Creating a Hardware Decoder Integrating FFmpeg with MediaCodec for Intel® Atom™-Based Android* Platforms

Introduction
The electronics market is teeming with a multitude of Android* devices with different versions of the OS and different hardware. As a result, online video ISVs have a difficult time delivering their content. Since Android 4.1+ devices will occupy the market in the future, this article shows how to integrate FFmpeg with MediaCodec to support the many types of video formats and create a hardware decoder for Intel® Atom™-based Android platforms.

Background
Google created MediaPlayer as a simpler way to control audio and video playback. However, MediaPlayer is limited in that it offers only three media formats: mp4, 3gpp, and mkv (beginning with Android 4.0). However, most online video ISVs use other video formats like FLV, etc. To play unsupported formats and be compatible with the greatest number of Android devices in the market, online video developers have adopted FFmpeg as their software decoder of choice. Alternatively, some developers integrate OpenMAX* (a well-known standard in the media industry) to access Android's low-level hardware decoder for Android 2.3 devices, a codec to enable Android 4.0 devices, and MediaCodec to enable Android 4.1+ devices. They had to buy most of the popular Android devices in the market for testing. Even then, they still had some compatibility problems, so they had to adopt FFmpeg for backup.

FFmpeg with MediaCodec solution analysis
FFmpeg [1] is the leading multimedia framework, able to decode, encode, transcode, mux, demux, stream, filter, and play pretty much anything that humans and machines have created. FFmpeg can also provide the function of parsing video streaming.

MediaCodec was first introduced in the Android 4.1 release, based on the Java* API, and provides the interface to access low-level system codecs, either hardware codec, or, as with the audio codec, a highly optimized software codec. Use of MediaCodec can provide reasonable performance as well as power savings.

Besides mp4, 3gpp, and mkv video formats, which Android MediaPlayer can support directly, many other popular video formats in the market, such as flv, mov, rm, rmvb, and so on, are not. For these unsupported formats, online video ISVs usually adopt FFmpeg as their software decoder, which will cost more CPU workload and power.
One solution is to integrate FFmpeg with MediaCodec, which will support all the different kinds of video formats and create a hardware decoder for Intel Atom-based Android platforms.

The solution is shown in the following diagram:

**Figure1. Integrating FFmpeg with MediaCodec solution**

In this solution, FFmpeg unpacks the video container as separate raw video data and audio data in a native layer. Then, the raw video data is transferred into MediaCodec's APIs in the Java layer for hardware decoder. The audio data is decoded by FFmpeg directly, which does not cost much CPU workload. Finally, FFmpeg syncs the audio and video by time stamp.

This solution supports some other popular video formats that aren't supported by MediaPlayer, and online video ISVs are finding it helpful.
Building FFmpeg for Android on x86

FFmpeg has supported Android since the 2.1 release, but the FFmpeg package did not have a build script for x86 on Android. Following are the steps for how to build FFmpeg for Android on x86:

1. On FFmpeg's web site, download the latest FFmpeg release: ffmpeg-2.2.4.tar.bz2 [2].

2. Copy the FFmpeg package to a Ubuntu* build machine and extract using this “tar” command:

   ```
   wangsy@ubuntu:~/Desktop$ tar xvf ffmpeg-2.2.4.tar.bz2
   ```

3. Set up ANDROID_NDK_HOME environment with the "export" command:

   ```
   export ANDROID_NDK_HOME= $ ANDROID_NDK_HOME :/~/android-ndk-r9c
   ```

4. Copy the following configuration file to ~/ffmpeg-2.2.4 and add the "run" permissions:

   ```
   wangsy@ubuntu:~/Desktop$ cp config_build_x86.sh ~/ffmpeg-2.2.4
   wangsy@ubuntu:~/Desktop$ sudo chmod a+x ~/ffmpeg-2.2.4/config_build_x86.sh
   ```

5. Run the configuration script and build using the "make" and "make install" commands:

   ```
   wangsy@ubuntu:~/Desktop/ffmpeg-2.2.4$ make
   wangsy@ubuntu:~/Desktop/ffmpeg-2.2.4$ make install
   ```

   The generated Android for x86 libs are under ~/Desktop/ffmpeg-2.2.4/android/x86/lib$.

Developers can copy these generated libs for development. Because YASM [3] assembler compiler and Intel® Streaming SIMD Extensions (Intel® SSE) [4] are enabled in the config_build_x86.sh, the generated Android for x86 libs are optimized for extreme performance on Intel Atom-based Android platforms.

Integrate MediaCodec Java APIs from FFmpeg Native Layer

The MediaCodec class can be used to access the low-level media codec, i.e., encoder/decoder components. It is easy to call it from the Java layer. But in this solution, the FFmpeg unpacks the raw video data under the native layer, so we need to parse the MediaCodec functions from native layer and transfer the raw video data to these APIs.
Demo code: Accessing the MediaCodec class from the native layer:

```c
struct classname
{
    const char *name;
    int offset;
};
static const struct classname classes[] = {
    { "android/media/MediaCodecList", OFF(media_codec_list_class) },
    { "android/media/MediaCodec", OFF(media_codec_class) },
    { "android/media/MediaFormat", OFF(media_format_class) },
    { "android/media/MediaFormat", OFF(media_format_class) },
    { "android/media/MediaCodec$BufferInfo", OFF(buffer_info_class) },
    { "java/nio/ByteBuffer", OFF(byte_buffer_class) },
    { NULL, 0 },
};
JNIEnv* env = NULL;
ATTACH_THREAD;
for (int i = 0; classes[i].name; i++) {
    *(jclass*)((uint8_t*)p_sys + classes[i].offset) =
    (*env)->FindClass(env, classes[i].name);
    if ((*env)->ExceptionOccurred(env)) {
        msg_Warn(p_dec, "Unable to find class %s", classes[i].name);
        (*env)->ExceptionClear(env);
        goto error;
    }
```
Demo code: Accessing the MediaCodec functions from the native layer:

```c
struct member
{
    const char *name;
    const char *sig;
    const char *class;
    int offset;
    int type;
};

static const struct member members[] = {
    { "toString", "()Ljava/lang/String;", "java/lang/Object", OFF(tostring), METHOD },

    { "getCodecCount", "()I", "android/media/MediaCodecList", OFF(get_codec_count), STATIC_METHOD },

    { "getCodecInfoAt", "(I)Landroid/media/MediaCodecInfo;", "android/media/MediaCodecList", OFF(get_codec_info_at), STATIC_METHOD },

    { "isEncoder", "()Z", "android/media/MediaCodecInfo", OFF(is_encoder), METHOD },

    { "getSupportedTypes", "()Ljava/lang/String;", "android/media/MediaCodecInfo", OFF(get_supported_types), METHOD },

    { "getName", "()Ljava/lang/String;", "android/media/MediaCodecInfo", OFF(get_name), METHOD },

    { "createByCodecName", "(Ljava/lang/String;)Landroid/media/MediaCodec;", "android/media/MediaCodec", OFF(create_by_codec_name), STATIC_METHOD },

    { "configure", "(Landroid/media/MediaFormat;Landroid/view/Surface;Landroid/media/MediaCrypto;I)V", "android/media/MediaCodec", OFF(configure), METHOD },

    { "start", "()V", "android/media/MediaCodec", OFF(start), METHOD },
};
```
{ "stop", "()V", "android/media/MediaCodec", OFF(stop), METHOD },
{ "flush", "()V", "android/media/MediaCodec", OFF(flush), METHOD },
{ "release", "()V", "android/media/MediaCodec", OFF(release), METHOD },
{ "getOutputFormat", "()Landroid/media/MediaFormat;", "android/media/MediaCodec", OFF(get_output_format), METHOD },
{ "getInputBuffers", "()[Ljava/nio/ByteBuffer;", "android/media/MediaCodec", OFF(get_input_buffers), METHOD },
{ "getOutputBuffers", "()[Ljava/nio/ByteBuffer;", "android/media/MediaCodec", OFF(get_output_buffers), METHOD },
{ "dequeueInputBuffer", "(J)I", "android/media/MediaCodec", OFF(dequeue_input_buffer), METHOD },
{ "dequeueOutputBuffer", "(Landroid/media/MediaCodec$BufferInfo;J)I", "android/media/MediaCodec", OFF(dequeue_output_buffer), METHOD },
{ "queueInputBuffer", "(IIIIJ)V", "android/media/MediaCodec", OFF(queue_input_buffer), METHOD },
{ "releaseOutputBuffer", "(IZ)V", "android/media/MediaCodec", OFF(release_output_buffer), METHOD },
{ "createVideoFormat", "(Ljava/lang/String;II)Landroid/media/MediaFormat;", "android/media/MediaFormat", OFF(create_video_format), STATIC_METHOD },
{ "setInteger", "(Ljava/lang/String;I)V", "android/media/MediaFormat", OFF(set_integer), METHOD },
{ "getInteger", "(Ljava/lang/String;)I", "android/media/MediaFormat", OFF(get_integer), METHOD },
{ "setByteBuffer", "(Ljava/lang/String;Ljava/nio/ByteBuffer;)V", "android/media/MediaFormat", OFF(set_bytebuffer), METHOD },
{ "<init>", "()V", "android/media/MediaCodec$BufferInfo", OFF(buffer_info_ctor), METHOD },
{ "size", "I", "android/media/MediaCodec$BufferInfo", OFF(size_field), FIELD },
{ "offset", "I", "android/media/MediaCodec$BufferInfo", OFF(offset_field), FIELD },
Using the method above, developers can parse the MediaCodec class and functions from native layer. For more information on how to use them, go to http://developer.android.com/reference/android/media/MediaCodec.html.

**Summary**

The Intel optimization method was developed to help developers integrate FFmpeg with MediaCodec to support many different types of video formats and create a hardware decoder for Intel Atom-based Android platforms.
Related Articles


References

About the Author
Songyue Wang is a senior application engineer in the Intel Software and Solutions Group (SSG), Developer Relations Division, Intel® Atom™ Processor Mobile Enabling Team. Songyue is responsible for Android app enabling on Intel Atom processors. He focuses on optimizing multimedia performance on the Bay Trail platform, working closely with the most popular online video providers in the PRC region to enable the H.265/HEVC encoder and decoder solution and Intel® Wireless Display differentiation features on Android for x86 platforms.
Notices

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product or der.

Copies of documents which have an order number and are referenced in this document or other Intel literature may be obtained by calling 1-800-548-4725 or going to http://www.intel.com/design/literature.htm

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

Intel, the Intel logo, and Atom are trademarks of Intel Corporation in the U.S. and/or other countries.

Copyright © 2014 Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.