参考資料

Application Note

TI 製 Sitara デバイスによる PipeWire の有効化

Paresh Bhagat, Vishnu Singh

概要

PipeWire は、Linux アプリケーションにおけるオーディオおよびビデオ処理の標準となった最新の低レイテンシ マルチメディア フレームワークであり、統合アーキテクチャを使用したリアルタイム機能によるグラフベース処理を提供します。このドキュメントでは、デュアル / クワッドコアの Arm Cortex-A53 プロセッサを搭載したテキサス インスツルメンツ製 Sitara デバイス ファミリで PipeWire を有効にする方法を説明します。PipeWire のマルチプロセス アーキテクチャにより、複数のアプリケーションが競合やリソースの競合なしにマルチメディア コンテンツをシームレスに共有できます。

このアプリケーション ノートでは、オーディオ処理のための PipeWire 統合、構成、性能ベンチマーク測定で Yocto ベースの組込み Linux イメージをビルドするプロセスについて詳しく説明します。

ターゲット アプリケーション:

- プロフェッショナル オーディオ機器
- スマート スピーカ、サウンドバー
- 車載インフォテインメント
- マルチメディア機能を搭載した産業用 HMI
- オーディオ機能を搭載した IoT デバイス

サポート対象のプラットフォーム:

- SK-AM62B-P1
- SK-AM62-LP
- AUDIO-AM62D-EVM
- TMDS62LEVM
- SK-AM62-SIP
- SK-AM62P-LP
- SK-AM62A-LP

目次

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

Linux のオーディオ環境は歴史的に細分化されており、以下のように、多様なニーズに異なるサブシステムが対応する形となっていました。

- ALSA：低レベルのハードウェア アクセス
- PulseAudio：デスクトップ オーディオ
- JACK：プロフェッショナル オーディオ
- GStreamer：マルチメディア パイプライン

こうした乱立状態は、オーディオ開発者にとっての課題となっています。複数の API をサポートし、さまざまなオーディオ サブシステム間で複雑なやり取りを管理しなければならないからです。

PipeWire は、Linux システムにおけるオーディオおよびビデオ処理を根本から変えるために設計された、最新のマルチメディア フレームワークおよびグラフベースの処理アーキテクチャです。PulseAudio、JACK、およびその他のマルチメディア フレームワークの機能を、単一の効率的な処理エンジンに統合する統合設計として機能します。

1.1 主な特長：

- 統合マルチメディア フレームワーク：プロフェッショナル オーディオ (JACK)、コンシューマ オーディオ (PulseAudio)、ビデオ処理に単一のソリューションで対応。
- 低レイテンシのパフォーマンス。
- セキュリティ モデル：コンテナ化されたアプリケーションの組込みサンドボックスをサポート。
- アプリケーションによるマルチメディア コンテンツ共有を可能にするマルチプロセス アーキテクチャ。
- オーディオとビデオのリアルタイム マルチメディア処理。

1.2 基本概念：

1.2.1 PipeWire サーバー

サーバーは、グラフベースのマルチメディア処理エンジンを管理するコア デーモンです。このサーバーはオーディオ、ビデオ、または MIDI データがさまざまな処理コンポーネント間を流れるメディア グラフの作成および実行を担います。

1.2.2 PipeWire クライアント

クライアントとは、メディア ストリームを生成または消費するために PipeWire サーバーに接続するアプリケーションのことです。メディア グラフ内にノードを作成し、オーディオ データやビデオ データを送受信します。

1.2.3 セッション マネージャー

PipeWire は、デバイス ルーティングやポリシー決定を単独で処理することはありません。これらのタスクは、デバイスを監視し、ストリームを自動的に接続するセッション マネージャーが担います。このドキュメントでは、WirePlumber をセッション マネージャーとして使用しています。

1.2.4 ノード、ポート、およびリンク

PipeWire 処理グラフは、ノード、ポート、およびリンクで構成されます。ノードは処理要素を表し、ポートは入力または出力インターフェイスとして機能します。また、リンクはノード間のポートを接続し、メディア データがグラフ内で受け渡されるようにします。

1.3 PipeWire の主要コンポーネント

- IPC およびグラフ処理を実装する PipeWire デーモン。
- PipeWire デーモンのオブジェクトを管理する PipeWire セッション マネージャーの例。
- PipeWire デーモンの内部状態を調査し利用するための一連のプログラム。
- PipeWire アプリケーションとプラグインを開発するための PipeWire ライブラリ。
- PipeWire デーモンと PipeWire ライブラリの両方で使用される SPA (Simple Plugin API)。

2 Linux オーディオ スタック

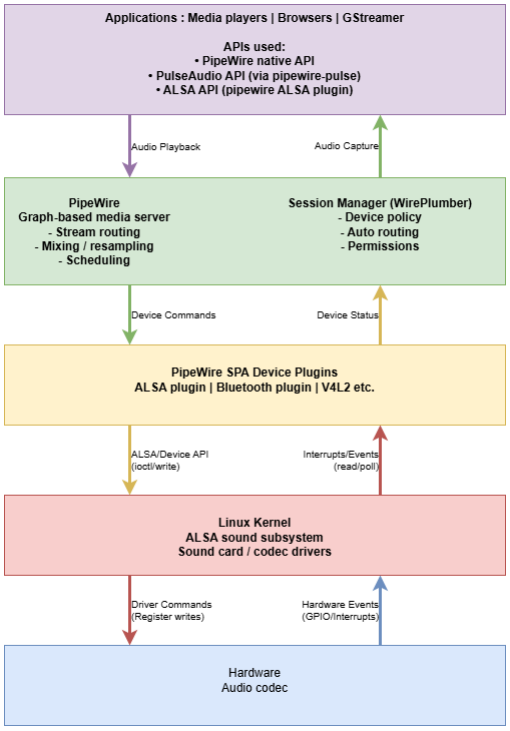

図 2-1. Linux オーディオ スタック

- **アプリケーション** (メディア プレーヤ、ブラウザ、GStreamer ベースのツールなど) は、オーディオ ストリームを生成し、ネイティブの PipeWire API または他のシステム (Jack、PulseAudio、ALSA) の互換性レイヤーを介して通信します。
- **PipeWire デーモン**は、これらのオーディオ ストリームを受信し、複数のソースからのオーディオのミキシング、ルーティング、スケジューリングを同時に扱う処理グラフを作成します。
- **セッション マネージャー** (WirePlumber) は、デバイスの選択、ストリーム ルーティン ルール、アプリケーションとハードウェア エンドポイント間の自動接続管理などのポリシー決定を適用します。
- **SPA (Simple Plugin API) プラグイン**は、ハードウェアを抽象化します。ALSA プラグインは、カーネルの ALSA サブシステムと特別にインターフェイスして、物理オーディオ デバイスにアクセスします。
- **カーネル ドライバ**は実際のハードウェア層 (オーディオ コーデック、サウンド カード、I2S バス、HDMI オーディオ インターフェイス) と通信し、物理的なオーディオ出力またはキャプチャ入力を生成します。

3 Yocto を使用した PipeWire サポート付き SDK イメージのビルド

このセクションでは、Yocto を介して PipeWire サポートを使用して書き込み可能なイメージを作成する手順を説明します。Yocto Project はオープン ソースのコラボレーション プロジェクトであり、開発者がハードウェア アーキテクチャにかかわりなく、Linux ベースのカスタム システムを製作するのに役立ちます。プロセッサ Linux SDK のビルドは、TI プラットフォームをターゲットとする OpenEmbedded と Yocto Project 向けの一連のレイヤーを提供する Arago プロジェクトをベースにしています。

3.1 ホストで Yocto ビルドを実行する手順

3.1.1 前提条件 (1 回限りのセットアップ)

推奨される Linux ディストリビューションは Ubuntu 22.04 です。次のコマンドを実行すると、Ubuntu Linux ディストリビューションに必要なツールをインストールできます。

```
$ sudo apt-get update

# Install packages required for builds
$ sudo apt-get -f -y install \
    git build-essential diffstat texinfo gawk chrpath socat doxygen \
    dos2unix python3 bison flex libssl-dev u-boot-tools mono-devel \
    mono-complete curl python3-distutils repo pseudo python3-sphinx \
    g++-multilib libc6-dev-i386 jq git-lfs pigz zstd liblz4-tool \
    cpio file lz4 debianutils iputils-ping python3-git python3-jinja2 \
    python3-subunit locales libacl1 unzip gcc python3-pip \
    python3-pexpect xz-utils wget \

$ sudo locale-gen en_US.UTF-8
```

Ubuntu は /bin/sh のデフォルト シェルが dash になっています。次のコマンドを実行して、bash を使用するように再構成します。

```
$ sudo dpkg-reconfigure dash
```

3.2 OE レイヤー セットアップのクローン作成

```
$ cd $HOME

$ git clone https://git.ti.com/git/arago-project/oe-layersetup.git tisdk

$ cd tisdk

$ ./oe-layertool-setup.sh -f configs/processor-sdk/processor-sdk-master-12.00.00.07.04-config.txt
```

3.3 PipeWire パッチのダウンロードと適用

イメージで PipeWire と WirePlumber のサポートを有効にするには、パッチが必要です。これらのパッチは、以下に適用する必要があります。

- meta-arago - Sitara デバイス向けの Arago ディストリビューション構成用 Yocto レイヤー
- meta-tisdk - Sitara デバイス向けの TI Foundational SDK 用 Yocto レイヤー

両方のレイヤーに必要なパッチについては、以下の表を参照してください。

表 3-1. Yocto PipeWire パッチ

パッチ番号	パッチ	説明
1	0001-recipes-multimedia-Add-pipewire-configuration-files.patch	8 チャネルおよび 2 チャネル オーディオ用の参照 PipeWire 構成ファイルを追加します。
2	0002-recipes-multimedia-Add-wireplumber-audio-configurati.patch	オーディオのデフォルト サービスを使用して WirePlumber 構成を追加します。
3	0003-recipes-core-arago-default-image-Add-pipewire-audio-.patch	arago-default-image で PipeWire オーディオ スタックを有効にします。

表 3-1. Yocto PipeWire パッチ (続き)

パッチ番号	パッチ	説明
4	0001-ti-apps-launcher-Remove-pulseaudio-service-dependenc.patch	OOB デモに PulseAudio が組み込まれている評価基板向けのイメージから、PulseAudio サービスを削除します。

パッチをダウンロードして適用する手順は次のとおりです。

```
$ cd $HOME

$ git clone https://github.com/TexasInstruments/Beyond-SDK.git -b main

$ cd $HOME/tisdk/sources/meta-arago

$ git am $HOME/Beyond-SDK/collaterals/appnotes/sdaa320-Enable_PipeWire_on_TI_Sitara_Devices/0001-recipes-multimedia-Add-pipewire-configuration-files.patch

$ git am $HOME/Beyond-SDK/collaterals/appnotes/sdaa320-Enable_PipeWire_on_TI_Sitara_Devices/0002-recipes-multimedia-Add-wireplumber-audio-configurati.patch

$ git am $HOME/Beyond-SDK/collaterals/appnotes/sdaa320-Enable_PipeWire_on_TI_Sitara_Devices/0003-recipes-core-arago-default-image-Add-pipewire-audio-.patch

$ cd $HOME/tisdk/sources/meta-tisdk

$ git am $HOME/Beyond-SDK/collaterals/appnotes/sdaa320-Enable_PipeWire_on_TI_Sitara_Devices/0001-ti-apps-launcher-Remove-pulseaudio-service-dependenc.patch
```

3.4 PipeWire イメージのビルド

次の最後のコマンドは tisdk-default-image をビルドします。これは、arago ファイルシステムと PipeWire サポートが有効なプロセッサ SDK イメージです。

```
$ cd $HOME/tisdk

$ cd build

$ . conf/setenv

# For RT (Real Time) Linux build
$ MACHINE=<machine> ARAGO_RT_ENABLE=1 bitbake -k tisdk-default-image

# For Non-RT Linux Build
$ MACHINE=<machine> bitbake -k tisdk-default-image
```

レイテンシが重視されるアプリケーションには、RT Linux ビルドを推奨します。

一方、MACHINE は次の値のいずれかです。

表 3-2. MACHINE の値

MACHINE	サポートされている評価基板
am62xx-evm	SK-AM62B-P1
am62xx-lp-evm	SK-AM62-LP
am62dxx-evm	AUDIO-AM62D-EVM
am62lxx-evm	TMDS62LEVM
am62xxsip-evm	SK-AM62-SIP
am62pxx-evm	SK-AM62P-LP
am62axx-evm	SK-AM62A-LP

結果として得られる WIC イメージは、**deploy-ti/images/<machine>/** ディレクトリに生成されます。

4 Sitara デバイス上での PipeWire のセットアップ

このガイドでは、例として SK-AM62B-P1、TMDS62LEVM、AUDIO-AM62D-EVM を使用し、SDK イメージのビルド方法と、その後の PipeWire の構成および使用方法について説明します。

4.1 ハードウェア

4.1.1 SK-AM62B-P1

SK-AM62B-P1 開発キットは、HDMI と LVDS 経由のデュアル ディスプレイをサポートするほか、包括的な産業用通信インターフェイス セットも備えており、HMI、PLC、オートメーションの各アプリケーションに適しています。

- SK-AM62B-P1
- Micro-SD カード (最小 32GB)
- USB Type-C 電源 (20W)
- USB-to-UART ケーブル
- 書き込みおよびコンソールへのアクセスのための Windows または Linux ホスト PC
- 3.5mm ジャック付きヘッドフォン

4.1.2 TMDS62LEVM

TMDS62LEVM 評価基板は、スケーラブルなパフォーマンス、豊富な組込み機能、広範な接続性、および電力 / 熱管理用ツールを提供する AM62L ファミリのアプリケーション プロセッサを使用した開発向けの低コストなプラットフォームを提供します。

- TMDS62LEVM
- Micro-SD カード (最小 32GB)
- USB Type-C 電源
- USB-to-UART ケーブル
- 書き込みおよびコンソールへのアクセスのための Windows または Linux ホスト PC
- 3.5mm ジャック付きヘッドフォン

4.1.3 AUDIO-AM62D-EVM

AUDIO-AM62D-EVM 評価基板 (EVM) は、低コストの拡張可能なプラットフォームであり、さまざまな使用事例にわたるマルチチャネル オーディオ アプリケーションのプロトタイプ製作と評価に適した設計を採用しています。

- AUDIO-AM62D-EVM
- Micro-SD カード (最小 32GB)
- USB Type-C 電源
- USB-to-UART ケーブル
- 書き込みおよびコンソールへのアクセスのための Windows または Linux ホスト PC
- オーディオ出力デバイス (スピーカ、TRS 互換)
- オーディオ入力デバイス (マイク、TRS 互換)

4.2 評価基板ブート モードの構成

4.2.1 SK-AM62B-P1

- 図 4-1 に、重要なケーブル接続、ポート、およびスイッチを示します。
- SD カード ブート モードの BOOTMODE スイッチの場所を書き留めます。
- 評価基板 SD カードのブート モード設定のセットアップ:
  - BOOTMODE [8 : 15] (SW2) = 0100 0000
  - BOOTMODE [0 : 7] (SW1) = 1100 0010
- 詳細については、「クイック スタート ガイド」を参照してください。

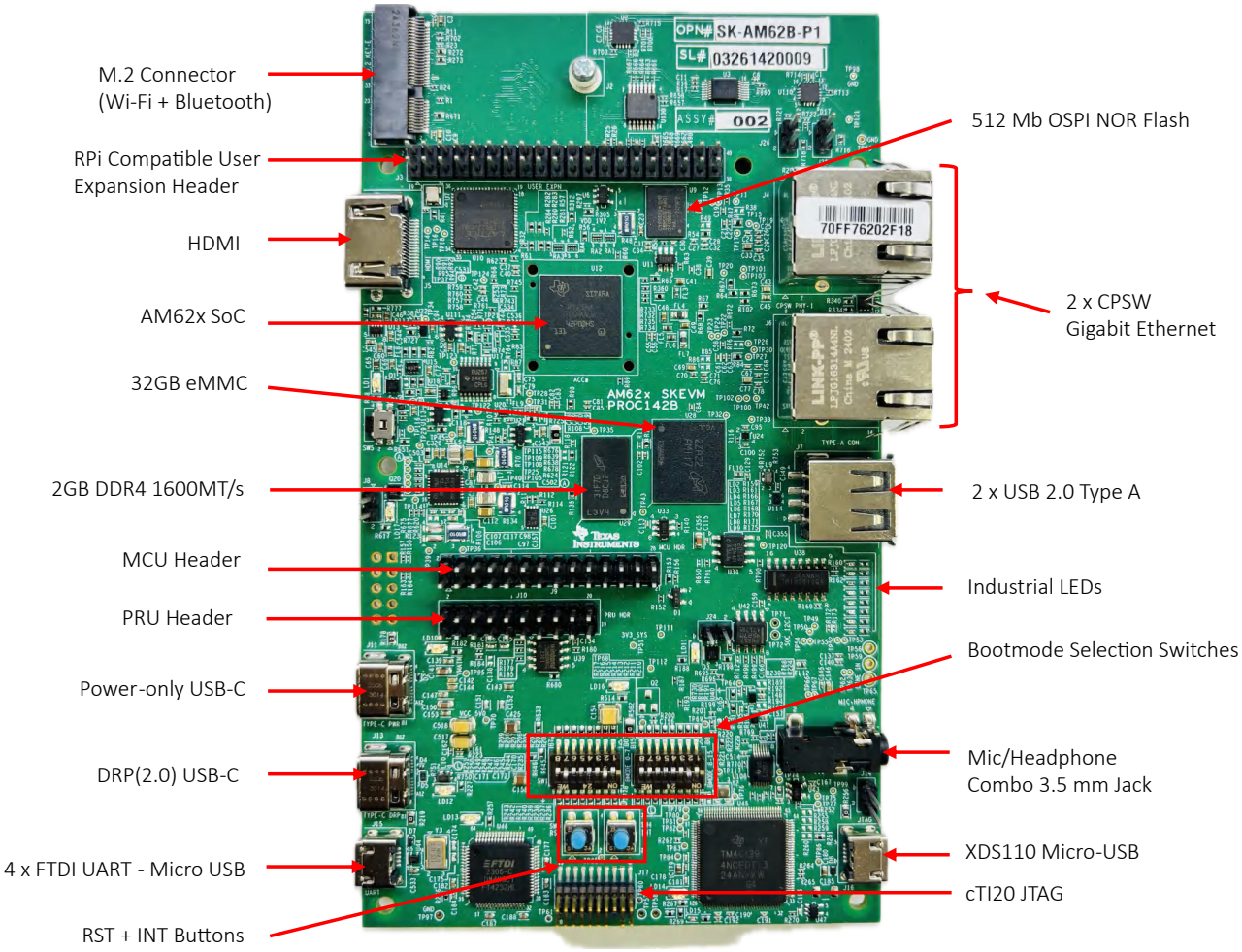

図 4-1. SK-AM62B-P1

4.2.2 TMDS62LEVM

- 図 4-2 に、重要なケーブル接続、ポート、およびスイッチを示します。
- SD カード ブート モードの BOOTMODE スイッチの場所を書き留めます。
- 評価基板 SD カードのブート モード設定のセットアップ:
  - BOOTMODE [8 : 11] (SW2) = 0100
  - BOOTMODE [11: 15] (SW3) = 0000
  - BOOTMODE [0 : 7] (SW4) = 1100 0010
- 詳細については、「クイック スタート ガイド」を参照してください。

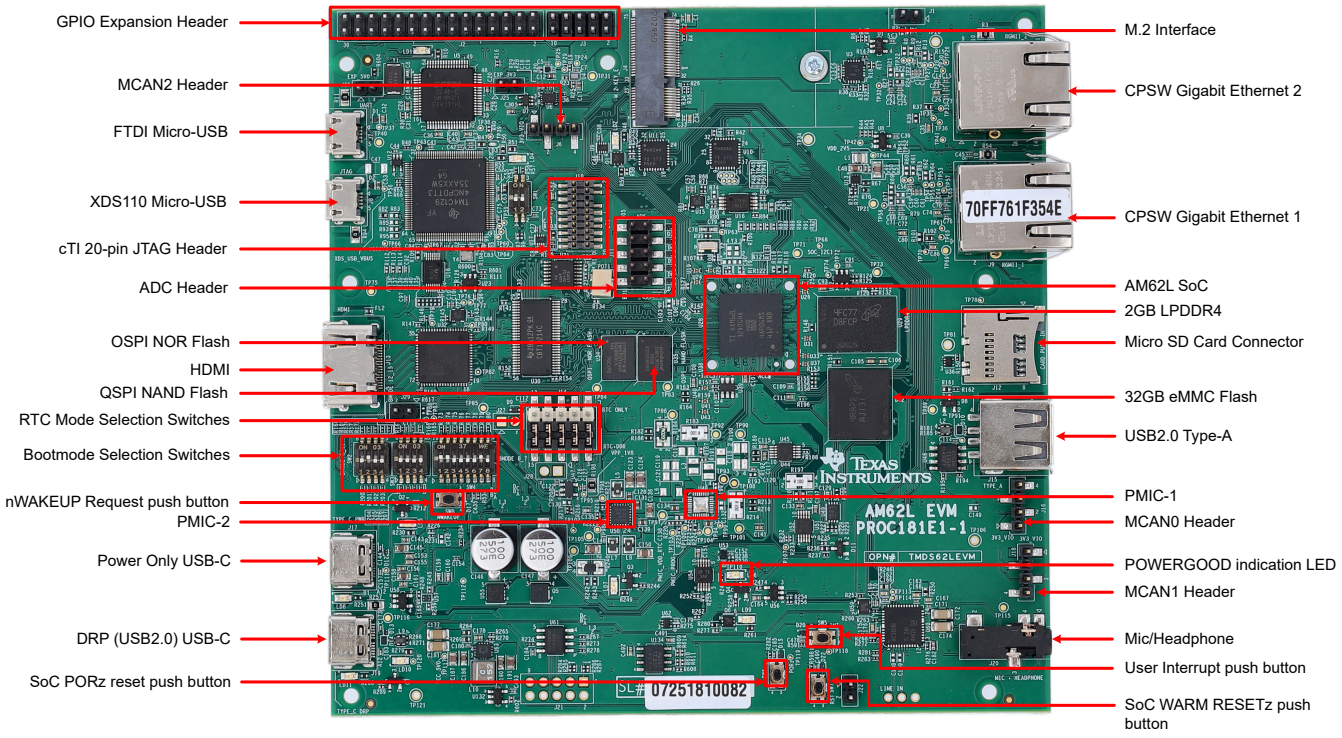

図 4-2. TMDS62LEVM

4.2.3 AUDIO-AM62D-EVM

- 下の図 4-3 に、重要なケーブル接続、ポート、スイッチを示します。
- SD カード ブート モードの BOOTMODE スイッチの場所を書き留めます。
- 評価基板 SD カードのブート モード設定のセットアップ:
  - BOOTMODE [8 : 15] (SW1) = 0100 0000
  - BOOTMODE [0 : 7] (SW2) = 1100 0010

- 詳細については、「クイック スタート ガイド」を参照してください。

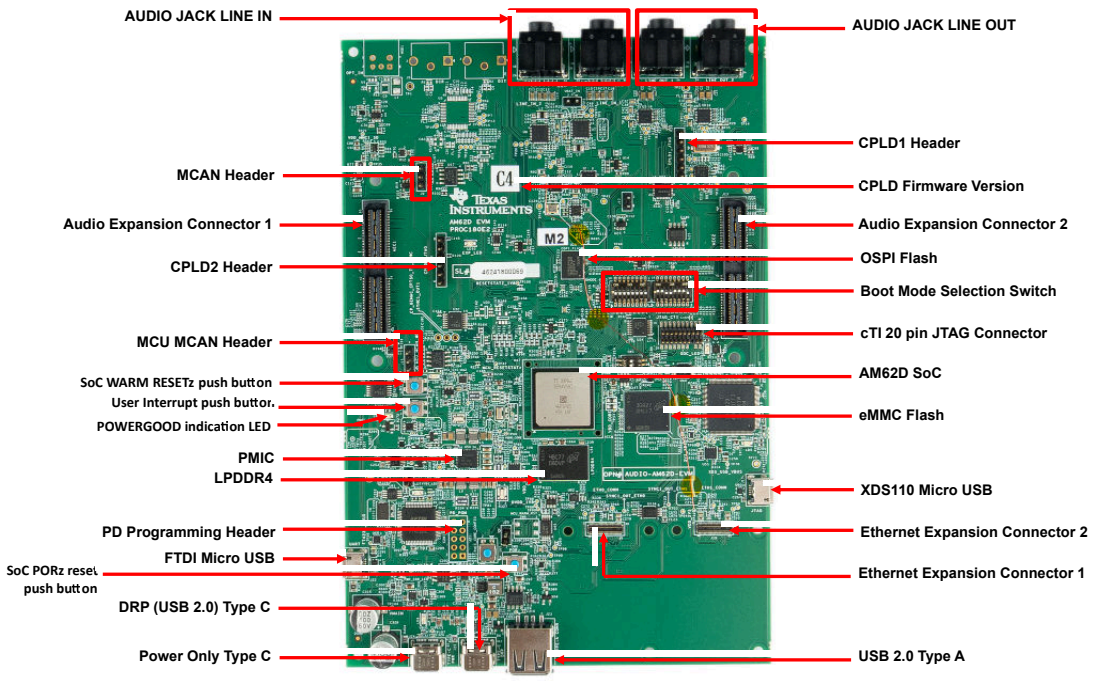

Figure 2-1. AM62D Audio EVM Top Side

SoC WARM RESETz push button

図 4-3. AUDIO-AM62D-EVM

4.3 UART コンソールのセットアップ

- UART を USB ケーブルで評価基板に接続します。
- ホスト マシン上で列挙されている UART COM ポートを特定します (Windows デバイス マネージャのポート (COM と LPT))。
- デバイス マネージャーの「ポート (COM と LPT)」の下に USB シリアル ポートが表示されていない場合は、FTDI の UART to USB ドライバをインストールしてください。
- ターミナル設定
  - ボーレート:115200
  - 8-N-1
- ターミナルを開き、デバイスがブートするのを待ちます。

4.4 SD カード イメージの書き込み

- PipeWire サポート付きのデフォルト WIC イメージは、**deploy-ti/<machine>/images** 内に **tisdk-default-image-<machine>-evm.rootfs.wic.xz** という名前で生成されます。
- Balena Etcher を使用して、WIC イメージを書き込みます
  - micro-SD カードを USB SD カードリーダーに挿入し、Etcher を起動します。
  - wic.xz イメージを選択します
  - SD カードを選択します
  - 「Flash」(書き込み) をクリックします。

4.5 SD カードを使用した評価基板のブート

- 基板上のブート モード ピンが SD カードのブート用であることを確認します。
- SD カードを SD カード スロットに挿入します。
- ホスト PC を USB Micro-B インターフェイスに接続して、システム コンソールにアクセスし、UART ログを表示します。
- 基板の電源を入れます。
- パスワードなしで「root」としてログインします。

```
Trying to boot from MMC2
Authentication passed
Authentication passed
Authentication passed
Authentication passed
Authentication passed
Starting ATF on ARM64 core...
.
.
.
Arago Project <machine> -
Arago 2025.01 <machine> -
<machine> login:
```

5 PipeWire の利用方法

5.1 サービス ステータスの確認

```
root@<machine>: systemctl status pipewire

root@<machine>: systemctl status wireplumber
```

5.2 PipeWire と Wireplumber の有効化

PipeWire と WirePlumber は、ユーザー スペース サービスとして動作するように設計されています。単一ユーザーの Arago イメージ (**root** のみ) では、システム サービスとして PipeWire を実行できます。ただし、マルチユーザー イメージ (例: 一部の Sitara プラットフォームにおける **weston**) では、マルチメディア アクセスを分離し、D-Bus セッション インスタンスや ALSA デバイス予約との競合を回避し、ユーザー セッション間での権限の不一致を防ぐために、ユーザーごとのセッションを有効にすることが強く推奨されます。

ブート時にサービスが自動的に開始されるようにします

```
root@<machine>: systemctl enable pipewire

root@<machine>: systemctl enable wireplumber
```

AUDIO-AM62D-EVM のセットアップを簡素化するため、WirePlumber のデフォルトのオーディオ デバイスを自動的に設定するための新しいサービスがパッチに含まれています。有効にするには、次のコマンドを使用します。

```
root@<machine>: systemctl enable set-audio-defaults
```

5.3 PipeWire と WirePlumber の起動

有効になっていない場合は、PipeWire と WirePlumber を起動します

```
root@<machine>: systemctl start pipewire

root@<machine>: systemctl start wireplumber

# For AUDIO-AM62D-EVM
root@<machine>: systemctl start set-audio-defaults
```

5.4 一般的な PipeWire コマンド

5.4.1 PipeWire サーバー内に現在存在する全オブジェクトの一覧

すべてのオブジェクトを一覧表示するには、**pw-cli list-objects** コマンドを使用します。

```
root@<machine>: pw-cli list-objects
id 0, type PipeWire:Interface:Core/4
object.serial = "0"
core.name = "pipewire-0"
id 1, type PipeWire:Interface:Module/3
object.serial = "1"
module.name = "libpipewire-module-rt"
id 2, type PipeWire:Interface:Module/3
object.serial = "2"
module.name = "libpipewire-module-protocol-native"
id 3, type PipeWire:Interface:SecurityContext/3
object.serial = "3"
id 4, type PipeWire:Interface:Module/3
object.serial = "4"
module.name = "libpipewire-module-profiler"
id 5, type PipeWire:Interface:Profiler/3
object.serial = "5"
```

5.4.2 ノードのみの一覧

すべてのノードを一覧表示するには、**pw-cli list-objects Node** を使用します。

```
root@<machine>: pw-cli list-objects Node
id 29, type PipeWire:Interface:Node/3
object.serial = "29"
factory.id = "11"
priority.driver = "200000"
node.name = "Dummy-Driver"
id 30, type PipeWire:Interface:Node/3
object.serial = "30"
factory.id = "11"
priority.driver = "190000"
node.name = "Freewheel-Driver"
id 31, type PipeWire:Interface:Node/3
object.serial = "31"
factory.id = "19"
node.description = "Audio Output"
node.name = "alsa_audio_sink"
media.class = "Audio/Sink"
id 32, type PipeWire:Interface:Node/3
object.serial = "32"
factory.id = "19"
node.description = "Audio Input"
node.name = "alsa_audio_source"
media.class = "Audio/Source"
```

5.4.3 特定オブジェクトの検査

pw-cli info <object-id> コマンドを使用して特定のオブジェクトを検査します。

```
root@<machine>: pw-cli info 31
id: 31
permissions: rwxm-
type: PipeWire:Interface:Node/3
* input ports: 0/0
* output ports: 8/129
* state: "suspended"
* properties:
* factory.name = "api.alsa.pcm.source"
* node.name = "alsa_audio_source"
* node.description = "Audio Input"
* media.class = "Audio/Source"
* api.alsa.period-size = "1024"
* node.driver = "true"
* api.alsa.disable-mmap = "false"
* api.alsa.disable-batch = "false"
* api.alsa.path = "hw:0,0"
* audio.rate = "48000"
* audio.channels = "8"
```

5.5 ステレオ オーディオの再生と録音

pw-play または **aplay** によるオーディオ再生:

```
root@<machine>: pw-play --target=alsa_audio_sink <path to wav file>

root@<machine>: aplay -r 48000 -f S32_LE -c 2 <path to wav file>
```

pw-record または **arecord** によるオーディオ録音:

```
root@<machine>: pw-record --target=alsa_audio_source record_stereo.wav

root@<machine>: arecord -r 48000 -f S32_LE -c 2 record_stereo.wav
```

6 構成

PipeWire セットアップは以下で構成されます。

- PipeWire デーモン
- セッション マネージャー (通常は WirePlumber)
- 互換性サーバー (PulseAudio、JACK)
- クライアント構成

これらのそれぞれには、JSON 構文を緩和した SPA JSON 形式を使用する独自の構成ファイルがあります。

表 6-1. 構成ファイル

ファイル	目的
pipewire.conf	PipeWire デーモンを構成します
client.conf	PipeWire クライアントを構成します
pipewire-pulse.conf	PulseAudio 互換サーバー
filter-chain.conf	オーディオ処理フィルタ

PipeWire では、メイン ファイルを直接修正するのではなく、特定の設定だけを無効にするドロップイン ファイルを使用することが推奨されます。以下のディレクトリの例を使用してください。

```
/etc/pipewire/pipewire.conf.d/
```

6.1 シンクおよびソースの構成

AUDIO-AM62D-EVM 用に追加されたシンクおよびソースという 2 つのリファレンス構成ファイルがあります。これは、3.5mm ジャックで 8 チャネルをサポートするためです。

必要なチャネルが 2 つだけの他の Sitara 評価基板では、追加構成は必要ありません。ただし、リファレンス構成は、必要に応じてサンプル レート、チャネル数、周期サイズ、その他のパラメータを変更するように調整できます。

90-pipewire-sink.conf

```
# PipeWire sink configuration for AM62D.
context.objects = [
    {
        factory = adapter
        args = {
            factory.name = api.alsa.pcm.sink
            node.name = "alsa_audio_sink"
            node.description = "Audio Output"
            media.class = "Audio/Sink"
            api.alsa.period-size = 1024
            api.alsa.headroom = 0
            api.alsa.disable-mmap = false
            api.alsa.disable-batch = false
            api.alsa.path = "hw:AM62D2EVM,0"
            audio.rate = 48000
            audio.channels = 8
            audio.position = [ FL FR FC LFE RL RR SL SR ]
        }
    }
]
```

91-pipewire-source.conf

```
# PipeWire source configuration for AM62D.
context.objects = [
    {
        factory = adapter
        args = {
            factory.name = api.alsa.pcm.source
            node.name = "alsa_audio_source"
            node.description = "Audio Input"
            media.class = "Audio/Source"
```

```
            api.alsa.period-size = 1024
            api.alsa.headroom = 0
            api.alsa.disable-mmap = false
            api.alsa.disable-batch = false
            api.alsa.path = "hw:AM62D2EVM,0"
            audio.rate = 48000
            audio.channels = 8
            audio.position = [ FL FR FC LFE RL RR SL SR ]
        }
    }
]
```

context.objects

PipeWire コンテキストで作成されたオブジェクトを定義するメイン構成配列。この配列内の各オブジェクトは、PipeWire グラフ内のノードになります。

factory = adapter

このオブジェクトが「adapter」ファクトリを使用して作成されることを指定します。PipeWire のアダプタは、異なる API 間をブリッジするために使用されます (この場合は ALSA から PipeWire へのブリッジ)。どちらの構成ファイルも、ALSA ハードウェアを PipeWire ノードにブリッジするために adapter ファクトリを使用します。

factory.name

方向 (出力または入力) を決定します。

node.name

内部 PipeWire ノード識別子。再生用に alsa_audio_sink を作成し、キャプチャ用に alsa_audio_source を作成します。

node.description

オーディオ アプリケーションに表示される、人間が判読できる名前。

media.class

PipeWire のメディア分類では、再生用には「Audio/Sink」、録音用には「Audio/Source」が使用されます。

api.alsa.path

ハードウェアへの直接アクセス用。オーディオ ハードウェアにアクセスできるのは PipeWire のみで、ALSA アプリケーションは PipeWire を介してアクセスする必要があります。

audio.channels

AUDIO-AM62D-EVM の場合、8 チャネル オーディオの入力と出力の両方を構成します。

これらの構成により、PipeWire のオーディオ グラフに次の 2 つの基本ノードが作成されます。

- シンク ノード:8 チャネル オーディオ再生用ターミナル エンドポイント
- ソース ノード:8 チャネル オーディオ キャプチャの開始点

詳細については、「Alsa の構成」を参照してください。

6.2 WirePlumber の構成

wpctl status コマンドを使用して、使用可能なすべてのオーディオ シンクおよびソースを一覧表示します。

```
root@<machine>: wpctl status
PipeWire 'pipewire-0' [1.6.0, root@am62dxx-evm, cookie:3333499771]
└─ Clients:
34. WirePlumber [1.6.0, root@am62dxx-evm, pid:9716]
58. WirePlumber [export] [1.6.0, root@am62dxx-evm, pid:9716]
94. wpctl [1.6.0, root@am62dxx-evm, pid:9753]
Audio
├─ Devices:
│  59. Built-in Audio [alsa]
│
├─ Sinks:
│  * 31. Audio Output [vol: 1.00]
│  68. Built-in Audio Stereo [vol: 0.40]
│
├─ Sources:
│  * 32. Audio Input [vol: 1.00]
│  69. Built-in Audio Stereo [vol: 1.00]
│
├─ Filters:
│
└─ Streams:
Video
├─ Devices:
│
├─ Sinks:
│
├─ Sources:
│
├─ Filters:
│
└─ Streams:
Settings
└─ Default Configured Devices:
0. Audio/Sink alsa_audio_sink
1. Audio/Source alsa_audio_source
```

wpctl inspect <id> を使用して、指定したオブジェクトに関する情報を表示します。

```
root@<machine>: wpctl inspect 31
```

シンクとソースの ID 番号を使用して、デフォルトのソースを手動で設定します。

```
root@<machine>: wpctl set-default 30

root@<machine>: wpctl set-default 31
```

WirePlumber の変更により、AUDIO-AM62D-EVM 向けの自動オーディオ デバイス構成システムが導入されました。これは、2 つの重要なファイルを使用しています。

- ***set-audio-defaults.sh***

このスクリプトは、WirePlumber の準備が完了するまで最大 30 秒待機する初期化シーケンスを実装した後、PipeWire のコマンド ライン ツール (pw-cli および wpctl) を使用して、PipeWire 構成ファイルで定義されている alsa_audio_sink および alsa_audio_source ノードを検索し、システムのデフォルトのオーディオ デバイスとして明示的に設定します。PipeWire は構成ファイルに基づいてオーディオ ノードを作成しますが、アプリケーションが使用するデフォルト デバイスとしてそれらを自動的に指定することはないため、この自動化が必要になります。

- ***set-audio-defaults.service***

付随する systemd サービスにより、WirePlumber の起動後に set-audio-defaults.sh スクリプトが自動的に実行されます。

デフォルトでは、set-audio-defaults.service により、AUDIO-AM62D-EVM は alsa_audio_sink と alsa_audio_source を使用します。wpctl コマンドを使用すると、オーディオのルーティング先を別のデバイス (例：USB) に変更できます。

7 性能ベンチマーク

テスト設定は次のとおりです:

- ハードウェア
  - SK-AM62B-P1
  - TMDS62LEVM
  - AUDIO-AM62D-EVM
- カーネル - ti-linux-6.18.y - 12.00.00.07
- Yocto - 12.00.00.07.04

PulseAudio と PipeWire の平均性能を測定して比較するため、複数のオーディオ アプリケーションを同時に起動します。

PulseAudio は、Sitara の全プラットフォーム向けパッケージで供給されています。本ドキュメントに記載されているパッチにより、サービス ファイルはパッケージング対象から除外されます。PulseAudio をバックグラウンド プロセスとして起動するには、次のコマンドを使用します。

```
root@<machine>: pulseaudio --daemonize
```

次のコマンドを使用して PulseAudio を停止します

```
root@<machine>: pulseaudio --kill
```

7.1 レイテンシ

このテストでは、PulseAudio および PipeWire の再生レイテンシを測定します。

PulseAudio のレイテンシは **pactl** で報告されるシンク レイテンシを使用して測定しました。このシンク レイテンシは、オーディオ出力で観測された実効的な再生遅延を反映しています。

```
root@<machine>: pactl list sinks | grep Latency
Latency: 1370744 usec, configured 1837500 usec
```

PipeWire のレイテンシは **pw-top** コマンドから取得したアクティブなストリーム ノードとシンク ノードのバッファ期間 (クォンタム) の合計として推定しました。これは、アプリケーションとデバイスによるバッファリングを表します。

```
root@<machine>: pw-top
S ID QUANT RATE WAIT BUSY W/Q B/Q ERR FORMAT NAME
S 29 0 0 --- --- --- --- 0 Dummy-Driver
S 30 0 0 --- --- --- --- 0 Freewheel-Driver
R 31 2048 48000 19.1us 504.7us 0.00 0.01 0 S32LE 8 48000 alsa_audio_sink
R 77 4800 48000 28.9us 125.1us 0.00 0.00 0 S16LE 8 48000 = pw-play
S 32 0 0 --- --- --- --- 0 alsa_audio_source
S 68 0 0 --- --- --- --- 0 alsa_output.platform-sound.stereo-fallback
S 69 0 0 --- --- --- --- 0 alsa_input.platform-sound.stereo-fallback
```

次の式を使用して、クォンタムおよびレートからのレイテンシを計算します。

$$\textbf{Latency(ms)} = \frac{\textbf{quantum}}{\textbf{rate}} \times \textbf{1000} \quad (1)$$

表 7-1. デフォルトのレイテンシ

デバイス	オーディオ サーバー	レイテンシ (ms)
SK-AM62B-P1	PulseAudio	約 1837ms
	PipeWire	約 143ms
TMDS62LEVM	PulseAudio	約 1837ms
	PipeWire	約 143ms
AUDIO-AM62D-EVM	PulseAudio	約 1837ms
	PipeWire	約 143ms

これらのレイテンシ値は、クォンタム、クロック レート、フラグメント数、フラグメント サイズなどの設定を変更することで、さらに削減できます。

7.2 CPU とメモリの使用状況

このテストでは、複数のオーディオ ストリームが同時にオーディオ サーバーを使用する場合の CPU 使用率とメモリ使用量を測定します。PULSE_LATENCY_MSEC を変更し、143msec (PipeWire のレイテンシ) に近いレイテンシで動作するように PulseAudio を設定します。

```
root@<machine>: export PULSE_LATENCY_MSEC=325

root@<machine>: pactl list sinks | grep Latency
Latency: 141702 usec, configured 142500 usec
```

PulseAudio では、レイテンシは厳密な設定ではなくターゲットとして扱われるため、PULSE_LATENCY_MSEC を使用して正確な値に設定することはできません。その結果、観測されたレイテンシは要求された値と大きく異なる可能性があります。実際のレイテンシは、内部バッファ フラグメンテーション、ハードウェアの制約、スケジューラの動作によって決まります。

レイテンシを制御する方法は他にもあります。例えば、フラグメント、フラグメント サイズなど、レイテンシに影響を与える変数を直接変更する方法もあります。

詳細については、Pulseaudio のドキュメントを参照してください。

表 7-2. CPU の負荷 (同一レイテンシ)

デバイス	オーディオ サーバー	CPU 使用率 (平均)
SK-AM62B-P1	PulseAudio	約 5%
	PipeWire	約 1%
TMDS62LEVM	PulseAudio	約 6%
	PipeWire	約 1%
AUDIO-AM62D-EVM	PulseAudio	約 5%
	PipeWire	約 1%

表 7-3. メモリ使用量 (同一レイテンシ)

デバイス	オーディオ サーバー	メモリ使用状況 (平均)
SK-AM62B-P1	PulseAudio	約 22,500 KB
	PipeWire	約 46,570KB (Wireplumber- ～ 22910KB)
TMDS62LEVM	PulseAudio	約 25,000KB
	PipeWire	約 33,400KB (Wireplumber- 約 30,190KB)
AUDIO-AM62D-EVM	PulseAudio	約 21,600 KB
	PipeWire	約 55,000KB (Wireplumber- 約 22,990KB)

注

PipeWire は、コア デーモンと WirePlumber セッション マネージャーが独立して動作するマルチプロセス設計のため、より多くのメモリを使用します。独立して動作するため、障害の分離が可能になります。WirePlumber がクラッシュした場合、オーディオの再生は中断されずに継続されます。PulseAudio の単一プロセス設計は、メモリ効率に優れていますが、コンポーネント障害に対する復元力で劣ります。

7.3 リサンプリング時の CPU およびメモリ使用状況

このテストでは、オーディオ サーバーがサンプル レート変換を実行し、同じレイテンシで設定した場合に発生する CPU のオーバーヘッドとメモリ使用状況を測定します。このテストでは、リサンプリングの効率を評価します。PulseAudio と PipeWire の両方とも、オーディオを 44.1kHz にリサンプリングする (ネイティブは 48kHz) 構成になっています。

PulseAudio の場合は、次のコマンドを使用して 44.1kHz にリサンプリングします。

```
root@<machine>: echo "default-sample-rate = 44100" >> /etc/pulse/daemon.conf
root@<machine>: echo "alternate-sample-rate = 44100" >> /etc/pulse/daemon.conf
```

PipeWire の場合は、次のコマンドを使用して 44.1kHz にリサンプリングします (現在のセッションの場合のみ)。

```
root@<machine>: pw-metadata -n settings 0 clock.force-rate 44100
```

永続的な構成にしたい場合は、次の内容を含む新しいカスタム **/etc/pipewire/pipewire.conf.d/99-custom.conf** を作成します

```
context.properties = {
    default.clock.rate = 44100
    default.clock.allowed-rates = [ 44100 ]
}
```

表 7-4. リサンプリングによる CPU 使用状況

デバイス	オーディオ サーバー	CPU 使用率 (平均)
SK-AM62B-P1	PulseAudio	約 24%
	PipeWire	約 3%
TMDS62LEVM	PulseAudio	約 29%
	PipeWire	約 3%
AUDIO-AM62D-EVM	PulseAudio	約 23%
	PipeWire	約 3%

表 7-5. リサンプリングによるメモリ使用状況

デバイス	オーディオ サーバー	メモリ使用状況 (平均)
SK-AM62B-P1	PulseAudio	約 21,600 KB
	PipeWire	約 52,750 KB (Wireplumber-40000)
TMDS62LEVM	PulseAudio	約 25,500 KB
	PipeWire	約 33,310 KB (Wireplumber-33800)
AUDIO-AM62D-EVM	PulseAudio	約 21,500 KB
	PipeWire	約 32,250 KB (Wireplumber-23000)

7.4 観察事項

テストされたワークロード (5 つの同時オーディオ ストリーム) は比較的軽量であり、CPU、マルチコア スケーリング、DDR の帯域幅に大きなストレスを与えないため、ベンチマークの結果はほとんどプラットフォームに依存しません。そのため、CPU 周波数、コア数、DDR 速度の違いが CPU 使用率、レイテンシ、およびメモリ占有率に与える影響は最小限に抑えられます。観察された小さな違いは、スケジューラのタイミング、バックグラウンド アクティビティ、キャッシュなどの要因によって、通常の実行間の変動の範囲内である可能性があります。

8 まとめ

このドキュメントでは、Yocto イメージのビルドからオーディオ フレームワークの構成とベンチマーク測定に至るまで、TI の Sitara デバイスで PipeWire を使用する方法を包括的に紹介しました。SK-AM62B-P1、TMDS62LEVM、および AUDIO-AM62D-EVM を基準プラットフォームとしてベンチマークを測定し、組込みアプリケーションにおける低レイテンシで高性能なオーディオ設計としての PipeWire の有効性を実証しました。

9 参考資料

1. PipeWire、PipeWire ドキュメント、Web ページ。
2. PulseAudio、PulseAudio ドキュメント、Web ページ。
3. テキサス・インスツルメンツ、SK-AM62B-P1、製品ページ
4. テキサス インスツルメンツ、『SK-AM62B-P1 プロセッサ SDK ドキュメント』、ソフトウェア開発ガイド
5. テキサス インスツルメンツ、『SK-AM62B-P1 クイック スタート ガイド』、クイック スタート ガイド。
6. テキサス・インスツルメンツ、TMDS62LEVM、製品ページ
7. テキサス インスツルメンツ、『TMDS62LEVM プロセッサ SDK ドキュメント』、ソフトウェア開発ガイド。
8. テキサス インスツルメンツ、『TMDS62LEVM クイック スタート ガイド』、クイック スタート ガイド。
9. テキサス・インスツルメンツ、AUDIO-AM62D-EVM、製品ページ
10. テキサス インスツルメンツ、『AUDIO-AM62D-EVM プロセッサ SDK ドキュメント』、ソフトウェア開発ガイド。
11. テキサス インスツルメンツ、『AUDIO-AM62D-EVM クイック スタート ガイド』、クイック スタート ガイド。

10 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、テキサス・インスツルメンツ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 テキサス・インスツルメンツ製品の選定、(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 テキサス・インスツルメンツ製品を使用するアプリケーションの開発の目的でのみ、テキサス・インスツルメンツ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。テキサス・インスツルメンツ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 製品は、ti.com に掲載されているか、あるいは当該 TI 製品と併せて提供される、TI の販売規約、TI の『一般的な品質に関するガイドライン』、またはその他の適用される規定に従って提供されます。テキサス・インスツルメンツがこれらのリソースを提供することは、適用される テキサス・インスツルメンツ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 :2026 年 6 月

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