Volume 2, No. 7, July 2011

Journal of Global Research in Computer Science

ISSN-2229-371X

RESERCH PAPER

Available Online at www.jgrcs.info

EFFICIENT AUDIO PROCESSING IN ANDROID 2.3

Kailash Pathak* and V.P. Singh
Computer Science & Engineering Department
Thapar University, Patiala
*kailashiete@gmail.com

Abstract: Android is a new mobile platform for mobile development, it is open platform. It has a rich set of applications. Audio/video is become very important part for the success of any mobile platform. Every day new idea is come in to the picture to improve the quality of audio/video. A part from quality, battery power management is also important. Battery power is always on the top of the requirement stack. In android multimedia architecture, audio effects are handling with Advance RISC Processor (ARM), which is core processor of mobile device. Many multimedia applications are handled by ARM processor so result is more battery power consumption due to availability of large multimedia application. In this research paper, a new algorithm is proposed for the audio routing to save the battery power. The audio effects will be integrated with the help of digital signal processor instead of ARM processor. The power consumption of this application is tested and proposed architecture takes less power as compare to the present architecture.

Key words: Android, Multimedia, Audio Routing, ARM

INTRODUCTION

Digital audio effects are used in various audio applications, such as multi-effectors, and mobile audio devices [2]. The main goal of digital audio effects is the modification of the sound characteristic of the input audio signal. There are many audio effect algorithms such as reverb, virtualizer, and stereo widening, Bass boost.

Related Work

There has been done huge research work for application-level optimizations for mobile phones. Matt Calder et al. illustrated the significant energy savings that can be achieved via batch scheduling of recurrent mobile phone applications. Recurrent applications are primarily characterized by repeated execution, mostly as a background process, to perform a task periodically. A secondary characteristic is that these applications are also delay tolerant. For phones with deep sleep modes, there is also a cost in terms of time for the phone to wake up and go to sleep. When these recurrent applications are left to schedule themselves independently of other recurrent applications, they can have a very negative impact on battery life by bringing the phone out of sleep mode an unnecessary number of times [3]. Xiao et al. has explored the power characteristics of 3G and Wi-Fi but focus on mobile YouTube viewing [25]. Haitao Wul et al. has been discussed the Power saving Mode (PSM) with integrated Wi-Fi (Network Interface Card), in their research work they proposed footprint, leveraging the Cellular information such as the overheard cellular tower IDs and signal strength. The number of unnecessary scans through the changes and history logs of the mobile user's location has been studied.

Android Media Layers

Maoqiang et al. explored the android media layers as shownin Figure [1].

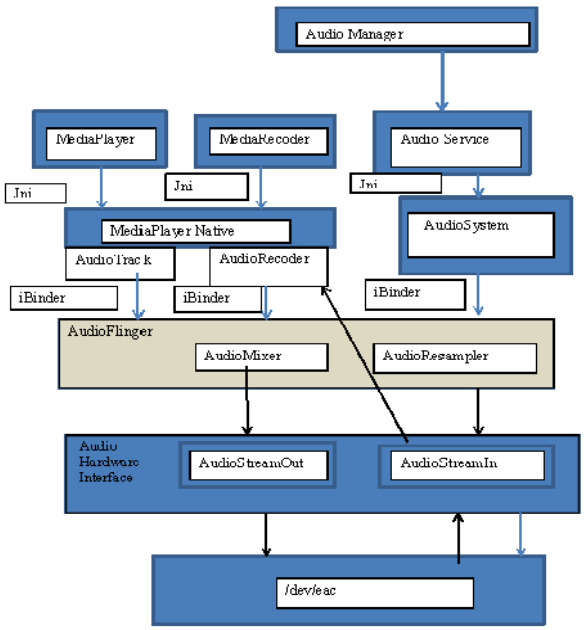

Figure 1. Android media layers

Media layers are used to control the playback of audio and video. Binder and IBinder perform inter process communication in android. Audio Flinger does mixing and resembling of android data during the audio processing.

AUDIO ROUTING IN ANDROID PLATFORM

The management of audio routing is scattered among various applications and services such as: Phone Application, Advanced Bluetooth Audio Distribution Profile Service (BluetoothA2dpService), Audio Service, With regards to routing the phone can be in one of three modes: NORMAL, RINGING, IN_CALL. The mode is selected by the Phone Application via Audio Manager set Mode (). To each of these modes corresponds a routing of the audio output to one of the following audio devices such as:

Speaker, Earpiece, Headset, Bluetooth Synchronous Connection Oriented (BT SCO). Audio (acoustic) post/pre-processing is controlled by Audio hardware according to the requested routing. The routing can be modified by some services or applications Phone Application (enable/disable BT SCO, enable/disable speaker phone), Headset Observer (enable/disable headset). An additional layer of routing is added for A2DP headset: A2DP output is enabled/ disabled by BluetoothA2dpService via audio manager, and handled by Audio Flinger. Audio Flinger uses a second mixer thread and hardware interface dedicated to A2DP.

Audio Flinger

Audio Flinger is mainly in charge of resembling and mixing audio streams (Audio Tracks) and sending the mix to the audio output at the Audio Hardware Interface. Audio Tracks have a stream type attribute that is used to control common volume settings and some routing decisions. The stream types are: MUSIC, RINGTONE, NOTIFICATION, ALARM, VOICE_CALL, and BLUETOOTH_SCO. It supports only two hardware interfaces; each one with a single audio output stream, one output is for bluetooth A2DP devices, the other one for all other audio devices attached to the platform (Speaker, Earpiece, Headset, and Bluetooth SCO). Hardware and A2DP outputs can be active simultaneously, the output to which a given track is sent is selected based on the stream type (MUSIC, SYSTEM, VOICE_CALL go to A2DP if enabled). The Hardware output is duplicated to the A2DP output when active.

AUDIO ROUTING

It Support multiple hardware outputs and inputs simultaneously. Support direct forwarding of audio input stream to hardware output. Audio routing is transparent process by Audio Flinger. By default no mixing, no sampling rate converter (SRC) to optimize performance for low sampling rate (SR) applications. Application such as: voice over IP (VOIP) has separate control of audio input and output routing.

Compressed Audio Processing

Android multimedia framework has compressed audio APIs for processing compressed audio, applications can send compressed audio directly to hardware to enable some kind of hardware tunneling mode. When A2DP headset is connected or low power mode is selected the switching to/from hardware tunneling to software/hardware decode mode is performed. Hardware tunneling do faster decoding than real-time decoding,

Audio framework Feeds a large pulse coded modulation (PCM) buffers to allow more frequent power collapse and power savings. It provides the controls of volume and audio post processing to the hardware tunneling**.**

Audio Effects: These effects provide a software implementation of environmental reverberation and graphic equalizer effects. Allow insertion of audio effect plug-ins on each track and replacement of default SRC algorithms. It also provides APIs to get/set hardware acoustic parameters for pre/post process implemented in audio hardware**.**

Routing Manager

As stated above, routing management in current implementation is scattered in different layers and modules of the media framework, making it hard to understand and maintain. The proposal of this research is to regroup all the routing intelligence under a single routing manager. Other applications or services won't send direct routing requests anymore to Audio Manager but instead will send events (e.g. indicating a state change or a device connection) that will be processed by the routing manager and transformed to a routing request send to audio hardware layer. Routing manager gives the control of the physical audio source device to the application in virtual mode, which is not directly visible to the application. The routing strategy becomes more complex implementation of different layers or modules.

Routing Work Flow: Routing modifications take place when a change in some variables governing a given strategy occurs:

a) A removable device is connected/disconnect. For instance when playing music and a wired headset is connected: reroute output from speaker to headset.
b) The phone state changes: Simultaneous output of a ring tone to both speakers and headset by opening a second output routed to headset if the phone state changes to RINGING.
c) The user selects speaker On/Off from in the call screen: The output is routed to speaker Phone Routing decision for each track follows the applicable strategy.

Routing decisions can yield to:

a) Opening/closing a secondary hardware output: A default output is always active but if the use case requires a second output, a new one is opened.
b) Reroute an existing output to a different device or set of devices
c) Route a track directly to a hardware output. This can happen if the audio format is compressed or if explicitly requested when opening the track.

COMPRESSED AUDIO ARCHITECTURE

The power efficient rendering of compressed audio via hardware tunneling is a key feature missing from vanilla media framework implementation. The implementation of this feature is not trivial because of the following issues:

a) The behaviour depends on actual hardware capabilities There must be some means to retrieve those capabilities and strategies defined for both cases where hardware tunnelling is supported or not.
b) The behaviour depends on current phone state, it may not be possible to use hardware tunnelling if the user interface (UI) is active and precise progress information must be displayed. This implies that smooth transition from/to hardware tunnelling mode must be provided.
c) Software mixer has volume and mute controls, which are apply as per the stream type.
d) The proposed solution must enable the use of hardware tunnelling for content types that are not handled by packet video (PV) opencore.

The proposal is based on the addition of support for compressed audio formats to Audio Tracks. An Audio Track opened with a compressed format will be handled by

a dedicated mixer thread in Audio Flinger connected directly to a compressed audio output at the Audio Hardware Interface. This mixer thread will coexist with other mixer threads handling the mixing of PCM Audio Tracks to PCM output streams at the Audio Hardware Interface. The change implies that compressed format is also supported at the Audio Hardware Interface level. The Audio Hardware and audio driver implementations must provide means to forward compressed audio to audio DSP at the same time as providing usual PCM output for streams handled by the software mixers. If the platform implements an openmax integration layer (IL) framework to manage hardware accelerated resources or hardware mixers, the audio output can be wrapped in a source component. Media Player Service and open core are required to automatically select playback over hardware tunneling.

In this research work two algorithms for bass booster and virtualizer audio effects have been discussed. Reverb effect can be implemented by the same idea as bass booster and virtualizer effect.

Algorithm of Bass Booster Audio Effect

Step1: Realizing the SL Engine in synchronous mode. If this fails call Exit On Error (Boolean).
Step 2: Set the source of audio data to play apply Data sinks for the audio player set the Interfaces for the audio player.
Step 3: Create output mix object which is used by the player and set the Output Mix Object in synchronous mode.
Step 4: Configuration of the player, Set arrays required [] and iid Array[] for required Interfaces with default TRUE
Step 5: Setup the data source structure for the URI locator_Field_descriptor.locator_Type
Step 6: Create the audio player, to play the file and if any error occurs call Exit On Error (error_result).Realize the player in synchronous mode.
Step 7: Start the data prefetching by setting the player and Wait until there's data to play.
Step 8: Set duration
SL_millisecond duration In Msec = SL_TIME_UNKNOWN;
Step 9: Configure Bass Boost
Step 10: Make sure player is stopped, Destroy the player object, Destroy Output Mix object.

Algorithm for Virtualizer Audio Effect

Step 1: Realizing the SL Engine in synchronous mode, If it get fails call Exit On Error (),
Step 2: Objects this application uses: one player and an ouput mix, and give Source of Audio data to play.
Step 3: Data sinks for the audio player SL Data Sink audio Sink;
SLData Locator_OutputMix locator_outputmix;
Play and Prefetch Status interfaces for the audio player.
Step 4: Get the SL Engine Interface which is implicit
Step 5: Configuration of the output mix, Create Output Mix object to be used by the Player Realize the Output Mix object in synchronous mode. Setup the data Sink structure.
Step 6: Set arrays required [] and iidArray [] for SL Prefetch StatusItf interfaces SLPlayItf is implicit.
Step 7: Setup the data source structure for the URI
Step 8 : Create the audio player, if it fails call Exit On Error ();
Step 9: Realize the player in synchronous mode and Get the SL Play Itf, SL Prefetch StatusItf and SL Android Stream TypeItf interfaces for the player.
Step 10: Playback and test, Start the data prefetching by setting the player To the paused state. Wait until there's data to play.
Step 11: Start playback
Step 12: Configure Virtualizer
Step 13: Switch Virtualizer on/off every TIME_S_ BET WEEN_VIRT_ON_OFF
Step 14: Make sure player is stopped, if it fails call Exit On Error (result);
Step 15: Destroy the player object.
Destroy Output Mix object.

RESULTS AND DISCUSSION

The main objective of this research work is to reduce the power consumption in the android multimedia framework, It is observed from figures 2- 4 that the audio effects, bass booster, reverb and virtualizer takes less power using the proposed audio routing algorithm as compared to present multimedia framework.

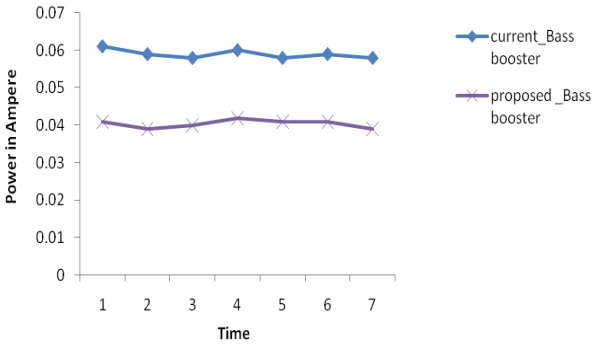

Figure 2. Power consumption in unit time for Bass booster effect.

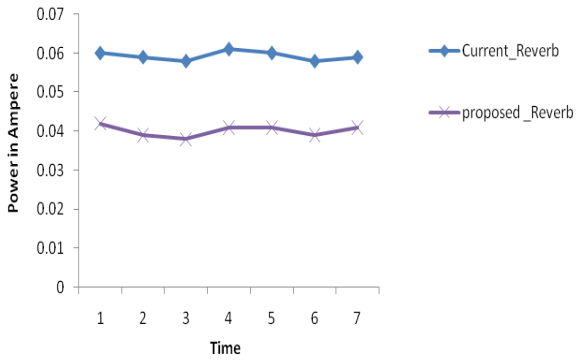

Figure 3. Power consumption in unit time for Reverb effect.

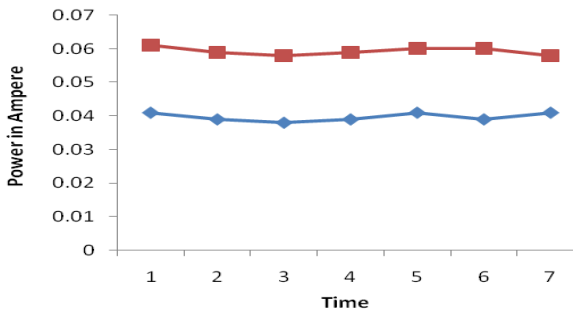

Figure 4. Power consumption in unit time for virtualizer effect.

CONCLUSION

It is evident from the results that the power consumption by proposed audio routing algorithm is reduce by the 20% power of mobile battery as compare to the present audio routing algorithm. The proposed framework may be further extended in android multimedia to accelerate the audio effects and more power gain.

REFERENCES

[1] Maoqiang Song, Wenkuo Xiong, Xiangling Fu, Research on Architecture of Multimedia and Its Design Based on Android, Internet Technology and Applications, 2010 International Conference, Wuhan 20-22 Aug. 2010, pp 1 – 4.

[2] Kyungjin Byun, Young-Su Kwon, Seongmo Park, and Nak-Woong Eum "Digital Audio Effect System-on-a-Chip Based on Embedded DSP Core", ETRI Journal, Vol 31, Number 6, December 2009 pp 732-740.

[3] Matt Calder and Mahesh K. Marina "Batch Scheduling of Recurrent Applications for Energy Savings on Mobile Phones" Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on, 21-25 June 2010. 978-1-4244-7151-5/10/$26.00 ©2010.

[4] Y. Xiao, R. Sri Kalyanaraman, A. Yla-Jaaski. "Energy Consumption of mobile YouTube: Quantitative Measurement and Analysis" 2008 the Second International Conference on Next Generation Mobile Applications, Services, and Technologies, pp.61-69,

[5] H. Wu, K. Tan, J. Liu, and Y. Zhang. "Footprint: Cellular Assisted Wi-Fi AP Discovery on Mobile Phones for Energy Savings." In Proc. ACM WiNTECH, 2009

[6] Yu-Sheng Lu, Chin-Ho Lee, Hung-Yen Weng, Yueh-Min Huang "Design and Implementation of digital TV widget for Android on multi-core platform" Computer Symposium (ICS), 2010 International, Tainan 16-18 Dec. 2010,pp 576 – 580.