AN14093

利用Falcon模式和内核优化，实现i.MX 8M和i.MX 9的快速启动

第1.2版—2024年2月26日 **应用笔记**

文档信息

信息	内容
关键词	AN14093、Falcon模式、内核优化、U-Boot优化、快速启动、Falcon启动、i.MX93、i.MX 8M、Linux
摘要	本文档指导如何缩短i.MX 8M和i.MX 9系列产品的启动时间。

1 介绍

本文档指导用户如何缩短以下芯片的启动时间：
- i.MX 8M系列（i.MX 8M Mini LPDDR4 EVK、i.MX 8M Nano LPDDR4 EVK 和 i.MX 8M Plus LPDDR4 EVK）
- i.MX 9系列（i.MX 93 A1 LPDDR4 EVK）

本文档的目标包括：
- 引导加载程序的优化
- Linux内核与用户空间的优化
- 对所有平台上默认启动时间与优化后启动时间的比较

1.1 软件环境

假设使用的是 Ubuntu 20.04 PC。

在 i.MX 8M 系列的优化过程中使用 Linux 板级支持包（BSP）版本6.1.22_2.0.0。对于i.MX 93 A1，使用 Linux BSP 6.1.36_2.1.0 版本。

使用以下预构建映像：
- **i.MX 8M Mini：**`imx-image-full-imx8mmevk.wic`
- **i.MX 8M Nano：**`imx-image-full-imx8mnevk.wic`
- **i.MX 8M Plus：**`imx-image-full-imx8mpevk.wic`
- **i.MX 93：**`imx-image-full-imx93evk.wic`

使用以下命令将预构建映像写入SD卡：

```
$  sudo dd if=<image_name>.wic of=/dev/sd<x> bs=1M status=progress conv=fsync
```

注意：*请检查您的卡读取器分区，并将*`sd<x>`*替换为相应的分区。*

1.2 硬件设置和设备

- 恩智浦i.MX 8MM EVK LPDDR4开发套件
- 恩智浦i.MX 8MN EVK LPDDR4开发套件
- 恩智浦i.MX 8MP EVK LPDDR4开发套件
- 为11x11mm LPDDR4内存专门设计的恩智浦i.MX 93 EVK开发套件
- microSD卡：本次实验选用了容量为32GB的SanDisk Ultra micro 高容量安全数字（SDHC）I级10速的存储卡。
- micro-USB（i.MX 8M）或Type-C（i.MX 93）数据线用于调试端口

2 综述

本节概述了为缩短启动时间所需进行的常规修改。

2.1 缩短引导加载时间

您可以采取以下两种方法之一来减少引导加载时间。

- **取消启动延迟**——与默认配置相比，可节省约两秒钟，同时所需的更改最小。在启动过程中，U-Boot不再等待按键输入。

- **实施Falcon模式**——与默认配置相比，可节省约四秒钟。它使得**二级**程序加载器（SPL，U-Boot 的一部分）能够直接加载内核，跳过完整 U-Boot 的加载过程。

2.2 缩短Linux内核启动时间

- **减少控制台消息**——可节省约三秒钟。在内核命令行中添加 quiet 参数。
- **通过删除驱动程序和文件系统精简内核**——默认情况下，内核映像包含许多驱动程序和文件系统（例如：UBIFS），以启用板卡支持的大部分功能。您可以根据使用情况，裁剪包含的驱动程序和文件系统列表。

2.3 缩短用户空间启动时间

- **更改初始化 Systemd 脚本中的运行顺序**——可节省约600毫秒。尽快启动所需进程，同时考虑其依赖关系。

3 测量

测量范围涵盖了从板载POR（上电复位）到INIT过程启动的整个阶段。

具体的测量设置详见启动时间测量方法文档。

表1. 测量的间隔

时间点	脉冲间的间隔	脉冲位置	启动阶段	
Boot ROM	nRST -> before ddr_init()	board/freescale/<board>/spl.c/board_init_f()	SPL	Timeline
DDR初始化	before ddr_init() -> after ddr_init()	board/freescale/<board>/spl.c/board_init_f()		
SPL初始化+加载U-Boot映像	after ddr_init() -> before image_entry()	common/spl/spl.c/jump_to_image_no_args()		
U-Boot初始化(init_sequence_f)	before image_entry() -> start init_sequence_r	common/board_r.c/board_init_r()	U-BOOT	
U-Boot初始化(init_sequence_r)	start init_sequence_r -> u-boot main_loop	common/main.c		
启动序列	u-boot main_loop -> before load_image	include/configs/<board>.h		
内核映像加载	before load_image -> after load_image	include/configs/<board>.h		
内核启动到 INIT 的过程	after load_image -> /sbin/init	get the timestamp during kernel boot	kernel	

4 前提条件

本节介绍在独立环境中编译U-Boot和Linux内核所需的软件配置。

- **安装必要的依赖项**

根据本指南，您需要准备一系列依赖项，包括Arm64交叉编译器。

```
$ sudo apt install flex bison libssl-dev gcc-aarch64-linux-gnu u-boot-tools
 libncurses5-dev libncursesw5-dev uuid-dev gnutls-dev
```

接下来，下载所需的源文件。将它们全部放置在同一目录下。

- **下载 imx-mkimage**

mkimage是一个工具，能够将SPL、U-Boot本身、ATF以及DDR固件整合成一个映像，这个U-Boot映像随后将被烧录到SD卡中。

```
$ git clone https://github.com/nxp-imx/imx-mkimage
$ cd imx-mkimage
$ git checkout lf-6.1.22-2.0.0
```

- **下载 ATF**

```
$ git clone https://github.com/nxp-imx/imx-atf
$ cd imx-atf
$ git checkout lf-6.1.22-2.0.0
```

- **下载 U-Boot**

```
$ git clone https://github.com/nxp-imx/uboot-imx
$ cd uboot-imx
$ git checkout lf-6.1.22-2.0.0
```

- **下载 Linux内核**

```
$ git clone https://github.com/nxp-imx/linux-imx
$ cd linux-imx
$ git checkout lf-6.1.22-2.0.0
```

- **下载双倍数据速率（DDR）固件**

请参考当前Linux版本的i.MX Linux版本说明，获取正确的“firmware-imx”版本。以下是如何获取i.MX固件的示例。请根据需要替换相应的版本号。

```
$ wget https://www.nxp.com/lgfiles/NMG/MAD/YOCTO/firmware-imx-8.20.bin
$ chmod +x firmware-imx-8.20.bin
$ ./firmware-imx-8.20.bin
```

- **[仅限i.MX 93]下载EdgeLock安全区域（ELE）固件**

请参考当前Linux版本的i.MX Linux版本说明，以获取正确的ELE固件版本。以下是如何获取ELE固件的示例，请根据需要替换相应的版本号。

```
$ wget https://www.nxp.com/lgfiles/NMG/MAD/YOCTO/firmware-sentinel-0.9.bin
$ chmod +x firmware-sentinel-0.9.bin
$ ./firmware-sentinel-0.9.bin
```

5 构建映像

如果您希望检查源文件和前提条件是否已正确下载，请按以下步骤操作。否则，暂时跳过这部分，直接实施第7节“引导加载程序优化”和第8节“内核空间优化”。

5.1 构建Arm可信固件

```
$ CROSS_COMPILE=aarch64-linux-gnu- make PLAT=<plat_name> bl31
```

根据您的目标平台，<plat_name>可以是imx8mn、imx8mm、imx8mp或imx93。

构建完成后的二进制文件位于build/<plat_name>/release/目录。

5.2 构建U-Boot

1. 从ATF（build/<plat name>/release/目录）复制bl31.bin文件到imx-mkimage/<platform>/目录。
2. 从firmware-imx软件包的 firmware/ddr/synopsys/目录复制所有lpddr4*文件到imx-mkimage/<platform>/目录。
3. **[仅限i.MX 93]**将firmware-sentinel的ELE固件容器mx93a1-ahab-container.img复制到imx-mkimage/iMX9/目录。
4. 编译U-Boot。

```
$ cd uboot-imx
$ make distclean
$ ARCH=arm CROSS_COMPILE=aarch64-linux-gnu- make <defconfig_file>
$ CROSS_COMPILE=aarch64-linux-gnu- make -j $(nproc --all)
```

要构建不支持Falcon模式的U-Boot（默认启动模式，用于检查一切是否正常编译），使用以下<defconfig_file>。

- 对于 i.MX 8MM，使用 imx8mm_evk_defconfig
- 对于 i.MX 8MN，使用 imx8mn_evk_defconfig
- 对于 i.MX 8MP，使用 imx8mp_evk_defconfig
- 对于 i.MX 93，使用 imx93_11x11_evk_defconfig

如要启用Falcon模式，<defconfig_file>是（请根据第7.3节 “Falcon模式的实现”后的内容，使用这个defconfig文件）：

- 对于i.MX 8MM，使用 imx8mm_evk_falcon_defconfig
- 对于i.MX 8MN，使用 imx8mn_evk_falcon_defconfig
- 对于i.MX 8MP，使用 imx8mp_evk_falcon_defconfig
- 对于i.MX 93，使用 imx93_11x11_evk_falcon_defconfig

5. 将 u-boot*.bin 和 spl/u-boot-spl*.bin 文件复制到 imx-mkimage/<platform>/目录。
6. 从 uboot-imx/arch/arm/dts/ 目录将 imx8mm-evk.dtb（适用于i.MX 8M Mini LPDDR4 EVK）或 imx8mn-evk.dtb（适用于i.MX 8M Nano LPDDR4 EVK）或 imx8mp-evk.dtb（适用于i.MX 8M Plus LPDDR4 EVK）或 imx93-11x11-evk.dtb（适用于i.MX 93 11x11 LPDDR4 EVK）复制到 imx-mkimage/<platform>/目录。
7. 将 uboot-imx/tools/目录中的 mkimage复制到 imx-mkimage/<platform>/目录，重命名为 mkimage_uboot。

```
$ cp uboot-imx/tools/mkimage imx-mkimage/<platform>/mkimage_uboot
```

8. 生成完整的U-Boot映像：flash.bin。

```
$ cd imx-mkimage
# for i.MX 8M*
$ make SOC=<plat_name> flash_evk
# for i.MX 93
$ make SOC=iMX9 flash_singleboot
```

其中 <plat_name> 可以是 iMX8MM、iMX8MN 或 iMX8MP。

5.3 构建Linux内核

```
$ cd linux-imx
$ ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- make imx_v8_defconfig
$ ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- make -j $(nproc --all) all
```

构建完成后的二进制映像文件位于`arch/arm64/boot`目录下。

6 将二进制文件写入SD卡

为了验证构建是否正确，请将生成的二进制文件写入SD卡，并启动板卡。

- **写入U-Boot映像：**

```
$ sudo dd if=flash.bin of=/dev/sd<x> bs=1k seek=<offset> conv=fsync
```

其中 `<offset>` 的值为：

- 32 — 适用于i.MX 8M Nano、i.MX 8M Plus和i.MX 93
- 33 — 适用于i.MX 8M Mini

- **写入Linux内核：**

```
$ sudo mount /dev/sd<x>1 /mnt
$ cp Image /mnt
$ umount /mnt
```

7 引导加载程序优化

本章包含以下内容：

7.1 默认启动模式

图1描述了默认的启动顺序。在上电或重置后，i.MX 8M执行存储在其只读存储器（ROM）中的**Boot ROM**（主程序加载器）。

Boot ROM负责配置片上系统（SoC），它会进行基础外设的初始化，例如设置相位锁定环（PLL）、调整时钟配置、初始化内存（SRAM）。然后，它会找到一个启动设备，并从中加载引导加载程序映像，这个映像可能包含U-Boot SPL、ATF、U-Boot等多个组件。

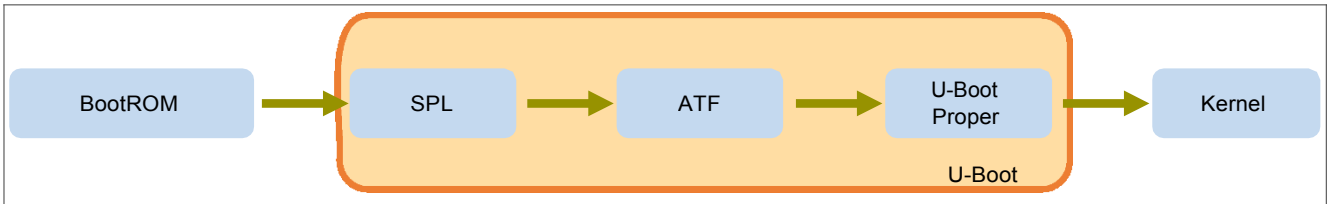

图1. 默认启动顺序

由于典型的U-Boot映像无法完全放入内部SRAM中，因此被分为两部分：**二级程序加载器（SPL）**和**U-Boot本身**。

SPL是引导加载程序的第一阶段，是一个较小的预加载器，与U-Boot共享源代码，但只包含一套能够适应SRAM的最小代码集。SPL被加载到SRAM中。它负责配置并初始化一些外设，尤其是动态随机存储器（DRAM）。之后，SPL会将ATF和U-Boot本身加载到DRAM。最终，SPL会跳转到ATF，而ATF随后会跳转到U-Boot本身。

Arm Trusted Firmware（ATF）是i.MX8*系列新近加入的组件，它为Armv8架构提供了可信赖的参考代码基础。它实现了各种Arm接口标准，包括电源状态协调接口（PSCI）。这个二进制文件通常包含在引导加载程序的二进制文件中，在U-Boot的早期阶段启动。如果没有ATF，内核无法设置必须在安全世界环境中执行的服务。

U-Boot本身是引导加载程序的第二阶段。它提供了一种灵活的方式来加载和启动Linux内核。同时，它还提供了一套简易工具，允许用户通过命令行界面与板上的硬件进行交互。U-Boot本身从DRAM运行，并初始化额外的硬件设备，如网络、USB和DSI/CSI。然后，它加载并准备设备树（FDT）。U-Boot的主要任务是加载和启动内核映像。

Linux内核从DRAM运行，并完全接管系统。从此刻起，U-Boot不再对系统有任何控制权。

7.2 Falcon模式

Falcon模式是U-Boot中的一项功能，它允许SPL启动Linux内核，实现快速启动。这种模式完全跳过了U-Boot的加载和初始化过程，显著缩短了引导加载程序所需的时间。

图2展示了Falcon模式的启动顺序。

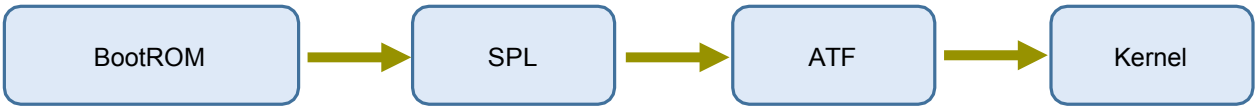

图2. Falcon模式的启动顺序

要实现这个模式，需要执行以下操作：

- 激活Falcon模式所需的一些特定配置。
- 提前准备好平面设备树（FDT）。
- 配置ATF以跳转至内核。
- 生成内核平面uImage树（FIT）映像，包含ATF和内核映像。

7.3 Falcon模式的实现

为了简化Falcon模式的实现，我们已经准备了一系列补丁来启用Falcon模式。

请下载AN14093SW.zip软件包以获取这些补丁，并按以下步骤操作：

1. **应用U-Boot补丁：**

```
$ cd uboot-imx
$ git am 0001-Enable-Fast-Boot-on-i.MX-8M-Family-and-i.MX-93.patch
```

这个补丁为每个平台（i.MX 8M和i.MX 93）创建Falcon配置文件，这些文件存放在`uboot-imx/configs/`目录下，命名为`<board> falcon_defconfig`。这些配置文件基于默认配置文件`<board>_defconfig`，并添加了如下Falcon支持。

启用的参数（设为y）

- `CONFIG_SPL_SERIAL`
- `CONFIG_CMD_SPL`——在U-Boot中启用spl export命令；这是执行第15步所必需的。
- `CONFIG_SPL_MMC`——允许SPL使用SPL MMC API从MMC读取数据。

- CONFIG_MMC_BROKEN_CD **（仅限i.MX 93）**
- CONFIG_SPL_FS_FAT——允许SPL从FAT分区读取数据。
- CONFIG_SPL_LOAD_FIT
- CONFIG_FIT
- CONFIG_SPL_OS_BOOT——激活Falcon模式。
- CONFIG_SPL_MMC_IO_VOLTAGE 和 CONFIG_SPL_MMC_UHS_SUPPORT——为SPL启用MMC高速传输功能，减少内核映像的加载时间（*对于i.MX 8MM不适用，因为由于OCRAM大小限制，不支持SPL DM）。
- CONFIG_LTO **（仅限i.MX 8MN和i.MX 93）**——通过添加链接时优化来减小二进制文件大小。在i.MX 8M Nano和i.MX 93上，为确保带有FAT支持的SPL映像适合，此项为必需。

禁用的参数（设为n）：

CONFIG_SPL_BOOTROM_SUPPORT

设置参数

CONFIG_SYS_SPL_ARGS_ADDR

其中：

- 对于i.MX 8MN、i.MX 8MM和i.MX 8MP，设置为0x43000000
- 对于i.MX 93，设置为0x83000000

CONFIG_SPL_FS_LOAD_PAYLOAD_NAME设置为u-boot.itb

CONFIG_SPL_FS_LOAD_KERNEL_NAME设置为Image.itb

CONFIG_SPL_FS_LOAD_ARGS_NAME设置为：

- 对于i.MX 8MM，为imx8mm-evk-falcon.dtb
- 对于i.MX 8MN，为imx8mn-evk-falcon.dtb
- 对于i.MX 8MP，为imx8mp-evk-falcon.dtb
- 对于i.MX 93，为imx93-11x11-evk-falcon.dtb

CONFIG_CMD_SPL_WRITE_SIZE设置为0xC000

CONFIG_FIT_EXTERNAL_OFFSET=0x3000 **（仅限i.MX 93）**

此外，补丁实现了spl_start_uboot()函数，位于uboot-imx/board/freescale/<board>/spl.c，其中<board>为imx8mm_evk、imx8mn_evk、imx8mp_evk或imx93_evk。此函数负责检查SPL是应该启动内核还是U-Boot。如果在启动过程中按下了'c'键，则函数返回1，这意味着必须启动U-Boot。否则，SPL应该启动内核。为了在操作状态中使以太网MAC能与PHY交互，对于i.MX 8M系列，必须从SPL重置PHY。这一步骤也包含在应用U-Boot补丁中。PHY的重置操作在board_init_r()函数中完成，该函数位于uboot-imx/common/spl/spl.c文件中。

2. **应用ATF补丁：**

```
$ cd imx-atf
$ git am 0001-Add-support-to-jump-to-kernel-directly-from-ATF.patch
```

ATF补丁增加了对直接跳转到内核的支持。由于ATF不支持在恩智浦平台上直接跳转到内核，因此必须在bl31_early_platform_setup2()函数中传递FDT地址，该函数位于imx-atf/plat/imx/imx8m/<board>/<board>_bl31_setup.c路径下，适用于i.MX 8M系列，以及imx-atf/plat/imx/imx93/imx93_bl31_setup.c路径下，适用于i.MX93。

3. **应用mkimage补丁：**

```
$ cd imx-mkimage
$ git am 0001-Add-scripts-for-Fast-Boot-implementation-for-i.MX 8M-.patch
```

mkimage补丁为U-Boot FIT映像源（u-boot.its）的uboot-1节点添加了“os”属性。当启用Falcon模式时，如果spl_start_uboot()返回1，则加载U-Boot时需要这一属性。否则，U-Boot无法启动。

此外，补丁还添加了脚本来为i.MX 93生成U-Boot FIT映像（因为在此版本中不存在）：`imx-mkimage/iMX9/mkimage_fit_atf.sh`。
此补丁添加的第二个脚本是用于生成内核FIT映像（ATF+内核）的脚本——这是实现Falcon模式所需的。此脚本适用于i.MX 8M系列和i.MX 93。

4. 按照第5.1节的说明构建ATF。将修改后的ATF二进制文件复制到`imx-mkimage/<platform>/`目录。
5. 按照第5.2节-Falcon模式的说明构建引导加载程序映像。根据第6节的说明写入生成的U-Boot二进制文件。
6. **[仅限i.MX93]**生成U-Boot FIT映像。构建`flash.bin`映像时，i.MX93的`u-boot.itb FIT`映像不会自动构建，因为没有生成它的脚本。

```
$ cd imx-mkimage/iMX9
$ DEK_BLOB_LOAD_ADDR=0x80400000 TEE_LOAD_ADDR=0x96000000
ATF_LOAD_ADDR=0x204e0000 ./mkimage_fit_atf.sh imx93-11x11-evk.dtb > u-boot.its
$ ./mkimage_uboot -E -p 0x3000 -f u-boot.its u-boot.itb
```

7. 将位于`imx-mkimage/<platform>`目录下的`u-boot.itb`二进制文件复制到SD卡的第一个（FAT）分区。
8. 在构建Linux内核之前，您可能希望根据第8.2节“删除不必要的驱动程序和文件系统”进行优化。根据第5.3节“构建Linux内核”构建Linux内核。
9. 生成内核FIT映像。
   FIT映像包含ATF和内核映像。SPL在Falcon模式期间加载此映像。由于SPL在Falcon模式期间不加载ATF映像，因此必须将ATF包含在FIT映像中。
   为了准备FIT映像（`Image.itb`），使用`mkimage_fit_atf_kernel.sh`脚本。将内核映像复制到`imx-mkimage/<platform>/`目录：

```
$ cp linux-imx/arch/arm64/boot/Image imx-mkimage/<platform>
```

   生成FIT映像：

   - 对于i.MX 8M

```
$ cd imx-mkimage/iMX8M
# for i.MX8MM
$ ATF_LOAD_ADDR=0x00920000 KERNEL_LOAD_ADDR=0x40200000 ../mkimage_fit_atf_kernel.sh
 > Image.its
# for i.MX8MN
$ ATF_LOAD_ADDR=0x00960000 KERNEL_LOAD_ADDR=0x40200000 ../mkimage_fit_atf_kernel.sh
 > Image.its
# for i.MX8MP
$ ATF_LOAD_ADDR=0x00970000 KERNEL_LOAD_ADDR=0x40200000 ../mkimage_fit_atf_kernel.sh
 > Image.its

# To generate the FIT binary run:
$ ./mkimage_uboot -E -p 0x3000 -f Image.its Image.itb
```

   - 对于i.MX93

```
$ cd imx-mkimage/iMX9
$ ATF_LOAD_ADDR=0x204e0000 KERNEL_LOAD_ADDR=0x80200000 ../
mkimage_fit_atf_kernel.sh > Image.its

# To generate the FIT binary run:
$ ./mkimage_uboot -E -p 0x3000 -f Image.its Image.itb
```

10. 将生成的`Image.itb`文件复制到SD卡的第一个（FAT）分区。
11. 准备扁平化设备树并将其写入SD卡。
    在Falcon模式启动时，准备设备树是一个关键步骤。通常，U-Boot在启动Linux时会进行FDT修正。这意味着U-Boot会向初始设备树添加内核参数和内存节点等修改。

这些参数可以在配置文件之一uboot-imx/configs/<board>_evk.h中找到，名称为bootarg。它们指定控制台参数，并告诉内核在哪里找到根文件系统。其中<board>为imx8mm、imx8mn、imx8mp或imx93。

有两种生成扁平化设备树的方法：

- **方法1**：手动向设备树添加所需的修正
- **方法2**：让U-Boot进行修正并保存生成的设备树

方法2更通用，需要的知识较少，但也更长且有几个其他步骤。

方法1：您可以尝试手动生成FDT，具体操作是在内核设备树中添加bootarg和内存节点。例如，对于i.MX 93，可以在linux-imx/arch/arm64/boot/dts/freescale目录下创建imx93-11x11-evk-falcon.dts文件，并添加以下代码行。内存节点是从U-Boot的DTS复制的。您可以根据使用情况，更改包含的设备树。在这种情况下，我们使用的是默认的内核设备树。

```
#include "imx93-11x11-evk.dts"

 / {
     memory {
         reg = <0x00 0x80000000 0x00 0x80000000>;
         device_type = "memory";
     };

     chosen {
         bootargs = "console=ttyLP0,115200 earlycon root=/dev/mmcblk1p2
 rootwait rw";
     };
 };
```

***注意：**在Falcon模式中，使用的内核设备树的名称必须与CONFIG_SPL_FS_LOAD_ARGS_NAME变量中设置的名称一致。*

要编译此DTS，必须将文件添加到同一目录下的Makefile中：

```
dtb-$(CONFIG_ARCH_MXC) += imx93-11x11-evk.dtb \ imx93-11x11-evk-falcon.dtb \
...
```

重新编译内核，以生成关联的设备树二进制文件，并将生成的imx93-11x11-evk-falcon.dtb文件复制到SD卡的第一个分区。

如果您选择了第一种方法，接下来只需启动板卡，Falcon模式便可顺利运行。

方法2：FDT可以通过在U-Boot阶段使用spl export命令来准备。要进入U-Boot，持续按下**C**键。这个命令的作用等同于运行bootm命令，直至设备树修正完毕。存储在内存中的设备树，就是Falcon模式所需的。这个映像必须保存到SD卡的启动分区。

前提条件：

a. 从Image构建内核传统的uImage文件。
   uImage是一种特殊的图像文件，在实际的内核Image之前添加了一个64字节的标头，其中指定了加载器信息（加载地址、入口点、操作系统类型等）。
   这种类型的图像是spl命令生成扁平化设备树所需的。

   - **对于i.MX 8M**

```
$ cd linux-imx/arch/arm64/boot
$ mkimage -A arm -O linux -T kernel -C none -a 0x43FFFFC0
-e 0x44000000 -n "Linux kernel" -d Image uImage
Image Name: Linux kernel
Created: Wed Jul 26 14:12:09 2023
Image Type: ARM Linux kernel Image (uncompressed)
Data Size: 31072768 Bytes = 30344.50 KiB = 29.63 MiB
Load Address: 43ffffc0
```

```
Entry Point: 44000000
```

- **对于i.MX 93**

```
$ cd linux-imx/arch/arm64/boot
$ mkimage -A arm -O linux -T kernel -C none -a 0x83FFFFC0
-e 0x84000000 -n "Linux kernel" -d Image uImage
Image Name: Linux kernel
Created: Wed Jul 26 14:14:09 2023
Image Type: ARM Linux kernel Image (uncompressed)
Data Size: 31072768 Bytes = 30344.50 KiB = 29.63 MiB
Load Address: 83ffffc0
Entry Point: 84000000
```

其中：
- A [架构]：设置架构。
- O [操作系统]：设置操作系统。
- T [图像类型]：设置图像类型。
- C [压缩类型]：设置压缩类型。
- n [图像名称]：为**图像指定名称**。
- d [图像数据文件]：使用**图像数据文件**中的图像数据。
- a [加载地址]：设置加载地址，使用十六进制数表示。
- e [入口点]：设置入口点，使用十六进制数表示。

b. 将内核的uImage文件复制到SD卡的EXT2分区。

```
$ sudo mount /dev/sd<x>2 /mnt
$ sudo mkdir -p /mnt/home/root/.falcon
$ sudo cp uImage /mnt/home/root/.falcon
$ sudo umount /mnt
```

要使用`spl export`命令来准备FDT，请按以下步骤操作：

i. 启动板卡进入U-Boot，并在自动启动顺序开始前停止。要进入U-Boot，必须在启动时按下'c'键。此时，由于SD卡上没有为Linux内核准备好的FDT，Falcon模式无法启动。

ii. 【**可选**】如果您需要一个与默认FDT不同的文件，请首先运行以下命令。该文件必须位于SD卡的FAT分区上。

```
u-boot=> setenv fdtfile <file_name>.dtb
```

iii. 将FDT加载到RAM中。

```
u-boot=> run loadfdt
43801 bytes read in 15 ms (2.8 MiB/s)
```

iv. 将内核的uImage文件加载到RAM中。

```
u-boot=> ext2load mmc 1:2 ${loadaddr} /home/root/.falcon/uImage
31072832 bytes read in 387 ms (76.6 MiB/s)
```

v. 如果您还希望优化内核的启动时间，请在进行下一步之前，根据第8.1节 "添加quiet"中的第2步运行相应命令。

vi. 加载内核启动所需的参数。

```
u-boot=> run mmcargs
```

vii. 准备FDT（以i.MX 93为例）。

```
u-boot=> spl export fdt ${loadaddr} - ${fdt_addr_r}
## Booting kernel from Legacy Image at 80400000 ...
   Image Name:   Linux kernel
```

```
   Created:      2023-07-19   6:57:40 UTC
   Image Type:   ARM Linux kernel Image (uncompressed)
   Data Size:    31072768 Bytes = 29.6 MiB
   Load Address: 83ffffc0
   Entry Point:  84000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 83000000
   Booting using the fdt blob at 0x83000000
Working FDT set to 83000000
   Loading kernel Image
   Using Device Tree in place at 0000000083000000, end 000000008300db18
Working FDT set to 83000000
subcommand failed (err=-1)
subcommand failed (err=-1)
   Using Device Tree in place at 0000000083000000, end 0000000083010b18
Working FDT set to 83000000
Argument image is now in RAM: 0x0000000083000000
```

***注意：**上文**加粗**显示的起始地址与结束地址之间的差值就是内存中修补后FDT的大小。将处理后FDT从RAM复制到SD卡的FAT分区，指定正确的复制大小作为最后一个参数。在上面的示例输出中，这将是 `0x83010b18 - 0x83000000 = 0x10b18`。*

```
# for i.MX 93
u-boot=> fatwrite mmc ${mmcdev}:${mmcpart} ${fdt_addr_r}   imx93-11x11-
evk-falcon.dtb 0x10b18
```

***注意：**存储的FDT文件名必须与第7.3节“Falcon模式的实现”中第1步设置的CONFIG_SPL_FS_LOAD_ARGS_NAME变量中的名称一致。否则，SPL不会将设备树加载到DRAM中，板卡无法启动。*

12. 重启后，默认情况下，板卡将以Falcon模式启动。

7.4 内存映射

图3展示了i.MX93在Falcon模式期间的内存映射。

Boot ROM加载SPL，SPL在片上RAM（OCRAM - 内部处理器内存）中运行。SPL负责初始化动态RAM（DDR），将ATF加载到OCRAM，然后将内核设备树和内核映像加载到DDR。SPL在DDR中预留了一块内存空间用于动态内存分配（malloc），这部分内存在SPL运行期间是不允许被覆盖的。

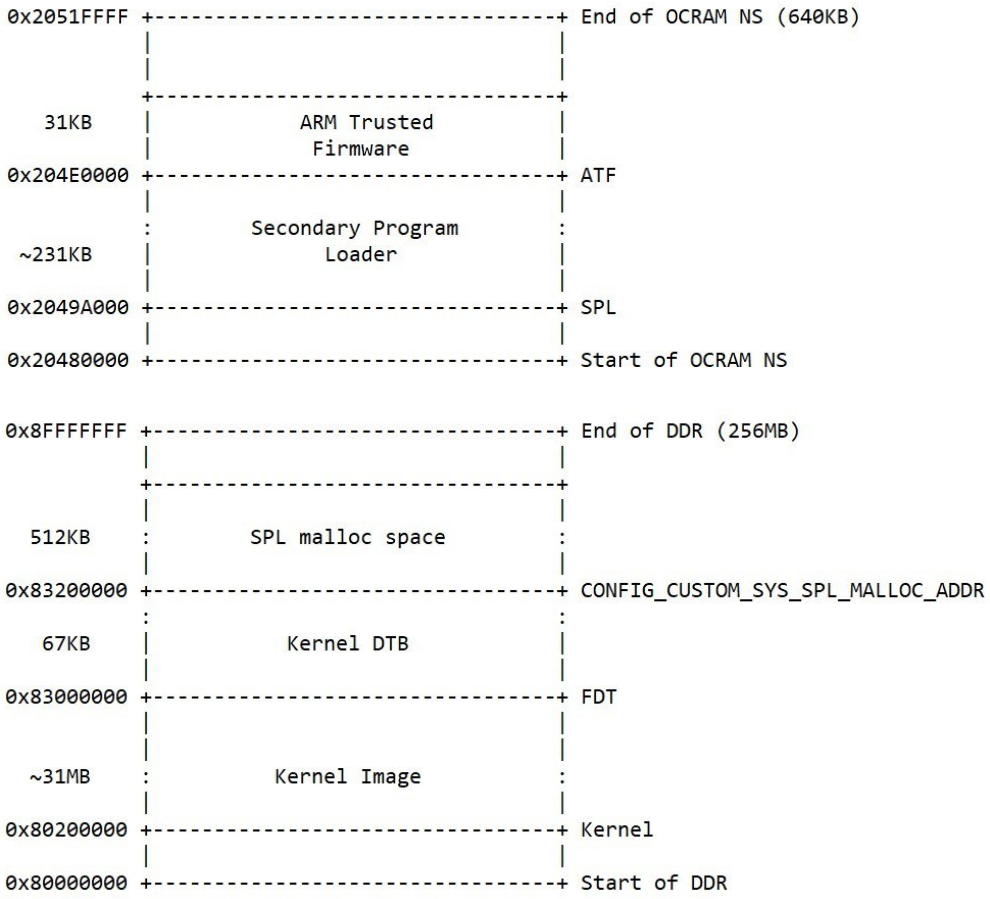

图3. i.MX93在Falcon模式期间的内存映射

表2列出了i.MX 8M系列的地址。

表2. i.MX 8M系列的地址

平台	SPL	ATF	内核映像	内核DTB
i.MX 8M Mini	0x007e1000	0x00920000	0x40200000	0x43000000
i.MX 8M Nano	0x00912000	0x00960000	0x40200000	0x43000000
i.MX 8M Plus	0x00920000	0x00970000	0x40200000	0x43000000

7.5 Falcon模式期间的函数调用

图4列出了在SPL Falcon模式期间调用的重要函数。

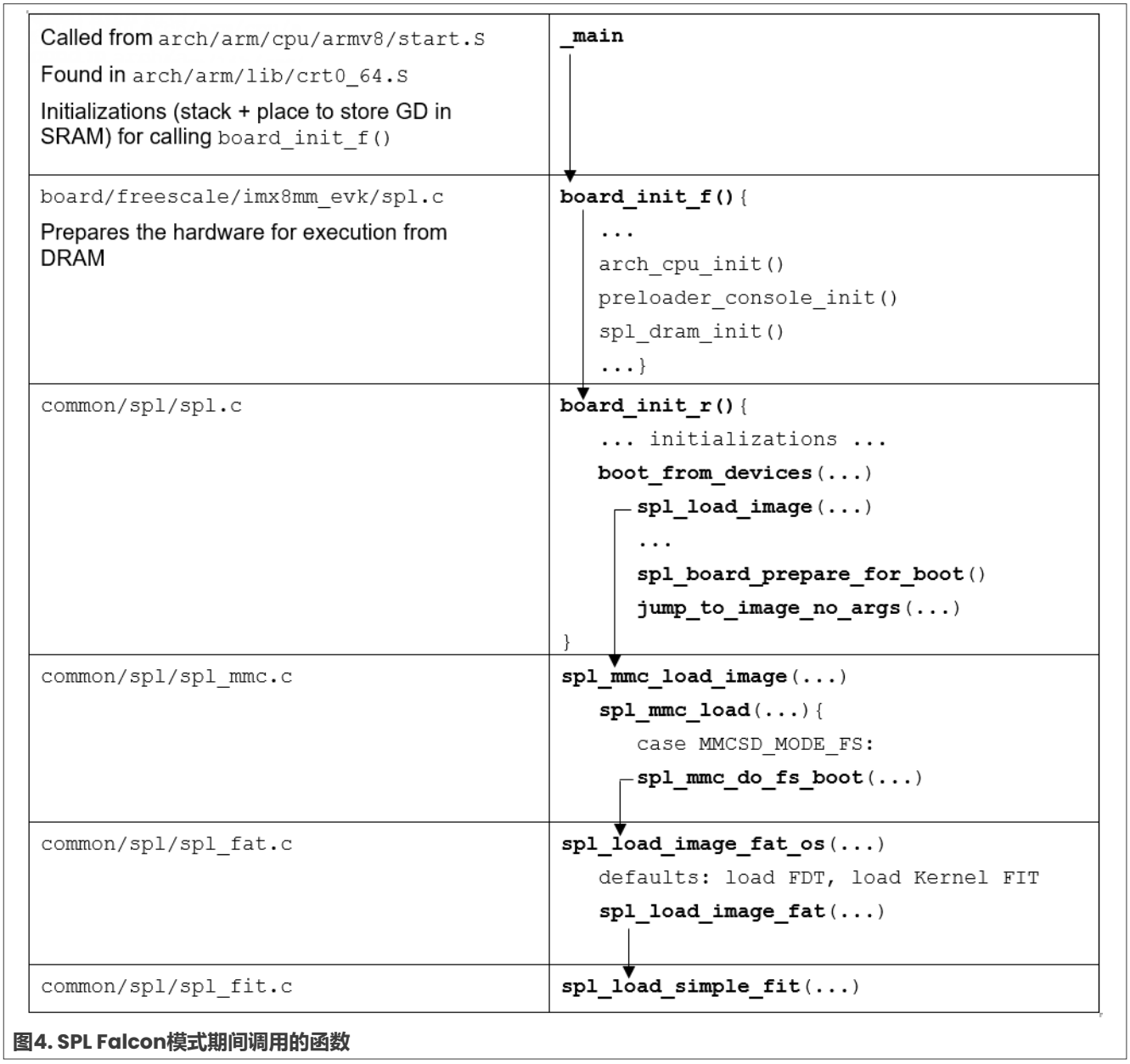

图4. SPL Falcon模式期间调用的函数

8 内核空间优化

本节列出了第8.1节“添加quiet”和第8.2节“删除不必要的驱动程序和文件系统”的步骤。

8.1 添加quiet

为了将内核启动时间缩短大约一半，建议在内核bootargs中加入quiet参数。这样做可以在Linux启动过程中抑制调试消息的输出。

注意：*需要使用spl export命令，重新生成包含新bootargs的设备树。*

1. 要进入U-Boot，启动时按住**C**键。

2. 添加quiet，编辑mmcargs参数。

```
u-boot=> edit mmcargs
edit: setenv bootargs ${jh_clk} console=${console} root=${mmcroot} quiet
u-boot=> saveenv
Saving Environment to MMC... Writing to MMC(1)... OK
```

3. 重新生成设备树并保存至SD卡，参见第7.3节 “Falcon模式的实现” 的第14步。

8.2 删除不必要的驱动程序和文件系统

根据您的具体需求，您可以通过删除不必要的驱动程序和文件系统来简化内核。利用bootgraph，您可以在启动时分析内核功能，这一内核特性能够帮助您图形化展示内核初始化过程中的各项活动。

要创建bootgraph，请按以下步骤操作：

1. 在内核bootargs中添加initcall_debug。
   a. 要进入U-Boot，启动时按住**C**键。
   b. initcall_debug.添加initcall_debug，编辑mmcargs参数。

```
u-boot=> edit mmcargs
edit: setenv bootargs ${jh_clk} console=${console} root=${mmcroot} quiet
 initcall_debug
u-boot=> saveenv
Saving Environment to MMC... Writing to MMC(1)... OK
```

2. 重新生成设备树并保存至SD卡，参见第7.3节 “Falcon模式的实现” 第11步。
3. 启动板卡并获取内核日志。

```
root@imx8mn-lpddr4-evk:~# dmesg > boot.log
```

boot.log文件记录类似以下日志的信息，可以通过这些数据来分析内核启动过程中每个函数的耗时。

```
[2.583922] initcall deferred_probe_initcall+0x0/0xb8 returned 0 after 895357
[2.583955] calling  genpd_power_off_unused+0x0/0x98 @ 1
[2.583977] initcall genpd_power_off_unused+0x0/0x98 returned 0 after 12 usec
[2.583984] calling  genpd_debug_init+0x0/0x90 @ 1
[2.584312] initcall genpd_debug_init+0x0/0x90 returned 0 after 321 usecs
[2.584333] calling  ubi_init+0x0/0x23c @ 1
[2.584627] initcall ubi_init+0x0/0x23c returned 0 after 286 usecs
```

4. 将生成的boot.log文件复制到主机PC上。回到主机PC，使用以下命令创建图表。

```
$ cd linux-imx/scripts
$ ./bootgraph.pl boot.log > boot.svg
```

您可以得到类似的结果，并分析内核启动时间的使用情况：

图5. 内核启动时间

5. 如果需要禁用某个驱动或功能，请更新内核配置。
   例如，我们禁用了内核的调试功能（这样可以减小映像大小）和 UBI 文件系统。
   a. 运行以下命令进入内核 `menuconfig`。

```
$ cd linux-imx
$ ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- make imx_v8_defconfig
$ ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- make menuconfig
```

   在 `menuconfig` 中，禁用 `CONFIG_UBIFS_FS` 和 `CONFIG_DEBUG_KERNEL`，操作类似于第 7.3 节 “Falcon 模式的实现”。生成的 `.config` 文件包含以下内容。

```
# CONFIG_UBIFS_FS is not set
# CONFIG_DEBUG_KERNEL is not set
```

   b. 构建新的内核映像。

```
$ ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- make all
```

   c. 重新生成内核FIT映像并将其复制到SD卡启动（FAT）分区，参见第7.3节 “Falcon模式的实现”第9步。
   d. *【可选】*如果您想在正常启动时使用修改后的内核，请将新的Image二进制文件复制到第一个SD卡启动分区。

9 用户空间优化

本节列出了“在systemd启动前运行应用程序”章节和“调整systemd单元的依赖关系”章节的步骤。

9.1 在systemd启动前运行应用程序

如果需要，可以在`systemd`启动之前运行一个程序。

1. 创建脚本`/home/root/newinit.sh`，通过该脚本可以在`systemd`启动前运行您的程序。以下是一个简单的示例，展示如何在启动`systemd`之前运行程序。请将`echo`行替换为您想要运行的应用程序。

```
#!/bin/sh
echo “Early start” > /dev/kmsg
exec /lib/systemd/systemd
```

2. 将此脚本设置为可执行状态。

```
$ chmod +x newinit.sh
```

3. 将`/sbin/init`的链接指向您的`newinit.sh`脚本。

```
$ ln -sf /home/root/newinit.sh /sbin/init
```

注意：*如需恢复至初始配置，请使用以下命令。*

```
$ ln -sf /lib/systemd/systemd /sbin/init
```

4. 重启板卡并查看内核日志。在dmesg中搜索"Early start"字符串，可以看到newinit.sh脚本已在init进程之前执行。

9.2 调整systemd单元的依赖关系

要缩短用户空间的启动时间，最简单的方法是调整应用程序运行的顺序。如需提前启动服务，需要更改Systemd操作的依赖关系。

在板卡上，打开/lib/systemd/system/<service_name>.service文件并调整单元依赖关系。例如，可以将<service_name>服务的启动顺序调整至local-fs-pre.target之前。

```
[Unit]
...
Before=local-fs-pre.target
DefaultDependencies=no
```

如果使用blame参数调用system-analyze命令，Systemd还提供一个名为systemd-analyze的工具，它会显示各个服务及其启动时间。

```
$ systemd-analyze blame
```

如要禁用某个服务，您可以使用systemctl disable命令。要禁用某些服务（特别是systemd提供的服务），可以使用systemctl mask命令。但请注意，在禁用服务时需谨慎，因为系统可能依赖这些服务才能正常运行。

10 结果

表3. 初始启动时间测量

	SPL			U-Boot				内核	
板卡	Boot ROM	DDR初始化	SPL初始化 + 加载U-Boot映像	U-Boot初始化 (init_ sequence_ f)	U-Boot 初始化 (init_ sequence_ r)	启动序列	内核映像加载	ATF+内核启动到INIT的过程	总时间
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)
i.MX 8MN	161	241	162	363	790	2894	333	3506	8450
i.MX 8MP	162	301	175	373	1726	4181	345	3627	10890
i.MX 8MM	142	265	117	412	812	2970	396	5002	10116
i.MX 93	369	111	117	628	1172	3271	412	3090	9170

表4. 优化后启动时间测量数据

	SPL				内核	
板卡	Boot ROM	DDR初始化	SPL初始化	内核映像加载[1]	ATF+内核启动到INIT的过程[2]	总时间
	(ms)	(ms)	(ms)	(ms)	(ms)	(ms)
i.MX 8MN	203	240	86	376	1185	2090
i.MX 8MP	187	301	97	382	1237	2204
i.MX 8MM[3]	139	265	63	1336	2956[3]	4759
i.MX 93	374	111	89	366	1391	2330

[1] CONFIG_DEBUG_KERNEL禁用，从而减小了内核映像的大小，这也相应减少了内核映像的加载时间。

[2] 使用quiet参数抑制内核日志消息。

[3] i.MX 8M Mini EVK并未像i.MX 8M Plus那样内置连接至PCIe端口的Wi-Fi模块。因此，在PCIe PHY初始化过程中，系统会等待活动链接，这一过程会占用额外时间。如果在PCIe接口上连接了Wi-Fi模块，内核的启动时间可以缩短至1215 ms，总体启动时间则减少至3018 ms。

11 关于本文中源代码的说明

本文中所示的示例代码具有以下版权和BSD-3-Clause许可：

2024年恩智浦版权所有；在满足以下条件的情况下，可以源代码和二进制文件的形式重新分发和使用本源代码（无论是否经过修改）：

1. 重新分发源代码必须保留上述版权声明、这些条件和以下免责声明。
2. 以二进制文件形式重新分发时，必须在文档和/或随分发提供的其他材料中复制上述版权声明、这些条件和以下免责声明。
3. 未经事先书面许可，不得使用版权所有者的姓名或参与者的姓名为本软件的衍生产品进行背书或推广。

本软件由版权所有者和参与者“按原样”提供，不承担任何明示或暗示的担保责任，包括但不限于对适销性和特定用途适用性的暗示保证。在任何情况下，无论因何种原因或根据何种法律条例，版权所有者或参与者均不对因使用本软件而导致的任何直接、间接、偶然、特殊、惩戒性或后果性损害（包括但不限于采购替代商品或服务；使用损失、数据损失或利润损失或业务中断）承担责任，无论是因合同、严格责任还是侵权行为（包括疏忽或其他原因）造成的，即使事先被告知有此类损害的可能性也不例外。

12 修订历史

表5. 修订历史

版本号	发布日期	说明
1.2	2024年2月26日	源代码中标注的版权日期。
1.1	2024年1月24日	添加了对i.MX 93 A1的支持。
1	2023年10月9日	首次发布。

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com.cn/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

目录

For more information, please visit: https://www.nxp.com.cn

Date of release: 26 February 2024
Document identifier: AN14093