Application Note

Enable PipeWire on TI Sitara Devices

Paresh Bhagat, Vishnu Singh

ABSTRACT

PipeWire is a modern, low-latency multimedia framework that has become the standard for audio and video handling in Linux applications, providing graph-based processing with real-time capabilities using unified architecture. This document demonstrates how to enable PipeWire on Texas Instruments' Sitara family of devices, which feature dual/quad-core Arm Cortex-A53 processors. PipeWire's multi-process architecture allows multiple applications to seamlessly share multimedia content without conflicts or resource contention.

This application note details the process for building a Yocto-based embedded Linux image with PipeWire integration, configuration and performance benchmarking for audio processing.

Target Applications:

- Professional audio equipment
- Smart speakers and soundbars
- Automotive infotainment
- Industrial HMI with multimedia
- IoT devices with audio capabilities

Supported platforms:

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

Table of Contents

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

The Linux audio landscape has historically been fragmented, with different subsystems serving different needs:

- ALSA for low-level hardware access
- PulseAudio for desktop audio
- JACK for professional audio
- GStreamer for multimedia pipelines

This fragmentation creates challenges for audio developers who must support multiple APIs and manage complex routing between different audio subsystems.

PipeWire is a modern multimedia framework and graph-based processing architecture designed to revolutionize audio and video handling in Linux systems. It serves as a unified design that consolidates the functionality of PulseAudio, JACK, and other multimedia frameworks into a single, efficient processing engine.

1.1 Key Highlights:

- Unified Multimedia Framework: Single solution for professional audio (JACK), consumer audio (PulseAudio), and video processing.
- Low Latency Performance.
- Security Model: Built-in sandboxing support for containerized applications.
- Multi-process architecture to let applications share multimedia content.
- Real-time multimedia processing on audio and video.

1.2 Basic Concepts:

1.2.1 PipeWire Server

The server is the core daemon that manages a graph-based multimedia processing engine. It handles the creation and execution of the media graph where audio, video, or MIDI data flows between different processing components.

1.2.2 PipeWire Clients

Clients are applications that connect to the PipeWire server to produce or consume media streams. They create nodes in the media graph to send or receive audio or video data.

1.2.3 Session Manager

PipeWire does not handle device routing or policy decisions by itself. These tasks are managed by a session manager, which monitors devices and automatically connects streams. In this document, WirePlumber is used as the session manager.

1.2.4 Nodes, Ports, and Links

The PipeWire processing graph consists of nodes, ports, and links. A node represents a processing element, ports act as input or output interfaces, and links connect ports between nodes to allow media data to flow through the graph.

1.3 PipeWire Main Components

- A PipeWire Daemon that implements the IPC and graph processing.
- An example PipeWire Session Manager that manages objects in the PipeWire Daemon.
- A set of Programs to introspect and use the PipeWire Daemon.
- A PipeWire Library to develop PipeWire applications and plugins.
- The SPA (Simple Plugin API) used by both the PipeWire Daemon and in the PipeWire Library.

2 Linux Audio Stack

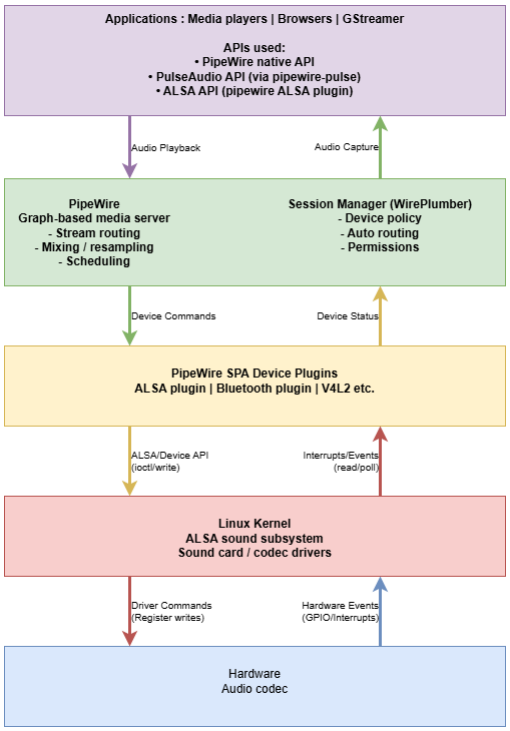

Figure 2-1. Linux Audio Stack

- **Applications** (media players, browsers, GStreamer-based tools etc.) generate audio streams and communicate through either the native PipeWire API or compatibility layers for other systems (Jack, PulseAudio, ALSA).
- **PipeWire daemon** receives these audio streams and constructs a processing graph to handle mixing, routing, and scheduling of audio from multiple sources simultaneously.
- **Session manager** (WirePlumber) applies policy decisions including device selection, stream routing rules, and automatic connection management between applications and hardware endpoints.
- **SPA (Simple Plugin API) plugins** provide hardware abstraction. The ALSA plugin specifically interfaces with the kernel's ALSA subsystem to access physical audio devices.
- **Kernel drivers** communicate with the actual hardware layer (audio codecs, sound cards, I2S buses, HDMI audio interfaces) to produce physical audio output or capture input.

3 Build SDK Image with PipeWire Support via Yocto

This section provides the steps to build a flashable image with PipeWire support via Yocto. The Yocto Project is an open-source collaboration project that helps developers create custom Linux-based systems regardless of the hardware architecture. The Processor Linux SDK build is based on the Arago project which provides a set of layers for OpenEmbedded and the Yocto Project targeting TI platforms.

3.1 Steps to Run Yocto Builds on Host

3.1.1 Prerequisites (One-time setup)

The recommended Linux distribution is Ubuntu 22.04. The following commands install the required tools on the Ubuntu Linux distribution.

```
$ 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
```

By default, Ubuntu uses *dash* as the default shell for /bin/sh. Reconfigure to use bash by running the following command:

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

3.2 Clone the oe-layer Setup

```
$ 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 Download and Apply PipeWire Patches

Patches are required to enable PipeWire and WirePlumber support in the image. These patches needs to be applied to:

- meta-arago - Yocto layer for Arago distribution configuration for Sitara devices.
- meta-tisdk - Yocto layer for TI Foundational SDK for Sitara devices.

The patches required for both the layers are described in table:

Table 3-1. Yocto PipeWire patches

Patch Number	Patch	Description
1	0001-recipes-multimedia-Add-pipewire-configuration-files.patch	Add reference PipeWire configuration files for 8-channel and 2-channel audio.
2	0002-recipes-multimedia-Add-wireplumber-audio-configurati.patch	Add WirePlumber configuration with audio defaults service.
3	0003-recipes-core-arago-default-image-Add-pipewire-audio-.patch	Enable PipeWire audio stack in arago-default-image.
4	0001-ti-apps-launcher-Remove-pulseaudio-service-dependenc.patch	Remove PulseAudio service from image for EVMs having PulseAudio integration in the OOB demo.

Following are the steps to download and apply the patches.

```
$ 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 Build PipeWire Image

The final command below builds the tisdk-default-image, which is the Processor SDK image with arago filesystem and PipeWire support enabled.

```
$ 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
```

RT Linux build is recommended for latency-sensitive applications.

Whereas MACHINE is one of the following values:

Table 3-2. MACHINE values

MACHINE	Supported EVMs
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

The resulting wic image is generated in ***deploy-ti/images/<machine>/*** directory.

4 Setup PipeWire on Sitara Devices

This guide uses the SK-AM62B-P1, TMDS62LEVM and AUDIO-AM62D-EVM as an example to demonstrate building an SDK image and subsequently configuring and using PipeWire.

4.1 Hardware

4.1.1 SK-AM62B-P1

The SK-AM62B-P1 development kit supports dual displays via HDMI and LVDS alongside a comprehensive set of industrial communication interfaces, making it designed for HMI, PLC, and automation applications.

- SK-AM62B-P1
- Micro-SD Card (minimum 32GB)
- USB Type-C power supply (20W)
- USB-to-UART cable
- Windows or Linux host PC for flashing and console access
- Headphone with a 3.5mm jack

4.1.2 TMDS62LEVM

The TMDS62LEVM evaluation module provides a low-cost platform for developing with the AM62L family of application processors offering scalable performance, rich embedded features, extensive connectivity, and tools for power and thermal management.

- TMDS62LEVM
- Micro-SD Card (minimum 32GB)
- USB Type-C power supply
- USB-to-UART cable
- Windows or Linux host PC for flashing and console access
- Headphone with a 3.5mm jack

4.1.3 AUDIO-AM62D-EVM

The AUDIO-AM62D-EVM evaluation module (EVM) is a low-cost expandable platform designed for developers to prototype and evaluate multi-channel audio applications across various use cases.

- AUDIO-AM62D-EVM
- Micro-SD Card (minimum 32GB)
- USB Type-C power supply
- USB-to-UART cable
- Windows or Linux host PC for flashing and console access
- Audio Output device (speaker, TRS compatible)
- Audio Input device (microphone, TRS compatible)

4.2 Configure EVM Boot Mode

4.2.1 SK-AM62B-P1

- Figure 4-1 shows some important cable connections, ports and switches.
- Take note of the location of the "BOOTMODE" switch for SD card boot mode.
- Setup EVM SD card boot mode setting:
  - BOOTMODE [8 : 15] (SW2) = 0100 0000
  - BOOTMODE [0 : 7] (SW1) = 1100 0010
- See Quick Start Guide for more details.

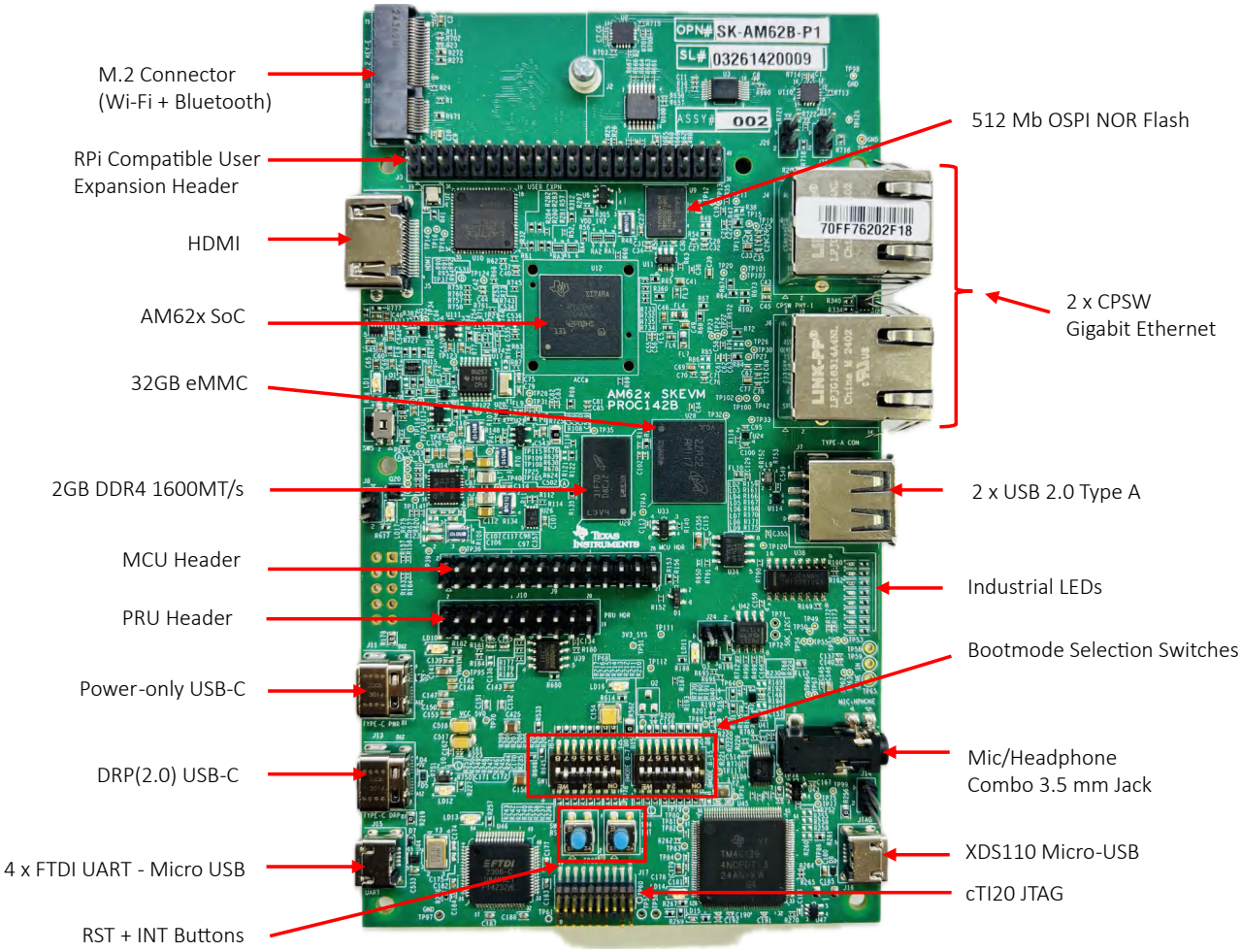

Figure 4-1. SK-AM62B-P1

4.2.2 TMDS62LEVM

- Figure 4-2 shows some important cable connections, ports and switches.
- Take note of the location of the *BOOTMODE* switch for SD card boot mode.
- Setup EVM SD card boot mode setting:
  - BOOTMODE [8 : 11] (SW2) = 0100
  - BOOTMODE [11: 15] (SW3) = 0000
  - BOOTMODE [0 : 7] (SW4) = 1100 0010
- See Quick Start Guide for more details.

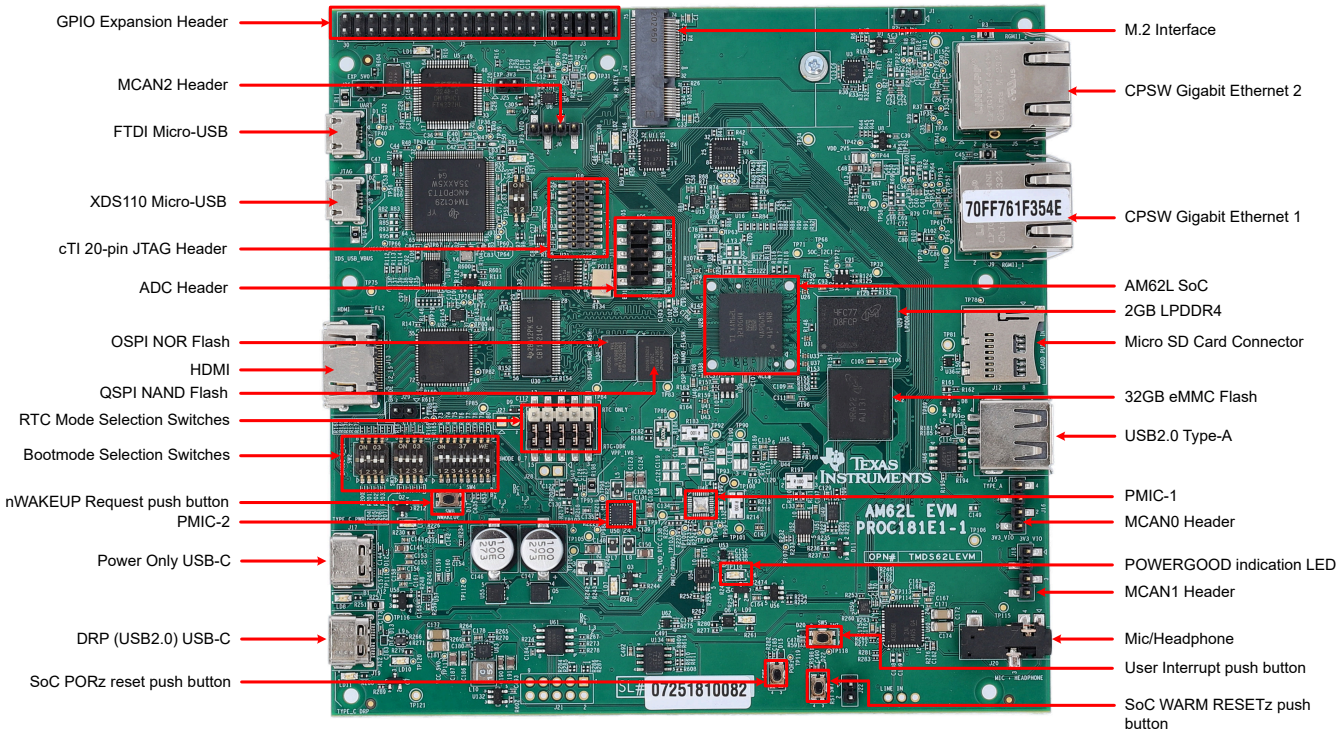

Figure 4-2. TMDS62LEVM

4.2.3 AUDIO-AM62D-EVM

- Figure 4-3 below shows some important cable connections, ports and switches.
- Take note of the location of the "BOOTMODE" switch for SD card boot mode.
- Setup EVM SD card boot mode setting:
  - BOOTMODE [8 : 15] (SW1) = 0100 0000
  - BOOTMODE [0 : 7] (SW2) = 1100 0010

- Refer Quick Start Guide for more details.

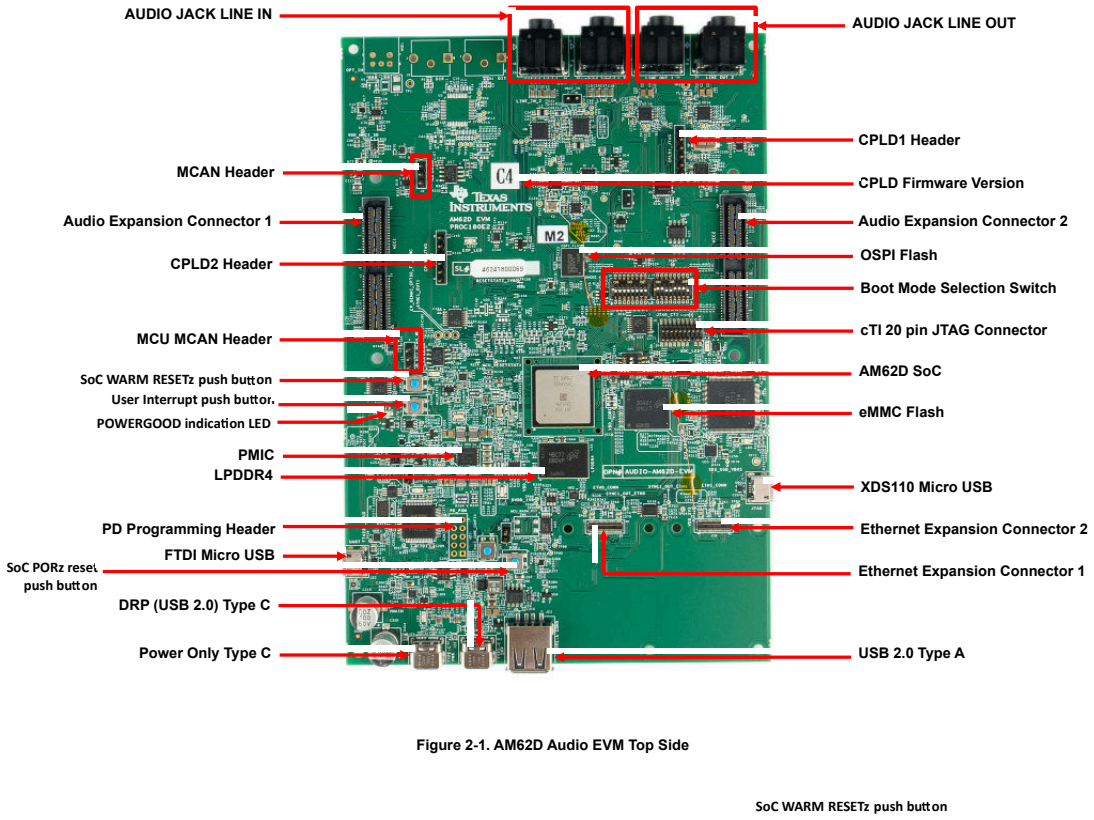

Figure 2-1. AM62D Audio EVM Top Side

SoC WARM RESETz push button

Figure 4-3. AUDIO-AM62D-EVM

4.3 UART Console Setup

- Connect UART to USB cable to EVM.
- Identify the UART COM port as enumerated on the host machine (Windows Device Manager Ports (COM & LPT)).
- If there are no USB serial ports listed in Device Manager under Ports (COM & LPT), then install the UART to USB driver from FTDI.
- Terminal Settings
  - Baud Rate: 115200
  - 8-N-1
- Open Terminal and wait for device to boot.

4.4 Flash the SD Card Image

- The default wic image with PipeWire support will be generated in ***deploy-ti/<machine>/images*** *with the name* ***tisdk-default-image-<machine>-evm.rootfs.wic.xz.***
- Use Balena Etcher to flash the wic image
  - Insert a micro-SD card into the USB SD card reader and start Etcher.
  - Select the wic.xz image
  - Select SD Card
  - Click *Flash*

4.5 Booting EVM with SD Card

- Make sure the boot mode pins on the board are for SD Card boot.
- Insert the SD Card into the SD Card slot.
- Connect the host PC to the USB Micro-B interface to access the system console and view UART logs.
- Power on the board.
- Log in as "root" with no password.

```
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 Use PipeWire

5.1 Check Service Status

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

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

5.2 Enable PipeWire and Wireplumber

PipeWire and WirePlumber is designed to run as a user-space service. In a single user Arago image (only ***root***), running PipeWire as a system service is acceptable. However, in a multi-user images (ex. ***weston*** in some Sitara platforms), it is strongly recommended to enable per-user sessions to isolate multimedia access, avoid conflicts with D-Bus session instances, ALSA device reservation, and prevent permission mismatches across user sessions.

Enable service to start automatically at boot

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

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

To simplify setup on the AUDIO-AM62D-EVM, a new service is included in the patches to automatically set the default audio devices in WirePlumber. To enable it, use following command:

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

5.3 Start PipeWire and WirePlumber

Start PipeWire and WirePlumber if not enabled

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

root@<machine>: systemctl start wireplumber

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

5.4 General PipeWire commands

5.4.1 List All Objects Currently in PipeWire Server

Use ***pw-cli list-objects*** command to list all objects.

```
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 List Only Nodes

Use ***pw-cli list-objects Node*** to list all nodes

```
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 Inspect Specific Object

Inspect specific object using command ***pw-cli info <object-id>*** .

```
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 Play and Record Stereo Audio

Play audio via ***pw-play*** or ***aplay*** :

```
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>
```

Record audio via ***pw-record*** or ***arecord*** :

```
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 Configuration

PipeWire setup consists of:

- PipeWire daemon
- Session manager (usually WirePlumber)
- Compatibility servers (PulseAudio, JACK)
- Client configuration

Each of these has its own configuration file which uses SPA JSON format, which is a relaxed JSON syntax.

Table 6-1. Configuration Files

Files	Purpose
pipewire.conf	Configures the PipeWire daemon
client.conf	Configures PipeWire clients
pipewire-pulse.conf	PulseAudio compatibility server
filter-chain.conf	Audio processing filters

Instead of modifying the main file, PipeWire recommends using drop-in files to override only specific settings. Use the example in the directory:

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

6.1 Sink and Source Configuration

There are two reference configurations files, sink and source added for AUDIO-AM62D-EVM for 8-channel support in 3.5 mm jacks.

For other Sitara EVMs, requiring only two channels, no additional configuration is necessary. However, the reference configuration can be adapted to modify sample rate, channel count, period size, and other parameters as needed.

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

Main configuration array defining objects created in PipeWire context. Each object in this array becomes a node in the PipeWire graph.

factory = adapter

Specifies that this object be created using the "adapter" factory. Adapters in PipeWire are used to bridge between different APIs (in this case, ALSA to PipeWire). Both configuration files use adapter factory to bridge ALSA hardware to PipeWire nodes.

factory.name

Determines direction: output vs input

node.name

Internal PipeWire node identifier. Creates alsa_audio_sink for playback and alsa_audio_source for capture.

node.description

Human-readable name in audio apps.

media.class

PipeWire media classification "Audio/Sink" for playback and "Audio/Source" for recording.

api.alsa.path

For direct hardware access. Only PipeWire can access the audio hardware and ALSA applications must go through PipeWire.

audio.channels

Configures both input and output for 8-channel audio in case of AUDIO-AM62D-EVM.

These configurations create two fundamental nodes in PipeWire's audio graph:

- Sink Node: Terminal endpoint for 8-channel audio playback
- Source Node: Starting point for 8-channel audio capture

For more information, please refer Alsa Configuration.

6.2 WirePlumber Configuration

Use the ***wpctl status*** command to list all the available audio sinks and sources.

```
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
```

Use ***wpctl inspect <id>*** to display information about the specified object.

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

Set defaults source manually by using the ID number of sinks and sources:

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

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

The WirePlumber changes introduces an automated audio device configuration system for AUDIO-AM62D-EVM through two key files:

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

This script implements an initialization sequence that waits up to 30 seconds for WirePlumber to become ready, then uses PipeWire's command-line tools (pw-cli and wpctl) to locate the alsa_audio_sink and alsa_audio_source nodes defined in the PipeWire configuration files and explicitly set them as the system's default audio devices. This automation is necessary because while PipeWire creates audio nodes based on the configuration files, it doesn't automatically designate them as the default devices that applications will use.

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

The accompanying systemd service ensures set-audio-defaults.sh script runs automatically after WirePlumber starts.

By default, the AUDIO-AM62D-EVM uses alsa_audio_sink and alsa_audio_source because of set-audio-defaults.service. wpctl command can be used to change audio routing to other devices (e.g., USB).

---

7 Performance Benchmarks

The test setup is as follows:

- Hardware
  - SK-AM62B-P1
  - TMDS62LEVM
  - AUDIO-AM62D-EVM
- Kernel - ti-linux-6.18.y - 12.00.00.07
- Yocto - 12.00.00.07.04

Concurrent audio applications will be launched to measure and compare the average performance of PulseAudio and PipeWire.

PulseAudio is packaged for all Sitara platforms. But service file is not packaged due to the patches mentioned in this document. To start PulseAudio as a background process, use the following command:

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

Use following command to stop PulseAudio

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

7.1 Latency

This test measures playback latency for PulseAudio and PipeWire.

For PulseAudio, latency was measured using the sink latency reported by ***pactl*** , which reflects the effective playback delay observed at the audio output.

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

For PipeWire latency was estimated by summing the buffer durations (quantum) of the active stream and sink nodes obtained from ***pw-top*** command, representing application and device buffering.

```
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
```

Use below formula to calculate latency from quantum and rate:

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

Table 7-1. Default Latency

Device	Audio Server	Latency (ms)
SK-AM62B-P1	PulseAudio	Approximately 1837 ms
	PipeWire	Approximately 143 ms
TMDS62LEVM	PulseAudio	Approximately 1837 ms
	PipeWire	Approximately 143 ms
AUDIO-AM62D-EVM	PulseAudio	Approximately 1837 ms
	PipeWire	Approximately 143 ms

These latency values can be further reduced by changing configuration for example quantum, clock rate, number of fragments, fragment size, and so on.

7.2 CPU and Memory Usage

This test measure CPU utilization and memory usage when multiple audio streams simultaneously use the audio server. Configure PulseAudio to work at latency closer to 143msec (PipeWire's latency) by changing PULSE_LATENCY_MSEC.

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

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

In PulseAudio, latency cannot be set to an exact value using PULSE_LATENCY_MSEC, as it is treated as a target rather than a strict configuration. As a result, the observed latency can differ significantly from the requested value. The actual latency is determined by internal buffer fragmentation, hardware constraints, and scheduler behavior.

There are other methods to control latency as well example by directly modifying variables that affect latency such as fragments, fragments size, and so on.

See Pulseaudio documentation for more details.

Table 7-2. CPU Load (Same Latency)

Device	Audio Server	CPU Usage (average)
SK-AM62B-P1	PulseAudio	Approximately 5%
	PipeWire	Approximately 1%
TMDS62LEVM	PulseAudio	Approximately 6%
	PipeWire	Approximately 1%
AUDIO-AM62D-EVM	PulseAudio	Approximately 5%
	PipeWire	Approximately 1%

Table 7-3. Memory usage (Same Latency)

Device	Audio Server	Memory Usage (average)
SK-AM62B-P1	PulseAudio	Approximately 22500 KB
	PipeWire	Approximately 46570 KB (Wireplumber-~22910KB)
TMDS62LEVM	PulseAudio	Approximately 25000 KB
	PipeWire	Approximately 33400 KB (Wireplumber-Approximately 30190KB)
AUDIO-AM62D-EVM	PulseAudio	Approximately 21600 KB
	PipeWire	Approximately 55000 KB (Wireplumber-Approximately 22990KB)

Note

PipeWire uses more memory due to its multi-process design, where the core daemon and WirePlumber session manager run independently. This separation provides fault isolation—if WirePlumber crashes, audio playback continues uninterrupted. PulseAudio's single-process design is more memory-efficient but less resilient to component failures.

7.3 CPU and Memory Usage with Resampling

This tests measures CPU overhead and memory usage introduced when the audio server performs sample rate conversion and configured at same latency. This test evaluates the efficiency of the resampling. Both PulseAudio and PipeWire are configured to resample audio to 44.1kHz (vs native 48kHz).

For PulseAudio, use the following commands to resample to 44.1kHz:

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

For PipeWire, use the following command to resample to 44.1kHz (for current session only)

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

For permanent configuration, create a new custom ***/etc/pipewire/pipewire.conf.d/99-custom.conf*** with following content

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

Table 7-4. CPU Usage with Resampling

Device	Audio Server	CPU Usage (average)
SK-AM62B-P1	PulseAudio	Approximately 24%
	PipeWire	Approximately 3%
TMDS62LEVM	PulseAudio	Approximately 29%
	PipeWire	Approximately 3%
AUDIO-AM62D-EVM	PulseAudio	Approximately 23%
	PipeWire	Approximately 3%

Table 7-5. Memory Usage with Resampling

Device	Audio Server	Memory Usage (average)
SK-AM62B-P1	PulseAudio	Approximately 21600 KB
	PipeWire	Approximately 52750 KB (Wireplumber-40000)
TMDS62LEVM	PulseAudio	Approximately 25500 KB
	PipeWire	Approximately 33310 KB (Wireplumber-33800)
AUDIO-AM62D-EVM	PulseAudio	Approximately 21500 KB
	PipeWire	Approximately 32250 KB (Wireplumber-23000)

7.4 Observations

The benchmark results are largely platform agnostic because the tested workload (5 concurrent audio streams) is relatively lightweight and does not significantly stress CPU, multicore scaling and DDR bandwidth . As a result, differences in CPU frequency, core count and DDR speed have minimal impact on CPU usage, latency, and memory occupancy. The small differences observed are likely within normal run-to-run variation due to factors such as scheduler timing, background activities, cache, and so on.

8 Summary

This document provided a comprehensive guide to using PipeWire on TI Sitara devices, from building a Yocto image to configuring and benchmarking the audio framework. Using the SK-AM62B-P1, TMDS62LEVM, and AUDIO-AM62D-EVM as reference platforms, the benchmarks demonstrate PipeWire's capability as a low-latency and high-performance audio design for embedded applications.

9 References

1. PipeWire, *PipeWire documentation*, webpage.
2. Pulseaudio, *Pulseaudio documentation*, webpage.
3. Texas Instruments, *SK-AM62B-P1*, product page.
4. Texas Instruments, *SK-AM62B-P1 Processor SDK Documentation*, software development guide.
5. Texas Instruments, *SK-AM62B-P1 Quick Start Guide*, quick start guide.
6. Texas Instruments, *TMDS62LEVM*, product page.
7. Texas Instruments, *TMDS62LEVM Processor SDK Documentation*, software development guide.
8. Texas Instruments, *TMDS62LEVM Quick Start Guide*, quick start guide.
9. Texas Instruments, *AUDIO-AM62D-EVM*, product page.
10. Texas Instruments, *AUDIO-AM62D-EVM Processor SDK Documentation*, software development guide.
11. Texas Instruments, *AUDIO-AM62D-EVM Quick Start Guide*, quick start guide.

10 IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 06/2026

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025