

U-boot 移植应用 开发指南

文档版本 V2.0

发布日期 2025-05-28

概述

本文主要介绍如何移植和烧写 U-boot (Bootloader)。

为了便于说明，本文定义了以下术语：

术语	说明
<platform>	芯片平台代号，代表某个系列的芯片，例如 xmfalcon、xmorca
<soc>	芯片名，代表某个具体的芯片，例如 xm7606v13、xm7206v11a
<toolchain>	代表工具链，例如 arm-gcc12.2.0-linux、arm-gcc12.2.0-linux-uclibceabi

关于<platform>：

<platform>	说明	对应的<soc>
xmfalcon	xm7606v1x 系列芯片平台代号	xm7606v13 xm7605v13 xm7605v12 xm7605v11 xm7506v13
xmorca	xm7206v1x 系列芯片平台代号	xm7206v11a xm7206v10b xm7206v10

关于<toolchain>：

<toolchain>	说明
arm-gcc12.2.0-linux	GCC12.2, ARM32 工具链, 配套 C 库为 Glibc-2.37, 仅适用于 xmfalcon 平台
arm-gcc12.2.0-linux-ucLibcabi	GCC12.2, ARM32 工具链, 配套 C 库为 uClibc-ng-1.0.48, 仅适用于 xmorca 平台

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示如不可避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不可避免则可能导致死亡或严重伤害的具有中等级风险的危害。

符号	说明
 注意	表示如不可避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2024-12-21	V1.0	DI 版本发布。
2025-03-04	V1.1	修正部分描述错误。
2025-05-28	V2.0	支持 xm7206v1x 系列芯片，考虑与 xm7606v1x 兼容，增加了一些术语定义便于说明。

目 录

1 概述.....	1
1.1 概述	1
1.2 U-boot 目录结构	1
2 移植 U-boot.....	3
2.1 U-boot 硬件环境	3
2.2 SDK 配置 U-boot.....	3
2.3 修改 u-boot 配置方法.....	4
2.4 编译 U-boot.....	5
2.5 u-boot 定制	5
2.5.1 关闭 boot env	5
2.5.2 缩减分区空间	6
3 烧写 U-boot.....	7
3.1 概述	7
3.2 通过 U-boot 命令行烧写 U-boot.....	7
3.2.1 SPI-Nor Flash 烧写方法.....	7
3.2.2 SPI-Nand Flash 烧写方法.....	8
3.2.3 eMMC 的 U-boot 烧写方法	8
4 附录.....	10
4.1 如何修改 boot 表格.....	10
4.2 Boot 表格中的板级配置.....	12

4.2.1 SYSBOOT1 寄存器.....	12
4.3 SPI-Nor 块保护命令	14
4.4 移植 U-boot 到其它构建环境	18
4.4.1 准备工作.....	18
4.4.2 编译.....	18

仅限深圳市安远威视科技有限公司使用

插图目录

图 4-1 Boot 表格示例 10

图 4-2 查看当前块保护信息 16

图 4-3 锁定整个器件 17

图 4-4 解除当前锁定状态 17

图 4-5 通过设置 level 值锁定指定区域 17

表格目录

表 1-1 U-boot 的主要目录结构.....	1
表 4-1 xmfalcon 平台的 SYSBOOT1 寄存器定义.....	12
表 4-2 xmorca 平台的 SYSBOOT1 寄存器定义.....	13
表 4-3 不同厂家不同器件的块保护锁定区域与 BP Level 对应表	15

1 概述

1.1 概述

Bootloader 采用 U-boot-2020.01, U-boot 是 DENX 社区的开源 boot, 支持众多芯片厂商的 SoC 和参考板, 支持网络、USB、SD 等多种常用外设, 支持 FAT 等文件系统, 社区长期活跃, 是当前主流的开源 boot。

1.2 U-boot 目录结构

U-boot 的主要目录结构如表 1-1 所示, 详细目录说明请阅读 U-boot 目录下的 README 文档。

表1-1 U-boot 的主要目录结构

目录名	描述
arch	各种芯片架构的相关代码、U-boot 入口代码
arch/xxx/lib	各种体系结构的相关代码, 如 ARM、MIPS 的通用代码
board	各种单板的相关代码, 主要包括存储器驱动等
include	头文件
include/configs	各种单板的配置头文件, 主要用于相对固定配置
configs	各种单板的配置文件, 主要用于变更较多的配置
common	各种功能通用功能的实现文件

目录名	描述
drivers	网口、Flash、串口等的开源驱动代码。
cmd	各种命令的实现文件
net	网络协议实现文件
fs	文件系统实现文件
env	环境变量实现文件
lotus	小系统的驱动和模块代码，包含网口、Flash、串口和 I2C 等模块
product	业务模块代码，包含升级、图形等模块

2 移植 U-boot

2.1 U-boot 硬件环境

移植 U-boot 前，需要先确认硬件单板与 RB 板的差异，主要包括：

- 存储器件，包括
 - 单板的 DDR SDRAM：DDR3、DDR4 和 LPDDR4
 - 单板的 Flash：eMMC、SPI Nor Flash 和 SPI-NAND Flash

当前版本已兼容的存储器件见兼容性器件列表文档《XM Flash DDR Support List.xlsm》、《XM Sensor Support List.xlsx》，如果需要移植新的 Flash 器件，请提兼容性验证需求！

- 管脚复用

管脚复用需要通过 boot 表格文档的 pinout 页配置，具体的寄存器配置说明请参考芯片 PIN_OUT.xlsx 文档。

2.2 SDK 配置 U-boot

步骤 1 配置 boot 表格

在 Windows 下，打开 SDK 中的 “configs/<soc>/reg” 目录下的对应 RB 板的配置表格，表格通常以单板层数和 DDR 型号命名。

须知

此处<SOC>代表芯片名，具体可用芯片名见文档开头的术语说明。

根据单板实际配置，修改管脚复用 pinout 页，表格修改方法参考 4.1 如何修改 boot 表格；

步骤 2 SDK 配置中指定 Boot reg 文件

```
CONFIG_XMEDIA_BOOT_REG_NAME=xxx
```

步骤 3 SDK 配置中指定 U-boot 配置

```
CONFIG_XMEDIA_BOOT_DEFCONFIG=<platform>_defconfig
```

----结束



说明

<platform>_defconfig 是缺省的配置，不同的特性可能会有不同的配置，对于快启场景，需要选择 <platform>_quickstart_defconfig。

2.3 修改 u-boot 配置方法

如果需要修改 u-boot 配置，步骤如下：

步骤 1 准备 SDK 配置。

拷贝需要编译 SDK 配置到 SDK 根目录下，详细的说明请参考《SDK 开发环境用户指南》。

步骤 2 SDK 下编译一次 uboot。

```
make uboot
```

步骤 3 进入 boot 编译输出目录，运行 menuconfig。

```
cd out/<soc>/boot_builddir/  
make ARCH=arm CROSS_COMPILE=<toolchain>- menuconfig
```

步骤 4 根据实际需要，修改配置项，并保存退出。如果了解 menuconfig 界面操作，可以参考界面顶部的帮助提示。

步骤 5 拷贝修改后的配置到 u-boot 源码目录。

```
cp .config source/configs/<platform>_defconfig
```

----结束

2.4 编译 U-boot

通常在 SDK 根目录下编译 U-boot，操作如下：

1. 编译

```
make uboot -j 16
```

生成的镜像名称为 uboot.bin

2. 清除

```
make uboot_clean
```

3. 编译烧写用的 boot（仅部分场景需要，见下面说明）

```
make bootstrap -j 16
```

生成的镜像名称为 bootstrap.bin

4. 清除烧写用的 boot

```
make bootstrap_clean
```

说明

通常 make uboot 即可编译全功能的 boot，支持启动、烧录、开启打印，包含 SPI-Nor、SPI-Nand、网口、USB 等常用外设驱动。一些特殊场景，比如快启，默认 quickstart 的配置会裁剪烧录、关闭打印、去掉不需要的驱动，因此需要单独编译用于烧录用的 boot，以便于调试，此时需要用 make bootstrap 命令来编译。

2.5 u-boot 定制

SDK 发布的 u-boot 提供了大部分常用的功能，包括启动、烧录、升级、网络、USB 等功能，产品化时可以定制修改，以下是一些定制修改方法。

2.5.1 关闭 boot env

SDK 发布版本默认会有 boot env 分区（通常镜像名为 bootargs.bin），该分区用于支持定制 boot 环境参数，方便用户调试，通常为 64K。如果不需要，可以裁剪去掉，需要做如下修改：

- 关闭 CONFIG_ENV_IS_IN_SPI_FLASH，并打开 CONFIG_ENV_IS_NOWHERE；

- 打开 CONFIG_USE_BOOTARGS, 并配置 CONFIG_BOOTARGS 为 bootargs 变量值, 参考原 boot env 的配置;
- 打开 CONFIG_USE_BOOTCOMMAND, 并配置 CONFIG_BOOTCOMMAND 为 bootcmd 变量值, 参考原 boot env 的配置。

2.5.2 缩减分区空间

- 考虑到便于维测调试, SDK 发布包默认为各个分区预留了一些余量, 例如, 通常 boot 分区预留 512K、bootenv 分区预留 512K 等, 这些分区大小通常比实际镜像会大, 用户可以根据自身定制镜像大小来调整分区大小, 注意, 需要考虑 Flash 块大小对齐。
- 调整分区大小需要注意以下几点:
 1. 调整 boot env 分区位置和大小后, 需要同步调整 boot 配置的 CONFIG_ENV_OFFSET、CONFIG_ENV_SIZE 和 SDK 配置的 CONFIG_XMEDIA_BOOT_ENV_SIZE;

CONFIG_XMEDIA_BOOT_ENV_SIZE 主要用于制作 boot env 镜像时给制作工具传递 boot env 的大小;

示例:

假定 boot env 分区原来是 512K 开始, 大小为 512K, 希望调整到 128K 开始, 大小为 64K, 则 CONFIG_ENV_OFFSET 由 0x80000 改为 0x20000, CONFIG_ENV_SIZE 由 0x80000 改为 0x10000, 同时, 调整 SDK 配置的 CONFIG_XMEDIA_BOOT_ENV_SIZE 为 0x10000;
 2. 调整内核分区起始地址和大小后, 需要同步调整 bootcmd 中读取内核的起始地址和大小;
 3. 如果有修改 rootfs 分区在分区表中的位置, 需要同步调整 bootargs 变量中 root 配置的分区序号; 例如, 假定原 rootfs 分区节点为/dev/mtdblock3, 如果在 rootfs 分区前插入了一个分区, 则应将 root=/dev/mtdblock3 改为 root=/dev/mtdblock4;
 4. 调整分区表后, 需要同步调整工具烧录的 xml 文件和量产烧录的相关配置文件。

3 烧写 U-boot

3.1 概述

u-boot 可以通过三种方法烧写：

1. 通过 FastBurn 工具烧写

调试场景，通常用 FastBurn 工具烧录 U-boot 即可，具体操作方式请参考《FastBurn 工具使用指南》；

2. 通过 SD 卡升级，具体操作方法参考《裸烧及非裸烧升级使用手册》；

3. 通过命令行手动操作烧录，此方法要求单板已有可运行的 boot。

须知

如果单板 flash 是空片或是板上已有的 boot 已损坏，则必需使用 FastBurn 或是 SD 卡升级进行烧写。

本章主要介绍如何通过命令行烧写 U-boot。

3.2 通过 U-boot 命令行烧写 U-boot

3.2.1 SPI-Nor Flash 烧写方法

SPI-Nor Flash 烧写方法如下：

步骤 1 boot 在内存中运行起来之后在超级终端中输入：

```
# mw.b <ddr_addr> ff 0x80000          /* 对内存初始化 */
# tftp <ddr_addr> uboot.bin           /* U-boot下载到内存，没有网口时可以用loady */
# sf probe 0                          /* 探测并初始化SPI-Nor flash */
```

```
# sf erase 0x0 0x80000 /* 擦除前512K，以实际boot分区大小为准 */  
# sf write <ddr_addr> 0x0 0x80000 /* 从内存写入SPI-Nor Flash */
```



说明

<ddr_addr>可用地址 0x41000000，以下相同，不再重述。

步骤 2 上述步骤操作完成后，重启检查 U-boot 是否烧写成功。

----结束

须知

在当前版本，使用 sf lock 可以对 SPI Nor Flash 进行块保护（Blocks Protect）。如果对 SPI Nor Flash 的某个块进行了块保护，这个块就变成只读，运行擦除和写命令都不会生效，而且掉电并不能失效块保护。在这种情况下，只有在执行 sf lock 0 命令，解除块保护之后，SPI Nor Flash 擦除和写操作才会起作用。详见 SPI-Nor 块保护命令章节。

3.2.2 SPI-Nand Flash 烧写方法

SPI-Nand Flash 烧写方法如下：

步骤 1 在内存中运行起来之后在超级终端中输入：

```
# mw.b <ddr_addr> 0xff 0x80000 /* 对内存初始化 */  
# nand erase 0 0x80000 /* 擦除前512K，以实际boot分区大小为准 */  
# tftp <ddr_addr> uboot.bin /* U-boot下载到内存，没有网口时可以用loady */  
# nand write <ddr_addr> 0 0x80000 /* 从内存写入NAND Flash */
```

步骤 2 重启系统检查 U-boot 是否烧写成功。

----结束

3.2.3 eMMC 的 U-boot 烧写方法

eMMC 烧写方法如下：

步骤 1 在内存中运行起来之后在超级终端中输入：

```
# mw.b <ddr_addr> 0xff 0x80000 /* 对内存初始化 */  
# tftp <ddr_addr> uboot.bin /* U-boot下载到内存，没有网口时可以用loady */
```



```
# mmc write 0 <ddr_addr> 0 0x400          /*从内存写入eMMC*/
```

说明

- mmc write 命令说明:

格式: mmc write <device num> addr blk# cnt

- 参数:

<device num>: 设备号

addr: 源地址

blk#: 目的地址的块序号

cnt: 块的数目, 每块大小为 512 字节

步骤 2 重启系统检查 U-boot 是否烧写成功。

----**结束**

4 附录

4.1 如何修改 boot 表格

⚠ 注意

Boot 表格包含了关键的启动配置，错误的修改可能导致启动失败等问题，修改前后请务必仔细校对，建议修改前后使用比较工具确认修改的正确性！

通常在修改管脚复用时需要修改表格，以下是具体步骤

- 步骤 1 在 Windows 下，双击 boot 表格文件；
- 步骤 2 选择底部的 pinout 页，图 4-1 是一个示例；

图4-1 Boot 表格示例

Module Name	PINOUT					
Base Address	0x0					
Priority	7					
For Standby Resume	N	页信息				
For Normal Boot	Y					
Register	Offset	Value (For reading, fill the expected value)	Delay	Read or Write	Number of Bits - 1 (For example, if you need to set 2 bits, fill '1' here)	Start Bit
iocfg_reg58	0x112C00d8	0x1001	0	write	31	0
iocfg_reg59	0x112C00dc	0x1001	0	write	31	0
iocfg_reg0	0x100C0000	0x1201	0	write	31	0
iocfg_reg1	0x100C0004	0x1101	0	write	31	0
iocfg_reg16	0x100C0008	0x1101	0	write	31	0
iocfg_reg17	0x100c000c	0x1101	0	write	31	0
iocfg_reg18	0x100c0010	0x1101	0	write	31	0
iocfg_reg19	0x100c0014	0x1101	0	write	31	0

表格包含多个页，每一页包含页信息和寄存器配置两部分，黄色区域是可修改部分，每一列的具体含义如下：

1. 页信息：

- Module Name: 当前页名称, 仅用于标识用, 不作实际使用, 建议与页命名一致, 便于交流;
- Base Address: 当前页所有寄存器的基地址, Offset 列以此处值为基地址;
- Priority: 当前页配置的优先级, 需与当前页在表格中的位置序号一致;
- For Standby Resume: 是否在待机后唤醒时配置;
- For Normal Boot: 是否在开机启动时配置。

2. 寄存器配置:

每一行代表一个寄存器操作, 包括是读还是写、需要读写的地址、需要读写的 bits、读写后的延时等, 具体如下:

- Register: 寄存器名称, 仅用于标识用, 不作实际使用, 建议与芯片手册命名一致, 便于交流;
- Offset: 寄存器偏移, 页信息中的 Base Address 与此处值的和即为最终需要操作的寄存器地址;
- Value: 如果是 Write, 则是需要写入的值, 如果是 Read, 则是期望读出的值;
- Delay: 读写操作后的延时, 以微秒 (us) 为单位;
- Read or Write: 标记当前是读还是写, 从下拉菜单中选择;
- Number of Bits - 1: 需要读写的比特数减 1, 例如, 需要写入 2 个 bit, 则此处填 1, 如果需要读写整个寄存器的 32 个 bit, 则此处填 31;
- Start Bit: 需要读写的起始比特, 以 0 开始, 例如, 需要写第 1bit, 则填 1, 如果需要读第 31bit, 则填 31。

步骤 3 根据上述描述, 按需修改, 然后保存, 并重新编译 boot 即可。Boot 编译过程中会调用 tools/linux/utlis/uboot_tools/regbin 工具将 boot 表格文件转换为 bin 格式的 .reg 文件, 并嵌入到 uboot 最终的镜像中, boot 初始过程中会解析 reg 文件, 并逐个配置到芯片中。

----结束

须知

1. 如果是需要增加寄存器，直接通过文档工具的插入或是在当前页的最后一项之后添加的方法可能是无效的，而**必须从已有的一行配置复制粘贴**，然后再修改为所需的寄存器配置；
2. 每一项填写的值前后都不能有空格。

4.2 Boot 表格中的板级配置

⚠ 注意

Boot 表格包含了关键的启动配置，错误的修改可能导致启动失败等问题，修改前后请务必仔细校对，建议修改前后使用比较工具确认修改的正确性！

4.2.1 SYSBOOT1 寄存器

此寄存器是芯片内的一个通用寄存器，用于配置部分板级相关的信息，在 boot 或是软件驱动中将使用这部分信息来做单板相关的配置，具体意义如下：

表4-1 xmfalcon 平台的 SYSBOOT1 寄存器定义

Bits	描述
[31:13]	保留，需固定配置 0
[12:11]	SDIO1 走线长度： 0: 2 inch 1: 4 inch
[10:9]	SDIO0 走线长度： 0: 2 inch 1: 4 inch
[8:7]	eMMC 走线长度： 0: 1 inch 1: 2 inch

Bits	描述
[6:5]	标识单板的层数： 0: 四/六层板 1: 二层板
[4]	标识当前单板 DDR 是否晶凯残片 DDR： 0: 非晶凯残片 DDR 1: 晶凯残片 DDR
[3]	保留，需固定配置 0
[2]	标识是否使用双 U 口： 0: 单 U 口 (U3 控制器 + U3&U2 PHY) 1: 双 U 口 (U3 控制器 + U3 PHY , U2 控制器 + U2 PHY)
[1]	标识网口 PHY 的输入是否有反相，即 EPHY_RSTN 输出的信号是否有经过反相后送到网口 PHY 芯片： 0: 直连 1: 反相。
[0]	标识网口网口 MAC 控制器与 PHY 接口： 0: RMII 1: RGMII

表4-2 xmorca 平台的 SYSBOOT1 寄存器定义

Bits	描述
[31:4]	保留，需固定配置 0
[3]	标识网口 MAC 控制器与 PHY 的接口类型： 0: 使用 bit[0]的定义 1: MII
[2]	保留，需固定配置 0

Bits	描述
[1]	标识网口 PHY 的输入是否有反相，即 EPHY_RSTN 输出的信号是否有经过反相后送到网口 PHY 芯片： 0: 直连 1: 反相。
[0]	标识网口 MAC 控制器与 PHY 接口： 0: RMII 1: RGMII

4.3 SPI-Nor 块保护命令

常用的 SPI Nor Flash 上都提供了块保护位 (Block Protect: 以下简称 BP) 来保护数据安全。

通过设置状态寄存器 (Status Register: 以下简称 SR) 中的 BP0、BP1、BP2、BP3 (某些厂家的芯片没有 BP3 或者存在 BP4) 几个 Bit 为 1(使能状态), 使器件中某些对应的块进入写保护状态, 这些 BP 位为非易失性位 (Non-volatile Bit), 设置之后可以掉电保持之前状态。

有的厂商还提供了对块保护锁定方向的设置, 可以设置保护的块从器件顶部 Top 开始还是从器件底部 Bottom 开始。通过设置配置寄存器 (Config Register: 以下简称 CR) 中的 TBPROT 位 (某些厂家的芯片 TBPROT 位位于 SR) 设置写保护锁定起始地址是从低地址 (Bottom) 还是从高地址 (Top) 开始。通常这个设置位属于 OTP (One Time Program) 类型, 默认状态为 0: BP 开始于 Top 部 (高地址), 一旦状态被设置为 1: BP 开始于 Bottom (低地址), 且将无法更改。

根据实际应用, 我们控制器从初始化开始就将 TBPROT 位置为 1, 即从 Bottom 低地址开始保护。

SPI-Nor 器件状态寄存器 SR 的默认初始值中, 所有 BP 位都为 0 (去使能状态), 此时器件上所有的块都处于未保护状态, 可以任意进行擦写操作。

设置所有 BP 位都为 1 (使能状态), 将使器件上所有的块都处于写保护状态, 任何擦写操作都将无效。

块保护以块 (Block) 为基本单位, 根据 BP 位的使能状态, 转换为十进制 level 值来设置锁定的块保护的。对于 3 个 BP 位的器件, 在 BP[0:0:0]到 BP[1:1:1]之间,

level 的取值范围为 0~7；对于 4 个 BP 位的器件，在 BP[0:0:0:0]到 BP[1:1:1:1]之间，level 的取值范围 0~10（或者 0~9，这是因为最小的锁定区域不能小于 1 Block）。

块保护锁定区域根据不同厂家不同器件而有所差异，如表 4-3 所示：

表4-3 不同厂家不同器件的块保护锁定区域与 BP Level 对应表

等级	MXIC MX25L-					ESMT F25L-
	12835F	25635F	25735F	6406E	1606E	64QA
0	解锁	解锁	解锁	解锁	解锁	解锁
1	0-64KB	0-64KB	0-64KB	0-4MB(64Block)	全锁 2MB(32Block)	0-4MB(64Block)
2	0-128KB	0-128KB	0-128KB	0-6MB(96Block)	0-1MB(16Block)	0-6MB(96Block)
3	0-256KB	0-256KB	0-256KB	0-7MB(112Block)	0-1.5MB(24Block)	0-7MB(112Block)
4	0-512KB	0-512KB	0-512KB	0-7.5MB(120Block)	0-1.75MB(28Block)	0-7.5MB(120Block)
5	0-1MB	0-1MB	0-1MB	0-7.75MB(124Block)	0-1.875MB(30Block)	0-7.75MB(124Block)
6	0-2MB	0-2MB	0-2MB	0-7.875MB(126Block)	0-1.9375MB(31Block)	0-7.875MB(126Block)
7	0-4MB	0-4MB	0-4MB	全锁 8MB (128Block)	全锁 2MB(32Block)	全锁 8MB (128Block)
8	0-8MB	0-8MB	0-8MB	-	-	-
9	全锁 16MB	0-16MB	0-16MB	-	-	-
10	-	全锁 32M	全锁 32M	-	-	-
等级	SPANSION S25FL-			WINBOND W25Q-		
	127S	256S	128FV	128BV	256FV	64FV
0	解锁	解锁	解锁	解锁	解锁	解锁
1	0-256KB	0-512KB	0-256KB	0-256KB	0-64KB	0-128KB
2	0-512KB	0-1MB	0-512KB	0-512KB	0-128KB	0-256KB

3	0-1MB	0-2MB	0-1MB	0-1MB	0-256KB	0-512KB
4	0-2MB	0-4MB	0-2MB	0-2MB	0-512KB	0-1MB
5	0-4MB	0-8MB	0-4MB	0-4MB	0-1MB	0-2MB
6	0-8MB	0-16MB	0-8MB	0-8MB	0-2MB	0-4MB
7	全锁 16MB	全锁 32MB	全锁 16MB	全锁 16MB	0-4MB	全锁 8MB
8	-	-	-	-	0-8MB	-
9	-	-	-	-	0-16MB	-
10	-	-	-	-	全锁 32MB	-
等级	GD GD25Q-			CFEON EN25Q-		
	128C	64	32	128	64	
0	解锁	解锁	解锁	解锁	解锁	
1	0-256KB	0-128KB	0-64MB	0-15.9375MB(255Block)	0-7.9375MB (127Block)	
2	0-512KB	0-256KB	0-128MB	0-15.875MB(254Block)	0-7.875MB (126Block)	
3	0-1MB	0-512KB	0-256MB	0-15.75MB(252Block)	0-7.75MB (124Block)	
4	0-2MB	0-1MB	0-512MB	15.5MB(248Block)	0-7.5MB (120Block)	
5	0-4MB	0-2MB	0-1MB	0-15MB(240Block)	0-7MB (112Block)	
6	0-8MB	0-4MB	0-2MB	0-14MB(224Block)	0-6MB (96Block)	
7	全锁 16MB	全锁 8MB	全锁 4MB	全锁 16MB(256Block)	全锁 8MB (128Block)	

根据 SPI-Nor Flash 上的块保护机制，u-boot 下新增 SPI-Nor 的块保护命令 lock。命令格式如下：



说明

SPI Nor 的 sf lock 命令需要 CONFIG_SPI_BLOCK_PROTECT，默认配置可能没有开启此开关。

- sf lock

可以查看当前设置的 BP level 值和 level 的取值范围以及当前锁定的区域范围，同时打印命令说明信息。如图 4-2 所示。

图4-2 查看当前块保护信息

```
lotus # sf lock
```



```
Get spi lock information
all blocks are locked.
Spi is locked. lock address[0 => 0x1000000]
```

```
sf lock level/all
```

Usage:

```
all: level(9), lock all blocks.
level(0): unlock all blocks.
set spi nor chip block protection level(0 - 9).
As usual: lock_len = chipsize >> (9 - level)
```

- **sf lock all**

锁定所有的块（整个器件），在表 4-3 所示中，等同于设置等级 level 为最大值，如图 4-3 所示。

图4-3 锁定整个器件

```
lotus # sf lock all
lock all blocks.
Spi is locked. lock address[0 => 0x1000000]
```

- **sf lock 0**

解除当前块保护锁定状态，此时器件上所有的块都处于未保护状态，可以任意进行擦写操作。如图 4-4 所示。

图4-4 解除当前锁定状态

```
lotus # sf lock 0
unlock all block.
```

- **sf lock <level>**

设置 BP level 值，根据 level 值，按照表 4-3 所示来保护对应的区域，这样处于块保护区域的块不可以进行正常擦写，如图 4-5 所示。

图4-5 通过设置 level 值锁定指定区域

```
lotus # sf lock 1
lock level: 1
Spi is locked. lock address[0 => 0x10000]
```

4.4 移植 U-boot 到其它构建环境

如果需要集成 U-boot 到其它构建环境中，需要将 U-boot 修改成可单独编译，以下具体说明如何修改。

4.4.1 准备工作

步骤 1 配置 Boot reg 文件

打开 SDK 中的 “configs/<soc>/reg” 目录下的配置表格，修改管脚复用，参见 4.1 如何修改 boot 表格。完成配置表格的修改后，保存表格。

步骤 2 生成表格

使用工具 tools/linux/utis/uboot_tools/regbin 将 boot 表格文件转成 bin 文件 reg_info.bin。

```
cd tools/linux/utis/uboot_tools/  
./regbin <SOC>_xxx.xlsx ./reg_info.bin
```

步骤 3 将生成的 reg_info.bin 复制到 source/bootloader/u-boot-2020.01/，并更名为.reg

```
cp ./reg_info.bin source/bootloader/u-boot-2020.01/.reg
```

步骤 4 拷贝 gzip 压缩工具并拷贝至 source/bootloader/u-boot-2020.01/tools 目录

```
cp tools/linux/utis/bin/gzip source/bootloader/u-boot-2020.01/tools/  
chmod +x source/bootloader/u-boot-2020.01/tools/gzip
```

----结束

完成以上步骤后，即可将 source/bootloader/ u-boot-2020.01 拷贝到其它构建环境相应目录中，本文后续仍以 source/bootloader/u-boot-2020.01 目录来说明。

4.4.2 编译

步骤 1 进入 u-boot-2020.01 目录

```
cd source/bootloader/u-boot-2020.01
```

步骤 2 配置编译环境

```
make ARCH=arm CROSS_COMPILE=<toolchain>- O=$(pwd)/../out <platform>_defconfig
```

步骤 3 编译 U-boot

```
make -j 16 ARCH=arm CROSS_COMPILE=<toolchain>- O=$(pwd)/../out u-boot-z.bin
```

编译成功后，会在\$(pwd)/../out 目录下生成 u-boot-<platform>.bin，将此文件烧录到单板上。

说明

编译参数说明：

1. ARCH=arm 表示编译 arm 架构的代码
2. CROSS_COMPILE 是指定工具链，注意<toolchain>后面要加横杆字符'-'
3. O=\$(pwd)/../out 指定编译输出文件的存放路径，此处为当前目录的上一级目录下的 out 目录，可以按需改为其它目录，如果此目录不存在，u-boot 会自行创建。
4. -j 16 是启动 16 个线程并行编译，可根据实际编译服务器的性能和内存资源调整。如果是在 PC 机本地建的虚拟机上编译，建减少并行编译线程数，否则可能造成编译过程中虚拟机因内存不足而卡死。

----结束