Application Note

在 TI J721S2 上从 OSPI NOR 闪存无 eMMC 引导 Linux

Johnny Kim

摘要

本应用手册介绍了如何在不依赖 eMMC 存储的情况下，完全从 Texas Instruments J721S2 平台上的 OSPI NOR 闪存引导 Linux。

演示的引导流程将所有引导元件存储在 OSPI NOR 闪存中，包括 tiboot3.bin、tispl.bin、U-Boot、Linux 内核映像、器件树二进制文件 (DTB) 和根文件系统。基于 UBI/UBIFS 的根文件系统在 OSPI 闪存内创建，并在系统启动期间用作 Linux 根文件系统。该实现已使用 Processor SDK Linux 在 J721S2 EVM 上进行验证。本应用手册介绍了 OSPI 闪存分区、UBI 和 UBIFS 卷创建、最小根文件系统的构建、UBIFS 内 Linux 工件的存储、U-Boot 配置，以及从 OSPI NOR 闪存完成 Linux 引导。最终的解决方案展示了无需 eMMC 存储的功能齐全的 Linux 引导链，为需要简化存储配置或基于 OSPI 的部署的系统提供了替代引导架构。

内容

商标

所有商标均为其各自所有者的财产。

1 简介

1.1 动机

德州仪器 (TI) J721S2 器件支持多种引导介质，包括 SD 卡、eMMC、OSPI NOR 闪存等。在许多生产系统中，OSPI NOR 闪存通常用于存储早期引导元件，例如 tiboot3.bin、tispl.bin 和 U-Boot。但是，对于基于 Linux 的系统，内核映像、设备树 blob (DTB) 和根文件系统由于具有较大的存储容量，通常存储在 eMMC 或其他大容量存储设备上。这导致引导体系结构依赖于多个存储设备。在某些应用中，需要消除对 eMMC 的依赖，并将所有引导元件合并到单个 OSPI NOR 闪存器件中。这种方法简化了存储管理、降低了系统复杂性，并支持在 eMMC 不可用的系统上进行部署。本应用手册演示了一个 Linux 引导解决方案，在该解决方案中，所有引导元件（包括引导加载程序、内核映像、DTB 和根文件系统）存储在 OSPI NOR 闪存中。根文件系统是使用 UBI 和 UBIFS 实现的，以提供强大的闪存感知存储设计。

1.2 范围

本应用手册的目标是演示和验证在不依赖 eMMC 存储的情况下从 J721S2 平台上的 OSPI NOR 闪存进行 Linux 引导。

讨论了以下各项：

- 在 OSPI NOR 闪存中创建 UBI/UBIFS 根文件系统
- 构建和验证最小 Linux 根文件系统
- 将 Linux 内核映像和 DTB 存储在 UBIFS 中
- 使用 U-Boot 从 UBIFS 加载内核映像和 DTB
- 使用 UBIFS 根文件系统引导 Linux
- 从 OSPI NOR 闪存引导整个系统

本文档中描述的过程已使用 Processor SDK Linux 在 J721S2 EVM 上进行了验证。

1.3 目标引导架构

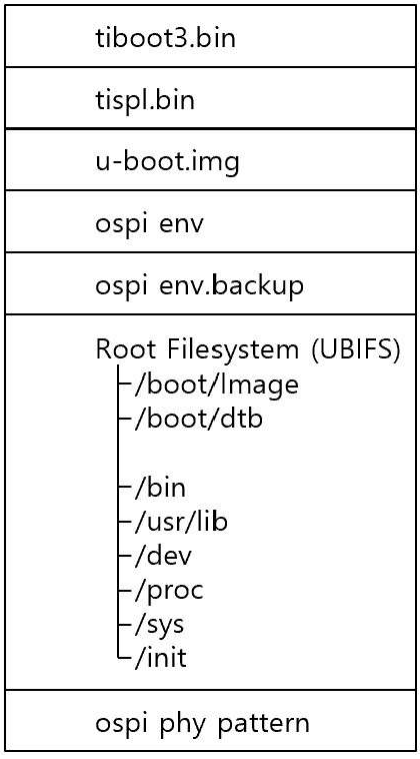

图 1-1. OSPI NOR 闪存布局

Linux 引导所需的所有软件元件都存储在 OSPI NOR 闪存中。U-Boot 从位于 OSPI 闪存中的 UBIFS 卷加载 Linux 内核映像和 DTB，Linux 随后装载与根文件系统相同的 UBIFS 卷。因此，无需 eMMC 存储即可执行完整的引导过程。

2 背景

2.1 J721S2 启动流程

J721S2 器件使用多级引导进程来初始化系统并加载 Linux 操作系统。

该器件上电时，首先执行片上 ROM 引导加载程序。根据配置的引导模式，ROM 从所选引导介质加载 tiboot3.bin。tiboot3.bin 映像会初始化基本器件资源并加载 tispl.bin。

tispl.bin 映像执行额外的系统初始化，包括 DDR 配置和加载系统固件元件。初始化完成后，tispl.bin 将加载并启动 U-Boot。

U-Boot 负责加载 Linux 内核映像和设备树 blob (DTB)、准备 Linux 引导参数，并将控制权转移到 Linux 内核。

本应用手册中实现的引导序列如下所示。

```
ROM → tiboot3.bin → tispl.bin → U-Boot → Linux Kernel → Linux User Space
```

这项工作的目标是将该引导链的所有元件存储在 OSPI NOR 闪存中，并消除对 eMMC 存储的任何依赖。

2.2 OSPI NOR 闪存概述

OSPI NOR 闪存通常用于 J721S2 平台上存储引导加载程序元件，因为它支持 ROM 引导加载程序直接访问并提供可靠的非易失性存储。

与 eMMC 器件相比，OSPI NOR 闪存通常提供更低的存储容量，但为嵌入式系统提供了更简单的存储架构。如果可用容量足够，单个 OSPI 器件可以包含引导加载程序映像、Linux 引导工件和根文件系统。

在此实现中，使用了 64MB OSPI NOR 闪存器件。对闪存进行分区以存储以下各项。

- tiboot3.bin
- tispl.bin
- u-boot.img
- U-Boot 环境
- UBI/UBIFS 根文件系统卷

Linux 内核映像、DTB 和根文件系统都存储在 UBI/UBIFS 分区中。

2.3 UBI 和 UBIFS 概述

NOR 和 NAND 闪存等原始闪存设备与 eMMC 和 SSD 等块存储设备大不相同。专为块设备（如 ext4）设计的 Linux 文件系统不能直接在原始闪存存储器上运行。

为了支持在原始闪存器件上进行可靠存储，Linux 提供了 UBI（未排序的块映像）层和 UBIFS（UBI 文件系统）。

UBI 位于 MTD 子系统和文件系统之间。它管理闪存特定的要求，例如逻辑擦除块映射、磨损均衡和坏块处理。UBIFS 是在 UBI 卷之上运行的闪存感知文件系统。

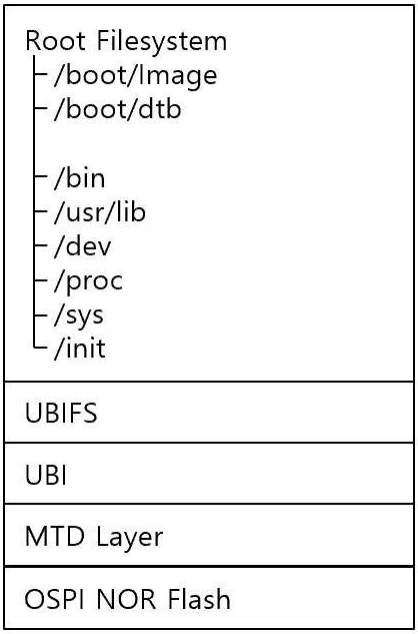

图 2-1. UBI/UBIFS 软件栈

使用 UBI 和 UBIFS 使 Linux 能够在 OSPI NOR 闪存上可靠地存储和访问文件，同时为 U-Boot 和 Linux 提供标准文件系统接口。

在本应用手册中，在 OSPI 根文件系统分区中创建了一个名为 rootfs 的 UBI 卷。此卷存储 Linux 内核映像、DTB 和 Linux 根文件系统。

3 OSPI 闪存布局

本章介绍了本应用手册中使用的 OSPI NOR 闪存分区布局。布局为引导元件、U-Boot 环境存储和基于 UBI/UBIFS 的 Linux 文件系统分配专用区域。分区组织使 Linux 引导所需的所有软件元件能够驻留在单个 OSPI NOR 闪存器件中。

3.1 分区布局

本应用手册中使用的 J721S2 EVM 包含 64MB OSPI NOR 闪存器件。对闪存进行分区以存储引导元件、U-Boot 环境数据，以及使用 UBI 和 UBIFS 实现的 Linux 根文件系统。

表 3-1. 用于实现的分区布局摘要

分区	尺寸	用途
ospi.tiboot3	512KB	次级引导加载程序
ospi.tispl	2MB	A72 SPL
ospi.u-boot	4MB	U-Boot 映像
ospi.env	256KB	U-Boot 环境
ospi.env.backup	256KB	U-Boot 备份环境
ospi.rootfs	55.8MB	UBI/UBIFS 文件系统
ospi.phypattern	256KB	PHY 模式存储

最大的分区 ospi.rootfs 用于存储 Linux 内核映像、设备树 blob (DTB) 和 Linux 根文件系统。分区格式化为 UBI 设备，并包含名为 rootfs 的 UBIFS 卷。

3.2 引导加载程序分区

OSPI NOR 闪存的第一部分存储初始化系统和启动 Linux 所需的软件元件。在上电期间，J721S2 ROM 代码从 ospi.tiboot3 分区加载 tiboot3.bin。tiboot3.bin 映像执行早期平台初始化，并从 ospi.tispl 分区加载 tispl.bin。tispl.bin 映像将初始化 DDR 存储器、加载系统固件，并从 ospi.u-boot 分区启动 U-Boot。然后，U-Boot 从位于 ospi.rootfs 分区中的 UBIFS 文件系统加载 Linux 内核映像和 DTB。ospi.env 和 ospi.env.backup 分区存储用于控制引导进程的 U-Boot 环境变量。备份分区可在环境损坏时提供冗余。

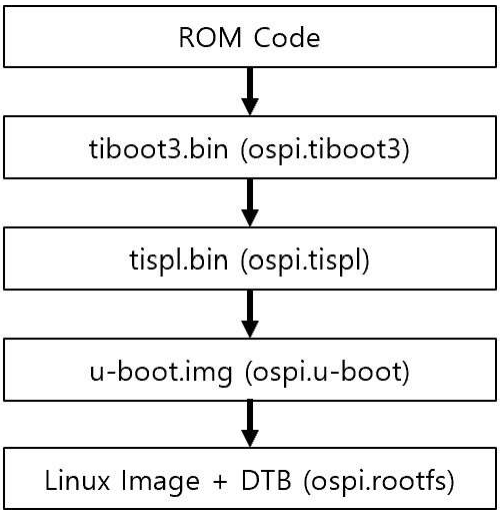

图 3-1. 引导元件流程

引导元件分区仅占用总闪存容量的一小部分，大部分 OSPI NOR 闪存可用于 Linux 存储。

3.3 根文件系统分区

ospi.rootfs 分区用于存储系统引导所需的所有 Linux 相关软件元件。

与在 eMMC 上存储内核映像、DTB 和根文件系统的传统 J721S2 Linux 系统不同，该实现将所有 Linux 工件存储在位于 OSPI NOR 闪存中的 UBIFS 卷内。

图 3-2. 根文件系统结构

分区格式化为 UBI 设备，并包含名为 rootfs 的 UBIFS 卷。卷存储以下项目。

- Linux 内核映像 (映像)
- 设备树 blob (*.dtb)
- Linux 根文件系统
- 启动脚本和运行时库

U-Boot 和 Linux 在引导过程中访问同一 UBIFS 卷。U-Boot 从卷中加载内核映像和 DTB，而 Linux 随后将该卷装载为其根文件系统。

在本应用手册所述的实现完成时，OSPI NOR 闪存包含 Linux 引导所需的所有软件元件。

- tiboot3.bin
- tispl.bin
- U-Boot
- Linux 内核映像
- 设备树 blob (DTB)
- Linux 根文件系统

这种架构可实现 Linux 引导，而无需任何 SD/eMMC 存储。

4 刷写引导元件

本章介绍了如何将引导元件刷写到 OSPI NOR 闪存中。在刷写映像之前，通过 Linux MTD 子系统验证 OSPI 分区布局，以识别相应的 MTD 器件。

4.1 引导元件概述

本应用手册中实现的 J721S2 引导进程使用存储在 OSPI NOR 闪存中的三个引导软件元件。

- tiboot3.bin – 次级引导加载程序
- tispl.bin – A72 SPL
- u-boot.img – U-Boot

在系统启动期间，ROM 代码从 OSPI NOR 闪存加载 tiboot3.bin。然后，引导进程通过 tispl.bin 和 u-boot.img 继续，接着从存储在根文件系统分区中的 UBIFS 文件系统加载 Linux。

4.2 识别 OSPI MTD 分区

Linux 通过存储器技术器件 (MTD) 子系统公开 OSPI NOR 闪存分区。每个分区都分配有一个可以通过 /dev/mtdX 接口访问的 MTD 设备。

可用的 OSPI 分区可在 J721S2 EVM 上按如下方式获得。

```
root@j721s2-evm:~# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00080000 00040000 "ospi.tiboot3"
mtd1: 00200000 00040000 "ospi.tispl"
mtd2: 00400000 00040000 "ospi.u-boot"
mtd3: 00040000 00040000 "ospi.env"
mtd4: 00040000 00040000 "ospi.env.backup"
mtd5: 037c0000 00040000 "ospi.rootfs"
mtd6: 00040000 00040000 "ospi.phypattern""
```

4.3 将引导元件刷写到 OSPI NOR 闪存中

识别 OSPI MTD 分区后，可以将引导元件刷写到 OSPI NOR 闪存中。

在开发过程中，引导元件映像是从用于引导 J721S2 EVM 的 SD 卡引导分区获得。引导元件映像的位置可在 /run/media/BOOT-mmcblk1p1 找到，如下所示。

```
root@j721s2-evm:~# ls -l /run/media/BOOT-mmcblk1p1/
total 4906
-rwxrwx--- 1 root disk  465805 Jun 13  2026 tiboot3-j721s2-gp-evm-j721s2-evm-
k3r5-2025.01+git97201+743712b-r0_tisdk_6_edgeai_5.bin
-rwxrwx--- 1 root disk  465805 Jun 13  2026 tiboot3-j721s2-gp-evm.bin
-rwxrwx--- 1 root disk  468394 Jun 13  2026 tiboot3-j721s2-hs-evm.bin
-rwxrwx--- 1 root disk  468394 Jun 13  2026 tiboot3-j721s2-hs-fs-evm.bin
-rwxrwx--- 1 root disk  465805 Jun 13  2026 tiboot3.bin
-rwxrwx--- 1 root disk 1326915 Jun 13  2026 tispl.bin
-rwxrwx--- 1 root disk 1359931 Jun 13  2026 u-boot.img
-rwxrwx--- 1 root disk     941 Jun 13  2026 uEnv.txt
-rwxrwx--- 1 root disk      14 Jun 13  2026 version
```

表 4-1 显示了引导元件映像及其目标 OSPI 分区之间的映射。

表 4-1. 引导元件刷写目标

图像	目标 MTD 设备	OSPI 分区
tiboot3.bin	/dev/mtd0	ospi.tiboot3
tispl.bin	/dev/mtd1	ospi.tispl
u-boot.img	/dev/mtd2	ospi.u-boot

引导元件使用 Linux **flashcp** 实用程序进行刷写。

```
root@j721s2-evm:~# flashcp /run/media/BOOT-mmcblk1p1/tiboot3.bin /dev/mtd0
root@j721s2-evm:~# flashcp /run/media/BOOT-mmcblk1p1/tispl.bin   /dev/mtd1
root@j721s2-evm:~# flashcp /run/media/BOOT-mmcblk1p1/u-boot.img  /dev/mtd2
```

刷写完成后，OSPI NOR 闪存包含进入 U-Boot 提示符所需的所有引导软件。下一节验证当电路板配置为 OSPI 引导模式时，是否可以成功加载已刷写的映像。

4.4 验证引导元件

刷写引导元件后，电路板将以 OSPI 引导模式进入 U-Boot 命令提示符。

验证期间观察到以下 U-Boot 横幅。

```
U-Boot 2025.01-00551-g743712b9ee4b (Jul 31 2025 - 11:32:44 +0000)
...
=>
```

成功访问 U-Boot 提示符确认所有引导元件均已正确刷写，并可以从 OSPI NOR 闪存加载。

在此阶段，完整的引导软件栈可在 OSPI NOR 闪存中使用。下一章将介绍如何在 ospi.rootfs 分区内创建基于 UBI/UBIFS 的根文件系统。

5 创建 UBIFS 根文件系统

如第 2.3 节所述，Linux 使用 UBI 层和 UBIFS 文件系统来管理原始闪存器件，例如 OSPI NOR 闪存。

在本应用手册中，ospi.rootfs 分区转换为 UBI 器件并创建一个名为 rootfs 的 UBIFS 卷。该卷稍后用于存储 Linux 内核映像、器件树二进制文件 (DTB) 和 Linux 根文件系统。

本章介绍了用于在 OSPI NOR 闪存上创建和验证 UBIFS 卷的过程。

5.1 准备 RootFS 分区

节 3 中引入的 OSPI 闪存布局保留 ospi.rootfs 分区用于 Linux 存储。

在创建 UBI 器件之前，必须删除分区中的任何现有数据。

```
root@j721s2-evm:~# ubiformat /dev/mtd5
ubiformat: mtd5 (nor), size 58458112 bytes (55.7 MiB), 223 eraseblocks of 262144 bytes (256.0 KiB),
min. I/O size 16 bytes
libscan: scanning eraseblock 222 -- 100 % complete
ubiformat: 223 eraseblocks have valid erase counter, mean value is 1
ubiformat: formatting eraseblock 222 -- 100 % complete
root@j721s2-evm:~#
```

格式化完成后，分区已准备好进行 UBI 连接。

5.2 创建 UBI 设备

可以使用 ubiattach 命令将格式化的分区附加到 Linux UBI 子系统。

```
root@j721s2-evm:~# ubiattach /dev/ubi_ctrl -m 5
[  261.852295] ubi0: attaching mtd5
[  261.864496] ubi0: scanning is finished
[  261.885069] ubi0: attached mtd5 (name "ospi.rootfs", size 55 MiB)
[  261.891218] ubi0: PEB size: 262144 bytes (256 KiB), LEB size: 262016 bytes
[  261.904222] ubi0: min./max. I/O unit sizes: 16/256, sub-page size 16
[  261.910748] ubi0: VID header offset: 64 (aligned 64), data offset: 128
[  261.917331] ubi0: good PEBs: 223, bad PEBs: 0, corrupted PEBs: 0
[  261.923343] ubi0: user volume: 0, internal volumes: 1, max. volumes count: 128
[  261.930563] ubi0: max/mean erase counter: 6/3, WL threshold: 4096, image sequence number:
1854191452
[  261.939690] ubi0: available PEBs: 219, total reserved PEBs: 4, PEBs reserved for bad PEB
handling: 0
[  261.948828] ubi0: background thread "ubi_bgt0d" started, PID 1175
UBI device number 0, total 223 LEBs (58429568 bytes, 55.7 MiB), available 219 LEBs (57381504 bytes,
54.7 MiB), LEB size 262016 bytes (255.8 KiB)
root@j721s2-evm:~#
root@j721s2-evm:~# ubinfo
UBI version:                    1
Count of UBI devices:           1
UBI control device major/minor: 10:126
Present UBI devices:            ubi0
```

成功的连接会创建一个与 OSPI 根文件系统分区关联的 UBI 器件。在这一阶段，存储层次结构将变为

```
OSPI NOR Flash
      |
      +-- /dev/mtd5
                |
                +-- ubi0
```

5.3 创建 UBIFS 卷

在新连接的 UBI 器件中创建名为 *rootfs* 的 UBIFS 卷。

```
root@j721s2-evm:~# ubimkvol /dev/ubi0 -N rootfs -m
Set volume size to 57381504
Volume ID 0, size 219 LEBs (57381504 bytes, 54.7 MiB), LEB size 262016 bytes (255.8 KiB), dynamic,
name "rootfs", alignment 1
root@j721s2-evm:~#
```

-m 选项将所有剩余的可用空间分配给卷。

由此得出的存储层次结构如下所示。

```
OSPI NOR Flash
      |
      +-- /dev/mtd5
                |
                +-- ubi0
                        |
                        +-- rootfs
```

在本应用手册的其余部分中会用到该卷。

5.4 验证 UBIFS 操作

UBIFS 卷可以使用 Linux UBIFS 驱动程序挂载。

创建挂载点并挂载卷。

```
root@j721s2-evm:~# mkdir -p /mnt/ospi_rootfs
root@j721s2-evm:~# mount -t ubifs -o compr=none ubi0:rootfs /mnt/ospi_rootfs
[  378.881540] UBIFS (ubi0:0): default file-system created
[  378.886938] UBIFS (ubi0:0): Mounting in unauthenticated mode
[  378.892793] UBIFS (ubi0:0): background thread "ubifs_bgt0_0" started, PID 1188
[  378.924191] UBIFS (ubi0:0): UBIFS: mounted UBI device 0, volume 0, name "rootfs"
[  378.931604] UBIFS (ubi0:0): LEB size: 262016 bytes (255 KiB), min./max. I/O unit sizes: 16
bytes/256 bytes
[  378.941248] UBIFS (ubi0:0): FS size: 54761344 bytes (52 MiB, 209 LEBs), max 219 LEBs, journal
size 2620160 bytes (2 MiB, 10 LEBs)
[  378.952889] UBIFS (ubi0:0): reserved for root: 2586513 bytes (2525 KiB)
[  378.959495] UBIFS (ubi0:0): media format: w5/r0 (latest is w5/r0), UUID 0D0FF060-F89F-4D12-
B474-1C6AA3288A8D, small LPT model
root@j721s2-evm:~#
```

可以使用标准的 Linux 文件操作来验证已挂载的文件系统。

例如，

```
root@j721s2-evm:~# echo hello > /mnt/ospi_rootfs/test.txt
root@j721s2-evm:~# sync
root@j721s2-evm:~# cat /mnt/ospi_rootfs/test.txt
hello
root@j721s2-evm:~#
```

除了在 Linux 上进行验证之外，还可以从 U-Boot 环境验证 UBIFS 卷。此步骤确保在存储 Linux 引导工件之前可以由 U-Boot 访问在 Linux 中创建的文件系统。

使用了以下命令。

=> ubi part ospi.rootfs

=> ubifsmount ubi0:rootfs

=> ubifsls /

示例输出如下所示。

```
=> ubi part ospi.rootfs
k3-navss-ringacc ringacc@2b800000: Ring Accelerator probed rings:286, gp-rings[96,20] sci-dev-id:272
k3-navss-ringacc ringacc@2b800000: dma-ring-reset-quirk: disabled
jedec_spi_nor flash@0: non-uniform erase sector maps are not supported yet.
cadence_spi spi@47040000: Pattern not found. Skipping calibration
SF: Detected s28hs512t with page size 256 Bytes, erase size 256 KiB, total 64 MiB
jedec_spi_nor flash@0: Software reset enable failed: -524
cadence_spi spi@47050000: Pattern not found. Skipping calibration
SF: Detected mt25qu512a with page size 256 Bytes, erase size 64 KiB, total 64 MiB
Could not find a valid device for spi-nand0
ubi0: attaching mtd7
ubi0: scanning is finished
ubi0: attached mtd7 (name "ospi.rootfs", size 55 MiB)
ubi0: PEB size: 262144 bytes (256 KiB), LEB size: 262016 bytes
ubi0: min./max. I/O unit sizes: 16/256, sub-page size 16
ubi0: VID header offset: 64 (aligned 64), data offset: 128
ubi0: good PEBs: 223, bad PEBs: 0, corrupted PEBs: 0
ubi0: user volume: 1, internal volumes: 1, max. volumes count: 128
ubi0: max/mean erase counter: 7/3, WL threshold: 4096, image sequence number: 1854191452
ubi0: available PEBs: 0, total reserved PEBs: 223, PEBs reserved for bad PEB handling: 0
=> ubifsmount ubi0:rootfs
UBIFS (ubi0:0): recovery needed
UBIFS (ubi0:0): recovery deferred
UBIFS (ubi0:0): UBIFS: mounted UBI device 0, volume 0, name "rootfs", R/O mode
UBIFS (ubi0:0): LEB size: 262016 bytes (255 KiB), min./max. I/O unit sizes: 16 bytes/256 bytes
UBIFS (ubi0:0): FS size: 54761344 bytes (52 MiB, 209 LEBs), journal size 2620160 bytes (2 MiB, 10
LEBs)
UBIFS (ubi0:0): reserved for root: 2586513 bytes (2525 KiB)
UBIFS (ubi0:0): media format: w5/r0 (latest is w4/r0), UUID 0D0FF060-F89F-4D12-B474-1C6AA3288A8D,
small LPT model
=> ubifsls /
                6  Fri May 30 02:46:03 2025  test.txt
=>
```

成功执行这些命令确认 U-Boot 可以正确识别在 Linux 中创建的 UBI 元数据和 UBIFS 卷，并且可以在存储 Linux 引导文件之前访问这些卷。

成功读取和写入操作确认如下。

- 可访问 OSPI NOR 闪存分区。
- UBI 设备工作正常。
- UBIFS 卷可以从 Linux 和 U-Boot 环境中挂载和使用。

此时，rootfs 卷已准备好存储下一章中所述的 Linux 内核映像、DTB 和根文件系统元件。

6 将 Linux 引导工件存储在 UBIFS 中

在第 5 章中创建的 UBIFS 卷是从 Linux 和 U-Boot 环境验证的。在此阶段，卷仅包含文件系统验证期间使用的文件。

若要完全从 OSPI NOR 闪存引导 Linux，UBIFS 卷必须填充 Linux 根文件系统、内核映像和器件树二进制文件 (DTB)。本章介绍了准备这些引导工件的过程。

6.1 挂载 UBIFS 根文件系统卷

首先，连接 UBI 器件并挂载 UBIFS 根文件系统卷。

```
root@j721s2-evm:~# ubiattach /dev/ubi_ctrl -m 5
[   24.823337] ubi0: attaching mtd5
[   24.835461] ubi0: scanning is finished
[   24.860253] ubi0: attached mtd5 (name "ospi.rootfs", size 55 MiB)
[   24.866388] ubi0: PEB size: 262144 bytes (256 KiB), LEB size: 262016 bytes
[   24.873293] ubi0: min./max. I/O unit sizes: 16/256, sub-page size 16
[   24.879662] ubi0: VID header offset: 64 (aligned 64), data offset: 128
[   24.886203] ubi0: good PEBs: 223, bad PEBs: 0, corrupted PEBs: 0
[   24.892210] ubi0: user volume: 1, internal volumes: 1, max. volumes count: 128
[   24.899434] ubi0: max/mean erase counter: 7/3, WL threshold: 4096, image sequence number:
1854191452
[   24.908559] ubi0: available PEBs: 0, total reserved PEBs: 223, PEBs reserved for bad PEB
handling: 0
[   24.917690] ubi0: background thread "ubi_bgt0d" started, PID 1174
UBI device number 0, total 223 LEBs (58429568 bytes, 55.7 MiB), available 0 LEBs (0 bytes), LEB
size 262016 bytes (255.8 KiB)
root@j721s2-evm:~# mkdir -p /mnt/ospi_rootfs
root@j721s2-evm:~#
root@j721s2-evm:~# mount -t ubifs -o compr=none ubi0:rootfs /mnt/ospi_rootfs
[   36.600493] UBIFS (ubi0:0): Mounting in unauthenticated mode
[   36.606414] UBIFS (ubi0:0): background thread "ubifs_bgt0_0" started, PID 1180
[   36.621952] UBIFS (ubi0:0): recovery needed
[   38.692922] UBIFS (ubi0:0): recovery completed
[   38.697465] UBIFS (ubi0:0): UBIFS: mounted UBI device 0, volume 0, name "rootfs"
[   38.705150] UBIFS (ubi0:0): LEB size: 262016 bytes (255 KiB), min./max. I/O unit sizes: 16
bytes/256 bytes
[   38.714909] UBIFS (ubi0:0): FS size: 54761344 bytes (52 MiB, 209 LEBs), max 219 LEBs, journal
size 2620160 bytes (2 MiB, 10 LEBs)
[   38.726619] UBIFS (ubi0:0): reserved for root: 2586513 bytes (2525 KiB)
[   38.733285] UBIFS (ubi0:0): media format: w5/r0 (latest is w5/r0), UUID 0D0FF060-F89F-4D12-
B474-1C6AA3288A8D, small LPT model
root@j721s2-evm:~#
```

6.2 使用 BusyBox 创建根文件系统结构

创建根文件系统所需的最小目录结构。

```
mkdir -p /mnt/ospi_rootfs/bin
mkdir -p /mnt/ospi_rootfs/usr/lib
mkdir -p /mnt/ospi_rootfs/dev
mkdir -p /mnt/ospi_rootfs/proc
mkdir -p /mnt/ospi_rootfs/sys
```

将 BusyBox 复制到根文件系统中。

```
cp /usr/bin/busybox /mnt/ospi_rootfs/bin/
```

复制所需的共享库。

```
cp /usr/lib/libm.so.6 /mnt/ospi_rootfs/usr/lib/
cp /usr/lib/libc.so.6 /mnt/ospi_rootfs/usr/lib/
cp /usr/lib/ld-linux-aarch64.so.1 /mnt/ospi_rootfs/usr/lib/
```

可以使用 *ldd* 命令来标识这些库。

```
ldd /usr/bin/busybox
```

备注

本应用手册使用基于 BusyBox 的最小根文件系统来降低复杂性并专注于基本的 Linux 引导过程。

6.3 验证 BusyBox 执行

验证 BusyBox 是否可以仅使用存储在根文件系统中的库执行，如下所示。

```
root@j721s2-evm:~# LD_LIBRARY_PATH=/mnt/ospi_rootfs/usr/lib /mnt/ospi_rootfs/bin/busybox --help
BusyBox v1.36.1 () multi-call binary.
BusyBox is copyrighted by many authors between 1998-2015.
Licensed under GPLv2. See source distribution for detailed
copyright notices.
:
:
root@j721s2-evm:~# LD_LIBRARY_PATH=/mnt/ospi_rootfs/usr/lib /mnt/ospi_rootfs/bin/busybox echo
"busybox test"
busybox test
```

这将确认 BusyBox 和所有必需的共享库已成功加载。

6.4 创建 BusyBox 命令链接

符号链接可用于无缝执行基本 Linux 命令。

```
cd /mnt/ospi_rootfs/bin

ln -sf busybox sh
ln -sf busybox mount
ln -sf busybox echo
ln -sf busybox cat
ln -sf busybox ls
ln -sf busybox mkdir
ln -sf busybox mknod
ln -sf busybox ps
ln -sf busybox dmesg
ln -sf busybox sleep
ln -sf busybox umount
ln -sf busybox uname
ln -sf busybox pwd
```

6.5 创建 init 脚本

创建根文件系统初始化脚本。

```
cat > /mnt/ospi_rootfs/init << 'EOF'
#!/bin/sh

echo ""
echo "Starting OSPI RootFS Initialization..."
echo ""

mount -t proc proc /proc
mount -t sysfs sysfs /sys
mount -t devtmpfs devtmpfs /dev 2>/dev/null

echo ""
echo "Booted from OSPI UBIFS RootFS"
echo ""

exec /bin/sh
EOF
```

使脚本可执行。

```
chmod +x /mnt/ospi_rootfs/init
sync
```

6.6 使用 chroot 测试根文件系统

使用 *chroot* 验证根文件系统是否可以独立运行。

```
root@j721s2-evm:~# chroot /mnt/ospi_rootfs /bin/sh
@j721s2-evm:/# pwd
/
@j721s2-evm:/# ls
bin       boot      dev       init      proc      sys       test.txt  usr
@j721s2-evm:/# /bin/busybox echo "chroot busybox OK"
chroot busybox OK
@j721s2-evm:/# /bin/busybox echo "chroot busybox OK"
chroot busybox OK
@j721s2-evm:/# cat /init
#!/bin/sh

echo ""
echo "Starting OSPI RootFS Initialization..."
echo ""

mount -t proc proc /proc
mount -t sysfs sysfs /sys
mount -t devtmpfs devtmpfs /dev

echo ""
echo "Booted from OSPI UBIFS RootFS"
echo ""

exec /bin/sh
@j721s2-evm:/#
```

6.7 复制 Linux 内核映像和 DTB

创建引导目录，复制内核映像和设备树。

```
root@j721s2-evm:~# mkdir -p /mnt/ospi_rootfs/boot
root@j721s2-evm:~# mkdir -p /mnt/ospi_rootfs/boot/dtb
root@j721s2-evm:~# cp /boot/Image /mnt/ospi_rootfs/boot/
root@j721s2-evm:~# cp /boot/dtb/ti/k3-j721s2-common-proc-board.dtb /mnt/ospi_rootfs/boot/dtb/
root@j721s2-evm:~# sync
```

7 从 UBIFS 配置 Linux 引导

创建最小根文件系统并将 Linux 内核映像和器件树复制到 UBIFS 卷后，下一步是配置 U-Boot 以加载这些文件并直接从 OSPI NOR 闪存引导 Linux。

以下各节介绍了如何配置引导参数、从 UBIFS 加载内核和器件树，以及验证 Linux 引导是否成功。

7.1 配置引导参数

Linux 需要引导参数来标识根文件系统位置和要执行的初始进程。

按如下方式配置引导参数。

```
setenv bootargs 'console=ttyS2,115200n8 earlycon=ns16550a,mmio32,0x02880000 ubi.mtd=ospi.rootfs
root=ubi0:rootfs rootfstype=ubifs rw init=/init'
```

以下列出了关键参数。

表 7-1. 主要参数

参数	说明
console=ttyS2,115200n8	Linux 控制台输出
earlycon=ns16550a,...	早期引导调试消息
ubi.mtd=ospi.rootfs	从 OSPI rootfs 分区连接 UBI 器件
root=ubi0:rootfs	使用 rootfs UBI 卷作为根文件系统
rootfstype=ubifs	根文件系统类型
rw	挂载根文件系统读取/写入
Init=/init	执行自定义初始化脚本

7.2 从 UBIFS 加载内核和 DTB

附加 OSPI rootfs 分区并挂载 UBIFS 卷。

```
ubi part ospi.rootfs
ubifsmount ubi0:rootfs
```

加载 Linux 内核映像并加载器件树。

```
ubifsload ${loadaddr} /boot/Image
ubifsload ${fdtaddr} /boot/dtb/k3-j721s2-common-proc-board.dtb
```

7.3 测试 Linux 引导

使用加载的内核映像和器件树引导 Linux。

```
booti ${loadaddr} - ${fdtaddr}
```

在引导期间，Linux 应连接 UBI 器件并挂载 UBIFS 根文件系统。

关键信息包括以下内容。

```
ubi0: attached mtd5 (name "ospi.rootfs", size 55 MiB)

UBIFS (ubi0:0): UBIFS: mounted UBI device 0, volume 0, name "rootfs"

VFS: Mounted root (ubifs filesystem)

Run /init as init process
```

内核启动初始化脚本后，应显示如下消息。

```
Starting OSPI RootFS Initialization...

Booted from OSPI UBIFS RootFS
```

最后，BusyBox shell 应该可以使用。

```
~ #
```

这证实了 Linux 已使用完全存储在 OSPI NOR 闪存中的根文件系统成功引导。

- Linux 内核从 OSPI NOR 闪存加载
- 器件树已从 OSPI NOR 闪存加载
- 根文件系统从 UBIFS 挂载
- /init 已成功执行
- BusyBox shell 已成功启动

这些观察结果表明，Linux 引导不再依赖于 eMMC 或 SD 存储器件。

7.4 创建可重复使用的引导命令

验证引导后，可以将命令组合成可重用的引导序列。

```
setenv ospi_boot '
ubi part ospi.rootfs;
ubifsmount ubi0:rootfs;
ubifsload ${loadaddr} /boot/Image;
ubifsload ${fdtaddr} /boot/dtb/k3-j721s2-common-proc-board.dtb;
setenv bootargs "console=ttyS2,115200n8 earlycon=ns16550a,mmio32,0x02880000 ubi.mtd=ospi.rootfs
root=ubi0:rootfs rootfstype=ubifs rw init=/init";
booti ${loadaddr} - ${fdtaddr}
'
```

执行如下所示的完整引导序列。

```
run ospi_boot
```

这样可以将 Linux 引导过程减少为单个 U-Boot 命令，从而简化未来的测试和部署。

8 完全从 OSPI NOR 闪存引导 Linux

8.1 自动执行 U-Boot 中的引导序列

上一章演示了如何使用手动输入的 U-Boot 命令进行 Linux 引导。虽然这种方法在开发和调试过程中很有用，但量产系统通常要求启动过程在上电后自动启动。

一种方法是在 U-Boot 配置中定义默认引导命令。这使 U-Boot 能够自动挂载 UBIFS 卷，从 OSPI NOR 闪存加载 Linux 内核映像和器件树，并在无需用户交互的情况下启动 Linux。可将默认引导命令添加到以下 U-Boot 配置文件：configs/j721s2_evm_a72_defconfig。

本示例中可以使用以下配置。

```
cd ti-processor-sdk-linux-adas-j721s2-evm-11_01_00_03/board-support/ti-u-boot-2025.01+git
cat >> configs/j721s2_evm_a72_defconfig << 'EOF'
CONFIG_USE_BOOTCOMMAND=y
CONFIG_BOOTCOMMAND="setenv bootargs 'console=ttyS2,115200n8 earlycon=ns16550a,mmio32,0x02880000
ubi.mtd=ospi.rootfs root=ubi0:rootfs rootfstype=ubifs rw init=/init'; ubi part ospi.rootfs;
ubifsmount ubi0:rootfs; ubifsload ${loadaddr} /boot/Image; ubifsload ${fdtaddr} /boot/dtb/k3-j721s2-
common-proc-board.dtb; booti ${loadaddr} - ${fdtaddr}"
EOF
```

在重新构建 U-Boot 并将更新的 u-boot.img 刷写到 ospi.u-boot 分区中后，电路板会在上电期间自动执行完整的 Linux 引导序列。

8.2 端到端引导流程

图 8-1 展示了从 OSPI NOR 闪存的完整 Linux 引导序列。

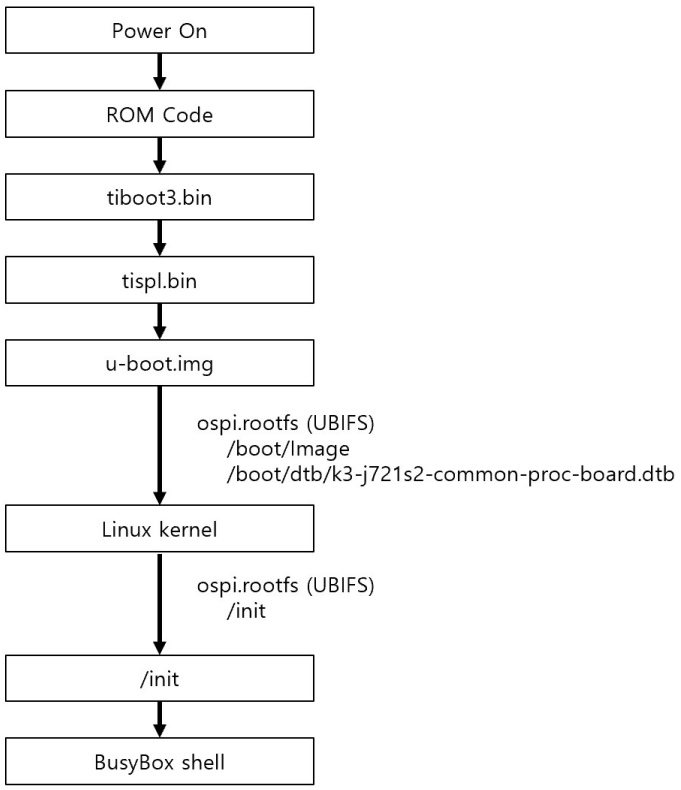

图 8-1. 端到端引导流程

U-Boot 挂载 UBIFS 卷后，会从 OSPI rootfs 分区加载 Linux 内核映像和器件树。Linux 随后挂载与其根文件系统相同的 UBIFS 卷，并执行 /init 脚本，这将启动 BusyBox shell 环境。

9 总结

本应用手册演示了如何使用基于 UBIFS 的根文件系统从 TI J721S2 平台上的 OSPI NOR 闪存完全引导 Linux。

最小的根文件系统是使用 BusyBox、一小组运行时库和一个自定义初始化脚本创建的。Linux 内核映像和器件树存储在同一 UBIFS 卷中，从而使 U-Boot 能够直接从 OSPI NOR 闪存加载所有必需的软件元件。生成的系统成功引导 Linux，无需 eMMC、SD 卡或其他外部存储器件。通过将 UBI 和 UBIFS 与基于 BusyBox 的最小根文件系统相结合，可以仅使用 OSPI NOR 闪存实现紧凑的自包含 Linux 引导解决方案。这种方法对于需要简化存储架构、减少元件数量或无需可移动存储介质即可运行的嵌入式系统特别有用。

10 参考资料

- 德州仪器 (TI)，*TDA4VE* 产品页面。
- 德州仪器 (TI)，*J721S2、TDA4AL、TDA4VL、TDA4VE、AM68A 技术参考手册*，技术参考手册。
- 德州仪器 (TI)，*TDA4VE TDA4AL TDA4VL Jacinto™ 处理器，器件版本 1.0*，数据表。
- 德州仪器 (TI)，*PROCESSOR-SDK-J721S2* 产品页面。

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月