

# i.MXRT Program Tool Usage and 2nd Bootloader Solution Demo

——Hands ON

CALVIN JI  
(纪成)  
MCU CAS TEAM  
JAN. 2022



PUBLIC USE



SECURE CONNECTIONS  
FOR A SMARTER WORLD

# Agenda

---

- 工具准备
- J-Link调试i.MXRT常见问题经验分享
- J-Link+JFlash+第三方烧写算法批量烧写i.MXRT
- NXP Easy MCU Second Bootloader方案演示

# 工具准备

# 工具准备

1. i.MXRT1060\_EVK
2. Keil IDE v5.xx, 请提前下载安装i.MXRT1062的pack包
3. J-Link调试器v9及以上版本
4. J-Link Commander和J-Flash Tool, 建议安装最新版本J-Link软件工具包
5. All in One Flash Algorithm for RT family代码包(<https://github.com/jicheng0622/All-in-One-Flash-Algorithm-for-RT1050-RT1020>)
6. NXP Easy MCU Boot代码包 ( [https://github.com/jicheng0622/nxp\\_easy\\_mcuboot](https://github.com/jicheng0622/nxp_easy_mcuboot) )
7. Blhost工具([https://www.nxp.com/webapp/Download?colCode=blhost\\_2.6.2&appType=license&location=null](https://www.nxp.com/webapp/Download?colCode=blhost_2.6.2&appType=license&location=null))



# J-LINK调试I.MXRT常见问题经验分享




# RT\_EVK板载仿真器更换J-Link固件

目前现有的RT\_EVK板载仿真器有两种，一种是基于MK20的传统OpenSDA，另一种是基于LPC4322 DAP-LINK，前者只存在于RT1052\_EVK和RT1021\_EVK，后续的RT系列EVK都会使用后者带高速USB接口的DAP-LINK，而且这两者更新J-Link固件的方式是不同的。

- 1. 针对传统OpenSDA的板载仿真器更新J-Link固件比较简单，直接到Segger官网下载最新的J-Link OpenSDAv2.1固件，然后在板子上电前按住复位按键再插入USB Cable就可以在电脑端枚举一个名为MAINTANCE的U盘，将J-Link固件直接拖进去即可完成升级；

<https://www.segger.com/products/debug-probes/j-link/models/other-j-links/opensda-sda-v2/>



		Products ▾ Downloads ▾ Purchase ▾ Support ▾ About Us ▾
J-Link Debug Probes > OpenSDA / - SDA V2		Technology ▾
MIMXRT1050-EVK-Hyperflash		2.1
MIMXRT1050-EVK-QSPI		2.1
MIMXRT1020-EVK		2.1



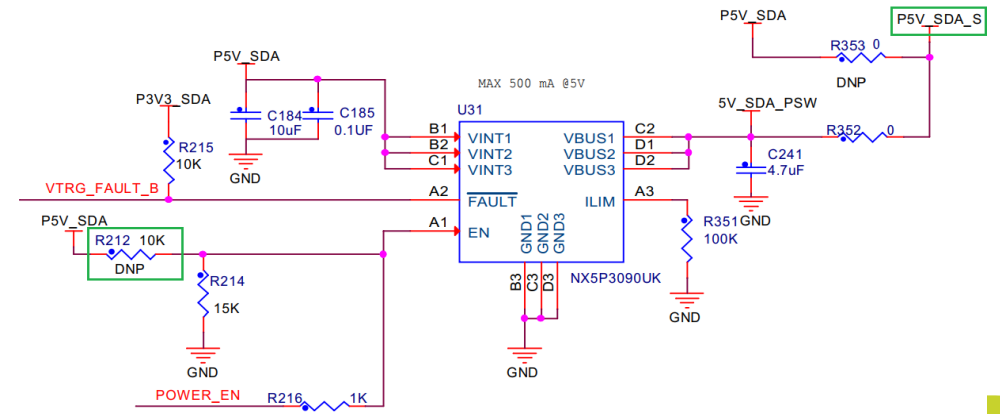
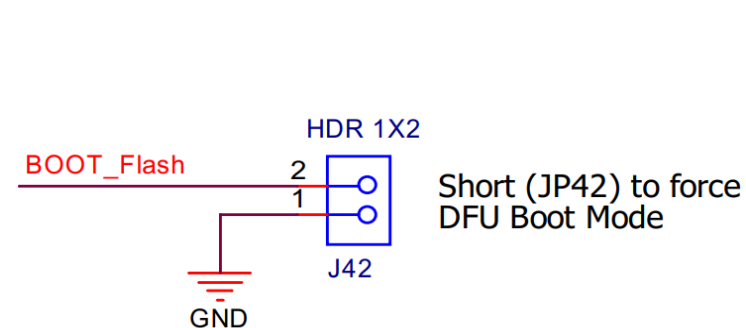
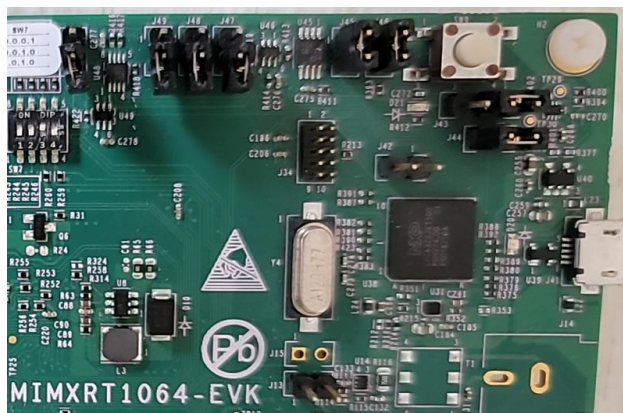
# RT\_EVK板载仿真器更换J-Link固件

2. 针对DAP-LINK板载仿真器更新J-Link固件稍麻烦些，虽然按住复位按键再插入USB也可以在电脑端枚举成MAINTANCE的U盘，但是往里拖入J-Link固件是无效的（说明它是无法通过DAP-LINK的Bootloader引导的），需要找到板子的DFU Boot跳线短接然后重新上电使LPC4322进入DFU模式，然后通过下面连接下载并安装LPCScript tool，找到C:\nxp\LPCScript\_2.1.2\_57\scripts路径下的program\_JLINK.cmd文件双击即可完成Jlink固件的更新，完成后记得断开DFU Boot跳线。

\* 注：如果想恢复DAP-LINK则按照同样方法双击上述目录下的program\_LPC4322\_DAPlink\_RT.cmd即可。

<https://www.nxp.com/design/microcontrollers-developer-resources/lpcscript-v2-1-2:LPCSCRIPT>

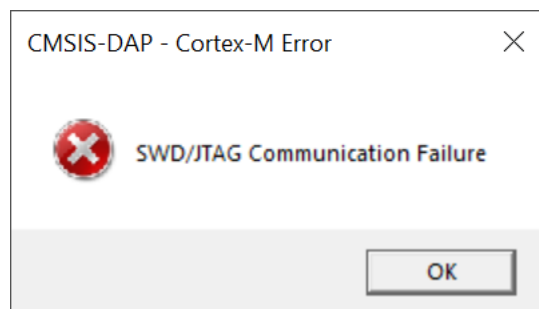
通过上述方法更新J-Link后，PC端可以枚举J-Link设备但是无法连接目标芯片内核，因为该固件没有使能右下图的POWER\_EN信号导致P5V\_SDA\_S没有输出，MCU电源不工作，此时需要手动焊接下图R212 10k电阻使能该电源信号或者直接短路R353即bypass NX5P390电压控制芯片。



# RT烧录失败或烧错固件再次连接不成功问题

使用板载DAP-LINK仿真器调试RT时由于下载环境不稳定，下错程序或者下载中途断电复位等问题导致下次再连接内核时报错如左下图所示，只能更改RT BootMode为SerialDownload模式重新下载一个正常的程序才能恢复。该问题在更换为J-Link固件或者在客户样机上使用J-Link调试开发的时候会改善很多如右下图所示大多数情况使用J-Link可以重新Halt CPU，所以一般建议使用J-Link开发RT系列。

\* 注：如果客户使用U-Link开发可能会遇到同样的问题，因为U-Link同样使用CMSIS-DAP接口。



```
Build Output
* JLink Info: InitTarget() start
* JLink Info: InitTarget()
* JLink Info: _TargetHalt: CPU halted
* JLink Info: InitTarget() end
* JLink Info: Found SW-DP with ID 0x0BD11477
* JLink Info: Failed to power up DAP
* JLink Info: InitTarget() start
* JLink Info: InitTarget()
* JLink Info: _TargetHalt: CPU halted
* JLink Info: InitTarget() end
* JLink Info: Found SW-DP with ID 0x0BD11477
* JLink Info: DPIDR: 0x0BD11477
* JLink Info: Scanning AP map to find all available APs
* JLink Info: AP[1]: Stopped AP scan as end of AP map has been reached
* JLink Info: AP[0]: AHB-AP (IDR: 0x04770041)
* JLink Info: Iterating through AP map to find AHB-AP to use
* JLink Info: AP[0]: Core found
* JLink Info: AP[0]: AHB-AP ROM base: 0xE00FD000
* JLink Info: CPUID register: 0x411FC271. Implementer code: 0x41 (ARM)
* JLink Info: Found Cortex-M7 r1p1, Little endian.
```



# J-Link Commander调试RT技巧——探针内部Register和Memory空间

在分析和定位RT异常复位或者启动失败的原因时，RT内部有两个寄存器很重要如下两图，即复位寄存器SRC\_SRSR和BootMode寄存器SRC\_SBMR1，我们可以通过J-Link Commander注入探针命令读取这两个寄存器内容作为辅助分析手段，具体方法如下：

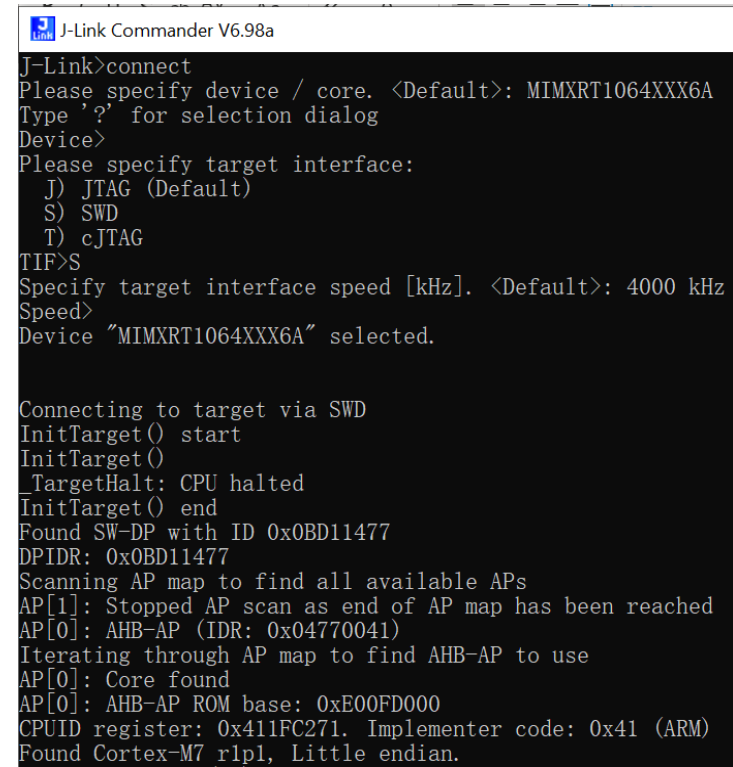
## 20.8.3 SRC Reset Status Register (SRC\_SRSR)

The SRSR is a write to one clear register which records the source of the reset events for the chip. The SRC reset status register will capture all the reset sources that have occurred. This register is reset on ipp\_reset\_b. This is a read-write register.

## 20.8.2 SRC Boot Mode Register 1 (SRC\_SBMR1)

The Boot Mode register (SBMR) contains bits that reflect the status of Boot Mode Pins of the chip. The reset value is configuration dependent (depending on boot/fuses/IO pads).

1. 在出现异常复位情况时（MCU已经正常跑起来了），我们连上J-Link并打开J-Link Commander，先敲入connect，然后按照提示步骤连接到内核，如右图所示；



```
J-Link Commander V6.98a
J-Link>connect
Please specify device / core. <Default>: MIMXRT1064XXX6A
Type '?' for selection dialog
Device>
Please specify target interface:
J) JTAG (Default)
S) SWD
T) cJTAG
TIF>S
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
Device "MIMXRT1064XXX6A" selected.

Connecting to target via SWD
InitTarget() start
InitTarget()
TargetHalt: CPU halted
InitTarget() end
Found SW-DP with ID 0x0BD11477
DPIDR: 0x0BD11477
Scanning AP map to find all available APs
AP[1]: Stopped AP scan as end of AP map has been reached
AP[0]: AHB-AP (IDR: 0x04770041)
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xE00FD000
CPUID register: 0x411FC271. Implementer code: 0x41 (ARM)
Found Cortex-M7 r1p1, Little endian.
```

# J-Link Commander调试RT技巧——探针内部Register和Memory空间

2. 查看RT参考手册（以RT1062为例）确定SRC\_SRSR寄存器地址为0x400F8008，然后继续在J-Link Commander窗口输入“mem32 0x400F8008 1”，即可返回该寄存器内容如下图，默认RT第一次运行时该寄存器为0x1，由该寄存器描述可知上一次触发MCU复位的原因因为POR\_B有效，即外部有信号拉低POR\_B或者MCU上电复位；

## 20.8 SRC Memory Map/Register Definition

SRC memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
400F_8000	SRC Control Register (SRC_SCR)	32	R/W	A048_0520h	<a href="#">20.8.1/1337</a>
400F_8004	SRC Boot Mode Register 1 (SRC_SBMR1)	32	R	0000_0000h	<a href="#">20.8.2/1339</a>
400F_8008	SRC Reset Status Register (SRC_SRSR)	32	R/W	0000_0001h	<a href="#">20.8.3/1340</a>
400F_801C	SRC Boot Mode Register 2 (SRC_SBMR2)	32	R	0000_0000h	<a href="#">20.8.4/1343</a>

SRC\_SRSR field descriptions (continued)

Field	Description
3 ipp_user_reset_b	Indicates whether the reset was the result of the ipp_user_reset_b qualified reset. 0 Reset is not a result of the ipp_user_reset_b qualified as COLD reset event. 1 Reset is a result of the ipp_user_reset_b qualified as COLD reset event.
2 csu_reset_b	Indicates whether the reset was the result of the csu_reset_b input. 0 Reset is not a result of the csu_reset_b event. 1 Reset is a result of the csu_reset_b event.
1 lockup_sysresetreq	Indicates a reset has been caused by CPU lockup or software setting of SYSRESETREQ bit in Application Interrupt and Reset Control Register of the ARM core. SW needs to set some GPR bit before writing SYSRESETREQ bit and use the GPR bit to distinguish if the reset is caused by SYSRESETREQ or CPU lockup. 0 Reset is not a result of the mentioned case. 1 Reset is a result of the mentioned case.
0 ipp_reset_b	Indicates whether reset was the result of ipp_reset_b pin (Power-up sequence) 0 Reset is not a result of ipp_reset_b pin. 1 Reset is a result of ipp_reset_b pin.

```
J-Link Commander V6.98a
I-Cache L1: 32 KB, 512 Sets, 32 Bytes/Line, 2-Way
D-Cache L1: 32 KB, 256 Sets, 32 Bytes/Line, 4-Way
Cortex-M7 identified.
J-Link>mem32 0x400F8008 1
400F8008 = 00000001
J-Link>
```



# J-Link Commander调试RT技巧——探针内部Register和Memory空间

3. 我们可以模拟一次软件复位来观察该寄存器的变化，先输入“w4 0x400F8008 1”即写1清零SRC\_SRSR对应的位，然后再次输入“mem32 0x400F8008 1”以确认寄存器位是否正确清0，接下来输入“r”即复位命令，默认该复位命令为软件复位命令（即写ARM内核寄存器位SYSRESETREQ触发CPU内核软复位），最后输入“mem32 0x400F8008 1”，可以看到SRC\_SRSR寄存器变为2，对照手册即上一次复位类型为软复位；

```
J-Link Commander V6.98a
I-Cache L1: 32 KB, 512 Sets, 32 Bytes/Line, 2-Way
D-Cache L1: 32 KB, 256 Sets, 32 Bytes/Line, 4-Way
Cortex-M7 identified.
J-Link>mem32 0x400F8008 1
400F8008 = 00000001
J-Link>w4 0x400F8008 1
Writing 00000001 -> 400F8008
J-Link>mem32 0x400F8008 1
400F8008 = 00000000
J-Link>r
Reset delay: 0 ms
Reset type NORMAL: Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
Reset: Halt core after reset via DEMCR.VC_CORERESSET.
Reset: Reset device via AIRCR.SYSRESETREQ.
J-Link>mem32 0x400F8008 1
400F8008 = 00000002
J-Link>
```

# J-Link Commander调试RT技巧——探针内部Register和Memory空间

4. 这一步我们模拟一次启动失败的情况，将RT1062\_EVK的SW7拨码由原来的“0010”（QSPI Flash启动）改为“1010”（SD卡启动），然后我们手动按复位按键，此时MCU处于启动失败的条件（因为我们没有插入SD卡）。我们需要重新打开J-Link Commander，继续输入“connect”一系列操作连接到内核，最后输入“mem32 0x400F8004 1”读取SRC\_SBMR1寄存器内容为0x40，即BT\_CFG[6](GPIO\_B0\_10)=1；

\* 注：客户打EMC的时候或者板子干扰比较大的时候，很容易出现MCU的假象死机（即MCU电源都正常且POR\_B也为高，这种情况实际是MCU复位后启动失败死在ROM里），通过这种方式可以有效判定外部哪个BT\_CFG管脚的干扰造成的。此外，mem32命令也可以通过AHB读的方式访问RT片上RAM空间，片外SDRAM和片外SPI NorFlash。

## 20.8.2 SRC Boot Mode Register 1 (SRC\_SBMR1)

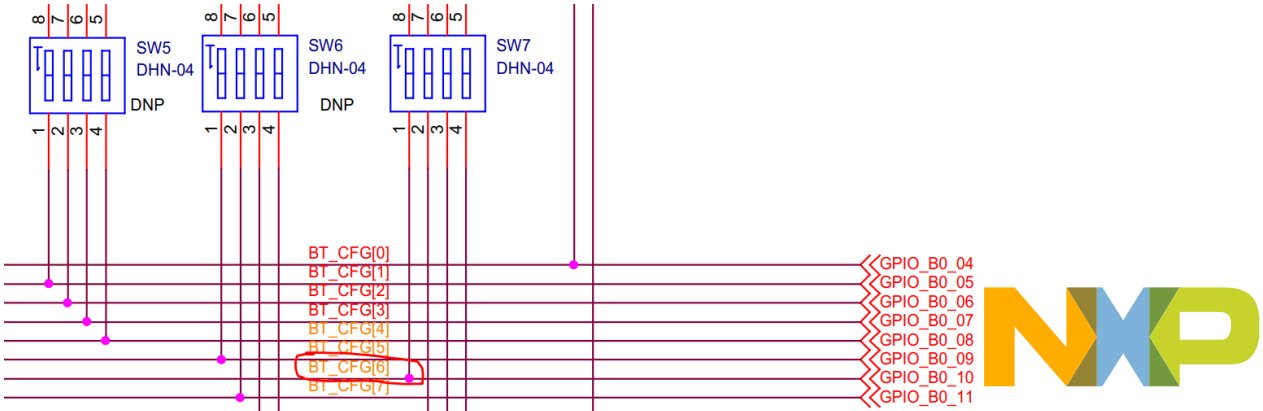
The Boot Mode register (SBMR) contains bits that reflect the status of Boot Mode Pins of the chip. The reset value is configuration dependent (depending on boot/fuses/IO pads).

If SRC\_GPR10[28] bit is set, this bit instructs the ROM code to use the SRC\_GPR9 registe as if it is SBMR1. This allows software to override the fuse bits and boot from an alternate boot source.

Address: 400F\_8000h base + 4h offset = 400F\_8004h

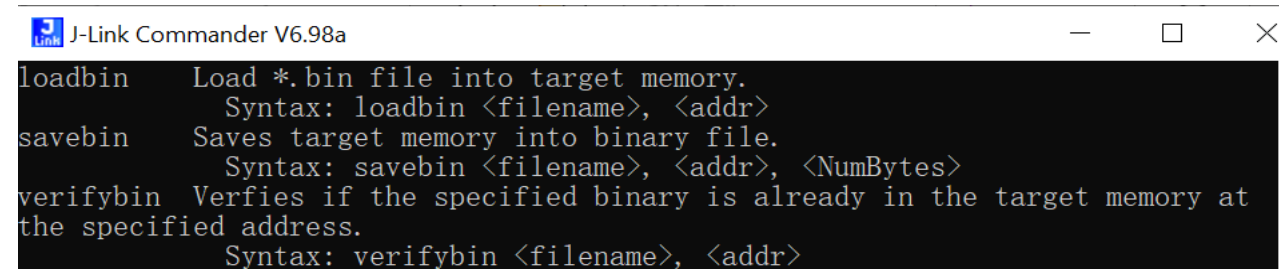
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BOOT_CFG4[7:0]								BOOT_CFG3[7:0]								BOOT_CFG2[7:0]								BOOT_CFG1[7:0]							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

```
J-Link Commander V6.98a
I-Cache L1: 32 KB, 512 Sets, 32 Bytes/Line, 2-Way
D-Cache L1: 32 KB, 256 Sets, 32 Bytes/Line, 4-Way
Cortex-M7 identified.
J-Link>mem32 0x400F8004 1
400F8004 = 00000040
J-Link>
```



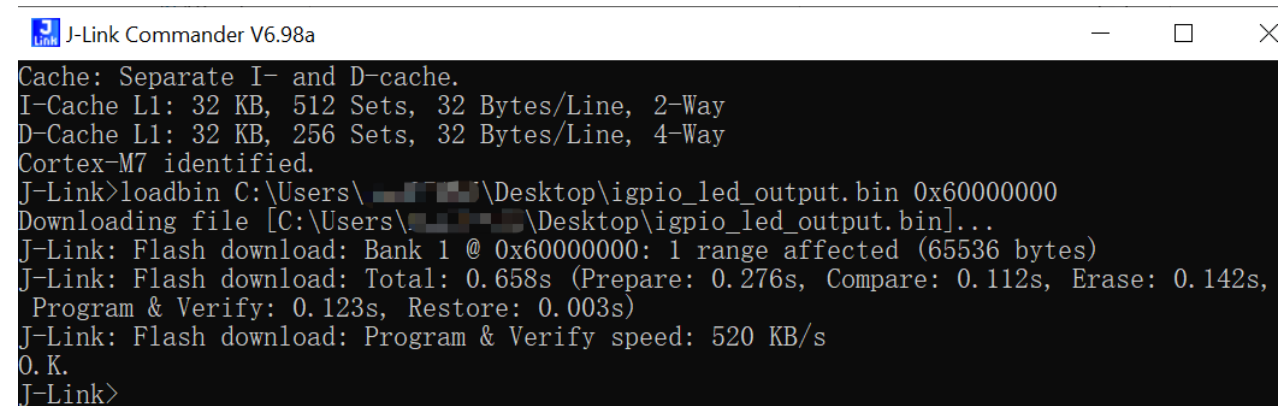
# J-Link Commander调试RT技巧——回读校验指定地址范围Flash内容

当用户需要检查Flash内容是否被意外篡改或者要回读Flash内容以确认当前运行的软件版本时，J-Link Commander也提供了这样的功能，即“savebin”和“verifybin”命令如下图，具体用法如下：



```
J-Link Commander V6.98a
loadbin  Load *.bin file into target memory.
          Syntax: loadbin <filename>, <addr>
savebin   Saves target memory into binary file.
          Syntax: savebin <filename>, <addr>, <NumBytes>
verifybin Verifies if the specified binary is already in the target memory at
the specified address.
          Syntax: verifybin <filename>, <addr>
```

1. 首先打开一个RT1062 SDK的一个例程（以gpio\led\_output例程为例）编译生成一个bin文件，将该bin文件拷贝到桌面作为待下载的目标image文件，然后打开J-Link Commander并连接到内核，然后输入“loadbin 桌面路径\igpio\_led\_output.bin 0x60000000”，即把目标image文件下载到RT Flash空间，然后按复位按键即可观察程序已经正常运行起来；



```
J-Link Commander V6.98a
Cache: Separate I- and D-cache.
I-Cache L1: 32 KB, 512 Sets, 32 Bytes/Line, 2-Way
D-Cache L1: 32 KB, 256 Sets, 32 Bytes/Line, 4-Way
Cortex-M7 identified.
J-Link>loadbin C:\Users\...\Desktop\igpio_led_output.bin 0x60000000
Downloading file [C:\Users\...\Desktop\igpio_led_output.bin]...
J-Link: Flash download: Bank 1 @ 0x60000000: 1 range affected (65536 bytes)
J-Link: Flash download: Total: 0.658s (Prepare: 0.276s, Compare: 0.112s, Erase: 0.142s,
Program & Verify: 0.123s, Restore: 0.003s)
J-Link: Flash download: Program & Verify speed: 520 KB/s
O.K.
J-Link>
```



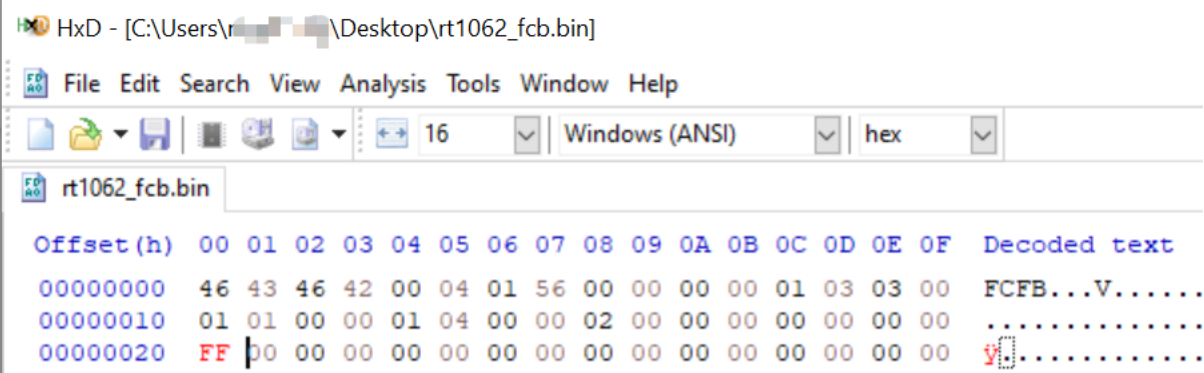


# J-Link Commander调试RT技巧——回读校验指定地址范围Flash内容

2. 接下来我们先回读Flash内容，为简单测试，这里我们只读0x60000000开头的512个字节回来（前512个字节为SPI Flash的配置参数即FCB区域），另外注意savebin的回读字节数参数为hex十六进制格式，在J-Link Commander窗口输入“savebin 桌面路径\rt1062\_fcb.bin,0x60000000,200”；

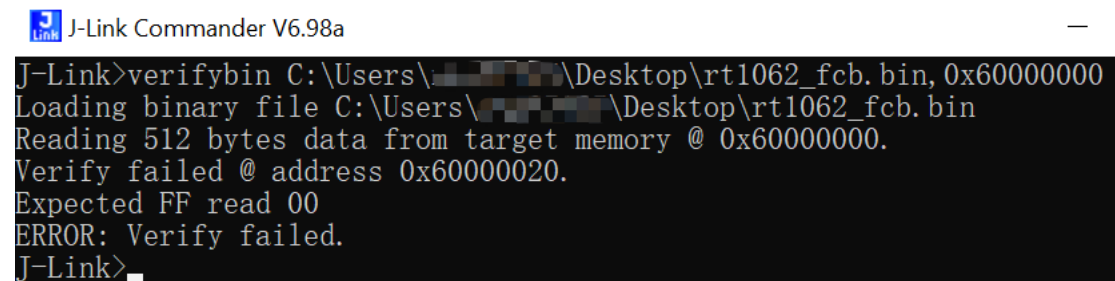
```
J-Link Commander V6.98a
J-Link>savebin C:\Users\...\Desktop\rt1062_fcb.bin,0x60000000,0x200
Opening binary file for writing... [C:\Users\...\Desktop\rt1062_fcb.bin]
Reading 512 bytes from addr 0x60000000 into file...O.K.
J-Link>
```

3. 使用二进制工具打开回读回来的文件rt1062\_fcb.bin，模拟篡改一个地址的内容，这里我们把0x20的地址由原来的00改成FF，然后保存；



# J-Link Commander调试RT技巧——回读校验指定地址范围Flash内容

4. 在J-Link Commander窗口输入“verifybin 桌面路径\rt1062\_fcb.bin,0x60000000”，运行结果如下图，可以校验出源文件与芯片实际物理地址的内容比对后第一个异常的地址并详细说明了该地址内容是如何变化的；



```
J-Link Commander V6.98a
J-Link>verifybin C:\Users\...\Desktop\rt1062_fcb.bin,0x60000000
Loading binary file C:\Users\...\Desktop\rt1062_fcb.bin
Reading 512 bytes data from target memory @ 0x60000000.
Verify failed @ address 0x60000020.
Expected FF read 00
ERROR: Verify failed.
J-Link>_
```

# J-LINK+JFLASH+第三方烧写算法 批量烧写I.MXRT



# J-Link+J-Flash+第三方算法批量烧写RT SPI Flash——简介

J-Flash是J-Link配套的一个非常强大的在线UI烧录软件，其自带了丰富且全面的各家MCU Flash烧写算法，但是由于RT外置SPI Flash，而外置Flash带来的问题就是不同厂家甚至同一厂家不同系列不同版本Flash的兼容性问题，目前J-Flash自带的RT外部SPI Flash算法仍然解决不了该问题（主要是各家Flash厂的JEDEC216版本SPI Flash四线模式使能的不同），庆幸的是J-Flash提供了第三方Flash烧写算法的接口标准（见下面链接），用户可以自定义算法适配自己的Flash参数，然后供J-Flash烧写时调用，替换自带的烧写算法：

[https://wiki.segger.com/Open\\_Flashloader](https://wiki.segger.com/Open_Flashloader)

## Open Flashloader

### Introduction

As the number of devices being available is steadily growing and sometimes in an early stage of the MCU development only a few samples/boards are available that may not be provided to third parties (e.g. SEGGER) to add support for a new device. Also the existence of the device may have confidential status, so it might not be mentioned as being supported in public releases yet. Therefore it might be desirable to be able to add support for new devices on your own, without depending on SEGGER and a new release of the [J-Link Software an Documentation Pack](#) being available.

The J-Link DLL allows customers to add support for new devices on their own. It is also possible to edit / extend existing device support by for example adding new flash banks (e.g. to add support for internal EEPROM programming or SPIFI programming). This article explains how new devices can be added to the DLL and how existing ones can be edited / extended.

### Included files

Filename	Content
FlashDev.c	Flash device description
FlashOS.h	Function prototypes, definitions and structures
FlashPrg.c	Flash algorithm itself (e.g. ProgramPage(), EraseSector())
main.c	Flash algorithm debug code (used by debug configuration, only)
Cortex_M_Startup.s	Cortex-M startup code (used by debug configuration, only)
ARM_Startup.s	Cortex-A/R startup code (used by debug configuration, only)
MemoryMap.xml	Memory map of the ST STM32F205RC
Placement_debug.xml	Debug configuration section placement file.
Placement_release.xml	Release configuration section placement file.
thumb crt0.s	Initialization file for Cortex-M (used by debug configuration, only)
crt0.s	Initialization file for Cortex-A/R(used by debug configuration, only)



# J-Link+J-Flash+第三方算法批量烧写RT SPI Flash——Hands-On

下面以RT1062\_EVK板子为例，详细展示下如何添加第三方Flash烧写算法到J-Flash的芯片支持库里面：

1. 首先到如下地址下载一个针对RT1020和RT1050（RT1060与1050是可以共用一个算法的）的第三方Flash烧写算法（由NXP CAS开发维护的，当然也可以参考同样的步骤添加其他第三方算法）；

<https://github.com/jicheng0622/All-in-One-Flash-Algorithm-for-RT1050-RT1020/tree/master/J-Flash>

2. 解压该项目后，下图所示的文件路径下的elf文件为我已经编译好的Flash烧写算法文件，当然如果用户想要修改该算法重新生成的话，可以安装最新版Segger Embedded Studio IDE，然后直接导入下图上一级路径下的源工程文件重新编译即可；

« J-Flash » FlashIMXRT1050RT1020_EVK_FlexSPI_All_in_One_Jflash_FastAlgo » JLinkDeviceXMLEntry			
Name	Date modified	Type	Size
FlashIMXRT1020_SPINor.elf	2021/1/30 22:56	ELF File	1,646 KB
FlashIMXRT1050_SPINor.elf	2021/1/30 22:55	ELF File	1,883 KB
JLinkDevices.xml	2021/1/30 22:57	XML Document	4 KB

# J-Link+J-Flash+第三方算法批量烧写RT SPI Flash——Hands-On

3. 将上一步文件路径下的FlashIMXRT1050\_SPINor.elf文件拷贝到J-Link软件安装路径下，具体如下图所示（C:\Program Files (x86)\SEGGER\JLink\Devices\NXP\iMXRT105x）：

OSDisk (C:) > Program Files (x86) > SEGGER > JLink > Devices > NXP > iMXRT105x

Name	Date modified	Type	Size
FlashIMXRT1050_SPINor.elf	2021/5/27 0:05	ELF File	1,886 KB
NXP_iMXRT105x_HyperFlash.elf	2021/3/2 22:22	ELF File	53 KB

4. 文件准备完毕，下面需要将该Flash算法文件与芯片型号绑定，即告知J-Flash工具芯片与烧录文件的映射关系，这是通过J-Link安装根目录下的JLinkDevices.xml文件来定义的，打开该文件并添加如下信息；

```
<!-- -->
<!-- NXP (iMXRT1062 QSPI Flash, 3rd Algorithm) -->
<!-- -->
<Device>
  <ChipInfo Vendor="NXP" Name="MIMXRT1062_3rdAlgo" WorkRAMAddr="0x20000000" WorkRAMSize="0x00080000" Core="JLINK_CORE_CORTEX_M7" />
  <FlashBankInfo Name="QSPI Flash" BaseAddr="0x60000000" MaxSize="0x04000000" Loader="Devices/NXP/iMXRT105x/FlashIMXRT1050_SPINor.elf" LoaderType="FLASH_ALGO_TYPE_OPEN" />
</Device>

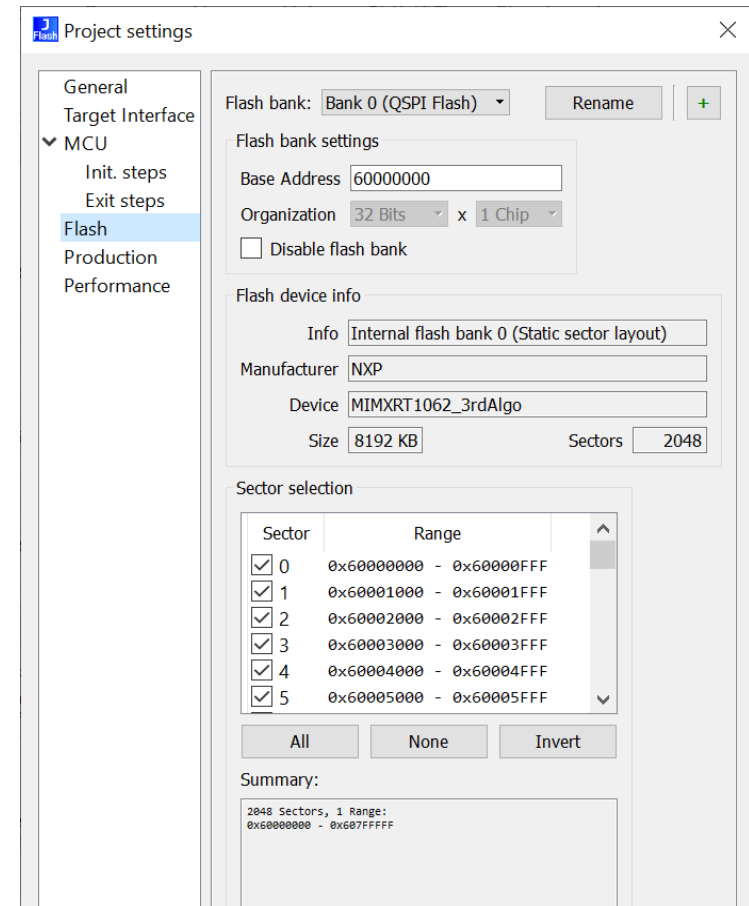
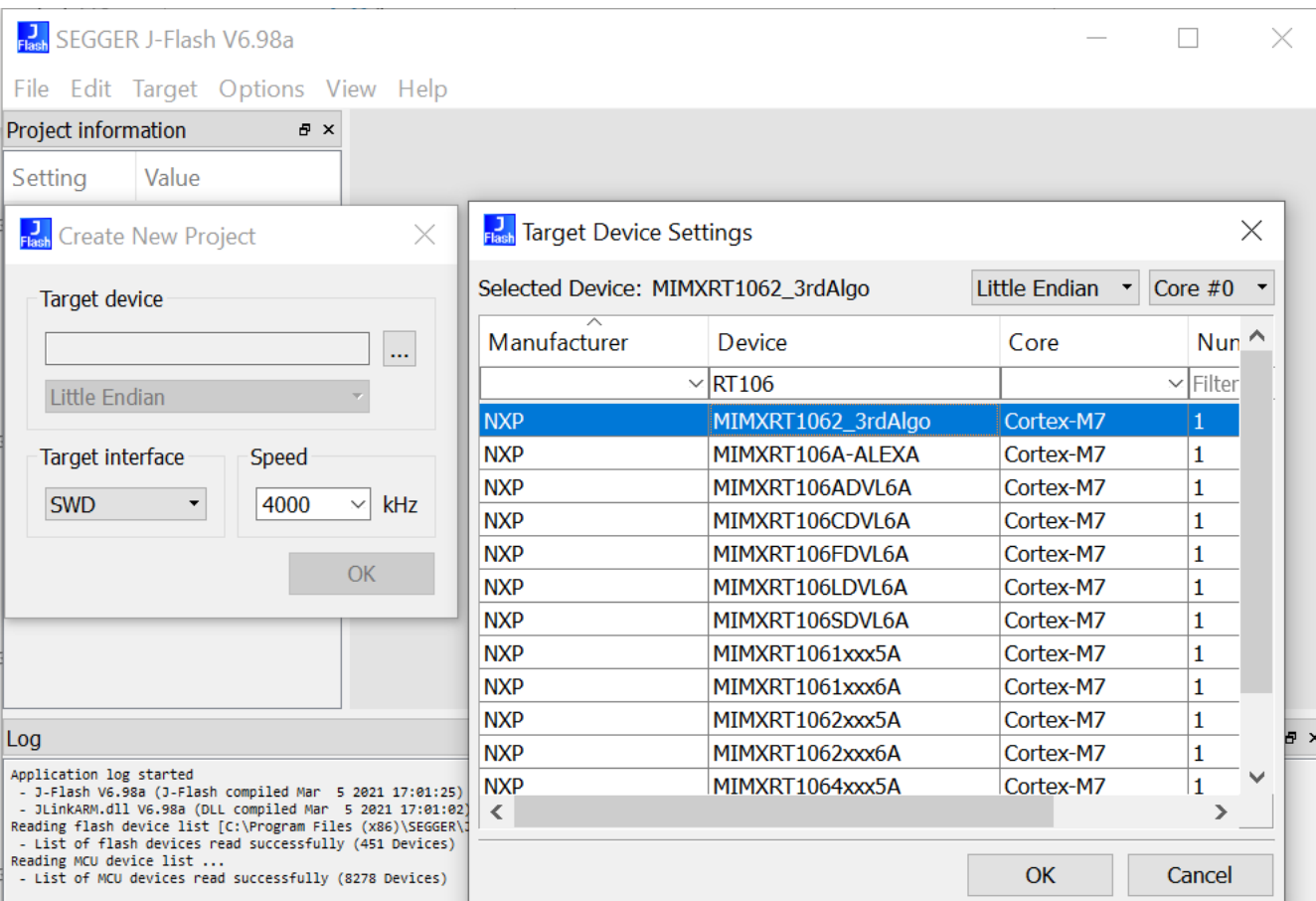
337 <Device>
338   <ChipInfo Vendor="NXP" Name="MIMXRT1052xxxxB" WorkRAMAddr="0x20000000" WorkRAMSize="0x00080000" Core="JLINK_CORE_CORTEX_M7" Aliases="MIMXRT1052xxx5B; MIMXRT1052CVL5B; MIMXRT1052
339   <FlashBankInfo Name="HyperFlash" BaseAddr="0x60000000" MaxSize="0x04000000" Loader="Devices/NXP/iMXRT105x/NXP_iMXRT105x_HyperFlash.elf" LoaderType="FLASH_ALGO_TYPE_OPEN" />
340 </Device>
341 <!-- -->
342 <!-- NXP (iMXRT1062 QSPI Flash, 3rd Algorithm) -->
343 <!-- -->
344 <Device>
345   <ChipInfo Vendor="NXP" Name="MIMXRT1062_3rdAlgo" WorkRAMAddr="0x20000000" WorkRAMSize="0x00080000" Core="JLINK_CORE_CORTEX_M7" />
346   <FlashBankInfo Name="QSPI Flash" BaseAddr="0x60000000" MaxSize="0x04000000" Loader="Devices/NXP/iMXRT105x/FlashIMXRT1050_SPINor.elf" LoaderType="FLASH_ALGO_TYPE_OPEN" />
347 </Device>
```



# J-Link+J-Flash+第三方算法批量烧写RT SPI Flash——Hands-On

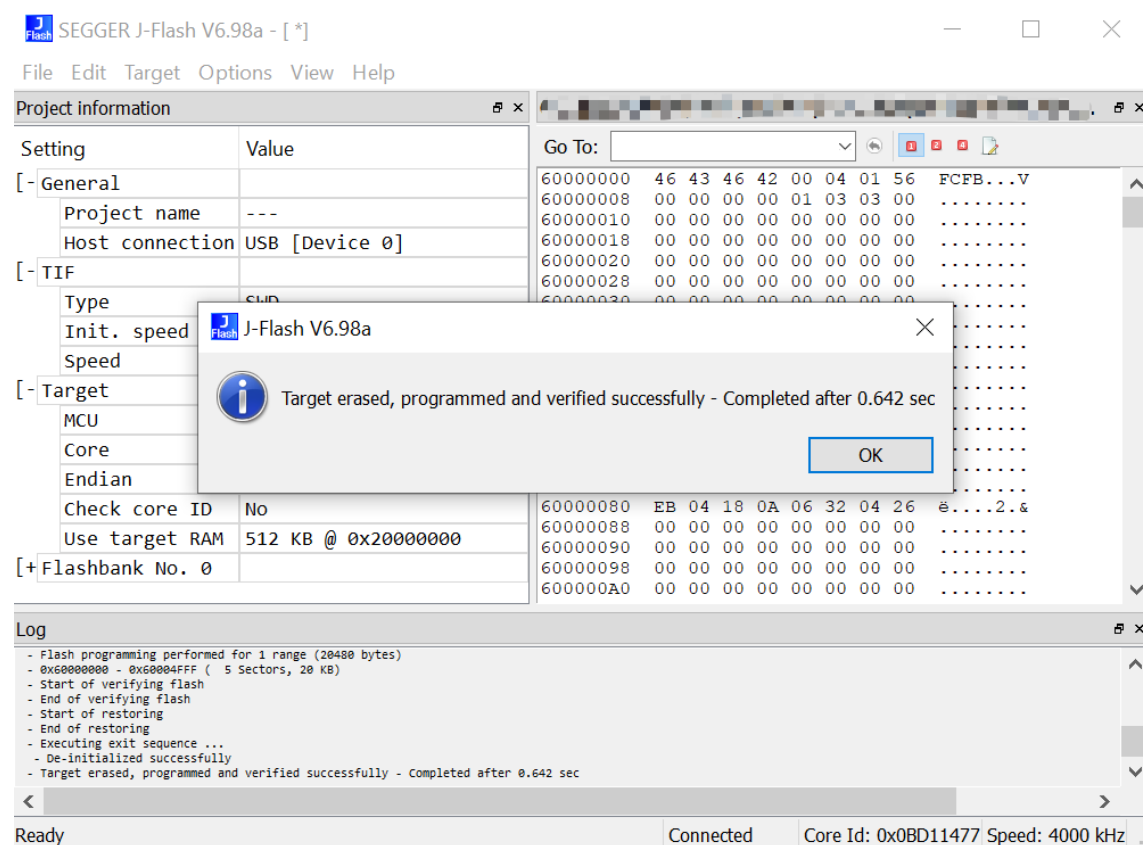
5. 打开J-Flash软件，新建project，在Target Device List里就可以找到我们新添加的第三方算法了，点击OK进去之后可以通过菜单Options->Project Settings->Flash查看新算法的一些配置信息，如下图：

\* 注：不推荐使用J-Flash Lite，功能受限，用不了Segger的快速算法。



# J-Link+J-Flash+第三方算法批量烧写RT SPI Flash——Hands-On

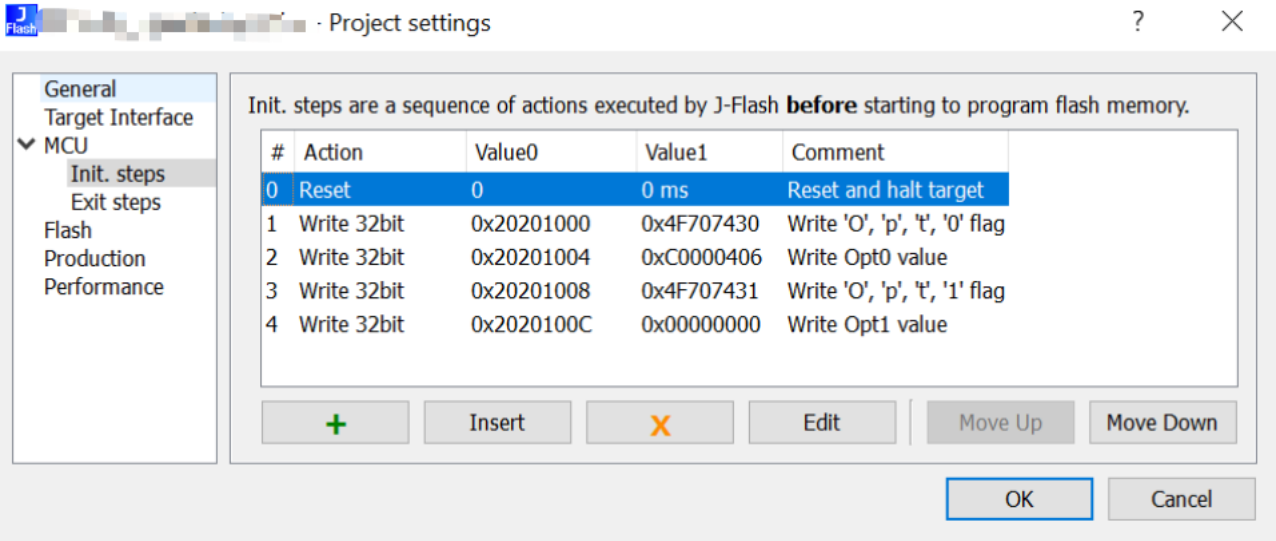
6. 打开RT1062 SDK的一个例程（这里以\gpio\led\_output为例），选择flexspi\_nor\_debug模式，编译生成hex或者srec文件，然后将其拖入到J-Flash窗口，点击F7或者菜单Target->Product Programming即可完成对板载SPI NorFlash的烧写；



# J-Link+J-Flash+第三方算法批量烧写RT SPI Flash——Hands-On

7. 如果客户样机板载QSPI Flash是非NXP官方EVK板载的型号，尤其是只支持JEDEC216老版本的QSPI Flash，则该算法默认参数可能会造成烧写失败或者烧写成功无法运行，这个是由于Flash内部的QE即4线使能位没有成功置1造成的，此时可以通过J-Flash的菜单Options->Project Settings->MCU init.steps添加如下参数实现同一算法不用修改即可不同厂家不同型号的QSPI Flash的适配（仅限本示例下载的第三方Flash烧写算法，内部添加了该处理）。

\* 注：详细流程强烈建议参考本示例第三方算法All-in-One-Flash-Algorithm-for-RT1050-RT1020J-Flash路径下的readme。



## 常用QSPI Norflash配置字示例

Flash型号	Opt0配置字
GD25Q32/Q64	0xC0000406
W25Q80/Q16/Q32/Q64	0xC0000205
MX25L6433F	0xC0000105
IS25LP032/064 (JEDEC216A)	0xC0000107(选配)
S26KS512SDPBHI02 (HyperFlash 1.8v, EVK_RT1050默认Flash)	0xC0233009



# NXP Easy MCU Second Bootloader 方案演示

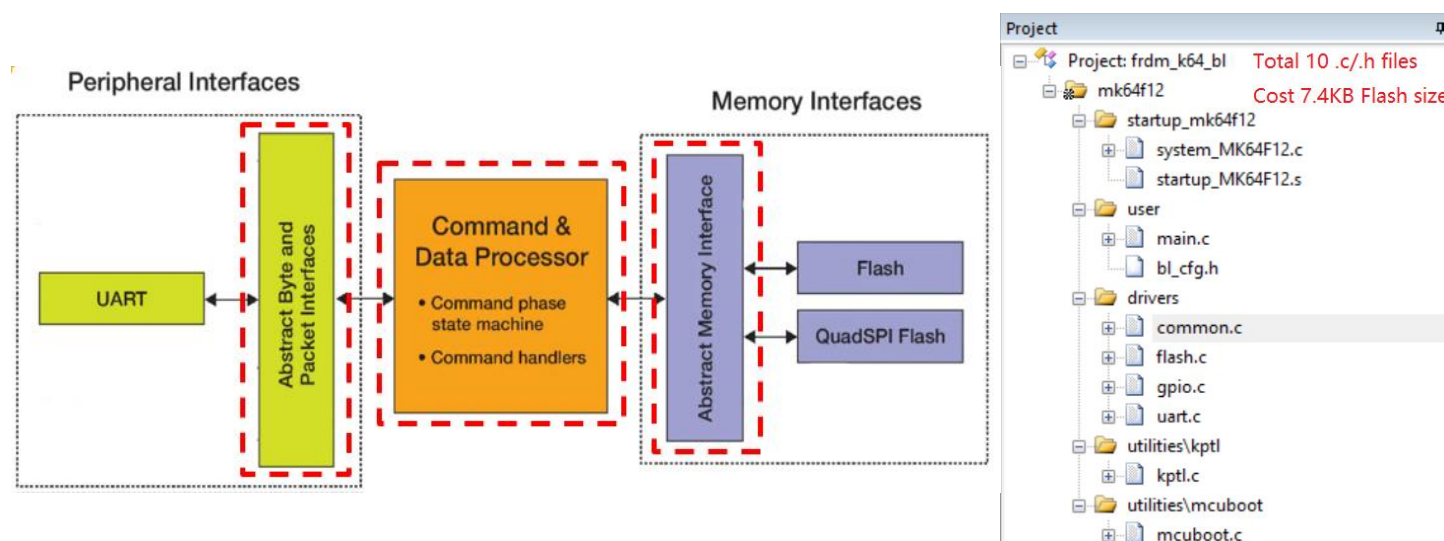




# NXP Easy MCU Second Bootloader——简介

二级Bootloader对需要固件升级的MCU用户来说是必不可少的，各家半导体厂商也会提供对应的解决方案，但是却较少用户会选择直接拿来使用它，而是把官方的方案作为基本框架然后嵌入自己的通信协议和通信接口，这就需要官方提供的二级Bootloader方案尽量简洁清晰易懂，且由于MCU本身存储空间限制，同时Bootloader又不是执行主要应用业务，所以Bootloader本身所占空间也要越小越好，基于此需求，NXP CAS team和SE team在官方MCU-BOOT基础上合作开发了一块简易的二级Bootloader——NXP Easy MCU Boot，其特点如下：

- 代码框架简洁清晰，所有功能仅需10个文件以内（RT系列稍多），易于用户移植和自定义开发；
- 占用空间小，对于LPC802只占用3KB；
- 通信协议框架兼容MCU-Boot，可以复用MCU-Boot的上位机工具Flash Tool GUI和Blhost命令行进行固件升级，以及文档；
- 支持Kinetis，LPC和i.MXRT系列；



项目地址: [https://github.com/jicheng0622/nxp\\_easy\\_mcuboot](https://github.com/jicheng0622/nxp_easy_mcuboot)



# NXP Easy MCU Second Bootloader——支持型号和文件目录结构

支持PART列表

PART	APP起始地址
K64	0x8000
KE02	0x8000
KE15	0x8000
LPC802	0x1000
LPC804	0x1000
KL26	0x8000
KE18F	0x8000
i.MXRT1021	0x40000
i.MXRT1052	0x40000
i.MXRT1062	0x40000
i.MXRT1064	0x40000
LPC51U68(支持UART和USB_HID两种boot loader)	0x8000

文件结构

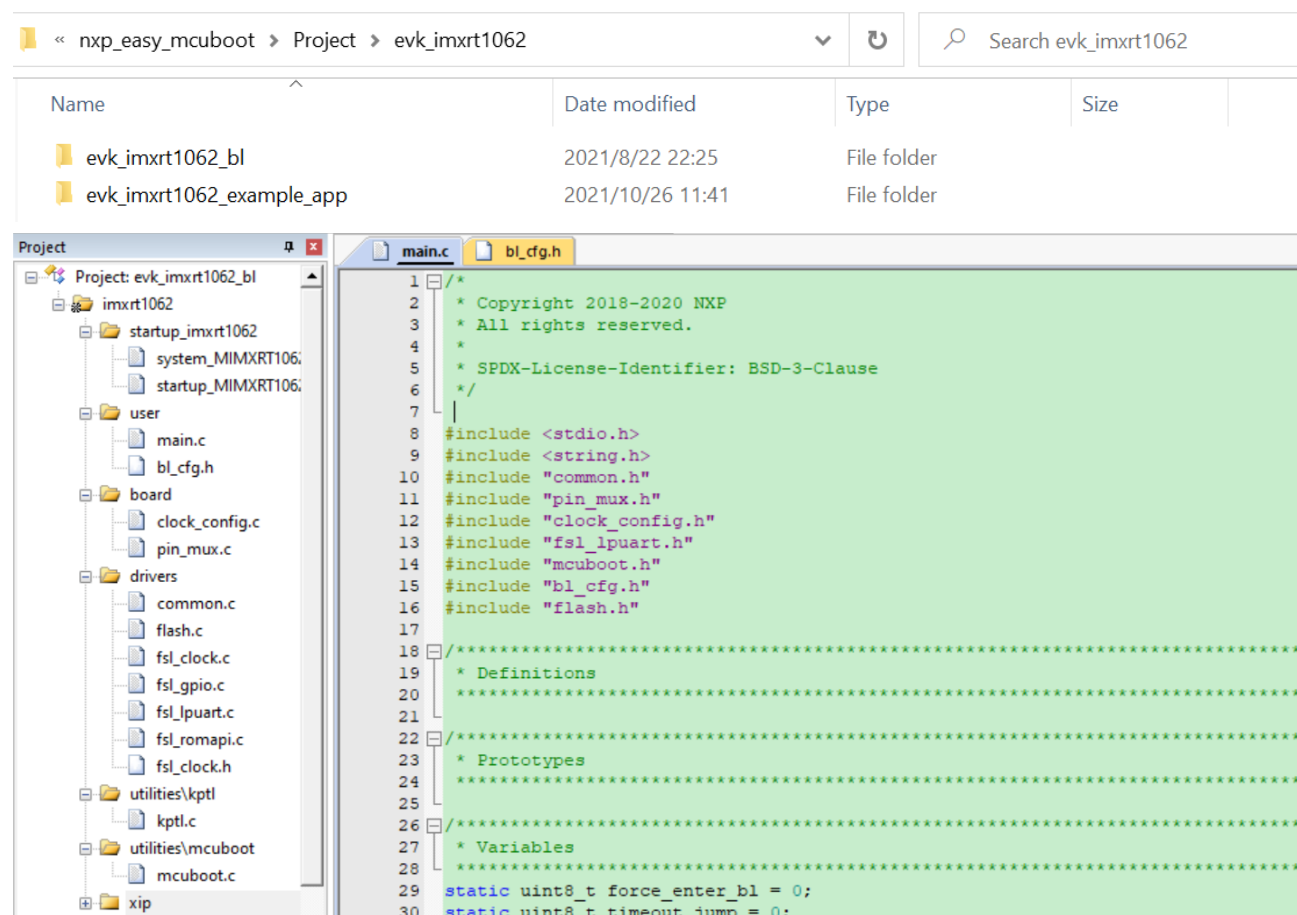
```
npx_easy_mcuboot
├── Libraries - 存放所有库文件, bootloader源文件
│   ├── drivers_k64 - NXP KinetisK64 驱动文件
│   ├── drivers_lpc800 - NXP LPC800驱动文件
│   ├── drivers_ke - NXP Kinetis KE0xZ驱动文件
│   ├── drivers_ke15 - NXP Kinetis KE1xZ和KE1xF驱动文件
│   ├── drivers_lpc800 - NXP LPC800驱动文件
│   ├── drivers_lpc800 - NXP LPC800驱动文件
│   ├── drivers_kl - NXP KL驱动文件
│   ├── drivers_rt1021 - NXP i.MXRT102x驱动文件
│   ├── drivers_rt1052 - NXP i.MXRT105x驱动文件
│   ├── drivers_rt1062 - NXP i.MXRT106x驱动文件
│   ├── startup - 所有MCU启动文件
│   └── utilities - 存放bootloader源代码
├── Project - example 工程
│   ├── frdm_k64 基于FRDM-K64的 bootloader示例工程
│   ├── frdm_ke02 基于FRDM-ke02的 bootloader示例工程
│   ├── frdm_ke15 基于FRDM-ke15Z的 bootloader示例工程
│   ├── lpc802 基于LPC802的 bootloader示例工程
│   ├── lpc804 基于LPC804的 bootloader示例工程
│   ├── frdm_kl26 基于frdm_kl26的 bootloader示例工程
│   ├── evk_imxrt1052_HyperFlash 基于evk_imxrt1052(HyperFlash)的 bootloader示例工程
│   ├── evk_imxrt1052_QspiFlash 基于evk_imxrt1052(QSPIFlash)的 bootloader示例工程
│   ├── evk_imxrt1021 基于evk_imxrt1021的 bootloader示例工程
│   ├── evk_imxrt1062 基于evk_imxrt1062的 bootloader示例工程
│   └── evk_imxrt1064 基于evk_imxrt1064的 bootloader示例工程
├── pc_tool - PC工具
│   ├── KinetisFlashTool.exe GUI工具, 直接使用这个下载程序
│   ├── KinetisFlashTool.ini
│   ├── blfwkdll.dll
│   └── bootloader.log
└── readme.md
```



# NXP Easy MCU Second Bootloader——Hands-On

下面我们以EVK\_RT1062开发板为例演示如何使用NXP Easy MCU Boot:

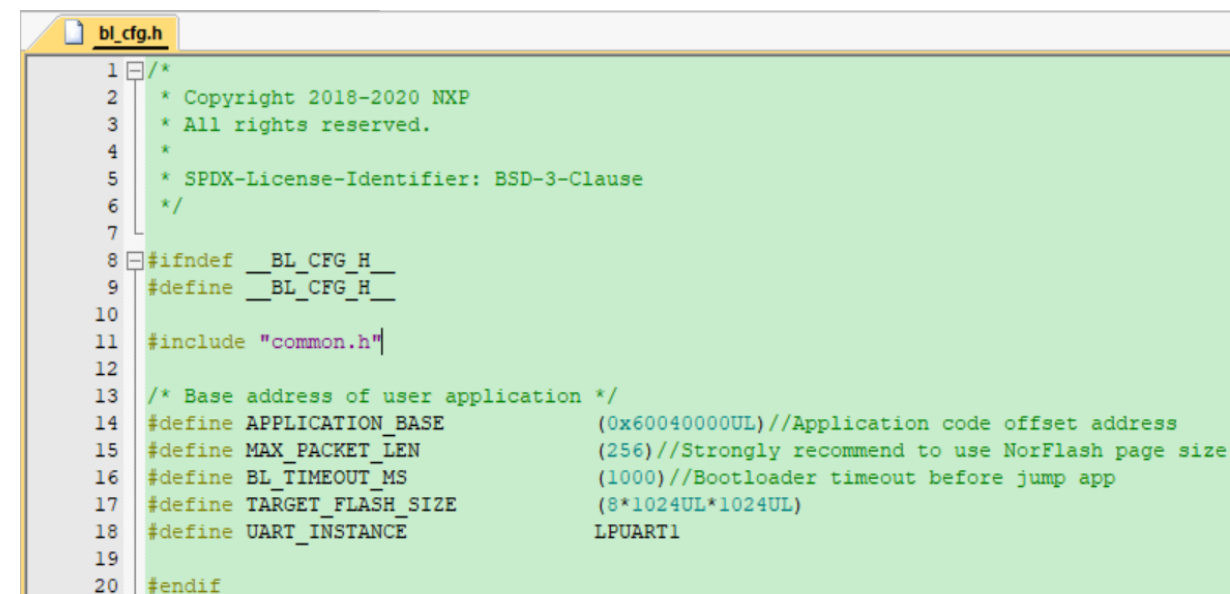
1. 首先从github仓库下载整个工程包并解压, 进入\nxp\_easy\_mcuboot\Project\evk\_imxrt1062路径, 可以看到两个工程如下图所示, 一个为2nd Bootloader工程 (evk\_i.mxrt1062\_bl), 另一个为应用工程示例 (evk\_i.mxrt1062\_example\_app), 先打开Bootloader工程, 结合Readme文档熟悉整个工程框架;



# NXP Easy MCU Second Bootloader——Hands-On

2. 对首次使用的用户来说，最先要关注的就是bl\_cfg文件，如下图所示，其中几个重要参数解释如下：

- APPLICATION\_BASE:  
用户应用代码的偏移地址，注意一定要以Flash Sector大小对齐（因为Bootloader要调用的Erase API的入参为该地址），对RT来说，QSPI Flash的sector一般以4KB为主，而HyperFlash的sector一般以256KB为主，所以这里为兼容RT1052\_EVK的HyperFlash，默认都配置为256KB（它也是4KB对齐的）；
- MAX\_PACKET\_LEN:  
上位机通过UART一次下发数据包的大小，建议配置为NorFlash的Page大小，一般为256 Bytes（如果不设置该参数则默认为64 Bytes），这样可以实现每次下发可以写入整个page提高烧写效率，最后一包如果不满足该大小则只写入剩余字节数；
- BL\_TIMEOUT\_MS:  
Bootloader等待超时时间参数，默认为1s，即上电或者复位后Bootloader等待时间超过1s且APPLICATION\_BASE存在有效应用代码则正常跳入应用，如果不存在有效代码则该参数会被忽略，Bootloader处于一直等待上位机更新命令状态；
- TARGET\_FLASH\_SIZE:  
整个Flash大小，实际填入所使用的SPI NorFlash大小即可；
- UART\_INSTANCE:  
当前使用的串口实例，注意如果将其修改为其他串口实例，对应的pin\_mux.c里要手动配置好管脚复用，它们并不直接关联。

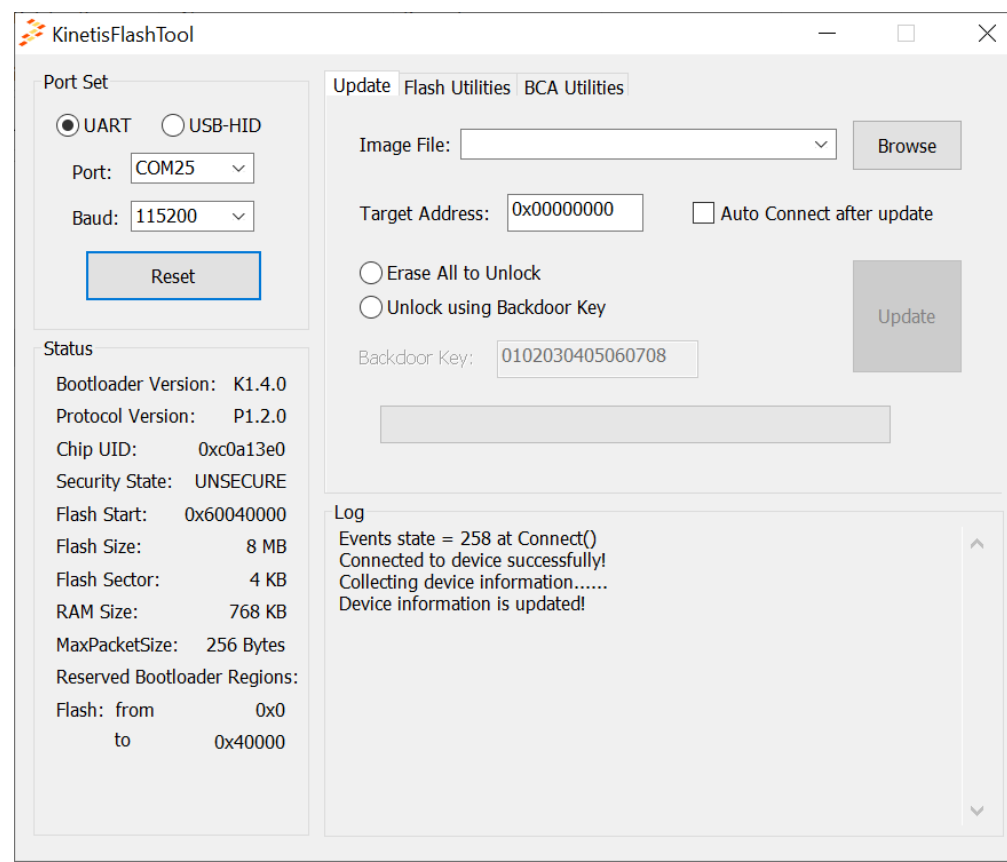


```
1  /*
2   * Copyright 2018-2020 NXP
3   * All rights reserved.
4   *
5   * SPDX-License-Identifier: BSD-3-Clause
6   */
7
8  #ifndef BL_CFG_H
9  #define BL_CFG_H
10
11  #include "common.h"
12
13  /* Base address of user application */
14  #define APPLICATION_BASE (0x60040000UL) //Application code offset address
15  #define MAX_PACKET_LEN (256) //Strongly recommend to use NorFlash page size
16  #define BL_TIMEOUT_MS (1000) //Bootloader timeout before jump app
17  #define TARGET_FLASH_SIZE (8*1024UL*1024UL)
18  #define UART_INSTANCE LPUART1
19
20 #endif
```

# NXP Easy MCU Second Bootloader——Hands-On

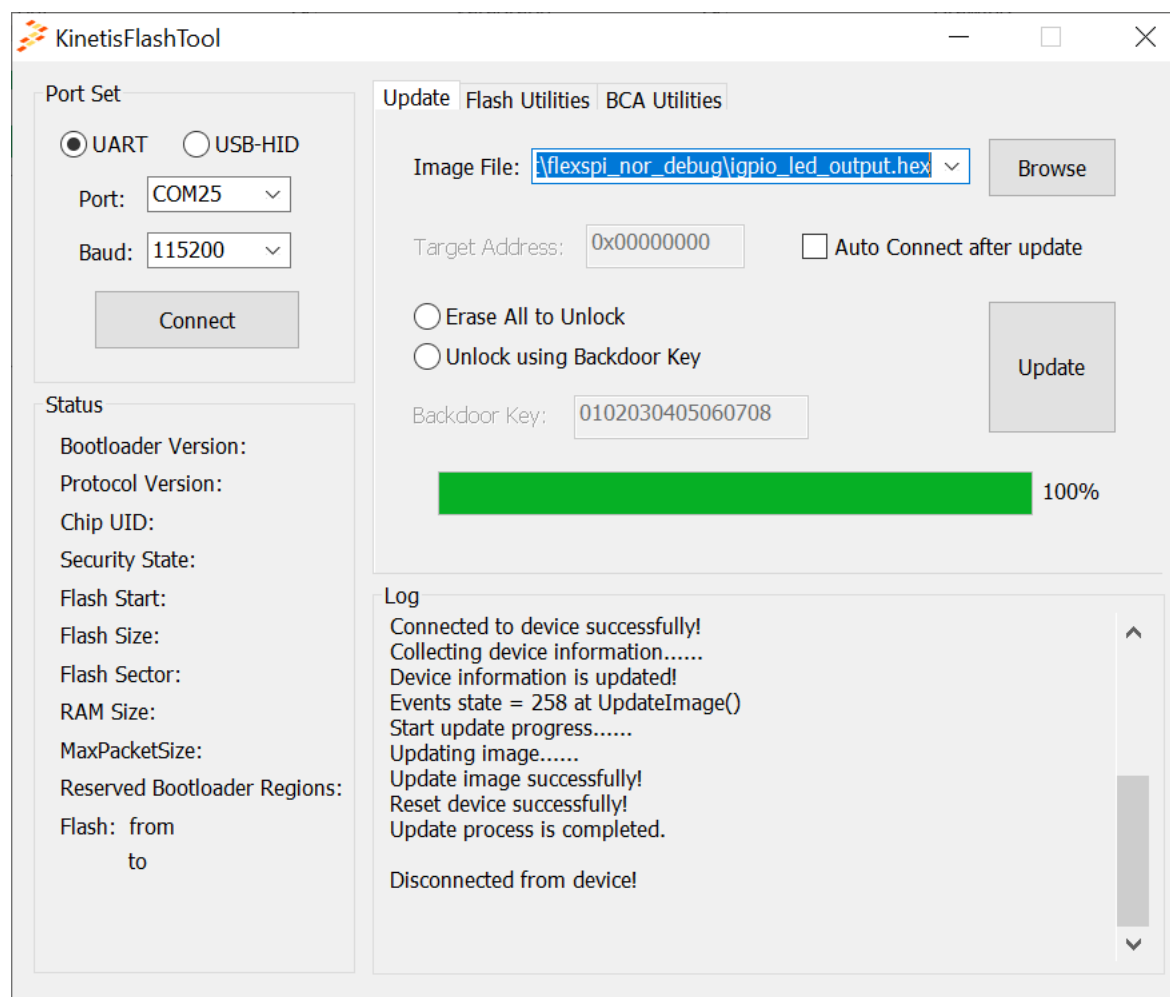
3. 编译Bootloader工程并下载到目标板运行，进入\nxp\_easy\_mcuboot\pc\_tool路径打开KineticFlashTool上位机，配置好串口端口以及波特率（默认115200bps），建议按照顺序先点击上位机Connect再复位芯片（这样确保上位机先发Ping包，芯片复位后在等待超时之前接收到ping包），连接成功后上位机会自动向目标芯片索取相关信息并显示出来，如下图：

\* 注：第一次下载Bootloader是很容易连接成功的（因为没有有效的应用代码，Bootloader一直停在Boot模式），一旦更新过一次固件之后，由于只有1s的超时且Bootloader里使用的ping包接收使用的轮询方式，有可能一次并不能连接成功，此时多试几次上述流程（先Connect再复位芯片）或者把默认的1s超时调长些会改善连接成功率。



# NXP Easy MCU Second Bootloader——Hands-On

4. 打开\Project\evk\_imxrt1062\evk\_imxrt1062\_example\_app\igpio\_led\_output路径下的应用工程示例，编译生成hex文件，然后在KinetisFlashTool工具中Image File处选择生成的hex文件，最后点击Update按钮即可成功完成升级，且升级成功后会自动复位MCU运行，结果如下图：



# NXP Easy MCU Second Bootloader——Hands-On

5. 接下来我们使用NXP官方的命令行工具Blhost来更新RT1062的固件，进入到下载下来的blhost.exe所在路径（\blhost\_2.6.2\bin\win），把前面生成的应用工程文件igpio\_led\_output.hex文件拷贝到该路径下，然后打开Windows自带的命令行工具cmd且进入到该路径下，敲入“blhost -p COMxx,115200 get-property 1”，并复位MCU，可以发现并没有ping成功，如下图，这是因为每次使用blhost下发一次命令只能发送一次ping包（前文的KinetisFlashTool可以连续发多次ping），而MCU超时又只有1s，所以这个时序很难把握；

```
C:\WINDOWS\system32\cmd.exe
C:\Users\<redacted>>cd C:\Users\<redacted>\Desktop\blhost_2.6.2\bin\win
C:\Users\<redacted>\Desktop\blhost_2.6.2\bin\win>blhost -p COM25,115200 get-property 1
Error: Initial ping failure: No response received for ping command.
C:\Users\<redacted>\Desktop\blhost_2.6.2\bin\win>_
```

6. 为解决该问题，我们可以把Bootloader的超时时间改长些，如下图，改成5s，然后重新编译并下载到目标板上，可以看到每次复位都会等待较长一段时间才会跳转到app；

```
bl_cfg.h
3  * All rights reserved.
4  *
5  * SPDX-License-Identifier: BSD-3-Clause
6  */
7
8  #ifndef BL_CFG_H
9  #define BL_CFG_H
10
11  #include "common.h"
12
13  /* Base address of user application */
14  #define APPLICATION_BASE (0x60040000UL) //Application code offset address
15  #define MAX_PACKET_LEN (256) //Strongly recommend to use NorFlash page size
16  #define BL_TIMEOUT_MS (5*1000) //Bootloader timeout before jump app
17  #define TARGET_FLASH_SIZE (8*1024UL*1024UL)
18  #define UART_INSTANCE LPUART1
```

# NXP Easy MCU Second Bootloader——Hands-On

7. 此时我们再次敲入“blhost -p COMxx,115200 get-property 1”（先不敲回车），然后复位芯片，接着敲入回车键执行，可以看到ping通且MCU正常返回响应包，接下来我们可以挨个试试get-property的命令如下图，如果想查看blhost都有哪些命令可以敲入“blhost -?”来查看，当然不是所有命令在MCU端都有实现的；

```
C:\WINDOWS\system32\cmd.exe
C:\Users\...\Desktop\blhost_2.6.2\bin\win>blhost -p COM25,115200 get-property 1
Ping responded in 1 attempt(s)
Inject command 'get-property'
Response status = 0 (0x0) Success.
Response word 1 = 1258357760 (0x4b010400)
Current Version = K1.4.0

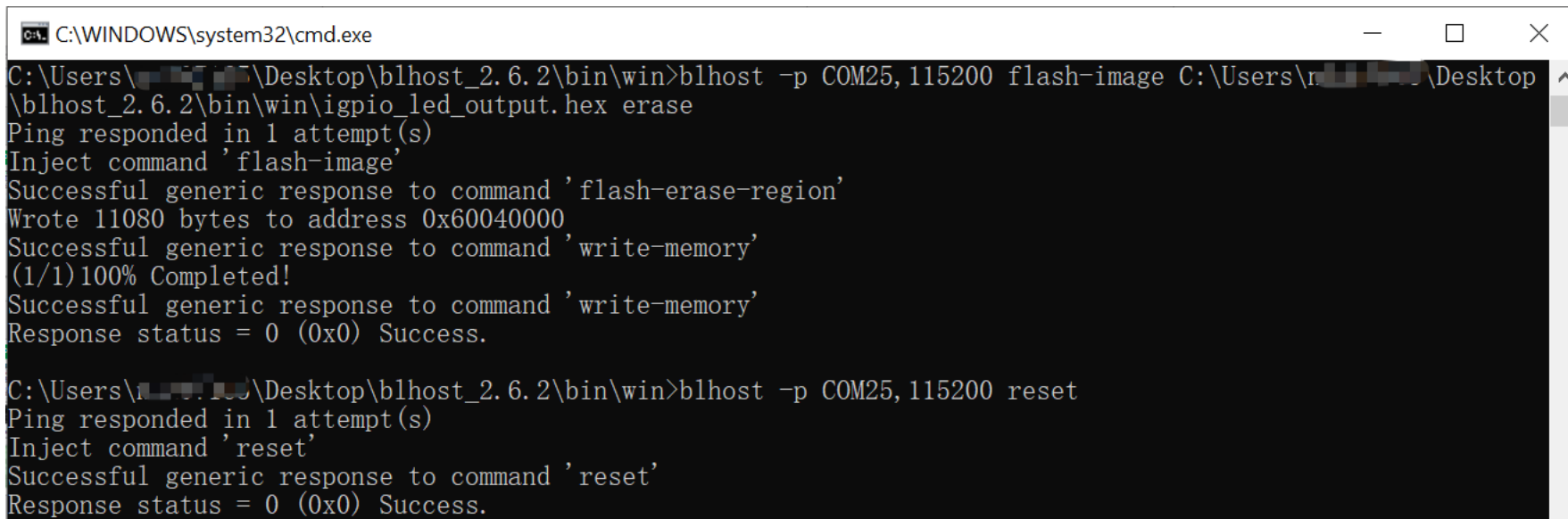
C:\Users\...\Desktop\blhost_2.6.2\bin\win>blhost -p COM25,115200 get-property 2
Ping responded in 1 attempt(s)
Inject command 'get-property'
Response status = 0 (0x0) Success.
Response word 1 = 1 (0x1)
Available Peripherals = UART

C:\Users\...\Desktop\blhost_2.6.2\bin\win>blhost -p COM25,115200 get-property 3
Ping responded in 1 attempt(s)
Inject command 'get-property'
Response status = 0 (0x0) Success.
Response word 1 = 1610874880 (0x60040000)
Flash Start Address = 0x60040000
```



# NXP Easy MCU Second Bootloader——Hands-On

8. 最后敲入“blhost -p COMxx,115200 flash-image 烧写文件完整路径 erase”，待烧写成功完成后，还需要发送reset命令让MCU执行复位运行，效果如下图；



```
C:\WINDOWS\system32\cmd.exe
C:\Users\...\Desktop\blhost_2.6.2\bin\win>blhost -p COM25,115200 flash-image C:\Users\...\Desktop\blhost_2.6.2\bin\win\igpio_led_output.hex erase
Ping responded in 1 attempt(s)
Inject command 'flash-image'
Successful generic response to command 'flash-erase-region'
Wrote 11080 bytes to address 0x60040000
Successful generic response to command 'write-memory'
(1/1)100% Completed!
Successful generic response to command 'write-memory'
Response status = 0 (0x0) Success.

C:\Users\...\Desktop\blhost_2.6.2\bin\win>blhost -p COM25,115200 reset
Ping responded in 1 attempt(s)
Inject command 'reset'
Successful generic response to command 'reset'
Response status = 0 (0x0) Success.
```

从上面的执行结果来看，实际上KinetisFlashTool GUI里面就是封装了Blhost一段一段的命令下发给MCU来执行的，这也是本开源项目很大的特点，兼容Blhost协议。如果用户想要实现自己的上位机则可以参考NXP官方的MCUBoot协议实现文档，MCUBLHOSTUG和MCUX\_Flashloader\_ReferenceManual，如下；

<https://www.nxp.com.cn/docs/en/user-guide/MCUBLHOSTUG.pdf>

<https://community.nxp.com/pwmxy87654/attachments/pwmxy87654/imxrt/1417/2/MCUX%20Flashloader%20Reference%20Manual.pdf>





SECURE CONNECTIONS  
FOR A SMARTER WORLD