | 28 |
| 5.7 | Support RSA Public Key Encryption . . . . .   | 28 |
| 6   | Flash Debugging Assistant . . . . .   | 30 |
| 6.1 | Configure the Communication Method . . . . .  | 31 |
| 6.2 | Read Flash ID . . . . .   | 31 |
| 6.3 | Read Flash . . . . .  | 32 |
| 6.4 | Erase Flash . . . . .   | 32 |
| 6.5 | Read and Write Registers . . . . .  | 34 |

|      |   |    |
|------|---|----|
| 7    | Advanced Features . . . . .                                       | 37 |
| 7.1  | Support Fuzzy Matching of Firmware Paths . . . . .                | 37 |
| 7.2  | Support ISP Programming Mode . . . . .                            | 39 |
| 7.3  | Support Compression and Programming . . . . .                     | 40 |
| 7.4  | Support eFuse Verification Selection . . . . .                    | 40 |
| 7.5  | Support Modifying the Erasure Method During Programming . . . . . | 42 |
| 7.6  | Support Skip Function for Erasing and Writing . . . . .           | 44 |
| 7.7  | Generate Mass Production Programming Files . . . . .              | 46 |
| 7.8  | BL602/BL702 Encryption and Signature Programming . . . . .        | 47 |
| 7.9  | Add Preprocessing Function . . . . .                              | 49 |
| 7.10 | Supports user-defined efusedata.bin Encryption Keys . . . . .     | 50 |
| 8    | Cautions . . . . .  | 51 |
| 8.1  | flash_prog_cfg.ini Has the Highest Priority . . . . .             | 51 |
| 8.2  | Program Option Name Length . . . . .                              | 51 |
| 8.3  | Crystal Oscillator Type Default Setting . . . . .                 | 51 |
| 8.4  | Firmware Exceeds Allocated Address Size . . . . .                 | 51 |
| 8.5  | Firmware Exceeds Flash Size . . . . .                             | 52 |
| 9    | Revision History . . . . .  | 54 |

BLFlashCube is a chip programming tool that supports the programming of user programs, partition tables, boot2, and user resources into the flash memory of the chip. The tool also provides eFuse programming functionality. This document primarily introduces the methods and relevant configurations for programming.

The main features of BLFlashCube are as follows:

1. Supports flash programming and verification of various types of files such as application program code.
2. Supports eFuse programming and verification on the chip.
3. Supports erasing, writing, and reading of flash memory for multiple flash models.
4. Download communication interfaces supported include UART, JLink, CKLink, and OpenOCD, with configurable program speeds.
5. Supports flash programming in chip encryption or signature modes.

Users can obtain the latest version of Flash Cube through [Bouffalo Lab Flash Cube](#).

## 1.1 Programming Interface Description

Double-click BLFlashCube.exe in the decompressed folder to enter the Flash Download program main page.

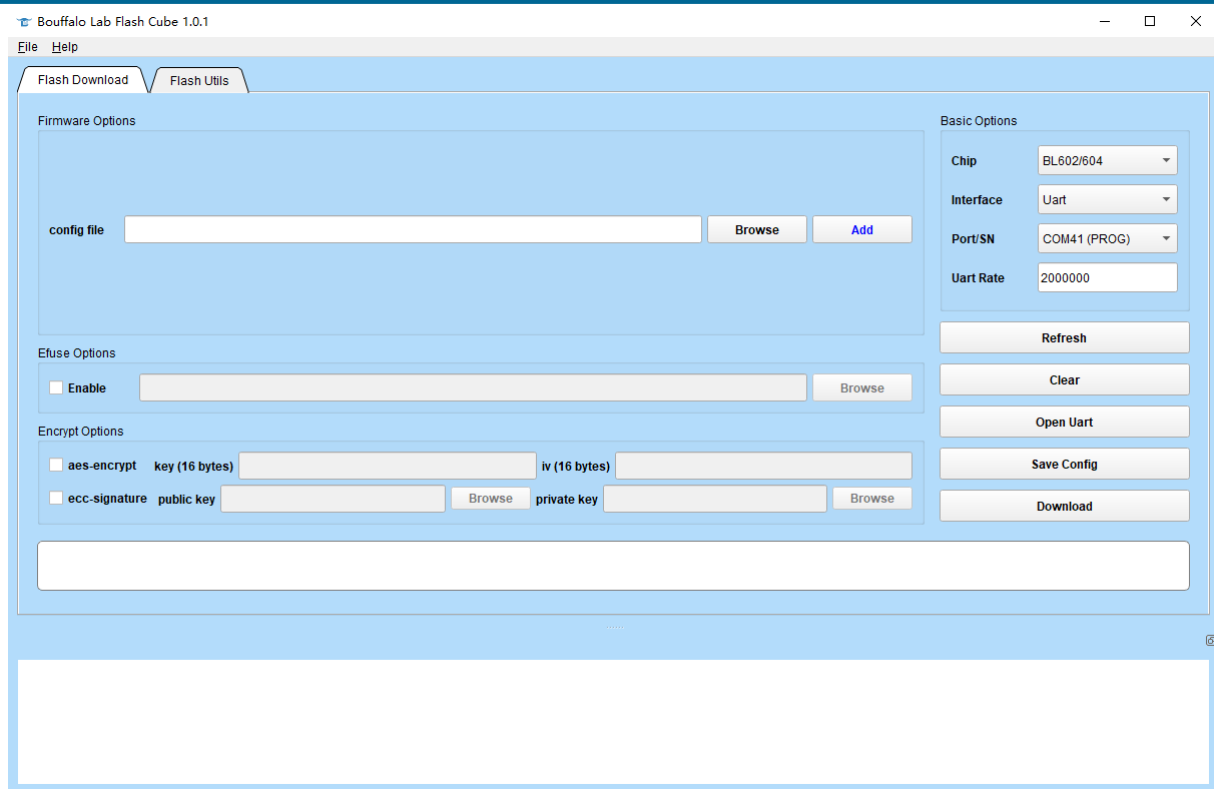


Fig. 1.1: Programming Main Interface

The main programming interface consists of the following parts:

- **Firmware Options** is used to select the programming configuration file. After selecting the configuration file used for programming through the **Browse** button, the specific programming project and programming address can be displayed.
- **Efuse Options** is used for eFuse programming. After checking **Enable**, select the corresponding efusedata.bin file through the **Browse** button. The “efusedata\_mask.bin” file must exist in the same directory for eFuse programming verification, otherwise programming errors will occur.
- **Encrypt Options** only exists in the interface of BL602 / BL702 and is used for programming in encryption or signature mode.
  - **aes-encrypt**: If encryption functionality is enabled, select the **aes-encrypt** option and enter the encryption Key and IV in the adjacent text boxes. The input should be in hexadecimal format, ranging from “0” to “F”. Each Byte consists of two characters, so the Key and IV should each be entered as 32 characters. It’s important to note that the last 8 characters (4 bytes) of the IV must be filled with zeros.
  - **ecc-signature**: If signature functionality is enabled, select the **ecc-signature** option and choose the public key and private key files in the adjacent public key and private key fields. The tool will generate a pk hash and write it to the eFuse.
- **Basic Options** is the relevant configuration for programming, such as chip type, programming method, serial port

number, etc.

- **Chip** is used to select the chip type that currently needs to be programmed. The tool supports the programming functions of multiple types of chips such as BL602/604, BL702/704/706, BL702L, BL808, BL606P and BL616/BL618.
- **Interface** is used to select the communication interface for programming. The optional interfaces are UART, Jlink, CKLink and Openocd. Users can proceed according to the actual physical connection.
- **Port/SN**: When selecting UART for programming, select the COM port number connected to the chip. When selecting Jlink/CKLink/Openocd, what is displayed here is the port number of the device. Users can click the Refresh button to refresh the COM number or port number.
- **Uart Rate** is the baud rate used for programming when selecting UART for programming. The recommended program frequency is 2M
- **JLink Rate** is the configuration of the programming speed when selecting JLink for programming. The default value is 1000
- The **right button** is the related function button
  - **Refresh** is used to refresh the serial port. When connecting a new serial port, you need to click the Refresh button to update the current serial port.
  - **Clear** is used to clear the status of the progress bar and LOG area display
  - **Open Uart**: When Interface is Uart, open the serial port selected by Port/SN
  - **Save Config**: Save the current Firmware Options programming items and addresses to the config file selected configuration file
  - **Download**: Download the selected programming item to Flash. If eFuse is checked, the checked eFuse data will also be written to the chip.
- **Bottom** displays the progress of programming and programming“LOG”
  - **Progress bar**: displays the current programming progress. If there is an error in programming, the error type will also be displayed in the progress bar.
  - **LOG area**: Display the log during the programming process when using the Download button. The startup log will be displayed when using the Open Uart button.

This article uses `examples/wifi/sta/wifi_ota` of `sdk` as an example to introduce the programming process of `bl616`.

## 2.1 Compile Application Firmware

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

After compilation is completed, the corresponding firmware `wifi_ota_bl616.bin` is generated in `build/build_out`. At the same time, this directory also contains `boot2`, `mfg` and other firmware.

## 2.2 Import Configuration File

Open `BLFlashCube` and enter the `Flash Download` page by default, click the `Firmware Options` area `Browse` button to select `flash_prog_cfg.ini` file in the `examples/wifi/sta/wifi_ota` directory. The updated interface is as shown below:

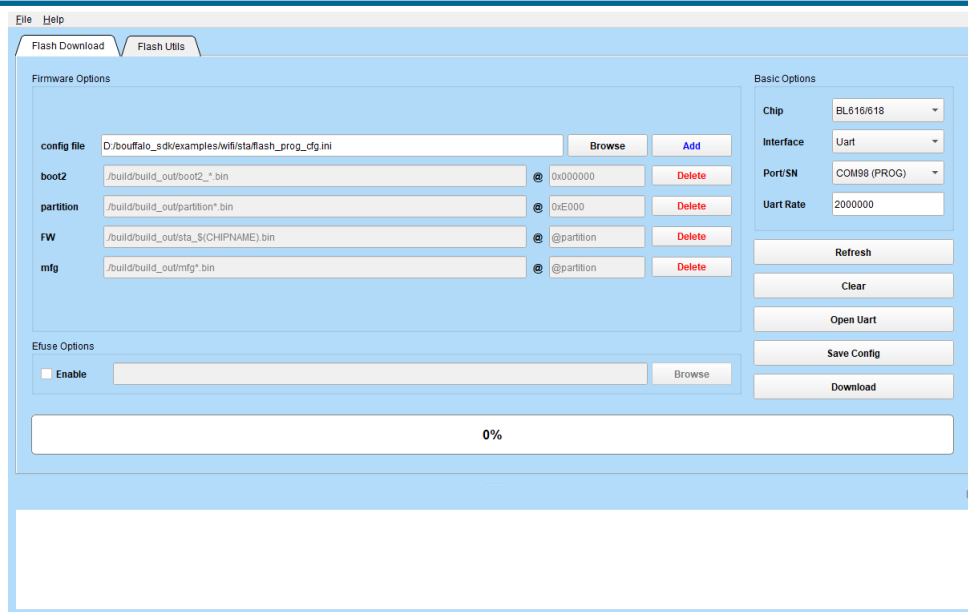


Fig. 2.1: Import Configuration File

## 2.3 Programming

Before programming, please confirm that the programming options are correct. Set the **Chip** option in the **Basic Options** section to BL616/618, select the corresponding programming method in the **Interface** option, and then click the **Refresh** button to update the serial port number/serial number.

- When choosing UART as the programming method, you need to set the BOOT pin of the board to a high level and then reset the chip, so that it enters the UART boot download mode (if the BOOT pin and Reset pin of the user's board are connected to the DTR and RTS of the USB to serial converter, there is no need to set manually. The download program will automatically set the pins to enter the UART boot program mode).
- When choosing Jlink as the programming method, you can keep the Boot pin at a low level to make it start from Flash.

Click **Download**, and the tool will program the file to the specified address according to the page configuration.



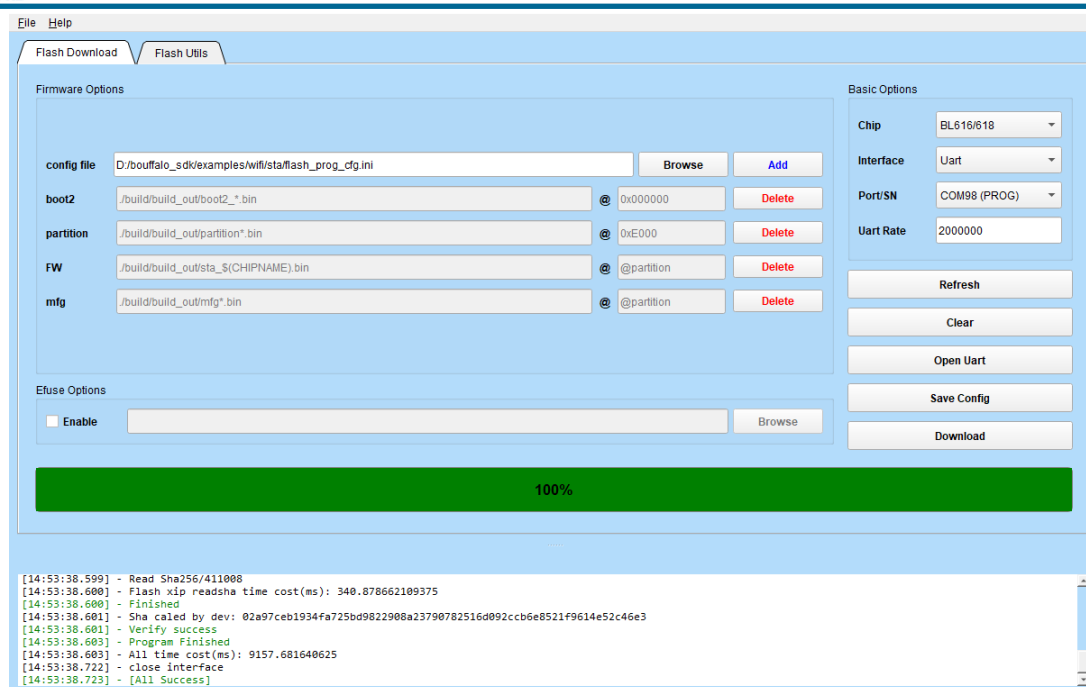


Fig. 2.2: Download Program Successful

When a green progress bar indicating 100% appears, as shown in the image below, it means that the program has been successfully downloaded.

**Note:** If the board is not connected, you can simply generate the complete image file by clicking the **Download** button.

## 2.4 Running the Program

After a successful program, set the BOOT pin of the board to a low level and reset the chip to start it up from Flash. At this point, the application program should start running.

The following image shows the effect of running the wifi/sta/wifi\_ota program.

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

Fig. 2.3: wifi\_ota Program Results

## Introduction to Programming Configuration File

According to the programming requirements of examples/wifi/sta/wifi\_ota, the configuration file flash\_prog\_cfg.ini contains information for programming firmware, including partition table, Boot2, Firmware, mfg, etc.

This section will introduce the components of the programming configuration file.

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

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

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

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

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

**cfg** are some configurations during programming. Just use the default values normally.

- erase: Set the erasure method during programming. The default erase = 1, which means that the Flash will be

erased according to the programming address and content size. `erase = 2` means that all Flash will be erased before programming. `erase = 0` means no erase operation will be performed before programming, and is generally not used.

- `skip_mode`: Set the area not to be operated during erasing and writing. The first parameter is the starting address, and the second parameter is the length. `skip_mode` supports configuring multiple areas at the same time, separated by “;”.
- `boot2_isp_mode`: Control whether to select isp programming mode. `boot2_isp_mode = 1` means using isp programming mode.

**boot2** is the boot2 firmware to be programmed

- `filedir`: relative path where boot2 firmware is located
- `address`: must use address 0

**partition** is the partition firmware to be programmed, and the partition name must be used.

- `filedir`: relative path of partition firmware
- `address`: specified by the partition table file *partition\_XXX.toml* provided by the SDK

**FW** is the application firmware to be programmed. Use “FW” to get it from the partition table.

- `filedir`: relative path to the application firmware. Where `wifi_ota` represents the application firmware name, and `$(CHIPNAME)` represents the chip type.
- `address`: Use “@partition” to automatically obtain the programming address from `partition.bin`. Users can also directly specify the programming address, such as `address = 0x10000`

**mfg** is the mfg firmware to be programmed, which can be obtained from the partition table using “mfg”.

- `filedir`: relative path of mfg firmware
- `address`: Use “@partition” to automatically detect the mfg address from `partition.bin`. Users can also directly specify the programming address, such as `address = 0x210000`.

## Additional Programming Options

For IOT multi-firmware programming, if you want to add a new programming item, users need to modify the configuration information.

- Modify the partition table file to add the partition information for the new program item.
- Add the configuration of the new program in the flash\_prog\_cfg.ini file.

The following is an example of how to add a test program item to bl616.

### 4.1 Modify the Partition Table File

In the “bsp/board/bl616dk/config” directory of the SDK, there is a partition table file named “partition\_xxx.toml”, you only need to modify this partition table file to add a new program item.

For example, program the prepared test.bin to 0x378000 of flash, the firmware size is 0x1000.

Add a new partition with name “test” in the partition table, where address0 is 0x378000, size is 0x1000, and other configurations use default values.

type indicates the partition type, Boot2 will boot the image according to type. When type is equal to 0, it means CPU0 boots the image, and when it is equal to 1, it means CPU1 boots the image. Therefore, customers should avoid 0 and 1 when customizing the partition table, otherwise it will be run by Boot2 as a runnable mirror boot. For details, please refer to “Partition Table Explanation Document”

```
[[pt_entry]]
type = 8
name = "test"
device = 0
address0 = 0x378000
size0 = 0x1000
address1 = 0
size1 = 0
```

(continues on next page)

```
# compressed image must set len, normal image can left it to 0
len = 0
```

After the modification, recompile the application firmware, and you will see “Create partition using partition\_XXX.toml” in the compilation log, which means the partition table file partition.bin has been successfully generated in the build/build\_out directory.

If you see log: “[Warning] No partiton file found in ../../bsp/board/bl616dk/config.go on next steps” , it means that the partition table file is not found, users need to check whether the partition table file exists in the “bsp/board/bl616dk/config” directory.

## 4.2 Modify the Program Configuration File

Open flash\_prog\_cfg.ini in the examples/wifi/sta/wifi\_ota directory and add the test program entry. The content of the configuration file after adding the test program entry is as follows:

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

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

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

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

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

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

In the new test program entry:

- `filedir` specifies the location of the programmed bin file, the example uses a relative path (relative to the path of the configuration file). It is recommended to copy the new programmed firmware to the same directory of FW.
- `address` specifies the program address, in the example, `@partition` is used to get the program address automatically from the partition table, or you can directly specify the program address `0x378000`. The advantage of using `@partition` is that if the partition has to change the address, you only need to modify the partition table file.

## 4.3 Programming

After successful modification, use the BLFlashCube tool to import the new program configuration file and then program it, the program interface after importing is as follows:

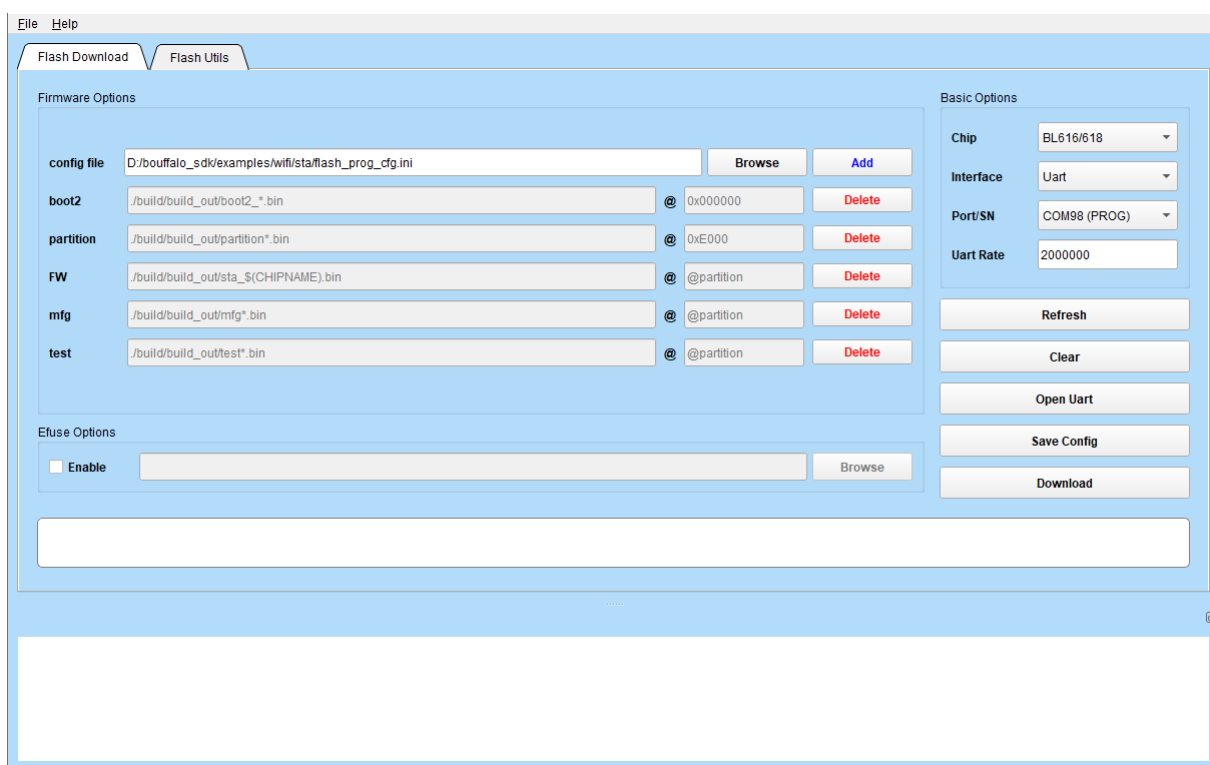


Fig. 4.1: The test programming item was successfully imported into the interface

The programming success interface and programming log are as follows:

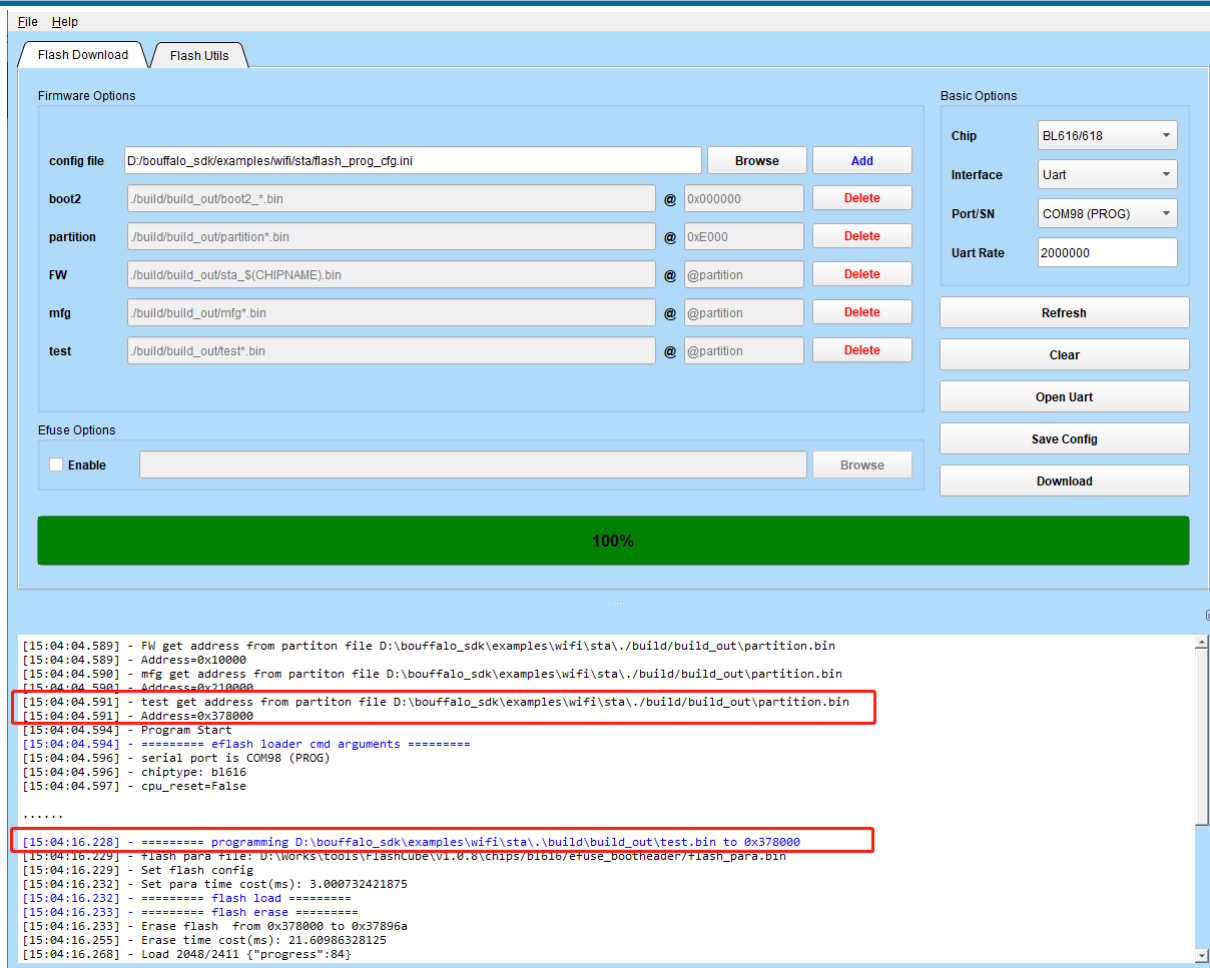


Fig. 4.2: Successfully downloaded the program



## Command Line Program

BLFlashCube provides a command line programming method, the executable file is BLFlashCommand.exe under Windows environment, and the executable file is BLFlashCommand-ubuntu under Linux.

Specific instructions for use are as follows.

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

flash-command

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

(continues on next page)

```

--efusefile           efuse write file
--cpu_id CPU_ID       cpu id
--key KEY             aes key
--iv IV              aes iv
--pk PK              ecc public key
--sk SK              ecc private key
--pk_str PK_STR       ecc public key string
--sk_str SK_STR       ecc private key string
--otpinindex OTPINDEX flash otp index
--lock               enable flash otp lock
--flash              Indicate flash operation
--flash_otp          Indicate flash otp operation
--efuse              Read or write efuse
--erase              Erase flash memory
--build              build pack
--read               Read from flash memory
--write              Write to flash memory
--start START        Start address (hex, e.g., 0x1A2B)
--end END             End address (hex, e.g., 0x1A2C)
--len LEN             Length (hex, e.g., 0x100)
--file FILE          File for reading or writing
--ram                Download image to RAM
--reset              Reset CPU after download
--efuse_encrypted EFUSE_ENCRYPTED
                    encrypted data to write

--whole_chip         Erase whole flash

```

Burning parameters: `--interface` specifies the programming interface, `--port` specifies the programming serial port number, `--chipname` specifies the chip type, `--baudrate` specifies the programming baud rate, and `--config` specifies the programming configuration file to be used.

## 5.1 Individual Firmware Programming

The command line utility can use the `--firmware` parameter to directly specify the firmware to be programmed, and by default, the firmware will be programmed to the start of address 0 of the Flash. To erase the entire flash before programming, you can add the `--whole_chip` option in the command line.

Take BLFlashCommand.exe as an example, the command to program wifi\_ota\_bl616.bin to Flash is as follows:

```

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

```

(continues on next page)

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

(continues on next page)

(continued from previous page)

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

## 5.2 IOT multi-firmware Programming

Refer to the “Additional Burning Options” section to modify the partition table file and programming configuration file.

Taking BLFlashCommand.exe as an example, the command to program the configuration items of flash\_prog\_cfg.ini to Flash is as follows:

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

(continues on next page)

```
[15:47:11.117] - mfg get address from partiton file D:\bouffalo_sdk\examples\wifi\sta\./build/build_out\  
-partition.bin  
[15:47:11.118] - Address=0x210000  
[15:47:11.119] - Program Start  
[15:47:11.119] - ===== eflash loader cmd arguments =====  
[15:47:11.120] - serial port is COM98  
[15:47:11.120] - chiptype: bl616  
[15:47:11.120] - cpu_reset=False  
[15:47:11.257] - com speed: 2000000  
[15:47:11.257] - ===== Interface is uart =====  
[15:47:11.257] - Bootrom load  
[15:47:11.257] - ===== get_boot_info =====  
[15:47:11.257] - ===== image get bootinfo =====  
[15:47:11.262] - default set DTR high  
[15:47:11.366] - usb serial port  
[15:47:11.428] - clean buf  
[15:47:11.431] - send sync  
[15:47:11.645] - ack is b'4f4b'  
[15:47:11.648] - shake hand success  
[15:47:12.160] - data read is b'0100160600000100279280015e64de05b91819000f758010'  
[15:47:12.160] - ===== ChipID: 18b905de645e =====  
[15:47:12.160] - Get bootinfo time cost(ms): 903.37841796875  
[15:47:12.160] - change bdrate: 2000000  
[15:47:12.160] - Clock PLL set  
[15:47:12.161] - Set clock time cost(ms): 0.26513671875  
[15:47:12.282] - Read mac addr  
[15:47:12.283] - MACADDR: 18b905de645e  
[15:47:12.284] - flash set para  
[15:47:12.284] - get flash pin cfg from bootinfo: 0x02  
[15:47:12.285] - set flash cfg: 1014102  
[15:47:12.285] - Set flash config  
[15:47:12.287] - Set para time cost(ms): 1.990966796875  
[15:47:12.287] - ===== flash read jedec ID =====  
[15:47:12.287] - Read flash jedec ID  
[15:47:12.287] - readdata:  
[15:47:12.287] - b'c8401600'  
[15:47:12.287] - Finished  
[15:47:12.287] - flash config Not found,use default  
[15:47:12.287] - jedec_id:c84016  
[15:47:12.287] - capacity_id:22  
[15:47:12.287] - capacity:4.0M  
[15:47:12.287] - get flash size: 0x00400000  
[15:47:12.287] - Program operation
```

(continues on next page)

```
[15:47:12.288] - Dealing Index 0
[15:47:12.288] - ===== programming D:\bouffalo_sdk\examples\wifi\sta\.\build\build_out\boot2_bl616_
->release_v8.0.7.bin to 0x000000
[15:47:12.288] - flash para file: D:\Works\tools\FlashCube\v1.0.8\chips/bl616/efuse_bootheader/flash_para.
->.bin
[15:47:12.288] - Set flash config
[15:47:12.292] - Set para time cost(ms): 3.126953125
[15:47:12.292] - ===== flash load =====
[15:47:12.292] - ===== flash erase =====
[15:47:12.292] - Erase flash from 0x0 to 0xa93f
[15:47:12.599] - Erase time cost(ms): 307.765380859375
[15:47:12.615] - decompress flash load 21796
[15:47:12.788] - Load 21796/21796 {"progress":100}
[15:47:12.789] - Write check
[15:47:12.806] - Flash load time cost(ms): 204.3623046875
[15:47:12.806] - Finished
[15:47:12.807] - Sha caled by host: 81bdd6bd9e028b2d1fa5da8d12aa4438353842d3f2a0b85e61a4efb00dd50fd0
[15:47:12.807] - xip mode Verify
[15:47:12.844] - Read Sha256/43328
[15:47:12.844] - Flash xip readsha time cost(ms): 36.54638671875
[15:47:12.844] - Finished
[15:47:12.845] - Sha caled by dev: 81bdd6bd9e028b2d1fa5da8d12aa4438353842d3f2a0b85e61a4efb00dd50fd0
[15:47:12.845] - Verify success
[15:47:12.846] - Dealing Index 1
[15:47:12.846] - ===== programming D:\bouffalo_sdk\examples\wifi\sta\.\build\build_out\partition.bin to 0x
->0xE000
[15:47:12.846] - flash para file: D:\Works\tools\FlashCube\v1.0.8\chips/bl616/efuse_bootheader/flash_para.
->.bin
[15:47:12.846] - Set flash config
[15:47:12.849] - Set para time cost(ms): 2.998779296875
[15:47:12.849] - ===== flash load =====
[15:47:12.849] - ===== flash erase =====
[15:47:12.849] - Erase flash from 0xe000 to 0xe133
[15:47:12.888] - Erase time cost(ms): 39.04541015625
[15:47:12.893] - Load 308/308 {"progress":100}
[15:47:12.893] - Write check
[15:47:12.894] - Flash load time cost(ms): 3.004150390625
[15:47:12.894] - Finished
[15:47:12.895] - Sha caled by host: 6c50e14f2776fb705aaffb46d9022f2c062a296303d27c9545715585ddf93625
[15:47:12.895] - xip mode Verify
[15:47:12.896] - Read Sha256/308
[15:47:12.896] - Flash xip readsha time cost(ms): 1.000732421875
[15:47:12.896] - Finished
```

(continues on next page)

```
[15:47:12.896] - Sha caled by dev: 6c50e14f2776fb705aaffb46d9022f2c062a296303d27c9545715585ddf93625
[15:47:12.896] - Verify success
[15:47:12.898] - Dealing Index 2
[15:47:12.898] - ===== programming D:\bouffalo_sdk\examples\wifi\sta\.\build\build_out\sta_bl616.bin to
-0x10000
[15:47:12.899] - flash para file: D:\Works\tools\FlashCube\v1.0.8\chips/bl616/efuse_bootheader/flash_para.
-bin
[15:47:12.899] - Set flash config
[15:47:12.901] - Set para time cost(ms): 1.998046875
[15:47:12.901] - ===== flash load =====
[15:47:12.902] - ===== flash erase =====
[15:47:12.902] - Erase flash from 0x10000 to 0xda61f
[15:47:14.840] - Erase time cost(ms): 1938.134033203125
[15:47:15.112] - decompress flash load 494024
[15:47:18.693] - Load 494024/494024 {"progress":100}
[15:47:18.694] - Write check
[15:47:18.706] - Flash load time cost(ms): 3865.414794921875
[15:47:18.707] - Finished
[15:47:18.710] - Sha caled by host: 6167624bb39d78d164eada22e9802520fb2bea0b526a5563fc4ea6568d557747
[15:47:18.710] - xip mode Verify
[15:47:19.396] - Read Sha256/828960
[15:47:19.396] - Flash xip readsha time cost(ms): 685.072998046875
[15:47:19.397] - Finished
[15:47:19.397] - Sha caled by dev: 6167624bb39d78d164eada22e9802520fb2bea0b526a5563fc4ea6568d557747
[15:47:19.397] - Verify success
[15:47:19.399] - Dealing Index 3
[15:47:19.399] - ===== programming D:\bouffalo_sdk\examples\wifi\sta\.\build\build_out\mfg_bl616_gu_
-af8b0946f_v2.26.bin to 0x210000
[15:47:19.400] - flash para file: D:\Works\tools\FlashCube\v1.0.8\chips/bl616/efuse_bootheader/flash_para.
-bin
[15:47:19.400] - Set flash config
[15:47:19.403] - Set para time cost(ms): 2.56005859375
[15:47:19.403] - ===== flash load =====
[15:47:19.403] - ===== flash erase =====
[15:47:19.403] - Erase flash from 0x210000 to 0x27457f
[15:47:20.325] - Erase time cost(ms): 922.190673828125
[15:47:20.461] - decompress flash load 222180
[15:47:22.110] - Load 222180/222180 {"progress":100}
[15:47:22.110] - Write check
[15:47:22.124] - Flash load time cost(ms): 1797.292236328125
[15:47:22.124] - Finished
[15:47:22.126] - Sha caled by host: 02a97ceb1934fa725bd9822908a23790782516d092ccb6e8521f9614e52c46e3
[15:47:22.126] - xip mode Verify
```

(continues on next page)

```
[14:39:48.564] -
[14:39:48.566] -
[14:39:48.566] - Bouffalolab
[14:39:48.570] -
[14:39:48.572] -
[14:39:48.573] -
[14:39:48.574] - Build:14:26:08,Mar 17 2023
[14:39:48.576] - Copyright (c) 2022 Bouffalolab team
[14:39:48.577] - ===== flash cfg =====
[14:39:48.578] - jedec id 0xEF4017
[14:39:48.580] - mid 0xEF
[14:39:48.581] - iomode 0x04
[14:39:48.581] - clk delay 0x01
[14:39:48.581] - clk invert 0x01
[14:39:48.582] - read reg cmd0 0x05
[14:39:48.582] - read reg cmd1 0x35
[14:39:48.583] - write reg cmd0 0x01
[14:39:48.584] - write reg cmd1 0x31
[14:39:48.585] - qe write len 0x01
[14:39:48.586] - cread support 0x00
[14:39:48.586] - cread code 0xFF
[14:39:48.587] - burst wrap cmd 0x77
[14:39:48.588] - =====
[14:39:48.590] - dynamic memory init success, ocram heap size = 150 Kbyte, psram heap size = 4096 Kbyte
[14:39:48.591] - sig1:ffffff
[14:39:48.592] - sig2:0000f32f
[14:39:48.592] - cgen1:9f7ffffd
[14:39:48.593] - @[0mbouffalolab />@[0mlwip init done
[14:39:48.925] - sys_mbox_new done!
[14:39:48.926] - sys_mutex_new done!
[14:39:48.927] - tcpip thread init done!
[14:39:48.927] - [I][rtparam] Starting wifi ...
[14:39:48.929] - [I][rtparam] rtparam>>xtal value 40000000
[14:39:48.929] - [I][rtparam] rtparam>>pwr_mode is bf
[14:39:48.930] - Empty slot:1
[14:39:48.931] - [I][rtparam] rtparam>>efuse wlan pwr_offset[14]: 1,1,1,1,1,1,0,0,0,0,0,0,0,0,
[14:39:48.932] - [I][rtparam] rtparam>>tlv wlan pwr_offset[14]: 0,0,0,0,0,0,0,0,0,0,0,0,0,0,
[14:39:48.934] - [I][rtparam] rtparam>>wlan pwr_offset[14]: 1,1,1,1,1,1,0,0,0,0,0,0,0,0,
[14:39:48.934] - Empty slot:0
[14:39:48.935] - No written slot found
[14:39:48.938] - [I][rtparam] rtparam>>no lp pwr_offset in efuse
[14:39:48.939] - [I][rtparam] rtparam>>tlv wlan lp pwr_offset[14]: 0,0,0,0,0,0,0,0,0,0,0,0,0,0,
[14:39:48.941] - [I][rtparam] rtparam>>wlan lp pwr_offset[14]: 0,0,0,0,0,0,0,0,0,0,0,0,0,0,
[14:39:48.943] - Empty slot:0
[14:39:48.944] - No written slot found
[14:39:48.946] - [I][rtparam] rtparam>>no bz pwr_offset in efuse
[14:39:48.946] - [I][rtparam] rtparam>>tlv bz pwr_offset[5]: 0,0,0,0,0,
[14:39:48.947] - [I][rtparam] rtparam>>bz pwr_offset[5]: 0,0,0,0,0,
[14:39:48.949] - [I][rtparam] rtparam>>pwr_11b[4]: 20,20,20,20,
[14:39:48.950] - [I][rtparam] rtparam>>pwr_11g[8]: 18,18,18,18,18,18,16,16,
[14:39:48.951] - [I][rtparam] rtparam>>pwr_11n_ht20[8]: 18,18,18,18,18,16,15,15,
[14:39:48.953] - [I][rtparam] rtparam>>pwr_11n_ht40[8]: 18,18,18,18,18,16,15,14,
[14:39:48.955] - [I][rtparam] rtparam>>pwr_11ac[16]: 15,15,14,14,15,15,14,14,
```

Fig. 5.1: Startup Log

### 5.3 RAM Programming

Command line tools can directly specify the firmware to be burned using the `--firmware` `--ram` parameter, which defaults to the RAM address corresponding to the different chip types.

Using BLFlashCommand.exe as an example, run the following command to write flash\_forbidden\_cmd\_test\_bl616.bin to the Flash:



```
D:\Works\tools\BLFlashCube\v1.1.2> .\BLFlashCommand.exe --interface=uart --chipname=bl616 --port=COM98 --
-baudrate=2000000 --firmware=D:\bouffalo_sdk\chiptest\bl616\flash\flash_forbidden_cmd_test\build\build_out\
-flash_forbidden_cmd_test_bl616.bin --ram

[15:47:57.661] - Serial port is COM98
[15:47:57.661] - =====
[15:47:57.681] - ===== image load =====
[15:47:57.688] - default set DTR high
[15:47:57.797] - usb serial port
[15:47:57.860] - clean buf
[15:47:57.905] - send sync
[15:47:58.128] - ack is b'4f4b'
[15:47:58.130] - shake hand success
[15:47:58.144] - get_boot_info
[15:47:58.146] - data read is b'0200160600000100679680013bf22c5042f41a000f758010c42a6dad'
[15:47:58.146] - ===== ChipID: f442502cf23b =====
[15:47:58.146] - last boot info: None
[15:47:58.146] - sign is 0 encrypt is 0
[15:47:58.167] - Download D:\bouffalo_sdk\chiptest\bl616\flash\flash_forbidden_cmd_test\build\build_out\
-flash_forbidden_cmd_test_bl616.bin
[15:47:58.167] - segcnt is 1
[15:47:58.175] - segdata_len is 34496
[15:47:58.258] - 4080/34496
[15:47:58.341] - 8160/34496
[15:47:58.425] - 12240/34496
[15:47:58.507] - 16320/34496
[15:47:58.590] - 20400/34496
[15:47:58.673] - 24480/34496
[15:47:58.756] - 28560/34496
[15:47:58.839] - 32640/34496
[15:47:58.877] - 34496/34496
[15:47:58.878] - Run img
[15:47:58.879] - Img run success
[15:47:58.879] - All time cost(ms): 1198.422607421875
[15:47:58.879] - True
```

## 5.4 Flash/Efuse Read Write Operation

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

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

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

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

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

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

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

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

```
# read data from efuse and save to read_file.bin
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --efuse --read --start=0x00 --end=0x1FF --
-rfile=read_file.bin
```

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

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

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

## 5.5 Flash Otp Read Write Operation(only support BL616D/BL616L)

This tool provides the flash otp function to erase and read/write data. It is suitable for applications requiring high security and data protection. Once data is locked after writing, it cannot be modified or erased. It is usually used to store fixed, important data, such as a device's serial number, encryption key, or configuration parameters. Due to its unchangeable nature, OTP memory has advantages in terms of security and tamper-resistance.

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

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

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

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

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

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

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

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

```
# erase flash_otp data by index
--interface=uart --chipname=bl616d --port=COM24 --baudrate=115200 --flash_otp --otpindex=0 --erase
```

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

```
# lock flash_otp data by index
--interface=uart --chipname=bl616d --port=COM24 --baudrate=115200 --flash_otp --otpindex=0 --lock
```

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

## 5.6 Board with Auto-Download Capability Supporting Automatic Operation Post-Programming

If the Boot and Reset pins of the board are connected to the USB-to-serial DTR and RTS, the downloader will automatically set the pins to be in the UART boot download state or the startup state. When the user uses the `--reset` parameter, the download program will automatically reset the board into the startup state after the firmware is burned.

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

## 5.7 Support RSA Public Key Encryption

The tool supports writing data to a specified location in efuse. Use `--data` to pass in unencrypted data, and use `--efuse_encrypted` to pass in encrypted efuse data (encrypted using RSA public key), which has higher security performance. The command line tool uses `--efuse_encrypted` to pass the encrypted efuse data `--addr` to the specified location, as shown in the following example.

```
# flash reset after being downloaded
--interface=uart --chipname=bl616 --port=COM24 --baudrate=2000000 --efuse_
--encrypted=3cfbc0fc209b1192bc52208f03ae5b2d880e3fb0ee26577d993d57fcb5bd8531979d022a68b6f41c7081e776ded777
bf23fb0a832e27c9bbab8b7d938171c6fb48a18bd0f21d8ba1a91c8e55509c1552d5343cd8d49e74c4fd12ba4c0734a0422bfcd48a
719f02f3c24d54f530f63a74341bdc94f5a5f96287bb4c02b3b90d288501bca4cfeeab413e391ffdec9c5216e0591ee72cd7bacc94
796cb0ecbd94e985bea160b8778849372e29f9471bbf3ea824f3a097d89c2d77e2834b61643c5c7f7aaf88970217cab069f51b6658
2c6a7d0494c0579f299535d6e2f3d65905fe23964b4eb865e285f6cd715973645511cee6825c7f52d8a77f6a482ecff9e62d9f32b4
d3c8c66e9db9cecf8ee5909c7bbe165f0fa06a8366434fdbcff031c1a6f0d742b13556ef8a5ede64f0be904ceb35180f7b8cd97262
c217ecfec2055d4d20636e6f67cb6b462f75169e37ab96c323d92ea8a7ca1c116c080be3ad8003479d666fbf06efaa111b073bf815
```

(continues on next page)

(continued from previous page)

```
9ed2a3066ef1a3371a9eadfc4e8de1734f34f003a5989ca3a1b8de22298a79f8ac3791da7787a7f921e0c0f7dfee4dfc4f75dcea13
9db544ba77b43682bdeef1ae8b33cdd5a9783cf4502c75e163a73a23f69596de41aa2af35722fe857a82a48ff376b37f61928a4f4c
c20b7e108e248eddb90dc608c657045bd46ca5151087892bfc31dec2cf04a8087b66e97e082ae02b2834751b179155bc11401a31a0
ef0eb9cfc7c1ed5af2de7e534d6144212177b9f4746e9a126c8c10b60515c0b4158738ebb961a10fc628b551fed7cd17553f351dba
7bbbe19007af7895bd800f1d2a0d6eb244f2695881077fdd87f969fafc67f390a072907f6ad654ee6557c0a590f0536135558dc37a
ad4d62c5f0c6ee460de94821b18146daa96938f716bd327211abe7febc1ceb195d96d468b9802680cae90f6dd1b0b6814ea7d90629
be0f6c61567156b04fe8c79362844ddfa2a15641550fe202c78ae0053725f826488e0b05cbcca16a39f4b45f69d9235ec3f37e0945
14727ae4a80443bfc669f953f1802e550d8af479dd8d4b68afe2fea5e0b62b03 --addr=0x0
```

## Flash Debugging Assistant

Select the **Flash Utils** option in the first line menu to enter the Flash Debugging Assistant interface. The Flash Debugging Assistant is used to get the Flash ID, read and erase the content of the specified address of the Flash, and read and write the values of the registers.

The screenshot displays the Flash Debugging Assistant software interface. At the top, there is a menu bar with 'File' and 'Help'. Below the menu bar are two tabs: 'Flash Download' and 'Flash Utils', with 'Flash Utils' currently selected. The main interface is divided into several sections:

- Flash Read/Erase:** Contains input fields for 'ID', 'Start Addr', and 'End Addr'. Each field has a corresponding button: 'Read ID', 'Read Flash', and 'Erase Flash'. There is also a checkbox labeled 'Whole Chip'.
- Register Read/Write:** Contains input fields for 'Command', 'Length', and 'Value'. Each field has a corresponding button: 'Read Register' and 'Write Register'.
- Basic Options:** A panel on the right side containing dropdown menus for 'Chip' (set to BL602/604), 'Interface' (set to Uart), and 'Port/SN' (set to COM96). It also has a text input for 'Uart Rate' (set to 2000000) and three buttons: 'Refresh', 'Clear', and 'Download'.
- Flash Program:** A section at the bottom with a 'flash addr' input field, an 'image file' input field, and a 'Browse' button.

At the bottom of the interface, there are two large, empty rectangular areas, likely for displaying data or logs.

Fig. 6.1: Flash Debug Assistant Interface

## 6.1 Configure the Communication Method

- Basic Options area configuration parameters include:
  - Chip: Used to select the type of chip that needs to be programmed currently, supports BL602/604, BL702/704/706, BL808, BL606P and BL616/BL618, etc.
  - Interface: Used to select the communication interface for programming, the optional interfaces are Jlink, UART, CKLink and Openocd, users choose according to the actual physical connection
  - Port/SN: When UART is selected for programming, the COM port number connected to the chip is selected here; when Jlink /CKLink /Openocd is selected, the port number of the device is displayed here. When Jlink /CKLink /Openocd is selected, the port number of the device is shown here. You can click the Refresh button to refresh the COM number or port number.
  - Uart Rate: When selecting UART for programming, you need to fill in the baud rate, the recommended program frequency is 2M.
  - JLink Rate: When selecting JLink for programming, the default value is 1000.

## 6.2 Read Flash ID

- Read Flash ID: Click Read ID.

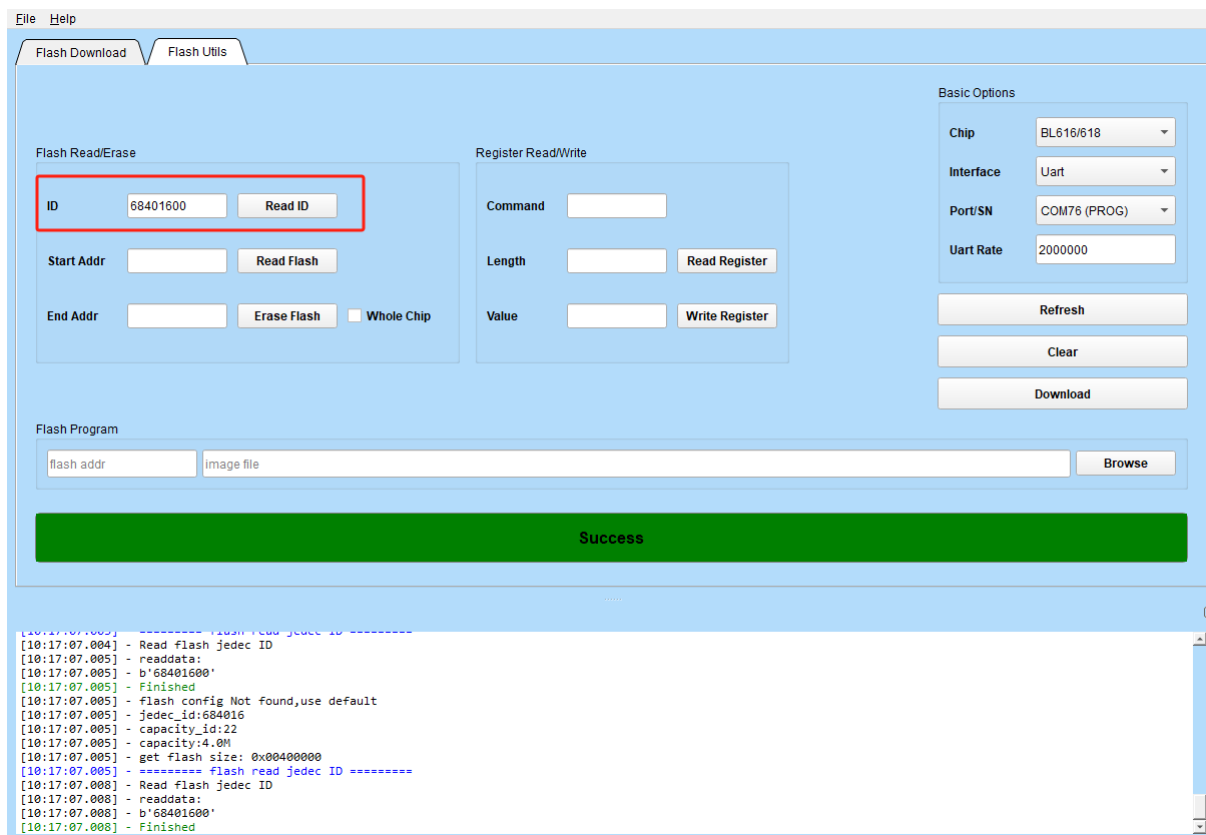


Fig. 6.2: Read flash ID

## 6.3 Read Flash

- Read the data in the specified address segment of Flash: fill in the Start Addr with the start address of the data to be read, and fill in the End Addr with the end address of the data to be read, click Read Flash, and the read content will be updated in the flash.bin file in the root directory of the tool.

The following example is to read data from Flash 0x0 ~ 0x2000 address.

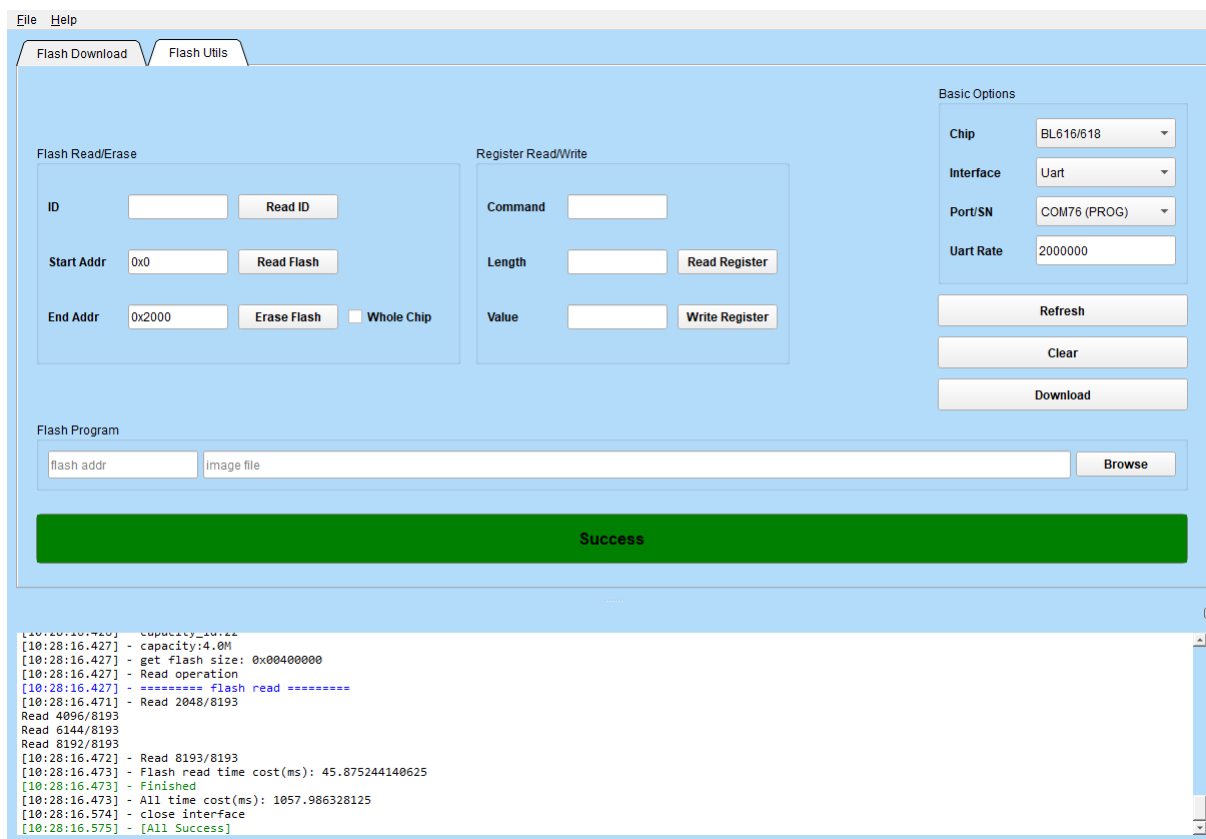


Fig. 6.3: Read data from flash 0x0 ~ 0x2000 address

## 6.4 Erase Flash

- To erase the data of the specified address segment of Flash: Fill in the Start Addr with the start address of the data to be erased, and fill in the End Addr with the end address of the data to be erased. After clicking Erase Flash, the tool will erase.



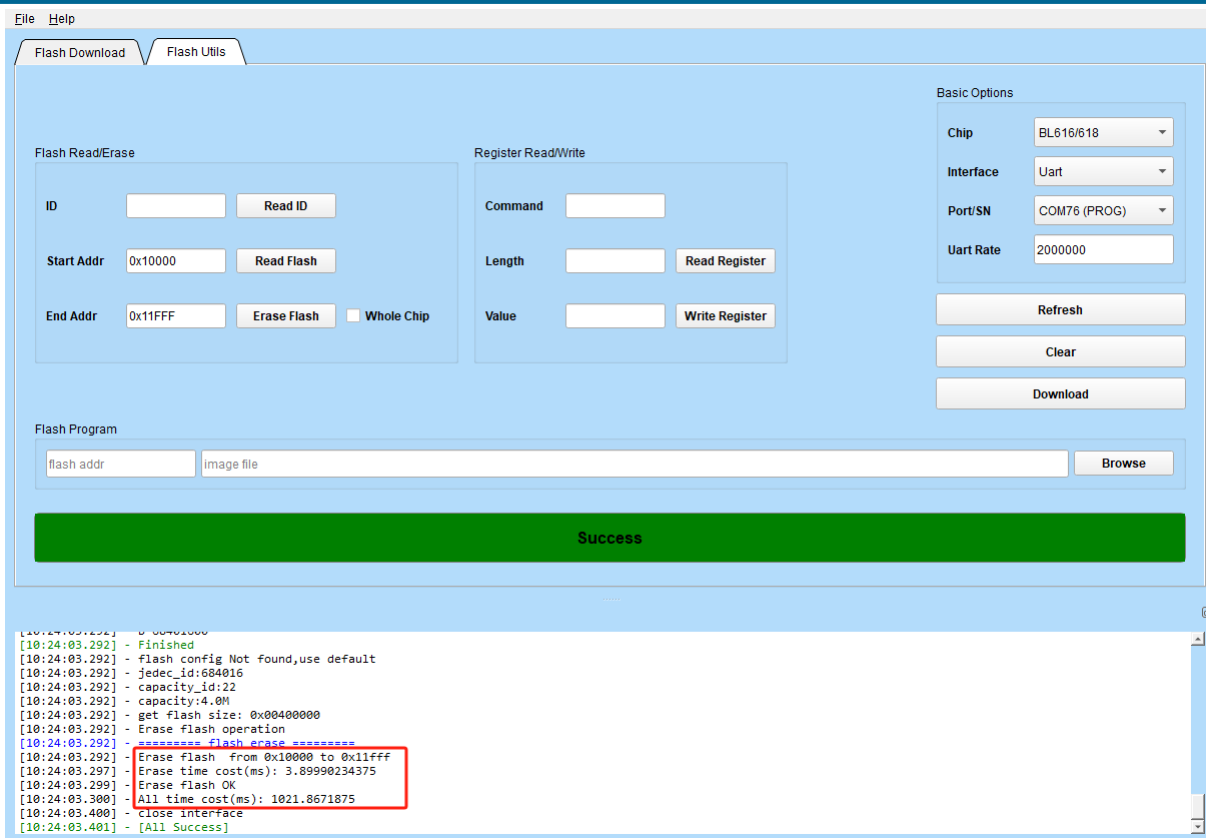


Fig. 6.4: Erase Flash Interface

**Note:** The unit of Flash erase is 4KB (0x1000), you need to be careful not to exceed the End Addr.

- To erase the Flash area of the whole chip, you need to check Whole Chip and then click Erase Flash.

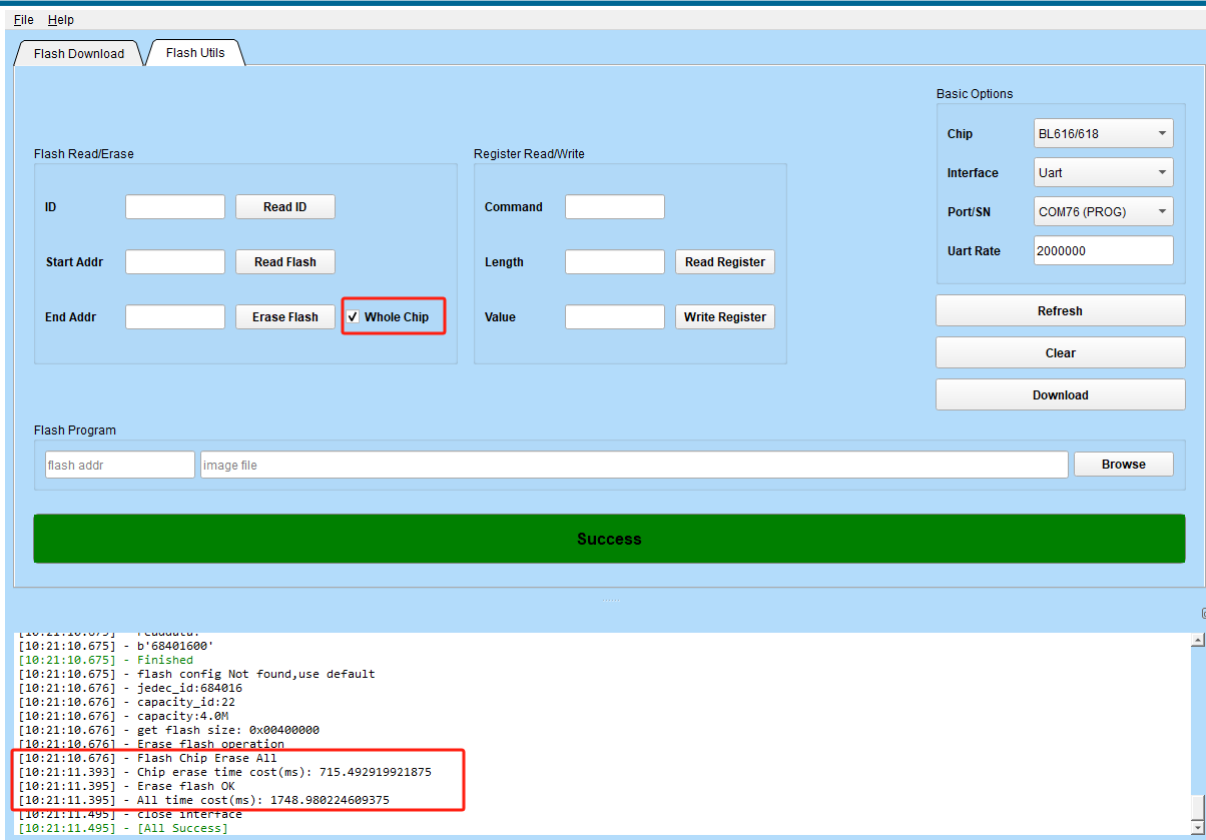
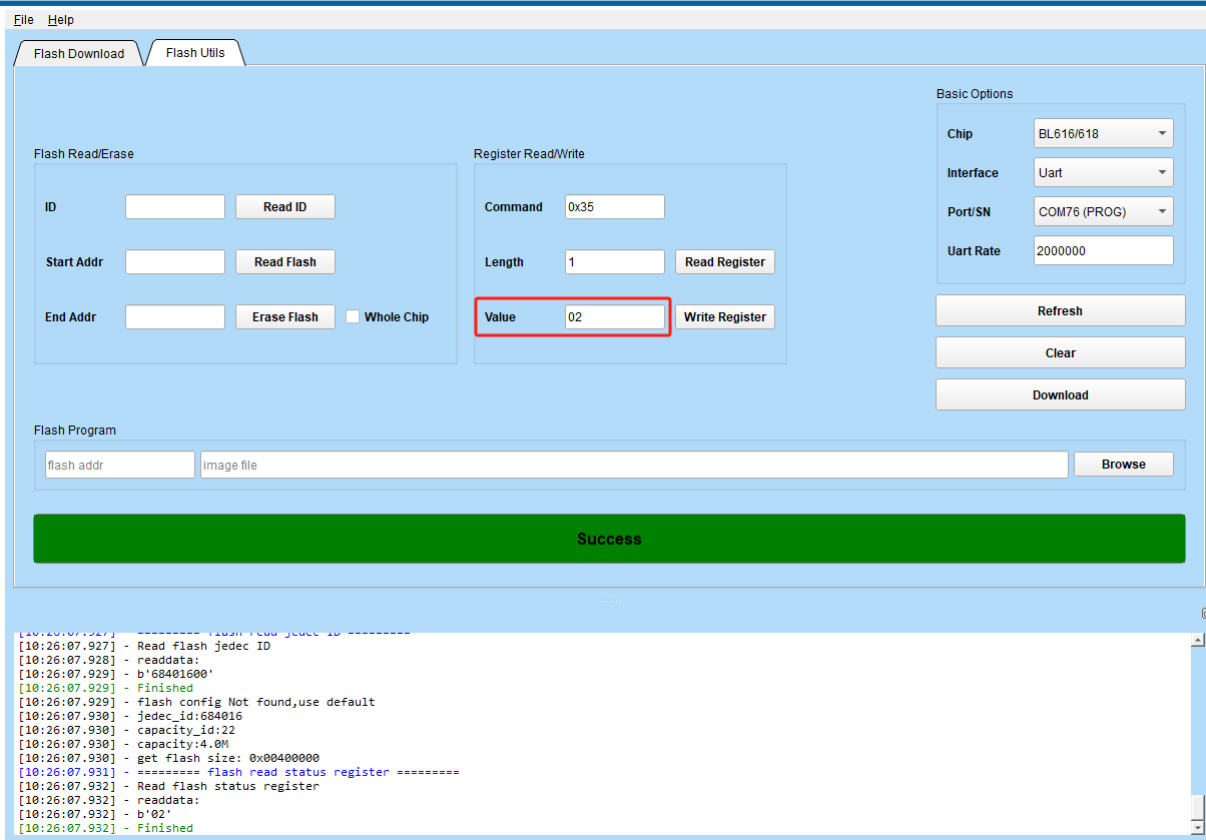


Fig. 6.5: Erase Flash Interface

## 6.5 Read and Write Registers

- Read the contents of the register: enter 0x05/0x35 in Command, fill in the number of bits to be read in Length, click Read Register, the read data will be displayed in Value



The screenshot displays the 'Flash Cube User Manual' software interface. At the top, there are tabs for 'Flash Download' and 'Flash Utils'. The 'Flash Utils' tab is active, showing a 'Flash Read/Erase' section with fields for 'ID', 'Start Addr', and 'End Addr', each with a corresponding 'Read' or 'Erase' button. A 'Whole Chip' checkbox is also present. To the right, the 'Register Read/Write' section includes a 'Command' field (set to 0x35), a 'Length' field (set to 1), and a 'Value' field (set to 02, highlighted with a red box). Buttons for 'Read Register' and 'Write Register' are provided. On the far right, 'Basic Options' include dropdowns for 'Chip' (BL616/618), 'Interface' (Uart), and 'Port/SN' (COM76 (PROG)), along with a 'Uart Rate' field (2000000) and 'Refresh', 'Clear', and 'Download' buttons. Below these sections is a 'Flash Program' area with 'flash addr' and 'image file' fields and a 'Browse' button. A large green bar with the text 'Success' is visible. At the bottom, a terminal window shows a log of operations, including 'Read flash jedec ID', 'flash config Not found, use default', and 'Read flash status register'.

Fig. 6.6: Read Register Interface

- Write Register: Enter 0x01/0x31 in Command, fill in the number of bits to be written in Length, fill in the written data in Value, and click Write Register.

File

Help

Flash Download

Flash Utils

Flash Read/Erase

ID

Read ID

Start Addr

Read Flash

End Addr

Erase Flash

☐ Whole Chip

Register Read/Write

Command

0x01

Length

1

Read Register

Value

00

Write Register

Basic Options

Chip

BL616/618

Interface

Uart

Port/SN

COM76 (PROG)

Uart Rate

2000000

Refresh

Clear

Download

Flash Program

flash addr

Image file

Browse

Success

```

[10:14:15.663] - ===== flash read jedec ID =====
[10:14:15.664] - Read flash jedec ID
[10:14:15.664] - readdata:
[10:14:15.664] - b'68401600'
[10:14:15.665] - Finished
[10:14:15.665] - flash config Not found,use default
[10:14:15.665] - jedec_id:684016
[10:14:15.665] - capacity_id:22
[10:14:15.665] - capacity:4.0M
[10:14:15.665] - get flash size: 0x00400000
[10:14:15.665] - ===== flash write status register =====
[10:14:15.665] - write_data 00
[10:14:15.666] - Write flash status register
[10:14:15.666] - Finished

```

Fig. 6.7: Write Register Interface

Flash Cube also provides some advanced programming features, which are realized by modifying the configuration file.

## 7.1 Support Fuzzy Matching of Firmware Paths

In the configuration file `flash_prog_cfg.ini` imported by the user, the firmware path can be similar to “`./build/build_out/helloworld*_$(CHIPNAME).bin`”, and the tool can match the test firmware that needs to be programmed.

The fuzzy matching method is used in the `flash_prog_cfg.ini` configuration file in the `examples/wifi/sta/wifi_ota` directory of the SDK.

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

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

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

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

(continues on next page)

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

- The filedir of [boot2] uses “. /build/build\_out/boot2\_\*.bin” fuzzy match to find the boot2\_bl616\_release\_v8.0.7.bin file in the build/build\_out directory.
- The filedir in [FW] uses “. /build/build\_out/wifi\_ota\*\_\$(CHIPNAME).bin” fuzzy match to find the wifi\_ota\_bl616.bin file in the build/build\_out directory, where \$(CHIPNAME) depends on the chip type selected for the Chip of the program interface.
- The filedir of [mfg] uses “. /build/build\_out/mfg\*.bin” fuzzy match to find the mfg\_bl616\_gu\_af8b0946f\_v2.26.bin file in the build/build\_out directory.

If more than one file is matched, the tool will indicate an error: “Error: Multiple files were matched!”

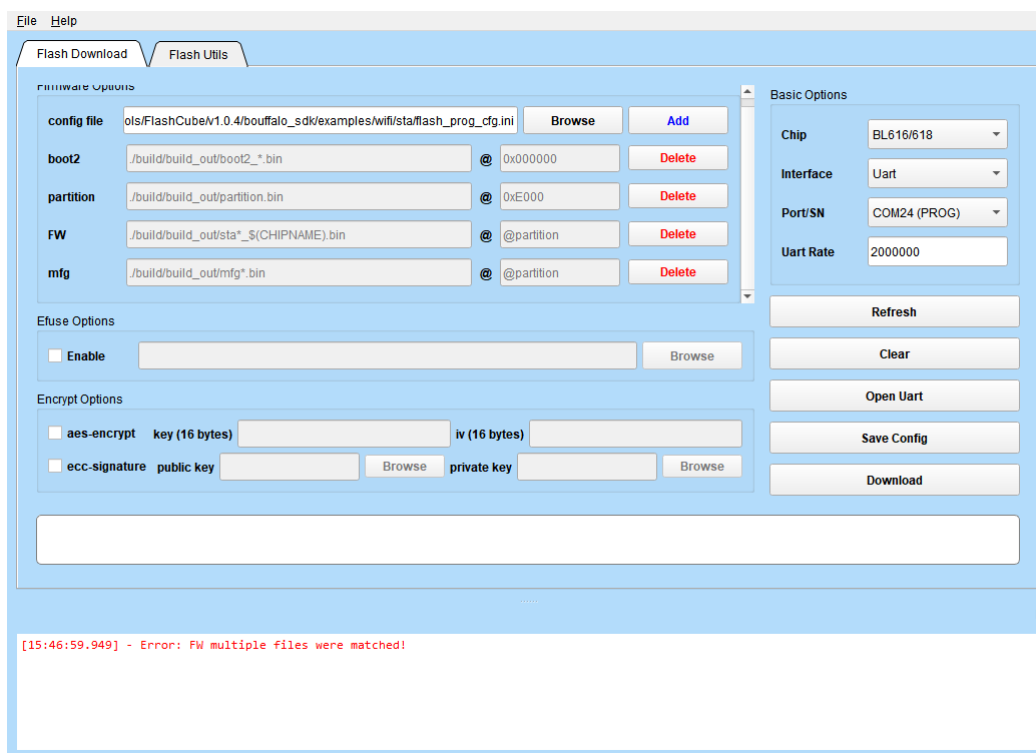


Fig. 7.1: Finding Firmware Errors

## 7.2 Support ISP Programming Mode

BLFlashCube supports ISP (In-System Programming) mode programming, please refer to the document “ISP\_-Download Instructions” for details.

Taking BL602 as an example, boot2\_isp\_mode controls whether to select the isp programming mode, and isp\_mode\_speed controls the baud rate configuration that triggers isp programming through communication with boot2. Among them, boot2\_isp\_mode is set in the user-defined programming configuration file, and isp\_mode\_speed is defined in “chips/bl602/eflash\_loader/eflash\_loader\_cfg.ini” of the tool. Modify “boot2\_isp\_mode = 0” in the custom configuration file to “boot2\_isp\_mode = 1”, and you can use the ISP programming mode.

The steps are as follows:

First, make sure that the Boot2 program has been programmed and started in the chip, and then modify “boot2\_isp\_mode = 1” in the configuration file. Click Download in the tool page, as shown in the figure below. During the programming process, it will prompt “Please Press Reset Key!”. At this time, the user needs to reset the chip within 5 seconds and see “read ready” or “isp ready”. The log indicates that the tool handshake is successful, and then just wait for the programming to complete.

If it is an automatically programmed board, after prompting “Please Press Reset Key!”, the tool will control the Reset pin to automatically reset the chip to perform the ISP programming process without manual operation.

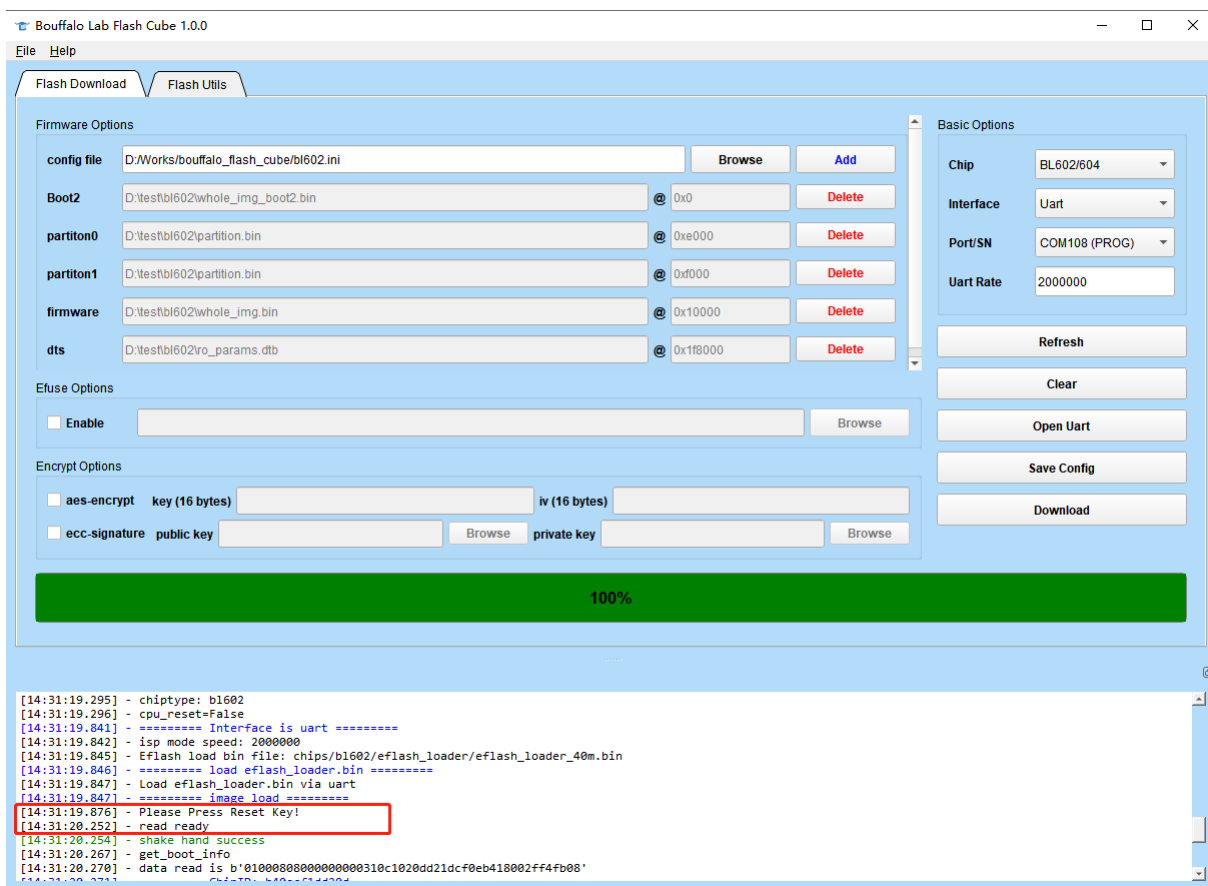


Fig. 7.2: ISP programming mode

## 7.3 Support Compression and Programming

In compression programming mode, the tool will compress each file being programmed. When transmitted to the chip through the serial port, the chip performs a decompression operation and writes the decompressed file into Flash. Compression and programming can greatly improve the programming speed. Among them, BL702 does not support compression and programming.

Taking BL602 as an example, open the “chips/bl602/eflash\_loader/eflash\_loader\_cfg.ini” file in the tool directory and modify “decompress\_write = false” to “decompress\_write = true”. When programming, the log shown in the figure below appears, indicating that the compression programming method is successfully used.

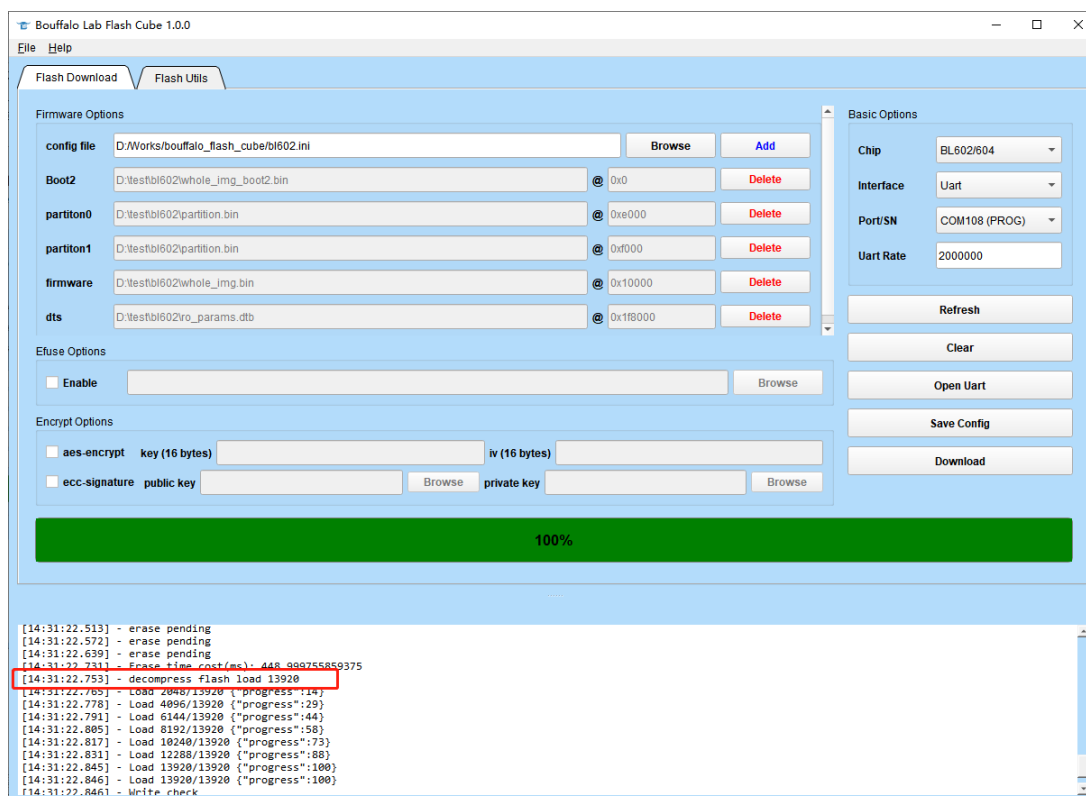


Fig. 7.3: Compression programming mode

## 7.4 Support eFuse Verification Selection

The Flash Cube tool supports eFuse programming, and efusedata.bin and efusedata\_mask.bin will be generated in the “build/build\_out” directory after the SDK is compiled. Among them, efusedata.bin is the bin file selected when eFuse is programmed, and efusedata\_mask.bin is used for eFuse verification.

Whether eFuse verification is configurable can be configured by modifying the factory\_mode parameter in the eflash\_loader\_cfg.conf file under the corresponding chip type (taking BL602 as an example, the file path is chips/bl602/eflash\_loader/eflash\_loader\_cfg.ini).

The default factory\_mode is false, which means that eFuse verification is not performed. When modifying factory\_mode = true, it means performing eFuse verification.



Take the repeated encryption and signing of the chip as an example. When the chip is programmed again, since the encrypted key or signature hash has been programmed and read-write protected, if `factory_mode = true`, it will display that the eFuse programming verification failed. This is a normal phenomenon, but it may cause confusion for customers.

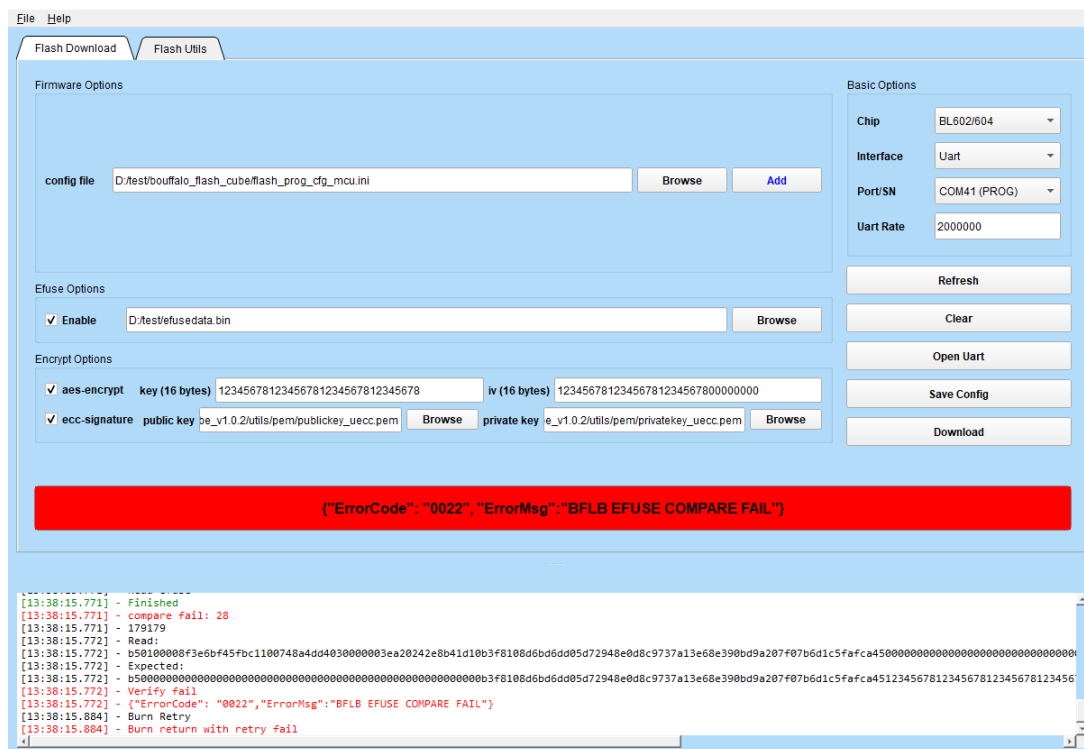


Fig. 7.4: Repeated encryption and signature programming eFuse verification failed

If you modify `factory_mode = false` and do not perform verification and programming, it will directly display that the programming is successful.

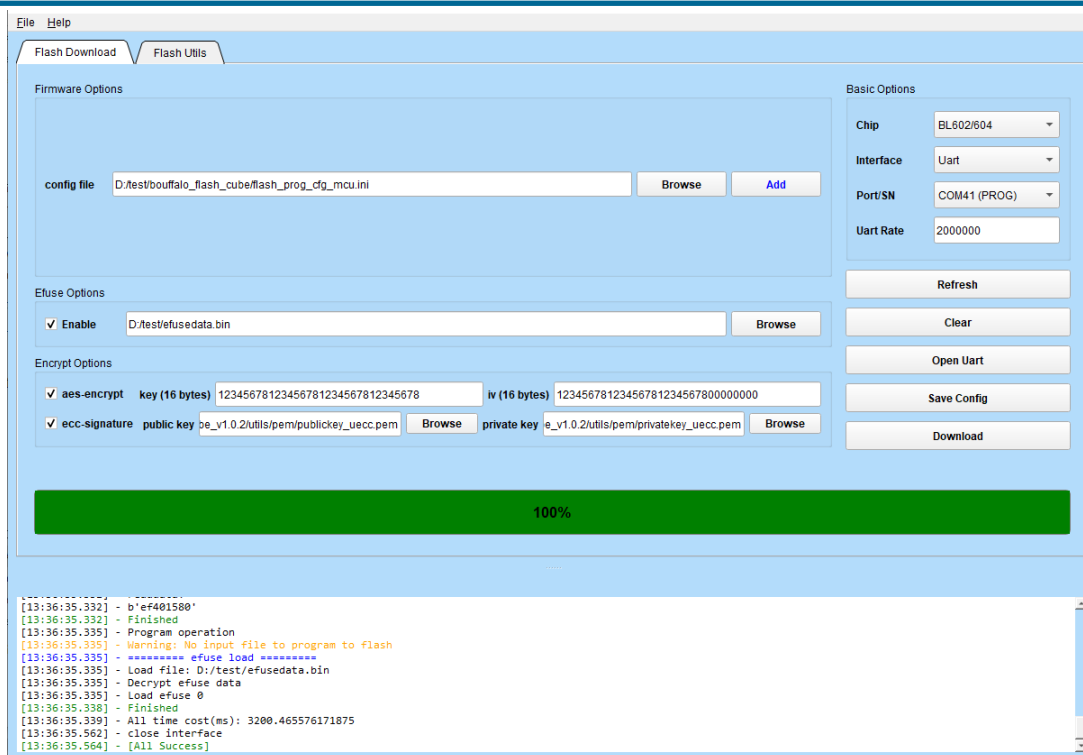


Fig. 7.5: Repeatedly encrypting and signing eFuse without verification

## 7.5 Support Modifying the Erasure Method During Programming

The tool supports Flash full erase and segmented erase, which are controlled by the erase parameter in the configuration file (flash\_prog\_cfg.ini) imported by the user.

The erase parameter under [cfg] in the configuration file is used to configure the erasure method of the tool. When erase = 0, it means direct programming without erasing. When erase = 1, it means erasing according to the programming address and content size during program. When erase = 2, it means all Flash will be erased before programming. The default programming mode in the tool is erase = 1, which erases according to the programming address and content size, and performs an erase operation before programming each file.

```
[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
```

Fig. 7.6: Erase according to the programming address and content size

When the programming mode is changed to erase = 2, the tool will erase all Flash before programming.

```
[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
```

Fig. 7.7: Full erase before programming

## 7.6 Support Skip Function for Erasing and Writing

When you do not want the specified area to be erased or written during flash programming, you can use the skip function to skip this area for programming. The skip parameter under [cfg] in the configuration file (flash\_prog\_cfg.ini) is used to set the skip function of tool erasing. Taking BL602 as an example, we do not want the address contents of 0x11000 ~ 0x12000 to be changed during the programming process. This can be achieved by modifying the value of skip\_mode. The first parameter is the starting address and the second parameter is the length.

Steps:

First, open the user-defined configuration file flash\_prog\_cfg.ini and modify “skip\_mode = 0x0, 0x0” to “skip\_mode = 0x11000, 0x1000” . The programming log after clicking the “Download” button is as shown below:

```
[16:06:14.252] - ===== programming D:\test\bl602\whole_img.bin to 0x10000
[16:06:14.255] - skip flash file, skip addr 0x0001000, 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
```

Fig. 7.8: The skip function of the IOT page

skip\_mode supports configuring multiple areas at the same time, separated by “;” .

Taking BL602 as an example, if you do not want the address contents of 0x11000 ~ 0x12000 and 0x13000 ~ 0x15000 to be changed during the programming process, you need to modify the value of skip\_mode in the configuration file flash\_prog\_cfg.ini to “skip\_mode = 0x11000, 0x1000; 0x13000, 0x2000” .

```
[16:00:20.419] - ===== programming D:\test\bl602\whole_img.bin to 0x10000
[16:00:20.423] - skip flash file, skip addr 0x0001000, 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 0x0001000, 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): 165.79760721875
[16:00:20.853] - decompress flash load 2048
[16:00:20.865] - Load 2048/2048 {"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
```

Fig. 7.9: The skip function of the IOT page

It can be seen from the programming log that the 0x11000 ~ 0x12000 and 0x13000 ~ 0x15000 areas will be skipped

during the programming process, and the contents of other areas will be erased and written separately.

## 7.7 Generate Mass Production Programming Files

During each programming, the tool will generate two files for mass production programming:

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

1. The whole\_flash\_data.bin file is a binary file. According to the partition table, all the image-related files to be programmed are arranged. Its content is completely consistent with the data layout in Flash. The composition of whole\_flash\_data.bin is as shown in the figure:

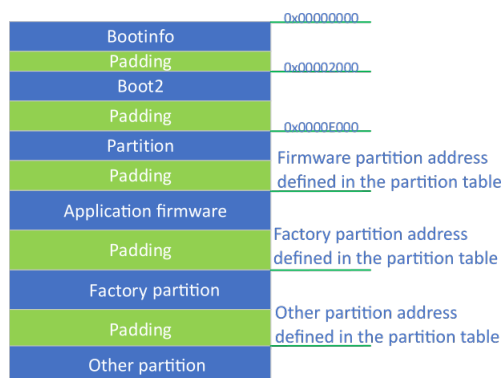


Fig. 7.10: whole\_flash\_data.bin composition

As can be seen from the figure, whole\_flash\_data.bin contains all bin files to be programmed and has been arranged according to the location of the partition table. This file can be directly programmed to the starting address of Flash using the single file mode of a Flash programmer or batch programming tool.

This file contains padding between different firmwares, resulting in a larger file. The disadvantage is that the programming time is long.

2. whole\_img.pack is a compressed file. It not only contains various files to be programmed, but also contains the configuration information of the files to be programmed. The composition of the compressed package is as shown in the figure below:

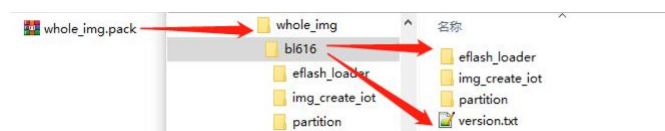


Fig. 7.11: whole\_img.pack composition

During mass production, you can import the development kit from the mass production programming tool interface. The mass production tool will decompress it on its own and program the files separately according to the configuration file. The programming speed is fast.

The generated whole\_flash\_data.bin and whole\_img.pack are stored in the “chips/bl616/img\_create” directory. Users can select mass production files according to their own circumstances.

| 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 |  |

Fig. 7.12: Generated Whole\_img image

## 7.8 BL602/BL702 Encryption and Signature Programming

With Flash Cube v1.0.8 and later versions, users can repeatedly program boards that have been encrypted and signed without having to provide the key and signature, just prepare eflash\_loader\_xx\_encrypt.bin in advance and put it into the eflash\_loader directory of the tool.

Take BL602 as an example, first copy the encrypted eflash\_loader\_40m\_encrypt.bin to the chips/bl602/eflash\_loader directory:

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

Fig. 7.13: Encrypt eflash\_loader\_encrypt.bin file

Open the Flash Cube’ s programming interface, and follow the following programming method without filling in the key and signature, you can successfully program the encrypted and signed board.

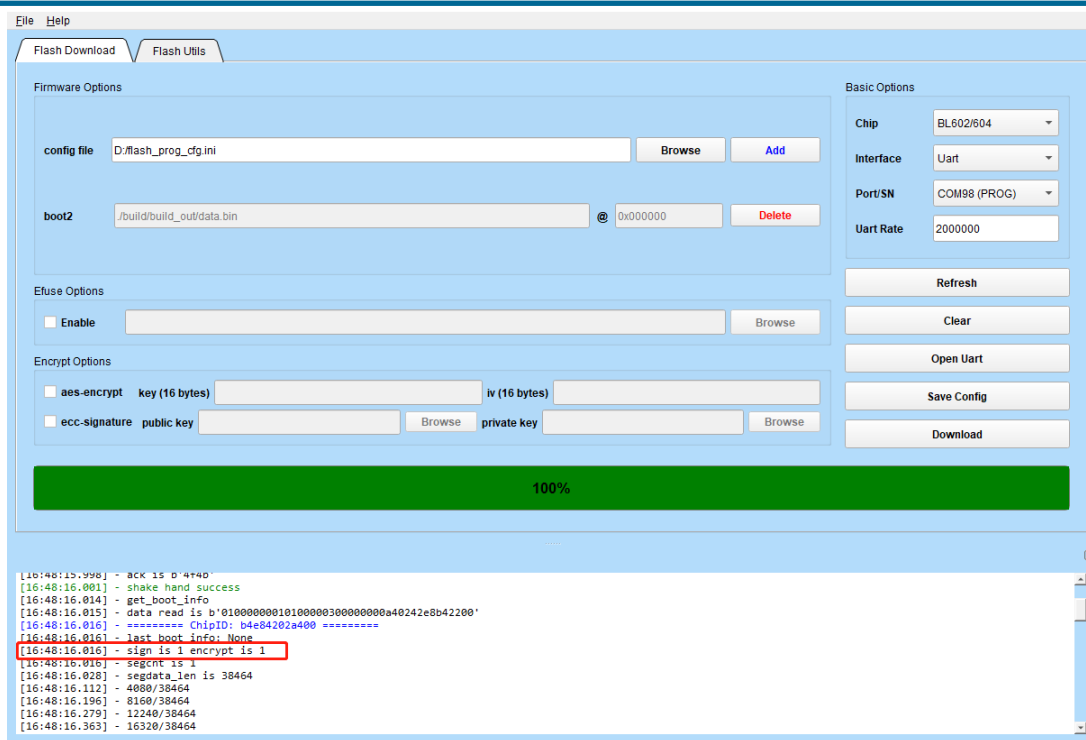


Fig. 7.14: Burning an encrypted and signed board

If the supplied eflash\_loader\_xx\_encrypt.bin key and signature do not match, the error “BFLB\_IMG\_SECTIONHEADER\_CRC\_ERROR” will be prompted.

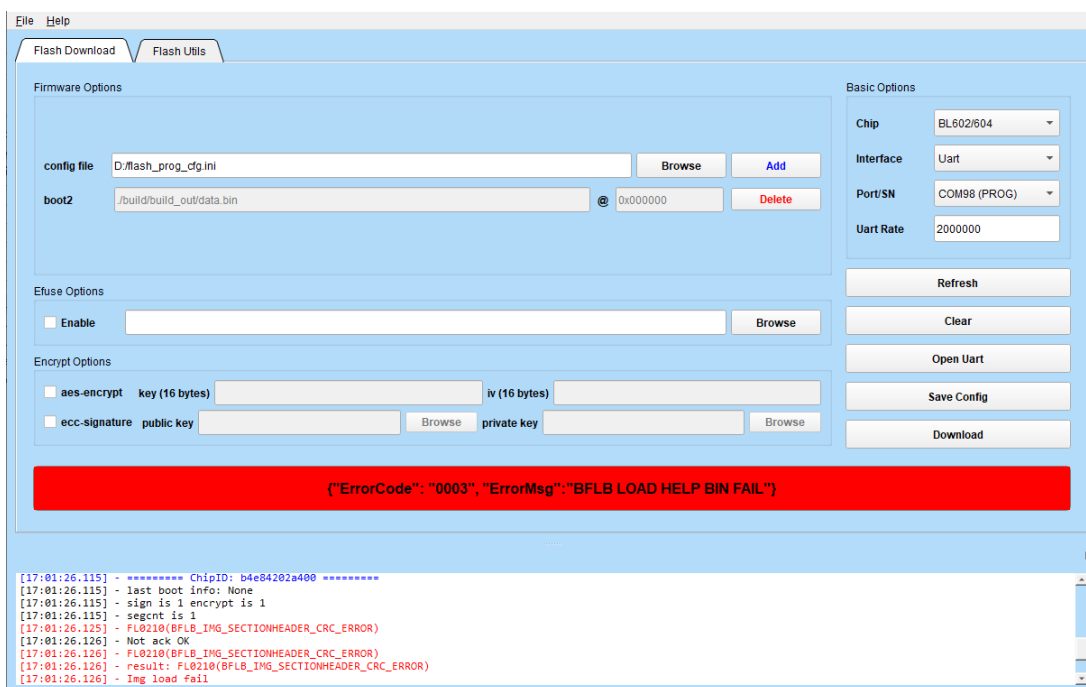


Fig. 7.15: Key mismatch burns boards that have been encrypted and countersigned



## 7.9 Add Preprocessing Function

If you want the Flash Cube to run a preprocessor before you burn or build a production firmware, you can add the `pre_program` and `pre_program_args` parameters to the custom burn configuration file.

- `pre_program` Fill path: The path of the preprocessor with the user-defined burning profile as the relative path.
- `pre_program_args` Fill in parameter: Multiple parameters are separated by Spaces. If the passed parameter is a path, support fuzzy matching(such as: ". /build\_out/wifi\_ota\*\_\$(CHIPNAME).bin" way, by the tool to match the need to burn the test firmware).

---

**Note:** `pre_program` The configuration item is automatically adapted to the platform, so you do not need to add a suffix(such as: .exe)

Windows ADAPTS to executable programs ending in .exe

Linux ADAPTS to executable programs ending in -ubuntu

Darwin ADAPTS to executable programs ending in -macos

---

For example, if the preprocessor is the `bflb_fw_post_proc` executable, perform the following steps:

Add `pre_program` and `pre_program_args` to the `flash_prog_cfg.ini` configuration file(Since `bflb_fw_post_proc.exe` and `flash_prog_cfg.ini` reside in the same directory, `pre_program=./bflb_fw_post_proc`).

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

Click Download in the tool page, as shown in the following figure. During the burning process, `bflb_fw_post_proc` will be run first to generate a common image, and then continue to burn according to the configuration items of `flash_prog_cfg.ini`.

```
# Actual preprocessing commands executed:
bflb_fw_post_proc.exe --chipname=bl616 --imgfile=./build out/wifi_ota_b1616.bin
```

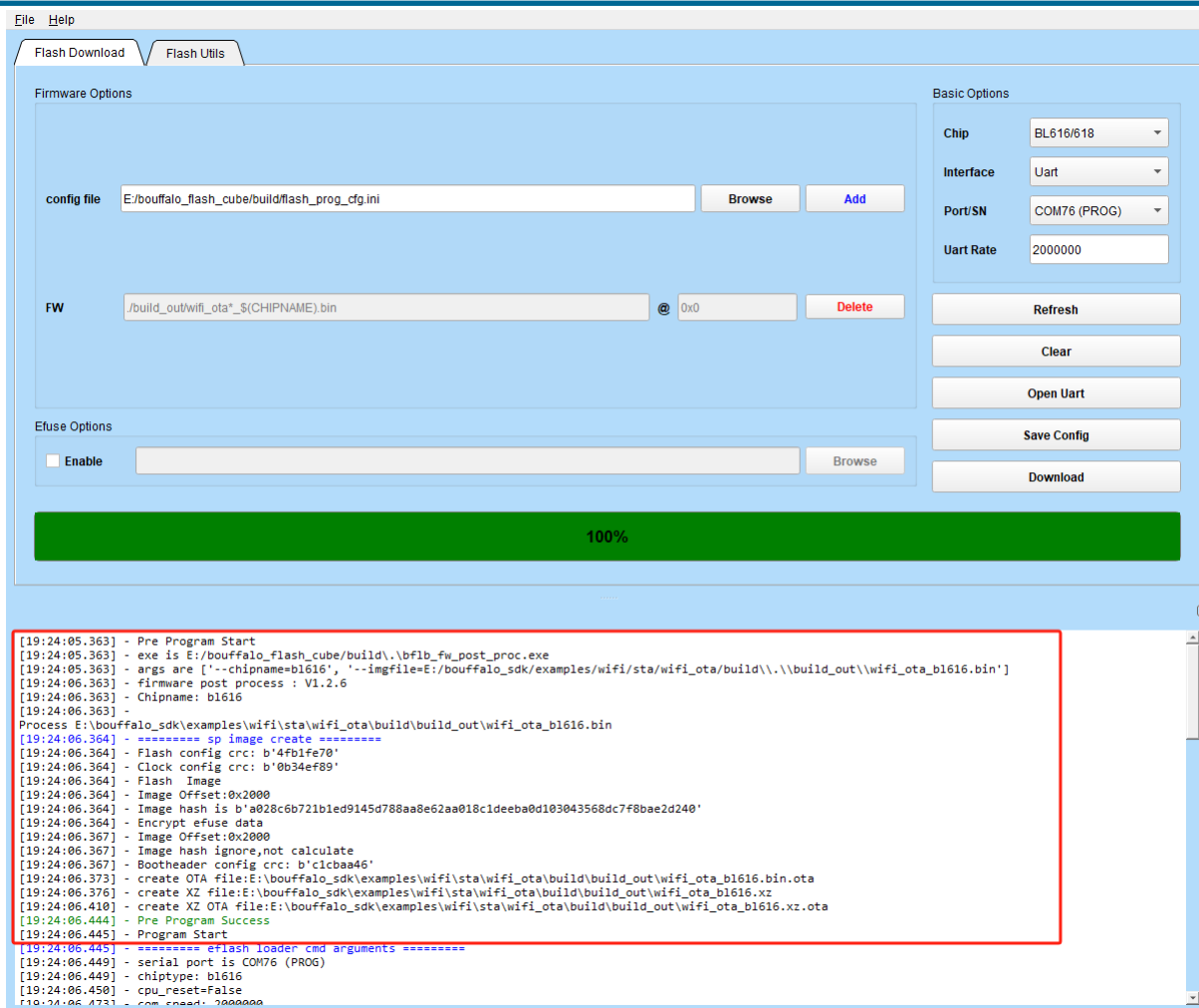


Fig. 7.16: Call pre\_program before burning

## 7.10 Supports user-defined efusedata.bin Encryption Keys

The tool provides higher security. You can customize the encryption key of efusedata.bin. To use this function, you need to use bflb\_img\_encryption\_tool.exe. Pass in the user-defined encryption key and use AES encryption to generate cfg.bin and the encrypted efusedata.bin.

If the root directory of the FlashCube tool contains cfg.bin, the user-defined and encrypted efusedata.bin can be successfully burned. Otherwise, the burning fails and an error message “Efuse crc check fail” is displayed.

## 8.1 flash\_prog\_cfg.ini Has the Highest Priority

The user-imported program configuration file (flash\_prog\_cfg.ini) contains a variety of function configurations, such as erase, skip\_mode, boot2\_isp\_mode and other functions. These functions are also configured in the directory corresponding to the chip type ( eflash\_loader/eflash\_loader\_cfg.conf ).

Among them, the erase, skip\_mode, and boot2\_isp\_mode functions are subject to the definitions in the programming configuration file imported by the user, and these configurations will be updated to the eflash\_loader\_cfg.conf file during programming. The updated eflash\_loader\_cfg.conf file is also used in the generated whole\_img.pack.

## 8.2 Program Option Name Length

The name field length of each partition in the partitioned table cannot exceed 10 characters.

## 8.3 Crystal Oscillator Type Default Setting

The crystal oscillator type of the BLFlashCube tool cannot be modified and is currently the default value set. Among them, BL602 is 40M, BL702 is 32M, and BL808/BL606P/BL616 is auto to automatically obtain the crystal oscillator type.

## 8.4 Firmware Exceeds Allocated Address Size

The tool will detect the programming address and the size of the programming file filled in by the user. When the address is repeated or the firmware programmed exceeds the allocated address, an error will be prompted.

Taking BL602 programming as an example, configure the programming address and programming file as shown below. When programming whole\_img.bin, the address position of the firmware will exceed 1 byte, and the tool prompts an error: Error: The file size exceeds the address space size!

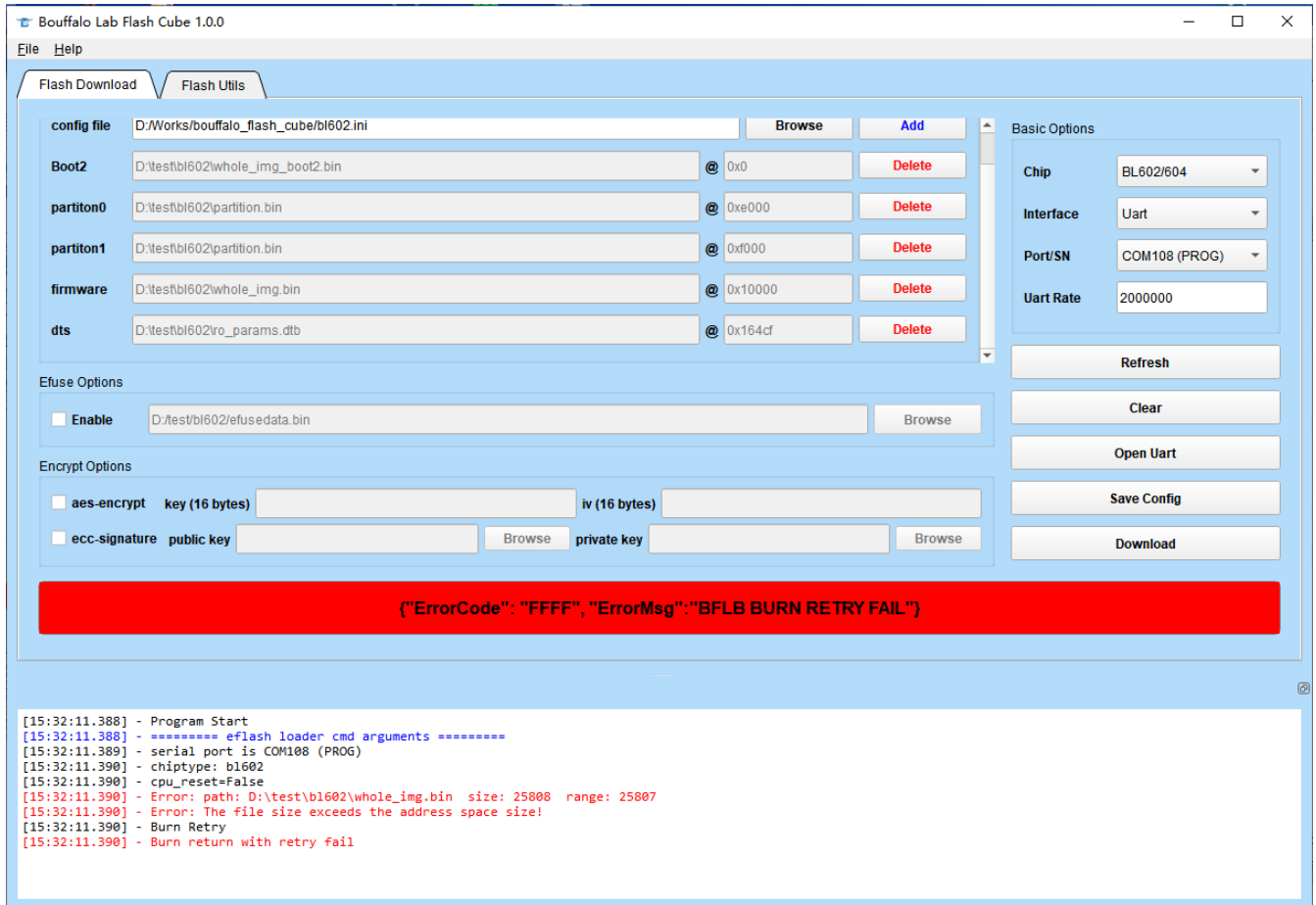


Fig. 8.1: Firmware size exceeded error

## 8.5 Firmware Exceeds Flash Size

Flash Cube v1.0.6 and later versions add a check that the programmed firmware exceeds the flash size. As shown in the figure below, if the programmed program is 2.1MB and the actual flash size is only 2M, the tool will prompt an error: “ErrorMsg” : “BFLB FLASH SIZE OVER FLOW” .

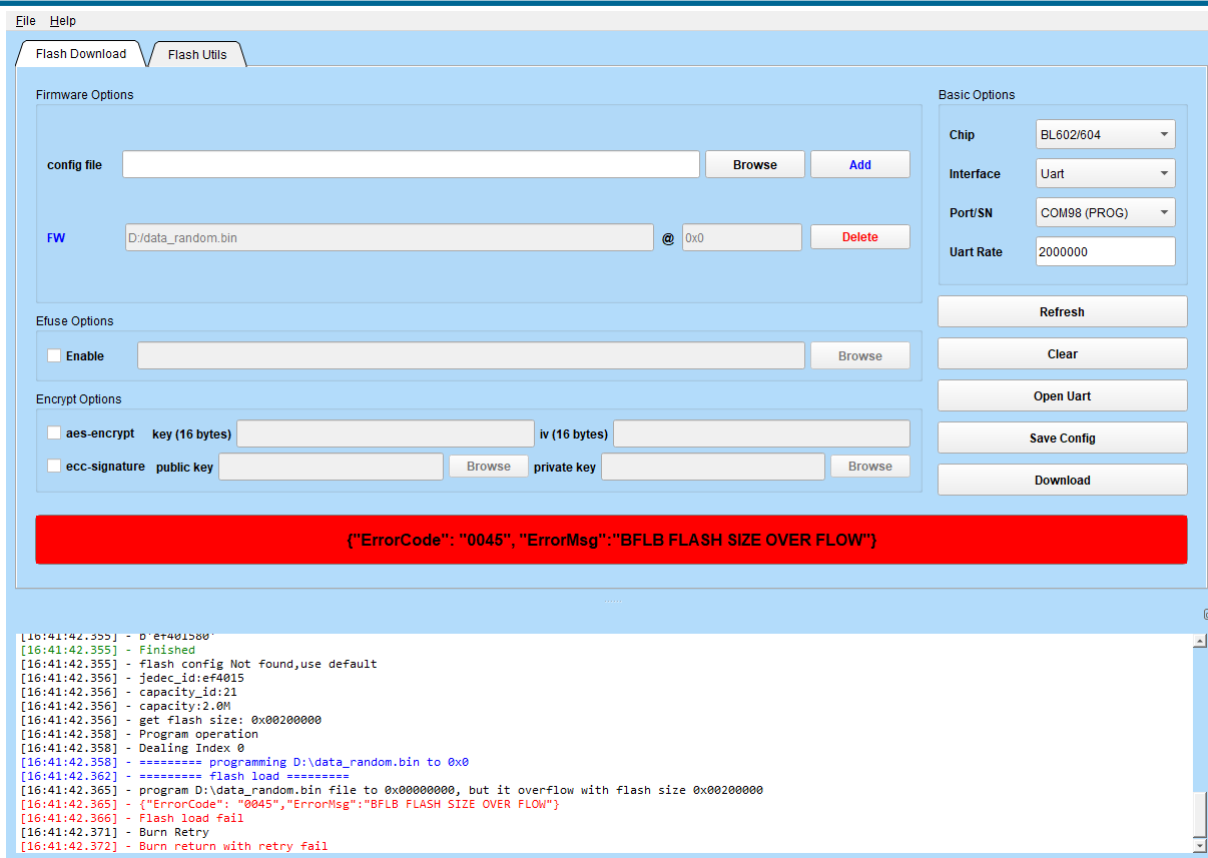


Fig. 8.2: Burning firmware exceeds flash size

## Revision History

Table 9.1: Revision history

| Version | Description  | Date       |
|---------|--|------------|
| 1.0     | Initial release  | 2022-10-18 |
| 1.1     | Add instructions for using the command line tool                                 | 2022-12-28 |
| 1.2     | Update command line -firmware parameters and encryption programming instructions | 2023-11-22 |
| 1.3     | Add eFuse read write operation   | 2024-3-8   |
| 1.4     | Update the ram reset efuse_encrypted command usage description                   | 2024-10-28 |
| 1.5     | Update the Flash Otp command description   | 2024-11-8  |
| 1.6     | Update the Flash Otp command with index and lock                                 | 2025-8-10  |