参考資料

Application Note

TI の J721S2 上の OSPI NOR フラッシュからの Linux ブート (eMMC なし)

Johnny Kim

概要

このアプリケーション ノートでは、eMMC ストレージを使用せずに、テキサス インスツルメンツの J721S2 プラットフォーム上で、OSPI NOR フラッシュから Linux 全体をブートする方法について説明します。

実証されたブート フローでは、tiboot3.bin、tispl.bin、U-Boot、Linux カーネル イメージ、デバイス ツリー ブロブ (DTB)、ルート ファイルシステムを含む、すべてのブート コンポーネントを OSPI NOR フラッシュに格納します。UBI/UBIFS ベースのルート ファイルシステムが OSPI フラッシュ内に作成され、システム起動時に Linux のルート ファイルシステムとして使用されます。この実装はプロセッサ SDK Linux を使用して J721S2 評価基板で検証されます。このアプリケーション ノートでは、OSPI フラッシュのパーティション分割、UBI および UBIFS ボリュームの作成、最小ルート ファイルシステムの構築、UBIFS 内の Linux アーティファクトの格納、U-Boot 構成、OSPI NOR フラッシュからの完全な Linux ブートについて説明します。結果として生じるソリューションでは、eMMC ストレージを必要としない完全に機能する Linux ブート チェーンが実証され、シンプルなストレージ構成または OSPI ベースの導入を必要とするシステムに対する代替ブートアーキテクチャを提供します。

目次

1 概要 3
1.1 疑問 3
1.2 範囲 3
1.3 ターゲット ブート アーキテクチャ 4
2 背景 4
2.1 J721S2 のブート フロー 4
2.2 OSPI NOR フラッシュの概要 5
2.3 UBI と UBIFS の概要 5
3 OSPI フラッシュ レイアウト 7
3.1 パーティション レイアウト 7
3.2 ブートローダーのパーティション 7
3.3 ルート ファイルシステム パーティション 8
4 ブート コンポーネントの書き込み 10
4.1 ブート コンポーネントの概要 10
4.2 OSPI MTD パーティションの識別 10
4.3 ブート コンポーネントの OSPI NOR フラッシュへの書き込み 10
4.4 ブート コンポーネントの確認 11
5 UBIFS ルート ファイルシステムの作成 12
5.1 RootFS パーティションの準備 12
5.2 UBI デバイスの作成 12
5.3 UBIFS ボリュームの作成 13
5.4 UBIFS の動作の確認 13
6 Linux ブート アーティファクトの UBIFS への保存 15
6.1 UBIFS ルート ファイルシステム ボリュームのマウント 15
6.2 BusyBox を使用したルート ファイルシステム構造の作成 15
6.3 BusyBox 実行の確認 16
6.4 BusyBox コマンド リンクの作成 16

English Document: SDAA437

6.5 init スクリプトの作成....16
6.6 chroot を使用したルート ファイルシステムのテスト....17
6.7 Linux カーネル イメージと DTB のコピー....17
7 UBIFS からの Linux ブートの構成....18
7.1 ブート引数の構成....18
7.2 UBIFS からカーネルと DTB をロード....18
7.3 Linux ブートのテスト....18
7.4 再利用可能なブート コマンドの作成....19
8 全面的な OSPI NOR フラッシュからの Linux ブート....20
8.1 U-Boot のブート シーケンスの自動化....20
8.2 エンド ツー エンドのブート フロー....20
9 まとめ....22
10 参考資料....22

商標

すべての商標は、それぞれの所有者に帰属します。

1 概要

1.1 疑問

テキサス インスツルメンツの J721S2 デバイスは、SD カード、eMMC、OSPI NOR フラッシュなど、複数のブート メディアをサポートしています。多くの量産システムでは、tiboot3.bin、tispl.bin、U-Boot などの初期ブート コンポーネントを格納するため、一般的に OSPI NOR フラッシュを使用しています。ただし、カーネル イメージ、デバイス ツリー ブロブ (DTB)、ルート ファイルシステムのストレージ容量は大きいため、Linux ベースの多くのシステムでは eMMC や他の大容量ストレージ デバイスに格納しています。その結果、ブート アーキテクチャでは複数のストレージ デバイスが必要になります。一部のアプリケーションでは、eMMC への依存を排除して、すべてのブート コンポーネントをシングル OSPI NOR フラッシュ デバイスに集約する方が望ましい結果が得られます。このようなアプローチを採用すると、ストレージ管理が簡単になり、システムの複雑さが軽減され、eMMC を利用できないシステムでの導入が可能になります。このアプリケーション ノートでは Linux ブート ソリューションの例を示しており、ここではブートローダー、カーネル イメージ、DTB、ルート ファイルシステムを含むすべてのブート コンポーネントが OSPI NOR フラッシュに格納されています。ルート ファイルシステムは UBI と UBIFS を使用して実装され、フラッシュ対応の堅牢なストレージ設計を実現します。

1.2 範囲

このアプリケーション ノートの目的は、eMMC ストレージに頼らない、J721S2 プラットフォーム上の OSPI NOR フラッシュからの Linux ブートを実証および検証することです。

以下の内容について説明します。

- OSPI NOR フラッシュ内の UBI/UBIFS ルート ファイルシステムの作成
- 最小限の Linux ルート ファイルシステムの構築と検証
- Linux カーネル イメージおよび DTB の UBIFS への格納
- U-Boot の使用による UBIFS からのカーネル イメージおよび DTB のロード
- UBIFS ルート ファイルシステムを使用した Linux のブート
- OSPI NOR フラッシュからのシステム全体のブート

このドキュメントに記載されている手順は、プロセッサ SDK Linux を使用して J721S2 評価基板で検証されました。

1.3 ターゲット ブート アーキテクチャ

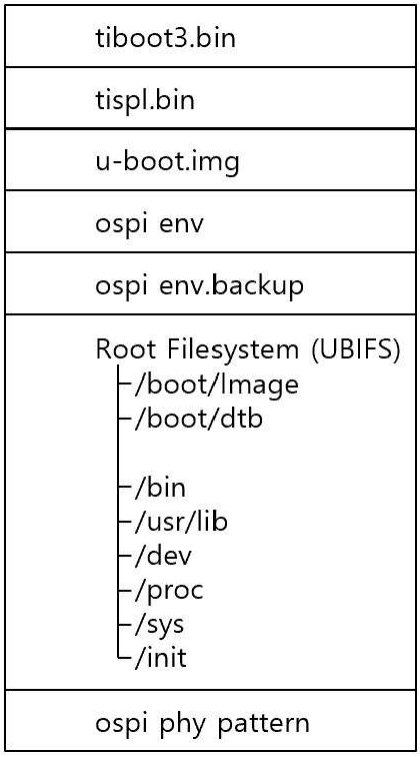

図 1-1. OSPI NOR フラッシュ レイアウト

Linux のブートに必要なすべてのソフトウェア コンポーネントは、OSPI NOR フラッシュに格納されます。U-Boot が OSPI フラッシュにある UBIFS ボリュームから Linux カーネル イメージと DTB をロードすると、Linux はルート ファイルシステムと同じ UBIFS ボリュームをマウントします。その結果、eMMC ストレージを必要とせずに、完全なブート プロセスが実行されます。

2 背景

2.1 J721S2 のブート フロー

J721S2 デバイスは、複数ステージのブート プロセスを使用してシステムを初期化し Linux オペレーティング システムをロードします。

デバイスの電源がオンになると、まずオンチップ ROM ブートローダーが実行されます。構成済みのブート モードに基づき、ROM は選択したブート メディアから tiboot3.bin をロードします。tiboot3.bin イメージは重要なデバイス リソースを初期化して tispl.bin をロードします。

tispl.bin イメージは、DDR の構成やシステム ファームウェア コンポーネントのロードなど、追加システムの初期化を実行します。初期化が完了すると、tispl.bin は U-Boot をロードし起動します。

U-Boot は、Linux カーネル イメージとデバイス ツリー ブロブ (DTB) のロード、Linux ブート引数の準備、Linux カーネルへの転送制御を扱います。

このアプリケーション ノートに実装されているブート シーケンスを以下に示します。

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

この作業の目的は、このブート チェーンのすべてのコンポーネントを OSPI NOR フラッシュ内に保存し、eMMC ストレージに依存しないようにすることです。

2.2 OSPI NOR フラッシュの概要

OSPI NOR フラッシュは、ROM ブートローダーによる直接アクセスをサポートする信頼性の高い不揮発性ストレージであるため、J721S2 プラットフォームでブートローダー コンポーネントを格納するために一般的に使用されます。

OSPI NOR フラッシュは通常 eMMC デバイスよりストレージ容量が低いですが、組込みシステム向けのよりシンプルなストレージ アーキテクチャを実現します。利用可能な容量が十分である場合は、シングル OSPI デバイスに、ブートローダー イメージ、Linux ブート アーティファクト、ルート ファイルシステムを含めることができます。

この実装では、64MB OSPI NOR フラッシュ デバイスを使用します。フラッシュは以下の項目を格納するようにパーティション分割されています。

- tiboot3.bin
- tispl.bin
- u-boot.img
- U-Boot 環境
- UBI/UBIFS ルート ファイルシステム ボリューム

Linux カーネル イメージ、DTB、ルート ファイルシステムはすべて UBI/UBIFS パーティション内に格納されます。

2.3 UBI と UBIFS の概要

NOR や NAND フラッシュなどの未加工フラッシュ デバイスは、eMMC や SSD などのブロック ストレージ デバイスとは根本的に異なります。ext4 のようなブロック デバイス向けに設計された Linux ファイルシステムは、未加工フラッシュ メモリ上での直接動作を意図していません。

未加工フラッシュ デバイスでの信頼性の高いストレージをサポートするため、Linux は UBI (アンソーテッド ブロック イメージ) 層と UBIFS (UBI ファイル システム) を提供しています。

UBI は MTD サブシステムとファイルシステムの間に位置しています。論理消去ブロック マッピング、ウェア レベリング、不良ブロック処理など、フラッシュ固有の要件を管理します。UBIFS は UBI ボリュームの最上部で動作するフラッシュ対応ファイルシステムです。

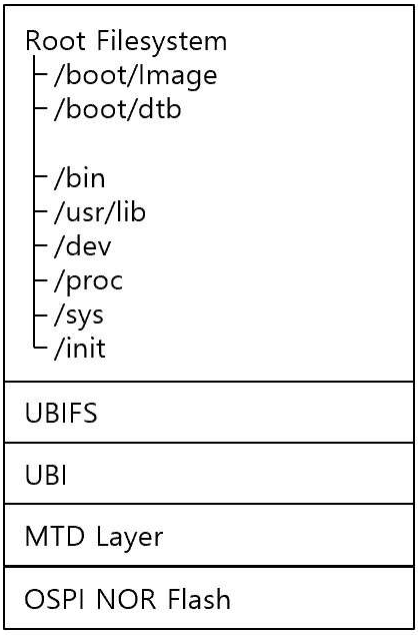

図 2-1. UBI/UBIFS ソフトウェア スタック

UBI と UBIFS を使用すると、Linux が OSPI NOR フラッシュに信頼性の高い方法でファイルを格納してアクセスできると同時に、U-Boot と Linux の両方に対する標準的なファイルシステム インターフェイスを提供できます。

このアプリケーション ノートでは、rootfs という名前の UBI ボリュームが OSPI ルート ファイルシステム パーティションに作成されます。このボリュームには、Linux カーネル イメージ、DTB、Linux ルート ファイルシステムが格納されます。

3 OSPI フラッシュ レイアウト

この章では、このアプリケーション ノート全体で使用する OSPI NOR フラッシュ パーティションのレイアウトについて説明します。このレイアウトでは、ブート コンポーネント、U-Boot 環境ストレージ、UBI/UBIFS ベースの Linux ファイル システム専用の領域が割り当てられます。パーティション構成により、Linux ブートに必要なすべてのソフトウェア コンポーネントをシングル OSPI NOR フラッシュ デバイス内に配置できます。

3.1 パーティション レイアウト

このアプリケーション ノートで使用する J721S2 評価基板には、64MB OSPI NOR フラッシュ デバイスが含まれています。ブート コンポーネント、U-Boot 環境データ、UBI と UBIFS を使用して実装された Linux ルート ファイルシステムを格納するため、フラッシュはパーティション化されます。

表 3-1. 実装に使用されるパーティション レイアウトの概要

パーティション	サイズ	目的
ospi.tiboot3	512KB	2 次段ブートローダー
ospi.tispl	2MB	A72 SPL
ospi.u-boot	4MB	U-Boot イメージ
ospi.env	256KB	U-Boot 環境
ospi.env.backup	256KB	U-Boot バックアップ環境
ospi.rootfs	55.8MB	UBI/UBIFS ファイルシステム
ospi.phypattern	256KB	PHY パターン ストレージ

最大のパーティション ospi.rootfs は、Linux カーネル イメージ、デバイス ツリー ブロブ (DTB)、Linux ルート ファイルシステムを格納するために使用されます。パーティションは UBI デバイスとしてフォーマットされ、rootfs という名前の UBIFS ボリュームが含まれます。

3.2 ブートローダーのパーティション

OSPI NOR フラッシュの最初の部分には、システムの初期化と Linux の起動に必要なソフトウェア コンポーネントが格納されています。電源投入時に、J721S2 ROM コードは ospi.tiboot3 パーティションから tiboot3.bin をロードします。tiboot3.bin イメージはプラットフォームの早期初期化を実行して、ospi.tispl パーティションから tispl.bin をロードします。tispl.bin イメージは DDR メモリを初期化し、システム ファームウェアをロードし、ospi.u-boot パーティションから U-Boot を起動します。次に U-Boot は Linux カーネル イメージと DTB を ospi.rootfs パーティションにある UBIFS ファイルシス

テムからロードします。ospi.env および ospi.env.backup パーティションには、ブート プロセスの制御に使用する U-Boot 環境変数が格納されます。バックアップ パーティションにより、環境破損時のための冗長性が得られます。

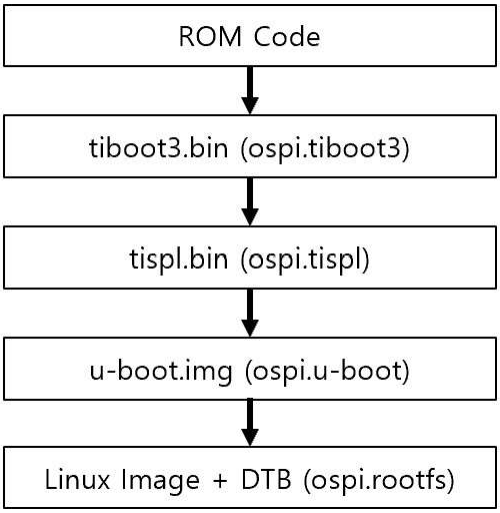

図 3-1. ブート コンポーネント フロー

ブート コンポーネントのパーティションが占有するのはフラッシュ全体の容量のごく一部のみで、OSPI NOR フラッシュのほとんどを Linux ストレージで使用できます。

3.3 ルート ファイルシステム パーティション

ospi.rootfs パーティションは、システム ブートに必要な Linux 関連ソフトウェア コンポーネントすべてを格納するために使用されます。

カーネル イメージ、DTB、ルート ファイルシステムを eMMC に格納する従来の J721S2 Linux システムとは異なり、この実装ではすべての Linux アーティファクトが OSPI NOR フラッシュ内の UBIFS ボリューム内に格納されます。

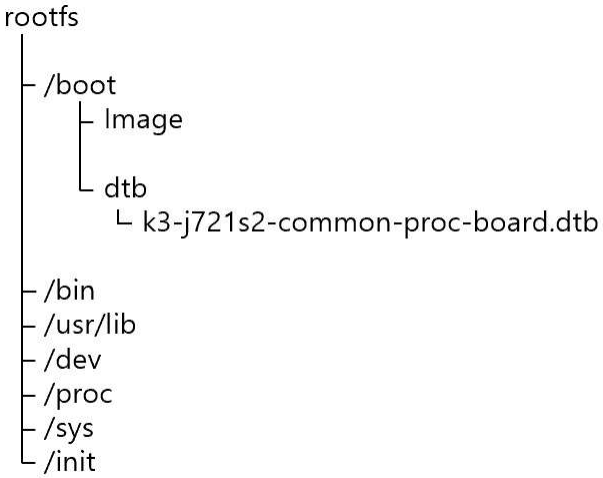

図 3-2. ルート ファイルシステムの構造

パーティションは UBI デバイスとしてフォーマットされ、rootfs という名前の UBIFS ボリュームが含まれます。ボリュームには以下の内容が格納されます。

- Linux カーネル イメージ (Image)
- デバイス ツリー ブロブ (*.dtb)
- Linux ルート ファイルシステム
- 起動スクリプトとランタイム ライブラリ

U-Boot と Linux は両方とも、ブート プロセス中にこの同じ UBIFS ボリュームにアクセスします。U-Boot はカーネル イメージと DTB をそのボリュームからロードしますが、Linux はその後、そのボリュームをルート ファイルシステムとしてマウントします。

このアプリケーション ノートで説明されている実装が完了すると、OSPI NOR フラッシュには Linux ブートに必要なすべてのソフトウェア コンポーネントが含まれるようになります。

- tiboot3.bin
- tispl.bin
- U-Boot
- Linux カーネル イメージ
- デバイス ツリー ブロブ (DTB)
- Linux ルート ファイルシステム

このアーキテクチャにより、SD/eMMC ストレージに依存しない Linux ブートが可能になります。

4 ブート コンポーネントの書き込み

この章では、ブート コンポーネントを OSPI NOR フラッシュに書き込む方法について説明します。イメージを書き込む前に、OSPI パーティション レイアウトが Linux MTD サブシステムを使用して検証され、対応する MTD デバイスが識別されます。

4.1 ブート コンポーネントの概要

このアプリケーション ノートで実装される J721S2 ブート フローでは、OSPI NOR フラッシュに格納された 3 つのブート ソフトウェア コンポーネントを使用します。

- tiboot3.bin – セカンダリ ブートローダー
- tispl.bin – A72 SPL
- u-boot.img – U-Boot

システム起動中、ROM コードは OSPI NOR フラッシュから tiboot3.bin をロードします。ブート プロセスは tispl.bin と u-boot.img を介して続行され、その後 Linux はルート ファイルシステム パーティションに格納された UBIFS ファイルシステムからロードされます。

4.2 OSPI MTD パーティションの識別

Linux は、メモリ テクノロジー デバイス (MTD) のサブシステムを介して OSPI NOR フラッシュ パーティションを公開しています。各パーティションには、/dev/mtdX インターフェイスを介してアクセスできる MTD デバイスが割り当てられます。

利用可能な OSPI パーティションは、J721S2 評価基板で次のように入手できます。

```
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 ブート コンポーネントの OSPI NOR フラッシュへの書き込み

OSPI MTD パーティションを識別すると、ブート コンポーネントを OSPI NOR フラッシュに書き込み可能になります。

開発中に、J721S2 評価基板のブートに使用する SD カードのブート パーティションからブート コンポーネント イメージを取得済みです。ブート コンポーネント イメージは、次のように /run/media/BOOT-mmcblk1p1 にあります。

```
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 は、ブート コンポーネント イメージとターゲット OSPI パーティション間のマッピングを示しています。

表 4-1. ブート コンポーネントの書き込みターゲット

画像	ターゲット MTD デバイス	OSPI の分離
tiboot3.bin	/dev/mtd0	ospi.tiboot3
tispl.bin	/dev/mtd1	ospi.tispl

表 4-1. ブート コンポーネントの書き込みターゲット (続き)

画像	ターゲット MTD デバイス	OSPI の分離
u-boot.img	/dev/mtd2	ospi.u-boot

ブート コンポーネントは Linux の **flashcp** ユーティリティを使用して書き込みされます。

```
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
```

書き込みが完了すると、U-Boot プロンプトに必要なすべてのブート ソフトウェアが OSPI NOR フラッシュに含まれるようになります。次のセクションでは、基板を OSPI ブート モードに構成する際に、書き込み済みイメージを正常にロードされることを確認します。

4.4 ブート コンポーネントの確認

ブート コンポーネントを書き込むと、基板は OSPI ブート モードで U-Boot コマンド プロンプトにアクセスできます。

検証中に以下の U-Boot バナーが観測されました。

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

U-Boot プロンプトへのアクセスに成功し、すべてのブート コンポーネントが正しく書き込まれ、OSPI NOR フラッシュからロードできます。

この段階で、完全なブート ソフトウェア スタックを OSPI NOR フラッシュで利用できます。次の章では、ospi.rootfs パーティション内に UBI/UBIFS ベースのルート ファイルシステムを作成する方法について説明します。

5 UBIFS ルート ファイルシステムの作成

セクション 2.3 で説明したように、Linux は UBI 層と UBIFS ファイルシステムを使用して、OSPI NOR フラッシュのような未加工フラッシュ デバイスを管理します。

このアプリケーション ノートでは、ospi.rootfs パーティションを UBI デバイスに変換し、rootfs という名前の UBIFS ボリュームを作成します。このボリュームは、Linux カーネル イメージ、デバイス ツリー ブロブ (DTB)、Linux ルート ファイルシステムを格納するために後で使用します。

この章では、OSPI NOR フラッシュ上で UBIFS ボリュームを作成して確認する手順について説明します。

5.1 RootFS パーティションの準備

セクション 3 で導入された OSPI フラッシュ レイアウトは、Linux ストレージ用に ospi.rootfs パーティションを予約しています。

UBI デバイスを作成する前に、パーティション内の既存のデータを削除する必要があります。

```
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:~#
```

フォーマットが完了すると、パーティションに UBI を接続する準備が整います。

5.2 UBI デバイスの作成

フォーマット済みのパーティションは、ubiattach コマンドを使用して Linux UBI サブシステムに接続できます。

```
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
```

接続に成功すると、OSPI ルート ファイルシステムのパーティションに関連付けられた UBI デバイスが作成されます。この段階で、ストレージ階層は以下のようになります

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

5.3 UBIFS ボリュームの作成

新しく接続した UBI デバイス内に、*rootfs* という名前の UBIFS ボリュームが作成されます。

```
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 オプションを指定すると、残りの空き容量すべてがそのボリュームに割り当てられます。

結果として得られるストレージ階層は次のとおりです。

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

このボリュームは、このアプリケーション ノートの残りの部分で使用されます。

5.4 UBIFS の動作の確認

UBIFS ボリュームは Linux UBIFS ドライバを使用してマウントできます。

マウント ポイントを作成してボリュームをマウントします。

```
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:~#
```

マウントされたファイルシステムは、標準の Linux ファイル操作を使用して検証できます。

たとえば、

```
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:~#
```

Linux での検証に加え、UBIFS ボリュームも U-Boot 環境から検証できます。この手順では、Linux ブート アーティファクトを格納する前に、Linux で作成されたファイルシステムに U-Boot からアクセスできるようにします。

以下のコマンドが使用されました。

=> ubi part ospi.rootfs

=> ubifsmount ubi0:rootfs

=> ubifsls /

出力例を次に示します。

```
=> 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
=>
```

これらのコマンドが正常に実行されると、Linux で作成された UBI メタデータと UBIFS ボリュームが U-Boot によって正しく認識され、Linux ブート ファイルが格納される前にアクセス可能であることが確認されます。

以下の読み取り/書き込み動作の成功が確認されます。

- OSPI NOR フラッシュ パーティションにアクセス可能。
- UBI デバイスが正常に機能する。
- UBIFS ボリュームが Linux と U-Boot の両方の環境からマウントして使用できる。

この時点で、rootfs ボリュームは、次の章で説明する Linux カーネル イメージ、DTB、ルート ファイルシステム コンポーネントを格納する準備ができています。

6 Linux ブート アーティファクトの UBIFS への保存

第 5 章で作成した UBIFS ボリュームは、Linux と U-Boot の両方の環境で検証されました。この段階では、ボリュームにはファイルシステムの検証時に使用するファイルのみが含まれています。

Linux を OSPI NOR フラッシュから完全にブートするには、UBIFS ボリュームに Linux ルート ファイルシステム、カーネル イメージ、デバイス ツリー ブロブ (DTB) を書き込む必要があります。この章では、これらのブート アーティファクトの準備に使用する手順について説明します。

6.1 UBIFS ルート ファイルシステム ボリュームのマウント

まず、UBI デバイスを接続し、UBIFS ルート ファイルシステム ボリュームをマウントします。

```
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 BusyBox を使用したルート ファイルシステム構造の作成

ルート ファイルシステムに必要な最小ディレクトリ構造を作成します。

```
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
```

BusyBox をルート ファイルシステムにコピーします。

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

必要な共有ライブラリをコピーします。

```
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/
```

これらのライブラリは *ldd* コマンドを使用して識別できます。

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

注

このアプリケーション ノートでは、BusyBox ベースの最小ルート ファイルシステムを使用して複雑さを軽減し、Linux の重要な ブート プロセスに焦点を当てています。

6.3 BusyBox 実行の確認

次のように、ルート ファイルシステムに格納されているライブラリのみを使用して BusyBox が実行可能であることを確認します。

```
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
```

これにより、BusyBox とすべての必要な共有ライブラリが正常にロードされたことが確認されます。

6.4 BusyBox コマンド リンクの作成

シンボリック リンクを使用して基本的な Linux コマンドをシームレスに実行できます。

```
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 init スクリプトの作成

ルート ファイル システムの初期化スクリプトを作成します。

```
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
```

スクリプトを実行可能にします。

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

6.6 chroot を使用したルート ファイルシステムのテスト

chroot を使用して、ルート ファイルシステムが独立して動作できることを確認します。

```
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 Linux カーネル イメージと DTB のコピー

ブート ディレクトリを作成して、カーネル イメージとデバイス ツリーをコピーします。

```
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 UBIFS からの Linux ブートの構成

最小ルート ファイルシステムを作成して Linux カーネル イメージとデバイス ツリーを UBIFS ボリュームにコピーした後のステップは、これらのファイルをロードして OSPI NOR Flash から Linux を直接ブートするように U-Boot を構成することです。

次のセクションでは、ブート引数を構成し、UBIFS からカーネルとデバイス ツリーをロードし、Linux ブートが正常に行われたことを確認する方法について説明します。

7.1 ブート引数の構成

Linux では、ルート ファイルシステムの場所と実行する最初のプロセスを識別するためのブート引数が必要です。

ブート引数を以下のように構成します。

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

主要なパラメータを以下に示します。

表 7-1. 主要なパラメータ

パラメータ	説明
console=ttyS2,115200n8	Linux コンソール出力
earlycon=ns16550a、...	早期ブート デバッグ メッセージ
ubi.mtd=ospi.rootfs	OSPI rootfs パーティションから UBI デバイスを接続
root=ubi0:rootfs	rootfs UBI ボリュームをルート ファイルシステムとして使用
rootfstype=ubifs	ルート ファイルシステム タイプ
rw	ルート ファイルシステムの読み取り/書き込みをマウント
init=/init	カスタム初期化スクリプトを実行

7.2 UBIFS からカーネルと DTB をロード

OSPI rootfs パーティションを接続して UBIFS ボリュームをマウントします。

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

Linux カーネル イメージをロードして、デバイス ツリーをロードします。

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

7.3 Linux ブートのテスト

ロードしたカーネル イメージとデバイス ツリーを使用して Linux をブートします。

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

ブート中、Linux は UBI デバイスを接続して、UBIFS ルート ファイルシステム ボリュームをマウントする必要があります。

主なメッセージは以下のとおりです。

```
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
```

カーネルが初期化スクリプトを起動すると、以下のようなメッセージが表示されます。

```
Starting OSPI RootFS Initialization...

Booted from OSPI UBIFS RootFS
```

最後に、BusyBox シェルが利用可能になります。

```
~ #
```

これにより、OSPI NOR Flash 内にすべてが格納されているルート ファイルシステムを使用して Linux が正常に起動したことが確認されます。

- Linux カーネルが OSPI NOR フラッシュからロード済み
- デバイス ツリーが OSPI NOR フラッシュからロード済み
- ルート ファイルシステムが UBIFS からマウント済み
- /init が正常に実行済み
- BusyBox シェルが正常に開始済み

これらの観察から、Linux のブートが eMMC または SD ストレージ デバイスに依存しなくなったことがわかります。

7.4 再利用可能なブート コマンドの作成

ブートが検証されると、コマンドを再利用可能なブート シーケンスに結合できます。

```
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}
'
```

以下のような、完全なブート シーケンスを実行します。

```
run ospi_boot
```

これにより、Linux ブート手順がシングル U-Boot コマンドに減り、将来のテストと導入が簡単になります。

8 全面的な OSPI NOR フラッシュからの Linux ブート

8.1 U-Boot のブート シーケンスの自動化

前の章では、手動で入力した U-Boot コマンドを使用して Linux ブートについて説明しました。このアプローチは開発中およびデバッグ中には便利ですが、量産システムの場合は、通常、電源投入後にブート プロセスを自動的に開始する必要があります。

1 つの方法は、U-Boot 構成でデフォルトのブート コマンドを定義することです。これにより、U-Boot は UBIFS ボリュームを自動的にマウントして、OSPI NOR フラッシュから Linux カーネル イメージとデバイス ツリーをロードし、ユーザーとの相互作用なしで Linux を起動できます。デフォルトのブート コマンドは、U-Boot 構成ファイル configs/j721s2_evm_a72_defconfig に追加できます。

この例では以下の構成を使用できます。

```
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
```

U-Boot を再構築して、更新済みの u-boot.img を ospi.u-boot パーティションに書き込むと、電源投入時に Linux ブート シーケンス全体が自動的にボードで実行されます。

8.2 エンド ツー エンドのブート フロー

図 8-1 は OSPI NOR フラッシュからの Linux ブート シーケンス全体を示しています。

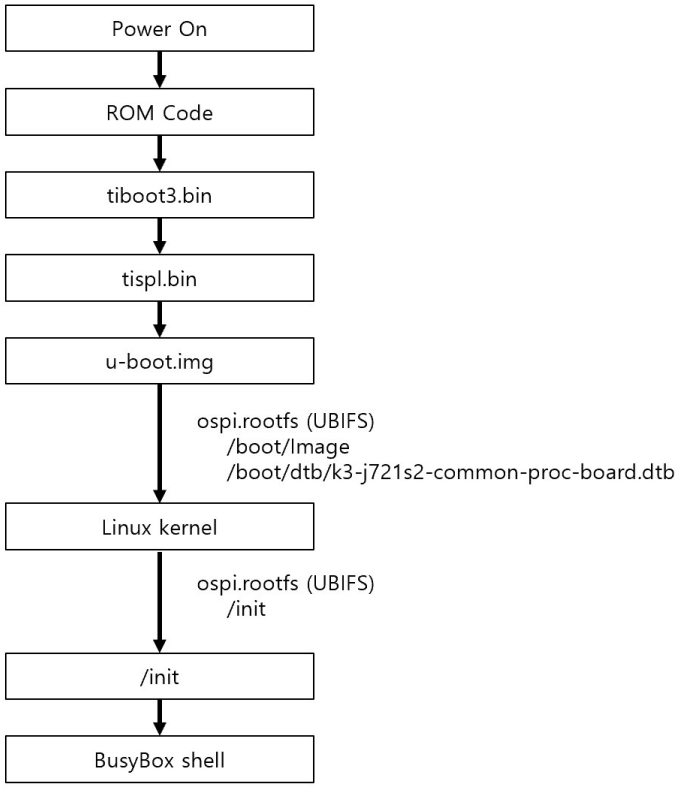

図 8-1. エンド ツー エンドのブート フロー

U-Boot が UBIFS ボリュームをマウントすると、OSPI rootfs パーティションから Linux カーネル イメージとデバイス ツリーがロードされます。次に Linux はルート ファイルシステムと同じ UBIFS ボリュームをマウントして、/init スクリプトを実行し BusyBox シェル環境を起動します。

9 まとめ

このアプリケーション ノートでは、UBIFS ベースのルート ファイルシステムを使用して、TI J721S2 プラットフォーム上で OSPI NOR フラッシュから Linux 全体をブートする方法について説明しました。

最小ルート ファイルシステムが、BusyBox、小さな一連のランタイム ライブラリ、カスタム初期化スクリプトを使用して作成されました。Linux のカーネル イメージとデバイス ツリーは同じ UBIFS ボリューム内に格納され、U-Boot は OSPI NOR フラッシュから必要なソフトウェア コンポーネントを直接ロードできます。完成されたシステムで、eMMC、SD カード、または他の外部ストレージ デバイスを必要とせずに、Linux が正常にブートされました。UBI と UBIFS を最小 BusyBox ベースのルート ファイルシステムと組み合わせることで、OSPI NOR Flash のみを使用したコンパクトで自己完結型の Linux ブート ソリューションを実装できます。このアプローチは、ストレージ アーキテクチャの簡素化、部品数の削減、またはリムーバブル ストレージ メディアなしの運用を必要とする組み込みシステムで特に役立ちます。

10 参考資料

- テキサス インスツルメンツ、TDA4VE、製品ページ。
- テキサス インスツルメンツ、『J721S2、TDA4AL、TDA4VL、TDA4VE、AM68A テクニカル リファレンス マニュアル』、テクニカル リファレンス マニュアル。
- テキサス インスツルメンツ、『TDA4VE TDA4AL TDA4VL Jacinto™ プロセッサ、シリコン リビジョン 1.0 データシート』、データシート。
- テキサス インスツルメンツ、PROCESSOR-SDK-J721S2、製品ページ。

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、 TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月