Integrating Audio Effects in Android™

By Sonal Gupta, Senior Engineer, Multimedia Systems & Audio Codecs, Ittiam Systems Pvt. Ltd

Sitting at the core of the newly emerging embedded devices is the Android™ Operating System. The open-source, complete mobile software stack by Google™, is spreading way beyond expectations. It offers developers the ability to build extremely rich and innovative applications, thereby making it a compelling platform for developers as well as users.

Although launched as a mobile phone OS, Android is rapidly overcoming the multimedia market in other domains as well. With applications like Internet Tablets, STBs, Digital Photo Frames, and High-End Media Players, it has made a prominent mark in the home multimedia space.

The most popular application use-case across these various consumer devices is that of a music player. It is essential for these devices to not only support multiple popular music formats, but also to style the music being listened to, giving the user a "feel good" experience. A wide range of audio effects are available today, from a simple equalizer to complex 3D audio algorithms that enable the users to personalize their music, reduce the listening fatigue and create virtual acoustic environments.

ARM dominates the market space for the mobile and consumer electronic devices with a wide range of power efficient, high performance cores such as ARM9E (ARMv5T), ARM11 (ARMv6) and Cortex™-A8 and -A9 with Neon™ (ARMv7) multimedia acceleration.

This article discusses some of the must-have audio effects for a device and how to integrate them into an ARM-based Android device.

Audio Effects on Mobile

With mobile phones being widely used as media players, the expectations of the user with respect to audio quality have increased. Unfortunately, owing to the small size and the nature of their use, mobile phones are not ideally suited to render a great listening experience. However, a well designed set of audio effects and post processing can be used to overcome these limitations and provide hi-fi audio quality.

The space constraint on mobiles causes a significant portion of the audio signal from the left speaker to reach the right ear of the listener and vice versa. This is termed as "acoustic crosstalk" and results in a mono-like sound image. **Stereo Widening** can be used to create a wide soundstage from closely spaced stereo speakers. When earphones are used, the sound image is created outside the head of the listener by means of HRTF (Head Related Transfer Function), thus reducing listening fatigue. At the same time effects such as **Virtual Surround** enables the listener to enjoy multichannel audio on the stereo speakers or earphones, with a surround effect. A must-have suite of audio-effects will typically include the below:

Audio Effect	Need
Loudness Compensation	To improve the listening experience at low volume
Stereo Widening	To reduce listening fatigue
Virtual Surround	To enjoy multichannel audio on stereo speakers, with a surround effect
Parametric Equalizer	To compensate for uneven frequency response of built-in speakers
Automatic Volume Leveler	To maintain consistent volume level across content

Enabling Audio Effects on Android

OpenCORE, Android's Media Framework

OpenCORE provides essential media features in Android. It enables developers to build devices that support music applications, video creation and playback, video telephony, podcast services, real-time streaming and more.

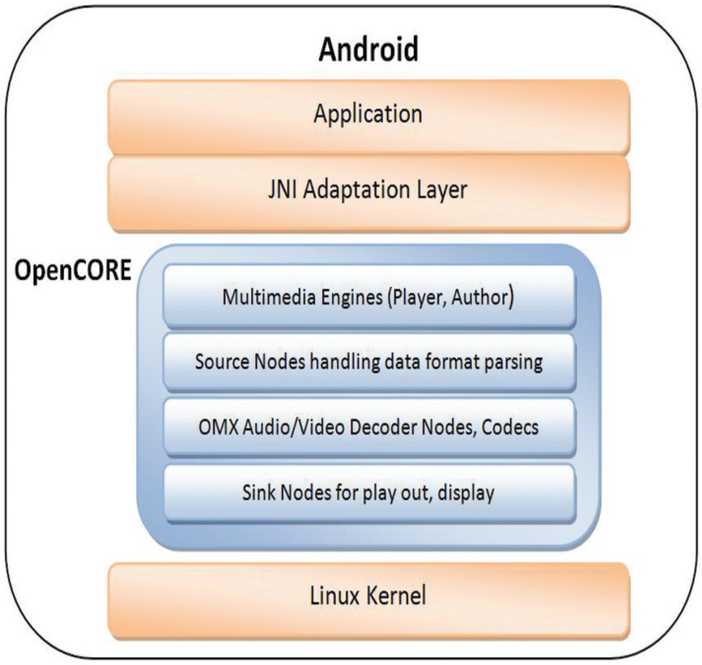

Figure 1: OpenCORE inside Android architecture

OpenCore uses an engine and node-based architecture underneath, written in the C++ programming language. The engine is responsible for managing and controlling the entire graph of nodes based on commands from the application. The types of nodes used and the graph configuration depend on the playback parameters such as source clip type and playback operation. In general, the node graph will start with a Source Node and end with a Sink Node (where the Sink Node can be replaced by an output media device directly).

It also includes a huge range of open-source multimedia codecs for Video (like MPEG-4, H.264), for Audio (like MP3, AAC, MIDI), as well as for Speech (like AMR NB, AMR WB), etc. OpenCORE uses OpenMAX IL to integrate the codecs within the framework. OpenMAX IL API (developed by the Khronos group) is a well known API used in embedded and mobile devices. It provides an abstraction layer for components, enabling portability across platforms.

With OpenCORE providing the complete backbone for multimedia playback, there is no native support available for audio effects/ audio post processing components and their integration mechanism. In the following sections, we describe in detail on how to integrate effects in Android.

Integrating Audio Effects into the Android Media Player

Integration of Effects in Android can be split into two broad steps.

- Adding the support of Effects libraries in the back-end of OpenCORE
- Adding the support from a Media Player Application into the OpenCORE media nodes [1]

Step 1: Adding the support of Audio Effects libraries in the back-end

OpenCORE provides various options for integrating a codec component. It can be added as an OpenMAX Component integrated with the OpenMAX codec nodes or as an OpenMAX codec node or as a Media I/O component. The Media I/O interface is used mostly for rendering the output, e.g. the display device for video. The possible approach for effects integration can be either as OpenMAX codec nodes or as an OpenMAX component. We evaluate both the approaches here.

As OpenMAX Codec Nodes: This approach requires changes in the Player Engine to support multiple audio effects. Each Effects component is implemented as a node, and created by the multimedia engine, when the datapath graph is created. The data transfer between audio decoder and Effects nodes is then handled by the Player Engine. It will come with a complexity of increasing number of nodes in the player graph, as more and more Effects get integrated. System Load will therefore also increase due to data transfer between multiple node threads.

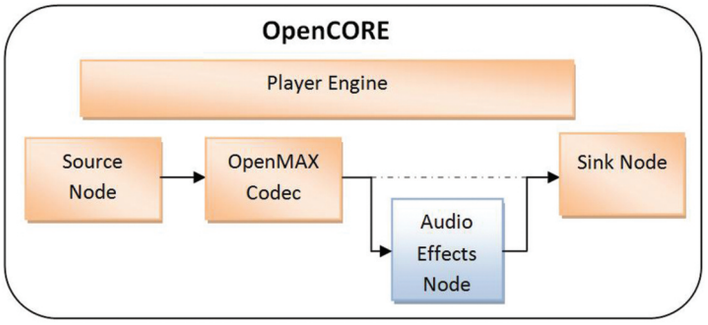

Figure 2: Integration of Audio Effects as OpenMAX codec nodes

As OpenMAX IL components: Since Audio Effects process the output from the audio decoders, it is best to place them at the same layer as audio decoders. OpenCORE has adopted the OpenMAX Integration Layer (IL) Interface for all multimedia codecs. This makes OpenMAX IL interface the straight-forward approach to integrate Audio Effects in Android.

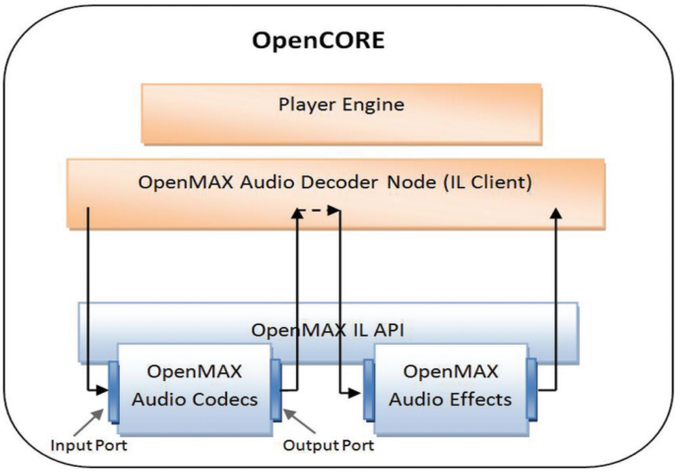

Figure 3: Integration of Audio Effects as OpenMAX IL components – Non-Tunneled Communication

The Audio Effect component is built as a static library, outside of Android, with ARM's highly optimized RVCT compiler. An OpenMAX IL wrapper is implemented around the component and built inside OpenCORE, along with the component library.

Audio Decoders and Audio Effects communicate with each other through the IL client using the IL APIs. The OMX Audio Decoder Node in OpenCORE acts as the OpenMAX IL client here. The creation, deletion and the normal data flow is maintained by this node. The node queues the commands and input buffers to the Audio Decoder component. The decoder then returns the processed input buffers back to the node. The node shall now pass the buffers to the Effects components, which are returned back to the node after applying the appropriate processing.

Although this approach has lesser system loading and delay as compared to the first approach, this can still cause a measureable increase in CPU loading. The data needs to be transferred from the OpenMAX audio node to the underlying OpenMAX components twice. This drawback of non-tunneled communication can be handled by setting up a tunnel between the two components.

Tunneled Communication between Audio Decoder and Audio Effects: The OMX Audio Decoder node design by default supports only the non-tunneled way of communication of OpenMAX IL. We explore the feature of tunneled communication provided by OpenMAX IL to further optimize system performance and for better utilization of resources.

Tunnel allows the two OpenMAX components to communicate directly with one another. The IL Core methods need not be used and datapath between the two tunneled ports can be directly managed by the components itself.

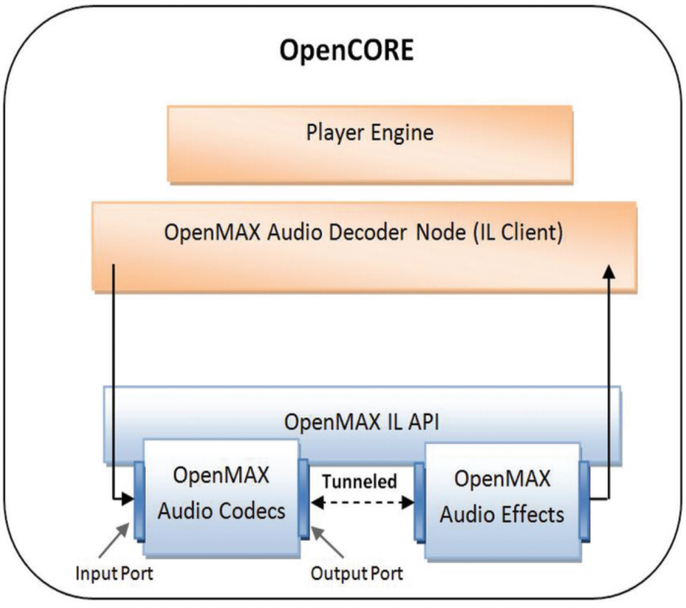

Figure 4: Integration of Audio Effects as OpenMAX IL components – Tunneled Communication

Let's take the example of an MP3 Decoder and Stereo Widening. In this case, the tunnel is established between the Output Port of the MP3 decoder and the Input Port of Stereo Widening Component. The OMX Audio Decoder Node establishes the tunnel at initialization, after checking whether the two ports required to be tunneled are compatible by checking Port parameters.

In the non-tunneled design, the OMX Audio Decoder Node was used to provide both input and output buffers to both the components. In the case of Tunneled communication, only input buffers are provided to the MP3 Decoder component and output buffers to the Stereo Widening component. The MP3 Decoder is the buffer allocator and supplier for its output port and the input port of the Stereo Widening component. The additional system load of transferring data through the OpenMAX IL API between the output port of MP3 and input port of Stereo Widening is removed. The transfer of data buffers happens through the already established tunnel now.

The OMX Audio Decoder Node needs to be modified to handle the creation/deletion, state transition of the Audio decoder as well as the Effects component. The Port Commands/Notifications/Callbacks need to be handled for both the components.

Comparing the above methods, the OpenMAX IL approach of integrating components with tunneling, is the optimal way, with minimal changes in the OpenCORE framework.

Extending it to multiple Audio Effects in tandem: Typically, a user would like to experience more than one audio effect at a time, and then determine which set of effects and presets suit them best. This approach can easily be used to tunnel one effect with another in a pipeline without much overhead.

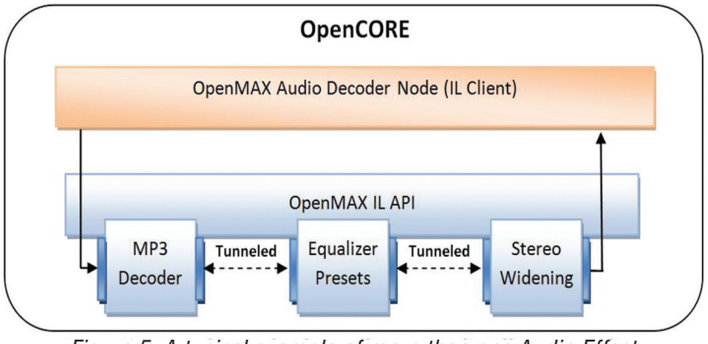

Figure 5: A typical example of more than one Audio Effect in a pipeline

The same approach can be extended to integrate post-processing effects for Video and Imaging codecs.

Step 2: Adding the support from Application Layer till OpenCORE

Since features like Audio Effects are tied to the user's choice, it is a requirement to provide control in an application to disable or enable a particular effect while playing music.

Android applications are written in Java language and run on Dalvik (Android's Java Virtual Machine). This Java application layer is interfaced to the underlying native C++ code of OpenCORE through a JNI adaptation layer implemented in Android.

The percolation of commands from application to the Media framework happens through various layers. The user commands from an application are passed through the OpenCORE media framework to the OMX Audio Decoder Node, which then passes these commands to the Audio Effects OpenMAX components using the OpenMAX APIs.

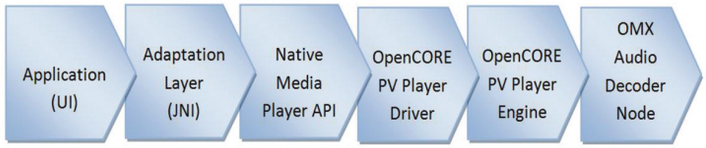

Figure 6: Informative call stack from Application Layer till OpenCORE

We need to extend the default Media Player application on Android, to add support for controlling Audio Effects. To do this, the Media Player class in Android (android.media.MediaPlayer) needs to be extended to add another API for Audio Effects. Once the API interface has been provided in the "android.media.MediaPlayer", the application can call this method with appropriate arguments to be passed downstream. The JNI Interface is also updated to translate this API from Java to the underlying C++ implementation of the Media Framework, OpenCORE. The JNI layer translates/passes it to the Media Player's (called PV Player) native implementation of this function.

Conclusion

To pass the command from PV Player into the Audio Decoder OMX Node, the Effects API gets translated from PlayerDriver layer commands to PlayerEngine commands, and then to the Audio Decoder OMX Node. Audio Decoder OMX Node then passes the command to the Effects component through OpenMAX API as explained in the previous section. This completes the integration of Audio Effects in Android Media Player. Enhancing the user experience while listening to music can not only give a differentiation to the end product, but also can be a key attraction for the user[2]. At Ittiam, we have enabled multiple audio effects and post processing solutions on Android. Ittiam has a wide range of high quality audio effects like Stereo Widening (for headphones and speakers), Virtual Surround for speakers, Parametric & Graphic Equalizers, Psychoacoustic Bass Enhancement, Reverb, Loudness Compensation and Automatic Volume Leveler to mention a few. With market-proven solutions in OpenCORE codec integration and in-depth knowledge of the Android application development, we have achieved demonstrable solutions on Android, showcasing complete Media Player applications with Audio effects.

References

1. Android Developer's website http://developer.android.com/
2. OpenMAX IL Specification http://www.khronos.org/#tab-openmax
3. OpenCORE http://www.opencore.net/

END