



**BL602**

## 在线编程 (ISP) 协议

版本: 1.1

版权 @ 2020

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

1	UART/SDIO 启动镜像格式	4
1.1	启动引脚	6
1.2	UART 握手	7
1.3	SDIO 握手	7
1.4	生成下载镜像文件	7
1.5	UART/SDIO 下载程序通信协议	8
1.5.1	Get boot info	8
1.5.2	Load boot header	9
1.5.3	Load public key (Optional)	11
1.5.4	Load signature (Optional)	12
1.5.5	Load AES IV (Optional)	12
1.5.6	Load Segment Header	13
1.5.7	Load Segment Data	13
1.5.8	Check image	14
1.5.9	Run image	14
1.5.10	错误应答帧	15
1.5.11	下载流程示意	16
2	Eflash_loader	18
2.1	下载并运行 Eflash_loader	18
2.2	Eflash_loader 通信协议	18
2.2.1	Chip Erase	19
2.2.2	Flash Erase	19
2.2.3	Flash Program	20
2.2.4	Flash Program Check	20

---

2.2.5	Flash Read . . . . .	21
2.2.6	SHA256 Read . . . . .	21
2.2.7	错误应答帧 . . . . .	22

## List of Figures

1.1	UART/SDIO 启动镜像 (不加密 & 不签名) . . . . .	5
1.2	UART/SDIO 启动镜像 (加密 & 签名) . . . . .	6
1.3	UART/SDIO 启动镜像 (加密 & 不签名) . . . . .	6
1.4	UART/SDIO 启动镜像 (签名 & 不加密) . . . . .	7
2.1	Eflash_Loader 启动镜像 . . . . .	19
2.2	Flash 地址空间示意 . . . . .	21

## UART/SDIO 启动镜像格式

BL602 系列芯片支持 UART/SDIO 启动。可以通过 UART/SDIO 接口将一段可执行程序下载到 RAM 中运行。下载程序的结构，需要满足 BL602 Bootrom 定义的格式。对于没有启动安全设定的应用程序，即不启用加密和签名的应用程序，其下载镜像的格式如下图所示：

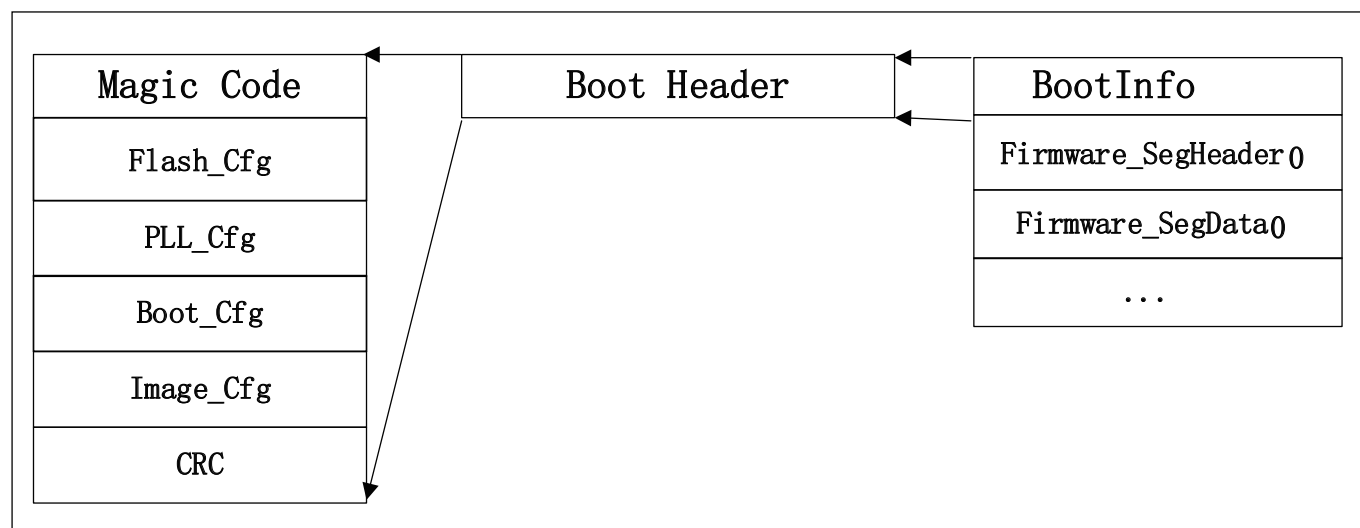


图 1.1: UART/SDIO 启动镜像 (不加密 & 不签名)

下载镜像由三部分组成：

- **BootInfo** 主要包含 **BootInfo** 的 **Magic Code**，Flash 的配置信息 (UART 下载不需要 Flash 信息，只是和 Flash 启动的镜像兼容)，PLL 配置信息，启动参数信息以及镜像配置信息等。
- **SegmentHeader** 下载程序或者数据段的段头信息，主要用于指定接下来传输的数据段，要放在内存的哪个地址以及数据长度等信息。
- **SegmenData** 下载程序或者数据段的主体。

**SegmentHeader** 和 **SegmenData** 可以有多个，具体的个数信息在 **BootInfo** 的镜像配置信息中设定。

对于启用加密和签名设定的下载镜像，其文件格式如下所示：

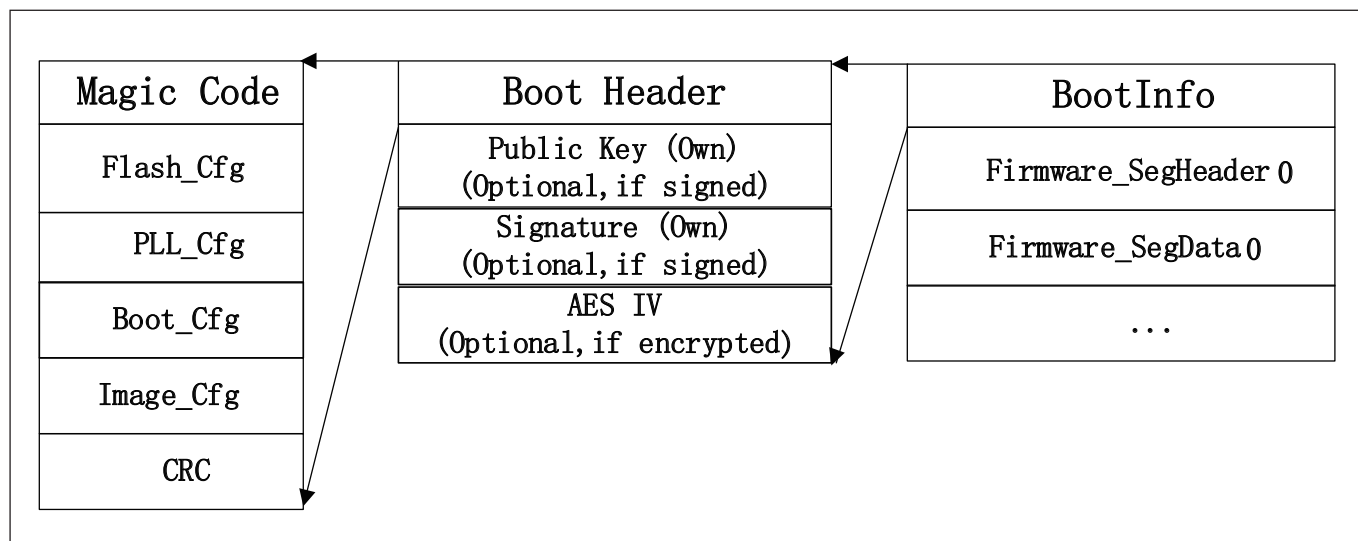


图 1.2: UART/SDIO 启动镜像 (加密 &amp; 签名)

与普通下载镜像相比，加密和签名的镜像，**BootInfo** 中需要包含公钥，签名以及 **AES IV** 等信息。对于启用加密但是不签名的下载镜像，文件格式如下所示：

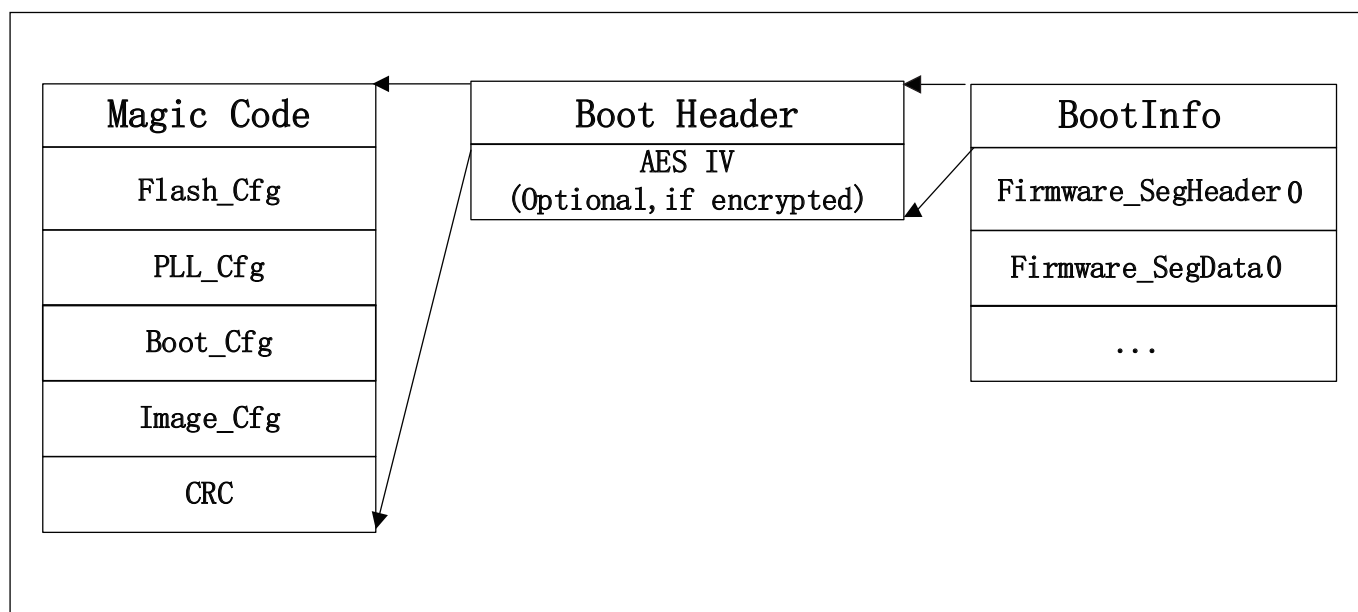


图 1.3: UART/SDIO 启动镜像 (加密 &amp; 不签名)

对于启用签名但是不启用加密的下载镜像，文件格式如下所示：

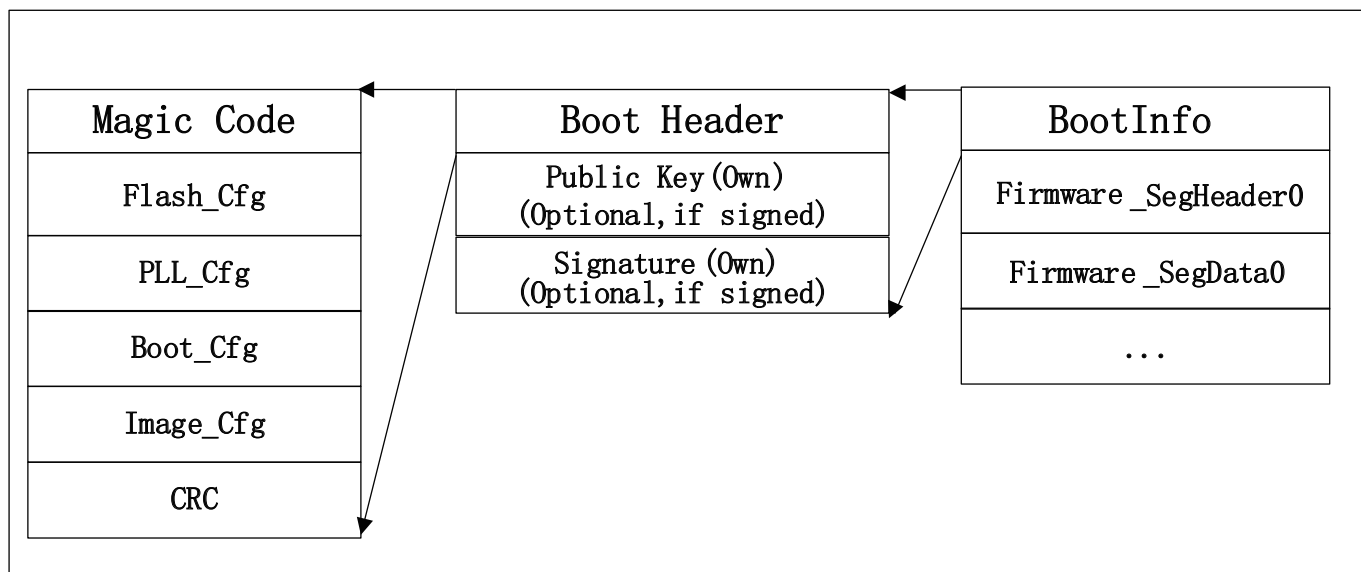


图 1.4: UART/SDIO 启动镜像 (签名 &amp; 不加密)

## 1.1 启动引脚

BL602 系列支持一路 UART，一路 SDIO 启动。

表 1.1: UART/SDIO 启动引脚分配

GPIO 引脚	Function	Comments
GPIO8	Boot Pin	
GPIO7	BL602 UART RXD	UART Channel1
GPIO16	BL602 UART TXD	
GPIO0	SDIO_CLK	SDIO Channel
GPIO1	SDIO_CMD	
GPIO2	SDIO_DATA0	
GPIO3	SDIO_DATA1	
GPIO4	SDIO_DATA2	
GPIO5	SDIO_DATA3	

若想要从 UART/SDIO 启动，需要将 GPIO8 拉高，然后复位芯片，Bootrom 会依次扫描 UART 和 SDIO 这两个接口，并在该接口等待握手信号，握手超时（2ms）后会进行下一个接口的扫描，如果在某个接口上握手成功，则进入接收数据处理流程，在数据处理期间，一旦发送错误或者超时（2s），会进行下一个接口的扫描，如此依次循环进行，直到收到合法的启动镜像，完成启动任务。

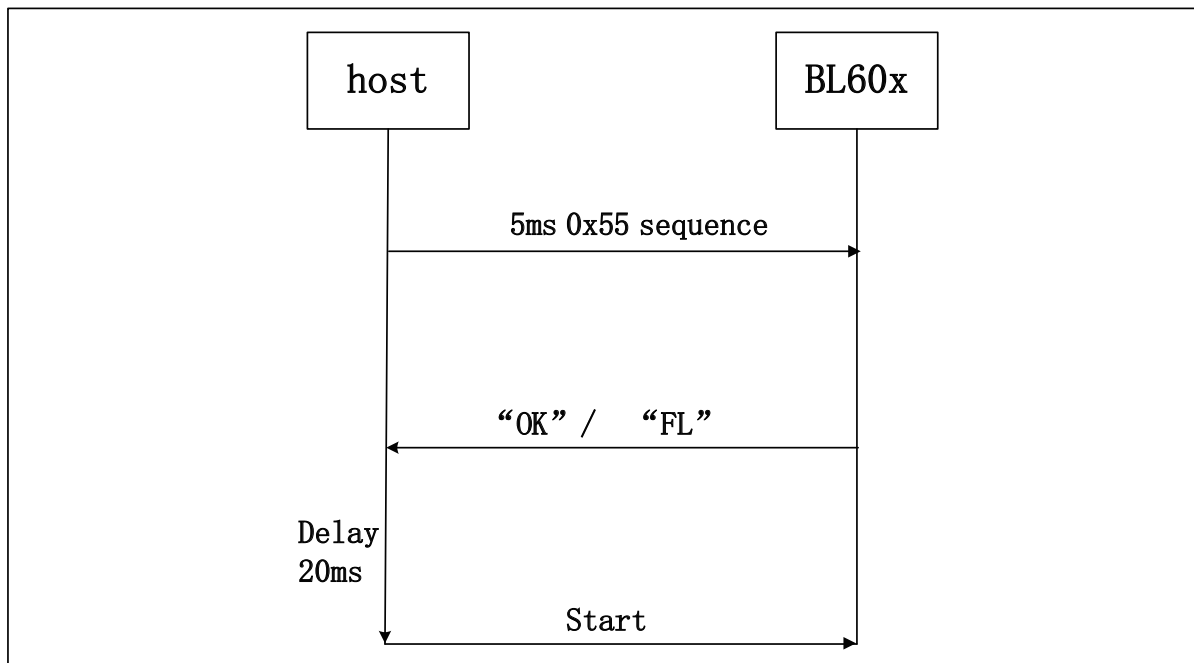
## 1.2 UART 握手

UART 通信的配置为 1bit 起始位，8bit 数据位，1bit 停止位，无奇偶校验位。

Bootrom 从 UART/SDIO 启动后，会循环检测 GPIO7 引脚的电平变化，当主机发送 0x55 数据串被捕捉后，Bootrom 开始计算当前串口波特率，并根据检测结果设定 UART 寄存器的值，并以当前波特率回复 “OK”。主机收到 “OK” 后可以正常的通信。UART 通信超时时间是 2s，Bootrom 在回复 “OK” 以后的 2s 内如果没有收到任何数据，或者在通信过程中，出现 2s 以内没有收到任何数据，则认为是通信超时，超时以后重新进入握手流程。

主机发送握手数据时间建议是 5ms，以便让 Bootrom 有充足的时间检测到握手信号，主机收到 “OK” 以后，建议延时 20ms 再通信，以防止后续的通信数据与之前的握手数据混在一起。

由于从 UART/SDIO 启动时，Bootrom 使用 RC32M 时钟，握手波特率建议不超过 500K。握手过程如下：



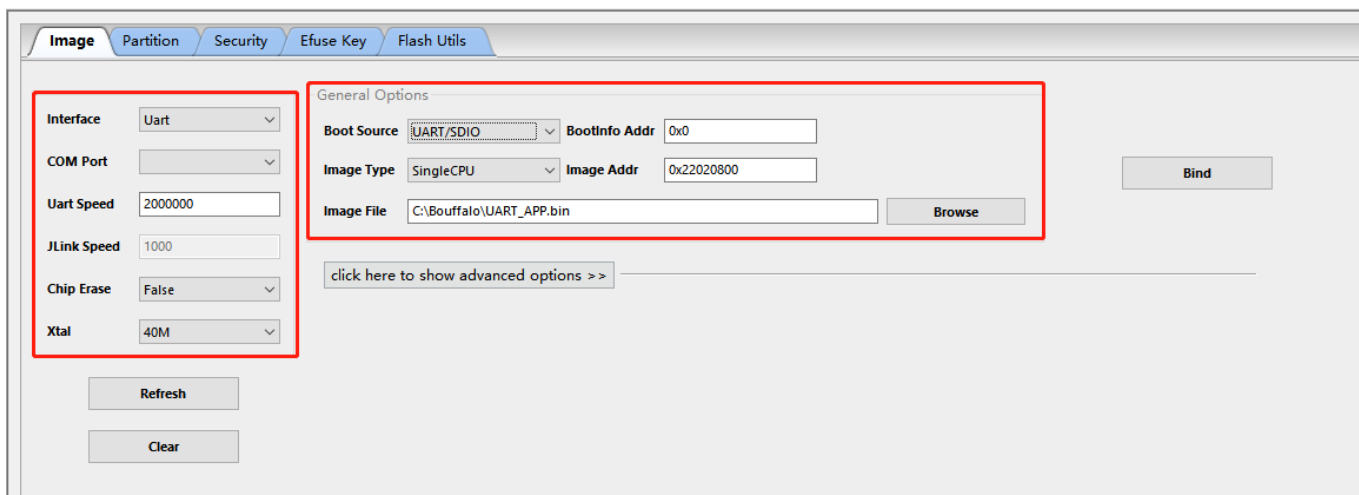
## 1.3 SDIO 握手

Bootrom 会等待 SDIO Host 写握手寄存器（SDU\_BASE+0x160），当 SDIO 查询到握手寄存器被写 1 以后，认为握手成功。Bootrom 会等待主机发送数据，并根据接收到数据命令进行处理，如果发生超时（2s 内没有接收到数据）会重新进入握手流程。

## 1.4 生成下载镜像文件

博流提供了 UART/SDIO 下载镜像生成工具，用户可通过下载 [Bouffalo Lab Dev Cube For Windows](#)，获取最新的 Dev Cube，运行 BLDevCube.exe，在 Chip Type 中选择 BL602/604，进入烧写界面。在 View 菜单中选择 MCU 选项，进入 MCU 程序下载界面。





如若只生成 UART/SDIO 下载镜像，则可只配置右侧的烧录镜像参数，具体配置如下：

- **Boot Source:** 选择 UART/SDIO，表示生成 UART/SDIO 启动镜像
- **BootInfo Addr:** 程序启动参数的存放地址，此处填写 0x0 即可
- **Image Type:** 默认为 SingleCPU
- **Image Addr:** 应用程序的下载地址，用户可根据实际程序的运行地址填写即可，例如 0x22020800
- **Image File:** 选择用户编译生成中的 RAM 程序。

完成选项配置后，点击 **Create&Download** 按钮，会生成对应的镜像文件。生成的文件路径为: bl602/img\_create2/img\_if.bin。img\_if.bin 就是满足 UART/SDIO 启动镜像格式的文件。

如需启用加密和签名功能，展开工具中的 **advanced options** 选项，完成配置后，同样点击 **Create&Download** 按钮即可。

## 1.5 UART/SDIO 下载程序通信协议

Bootrom 在完成 UART/SDIO 通信握手后，即可进入正常的下载程序通信流程，下面详细介绍通信过程。需要注意的是，Bootrom 所能接收的协议数据最大长度为 4096bytes。

### 1.5.1 Get boot info

表 1.2: Host->BL602

cmdId(1byte)	Rsvd(1byte)	Len_lsb(1byte)	Len_msb(1byte)
0x10	0x00	0x00	0x00

表 1.3: BL602->Host

'OK' (2bytes)	Len_lsb(1byte)	Len_msb(1byte)	BootRom Version(4bytes)	OTP info(16bytes)
0x4F 0x4B	0x14	0x00		

这是主机与 BL602 通信的第一条指令，读取 BL602 相关信息。主机要根据 **sign\_type** 判断 BL602 是否要求接收签名的镜像；根据 **encrypted** 判断 BL602 是否要求接收加密的镜像。如果已知芯片没有启动加密和签名，则可跳过对该信息的解析。

表 1.4: 判断是否要签名

	2b' 00	其它
sign_type	不签名	签名
encrypted	不加密	加密

## 1.5.2 Load boot header

表 1.5: Host->BL602

cmdId(1byte)	Rsvd(1byte)	Len_lsb(1byte)	Len_msb(1byte)	BootHeader(176bytes)
0x11	0x00	0xb0	0x00	

表 1.6: BL602->Host

'OK' (2bytes)
0x4F 0x4B

176bytes 的 BootHeader 结构体如下：

```
__PACKED_STRUCT boot_flash_cfg_t
{
    uint32_t magiccode;          /*'FCFG'*/
    SPI_Flash_Cfg_Type cfg;
    uint32_t crc32;
};

__PACKED_STRUCT sys_clk_cfg_t
{
    uint8_t xtal_type;
```

(下页继续)

(续上页)

```
uint8_t pll_clk;
uint8_t hclk_div;
uint8_t bclk_div;

uint8_t flash_clk_type;
uint8_t flash_clk_div;
uint8_t rsvd[2];
};

__PACKED_STRUCT boot_clk_cfg_t
{
    uint32_t magiccode;          /*'PCFG'*/

    struct sys_clk_cfg_t cfg;

    uint32_t crc32;
};

__PACKED_STRUCT bootheader_t
{
    uint32_t magiccode;          /*'BFXP'*/
    uint32_t revision;

    struct boot_flash_cfg_t flashCfg;
    struct boot_clk_cfg_t clkCfg;
    __PACKED_UNION {
        __PACKED_STRUCT {
            uint32_t sign          : 2; /* [1: 0] for sign*/
            uint32_t encrypt_type  : 2; /* [3: 2] for encrypt */
            uint32_t key_sel       : 2; /* [5: 4] for key sel in boot interface*/
            uint32_t rsvd6_7       : 2; /* [7: 6] for encrypt*/
            uint32_t no_segment    : 1; /* [8] no segment info */
            uint32_t cache_enable  : 1; /* [9] for cache */
            uint32_t notload_in_bootrom : 1; /* [10] not load this img in bootrom */
            uint32_t aes_region_lock : 1; /* [11] aes region lock */
            uint32_t cache_way_disable : 4; /* [15: 12] cache way disable info*/
            uint32_t crc_ignore    : 1; /* [16] ignore crc */
            uint32_t hash_ignore   : 1; /* [17] hash crc */
            uint32_t halt_ap       : 1; /* [18] halt ap */
            uint32_t rsvd19_31     : 13; /* [31:19] rsvd */
        };
    };
};
```

(下页继续)

(续上页)

```

    } bval;
    uint32_t wval;
    }bootcfg ;

    uint32_t segment_cnt;

    uint32_t bootentry;      /* entry point of the image*/

    uint32_t flashoffset;

    uint8_t hash[BFLB_BOOTROM_HASH_SIZE];    /*hash of the image*/

    uint32_t rsv1;
    uint32_t rsv2;
    uint32_t crc32;
};

```

## 1.5.3 Load public key (Optional)

表 1.7: Host->BL602

cmdId(1byte)	Rsvd(1byte)	Len_lsb(1byte)	Len_msb(1byte)	PKey(68bytes)
0x12	0x00	0x44	0x00	

表 1.8: BL602->Host

'OK' (2bytes)
0x4F 0x4B

只有当镜像有签名的时候主机才发送这个命令。没有启动签名的情况下，应跳过这个命令的发送。68 字节的 Public Key 结构体如下：

```

__PACKED_STRUCT pkey_cfg_t
{
    uint8_t ekeyx[32];      /*ec key in boot info*/
    uint8_t ekeyy[32];      /*ec key in boot info*/
    uint32_t crc32;
};

```

### 1.5.4 Load signature (Optional)

表 1.9: Host->BL602

cmdId(1byte)	Rsvd(1byte)	Len_lsb(1byte)	Len_msb(1byte)	Signature(Nbytes)
0x14	0x00	N&0xFF	(N&0xFF00)>>8	

表 1.10: BL602->Host

'OK' (2bytes)
0x4F 0x4B

只有当镜像有签名的时候主机才发送这个命令。没有启动签名的情况下，应跳过这个命令的发送。**Signature** 的有效长度不是固定的，其结构体示意如下：

```
__PACKED_STRUCT sign_cfg_t
{
    uint32_t sig_len;
    uint8_t signature[sig_len];
    uint32_t crc32;
};
```

主机发送 signature 时可先读取 sig\_len 获得需要发送的 signature 长度为 sig\_len+8。

### 1.5.5 Load AES IV (Optional)

表 1.11: Host->BL602

cmdId(1byte)	Rsvd(1byte)	Len_lsb(1byte)	Len_msb(1byte)	AES IV(20bytes)
0x16	0x00	0x14	0x00	

表 1.12: BL602->Host

'OK' (2bytes)
0x4F 0x4B

只有当镜像有加密的时候主机才发送这个命令。没有启动加密的情况下，应跳过这个命令的发送。

20bytes 的 AES IV 结构体如下：

```
__PACKED_STRUCT aesiv_cfg_t
{
    uint8_t aesiv[16];
    uint32_t crc32;
};
```

## 1.5.6 Load Segment Header

表 1.13: Host->BL602

cmdId(1byte)	Rsvd(1byte)	Len_lsb(1byte)	Len_msb(1byte)	Seg header(16bytes)
0x17	0x00	0x10	0x00	

表 1.14: BL602->Host

'OK' (2bytes)
0x4F 0x4B

UART/SDIO 启动镜像支持多个 segment，每个 segment 的数据和代码可以由启动程序加载到 Segheader 指定的地址上。而镜像中 segment 的个数由 BootHeader 中的 segment\_cnt(具体参考 2.2) 成员决定。主机需要在 Load boot header 过程中记录下这个变量，然后循环 segment\_cnt 次 Load Segment Header 与 Load Segment Data。

16bytes 的 seg header 用下面的结构体描述：

```
__PACKED_STRUCT segment_header_t
{
    uint32_t destaddr;
    uint32_t len;
    uint32_t rsvd;
    uint32_t crc32;
};
```

## 1.5.7 Load Segment Data

表 1.15: Host->BL602

cmdId(1byte)	Rsvd(1byte)	Len_lsb(1byte)	Len_msb(1byte)	Seg Data(Nbytes)
0x18	0x00	N&0xFF	(N&0xFF00)>>8	

表 1.16: BL602-&gt;Host

‘OK’ (2bytes)
0x4F 0x4B

对于一个 **Segment Data**，由于一个协议帧 4096bytes 的限制，可能需要多次发送 **Load Segment Data** 来传输数据。这里要保证多次传递的数据帧中数据长度之和与 **Segment Header** 中描述的 **len** 相等。

### 1.5.8 Check image

表 1.17: Host-&gt;BL602

cmdId(1byte)	Rsvd(1byte)	Len_lsb(1byte)	Len_msb(1byte)
0x19	0x00	0x00	0x00

表 1.18: BL602-&gt;Host

‘OK’ (2bytes)
0x4F 0x4B

镜像下载到 **RAM** 后，必须检查镜像的完整性与合法性。

### 1.5.9 Run image

表 1.19: Host-&gt;BL602

cmdId(1byte)	Rsvd(1byte)	Len_lsb(1byte)	Len_msb(1byte)
0x1A	0x00	0x00	0x00

表 1.20: BL602-&gt;Host

‘OK’ (2bytes)
0x4F 0x4B

在 **Check image** 命令返回 **OK** 的情况下,通过该命令可以运行下载到 **RAM** 中的镜像,BL602 执行该命令后就从 **UART/SDIO** 启动程序跳转到下载的镜像程序中去运行。

### 1.5.10 错误应答帧

以上 BL602Host 的应答帧都是正确情况时的回复，如果在通信过程中出现错误，Bootrom 错误返数据格式如下，用户可根据错误代码查询错误原因：

表 1.21: Host->BL602

'FL' (2bytes)	Error_Code_LSB(1byte)	Error_Code_MSB(1byte)
0x46 0x4C		

Error\_Code 列举如下：

```
/*error code definition*/
typedef enum tag_bootrom_error_code_t
{
    BFLB_BOOTROM_SUCCESS=0x00,

    /*flash*/
    BFLB_BOOTROM_FLASH_INIT_ERROR=0x0001,
    BFLB_BOOTROM_FLASH_ERASE_PARA_ERROR=0x0002,
    BFLB_BOOTROM_FLASH_ERASE_ERROR=0x0003,
    BFLB_BOOTROM_FLASH_WRITE_PARA_ERROR=0x0004,
    BFLB_BOOTROM_FLASH_WRITE_ADDR_ERROR=0x0005,
    BFLB_BOOTROM_FLASH_WRITE_ERROR=0x0006,
    BFLB_BOOTROM_FLASH_BOOT_PARA=0x0007,

    /*cmd*/
    BFLB_BOOTROM_CMD_ID_ERROR =0x0101,
    BFLB_BOOTROM_CMD_LEN_ERROR=0x0102,
    BFLB_BOOTROM_CMD_CRC_ERROR=0x0103,
    BFLB_BOOTROM_CMD_SEQ_ERROR=0x0104,

    /*image*/
    BFLB_BOOTROM_IMG_BOOTHEADER_LEN_ERROR=0x0201,
    BFLB_BOOTROM_IMG_BOOTHEADER_NOT_LOAD_ERROR=0x0202,
    BFLB_BOOTROM_IMG_BOOTHEADER_MAGIC_ERROR=0x0203,
    BFLB_BOOTROM_IMG_BOOTHEADER_CRC_ERROR=0x0204,
    BFLB_BOOTROM_IMG_BOOTHEADER_ENCRYPT_NOTFIT=0x0205,
    BFLB_BOOTROM_IMG_BOOTHEADER_SIGN_NOTFIT=0x0206,
    BFLB_BOOTROM_IMG_SEGMENT_CNT_ERROR=0x0207,
    BFLB_BOOTROM_IMG_AES_IV_LEN_ERROR=0x0208,
```

(下页继续)



(续上页)

```
BFLB_BOOTROM_IMG_AES_IV_CRC_ERROR=0x0209,  
BFLB_BOOTROM_IMG_PK_LEN_ERROR=0x020a,  
BFLB_BOOTROM_IMG_PK_CRC_ERROR=0x020b,  
BFLB_BOOTROM_IMG_PK_HASH_ERROR=0x020c,  
BFLB_BOOTROM_IMG_SIGNATURE_LEN_ERROR=0x020d,  
BFLB_BOOTROM_IMG_SIGNATURE_CRC_ERROR=0x020e,  
BFLB_BOOTROM_IMG_SECTIONHEADER_LEN_ERROR=0x020f,  
BFLB_BOOTROM_IMG_SECTIONHEADER_CRC_ERROR=0x0210,  
BFLB_BOOTROM_IMG_SECTIONHEADER_DST_ERROR=0x0211,  
BFLB_BOOTROM_IMG_SECTIONDATA_LEN_ERROR=0x0212,  
BFLB_BOOTROM_IMG_SECTIONDATA_DEC_ERROR=0x0213,  
BFLB_BOOTROM_IMG_SECTIONDATA_TLEN_ERROR=0x0214,  
BFLB_BOOTROM_IMG_SECTIONDATA_CRC_ERROR=0x0215,  
BFLB_BOOTROM_IMG_HALFBAKED_ERROR=0x0216,  
BFLB_BOOTROM_IMG_HASH_ERROR=0x0217,  
BFLB_BOOTROM_IMG_SIGN_PARSE_ERROR=0x0218,  
BFLB_BOOTROM_IMG_SIGN_ERROR=0x0219,  
BFLB_BOOTROM_IMG_DEC_ERROR=0x021a,  
BFLB_BOOTROM_IMG_ALL_INVALID_ERROR=0x021b,  
  
/*IF*/  
BFLB_BOOTROM_IF_RATE_LEN_ERROR=0x0301,  
BFLB_BOOTROM_IF_RATE_PARA_ERROR=0x0302,  
BFLB_BOOTROM_IF_PASSWORDERROR=0x0303,  
BFLB_BOOTROM_IF_PASSWORDCLOSE=0x0304,  
  
/*MISC*/  
BFLB_BOOTROM_PLL_ERROR=0xfffc,  
BFLB_BOOTROM_INVASION_ERROR=0xfffd,  
BFLB_BOOTROM_POLLING=0xfffe,  
BFLB_BOOTROM_FAIL=0xffff,
```

```
}bootrom_error_code_t;
```

### 1.5.11 下载流程示意

对于一个没有启用加密和签名的程序，在只有一个 **segment** 的时候，其下载流程示意如下：

1. 设置 BL602 从 UART/SDIO 启动
2. 打开串口, 设置通信波特率, 打开要下载的文件 `fp=open("img_if.bin","rb")`
3. 发送 5ms 的握手信号, `UART_Send(0x555555...)`
4. 等待接收 602 的 OK 应答, 延时 20ms
5. 发送 `get boot info` 命令
6. 等待接收 4+20 字节的应答
7. 读取 176 字节的数据, `data=fp.read(176)`, 使用 `load boot header` 命令发送 176 字节的 `BootHeader`
8. 等待接收 OK 应答
9. 读取 16 字节的数据, `data=fp.read(16)`, 解析 `SegmentData` 的总长度 `segDataLen`, 使用 `load segment header` 命令发送 16 字节的 `SegmentHeader`
10. 等待接收 OK 应答
11. `sendDataLen=0;`  
    `while sendDataLen<segDataLen:`  
        `readDataLen=segDataLen-sendDataLen`  
        `if readDataLen>4096-4:`  
            `readDataLen=4096-4:`  
        读取 `readDataLen` 字节数据, `data=fp.read(readDataLen)`  
        使用 `load segment data` 命令发送 `readDataLen` 字节的 `SegmentData`  
        `sendDataLen+=readDataLen`  
        等待接收 OK 应答
12. 发送 `Check image` 命令检查镜像, 等待接收 OK 应答
13. 发送 `Run image` 命令运行程序, 等待接收 OK 应答

以上过程中, 如 `Bootrom` 返回错误, 则终止下载流程。

Eflash\_loader 是由博流提供的用于 Flash 烧写、读取、校验的可执行程序，可以通过 UART/SDIO 下载到 RAM 中运行。Eflash\_Loader 镜像没有加密与签名，并且只有一个 segment，其镜像结构如下图所示：

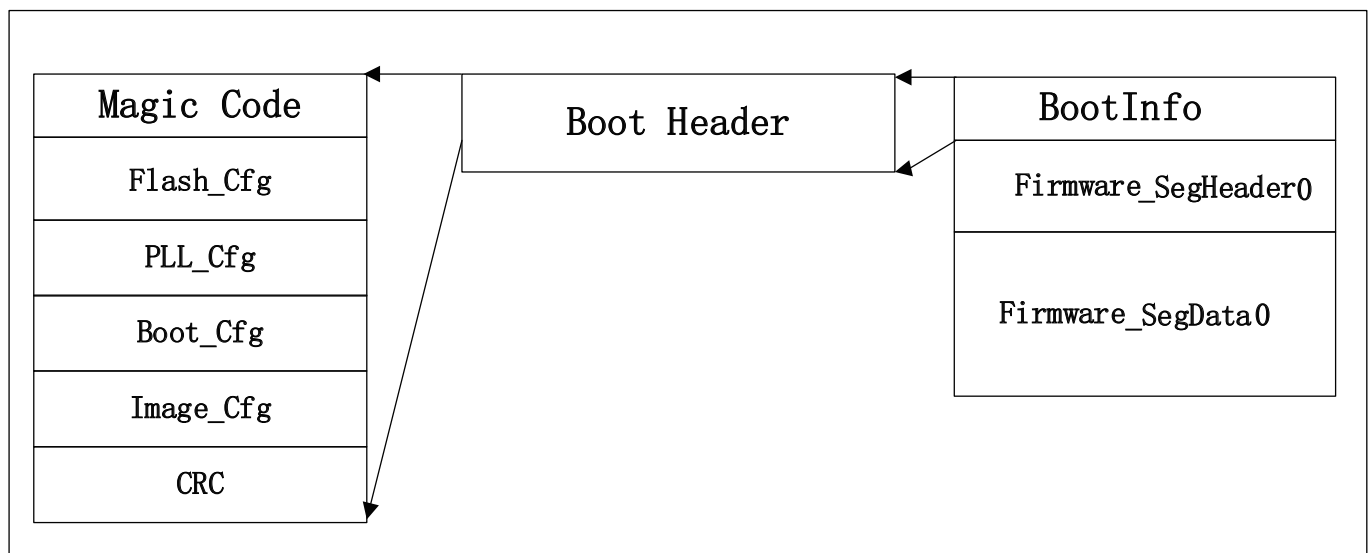


图 2.1: Eflash\_Loader 启动镜像

## 2.1 下载并运行 Eflash\_loader

通过上述 1.5.11 步骤，可以下载 Eflash\_loader 到 RAM 并运行：握手，Get boot info, Load boot header, Load Segment Header, Load Segment Data, Check image, Run Image。

## 2.2 Eflash\_loader 通信协议

主机通过 UART/SDIO 把 Eflash\_loader 下载到 RAM 中并运行后，主机继续通过 UART 接口与 Eflash\_loader 通信。UART 引脚同 1.1，握手过程同 1.2，Eflash\_loader 中配置了高精度的 PLL，可以使用较高的波特率进行握手通信，建议使用的波特率为 115200、1M、2M、2.5M。握手成功后，主机通过以下协议实现 Flash 烧录功能：

## 2.2.1 Chip Erase

表 2.1: Host->BL602

cmdId(1byte)	cksum(1byte)	Len_lsb(1byte)	Len_msb(1byte)
0x3C	Cksum for len	0x00	0x00

表 2.2: BL602->Host

'OK' (2bytes)
0x4F 0x4B

该命令用于擦除整片 Flash。这里参与校验和计算的数据为 cksum 字节后的所有数据 (以下指令与此相同)。校验是可选项，如果不想开启校验，可以将 cksum 设置为 0。

假设有 data\_len 长度 (包含 Len\_lsb 和 Len\_msb) 的数据要参与计算，计算校验和伪代码如下：

```
uint32 sum;
uint8 cksum
while i<data_len
    sum+=data[i]
chsum=sum&0xff
```

## 2.2.2 Flash Erase

表 2.3: Host->BL602

cmdId(1byte)	cksum(1byte)	Len_lsb(1byte)	Len_msb(1byte)	Start _ addr(4bytes)	End_addr(4bytes)
0x30	Cksum datas behind	0x08	0x00		

表 2.4: BL602->Host

'OK' (2bytes)
0x4F 0x4B

该命令用于擦除指定地址空间的 Flash。

Flash 的地址从 0 开始，这里以 1M Flash 为例示意 Flash 地址空间：

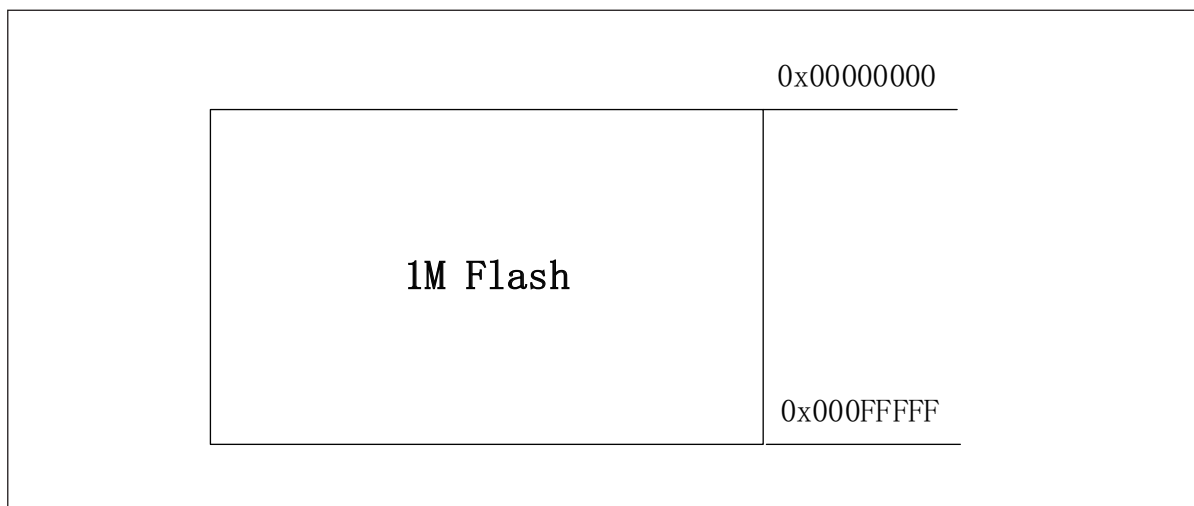


图 2.2: Flash 地址空间示意

### 2.2.3 Flash Program

表 2.5: Host-&gt;BL602

cmdId(1byte)	cksum(1byte)	Len_lsb(1byte)	Len_msb(1byte)	Start - addr(4bytes)	payload(Nbytes)
0x31	Cksum datas behind	(N+4)&0xff	((N+4)>>8)&0xff		

表 2.6: BL602-&gt;Host

'OK' (2bytes)
0x4F 0x4B

写入 Nbytes 数据到 Flash 指定地址空间。由于 Eflash\_loader 中使用的缓冲区的限制，payload 最大为 8Kbytes。

### 2.2.4 Flash Program Check

表 2.7: Host-&gt;BL602

cmdId(1byte)	cksum(1byte)	Len_lsb(1byte)	Len_msb(1byte)
0x3A	Cksum for len	0x00	0x00

表 2.8: BL602-&gt;Host

'OK' (2bytes)
0x4F 0x4B

该命令用于 Flash 烧写数据全部发送完毕后，用来确认 Flash 数据烧写过程中是否出现错误。如果 Flash 烧写全部正确，则返回 OK。否则返回 FL+ 错误代码，此处的错误代码是 BFLB\_EFLASH\_LOADER\_FLASH\_WRITE\_ERROR，详见错误应答帧。

## 2.2.5 Flash Read

表 2.9: Host->BL602

cmdId(1byte)	cksum(1byte)	Len_lsb(1byte)	Len_msb(1byte)	Start - addr(4bytes)	Read - len(4bytes)
0x32	Cksum datas behind	0x08	0x00		

表 2.10: BL602->Host

'OK' (2bytes)	Len_lsb(1byte)	Len_msb(1byte)	payload(Nbytes)
0x4F 0x4B	N&0xff	(N>>8)&0xff	

该命令从 Flash 指定地址空间读取 Nbytes 数据。由于 Eflash\_loader 中使用的缓冲区的限制，Read\_len 最大为 8K。

## 2.2.6 SHA256 Read

表 2.11: Host->BL602

cmdId(1byte)	cksum(1byte)	Len_lsb(1byte)	Len_msb(1byte)	Start - addr(4bytes)	Read - len(4bytes)
0x3D	Cksum datas behind	0x08	0x00		

表 2.12: BL602->Host

'OK' (2bytes)	Len_lsb(1byte)	Len_msb(1byte)	payload(32bytes)
0x4F 0x4B	0x20	0x00	

该命令用于快速校验 Flash 烧写是否正确。主机发送要计算的 Flash 数据的起始地址及长度，BL602 返回该段数据的 SHA256 值。主机也同步计算刚刚烧录文件的 SHA256，然后与返回结果对比，可快速的校验 Flash 是否烧写正确。

## 2.2.7 错误应答帧

以上 BL602Host 的应答帧都是正确情况时的回复，Eflash\_loader 错误返回帧格式如下：

表 2.13: Host->BL602

'FL' (2bytes)	Error_Code_LSB(1byte)	Error_Code_MSB(1byte)
0x46 0x4C		

Error\_Code 列举如下：

```
typedef enum tag_eflash_loader_error_code_t
{
    BFLB_EFLASH_LOADER_SUCCESS=0x00,

    /*flash*/
    BFLB_EFLASH_LOADER_FLASH_INIT_ERROR=0x0001,
    BFLB_EFLASH_LOADER_FLASH_ERASE_PARA_ERROR=0x0002,
    BFLB_EFLASH_LOADER_FLASH_ERASE_ERROR=0x0003,
    BFLB_EFLASH_LOADER_FLASH_WRITE_PARA_ERROR=0x0004,
    BFLB_EFLASH_LOADER_FLASH_WRITE_ADDR_ERROR=0x0005,
    BFLB_EFLASH_LOADER_FLASH_WRITE_ERROR=0x0006,
    BFLB_EFLASH_LOADER_FLASH_BOOT_PARA_ERROR=0x0007,
    BFLB_EFLASH_LOADER_FLASH_SET_PARA_ERROR=0x0008,
    BFLB_EFLASH_LOADER_FLASH_READ_STATUS_REG_ERROR=0x0009,
    BFLB_EFLASH_LOADER_FLASH_WRITE_STATUS_REG_ERROR=0x000A,

    /*cmd*/
    BFLB_EFLASH_LOADER_CMD_ID_ERROR =0x0101,
    BFLB_EFLASH_LOADER_CMD_LEN_ERROR=0x0102,
    BFLB_EFLASH_LOADER_CMD_CRC_ERROR=0x0103,
    BFLB_EFLASH_LOADER_CMD_SEQ_ERROR=0x0104,

    /*image*/
    BFLB_EFLASH_LOADER_IMG_BOOTHEADER_LEN_ERROR=0x0201,
    BFLB_EFLASH_LOADER_IMG_BOOTHEADER_NOT_LOAD_ERROR=0x0202,
    BFLB_EFLASH_LOADER_IMG_BOOTHEADER_MAGIC_ERROR=0x0203,
    BFLB_EFLASH_LOADER_IMG_BOOTHEADER_CRC_ERROR=0x0204,
    BFLB_EFLASH_LOADER_IMG_BOOTHEADER_ENCRYPT_NOTFIT=0x0205,
    BFLB_EFLASH_LOADER_IMG_BOOTHEADER_SIGN_NOTFIT=0x0206,
    BFLB_EFLASH_LOADER_IMG_SEGMENT_CNT_ERROR=0x0207,
```

(下页继续)

(续上页)

```
BFLB_EFLASH_LOADER_IMG_AES_IV_LEN_ERROR=0x0208,  
BFLB_EFLASH_LOADER_IMG_AES_IV_CRC_ERROR=0x0209,  
BFLB_EFLASH_LOADER_IMG_PK_LEN_ERROR=0x020a,  
BFLB_EFLASH_LOADER_IMG_PK_CRC_ERROR=0x020b,  
BFLB_EFLASH_LOADER_IMG_PK_HASH_ERROR=0x020c,  
BFLB_EFLASH_LOADER_IMG_SIGNATURE_LEN_ERROR=0x020d,  
BFLB_EFLASH_LOADER_IMG_SIGNATURE_CRC_ERROR=0x020e,  
BFLB_EFLASH_LOADER_IMG_SECTIONHEADER_LEN_ERROR=0x020f,  
BFLB_EFLASH_LOADER_IMG_SECTIONHEADER_CRC_ERROR=0x0210,  
BFLB_EFLASH_LOADER_IMG_SECTIONHEADER_DST_ERROR=0x0211,  
BFLB_EFLASH_LOADER_IMG_SECTIONDATA_LEN_ERROR=0x0212,  
BFLB_EFLASH_LOADER_IMG_SECTIONDATA_DEC_ERROR=0x0213,  
BFLB_EFLASH_LOADER_IMG_SECTIONDATA_TLEN_ERROR=0x0214,  
BFLB_EFLASH_LOADER_IMG_SECTIONDATA_CRC_ERROR=0x0215,  
BFLB_EFLASH_LOADER_IMG_HALFBAKED_ERROR=0x0216,  
BFLB_EFLASH_LOADER_IMG_HASH_ERROR=0x0217,  
BFLB_EFLASH_LOADER_IMG_SIGN_PARSE_ERROR=0x0218,  
BFLB_EFLASH_LOADER_IMG_SIGN_ERROR=0x0219,  
BFLB_EFLASH_LOADER_IMG_DEC_ERROR=0x021a,  
BFLB_EFLASH_LOADER_IMG_ALL_INVALID_ERROR=0x021b,
```

```
/*IF*/
```

```
BFLB_EFLASH_LOADER_IF_RATE_LEN_ERROR=0x0301,  
BFLB_EFLASH_LOADER_IF_RATE_PARA_ERROR=0x0302,  
BFLB_EFLASH_LOADER_IF_PASSWORDERROR=0x0303,  
BFLB_EFLASH_LOADER_IF_PASSWORDCLOSE=0x0304,
```

```
/*MISC*/
```

```
BFLB_EFLASH_LOADER_PLL_ERROR=0xfffc,  
BFLB_EFLASH_LOADER_INVASION_ERROR=0xfffd,  
BFLB_EFLASH_LOADER_POLLING=0xfffe,  
BFLB_EFLASH_LOADER_FAIL=0xffff,
```

```
}eflash_loader_error_code_t;
```