Application Note

Nexus

The Nexus Transcoder

STB_Nexus-AN400-R

5300 California Avenue • Irvine, CA 92617 • Phone: 949-926-5000 • Fax: 949-926-5203

April 25, 2013

Revision History

Revision	*Date*	*Change Description*
STB_Nexus-AN400-R	04/25/13	Initial release

Table of Contents

List of Figures

List of Tables

About This Document

Purpose and Audience

This document describes techniques needed to implement a ViCE2-based transcode system using Nexus. The intended audience is the Nexus user, who is assumed to have a general knowledge of Nexus, has run and studied the nexus/examples/encoder/* and rockford/unittests/nexus/encoder/* applications, and can therefore build and run simple Nexus transcode applications.

Acronyms and Abbreviations

In most cases, acronyms and abbreviations are defined on first use.

For a comprehensive list of acronyms and other terms used in Broadcom documents, go to: http://www.broadcom.com/press/glossary.php.

Document Conventions

The following conventions may be used in this document:

Convention	***Description***
Bold	User input and actions: for example, type **exit**, click **OK,** press **Alt+C**
`Monospace`	Code: `#include <iostream>` HTML: `<td rowspan = 3>` Command line commands and parameters: `wl [-l] <command>`
< >	Placeholders for *required* elements: enter your <username> or `wl <command>`
[ ]	Indicates *optional* command-line parameters: `wl [-l]` Indicates bit and byte ranges (inclusive): [0:3] or [7:0]

References

The references in this section may be used in conjunction with this document.

Note: Broadcom provides customer access to technical documentation and software through its Customer Support Portal (CSP) and Downloads and Support site (see Technical Support).

For Broadcom documents, replace the “xx” in the document number with the largest number available in the repository to ensure that you have the most current version of the document.

Document (or Item) Name	***Number***	***Source***
Broadcom Items		
[1] Nexus Software Architecture Guide	STB_Nexus-SWUM1xx-R	docSAFE
[2] Nexus Usage Manual	STB_Nexus-SWUM2xx-R	docSAFE
[3] Nexus Development Guide	STB_Nexus-SWUM3xx-R	docSAFE

Technical Support

Broadcom provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates through its customer support portal (https://support.broadcom.com). For a CSP account, contact your Sales or Engineering support representative.

In addition, Broadcom provides other product support through its Downloads and Support site (http://www.broadcom.com/support/).

Section 1: Introduction

This document describes techniques needed to implement a ViCE2-based transcode system using Nexus. The intended audience is the Nexus user, who is assumed to have a general knowledge of Nexus, has run and studied the nexus/examples/encoder/* and rockford/unittests/nexus/encoder/* applications, and can therefore build and run simple Nexus transcode applications.

Nexus Transcode Components

Nexus has the transcode interfaces given in Table 1.

Table 1: Nexus Transcode Interfaces

Capability	***Explanation***
NEXUS_VideoEncoderHandle	Handle to a video encoder (ViCE2 hardware)
NEXUS_AudioMuxOutputHandle	Handle to an audio mux output (Audio DSP)
NEXUS_AudioEncoderHandle	Handle to an audio encoder (Audio DSP). It must connect to an audio mux output interface to output the encoded audio buffer.
NEXUS_DisplayHandle	Handle to an encode display that can have a simple timing generator (STG HW). The interface between the encode display and ViCE2 hardware is called the “Virtual Displayable Point”
NEXUS_StreamMuxHandle	Handle to a stream mux that can multiplex the audio and video encoders’ output along with user-provided system data to an MPEG-2 TS stream (with XPT HW assistance)
NEXUS_FileMuxHandle	Handle to a file mux that can multiplex the audio and video encoders’ output to a container file, such as an MP4 or ASF file
NEXUS_SyncChannelHandle	Handle to an AV sync module that can synchronize the audio and video decoders up to the displayable point.
NEXUS_StcChannelHandle	Handle to a transport STC channel that tracks a timebase and can be configured in MOD300 or binary mode.

In order to explain how certain Nexus interfaces work, this manual may reference lower-level Magnum APIs or actual hardware blocks. Table 2 provides a mapping of these three layers.

Table 2: Mapping of Nexus, Magnum, and Hardware Blocks

Nexus Interface	***Magnum Module***	***Hardware Block***
VideoEncoder	VCE (Video Coding Engine)	ViCE2
Audio	APE (Audio Process Engine)	RAAGA
Display	VDC (Video Display Controller)	STG/BVN
StreamMux	Muxlib/XPT	XPT
FileMux	Muxlib	File I/O
StcChannel	XPT	XPT

This document is also related to the following transcoder source topics:

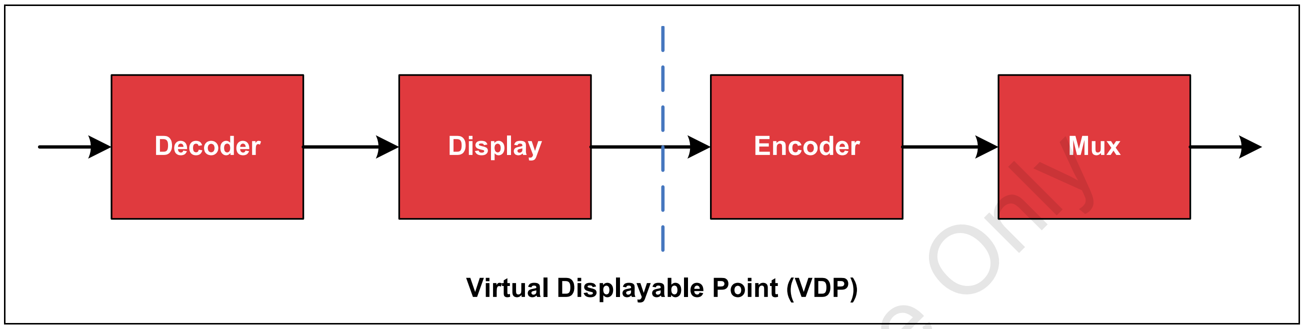
Table 3: Transcoder Source Topics

Topic	***Details***
NEXUS_VideoDecoder	Decode MPEG-2/H.264/MPEG-4, Part2/VC1/MVC/SVC/VP8/Sorenson/AVS video using video decoder HW. See Reference [2] on page 8 for documentation.
NEXUS_AudioDecoder	Decode various audio formats using audio DSP. See Reference [2] on page 8 for documentation.
NEXUS_HdmiInput	Some SoCs have HDMI input with both uncompressed video and audio source for transcoder. See Reference [2] on page 8 for documentation.
NEXUS_ImageInput	USB Camera YUV input may be fed to transcoder via the NEXUS image input interface. See Reference [2] on page 8.

Transcoder Overview

The ViCE2-based transcoder system is partitioned into source decoder/display side and encoder/multiplexer (mux) side with the so-called Virtual Displayable Point (VDP) to mimic the regular decoder box's display output in terms of an AV frame synchronization.

Figure 1: Transcoder System Partition



This allows the transcoder system to reuse the most of the regular decoder system. This partition also simplifies the encoder design, since the source discontinuity won't propagate to the encoder but is filtered by the decoder-side TSM scheme as it is in the regular decoder box. The encoder can assume its immediate input audio and video frames are synchronized and continuous, as if they were being shown on a real display device.

Section 2: Features

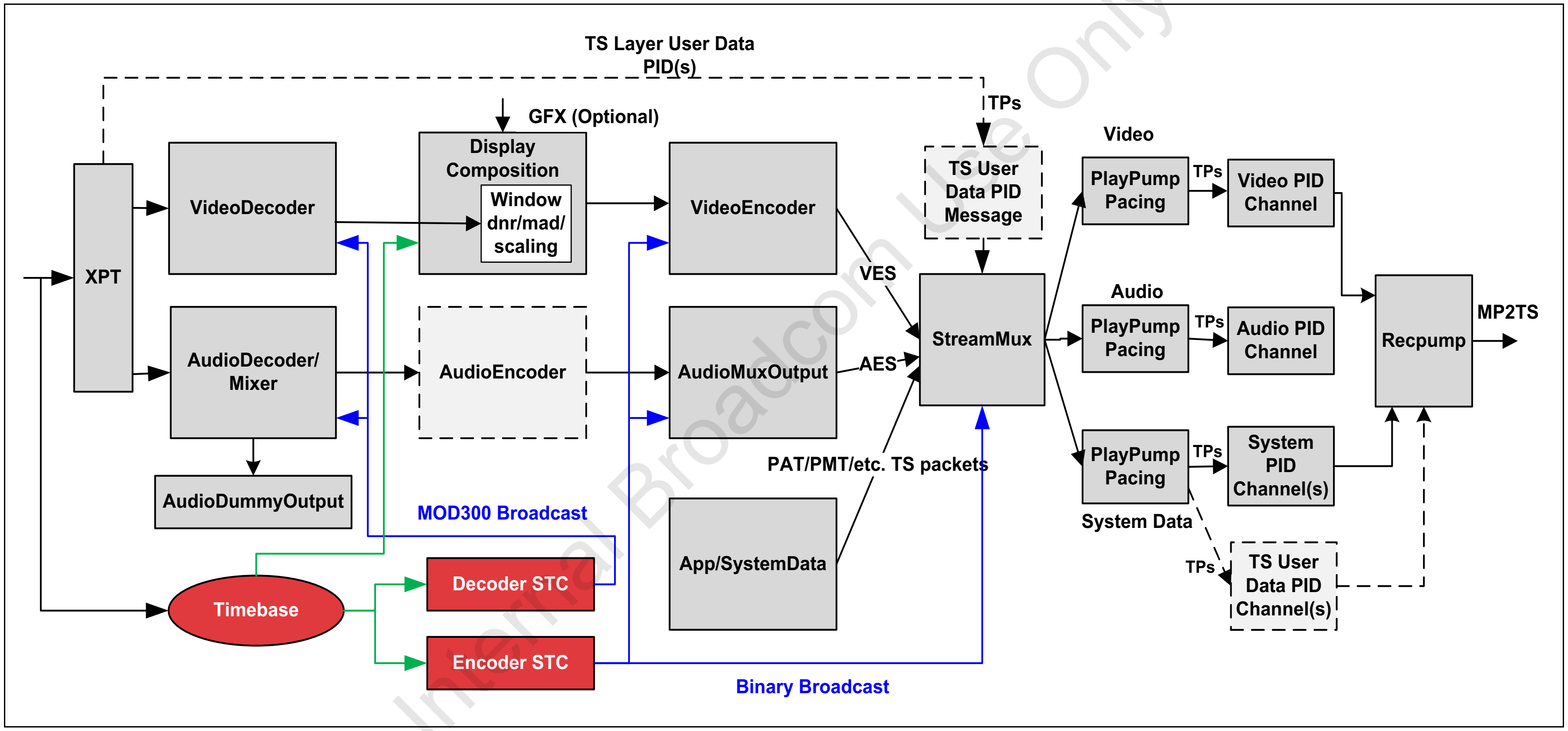
The NEXUS StreamMux module currently supports MPEG-2 TS output. The FileMux module currently supports MP4 ~~and ASF~~ container output. The audio/video codecs and formats supported for each mux container output may be different from chip to chip. Please consult with the relevant support team to get the complete feature list.

Note: Specific third-party codec or container support requires proper licenses in place to get the source code distribution.

TS StreamMux

Figure 2 shows both video and audio transcoder components.

Figure 2: TS StreamMux System Usage Diagram



The output goes through a TS stream mux via PlayPump, with PES pacing and TS packetization, then is multiplexed by RecPump as an MPEG-2 TS stream. The input of audio and video decoders could be a live tuner, XPT playback, etc., the same as the regular decoder box input. The audio encoder could be removed and audio decoder would be in pass-through mode for the transcoder.

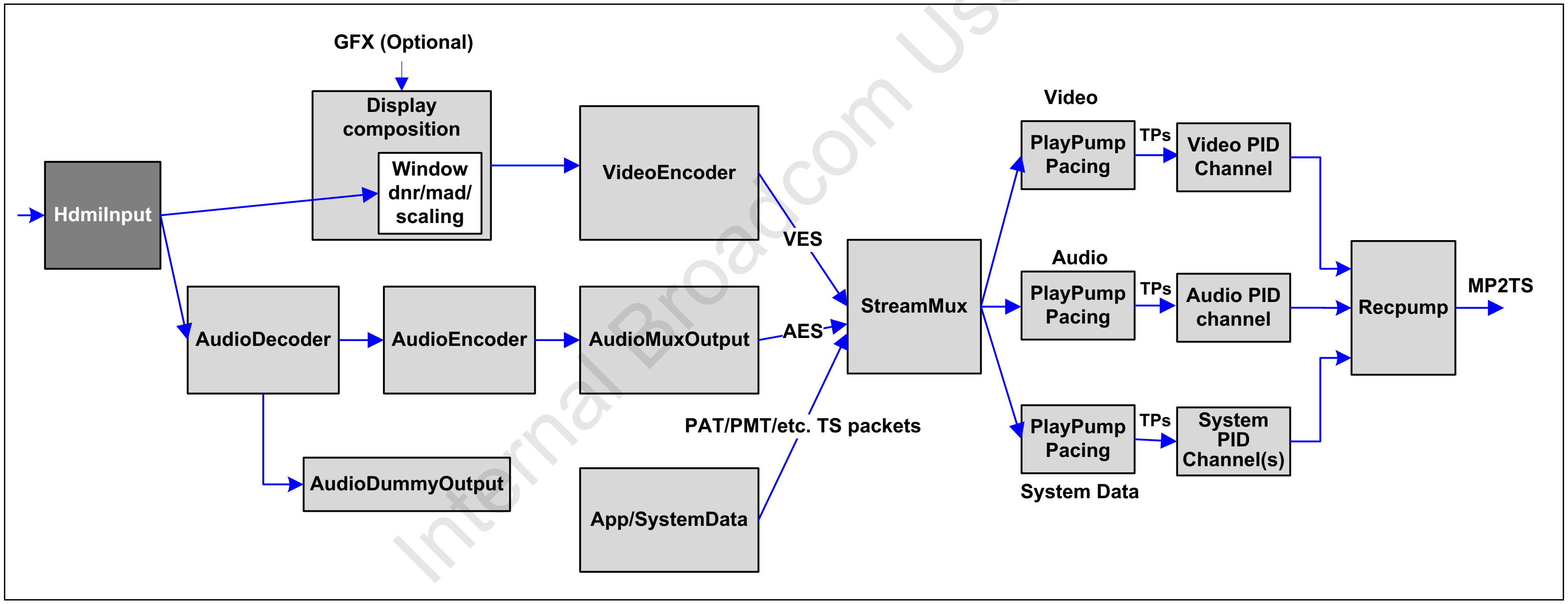
See the following code files:

- nexus/examples/encoder/transcode_playback_to_ts.c, transcode_qam_to_ts.c
- rockford/unittests/nexus/encoder/transcode_ts.c.

TS StreamMux with HDMI Input

Figure 3 shows HDMI input being transcoded into TS output.

Figure 3: HDMI Input Transcode



This diagram shows the HDMI input being transcoded into TS output. The HDMI input video is uncompressed and fed directly to the video window, while the audio could be compressed or uncompressed, and is fed to the compressed or PCM audio decoder.

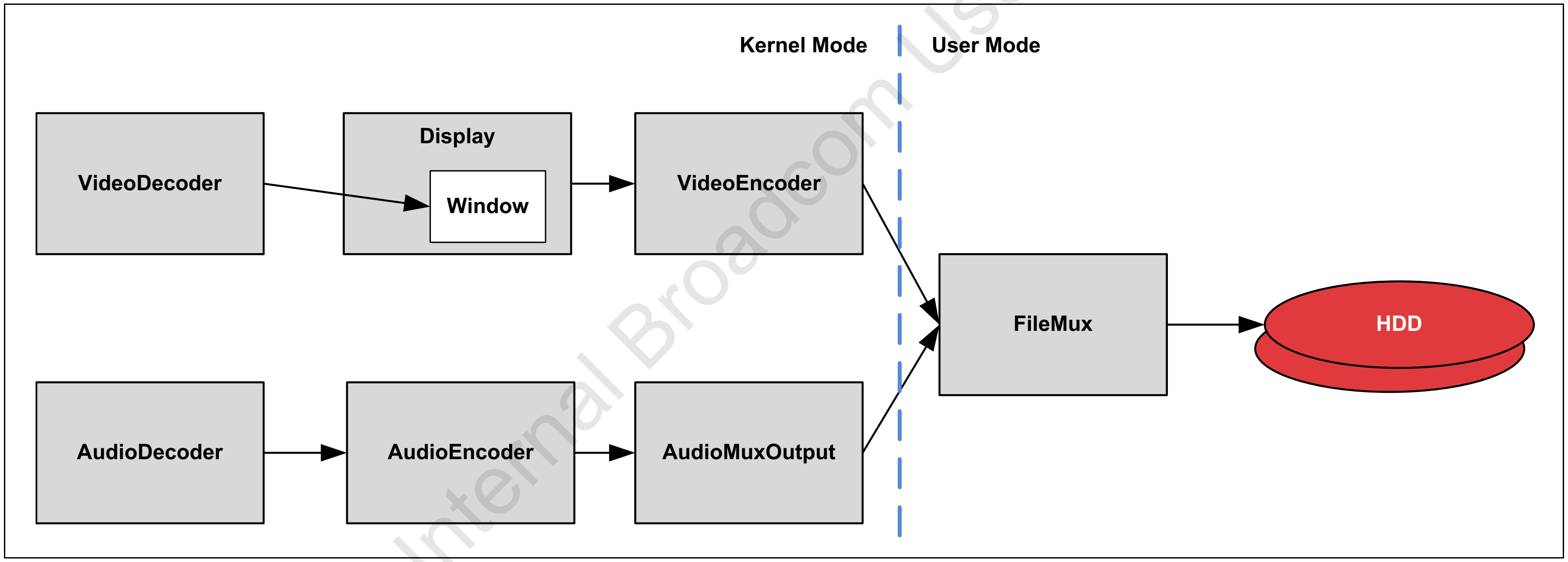
See the following code files for examples:

- nexus/examples/encoder/transcode_hdmi_to_ts.c
- rockford/unittests/nexus/encoder/transcode_ts.c.

File Mux

Figure 4 shows the transcoder output being sent to a file MUX module to be recorded as a media container file.

Figure 4: File Mux



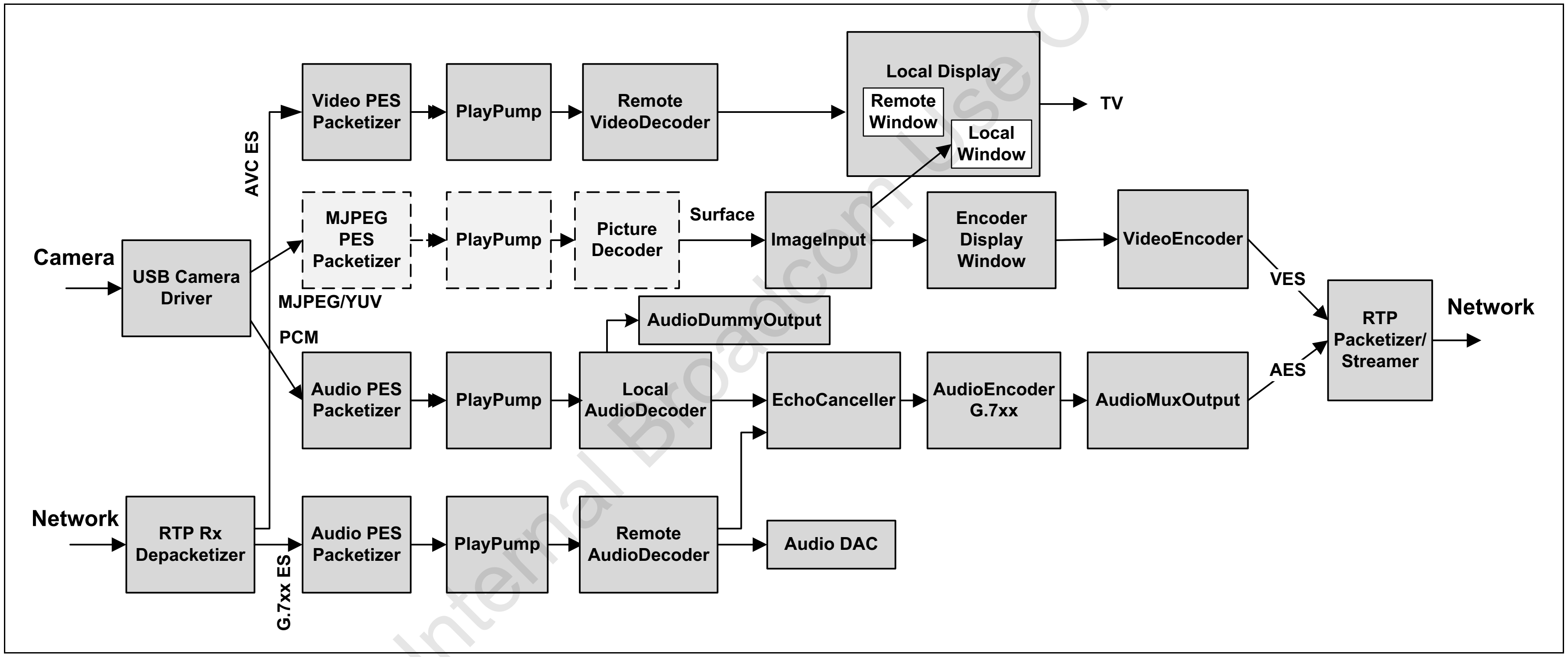
When NEXUS is built for kernel mode (`export NEXUS_MODE=proxy`), the NEXUS FileMux module is running in the user mode, since it operates on the file I/O. Note that ASF mux support is discontinued in the latest reference software releases.

See rockford/unittests/nexus/encoder/transcode_mp4.c.

ES Output

Applications can directly get the elementary streams (ES) out of VideoEncoder and AudioMuxOutput without going through the stream mux or file mux module. One usage may be for video conferencing; that application may packetize the H.264 video ES and G.7xx audio ES outputs into an RTP stream to send to the remote conferencing party.

Figure 5: Video Conference Usage



Another application usage of the ES output of the encoder is to construct an application-specific container output, e.g., fragmented MP4 output or a customer-proprietary ASF output stream.

For details on how to get the encoders' output buffers, see nexus/examples/encoder/encode_video.c and encode_audio_es.c.

Section 3: Application Notes

System

STC Channel

Each Real-Time mode transcoder's audio/video encoders and mux should share the same binary mode broadcast STC channel for AV sync and MUX pacing. The audio/video decoders also share a separate MOD300 mode STC channel for lipsync of the Virtual Displayable Point. To achieve end-to-end transcoder synchronization without slip, the decoder's mod300 STC channel and the encoder's binary STC channel should share the same timebase.

However, in Non-Real-Time mode (file-to-file transcode), the transcoder's audio and video decoders should use different STC channels and the two STC channels must be paired with some tolerant window to allow As Fast As Possible (AFAP) transcode and still keep the AV sync at the Virtual Displayable Point. Note in NRT mode, the audio and video STC channels are separately triggered to increment by the audio decoder and the video display with frame-intervals instead of the 27 MHz clock. The reason NRT mode audio and video decoders have different STC channels is because the audio and video frame intervals are typically different.

AV Sync

The transcoder's AV sync is partitioned into two parts: the decoder side and the encoder side.

The decoder side is responsible for keeping the AV sync of the Virtual Displayable Point and utilizes the regular decoder AV sync algorithm found in typical decoder boxes. In this regard, the NEXUS_SYNC_CHANNEL is reused to maintain the lipsync of the Virtual Displayable Point at the audio and video encoders input.

The encoder side just needs to balance the A2P delay or the arrival-to-the-presentation delay (from the Virtual Displayable Point at the input of the encoders to the downstream decoder's display) of the audio and video encoder buffer model. The encoder buffer model derives a constant A2P delay, given a set of encoding configurations to assure no overflow and underflow at the encoder buffer and downstream HRD decoder buffer.

Given that the XPT input audio/video frames carry the original PTS (coded or interpolated by the decoder TSM), and the original PTS is carried forward to the encoder output frames, it's possible to compute the transcoder AV sync error on the fly by correlating the audio and video transcoder output descriptors' original PTS and PTS:

AV sync error = (aPts – aOpts) – (vPts – vOpts);

If >0, video leads; otherwise, video lags; where aPts and vPts are the encoder-generated PTS per frame, aOpts and vOpts are the original PTS per frame.

See the nexus/examples/encoder/transcode_ ts.c for the transcoder AV sync setup and measurement.

Non-Real-Time Mode

Non-Real-Time (NRT) mode is a file-to-file transcode operation in which each component executes as fast as possible (AFAP) and is not tied to a fixed rate. It allows faster-than-real-time operation, and can go a lot faster if each component's throughput capacity is under-utilized in real-time mode (for example, an SD decode → SD transcode). Obviously, NRT mode could be slower than real-time operation as well, if the instructed transcode configuration exceeds the hardware throughput limitation or allocated system bandwidth.

Under NRT mode, the key issues are:

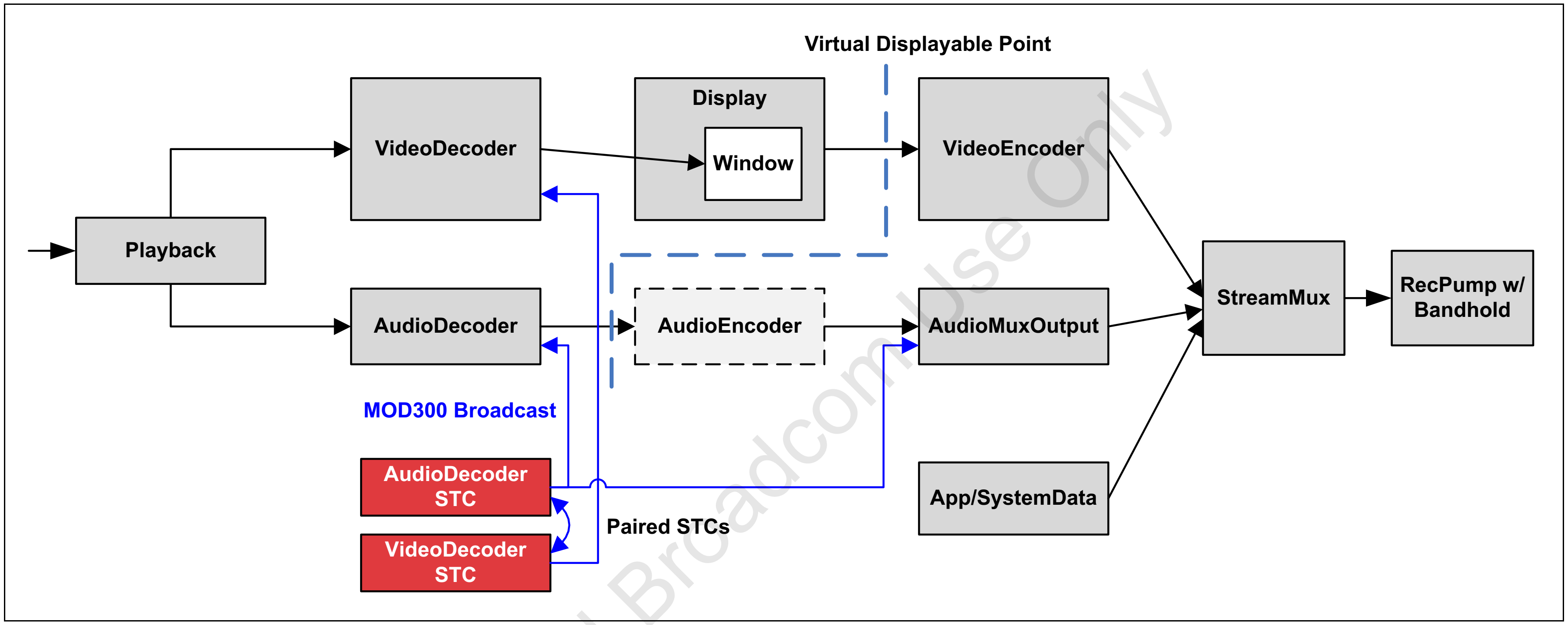
- How to control the transcoder data flow so as to not overrun to corrupt data and to avoid under-running to utilize the hardware throughput capability efficiently for the AFAP operation.
- How to synchronize the audio and video transcoder pipelines so that one is not to far ahead of another, resulting in significant buffering for AV sync and a long transition of AV sync loss when the source file hits discontinuity.

Broadcom's hardware/firmware/software architecture ensures the flow control of the NRT mode transcoder for the first item. The application just needs to instruct the relevant end-to-end NEXUS modules to operate in NRT mode via the API.

To properly operate in NRT mode, all the relevant NEXUS modules need to be put into NRT mode end-to-end, including the decoders, display, encoder, and MUX modules. The source playback module should also give a NULL STC channel, as no trick mode is needed at playback for NRT mode. The source file playback is inherently able to stall the file input when the downstream decoders stall to avoid overflowing the decoder buffer. The RecPump after the stream mux should enable the band-hold feature to be able to stall the upstream MUX module to prevent the record buffer being overflown. This means that the NRT mode file-to-file transcoder is under end-to-end flow control.

This is shown in Figure 6.

Figure 6: NRT Mode STC Usage



To properly synchronize the NRT mode audio and video decoders at the virtual displayable point, two separate STC channels need to be assigned to the audio and video decoders, and the two STC channels need to be paired with certain AV windows via the NEXUS_SYNC_CHANNEL API to prevent the audio and video racing ahead of each other by too much (constrained by AV_WINDOW). The NRT mode audio and video decoder STCs are controlled by audio and video timestamp management (TSM) agents to advance frame-by-frame to be as fast as possible.

The AudioMuxOutput should use the same STC channel as the audio decoder in NRT mode, which is different from the RT mode transcoder.

See examples in /nexus/examples/encoder/transcode_playback_to_ts_afap.c and /rockford/unittests/nexus/encoder/transcode_ts.c.

System Data

PCR packets are generated and inserted by the system data channel of the TS StreamMux, internally complying with strict timing. The user just needs to specify the NEXUS_StreamMuxPcrPid information in the NEXUS_StreamMuxStartSettings.

The application may add extra system data to the TS StreamMux via the NEXUS_StreamMux_AddSystemDataBuffer API, for example to insert PAT/PMT packets.

For the MPEG-2 TS StreamMux, the non-PCR system data has to be complete TS packets. The application is responsible for the compliance of the TS packets. For example, the CRC at the end of PAT/PMT TS packets need to be correct, the continuity counter in the TS packet headers needs to be incremented properly for each system data PID, etc.

See /nexus/examples/encoder/transcode_playback_to_ts.c and /rockford/unittests/nexus/encoder/transcode.c.

StreamMux Shutdown

To support clean shutdown of the stream mux, there is a “finish” API for the StreamMux module. The application can hook up a callback to wait for the clean finish of the stream mux before stopping it.

Also note that the actual TS stream mux module owns and opens the audio and video PID channels when started, so the stream mux stop function closes the audio and video PID channels. That implies the downstream RecPump or record must remove the audio and video PID channels from the stream mux output before the application stops the stream mux during shutdown.

See /nexus/examples/encoder/transcode_playback_to_ts.c etc.

FileMux Shutdown

To support the clean shutdown of the file mux (especially MP4 file mux), there is a “finish” API for the FileMux module. The application should hook up a callback to wait for the clean finish of the file mux before stopping it. This is especially important for the MP4 file mux, since it needs to finalize the MP4 file (writing the “moov” box) to make the container valid and playable.

The MP4 file format has the option of enabling progressive download support, which requires the moov box (file offsets, metadata, etc.) in front of the mdat box (actual ES data). However, the moov box cannot be generated until the whole MP4 “mdat” box is available, i.e., at the end of the transcode. This means that the progressive downloadable MP4 file mux finish would take a long time to generate a moov and copy mdat into an output file, especially for a long file.

One way to speed up the MP4 file mux finish is to disable the progressive download feature (the MP4 finishes instantly since the mdat box can be written to the output file while transcoding operates without waiting until the stop time and the moov box can be written at the end of file. The other way to speed up the MP4 finish time is to increase the relocationBuffer size in the NEXUS_FileMuxCreateSettings.mp4 structure.

See /rockford/unittests/nexus/encoder/transcode_mp4.c.

Video

A video transcoder path consists of a video decoder, a video encoder display, a video window that connects with the video decoder input, and a video encoder. The video encoder display can optionally take in a graphics input in addition to the video input, similar to the regular display.

Static Settings

The video encoder static settings are in NEXUS_VideoEncoderStartSettings:

```
typedef struct NEXUS_VideoEncoderStartSettings
{
    NEXUS_DisplayHandle input;
    bool interlaced;
    bool nonRealTime;
    bool lowDelayPipeline;
    bool encodeUserData;
    NEXUS_VideoCodec codec;
    NEXUS_VideoCodecProfile profile;
    NEXUS_VideoCodecLevel level;
    NEXUS_VideoEncoderBounds bounds; /* Encoder Bounds - If the bounds are known and specified,
       encoder latency can be improved. These bounds are for a single encode session.  E.g. if
       the output frame rate is known to to be fixed at 30fps, then the output frame rate min/
       max can be set to 30. */
    unsigned rateBufferDelay;       /* in ms.  Higher values indicate better quality but more
       latency.  0 indicates use encoder's defaults */
    NEXUS_StcChannelHandle stcChannel;
} NEXUS_VideoEncoderStartSettings;
```

Dynamic Settings

These are the encoder settings that are changeable on-the-fly in the NEXUS_VideoEncoderSetSettings structure after the encoder is started:

```
typedef struct NEXUS_VideoEncoderSettings
{
    unsigned bitrateMax;      /* units of bits per second */
    unsigned bitrateTarget;   /* default 0: CBR and target=max; non-zero: VBR target bitrate */
    bool variableFrameRate;   /* default false since not every decoder can handle variable
                                 framerate streams */
    bool enableFieldPairing;  /* to enable PicAFF with repeat cadence detection, to improve
                                 bit efficiency */

    NEXUS_VideoFrameRate    frameRate;
    NEXUS_VideoEncoderStreamStructure streamStructure; /* GOP structure */
    unsigned encoderDelay;     /* encoder delay, should be within NEXUS_VideoEncoderDelayRange.min
                                ... NEXUS_VideoEncoderDelayRange.max range */
} NEXUS_VideoEncoderSettings;
```

Note that the transcoder does support dynamic resolution, but the encode resolution is set as a part of the display format setting of the encoder display, instead of video encoder setting.

Correct Encode Table

The Correct Encode Table is concerned about the display aspect of an encoder/transcoder system. Given the input source format and encode format, it addresses the behaviors of the Display Manager, display, and video encode modules to help the encoded stream achieve correct display on the target decoder system.

The general recommendation is to set the encoder display rate to 60/59.94 Hz for ATSC-based output formats and to 50 Hz for PAL-based output formats.

If one requires frame rate pass-through transcoding, for example, the input is a 24/23.97p movie and the encode frame rate is 24/23.97p as well, so the encoder display rate can be set to 24/23.97p. This may help the NRT mode transcoder to run faster, compared to always setting display rate of 60 Hz, for example.

Low Delay Usage

Video conferencing requires low latency end-to-end, from the camera input to the remote decoder display. It requires careful system level design to achieve. This section only touches the video encoder configurations available for low-delay usage.

The Broadcom video encoder supports the following fine control of the low-delay configuration. To enable low delay configuration for the video encoder, for example, use the following code:

```
/* Avoid B-frame in GOP structure */
 videoEncoderConfig.streamStructure.framesB = 0;

/* disable Inverse Telecine Field Pairing to reduce delay */
 pVideoEncoderConfig->enableFieldPairing = false;

 /* 0 to means default 750ms rate buffer delay; change it to lower encode delay
     in ms at cost of quality */
 pVideoEncoderStartConfig->rateBufferDelay = 50; /* try 50 ms for example */

 /* Higher input minimum framerate for lower delay!
  * Note: lower minimum framerate means longer encode delay */
 pVideoEncoderStartConfig->bounds.inputFrameRate.min = NEXUS_VideoFrameRate_e59_94;

 /* higher minimum output frame rate for lower delay! */
 pVideoEncoderStartConfig->bounds.outputFrameRate.min = NEXUS_VideoFrameRate_e29_97;
 pVideoEncoderStartConfig->bounds.outputFrameRate.max = NEXUS_VideoFrameRate_e30;

 /* lower max resolution for lower encode delay */
 pVideoEncoderStartConfig->bounds.inputDimension.max.width = 720;
 pVideoEncoderStartConfig->bounds.inputDimension.max.height = 480;

 /*  Enable the video encoder pipeline low delay (MB row pacing);
     Note, lowDelayPipeline cannot be set true if NEXUS_VideoEncoderOpenSettings.type !=
     NEXUS_VideoEncoderType_eSingle; */
 pVideoEncoderStartConfig->lowDelayPipeline = true;
```

```
To reduce delay for video encoder display:

/* Full-size video may avoid 1-frame BVN capture delay */
NEXUS_VideoWindowSettings.forceCapture = false;
```

See rockford/unittests/nexus/encoder/transcode_ts.c for example of how to configure video encoder delay parameters.

Fast Channel Change

To support fast channel change for transcoder output live streaming, the video encoder offers the “eImmediate stop” mode that flushes/drops the internal buffered frames, instead of the “eNormal mode” that has to wait for the encoder internal buffer to be completely encoded:

```
/*
    Summary:
    Options for video encoder stop
 */
typedef enum NEXUS_VideoEncoderStopMode
{
    NEXUS_VideoEncoderStopMode_eImmediate,
    NEXUS_VideoEncoderStopMode_eAbort,
     …
} NEXUS_VideoEncoderStopMode;

typedef struct NEXUS_VideoEncoderStopSettings
{
    NEXUS_VideoEncoderStopMode mode;
} NEXUS_VideoEncoderStopSettings;
```

The video encoder also offers an adaptive low-delay mode for NEXUS_VideoEncoderStartSettings:

```
bool adaptiveLowDelayMode;
```

If set, the encoder will drop all incoming frames until the first decoded frame is seen. The first frame will be encoded with a low delay. Then delay will automatically ramp to the value specified in NEXUS_VideoEncoderSettings.encoderDelay, in the following frames.

The adaptive low-delay mode start helps a downstream live decoder show the video faster than the normal mode. Once the encoder delay is ramped to the specified delay (typically higher than low delay), the encoding quality can be retained as normal.

Camera Input

Camera input for video conferencing typically is done via the USB interface. The uncompressed YUV video or compressed MPEG stream decoded via the picture decoder becomes YUV video. Both need to go through the image input module to display, and then to the video encoder.

Other than the encoder low-delay configurations, there are optimization items in the camera driver to reduce the latency from the camera input to the NEXUS image input. The display path may also disable forced capture to reduce latency.

Audio

Audio Dummy Output

Note that in real-time transcode, the audio decoder needs to be driven with an actual output or dummy output, as shown above, besides the transcoder path. The BCM7425 supports four audio dummy outputs that can drive both the compressed and decompressed audio decoder. Attaching the dummy output here can save the actual audio output, like DAC, I^{2}S, etc. for the local display usage instead of being tied to the transcoder.

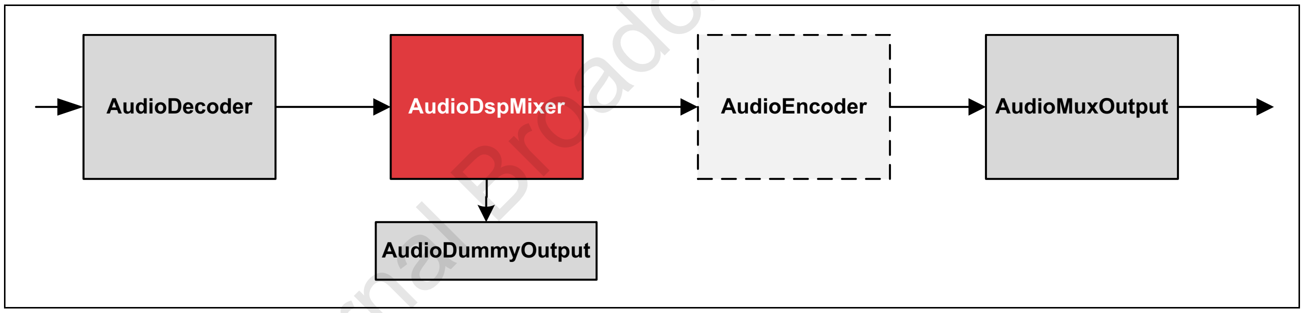
In non-real-time transcode usage, there is no need to attach the actual audio output or dummy output to the audio decoder.

See /rockford/unittests/nexus/encoder/transcode_ts.c.

Audio DSP Mixer

To avoid source discontinuity propagating to the audio mux output that might confuse the stream mux operation and result in dropped audio, it is recommended that you connect the audio DSP mixer with the audio transcoder path's decoder output, before the audio encoder (transcode mode) or audio mux output (pass-through mode). This is shown in Figure 7.

Figure 7: Audio DSP Mixer Usage



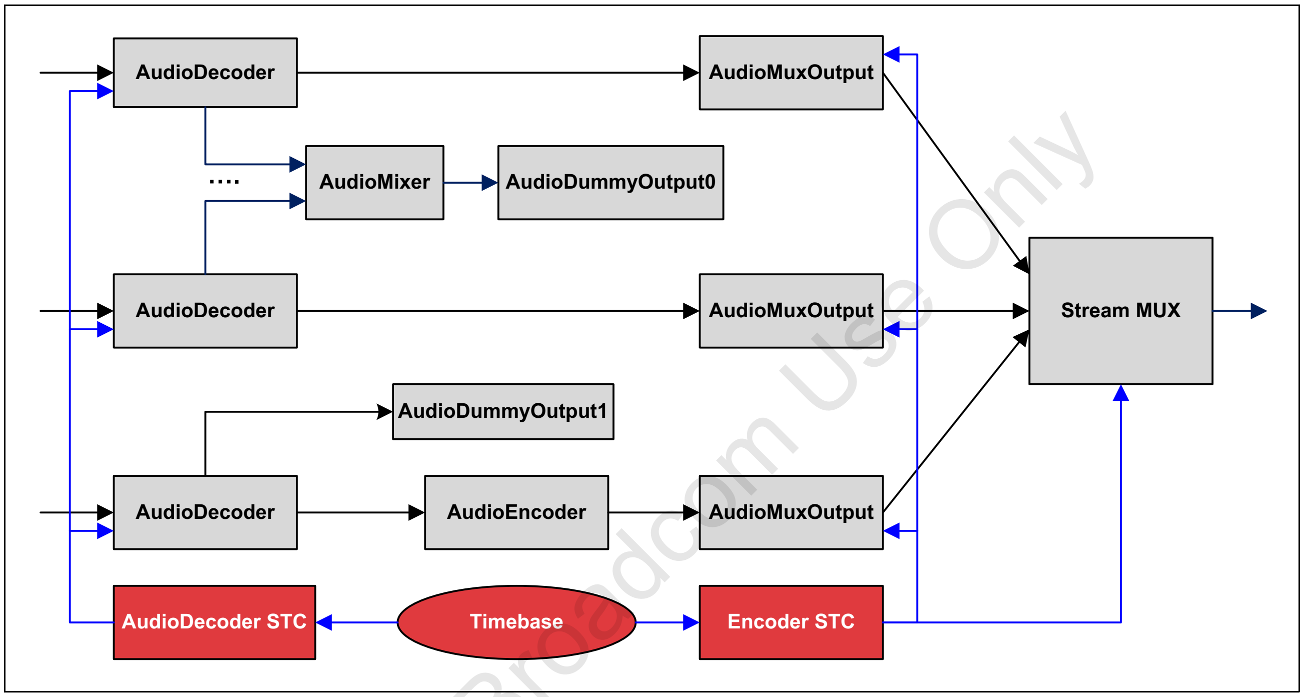
The audio DSP mixer output will be always continuous for the audio transcoder path, even when the source audio stream is discontinued or wrapped around. Note, however, that this is the DSP mixer that consumes the audio DSP task resource and may have an audio memory footprint that is scaled to support multiple audio transcodes.

See /rockford/unittests/nexus/encoder/transcode_ts.c.

Multiple Audios

The Nexus transcoder supports a feature that a source stream has a single video channel and multiple audio PIDs (up to 6x PIDs) to be compressed into a different stream, with video being transcoded (in terms of codec/ bitrate/resolution/etc.), while keeping the multiple audios passed through or one of those audios being transcoded with the rest being passed through. All the audio PIDs and video channel are based on the same timebase PCR. This works in both Real-Time mode and Non-Real-Time mode, and is shown in Figure 8.

Figure 8: Multiple Audio Channels Usage



The non-DSP audio mixer allows multiple audio decoder outputs (compressed or not) to be connected with a single dummy output. It is useful to have a single mixer/dummy output to drive multiple audio pass-through decodes to save the audio usage of actual outputs or dummy outputs to drive multiple audios.

Note that the pass-through audio decoder cannot share the same mixer with the decompression audio decoder. Also note that the AudioMuxOutput uses the encoder STC in RT mode, but uses the audio decoder STC in NRT mode. So Figure 8 shows the RT mode multiple audios' transcode usage. In NRT mode, the same AudioDecoder STC should be routed to the multiple AudioMuxOutputs as well.

See /nexus/examples/encoder/transcode_playback_to_ts_6xaudio.c.

Audio Speech Codecs

Conferencing audio typically uses G.711 or G.729 low-latency speech codecs. The USB camera input probably comes with uncompressed PCM audio.

Since a G.711 codec is simple enough to perform in the host software, the custom speech echo cancelation algorithm could be implemented in the application as well to avoid hardware latency.

It is preferable to have the more complex G.729 codec done in the audio DSP to offload CPU cycles. In this case, the echo cancelation algorithm is best performed in the audio DSP instead of in the host application as well, to avoid unnecessary latency between the host and the DSP data flow.

See /rockford/unittests/nexus/audio/echo_canceller.c.

User Data

Closed Caption and ES Layer User Data

The user data transcoding currently supports the following Closed Caption pass-through formats:

- CEA-608-E
- CEA-708-D
- A/53 D
- SCTE 21 2001
- SCTE 20 2001

The NEXUS API that turns on the Closed Caption user data transcoding is a boolean in the NEXUS_VideoEncoderStartSettings structure:

```
bool encodeUserData;
```

See /rockford/unittests/nexus/encoder/transcode_ts.c

The AFD and bar user data in the source stream will be decoded to assist the video processing pipeline to automatically crop the bar area, so that the AFD/Bar won’t need to be encoded in H.264 output. When bar areas are cropped, the video aspect ratio would be automatically adjusted and coded properly for H.264 output.

See /rockford/unittests/nexus/encoder/transcode_ts.c as an example.

Transport Layer User Data

The supported TS layer VBI user data specs are:

- ANSI/SCTE 127 2007
- ETSI EN 300 472 v1.3.1 (2003-05)
- ETSI EN 301 775 v1.1.1 (2000-11)

Subtitle, Teletext, and AMOL VBI user data carried in the MPEG-2 TS layer can be passed through the TS stream mux via the following API:

```
typedef struct NEXUS_StreamMuxUserDataPid {
    NEXUS_MessageHandle message; /* The userdata is expected to arrive as PES packets encapsulated
         in TS packets. Application is responsible for calling NEXUS_Message_Start prior to calling
         NEXUS_StreamMux_Start and call NEXUS_Message_Stop after mux session was completed */
} NEXUS_StreamMuxUserDataPid;
typedef struct NEXUS_StreamMuxStartSettings
{
…
    NEXUS_StreamMuxUserDataPid userdata[NEXUS_MAX_MUX_PIDS];
}
```

For TS layer VBI user data pass-through, in addition to the user data bitstreams, the application also probably needs to pass through the PSI data in the PMT table that describes the VBI user data services.

See /rockford/unittests/nexus/encoder/transcode_ts.c as an example.

Debug Tools

In addition to the regular BDBG debug modules message log, there are extra debug log tools in the relevant transcoder modules.

Video Encoder

The run-time API to get the video encoder status is NEXUS_VideoEncoder_GetStatus, which reports the currently encoded pictures count, received pictures count, error flags, etc.

The user could also dump the video encoder output buffers to files if compiling VCE PI with the following flags in user mode:

```
export BVCE_DUMP_OUTPUT_CDB=y
export BVCE_DUMP_OUTPUT_ITB=y
```

The encoder FW log can be dumped to file with the following compile flag set:

```
export BVCE_DUMP_ARC_DEBUG_LOG=y
```

Audio Transcoder

The audio transcoder can be debugged with the same method as the regular decoder box. The latest Nexus audio module also adds software debug support of the DSP log into a file:

```
#include “nexus_audio_dsp.h”
typedef struct NEXUS_AudioModuleSettings
{
     /* ….. */
    NEXUS_AudioDspDebugSettings dspDebugSettings;   /* DSP Debug settings */
} NEXUS_AudioModuleSettings;
```

To log the DSP debug into a file, the run-time environments on box are:

```
# export audio_debug_file=<file name>
# export audio_uart_file=<file name>
# export audio_core_file=<file name>
```

Mux

To log the mux’s input, the audio and video encoders’ per-frame-based descriptors (including PTS/DTS/ESCR/frame_size/flags, etc.), the compile time flags are:

```
export BMUXLIB_INPUT_DUMP_DESC=y
export BMUXLIB_INPUT_DUMP_FRAME_DESC=y
```

The video and audio encoders’ frame descriptors would be logged into separate BMUXLIB_INPUT_*.csv files for each input ES stream.

To log the TS mux's output transport descriptors that drive the XPT playback pacing of TS mux data, the compile-time flag is:

```
export BMUXLIB_DUMP_TRANSPORT_DESC=y
```

The transport pacing descriptors would be logged into separate BMUXLIB_TRANSPORT_DESC_xx.csv files for each TS mux playpump channels.

This works with the Nexus user mode build.

Display

The video encoder display can be debugged via a method similar to regular display module; for example, to detect and log BVN errors in the transcoder display path from console, set following compile flag:

```
export BVDC_SUPPORT_BVN_DEBUG=y
```

To log the display RULs to a binary file for offline debug of display errors, set the following compile flag for a user mode NEXUS build:

```
export BRDC_USE_CAPTURE_BUFFER=y
```

Then at the runtime box console, set the following:

```
# export rul_captures=<filename>
```

When you suspect the display is not delivering the correct picture information to the video encoder, you can turn on the debug message for the BVDC_DISP_STG module at run-time.

Limitations

Video Throughput

The BCM7425/BCM7435 video encoder hardware core supports up to 1080p30 or 1080i60 throughput in single-transcoder platform usage, even though the BVN transcoder path supports up to 1080p60 throughput, in which case the video encoder drops every second frame to encode 1080p30. For dual-transcoder platform usage, the combined throughput of the dual-transcoder channels is also restricted by this maximum throughput. That said, the dual transcoder platform supports up to dual-720p30 transcoders or 1440x1080i60 + 480p30 dual transcoders. However, the BCM7435 has two hardware cores, so it supports up to quad-720p30 transcodes.

Since the B-frame requires two reference pictures that consume a lot of bandwidth, the BCM7425 only supports the B-frame for up to half-HD (720p30) encode.

The BCM7445 video encoder hardware core throughput is doubled, so true B-frame support for HD (1080p30 and 1080i) will be available.

The BCM7445 also has two video encoder cores, each with doubled throughput, so the BCM7445 supports quad-1080p30 transcodes.

Video Decoder Resource

Note that a video transcoder expects to have a video decoder and corresponding resource set aside from the system. This means that for a BCM7425-based system that supports three independent HD video decoders, the dual-transcoder platform would only leave one video decoder for the local displays (i.e., no PIP decode).

This means that the BCM7425 dual transcodes cannot coexist with the main+PIP local display system.

The BCM7445 has more orthogonal resources available for the system, so it’s more flexible. Please see the relevant SoC system usage document for more details.

Number of Audio Channels

The audio DSP system does not have any hard limitation on the number of audio channels as far as the DSP MHz cap supports. However, typically the number of audio transcode channels matches the video path. In other words, a single-transcoder platform supports up to one audio decoder and one audio encoder. Dual-transcoder platforms support up to two audio decoders and two audio encoders.

If the rest of system is idle, the BCM7425 supports up to 6x audio transcoder pass-through or 5x audio transcoder pass-through channels plus one audio transcode channel simultaneously, when those are based on the same PCR.

Please see the relevant SoC system usage document for more details.

Connecting everything®

BROADCOM®

BROADCOM CORPORATION
5300 California Avenue
Irvine, CA 92617

Phone: 949-926-5000
Fax: 949-926-5203
E-mail: info@broadcom.com
Web: www.broadcom.com

STB_Nexus-AN400-R April 25, 2013