Optimizing the Audio Decoding Based Upon Hardware Capability: An Android NUPlayer Implementation

Prasad Renukdas, Prof. Swarnalatha P

School of Computer Science and Engineering, VIT, Vellore, Tamil Nadu, India

Abstract— *Performance in android based devices is a major concern. A number of frame-drops can be seen while playing a 4k video on a less powerful device. A multi-purpose android set-top box should be capable of playing the 4k videos alongside of its basic functionalities. Sometimes the hardware allotted for the set-top box may not be up to the mark. Hence, it is difficult to handle 4k videos at a smoother rate. To tackle the above problem we have come up with the solution which tunnels the compressed/uncompressed audio track directly to the Smart-TV based upon its capability. The solution is built for ST's set top box. In this paper we discuss basics of set top box, android audio architecture and the solution for frame drop backed by the experimental results and by using standard testing methods.*

Keywords— Smart-TV, tunneling, 4k videos, frame drops.

I. INTRODUCTION

Android set-top box is an ongoing R&D project at in most of the electronic tech giants like STMicroelectronics. This set-top box runs the latest android lollipop. As the cable television and satellite television industries enjoy years of sustained growth, cable/satellite operators are now investing in the "digitization" of their systems. This is resulting in the transition of the TV transmission infrastructure from an analog environment to a digital one. To use current analog television sets to receive these digital broadcasts Set Top Box (STB) is necessary. Figure 1 show the flow of signal from satellite antenna to TV.

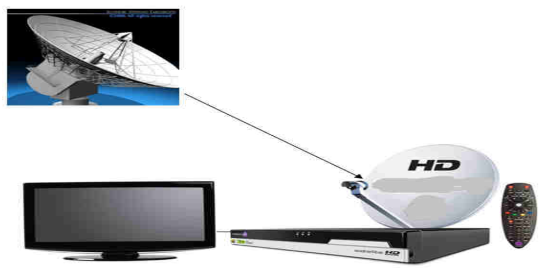


Fig.1: diagram showing flow of signal from satellite antenna to TV

Set-Top Box or STB has become an integral part of TV viewing in many parts of the world. We commonly see this sleek looking device sitting on side of TVs. Though this device looks slim and simple but it is one of the most complex embedded systems today. STBs are increasing their feature set day by day. Few of the common features in current generation STBs are time shift mode viewing, recording, Internet based viewing, video on demand, Full High definition video output etc.

In this paper we will be discussing the basic android audio architecture which is the preliminary requirement for understanding how the audio calls are handled. We will also discuss analysis and design of our system. In the final step we will see snippets of implementation and test results.

II. OVERVIEW OF THE PROPOSED SYSTEM

In this section we will see the general architecture of STB. We will also see the android audio architecture which will give us a brief understanding regarding android audio methods invocations.

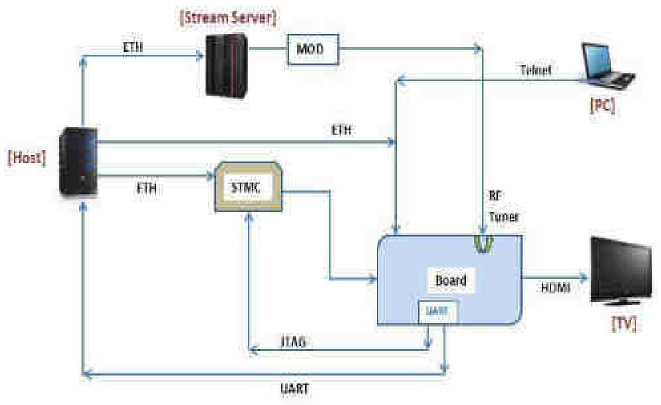


Fig. 2:STB Architecture

Figure 2 shows the complete set up used for the implementation and testing of the module. We used ST's Cannes 2.5 board for development and testing. An STMC box was used for sending the images to the Board via host computer. Stream server is an optional entity which was used for testing live streams.

SoC

The System on chip contain two ARM processors ST SDK2 like Orly B2020, one SH-4 (used for floating point calculations) on ST SDK1 like lille, leage c, cardiff, one ST231 and one MALI (used as a graphics engine) processor.

Stream Server

The stream server contains all the streams to be viewed on the receiver in digital format, it is connected to a MOD card which converts the signal to Radio Frequency and sends it the TUNER card on the board.

III. SDK2 SOFTWARE OVERVIEW

ST has a comprehensive media framework based on hardware codec blocks available in Cannes 2 (STiH410) platform. The framework is known as Streaming Engine (henceforth called SE) which has a complete player implementation (including AV sync) and is implemented as a kernel module. However Android mandates the use of OMX IL APIs for all codec related functionality and hence ST provides an OMX IL layer based on the underlying SE kernel module. The OMX IL is implemented in both user space and kernel space. The kernel space module is called OMXSE (OMX interface to the underlying Streaming Engine). Apart from this the V4L2 and DVB APIs are also used. The user space OMX IL consists of a set of C++ classes for audio and video omx components and ports which handle the OMX state machine. Internally they make use of the V4L2 and OMXSE kernel drivers to access the underlying hardware codecs.

The following figure shows implementation of the OMX IL.

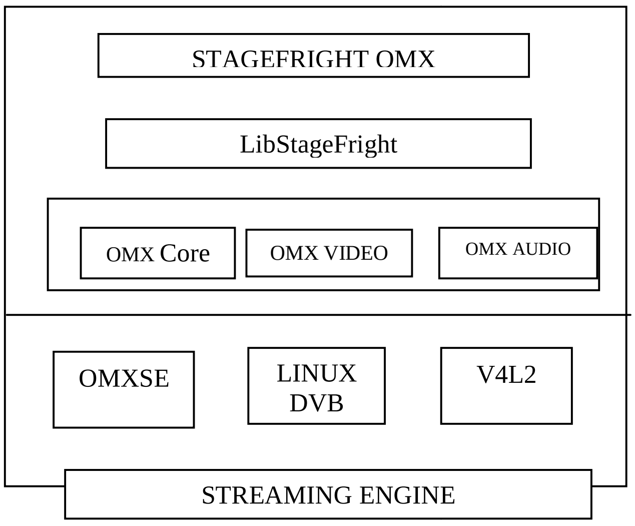


Fig.3:SDK2 architecture

IV. NUPLAYER OVERVIEW

NuPlayerDriver provides the implementation of MediaPlayerInterface and is used for all kinds of streaming playback. However NuPlayerDriver is more of a wrapper and the actual implementation is through the NuPlayer class. NuPlayer control flow is fully asynchronous and based on the ALooper/AHandler classes described in the following diagram:

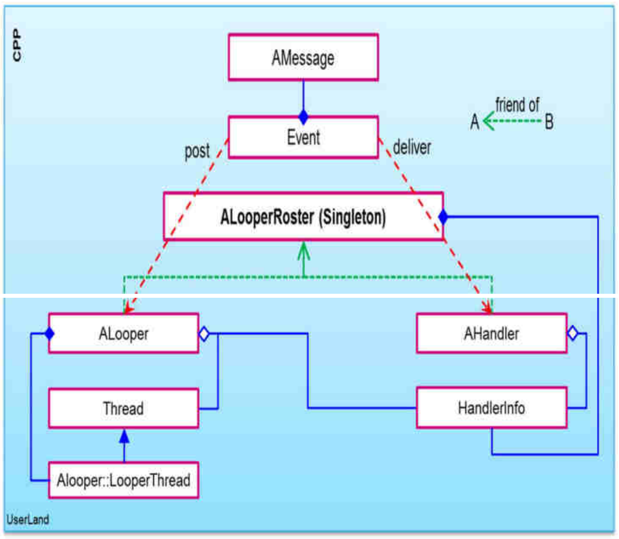


Fig.6:NUPlayer overview

Foundation Classes- Messages

Class names start with letter “A” for “asynchronous”. Following are the main classes

- AMessage–Represents data which needs to be handled asynchronously. Contains a field called “what” representing the kind of message. Contains an array of data items. Each data item has a unique name and by using a C union support is provided for all kinds of data types including object pointers. Has methods to convert a Binder parcel into an AMessage and vice-versa. Event–An event is a timed “AMessage”.
- AHandler–Represents an entity which can process an AMessage. Processing is done in onMessageReceived() function.
- ALooper–Represents a thread which runs an infinite loop. Maintains a queue of events. Anyone can post an event to an “ALooper” which is put at the end of event queue. The thread loop removes events from the queue (based on timing of the AMessage) and delivers them to an appropriate handler.
- ALooperRoster–Singleton object which keeps a list of ALoopers and associated AHandlers. The interaction

between looper and handler is done via this singleton object. Each looper needs to associate a handler with itself. When processing the event from its queue, a looper delivers the message to the associated handler.

NUPlayer Classes

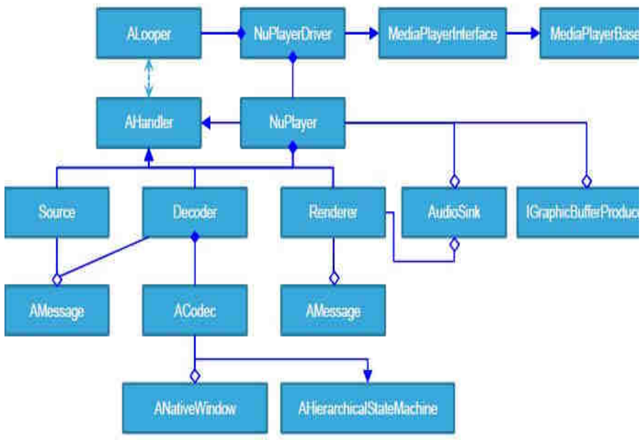


Fig.4:NUPlayer Classes and their relation

NuPlayer is designed for streaming use-cases and exclusively follows asynchronous message based control mechanism. The IMediaPlayerService::create() API gives an IMediaPlayer interface and calling IMediaPlayer::setDataSource() creates a MediaPlayerBase object which is an NuPlayerDriver object in case of streaming playback. NuPlayerDriver's main role is to act as a looper to handle message passing mechanism while NuPlayer acts as the message handler and performs the actual functions of a media player. NuPlayer contains an object each of a Source, a Decoder and a Renderer. Each of these classes is defined inside the scope of NuPlayer class. Each of them acts a message handler and is associated with the same looper which is implemented by NuPlayerDriver.

NuPlayer also maintains reference to the audio sink and native window passed by the application/client. These are used for audio/video rendering. Note that the Renderer class only manages the A/V sync and actual rendering is done by the Decoder class itself.Decoder also maintains the reference to application passed native window and uses it to get decoder output buffers and rendering the video output. Actual decoding is implemented via ACodec class which is the "asynchronous" equivalent of OMXCodec (part of Stagefright player) and the Decoder class is just a wrapper for ACodec. ACodec implements AHierarchicalStateMachine and follows the standard pattern of OMX component state.

In a typical playback usecase, the application invokes following set of API calls.

```
MediaPlayer player = new MediaPlayer();
player.setDataSource(url);
player.setSurface(surface);
player.prepare();
player.start();
player.stop();
player.release();
```

NUPlayer DataFlow Sequence diagram

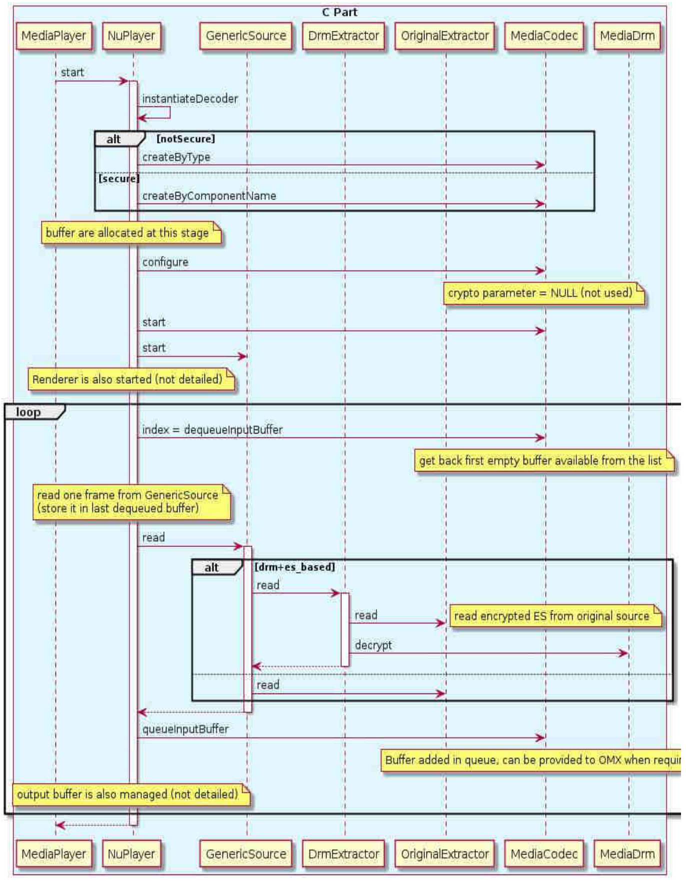

C Part
MediaPlayer
NuPlayer
GenericSource
DrmExtractor
OriginalExtractor
MediaCodec
MediaDrm
start
instantiateDecoder
alt
[notSecure]
createByType
[secure]
createByComponentName
buffer are allocated at this stage
configure
crypto parameter = NULL (not used)
start
start
Renderer is also started (not detailed)
loop
index = dequeueInputBuffer
get back first empty buffer available from the list
read one frame from GenericSource (store it in last dequeued buffer)
read
alt
[drm+es_based]
read
read
read encrypted ES from original source
decrypt
read
queueInputBuffer
Buffer added in queue, can be provided to OMX when requi
output buffer is also managed (not detailed)

Fig.5: NUPlayer sequence diagram for DataFlow

V. IMPLEMENTATION AND TEST RESULT SNIPPETS

A demon is created which reads the TV's capabilities for audio decoding. If TV is capable of decoding the audio stream, AUDIO_OFFLOAD flag is set in the NUPlayer class which bypasses audio directly to the TV hardware through hal. When the streams are passed by NUPlayer to the underlying audio player, it will simply forward it to the TV. Following are the steps in the implementation:

1] Daemon reads the HDMI capability on startup.

2] Read data is stored in the data structure.

3] Following steps take place when a 4k Video stream request takes place:

3.1 NUPlayer reads the hardware capability

3.2 Is TV capable of decoding?

3.3 If yes then pass the compressed audio streams directly to TV.

4] Audio and Video sync is handled by TimedQueueEvent

Above method reduces the overhead of audio decoding under the assumption that TV is capable of performing audio decoding. If TV is not capable then the the decoding will be handled by the board itself.

Following is the code snippet for the daemon reading hardware capability.

```
#include <string.h>

#include "client/inc/hdmi_hotplug_client.h"

static void help() {
    printf("usage:\n");
    printf("\tto get video/audio standard: # hdmi_hotplug_test\n");
    printf("\tto set video standard: # hdmi_hotplug_test -v <v4l2std
standard>\n");
    printf("\tto set audio standard: # hdmi_hotplug_test -a
<LPCM/AC3/\"Dolby Digital+\">\n");
    printf("\tto set a/v standard in 1 command: # hdmi_hotplug_test -v
<v4l2std standard> -a <LPCM/AC3/\"Dolby Digital+\">\n");
}

int main(int argc, char** argv) {
    if(argc == 1) {
        char desc[20];
        int ret = stmHdmiHotplug_getResolution(desc);
        if(ret == 0) {
            printf("Current display standard is '%s'\n",desc);
        } else {
            printf("Cannot get Current display standard\n");
        }
        ret = stmHdmiHotplug_getAudMode(desc);
        if(ret == 0) {
            printf("Current audio standard is '%s'\n",desc);
        } else {
            printf("Cannot get Current audio standard\n");
        }
    } else if(argc > 2 && (argc+1)%2 == 0) {
        char prevDesc[20], newDesc[20];
        prevDesc[0] = newDesc[0] = '\0';
        for(int i = 1; i<argc; i+=2) {
            prevDesc[0] = newDesc[0] = '\0';
            if(!strcmp(argv[i], "-v")) {
                int ret = stmHdmiHotplug_setResolution(argv[i+1],
prevDesc, newDesc);
                if(ret >= 0) {
                    printf("Current display standard is '%s'\n",prevDesc)
                    printf("New display standard is '%s'\n",newDesc);
                } else {
                    printf("Cannot set New display standard\n");
                }
```

Fig. 6 Daemon code snippet

Performance testing

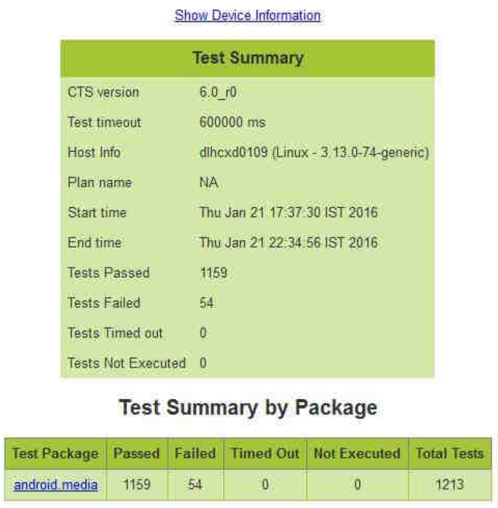
Performance of 4k videos was tested using a monitor tool. Monitor tool is an ST-internal tool. It displays the frame drop rates, total frames displayed, queued, repeated and released.

We recorded certain readings before the implementation. Later we ran the same video for the same build. Following are some of the screenshots before and after the changes.

Fig.7: Monitor reading before the changes

Fig.8: Monitor tool reading after changes

We also ran CTS(Compatibility test Suite) by google for media playback. Initially there were a lot of test failures. After the implementation the failures were greatly reduced. Following are some of the snippets for CTS ran for media package.



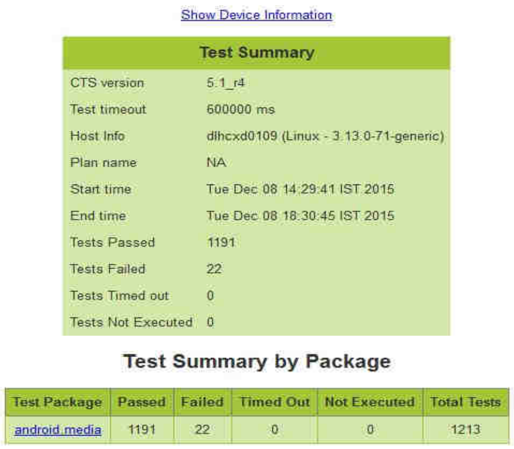
Show Device Information

Test Summary	
CTS version	6.0_r0
Test timeout	600000 ms
Host Info	dlhcxd0109 (Linux - 3.13.0-74-generic)
Plan name	NA
Start time	Thu Jan 21 17:37:30 IST 2016
End time	Thu Jan 21 22:34:56 IST 2016
Tests Passed	1159
Tests Failed	54
Tests Timed out	0
Tests Not Executed	0

Test Summary by Package

Test Package	Passed	Failed	Timed Out	Not Executed	Total Tests
android.media	1159	54	0	0	1213

Fig.9: CTS results before our patch



Show Device Information

Test Summary

CTS version	5.1_r4
Test timeout	600000 ms
Host Info	dlhcxd0109 (Linux - 3.13.0-71-generic)
Plan name	NA
Start time	Tue Dec 08 14:29:41 IST 2015
End time	Tue Dec 08 18:30:45 IST 2015
Tests Passed	1191
Tests Failed	22
Tests Timed out	0
Tests Not Executed	0

Test Summary by Package

Test Package	Passed	Failed	Timed Out	Not Executed	Total Tests
android.media	1191	22	0	0	1213

Fig.10: CTS results after our patch

If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

Results

We successfully improved the 4k video playback on STB by tunneling the audio directly to the tv. An initial lag was seen after the video starts. Later the video plays smoothly with very little frame drops.

)

VI. ACKNOWLEDGMENT

I would like to thank Deepak karda for mentoring me in this project. I would also like to extend my thanks to Munish Bhardwaj and Prof. Swarnalatha for all the encouragement and support.

REFERENCES

[1] J. Barba, J.C. López, D. de la Fuente, F. Rincón, “OpenMax Hardware Native Support for Efficient Multimedia Embedded Systems”

[2] Karim Yagbmour, Embedded Android, published by Oreilly

[3] Sang-Pil Moon, Joo-Won Kim, Kuk-Ho Bae, Jae-Cheon Lee and Dae-Wha Seo, “Embedded Linux Implementation on a Commercial Digital TV System,” in IEEE Transactions on Consumer Electronics, Vol. 49, No. 4, NOVEMBER 2003

[4] Damian Hobson-Garcia, Katsuya Matsubara, Takanari Hayama, Hisao Munakata, “Integrating a Hardware Video Codec into Android Stagefright using OpenMAX IL”