Whitepaper

Codec Abstraction Framework

Design Considerations for Embedded Applications

July-2009

Author: Vasudev Prasad S, Chaitanya Kumar Matcha

Abstract

Embedded applications have found their way into various markets like music players, video recorders, conferencing systems, unmanned vehicles and many more. These applications use combination of multiple video and audio codecs responsible for compression and decompression of media. Usually, such codecs are available in multiple flavors on multiple platforms, and the API of codecs also differ due to multiple reasons: Nature of codec (audio, video, image, speech etc.), standard (MPEG-2, H.264, etc.), profiles of codec (Main Profile vs. Base Profile H.264), processor wrappers using popular interfaces (ex: various versions of XDM, openMAX, etc.). Although modularity and layering provide an abstraction of the underlying components, serious challenges needs to be overcome in building stable and maintainable framework. The challenges are more serious when the framework needs to span across multiple platforms and multiple systems. This paper provides insight into designing such a **Codec Abstraction Framework**.

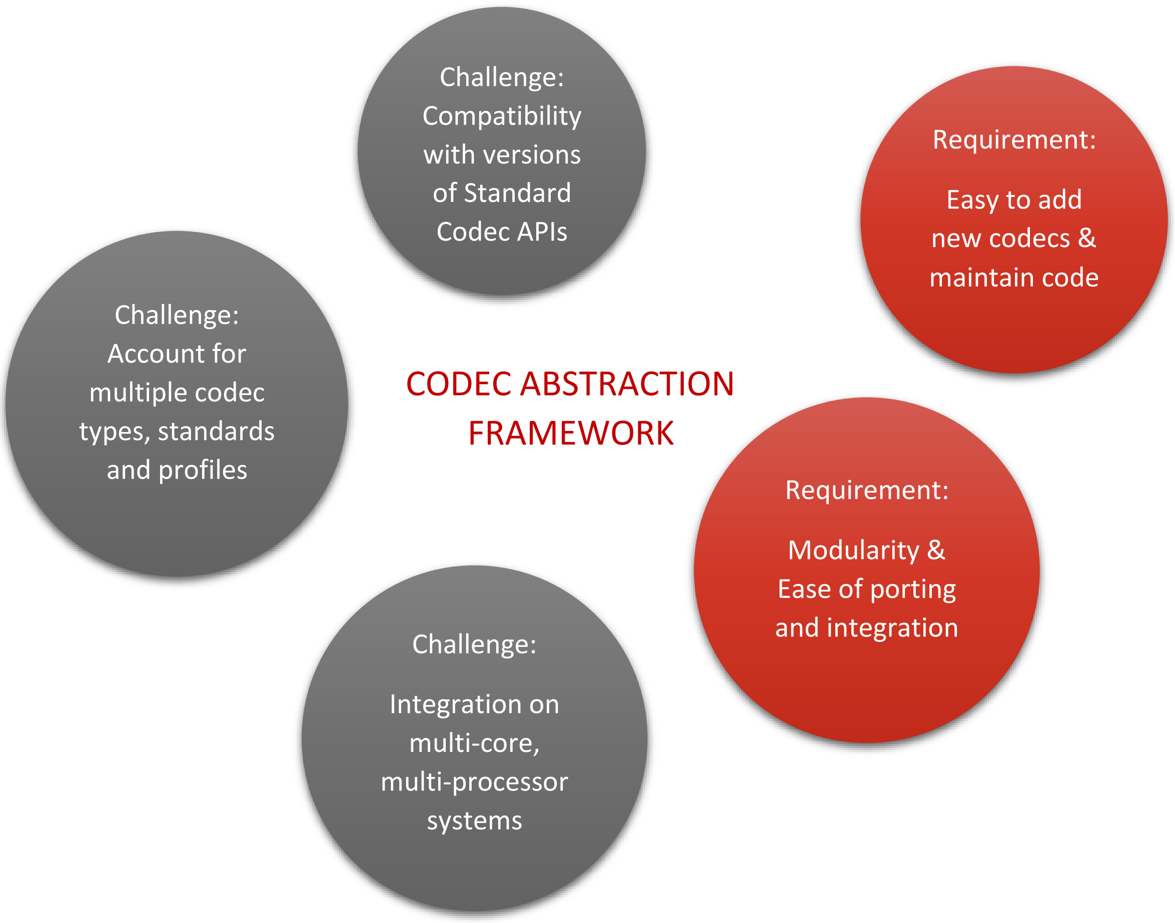
Challenge: Compatibility with versions of Standard Codec APIs
Requirement: Easy to add new codecs & maintain code
Challenge: Account for multiple codec types, standards and profiles
CODEC ABSTRACTION FRAMEWORK
Requirement: Modularity & Ease of porting and integration
Challenge: Integration on multi-core, multi-processor systems

Figure 1 – Challenges and Requirements for a Codec Abstraction Framework

Contents

Figures

Introduction

In order to address the problem described, it is highly desirable to have a single point of access to all codecs across the variations of APIs they export. This point of access should then provide a uniform interface to codecs and facilitate an easy application realization & porting across all the variants of the codec. This is what we term as the Codec **Abstraction Framework (CAF)**. This paper details the design considerations in building such Codec Abstraction Framework and proposes a design that addresses these considerations. The framework abstracts the applications from the underlying codec APIs and/or processor variations.

Embedded applications can be using a variety of codecs simultaneously on homogeneous environments (single processor as DSP only or GPP only) or heterogeneous environments (master slave like GPP + DSP). The codecs use the underlying resources and memory for their operation. If multiple codecs are to operate simultaneously we need a mechanism or a framework to manage the resources, communicate between GPP and DSP if needed and make cooperation of codecs possible. It should also abstract the application from the environment below and provide a logical set of API across all platforms. It should be able to schedule different operations in the case of multiple codecs and in heterogeneous environments.

Such frameworks form a basic abstraction layer. **TI's Codec Engine (CE)** and **Ittiam's Component Manager (CMR)** are examples of such frameworks. In spite of the abstraction provided by these frameworks, there are challenges in building scalable and portable applications. Since TI's Codec Engine is widely adopted, this paper concentrates mainly on building a framework above CE.

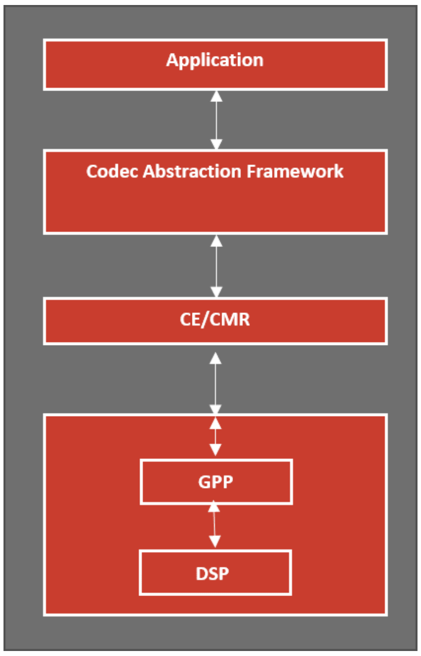

Figure 2 - Role of Codec Abstraction Framework

Design Considerations

As explained earlier, the major requirement is to provide abstraction from the varying API across codecs. This issue of varying API across codecs was to an extent resolved with the help of standard interfaces like XDM and XMI as well as frameworks like CE and CMR. They don't yet provide a uniform interface as XDM and XMI API are extendable and hence can vary across codecs. The CE's interface is not uniform across different XDM standards. Hence codecs on different XDM versions have different CE API. This framework that provides a complete abstraction of codec API is called **Codec Abstraction Framework** and has following design considerations:

- **Single point of access to all codecs**: The framework should provide a single point of contact for all codecs. The framework should also provide a single codec package which contains all the codecs. All applications will use this package without any modification.
- **Uniform API for all codecs**: The framework should export a common API for all codecs irrespective of the underlying codecs used. Applications will use this common API for all codecs being used.
- **Transparency**: The Framework, along with being a single point of contact to all codecs, should be able to configure the parameters statically and dynamically. In other words it should bring out all the features the codec exports and support the modes of operation the codec supports (like constant bit rate- CBR and variable bit rate- VBR).
- **Easier migration of codecs to new XDM standard**: The codec upgrading from the present XDM version to further XDM version should be possible with minimal changes in the application. The framework should be designed to accommodate newer versions of XDM without much change in the code base.
- **Scalability**: The Framework should allow easy addition of new codecs. The design and architecture of the framework should be easy, simple and understandable to facilitate easy integration of codecs. The additional layers and frameworks should aid in development and not hinder the same.
- **Portability**: The framework should be easily portable to new platforms. This can be done easily if the code base is same. If the code base varies then this becomes a new challenge and maintainability takes a hit as there will be separate database for each platform and a small improvement, bug fix has to be absorbed into each database.
- **Maintainability**: The framework should be simple and easily maintainable. This also restricts the replication of source code in the framework. This is also a major challenge for the framework designer. The methods to build the framework should be easy.

CAF Design

Before we get into the design of the Codec Abstraction Framework, there are some assumptions. These assumptions are dependent on the codecs. They are:

- Codec symbols and structures will be unique to that particular codec and flavor and no two codecs/flavors will use the same symbols. If two codecs/flavors on the same platform use the same symbols, then there will be symbol clashes which the framework cannot handle.
- The flow of control across different codecs will be the same i.e. the order in which the calls are made will be very similar across codecs and platforms. (making control call only after creating the codec types)
- Same component manager (CE/CMR) is used by all codecs needed by any application.

With these assumptions, the design of Codec Abstraction Framework is arrived using bottom up approach. The codecs directly interact with CE/CMR. Hence the bottom most layer of the framework should provide an abstraction of the underlying component manager. This layer will make the appropriate calls depending on the component manager being used. Let us call this layer the **Component Framework Layer (CFL)**. This will have source files for different component manager.

The CFL layer will export the following API:

- `cfl_component_create()`
- `cfl_component_control()`
- `cfl_component_process()`
- `cfl_component_destroy()`

The CFL API will make calls to CE/CMR depending on the underlying framework used. As the scope of the paper is limited to XDM compliant systems, CFL will make calls to CE.

The next layer is the layer which interfaces with the application and translates the frameworks API to the respective codecs API. This can be called the **Codec Translation Layer (CTL)**. This layer should be common across platforms. In this layer each codec in the framework has its own translation file which does the translation and interfaces with CFL. The file will be called `ctl_<codec>.c`. This layer has to map the API depending on the platform, the XDM version and the flavor of the codec being used. The translation files will export a registry of functions similar to the CFL functions.

The appropriate translation function calls are made through a common function. This function is common only for a common group of codecs. The codecs are grouped into 8 groups according to the functionalities – Video Image Audio and Speech (VISA) Encoders and Decoders similar to XDM/XMI standard. These common file can be named `caf_ctl_videnc.c, caf_ctl_viddec.c` and so on for the other groups. The common API exported by the framework is implemented in these files. These files, based on the codec, flavor and platform translates the frameworks call to the respective codec translation call using the input parameter and the registry of functions exported as shown in Figure 3.

The structure of the common API implemented in the common translation files is:

```
caf_<VISA_grp><enc/dec>_<CE_call>
```

Video encoders create call will be: `caf_videnc_create ().`

Audio decoders process call will be: `caf_auddec_process ().`

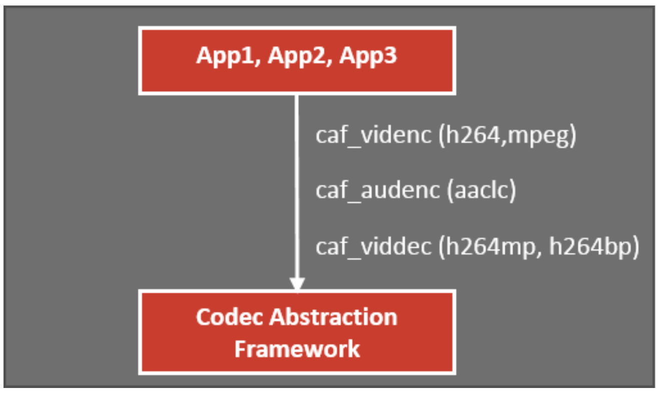

Figure 3 - Multiple applications accessing codecs through single interface (CAF)

The translation happening in CTL can be made simple by taking advantage that the XDM API. Hence the translation for a particular codec can happen in the same source file for all of its flavors as long as the XDM version remains the same. Similarly the translation can happen in the same source file for a particular codec present on different platforms. If the XDM version changes, the codec API will also change. This means that the translation for that particular codec has to happen independently and hence a new translation layer file (CTL) will be created.

In cases where the XDM version between flavors remain the same, but there are lot of other changes, then also the translation can be done independently creating a new translation layer file (CTL).

The scalability of the above design is explained below by considering different updates that can happen to the CTL:

- **New Codec Addition on a particular platform** – E.g. H.264 on DM6467 exists and MPEG4 on DM6467 to be integrated. Before the codec gets integrated, it should be made sure that there are no common symbols. Once this is ensured, a new CTL file is created for the new codec and integrated into the framework.
- **New flavor of an existing codec on a platform** – E.g. H.264BP exists on DM6467 and H.264MP needs to be integrated. Common symbols between these codecs should not exist. A new CTL file will be created only if the XDM version of the flavor is different from the one already existing on the framework. If the XDM versions do not differ, the previous CTL file should be extended to include this flavor. The code specific to the new flavor should be placed within if-else code delimiters. The flavor of the codec to be used (H.264BP or H.264MP) should be a configurable parameter.
- **New Codec on a different platform** – E.g. H.264 on DM6467 present, H.264 on DM6446 to be integrated. A new CTL file should be created if the XDM version varies. If not, the already existing file should be extended to DM6446. The platform specific code should be placed inside #if-#else code delimiters. If the difference between them is large then a new CTL file for that platform can be created for easier maintainability.
- **New platform addition** – E.g. DM365 platform needs to be added. The code has to be complied for the new platform using the proper tool chains. The platform specific code should be placed inside #if-#else delimiters. If the changes are too many then a new CTL file should be created.

Now that we have a Framework with a common API, let us take a look at the control flow. Assume the application calls `caf_viddec_create ()`. The application will call this function with the required codec and the parameters required to create the codec. Assume H.264 Decoder is the required codec. This function is implemented in the `caf_ctl_viddec.c`. This function has the registry of functions of all the codecs. Hence depending on the codec requested the respective codecs function will be called. These functions will be implemented in the `ctl_h264dec.c`. This file will perform codec related translations and will call `cfl_component_create()` which will call `VIDDEC_create()` call of Codec Engine and the codec is instantiated. The other calls are also handled in the same manner

Further Challenges

The assumptions made above need not necessarily hold true. They can be violated sometimes and handling these becomes a big challenge to the framework.

- **Same structures across different codecs** – Even though we assume that all the codecs will have unique symbols and structures, sometimes the different flavors of the same codec or two different codecs have the same structures. As there are individual translational for each codec, the codec header files will be present in their respective files only. They will not be present elsewhere. Hence two codecs having the same structure will not cause an issue, as these two codecs header files together in a single file will not be present in the framework.
- **Same structures across same codec** – The challenge is when two flavors of the codecs have the same structure. If the XDM version of these flavors is different, they will be having different translational files. They can be treated as two different codecs for which the solution

exists above. If the XDM version does not vary then a new translation file for that flavor has to be created. This is done to ensure simultaneous operation of these flavors. If the change is taken in under #if-#else delimiters, then only one flavor will be able to run. The changes cannot be taken under if-else delimiters as there will be more than one header file with same structure.

- **Call sequence across codecs** – The next challenge is the call sequence across codecs. Providing abstraction of call sequences will limit the feature set of codec that application can use. Hence the framework cannot provide abstraction of the call sequences of codecs and this should be handled in the application.

Ittiam Solutions

Ittiam has software solutions with realizations of CAF on multiple embedded processors and operating systems based on the above principles. These field-proven solutions have been delivered to several Tier-1 and Tier-2 customers in Industrial, Aerospace, Enterprise, Medical and Defense markets.

Conclusion

To build scalable and portable embedded applications there is a definite need for a Codec Abstraction Framework. There are several challenges while building the same like single point of access to all codecs, uniform API, transparency, easy migration, scalability, maintainability and portability. Assuming a little standardization of codec API this paper proposes a design that addresses several of these challenges and helps in building a framework which is scalable and portable.

Disclaimer