Application Report

SPRACG8–September 2018

Audio Post Processing Engine on Jacinto™ DRA7x Family of Devices

Danny Jochelson and Stephen Molfetta

ABSTRACT

The automotive experience demands the tools for manufacturers to create high-quality audio for the vehicle occupants, while also enabling those same occupants to customize audio settings for their desired tastes. A myriad of audio input sources, including CD/DVD, radio, aux input, streaming music, *Bluetooth*® audio, navigation, alerts, and other notifications, routed to multiple output playback zones necessitates an audio subsystem tailored for the automotive audio market. TI's Audio Post Processing Engine (APPE) on Jacinto devices provides a common audio framework for automotive OEMs to enable this user customization, while also allowing Tier 1 providers and OEMs to fine tune their audio for the best possible out-of-the-box user experience. By implementing audio processing algorithms and routing within TI's C66x Digital Signal Processor (DSP) on Jacinto 6 DRA7x single chip solutions, automakers can reduce hardware system cost and integration complexity. This same audio solution can also be leveraged across multiple operating systems, such as QNX®, Linux®, and Android™. APPE provides real-time controls from the High-Level Operating System (HLOS), allowing customers with limited DSP experience to leverage a large suite of audio algorithms, including dynamic range compression, equalizers, mixers, and volume controls. Automotive manufacturers can also easily add additional algorithms to further differentiate their end platform.

Contents

List of Figures

List of Tables

Trademarks

1 Introduction

1.1 Automotive Consumer Needs

Decades ago, creating a car audio system was fairly straightforward. One audio source was played, and everyone in the car listened to the same song, whether that source was radio, an 8-track tape, or a compact disk. Audio speakers would have a single equalization preset, and the user would have a single volume knob that would apply a basic gain to the audio. Today, automotive audio has a complicated list of requirements, driven by customers' expectations of their current media consumption in the home. More specifically:

- Individual consumption of media – Multiple listening zones are more common, with one or two rear-seat headphones added to the typical cabin audio configuration. Each of these listening zones needs to be able to select from a multitude of media inputs. Some zones may want to listen to the same media, or all the zones may want to listen to separate media.
- User sound customization – Automotive manufacturers tend to provide multiple presets for equalization settings (for example, “Rock” mode, “Pop” mode) for cabin audio. In addition, rear-seat entertainment systems need audio post processing like volume controls, bass/treble controls, balance/fade knobs, and equalization settings.
- Multi-channel (> 2 channel) speaker outputs – Most car cabin speaker configurations now have at least 4 speakers, with 6-channel (5.1 audio) and 8-channel (7.1 audio) occurring as well. With these multi-channel outputs, basic upmixing of standard stereo streams needs to be performed.
- Multi-channel media playback – While standard stereo audio inputs like Aux Line Input, CD's, and Radio still need to be supported, multi-channel audio from DVD, Blu-ray™, and HDMI inputs also need to play back. These multi-channel inputs need to have a basic downmix to match the appropriate cabin configuration.
- Vehicle-dependent audio settings – Some audio controls depend upon the current status of the automobile. A common example is a speed-dependent volume control where the actual loudness of the audio output is raised as the speed of the vehicle increases (due to the increased noise floor at higher speeds).
- Multi-stream mixing – While multimedia is playing, some additional audio streams/notifications may need to be mixed with the multimedia audio. This audio usually needs to be mixed into individual channels (that is, not necessarily all of the outputs) within the vehicle.

Figure 1 shows a typical audio playback architecture for an automotive infotainment system today.

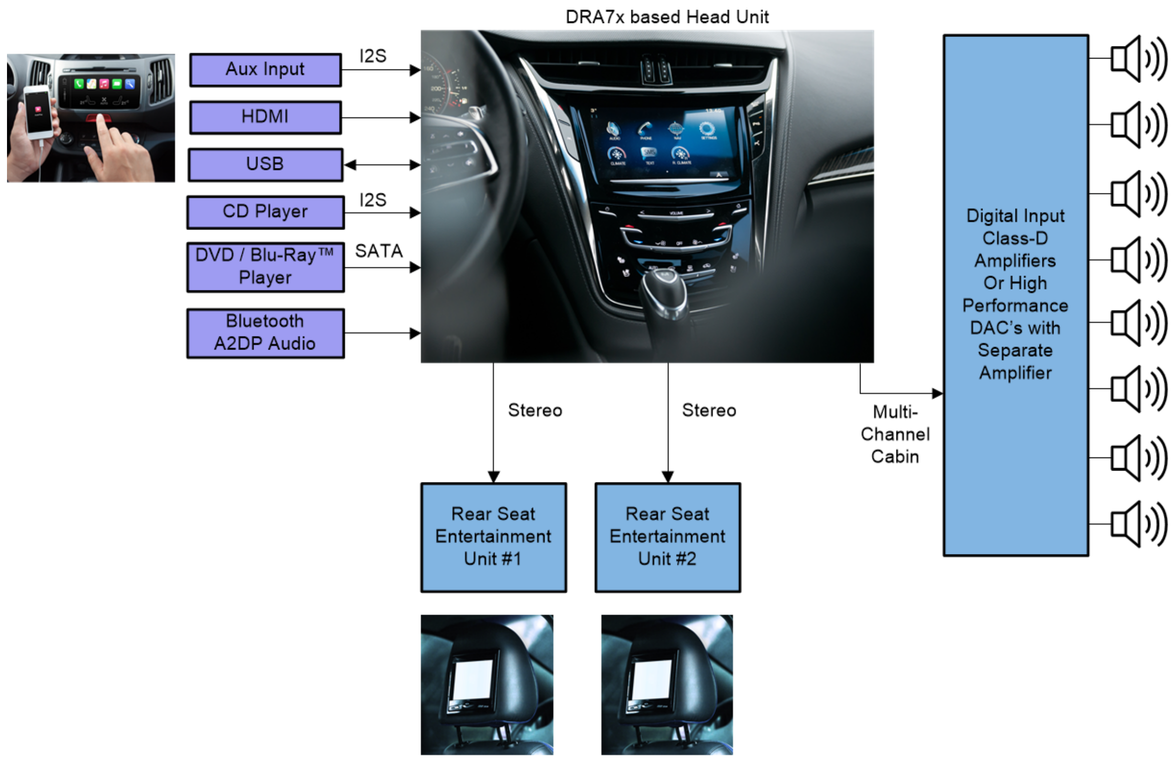

DRA7x based Head Unit
Aux Input
HDMI
USB
CD Player
DVD / Blu-Ray™ Player
Bluetooth A2DP Audio
I2S
I2S
SATA
Stereo
Stereo
Multi-Channel Cabin
Rear Seat Entertainment Unit #1
Rear Seat Entertainment Unit #2
Digital Input Class-D Amplifiers Or High Performance DAC's with Separate Amplifier

Figure 1. Typical Infotainment Audio Inputs and Outputs

Each source may vary in the number of audio channels as well as the audio sample rate. For example, Audio CD content is recorded at 44.1 kHz, but most movie audio from DVD's plays at 48 kHz. As audio systems typically operate at a fixed sample rate to the speaker outputs, sample rate conversion is required to match all input source rates to the output rate. Car systems also need to playback audio files to handle events like navigation instructions, button presses, alerts, and other notifications. Table 1 summarizes possible inputs to a car audio system. Note that not all of these audio streams would likely occur simultaneously to the same audio output zone.

Table 1. Typical Audio Inputs to Automotive Audio Subsystem

Input Sources	Typical Sampling Rate	Typical Number of Input Channels
Music (CD Player)	44.1 kHz	2
Movies (DVD/Blu-ray™ Player)	48 kHz	6 or 8
HDMI	32 kHz, 44.1 kHz, 48 kHz	2 (required), 6 (optional) (1)
Bluetooth A2DP	44.1 kHz or 48 kHz (2)	Up to 2
USB	44.1 kHz, 48 kHz, or 96 kHz	2 or 6
Radio	44.1 kHz	2
Aux Line Input	44.1 kHz or 48 kHz	2
Navigation	8 kHz - 48 kHz	Up to 2
Button Presses	44.1 kHz or 48 kHz	Up to 2
Notifications	44.1 kHz or 48 kHz	Up to 2
Chimes	44.1 kHz or 48 kHz	Up to 2

(1) Based on HDMI v1.4a.

(2) According to Advanced Audio Distribution Profile Specification, v1.0.

1.2 Automotive Manufacturer Needs

While the end consumer needs control over basic audio routing and customization settings, the car manufacturers (and their partners) need tuning controls to maximize the audio performance of their cabin speakers. Speaker equalization is utilized to tune the output frequency response through the speakers and compensate for the cabin acoustics. Crossover filtering for bass and mid/high speakers limits the frequency range of the signal to what can be efficiently reproduced by the given speaker or driver type. Since all speakers are not equidistant from the listener in a car, some speaker sounds need to be delayed to allow the driver to hear the appropriate sound localization. Lastly, the dynamic range of the audio signal needs to be limited before being sent to the amplifier and speaker to ensure that these do not saturate and cause distortion.

2 Audio Post Processing Architecture

With heterogeneous multicore system-on-chips (SoCs) like Jacinto 6, a typical infotainment system can run an HLOS such as QNX, Linux, or Android on the main processing unit and designate other CPUs, like Digital Signal Processors (DSP), for specific applications like radio and audio post processing. TI's Audio Post Processing Engine (APPE) is a DSP-based software framework designed to fulfill the audio post-processing needs of both the car customer and car manufacturer. It is provided in full source code in the Processor SDK – Automotive Radio & Audio SDK and runs on the C66x DSP on Jacinto 6 using TI's Real-Time Operating System (RTOS), SYS/BIOS [(1)]. This DSP-based audio architecture provides several advantages:

- Leverages the efficiency of TI's C66x DSP for complex audio and radio use-cases.
- Offloads processing on the HLOS to handle more applications simultaneously.
- Enables lower latency for radio, auxiliary inputs, or other audio sources compared to running through the HLOS.
- Provides a single audio post processing solution that can then be leveraged across multiple projects that can have different required operating systems.
- Provides a variety of DSP-optimized audio post processing algorithm libraries, which allow automotive manufacturers to tune the audio characteristics in the vehicle and expose controls to common functions for the consumer.
- Allows developers and manufacturers to differentiate their audio solutions (through their own development or through 3rd parties by adding custom algorithms to the DSP).
- Allows usage of a single audio solution for early audio (prior to start of kernel boot) with minimal dependencies on a particular HLOS.

(1) For more information on TI's SYS/BIOS package, see the http://processors.wiki.ti.com/index.php/Category:SYSBIOS wiki.

APPE is a flexible multi-input, multi-output audio processing and routing framework that is tightly coupled with the host HLOS of choice. Audio inputs, or “sources”, can come from an assortment of applications running on the various CPUs, including from the HLOS sound card interfaces (for example, two main audio drivers, up to three prompts, and one chime), Software Defined Radio running concurrently on the DSP, or directly through the input audio serial port into the DSP. Similarly, audio outputs, or “zones”, can play out directly through the output serial port on the DSP or be sent to another application running on the DSP or HLOS for further processing or system routing. Multiple output zones enables separate controls and routing for cabin audio along with multiple rear-seat headphones. TI’s Audio Manager, an HLOS library that is also provided in “Processor SDK – Automotive Radio & Audio”, exposes an API to allow a host application running on the HLOS to easily and dynamically configure and control the APPE.

Figure 2 displays a typical system integrating APPE on Jacinto 6.

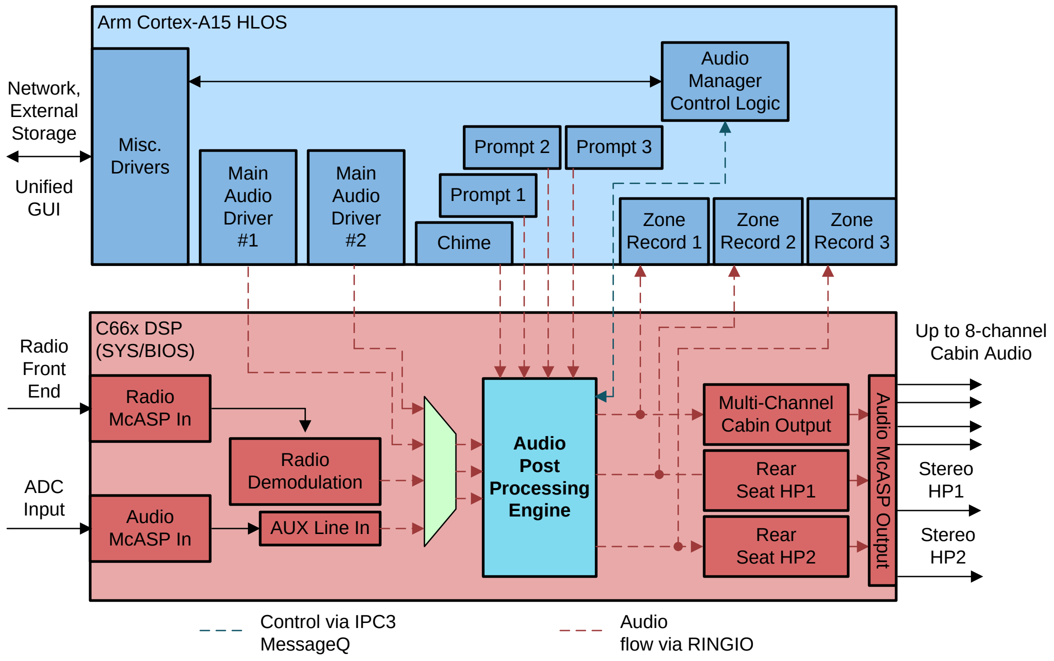


Figure 2. Audio Post Processing Engine (APPE) – High-Level System Architecture

All control for the Audio Post Processing Engine is done from the HLOS and is achieved via the MessageQ interface, which is a part of TI’s Interprocessor Communication (IPC3) package [(2)]. All audio inputs and outputs for APPE are RingIO components. Also built upon the IPC3 package, the RingIO component utilizes a single interface for seamless audio streaming either between processing cores or within a single core. This software component sends both audio data and attributes via a single entity, while also enabling automatic callback notifications to ensure audio buffers avoid underruns or overruns. If the routings of the APPE need to be expanded (for example, additional inputs), more RingIO instances can be created as the developer customizes their framework for their end system.

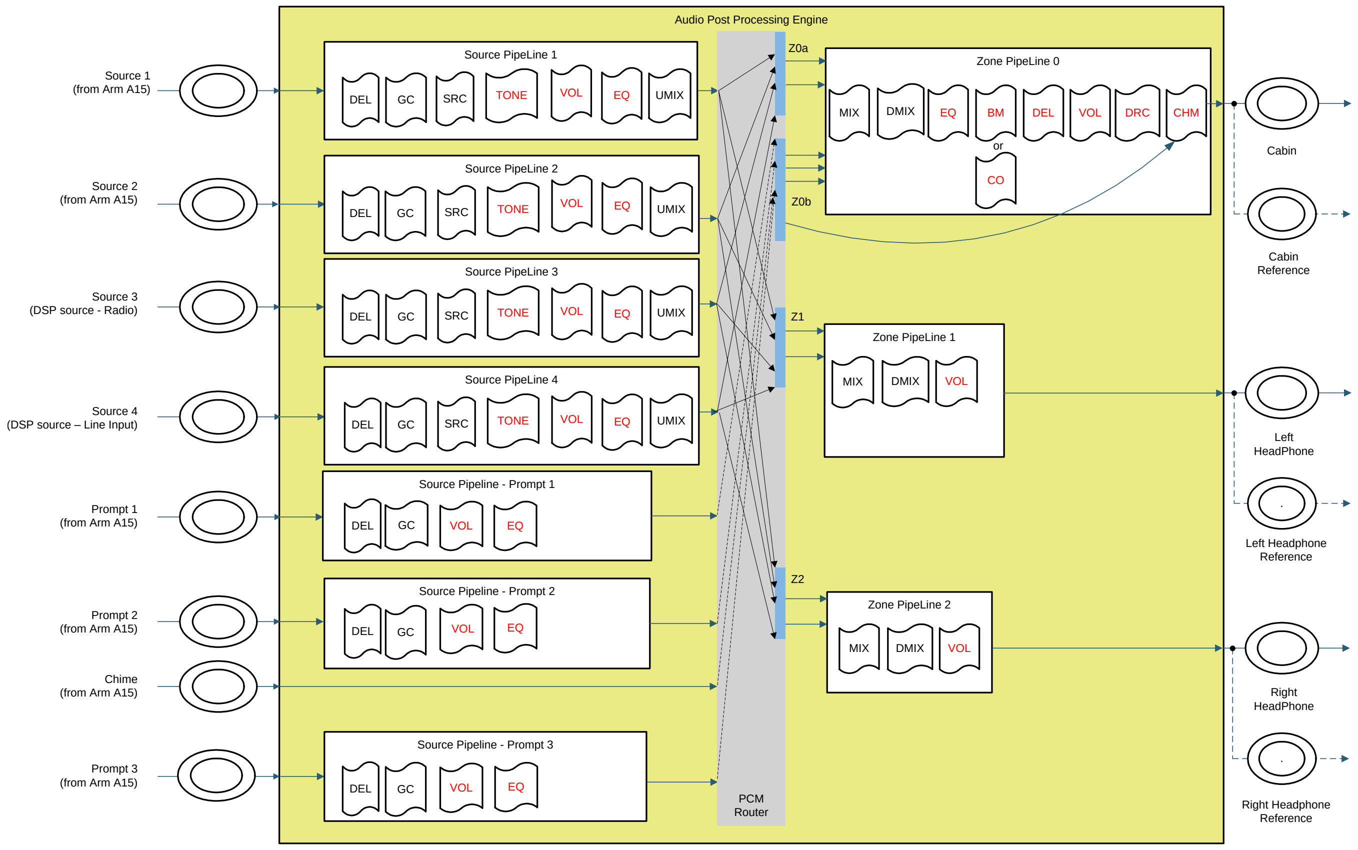
(2) For more information on the IPC3 package, see the IPC 3.x wiki and the *IPC Users Guide* wiki.

The inputs to APPE can be separated into three categories: main audio sources, prompts, and chimes. Table 2 describes the attributes of these input types.

Table 2. Audio Inputs to APPE

Input Type	Description	Number of Channels	Input Sampling Rate
Main Audio	Main entertainment audio intended to be consumed by the user.	2, 4, 6 (5.1 Audio), or 8 (7.1 Audio)	8 kHz, 11.025 kHz, 16 kHz, 22.05 kHz, 32 kHz, 44.1 kHz, 48 kHz, 96 kHz
Prompt	Notifications and other auxiliary audio sources that are intended to be mixed simultaneously with the main audio in the cabin zone.	1 or 2	Same as output sampling rate (44.1 kHz or 48 kHz)
Chime	Alerts that need to be mixed with the other streams after all processing is done for the cabin zone.	1	Same as output sampling rate (44.1 kHz or 48 kHz)

APPE provides the flexibility to configure a variety of audio routing scenarios to meet the demands of the automotive manufacturer. Through the PCM routing crossbar, any main audio source could be routed to any or all zones at a given time. Each zone is able to play back a single main source at a time. [3] Additionally, the main cabin zone can mix prompt and chime sources with this main audio source. Figure 3 shows more details for the default internal architecture of this framework.



(1) Red text - User run-time controls available.

(2) Black text - Control based on input and output audio attributes and routings.

Figure 3. APPE Default Internal Architecture

[3] The only time a zone accepts more than one main audio source is during a crossfade, when two main sources briefly are played simultaneously through the zone while the crossfade is in progress.

The APPE consists of a processing thread that manages a collection of audio processing "pipelines", with each pipeline containing a sequence of audio processing algorithms. Many of the algorithms can be controlled in real-time from the Audio Manager application, having the ability to query the status, enable/disable processing, and set individual parameters. Some algorithms do not expose a control interface but are automatically configured by the framework based on the desired output configuration and input audio stream parameters. (4)

Audio pipelines are divided into three distinct types:

- Zone pipeline – Outputs of zone pipelines are tied to a particular output listening zone. A common case would include a cabin zone (typically 4, 6, or 8 channels), plus 2 rear seat entertainment zones (typically stereo outputs). Zone pipelines contain processing that is specific to the output listening zone. Some common audio algorithms contained in Zone pipelines include listening volume controls (including balance/fade), tone controls, and speaker limiters. (5) The number of channels that the algorithms process is determined by the build-time output configuration for that zone.
- Source pipeline – Source pipelines handle all main audio inputs to APPE. The default configuration contains four source pipelines: two inputs from Arm® audio cards, Software Defined Radio, and auxiliary line input. Additional sources can be defined by the developer as necessary. Each of these input sources would be the primary audio to which the user is listening and could potentially require distinct audio processing. For example, separate volume controls and equalization settings could be contained in separate source pipelines. The outputs of source pipelines can be individually routed to any zone pipeline, and this routing can be changed in real-time via the Audio Manager application, with optional automatic crossfading performed by the mixer at the start of each zone pipeline (6). A single source pipeline could be routed to all three zones (7), or a separate source could be routed to each zone pipeline. The zone pipeline can receive only one source pipeline output at a time, with the exception being during a crossfade when transitioning between two input sources.
- Prompt pipeline – Prompt inputs, such as navigation, button presses, and other notifications, are sent through these pipelines. They are similar to the source pipelines in that individual processing like separate volume and equalizer settings could be applied to the prompt sources. The output of each prompt pipeline is then routed to the cabin zone mixer for simultaneous mixing with the main audio being routed to the cabin (see "Z0b" router in the diagram, which shows these three prompts routed to cabin zone). The cabin mixer can then set the mixing levels for these monophonic or stereo prompts into each of the output channels at the beginning of the zone pipeline. Thus, through control of the cabin zone mixer, prompts can be routed to individual channels. For example, navigation audio could be easily routed to the front two channels of the cabin, while another notification could be sent through another prompt to the rear cabin channels.

On the output of each zone, in addition to being sent to the audio interface and then digital-to-analog converters (DACs), a copy of the output of each zone is sent back to the Arm Cortex®-A15. For more information, see "Cabin Reference", "Left Headphone Reference", and "Right Headphone Reference" in Figure 3. This allows recording of the zone output, which is useful when testing or tuning individual algorithms and pipelines. In addition, this provides a mechanism for any voice processing (echo cancellation) done on the Arm Cortex-A15 to receive the audio sent to the speaker as a downlink echo reference.

The APPE internal architecture diagram shown is the default software configuration. In addition to runtime controls for audio algorithms and routing, the framework can be configured at build-time for the following:

- Number and types of pipelines
- Number and order of the algorithms in each pipeline
- Total number of output channels per zone output
- Total number of output zones
- APPE processing buffer sizes (for latency tuning)
- Output bit depth (8)
- Output sampling rate for APPE (for example, 44.1 kHz or 48 kHz) (startup time)

(4) For more information on how algorithms are typically used and their functionality, see Table 3.
(5) For more details on all available audio algorithms for Audio Post Processing Engine, see Section 3.
(6) See "Z0a", "Z1", and "Z2" inputs to Zone pipeline in the figure, which routes two inputs to the zone for Mixer crossfading.
(7) Note that routing of a single source pipeline to all zones is supported on Linux/QNX, but not on Android.
(8) By default, APPE performs 24-bit processing (in 32-bit container), and converts to 16 bits at its output. The build flag can be used to change APPE to send out the 32-bit container to the DAC.

3 Audio Post Processing Algorithms

Audio processing algorithms are included in the Jacinto 6 Processor SDK – Automotive Radio & Audio package for usage within the APPE framework. Table 3 summarizes the capabilities of each of these algorithms.

Table 3. Algorithms Available for APPE Framework

Algorithm	Label	Usage	Details	Runtime Control or Automatic Framework Control
Volume with Loudness	VOL	User volume control	Includes master volume, per-channel volume (fade/balance), and loudness filtering	Runtime Control
User Equalizer	EQ	User shapes frequencies for their preferences (for example, “rock”, “pop” settings)	10-band equalization – second-order peaking filters applied on all channels	Runtime Control
Speaker Equalizer	EQ_MULTI	Manufacturer tunes individual speaker channels	10-band equalization for each channel – second-order peaking filters applied separately on each channel	Runtime Control
Tone Control	TONE	User Bass/Mid/Treble settings	Bass and treble shelf filters, mid peaking filter	Runtime Control
Mixer	MIX	Automatically crossfade main audio sources, simultaneously mix in prompts into individual channels	Set coefficients for mixing prompts into each of its output channels (cabin zone only)	Runtime Control (when not playing)
Crossover filtering	CO	Manufacturer tunes crossover for woofer vs. satellites	Separate bass frequencies into woofer channel and higher frequencies into the satellite channels. Configurable filter types.	Runtime Control
Bass Management	BM	Manufacturer tunes crossover for woofer vs. satellites with finer tuning for bass/treble balance	Includes similar crossover filter as crossover component, but adds controls for bass/treble balance before and after crossover.	Runtime Control
Delay	DEL	Manufacturer tunes spatial audio image in vehicle	Per-channel speaker delay to control audio “location” and align audio amongst the speakers	Runtime Control
Chime component	CHIME	Automatically mix monophonic chime audio into individual channels	Set coefficients for mixing chime into each of its output channels (cabin zone only)	Runtime Control (when not playing)
Dynamic Range Compression	DRC	Manufacturer tunes for better listening in noisy car environment and limit output to not saturate the speakers or amplifier (1)	5-section DRC (gate, expansion, constant, compression, and limiter sections), configurable thresholds and time constants	Runtime Control
Downmix Module	DMIX	Automatically dowmix higher-channel inputs to lower-channel output zones	Handles 8-ch, 6-ch, 4-ch inputs and can convert to 6-ch, 4-ch, or 2-ch output	Automatic
Upmix Module	UMIX	Automatically upmix lower-channel inputs to higher-channel output zones	Handles 2-ch input and can convert to 4-ch or 6-ch (5.1 Audio), or 8-ch (7.1 Audio) outputs	Automatic
Gain Control	GC	Automatically applies basic gain on source inputs internally within framework	Applies simple gain control. Used to help ramp up beginning of playback and ramp down at the end of playback or in case of input discontinuity.	Automatic
Synchronous Sample Rate Converter	SRC	Automatically sample rate converts to APPE’s output sampling rate	Convert 8 kHz, 11.025 kHz, 16 kHz, 22.05 kHz, 32 kHz, 44.1 kHz, 48 kHz, or 96 kHz to output APPE sampling rate (for example, 44.1 kHz or 48 kHz)	Automatic

(1) Either DRC or VOL module could be utilized to implement a speed-based volume control, as the HLOS can modify the runtime controls based upon the vehicle's speed.

4 Algorithm Expandibility

As a software framework, other algorithms can be easily integrated into the framework and initiated within pipelines. For individual algorithms, TI's eXpressDSP™ Algorithm Interoperability Standard (xDAIS) [9] is utilized for the algorithms provided with the package (listed in Section 3). The xDAIS standard provides an easy way to create/delete algorithm instances, expose basic apply calls, and control algorithm status information. Each algorithm has a small Audio Processing Algorithm (APA) layer that allows algorithm processing configurations to be adapted in real time. This APA layer provides a standard interface for APPE to control individual algorithm parameters (for example, equalizer band gains and per-channel volume controls) and retrieve algorithm status. The APA may also update the algorithm's processing based on dynamic audio stream parameters (for example, the number of channels that the algorithm processes varies based on the current input/output audio stream parameters for that APPE pipeline). Each pipeline then has a single Audio Processing Pipeline (APP) interface, which sequentially invokes the APA calls for each algorithm within the pipeline.

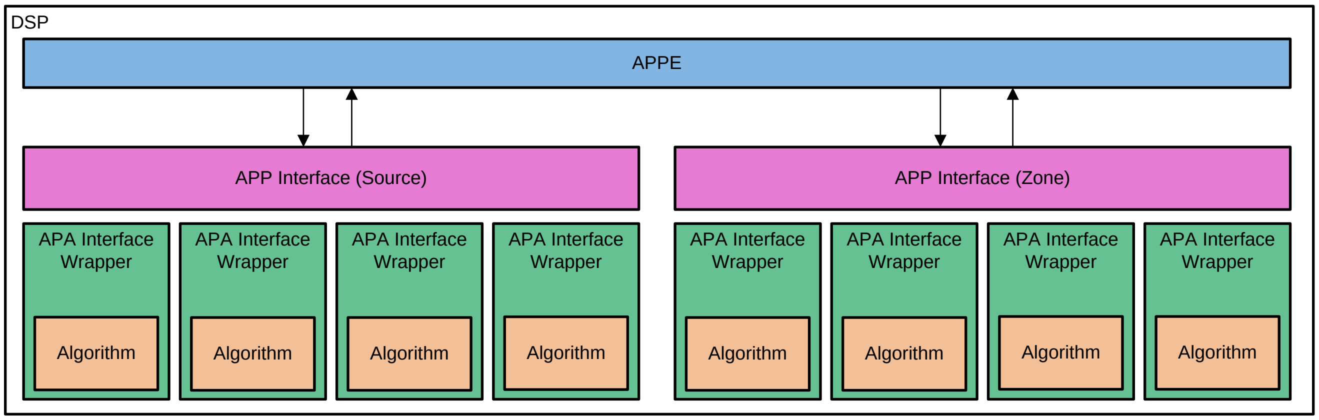


Figure 4. Audio Algorithm Interfaces Within APPE

The Jacinto 6 Processor SDK – Automotive Radio & Audio package provides sample code for the xDAIS and APA interfaces, thereby allowing easy integration of custom and/or 3rd party audio algorithms.

5 Typical Framework Routing Examples

This framework contains an extensive set of manual controls that can be initiated from the Audio Manager application. As mentioned in Table 3, the framework also automatically configures some algorithms, based upon the input audio source and the output zone configuration. The examples shown in Figure 5 highlight the framework's automatic runtime configuration.

Assuming the output Cabin zone is set at build-time to 6 output channels (5.1 audio) at 44.1 kHz, the main audio source pipelines will be configured based upon the input audio. Assuming audio is played back from the HLOS to the source pipeline #1, the audio driver on that processor will extract the audio information (for example, number of channels, sampling rate) from the WAV header on the audio file and send this information via RingIO. When the APPE receives this information, it automatically configures the pipelines and algorithms.

[9] For more information on xDAIS, see *A Technical Overview of eXpressDSP-Compliant Algorithms for DSP Software Producers* and *eXpressDSP Algorithm Standard – xDAIS Developer's Kit and xDM*.

Figure 5 displays a few examples of this configurability.

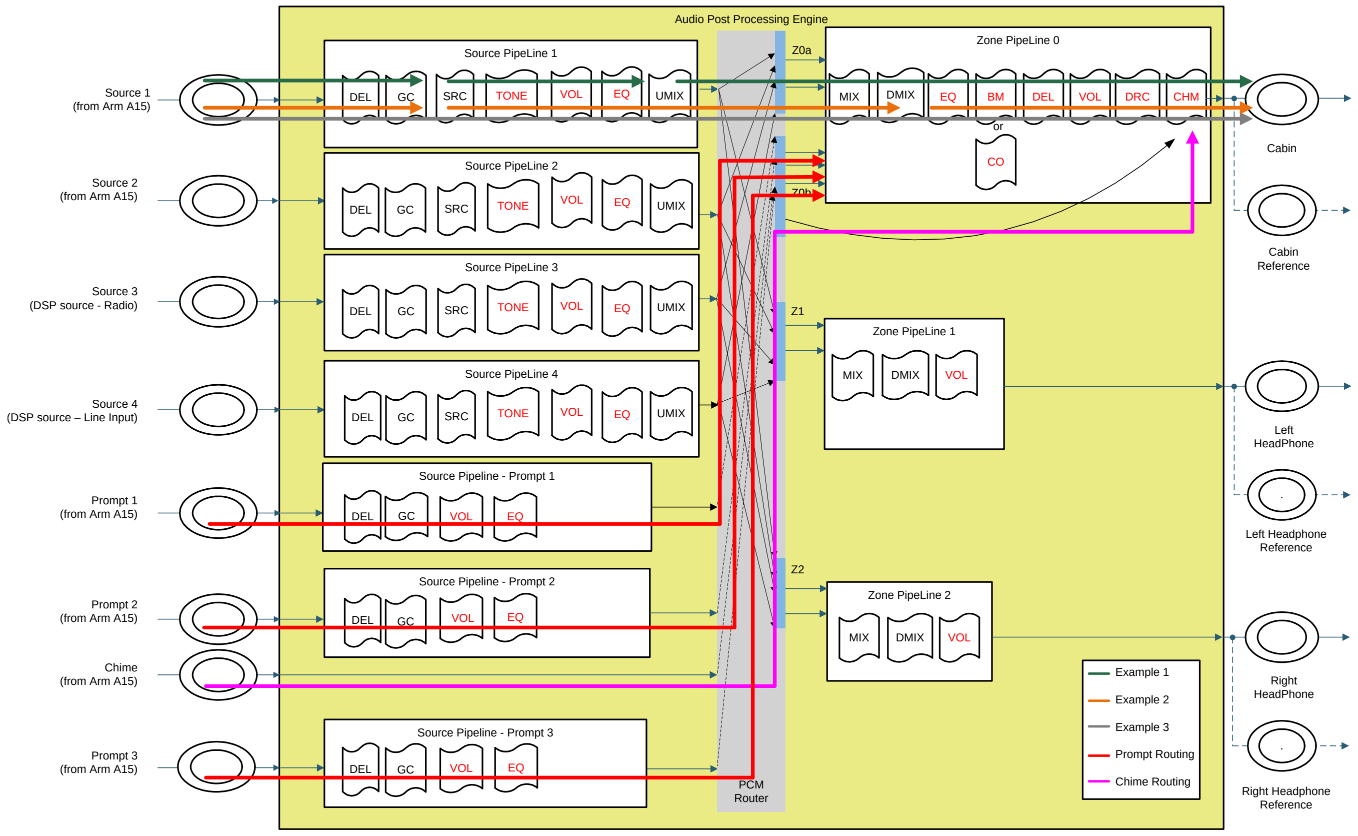


Figure 5. Framework Example - Different Input Formats, Prompt/Chime Routing

In Example #1, a 96 kHz, stereo file is played to the Source Pipeline #1 (shown via the green arrows in Figure 5). At the start of playback, the sample rate converter algorithm will be configured for conversion from 96 kHz to 44.1 kHz. After the sample rate conversion, all algorithms in the source pipeline will process the 44.1 kHz, stereo audio data. At the end of the source pipeline, the upmix algorithm will convert the stereo stream to 6 channels. This allows the cabin zone pipeline to perform all of its processing at 44.1 kHz on 6 channels of audio.

As a second example, a 48 kHz, 8-channel file is played to the Source Pipeline #1 (shown via the orange arrows in Figure 5). At the start of playback, the sample rate converter algorithm will be configured to convert the 8-channel audio from 48 kHz to 44.1 kHz. Thus, after the sample rate converter algorithm in the source pipeline, the algorithms process the 44.1 kHz, 8-channel data. This audio data is sent to the cabin zone pipeline. Toward the beginning of the zone pipeline, the downmix algorithm converts the 8-channel audio data to 6-channel audio. This allows the remainder of the cabin zone pipeline to perform all of its processing at 44.1 kHz on 6 channels of audio.

As a final example, if a 44.1 kHz, 6-channel audio file is played instead (shown via the grey arrow in Figure 5), then all algorithm processing in the source and zone pipeline are configured for 44.1 kHz, 6-channel audio data, while the sample rate converter, upmix, downmix algorithms are bypassed (since they are unnecessary).

When prompts are played back from the HLOS, the audio is processed through their dedicated prompt pipelines and sent to the cabin zone (Zone 0), and the mixer component mixes these into the individual channels at the start of Zone 0 (see the red lines in Figure 5).

Lastly, the chime input is routed directly to the chime component for mixing (shown via the pink line in Figure 5). The chime source is not routed to the start of the cabin zone because this ensures that cabin volume setting for multimedia playback does not affect the chime levels. This helps ensure that chimes are always reproduced at the same volume, and also minimizes the latency for these important audio signals.

6 Integration With High-Level Operating System

TI's Audio Manager Library, which provides the controls for APPE, is provided in the Jacinto 6 Processor SDK – Automotive Radio & Audio package as part of an example application that can be run on QNX, Linux, and Android. This application integrates the Audio Manager with two other related manager libraries: a DSP Manager that is used for master DSP control and initialization, and a Radio Manager that provides a similar API for TI's Software Defined Radio stack. The application further integrates various HMI layers to provide users convenient control through a command line interface or for an external GUI communicating over a raw UART or TCP/IP socket. A supervisor provides a light glue between the HMI and the various manager libraries, routing commands from the HMI to the appropriate manager, or for relaying callback information from the manager back to the host HMI.

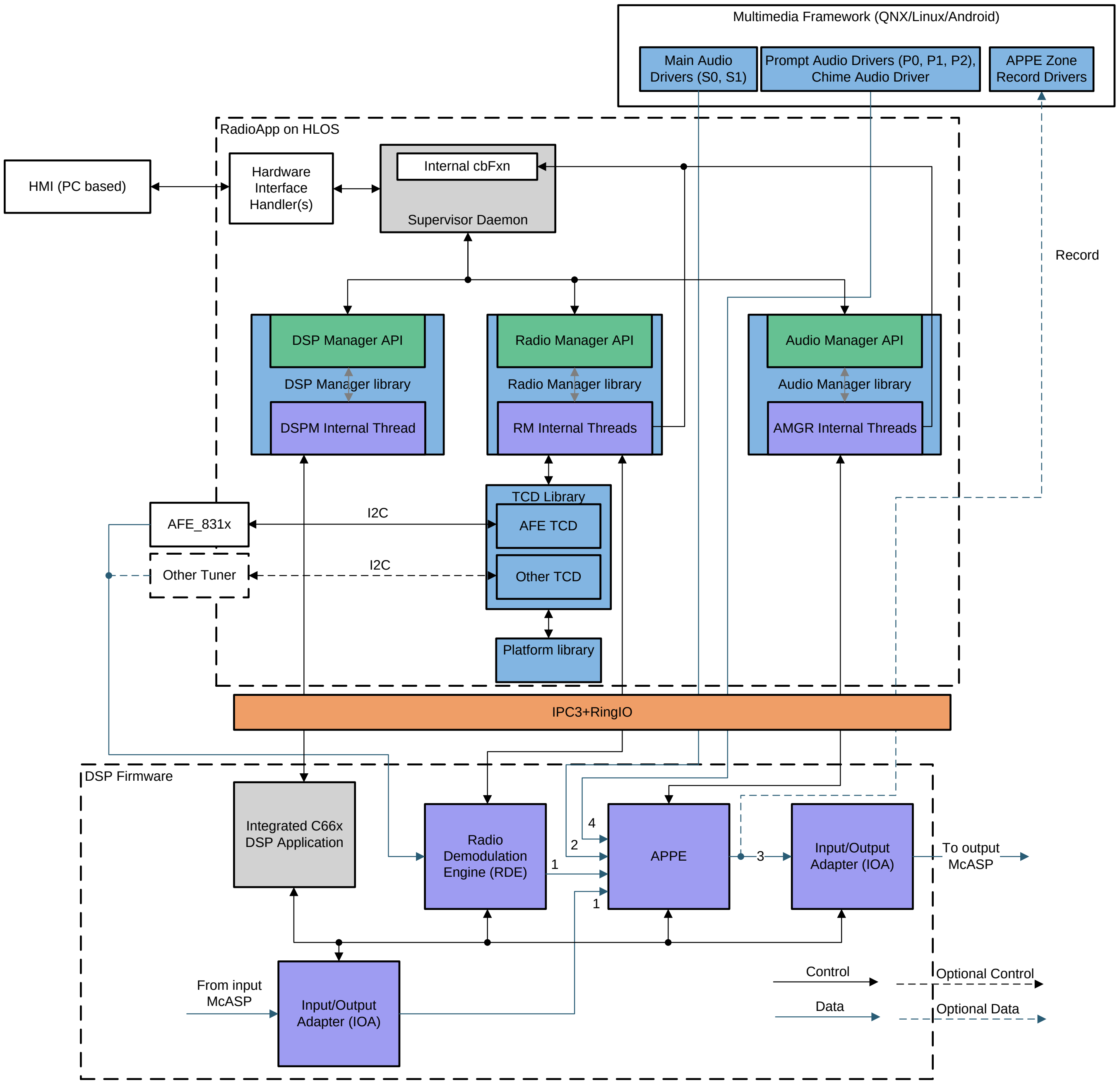


Figure 6. Operating System Integration

Audio playback from the HLOS to APPE is achieved via drivers, included in the SDK, that create sound cards on the HLOS and present a standard audio library interface to the developer. The interface for this depends on the HLOS of choice:

- For Android, an APPE HAL is provided that allows streaming for main audio (S0, S1 cards), prompts (P0, P1, and P2 cards), and a chime (C0 card). It also provides recording cards to record a copy of the output of each APPE zone.
- For Linux, an ALSA I/O user space plugin is provided with similar playback and record cards as the APPE HAL.
- For QNX, an io-audio dll is provided to enable multimedia audio playback (S0, S1 cards). [10]

The final rendering of APPE output has flexibility to either be sent directly to the audio serial port from the DSP or back to the HLOS for playback out through the host OS's audio stack. A copy of each zone's output audio can also be sent back to the HLOS for either recording or sending an echo reference to an echo cancellation algorithm. APPE HAL and ALSA I/O plugin record cards can receive this audio output on Android and Linux, respectively. These playback and capture options are default behavior but serve to highlight that the audio inputs and outputs could be routed from or to any application with a RingIO interface and is ultimately flexible enough to meet system requirements for a wide range of audio architectures.

7 Summary

In order to serve the continuously growing needs in automotive audio, TI's Audio Post Processing Engine provides a robust processing framework to handle a multitude of audio inputs routed to multiple audio output zones. Allowing individual media consumption with customized sound settings, the framework enables multi-channel audio inputs and outputs that are extensively needed in today's automotive market. Offering a variety of audio algorithms, this framework provides tools to enable both user controls and manufacturer tuning knobs. With an eye toward expandibility, customers can also differentiate their audio solution with new algorithms. This framework offers audio routing targeted at automotive use cases, while being highly integrated with typical High-Level Operating System audio frameworks in QNX, Linux, and Android.

8 References

- Welcome to SYS/BIOS
- IPC 3.x wiki
- *IPC Users Guide* wiki
- *eXpressDSP Algorithm Standard – xDAIS Developer's Kit and xDM*
- *DRA75x/74x SoC for Automotive Infotainment Silicon Revision 2.0, 1.1 Technical Reference Manual*
- *DRA75x, DRA74x Infotainment Applications Processor Silicon Revision 2.0 Data Manual*
- *A Technical Overview of eXpressDSP-Compliant Algorithms for DSP Software Producers*

[10] For support on QNX, all playback of prompts and chimes is achieved by playing back PCM data from the Audio Manager instead. Recording of Output Zones is also handled by the Audio Manager instead of using io-audio drivers.