**Standard for RealTek DVD Recordable**

Date: 7/13/2011

**DVD-VENUS APPOverview**

RealTek specification on DVD Recordable Technology

# Specification for DVD-VENUS: Venus System Software Overview

**Note**

This document is subject to review.

Chen Ma
chen.ma@realcomtec.com

| Version | Date | Author |
|---|---|---|
| 0.0.1 | 6/22/2004 | Chen Ma |
| | | |

| Change List | Version | Description |
|---|---|---|

# Table of Contents

# 1 List of Figures

# 2 Introduction

## 2.1 Purpose

This document describes the Venus DVR (digital video recorder) design without go into the implementation details. It serves as the start point to understand the design scope and the terms that are used in other documents.

## 2.2 Scope

It covers the description of each software component inside the DVR system, including boot code, BSP (board support package), root application, shell, resource management, OSAL (OS abstraction layer), platform library, StreamClass architecture and basic description of the application.

## 2.3 Glossary

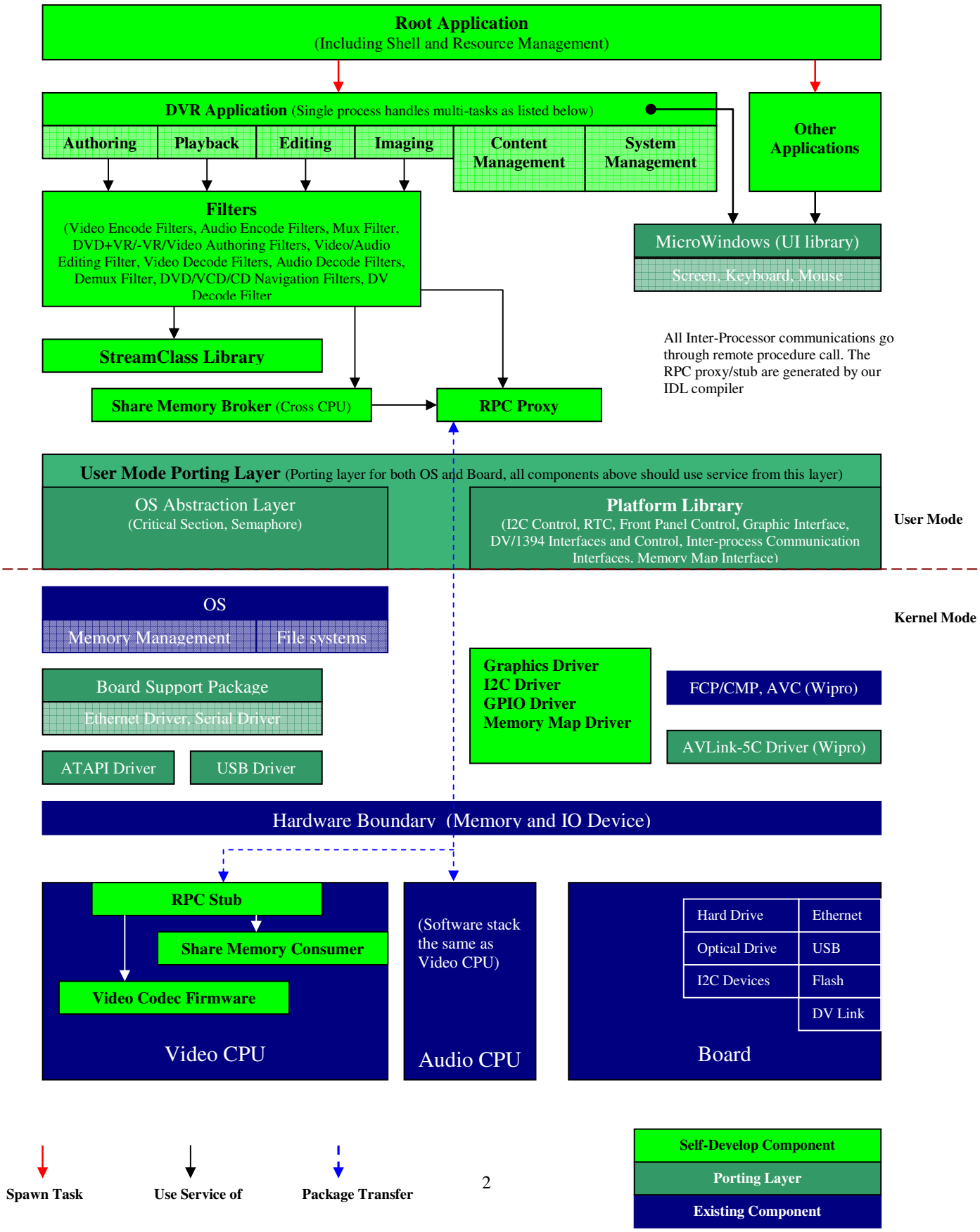
- StreamClass: A media streaming architecture bases on filter connection. See Stream Class Specification.
- Flow: A collection of Filters and Pins that can perform a media-streaming task. For example, DVD playback flow.
- OSAL: Operating System Abstraction Layer.

## 2.4 References

- Operating System Abstraction Layer (OSAL)
- Stream Class Specification
- System Boot Specification
- Platform Library Interfaces (PLI)

## 2.5 Overview of Document

## 3 Visualize Software Components in the System CPU



## 4 Root Application

Root application spawn from kernel initialization. The root task should keep alive always to serve as system resource manager and system watchdog thread. The root task will spawn debug shell on Video out when authorized connection is established. Root application is responsible to kill the applications that stop to response to watchdog call (a watchdog call is an inter-process communication that issue from watchdog application to the target application).

## 5 DVR Application

There are two applications running on the Venus system in the most basic configuration. One is root application as described above. Another one is the DVR application that spawn all the working threads to handle A/V media contents, including Audio/Video playback, authoring, editing, image viewing, etc. DVR application run in different process space than the root application. In case DVR application fail to response, either segmentation fault or drawn in infinite loop, the root application is responsible to kill the DVR application and try to recover from the failure and re-start the DVR application. DVR application communicate with Audio/Video codec firmware via remote procedure call (RPC) through shared memory channel. See RPC section below. DVR application also handle the user interface using UI library. System-wise control (board control and IO control) is provided by platform library, including I2C bus protocol, front panel control, etc.

## 6 UI Library

To enable high quality user experience, we provide window system base on MicroWindows implementation. It support window programming API conform to Microsoft windows programming API. It also provide partial of GDI interfaces that exists in Microsoft GDI implementation. The main goal of the UI library is to provide a programming friendly environment for windows programmer, so that they don't have to learn another proprietary window programming interface.
Under the window system, our library support DirectDraw-like 2D graphic interfaces. User can lock a surface (a frame) and linearly access any pixel via our hardware acceleration, using BitBlt with our hardware support, and flexible alpha blending functions that enhance the look and feel of drawing objects. See Microwindows API and Venus 2D Graphics API for detail.

## 7 Platform Library

## 8 Stream Class

Stream Class is designed to address the abstraction of multimedia components, the interaction between the multimedia components, and the interaction among application and multimedia components.

StreamClass simplifies media playback, format conversion, and capture tasks through a filter-connected architecture. At the same time, it provides access to the underlying stream control architecture for applications that require custom solutions. You can also create your own StreamClass components to support new formats or custom effects.

Examples of the applications you can write with StreamClass include DVD players, video editing applications, DV to MPEG transcoding, and digital video capture applications. See Stream Class Specification for detail.

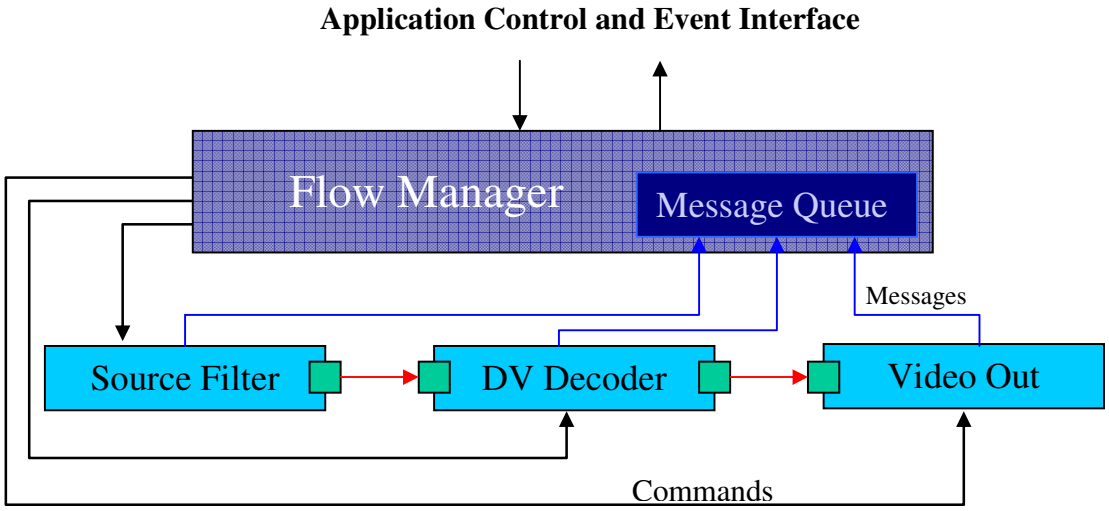



**Figure 2 The example DV decode flow**

Figure 2 demonstrates the DV decode flow under Stream Class architecture. The Flow Manager is a class that serves as a gateway between Filters and Application. Application first creates all the instance of the filters, in here, they are Source Filter, DV decode filter and Video out filter. Application then passes the reference of the filters to Flow Manager. Flow Manager behaves like a container; it collects the reference to all the filters. The application then manually connects the filters through Flow Manager connection utility interfaces. Filter connects to other filters using Pin. When application sends a **run** command to the flow manager, the flow manager will send the command to all the filters under its collection. When DV play to the end, the filter underlying will send the stream-end message to Flow Manager. Application can receive the message via the Flow Manager event interface.

# 9 Stream Class Filters

StreamClass (see Stream Class Specification) uses a modular architecture, where a filter does each stage of processing. StreamClass provides a set of standard filters for applications to use, and developers can write their own custom filters that extend the functionality of StreamClass. Each filter is connected to one or more other filters. The connection points are pre-implemented class, called *pins*. Filters use pins to move data from one filter the next. The arrows in the diagram (figure 2) show the direction in which the data travels. In StreamClass, a set of filters is called a *flow*.

Filters have three possible states: running, stopped, and paused. When a filter is running, it processes media data. When it is stopped, it stops processing data. The paused state is used to cue data before running; the section Data Transfer in the Flow in StreamClass Specification describes this concept in more detail. With very rare exceptions, state changes are coordinated throughout the entire Flow; all the filters in the flow switch states in unison. Thus, the entire Flow is also said to be running, stopped, or paused.

Filters can be grouped into several broad categories:

- A *source* filter introduces data into the flow. The data might come from a file, a camera, or anywhere else. Each source filter handles a different type of data source.
- A *transform* filter takes an input stream, processes the data, and creates an output stream. Encoders and decoders are examples of transform filters.
- *Renderer* filters sit at the end of the chain. They receive data and present it to the user. For example, a video renderer (Video out) draws video frames on the display; an audio renderer (Audio out) sends audio data to the audio DAC; and a file-writer filter writes data to a file.
- A *demux* filter splits an input stream into two or more outputs, typically parsing the input stream along the way. For example, the MPEG demux parses a MPEG program stream into separate video and audio streams.
- A *mux* filter takes multiple inputs and combines them into a single stream. For example, the MPEG program mux performs the inverse operation of the MPEG program demux. It takes audio and video streams and produces a MPEG program stream.

The distinctions between these categories are not absolute. For example, the DVD Navigator filter acts as both a source filter and a demux filter.

## 9.1 The Filters provided in our reference design

The Venus reference design kits will provide the following filters for customer to write their own application.

9.1.1 DVD Navigation Filter
DVD Navigator. It read from source, either DVD disk or other storage with UDF file system, and send out Audio Elementary Stream, Video ES, and subpicture stream. This filter also support DVD+RW playback.

9.1.2 VCD/CDDA/DVVD-VR/MP3.. Navigation Filter
A combo navigator provides navigation on multiple format.

9.1.3 MPEG I/II Video Decode Filter
It is a hardware accelerated Filter that decode MPEG I/II video elementary stream.

9.1.4 Audio Decode Filter
It is a hardware accelerated Filter that can decode MPEG I layer I/II/III audio elementary stream, AC3, AAC and other common audio formats.

9.1.5 SPU Decode Filter
A hardware accelerated subpicture decoder.

9.1.6 Video Out Filter
The video output control. User can use this filter to change hue, brightness and contrast. Other features may also apply.

9.1.7 Audio Out Filter
The audio output control. Use it to change volume and sampling rate.

9.1.8 Video In Filter
The video capture device. This device can be chosen to capture live video or capture from memory (for example, video out loop back and DV decode frames).

9.1.9 Audio In Filter
Choose from SPDIF or I2C.

9.1.10 MPEG I/II Video Encode Filter
MPEG encoder that supports 420 input or RGB input.

9.1.11 Audio Encode Filter
Two channel Audio Encoder that supports MPEG layer I,II, MP3, AC3 and AAC encode.

9.1.12 Mux Filter
The program stream muxer that generate DVD-Video, DVD+RW and DVD-VR compatible vob stream.

9.1.13 DVD Video/+RW Authoring Filter
Generate DVD Video/+RW compatible content with DVD Menu.

9.1.14 DVD –VR Authoring Filter
Generate DVD-VR compatible content.

9.1.15 Editing Filter
Non-linear Editing features. Simple cut and paste, cross fade transition effect.

9.1.16 Imaging Filter
View JPEG.

9.1.17 DV decode Filter
Decode DV25 bit stream.

## 10 OS Abstraction layer

In order to provide consistent software component on top of different operation systems, a thin layer consists of OS related utilities have been identified to provide the OS related services as the supplement to POSIX standard.

The layer define the following categories utilities

1. Interrupt Handling: Basic enables and disables interrupts.
2. Task Handling: Spawn, suspend and delete task.
3. Event Handling: Create, signal, check and delete events.
4. Semaphore Handling: Create, take, give and delete semaphores.

See Operating System Abstraction Layer document for details.

## 11 Inter-Processor Communication

The system CPU will control the other two CPUs through (RPC) Remote Procedure Call. All the commands and messages are also embedded inside RPC. The software in system CPU side creates and controls the software components sit on Video/Audio CPU. In the case mentioned, the system CPU software is the client of RPC, Video/Audio software components that receive the RPC is called RPC server.

In order to access the remote server portion of an application, special function calls, RPCs, are embedded within the client portion of the client/server application program. Because they are embedded, RPCs do not stand alone as a discreet middleware layer. When the client program is compiled, the compiler creates a local stub for the client portion and another stub for the server portion of the application. These stubs are invoked when the application requires a remote function and typically support synchronous calls between clients and servers.

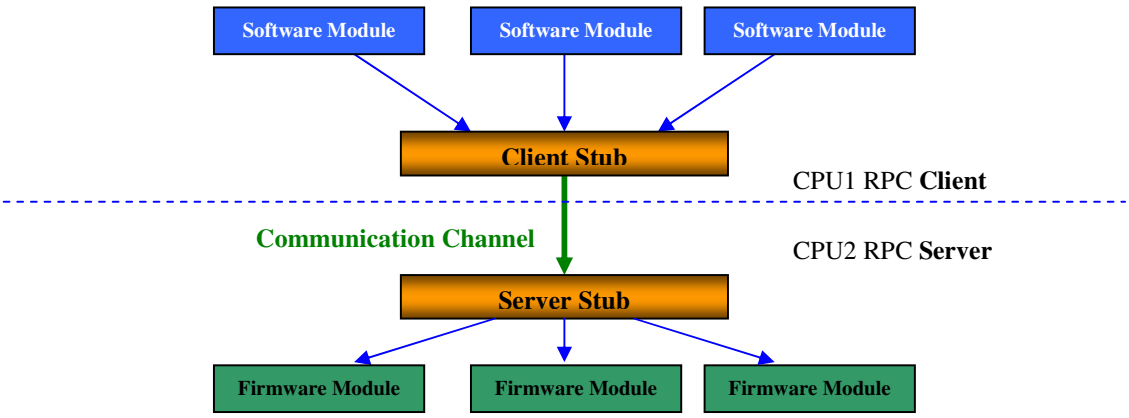



**Figure 3 Inter-Processor communication using RPC**

By using RPC, the complexity involved in the development of distributed processing is reduced by keeping the semantics of a remote call the same whether or not the client and server are collocated on the same system. However, RPC increases the involvement of an application developer with the complexity of the master-slave nature of the client/server mechanism.

RPC increases the flexibility of architecture by allowing a client component of an application to employ a function call to access a server on a remote system. RPC allows the remote component to be accessed without knowledge of the ‘network’/ ‘share memory’ address or any other lower-level information. Most RPC use a synchronous, request-reply (sometimes referred to as "call/wait") protocol that involves blocking of the client until the server fulfills its request. Asynchronous ("call/nowait") implementations are also available but needs extra care.

User defines an interface definition file to describe the function calls for communicating cross the processors. We will provide IDL compiler, **rpcgen**, to compile the interface definition file and generate all necessary files to handle inter-processor communication. See Venus Inter-Processor Communication for detail.

## 12 Communicate with Audio and Video Codec

This section describe the low level communication protocol between system side Filters and the Audio/Video firmware. It is not used directly by the system side applications, instead, the Filter developer will use this protocol to control the underlying AV codec. The communication between system CPU and Audio/Video CPU is achieved by remote procedure call. The server side (Audio/Video CPU) provide services that can be called by the system CPU. Refer to Inter-processor communication for detail.
The communication protocol on top of RPC is the rules about how system side Filters control the underlying A/V codec. Below is a simplified example describing the protocol to control Audio codec.

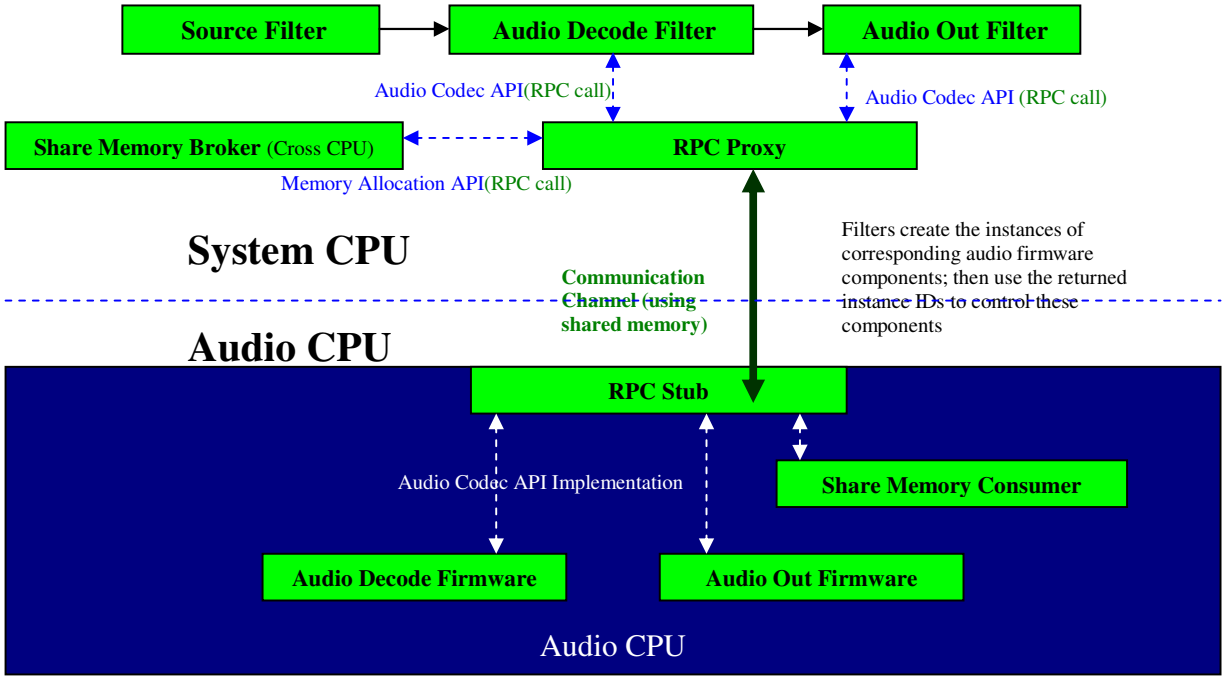



**Figure 4 The Audio Decoder Operation Scheme**

Figure 4 demonstrates an audio decoding flow under StreamClass architecture. The Audio Decoder filter and Audio Out filter on the system side create and communicate with their peer firmware components through remote procedure call (RPC).

### The sample audio decoding flow

Application first creates all the instance of the filters. In here, they are Source Filter (to retrieve the audio stream), Audio Decode filter and Audio Out filter. All filters are

created under the system CPU. Since the actual audio decoding and outputting are happen under Audio CPU, the audio decode and output filters are wrappers of their peer firmware components.
The construction of Audio Decode filter and Audio Out filter will send **Create** RPC call to the audio CPU. The **Create** function implemented in the audio CPU side will actually create the firmware instances and return the instance ID back to system CPU. All the filters then use the returned instance ID to control corresponding firmware component. For filters that control the underlying audio codec, each filter on the system CPU side will in general has one peer firmware component, so will have an unique instance ID to identify this firmware component.
The filters will use the returned instance ID to communicate with the corresponding firmware components. They can, for example, issue **Run**, **Stop**, **Pause** RPC calls with the instance ID as parameter to control the Audio codec.

For detailed specification, refer to Audio Application Interface and Video Application Interface.

## 13 Real-Time Design

User Interface Design

< Refer to User Interface material in the SRS and supplement with any design considerations not mentioned there. You should discuss the expected effectiveness of your design. >

## 14 Help System Design

## 15 Index

< generate here >