NTV2 HEVC Supplement

NTV2 Software Development Kit

AJA VIDEO SYSTEMS
180 Litton Drive • Grass Valley • CA • 95945

About this information

Revised August, 2015

Welcome to the AJA Video NTV2 HEVC Supplement Guide!

How this document will help you

This document provides an overview of HEVC support in the NTV2 SDK and describes how to get started developing custom applications for AJA video capture/playout/ hardware with HEVC capabilities.

This document assumes experience in developing video and audio processing applications and a good working knowledge of C and C++.

For more information about AJA's OEM product line:

- visit **http://www.aja.com/en/products/oem/**
- for registered OEMs, visit **https://sdksupport.aja.com/**

How to send your comments

Your feedback is important in helping us to provide accurate, high-quality information. If you have comments about this document, contact us in one of the following ways:

- If you're a registered AJA OEM Partner, log on to the AJA OEM SDK support site at **https://sdksupport.aja.com/** and create a ticket.

In any of this correspondence, please be sure to include the name of the document, the publication number (if any), the version of the SDK you're using, and, if applicable, the specific location (e.g., page number or section heading) of the text that you are commenting on.

When you send information to AJA, you grant AJA a nonexclusive right to use or distribute the information in any way that AJA believes appropriate without incurring any obligation to you.

Change Log

0.1	(2015Jun09) — First public release.
0.2	(2015Jul31) — Demo app updates, hevcmaintence added.
0.3	(2015Aug27) — Updated for AJA Corvid HEVC hardware.

Glossary Of Terms

DMA	Acronym for Direct Memory Access, a fast method for transferring data to-and-from a peripheral device into host memory (or another peripheral device) without directly involving the host's CPU(s).
driver	Low-level software that operates at the operating system "kernel" level to control a peripheral device.
field	The odd or even horizontal lines that comprise a single frame of interlaced video.
frame	A complete video image.
frame buffer	A contiguous chunk of RAM used to store one or more fields/frames of video. It can also store audio and ancillary data. Frame buffers exist both on the host computer and on AJA devices.
HEVC	*High Efficiency Video Coding*, H.265/HEVC is a video compression standard, a successor to H.264/MPEG-4 AVC (Advanced Video Coding).
host	The computer that is connected to the AJA device that is running the capture or playout application that's using AutoCirculate.
ISR	*Interrupt Service Routine*, usually a driver function that gets called by the host operating system "kernel" to handle an interrupt that may be relevant to the device being managed by the driver. Also *Interrupt Status Register*, which exists on a peripheral device that indicates the state of the device at the conclusion of an interrupt-driven data transfer with another device or host.
MB86M31	HEVC/H.265 Real-time encoder hardware, developed by Socionext, a new company established by the consolidation of the System LSI businesses of Fujitsu Limited and Panasonic Corporation. This real time 4K encoder is used in AJA HEVC capable hardware and is supported in this SDK.
SDI	*Serial Digital Interface*, a family of video interfaces standardized by SMPTE for transferring video and audio data between two machines.
VBI	*Vertical Blanking Interval*, which is the length of time between the end of the last line of a video field/frame and the start of the first line of the next frame/field. Also *Vertical Blanking Interrupt*, which signals the AJA device driver when the next field or frame should be handled by AutoCirculate.

Contents

1 — Overview

The NTV2 HEVC Software Development Kit (SDK) is a suite of classes and data types which allow end-users to access and control nearly any NTV2-compatible AJA device using the C++ programming language. It also provides a suite of classes and data types which allow end users to access and control the AJA Corvid HEVC device which is equipped with an MB86M31 real time HEVC/H.265 encoder from Socionext.

All of the user level code is platform-independent including the HEVC specific API's, HEVC demo application and HEVC Monitor, however kernel support for the AJA Corvid HEVC device has only been added to the Linux driver at this time. Specifically this driver has been designed to support AJA custom driver API's to communicate with the AJA Corvid HEVC video and audio systems and also the MB86M31 real time encoder on the AJA Corvid HEVC. The HEVC API's in this SDK we designed by AJA to be more closely aligned with the NTV2 SDK but still provide all the functionality of the MB86M31-Evaluation Board SDK from Socionext.

The purpose of the HEVC SDK is to enable third-parties to easily access and/or control the video, audio or ancillary data entering the AJA Corvid HEVC, and also provide access and control to the MB86M31 real time encoder. The SDK provides support at various layers (see figure 1).

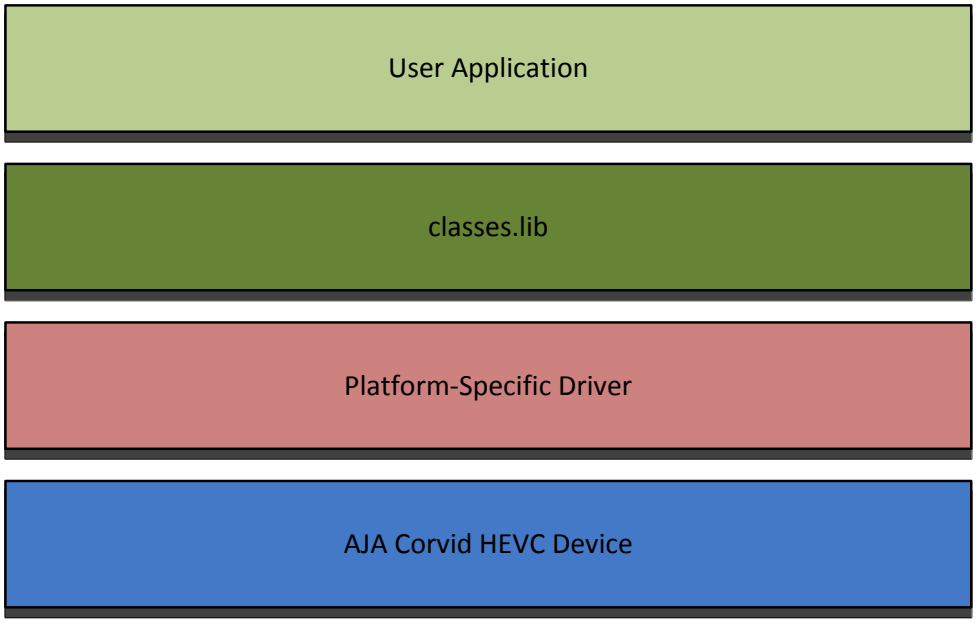


Figure 1 — The NTV2 architecture. The upper three layers comprise the SDK.

The Linux driver runs at the "kernel" level and handles low-level communication with the device. It is a required component of the SDK and provides the user-space library with the means

to communicate and control the device. HEVC support for taking directly to the MB86M31 real time encoder has been added to the AJA NTV2 Linux driver. This driver will be able to talk to both a standard AJA IO device and the AJA Corvid HEVC equipped with an MB86M31 real time encoder using custom API's defined in the user level side of the SDK.

The "classes" library is the principal user-space library that an application must link with in order to access and control AJA devices. In this class we also provide a set of API's to access and control the AJA Corvid HEVC equipped with an MB86M31 real time encoder. It implements a suite of C++ classes which an OEM application can instantiate and use to perform various operations on the AJA device and the MB86M31.

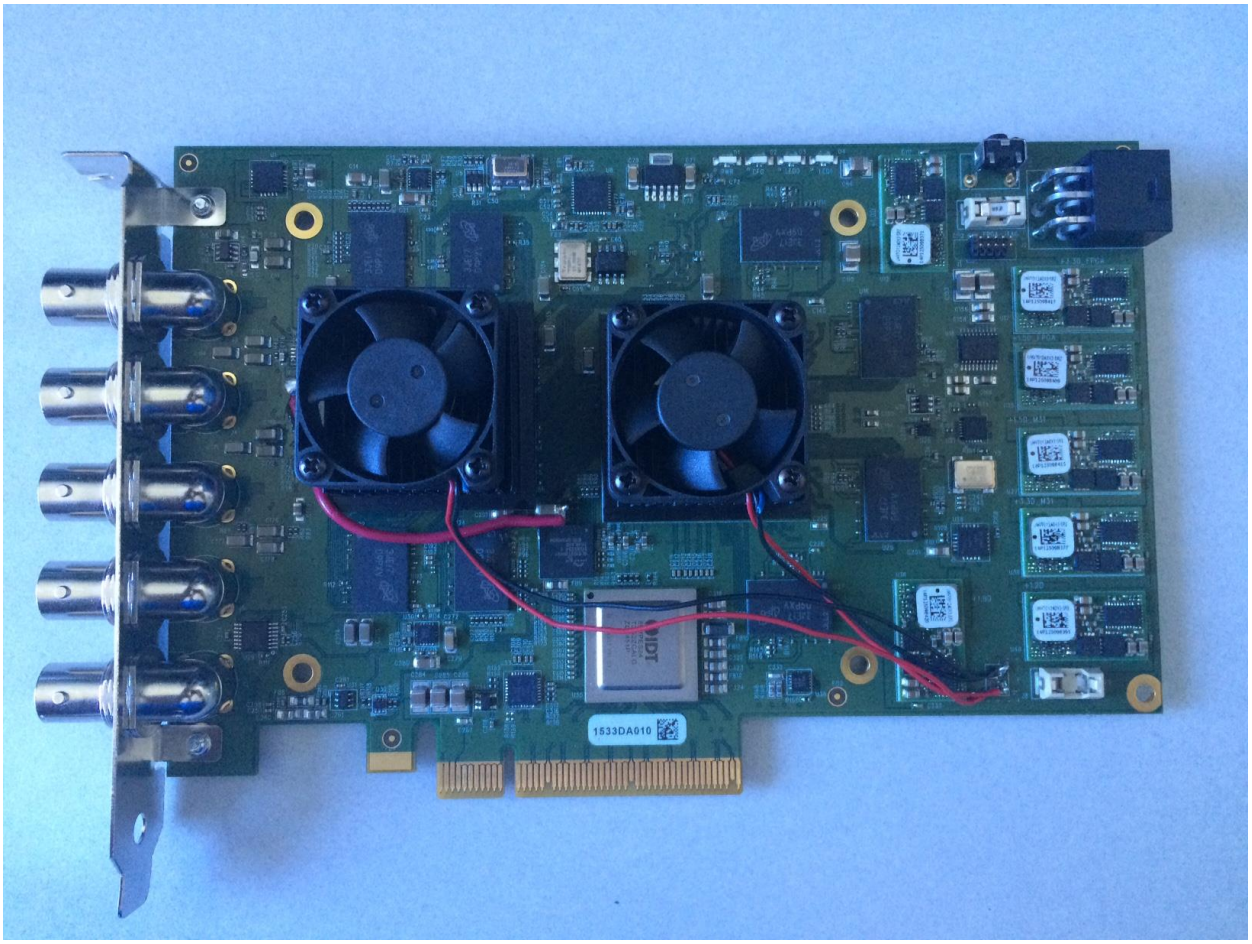
At the User Application layer we also provide demo applications, diagnostic tools, and common applications that can be used to maintain the device and allow users to update the firmware for the AJA Corvid HEVC and the MB86M31 real time encoder.

2 — Installing the Hardware

The AJA Corvid HEVC must be installed in appropriate PCI Express slots. The minimum requirements are a PCI Express Gen2 x 8 lane.

Figure 2 — AJA Corvid HEVC

The AJA Corvid HEVC is equipped with 4 SDI inputs and a reference/LTC Input. The first SDI input (SDI-1) is located at the top of the card and SDI-2 through SDI-4 follow. The very bottom connector is for reference or LTC input. The AJA Corvid HEVC is capable of capturing multiple channels of SD or HD video or one channel of UHD video. In multi channel record mode the input video can be different formats.



It is necessary to supply external power to the AJA Corvid HEVC using the ATX 6-Pin connector located on the back of the card (see figure 3).

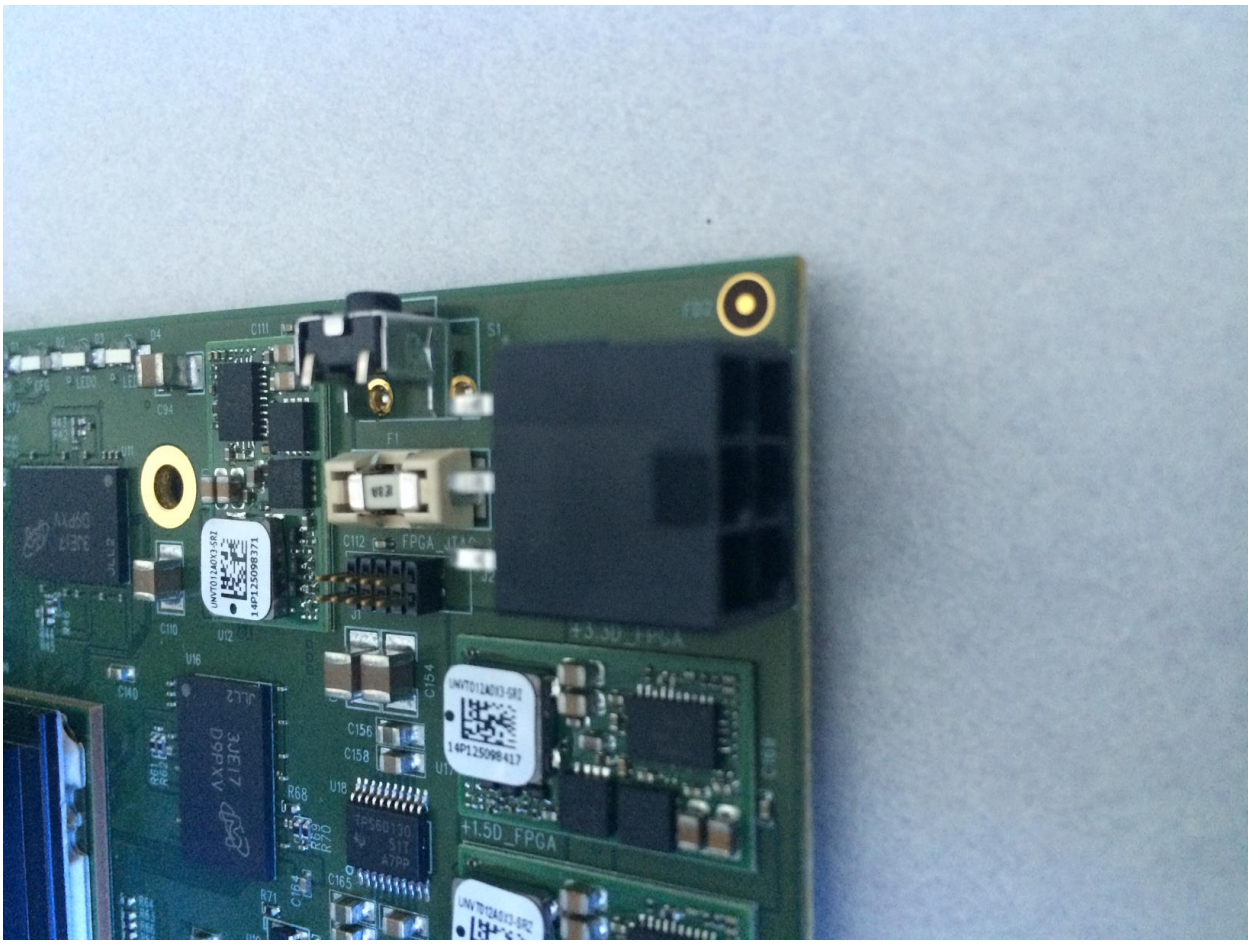


Figure 3 — Closeup of AJA Corvid HEVC ATX 6-Pin connector

3 — Installing the HEVC Linux SDK

Registered OEMs are given login credentials to access AJA's SDK support site, which can be securely accessed using any modern web browser at the following web address:

https://sdksupport.aja.com/

Building the Libraries on Linux

On Linux, the libraries and device drivers are not pre-built, and must be compiled and linked on the host they are expected to run on.

1) Be sure you have the requisite third-party software packages installed: **g++**, **libasound2**, **libsound2-dev**, and **libncurses5-dev**. Please refer to the documentation for your Linux distro for the correct utility for managing packages. Note that package managers often install dependent packages, so it may not be necessary to explicitly load all the packages in the aforementioned list. (The Knowledgebase article from which the SDK was downloaded is the authoritative source for this information.)

1) To build the Qt-based demo applications, you'll need to install Qt (the **qt5-dev-tools** package). If your Linux distro's package manager can't get it, AJA recommends downloading and building Qt directly from sources. For more information about Qt, please visit **http://qt.digia.com/**. Be sure the path to Qt's "bin" directory is in the current path.

2) Using a terminal, change to the '**ntv2projects**' directory in the SDK.

3) Set the environment variable **`AJA_HEVC=MB31`** to include the HEVC API and **`AJA_DEBUG=1`** if a debug build is required.

4) Enter the '**make**' command. This will build all libraries, tools, utilities, demo applications, and the device driver. The built executables will appear in the "../**bin**" directory.

5) Load the device driver: **`sudo ../bin/loadOEM2K`**

6) You may verify that the driver is installed by issuing the **`lsmod`** command, and looking for "XENA2" in the list.

Installing the HEVC firmware

There is an HEVC monitor application (qthevcmon) in the bin directory. Run this application and select the Debug tab to display the currently installed driver and firmware versions. The firmware versions will be labelled as either 'OK' or 'Iffy'. OK means that they are the same firmware version that was used for SDK testing. Iffy means that the firmware has not been tested with this version of the SDK and should be updated.

To update the HEVC firmware, navigate into the ntv2projects/commonapps/hevcmaintenance directory and execute the flash_all.sh script. The script will install all the codec firmware then

remind you to power cycle the system when it is complete. After the power cycle use the HEVC monitor application to confirm that the correct firmware is installed.

Breaking Down the Build for HEVC

On Linux, you can build just the parts you need for HEVC development.

1) Build and load the Linux Driver.
    a. **`cd ntv2projects/LinuxDriver`**
    b. **`make`**
    c. **`cd ../../bin`**
    d. **`sudo ./loadOEM2K`**

2) Build the "classes" library.
    a. **`cd ntv2projects/classes`**
    b. **`make`**
    c. The result will be at ../../lib

3) Build the "ntv2firmwareinstaller" program and update the NTV2 firmware.
    a. **`cd ntv2projects/ commonapps / ntv2firmwareinstaller`**
    b. **`make`**
    c. **`cd ../../../bin`**
    d. **`./ntv2firmwareinstaller –p "path to NTV2 bitfile"`**

4) Build the "hevcmaintenance" program and update the codec firmware.
    a. **`cd ntv2projects/commonapps/hevcmaintenance`**
    b. **`make`**
    c. **`./flash_system.sh (flashes system file)`**
    d. **`./flash_mcpu.sh (flash mcpu file)`**
    e. **`./flash_mode.sh (flash single and multi mode files)`**

5) Build the HEVC Monitor application using QTCreator. You will find the QT pro file at **`ntv2projects/commonapps/qthevcmon/qthevcmon.pro`**. The executable will be at the ntv2projects level in the bin directory.

6) Build and run the HEVC demo application:
    a. **`cd ntv2projects/demoapps/ntv2encodehevc`**
    b. **`make`**
    c. **`cd ../../../bin`**
    d. **`./ntv2encodehevc -?`**

4 — Installing the HEVC Windows SDK

Registered OEMs are given login credentials to access AJA's SDK support site, which can be securely accessed using any modern web browser at the following web address:

https://sdksupport.aja.com/

Installing the Windows SDK

Before installing the AJA NTV2 HEVC Windows SDK remove any earlier installed AJA NTV2 SDKs or HEVC SDKs using the Windows control panel programs and features. Older SDKs used a Windows msi installer for both the SDK and the driver. This is not strictly necessary but it will clean the older SDK from the system directories. The new SDK is delivered as a zip file with an included driver installer. This means that the SDK will no longer be installed to the Windows Program Files directory.

To use the new SDK decompress the zip file to a working directory. At the top level you will see the following files.

```
APIandSamples

DiagnosticsAndTools

Docs

Driver

ntv2driver-x.x.x.msi
```

'APIandSamples'

This directory contains the SDK source files described in the HEVC SDK contents section.

'DiagnosticsAndTools'

This directory contains prebuilt SDK tools.

'Docs'

This directory contains the SDK documentation.

'Driver'

This directory contains the NTV2 debug and release driver files for debugging.

'ntv2driver-x.x.x.msi'

This is the installer for the NTV2 driver.

Run the ntv2driver-x.x.x.msi to install the NTV2 driver that was tested with this SDK. The driver installer also installs the Microsoft Visual Studio runtime environment required to use the prebuilt SDK libraries and tools.

Building the Windows SDK

To build the SDK libraries and samples, open the Windows Visual Studio solution file (APIandSamples\ntv2projects\winworkspace\ntv2_vs12.sln. Choose the solution platform (win32 or x64) and the configuration (debug or release) for the build and select build all. This will build the libraries to the APIandSamples\lib directory and the binaries to the APIandSamples\bin directory.

Installing the HEVC firmware

There is an HEVC monitor application (qthevcmon.exe) in the DiagnosticsAndTools directory. Run this application and select the Debug tab to display the currently installed driver and firmware versions. The firmware versions will be labelled as either 'OK' or 'Iffy'. OK means that they are the same firmware version that was used for SDK testing. Iffy means that the firmware has not been tested with this version of the SDK and should be updated.

To update the HEVC firmware, extract the hevcmaintenance.zip file in the DiagnosticsAndTools directory. Navigate into the hevcmaintenance directory and double-click on the flash_all.bat file. The script will install all the codec firmware then remind you to power cycle the system when it is complete. After the power cycle use the HEVC monitor application to confirm that the correct firmware is installed.

Running an HEVC encode

The new NTV2 HEVC encode sample application is ntv2encodehevc. Connect a 720p50/59.94/60 or 1080p50/59.94/60 source to the SDI connector near the top of the CorvidHEVC board. Build and run the ntv2encodehevc application. The output should look similar to the following.

```
D:\Projects\ntv2sdk_hevc\bin\Win32>ntv2encodehevc.exe

Capture: M31_FILE_1920X1080_420_8_60p

           Capture  Capture

   Frames   Frames   Buffer

Processed  Dropped    Level

      721        0        1

Capture last frame number 840
```

```
Video file last frame number 840

D:\Projects\ntv2sdk_hevc\bin\Win32>
```

Type ctrl-c to terminate the application. There is more information about the ntv2encodehevc application in the NTV2EncodeHEVC Demo Application section.

While the ntv2encodehevc application is running you can monitor the encoding operation using the HEVC monitor application (qthevcmon.exe) in the DiagnosticsAndTools directory. See the HEVC Monitor section for more information.

5 — HEVC SDK Contents

This section explores the contents of the SDK, some directories are specific to the HEVC Linux SDK and some to the HEVC Windows SDK.

```
ajaapi
   ajastuff
   gpustuff
ajalibraries
   ajaanc
   ajacc
bin
docs
   gpu
lib
ntv2projects
   classes
   codecs
   commonapps (linux)
   demoapps
   democlasses
   fltk (linux)
   includes
   linuxapps (linux)
   linuxclasses (linux)
   linuxdriver (linux)
   winclasses (windows)
   winworkspace (windows)
```

‘ajaapi’

This directory contains the ‘**ajastuff**’ directory, which has several platform-independent utility classes for threading, locking, etc. It also contains the ‘**gpustuff**’ directory.

‘ajalibraries’

This directory contains the ‘**ajaanc**’ directory, which contains the AJA ancillary data library and the ‘**ajacc**’ directory, which contains the AJA closed-caption library.

‘bin’

This is the destination directory for all executable files and the dynamically-loaded libraries they need to run.

‘docs’

This directory is where all documentation can be found.

‘lib’

This is the destination directory for all static libraries.

‘ntv2projects’

This directory contains the NTV2 SDK, which has several sub-directories of interest:

'classes'	The principal NTV2 classes and header files.
'codecs'	The principal user level NTV2 HEVC classes and header files.
'commonapps'	The sources for the 'Cables' and 'Watcher' tools (Linux only).
'demoapps'	Source files, make files and project files that build the NTV2 demonstration applications, which utilize the "democlasses".
'democlasses'	Source files and header files for the NTV2 demonstration classes used by the demonstration applications.
'fltk'	A lightweight GUI library used to build the 'Cables' and 'Watcher' tools (Linux only).
'includes'	The generic NTV2 header files.
'linuxapps'	Source code and makefiles for Linux-specific utilities and test programs.
'linuxclasses'	Source code for Linux-specific classes.
'linuxdriver'	Source code for Linux driver.
'winclasses'	Source code for Windows-specific classes.
'winworkspace'	Windows Visual Studio workspaces.

6 — HEVC Impementation

Overview

This section explores the HEVC specific API's of the SDK. AJA has borrowed extensively from the MB86M31-Evaluation Board SDK, but rather than copy directly we have implemented our very own API's which are much closer aligned with how the NTV2 SDK works. The MB86M31 encoder implementation is an extension of the current NTV2 API's, with it's own classes to initialize, configure, control, and stream data to and from the encoder.

The HEVC Files

The SDK is comprised of many files so we felt it might be handy to point out files in the SDK that are specific to HEVC support. Here you can find definitions, enums, structures, classes, headers, and both user and kernel level source code. We've included everything so nothing is hidden.

```
ntv2linuxhevc_12_3.x.x
   ntv2projects
      classes
         ntv2driverinterface.cpp
         ntv2driverinterface.h
      codecs
         hevc
            m31
               ntv2m31.cpp
               ntv2m31.h
               ntv2m31cparam.cpp
               ntv2m31cparam.h
               ntv2m31ehparam.cpp
               ntv2m31ehparam.h
               ntv2m31vaparam.cpp
               ntv2m31vaparam.h
               ntv2m31vinparam.cpp
               ntv2m31vinparam.h
               ntv2m31viparam.cpp
               ntv2m31viparam.h
      commonapps
         hevcmaintenance
            config_common.bin
            dram_init_param.bin
            dram_test_all_ch.sh
            dram_test_ch_a.sh
            dram_test_ch_b.sh
            dram_test_ch_c.sh
            dram_test_ch_d.sh
            flash_mcpu.sh
            flash_mode.sh
            flash_system.sh
            m31_fw
               hevc_enc_fw_multi.bin
```

```
            hevc_enc_fw_single.bin
            mcpu_fw.bin
            system_fw.bin
         main.cpp
         Makefile

      qthevcmon
         controltab.cpp
         controltab.h
         debugtab.cpp
         debugtab.h
         main.cpp
         mainwindow.cpp
         mainwindow.h
         mainwindow.ui
         qthevcmon.pro
         streamtab.cpp
         streamtab.h
   demoapps
      ntv2encodehevc
         main.cpp
         Makefile
   democlasses
      ntv2encodehevc.cpp
      ntv2encodehevc.h


   includes
      ntv2linuxpublicinterface.h
      ntv2publicinterface.h
      ntv2m31enums.h
      ntv2m31publicinterface.h

   linuxclasses
      ntv2linuxdriverinterface.cpp
      ntv2linuxdriverinterface.h
   linuxdriver
      driverdbg.h
      hevcapi.c
      hevccommand.c
      hevccommand.h
      hevccommon.h
      hevcconstants.h
      hevcdriver.c
      hevcdriver.h
      hevcinterrupt.c
      hevcinterrupt.h
      hevcparams.c
      hevcparams.h
      hevcpublic.h
      hevcregister.c
      hevcregister.h
      hevcstream.c
      hevcstream.h
      registerio.c
      registerio.h
```

The Kernel

Five driver API's have been added to support HEVC.

```
bool HevcMessageGetDeviceInfo (HevcMessageInfo* pMessage);
bool HevcMessageSendCommand (HevcMessageCommand* pMessage);
bool HevcMessageVideoTransfer (HevcMessageTransfer* pMessage);
bool HevcMessageGetStatus (HevcMessageStatus* pMessage);
bool HevcMessageDebugInfo (HevcMessageDebug* pMessage);
```

In addition the above HEVC specific functions, the standard NTV2 ReadRegister and WriteRegister functions have been modified to recognize register reads and writes to the MB86M31 encoder. The PARM setup classes in ntv2projects/codecs/hevc/m31 make extensive use of Read/WriteRegister to read and write individual PARAM fields in the MB86M31 space. The interface is identical to Read/WriteRegister in the NTV2 space.

Initialize

In order to configure and use the MB86M31 encoder we must insure it is in the "init" state. One way to do this is to use the CNTV2m31 helper class. This class is defined in ntv2m31.cpp and is part of the "classes" lib. The code might look something like:

```
#include "ntv2m31.h"

AJAStatus InitHEVC(CNTV2Card* device)
{
	// Allocate our M31 helper class
	CNTV2m31 m31 = new CNTV2m31(device);

	HevcMainState   mainState;

	m31->GetMainState(&mainState);
	if (mainState != Hevc_MainState_Init)
	{
		if (!m31->Reset()) return AJA_STATUS_INITIALIZE;

		// After a reset we should be in the boot state so lets check this
		m31->GetMainState(&mainState);
		if (mainState != Hevc_MainState_Boot) return AJA_STATUS_INITIALIZE;

		// Now we can go to the init state
		if (!m31->ChangeMainState(Hevc_MainState_Init, Hevc_EncodeMode_Single))
	  return AJA_STATUS_INITIALIZE; }

		// Make sure we got there
		m31->GetMainState(&mainState);
		if (mainState != Hevc_MainState_Init) return AJA_STATUS_INITIALIZE;

		// This zeros out all of the param space in the M31 for every channel
		// It is necessary to do this the very first time you bring up
		// the device, otherwise when you begin param setup you will end up
		// with uninitialized fields in the sparse registers.  The M31 does
		// not like this.
		m31->ClearAllParams();

	}
	return AJA_STATUS_SUCCESS;
}
```

Params

The MB86M31 has a number of parameters that can be written too to setup and configure the encoder. The parameters have been broken up into a number of categories. Currently they are:

- CParams
- VIParams
- VINParams
- VAParams
- EHParams
-

AJA has provided a set of classes to manage each of these param categories which can be found in ntv2projects/codecs/hevc/m31. The M31 helper class also provides a wrapper that calls each of these categories to help setup the encoder for a specific preset. The AJA version of the HEVC presets are a table of enums found in ntv2projects/includes/ntv2m31enums.h. We included both "file" and "vif" presets. The HEVC Encode demo application uses these enums and helper class functions to setup the encoder to encode a specific format. Here is an example of how to setup the encoder for a given preset:

```
#include "ntv2m31.h"

AJAStatus LoadAndWritePreset(CNTV2Card* device, M31VideoPreset m31Preset)
{
	// Allocate our M31 helper class
	CNTV2m31 m31 = new CNTV2m31(device);

	HevcMainState	mainState;

	// We need to be in the init state to setup all of the encoder params
	m31->GetMainState(&mainState);
	if (mainState == Hevc_MainState_Init)
	{
		// This function will load up the default parameters for a preset into
		// structures inside of the helper class for ever param category. This
		// function is channel independent.
		if (!m31->LoadAllParams(m31Preset)) return AJA_STATUS_INITIALIZE;

		// This function will write out all the params in the local structures to
		// the MB86M31 to a specific channel (in this case channel 0).
		if (!m31->SetAllParams(M31_CH0)) return AJA_STATUS_INITIALIZE;

	}
	return AJA_STATUS_SUCCESS;
}
```

If you wanted to customize the setup and do something slightly different than what the preset provides you can do that by loading all default params using the closest preset then manipulate the channel structures directly before you write them out. I suggest you don't try this unless you have reviewed all of the parameters and have a good understanding of how to setup the encoder manually. There are many duplicate params spread across all of the param categories and it would be easy to miss changing some. Bit depth is a param used in several of the param

categories so you have to change all of them. Below is an example of how you might change the bitdepth manually:

```
if (mainState == Hevc_MainState_Init)
{
        if (!m31->LoadAllParams(m31Preset)) return AJA_STATUS_INITIALIZE;

        // Change the channel structs before you write them out
        m31->mVAParamsChannel.vaBitDepth = 10;
        m31->mVInParamsChannel.vInBitDepth = 10;
        m31->mVInParamsChannel.vInBitDepthOut = 10;
        m31->mEHParamsChannel.eHBitDepth = 10;

        if (!m31->SetAllParams(M31_CH0)) return AJA_STATUS_INITIALIZE;

}
```

Control

The MB86M31 is controlled by changing the state of encode and video in components, and overall main state. This is done using the helper class functions:

```
bool ChangeMainState (HevcMainState mainState, HevcEncodeMode encodeMode);
bool ChangeEHState (HevcEhState ehState, uint32_t streamBits);
bool ChangeVInState (HevcVinState vinState, uint32_t streamBits);
```

To begin an encode you would first initialize the HEVC, setup the encoder using an M31VideoPrest, then issue the following control commands:

```
#include "ntv2m31.h"

AJAStatus BeginEncoder(CNTV2Card* device, uint32_t streamBits)
{
        // Allocate our M31 helper class
        CNTV2m31 m31 = new CNTV2m31(device);

        if (!m31->ChangeMainState(Hevc_MainState_Encode, Hevc_EncodeMode_Single))
                return AJA_STATUS_INITIALIZE;

        if (!m31->ChangeVInState(Hevc_VinState_Start, streamBits))
                return AJA_STATUS_INITIALIZE;

        if (!m31->ChangeEhState(Hevc_EhState_Start, streamBits))
                return AJA_STATUS_INITIALIZE;

        return AJA_STATUS_SUCCESS;
}
```

The current model used to stop the codec is to mark the last frame to be encoded and have each thread monitor its progress through the pipeline. After processing the last frame each thread discontinues further frame processing and the pipeline starves. Once the last HEVC frame has been transferred from the codec, it can be moved cleanly to the stop state by doing the following:

```
AJAStatus StopEncoder(CNTV2Card* device, uint32_t streamBits)
{
```

```
	// Allocate our M31 helper class
	CNTV2m31 m31 = new CNTV2m31(device);

	if (!m31->ChangeEhState(Hevc_EhState_ReadyToStop, streamBits))
		return AJA_STATUS_INITIALIZE;

	if (!m31->ChangeEhState(Hevc_EhState_Stop, streamBits))
		return AJA_STATUS_INITIALIZE;

	if (!m31->ChangeVInState(Hevc_VinState_Stop, streamBits))
		return AJA_STATUS_INITIALIZE;

	if (!m31->ChangeMainState(Hevc_MainState_Init, Hevc_EncodeMode_Single))
		return AJA_STATUS_INITIALIZE;

	return AJA_STATUS_SUCCESS;
}
```

VideoTransfer

Frames are sent to and from the encoder using the following API's located in the M31 helper class.

```
bool RawTransfer (uint8_t* pBuffer, uint32_t dataSize, bool lastFrame);
bool EncTransfer (uint8_t* pBuffer, uint32_t bufferSize, uint32_t& dataSize,
bool& lastFrame);
```

Once the encoder has started you will pass uncompressed YUV planar frames into the encoder using RawTransfer, and when the encoder has encoded the frame you can get them back using EncTransfer. See the discussion on the HEVC demo application.

7 — HEVC Monitor

Overview

The NTV2 HEVC monitor is a development tool that provides setup, control and state information for the AJA Corvid HEVC board. The Corvid HEVC board is a standard NTV2 video board coupled with an HEVC hardware codec. The NTV2 driver controls both the NTV2 components and the HEVC codec. The HEVC monitor is a window to the NTV2 driver which tracks the hardware state and video transfer queues to and from the codec. The NTV2 components can be monitored using the standard NTV2 Watcher and NTV2 Cables applications.

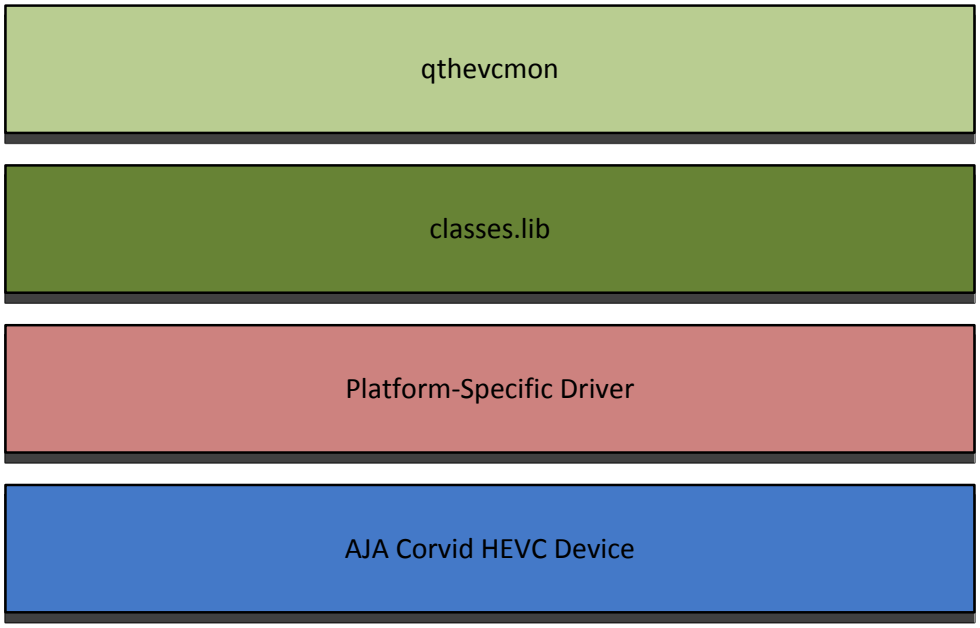


Figure 4 — HEVC Monitor Application software stack

The HEVC Monitor is a Qt application with a tabbed presentation. The layout of the interface is tightly coupled to the codec hardware and control architecture. The HEVC codec provides encoding setup registers and 3 transaction interfaces to control state and transfer video data.

The top line of the HEVC monitor contains a board selector and reset button. The reset button will reset the codec hardware and all driver control logic and queues. It should only be used when the codec is idle or as reached and error state. The result of a reset during codec operation will be random! The “Control” tab provides access to the codec command interface and encoding setup parameters. The “Stream” tab monitors the codec DMA interfaces and maintains statistics for the driver queues that stream data. The “Debug” tab provides driver and firmware revision information and can enable/disable driver debug output to the system log file.

Control Tab

The HEVC monitor control tab provides access the codec command interface and encoding setup parameters.

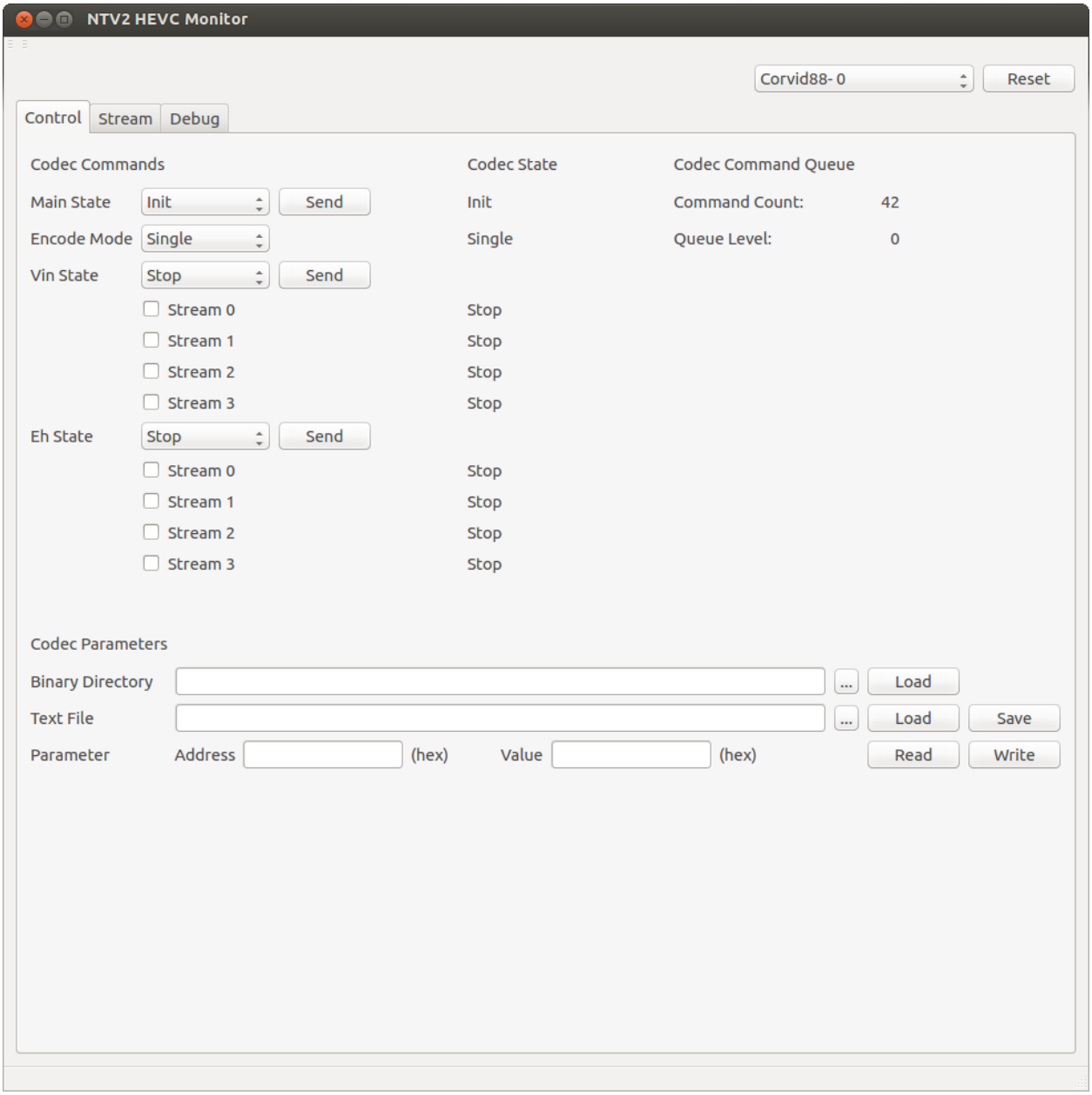


Figure 5 — HEVC Monitor Control Tab

The “Codec Commands” interface allows the user to send state change commands to the codec. The desired state is selected for the main, video input and encoder streams using the drop down

boxes. The send buttons sends the request to the codec. The stream check boxes determine which streams receive the stream state commands. There is no filtering of the commands to insure the codec remains in an operational state. See the codec documentation for detailed descriptions of the codec state machine.

The “Codec State” data is the current state of the codec state machine. The main state will indicate “Boot” after a codec reset. The “Init” state indicates that the codec is ready to receive encode parameters. The “Encode” state indicates the codec is ready to encode. The encode mode can be “Single” or “Multiple” which controls how many streams the codec can encode. The Vin and Eh states can be “Stop” or “Start”. In the stop state the encoder can be reconfigured using the codec parameters interface. In the start state the codec is ready to encode video frames.

The “Codec Command Queue” information displays the current state of the NTV2 driver command queue. It contains the current count of the commands issued to the codec between resets and the current command queue level.

The “Codec Parameters” interface can load and dump codec encoding parameters. The binary directory is used to specify a directory containing the binary parameter files provided by the codec vendor. The codec parameters can also be saved and loaded from a single text file. The file is formatted as simple 32 bit address/value pairs that can be viewed with a text editor and compared using standard text file comparison applications. This can be useful in debugging encoder settings. There is also a parameter peek/poke interface to allow examining and changing individual parameter registers.

Stream Tab

The “Stream” tab monitors the codec DMA interfaces and maintains statistics for the driver queues that stream data.

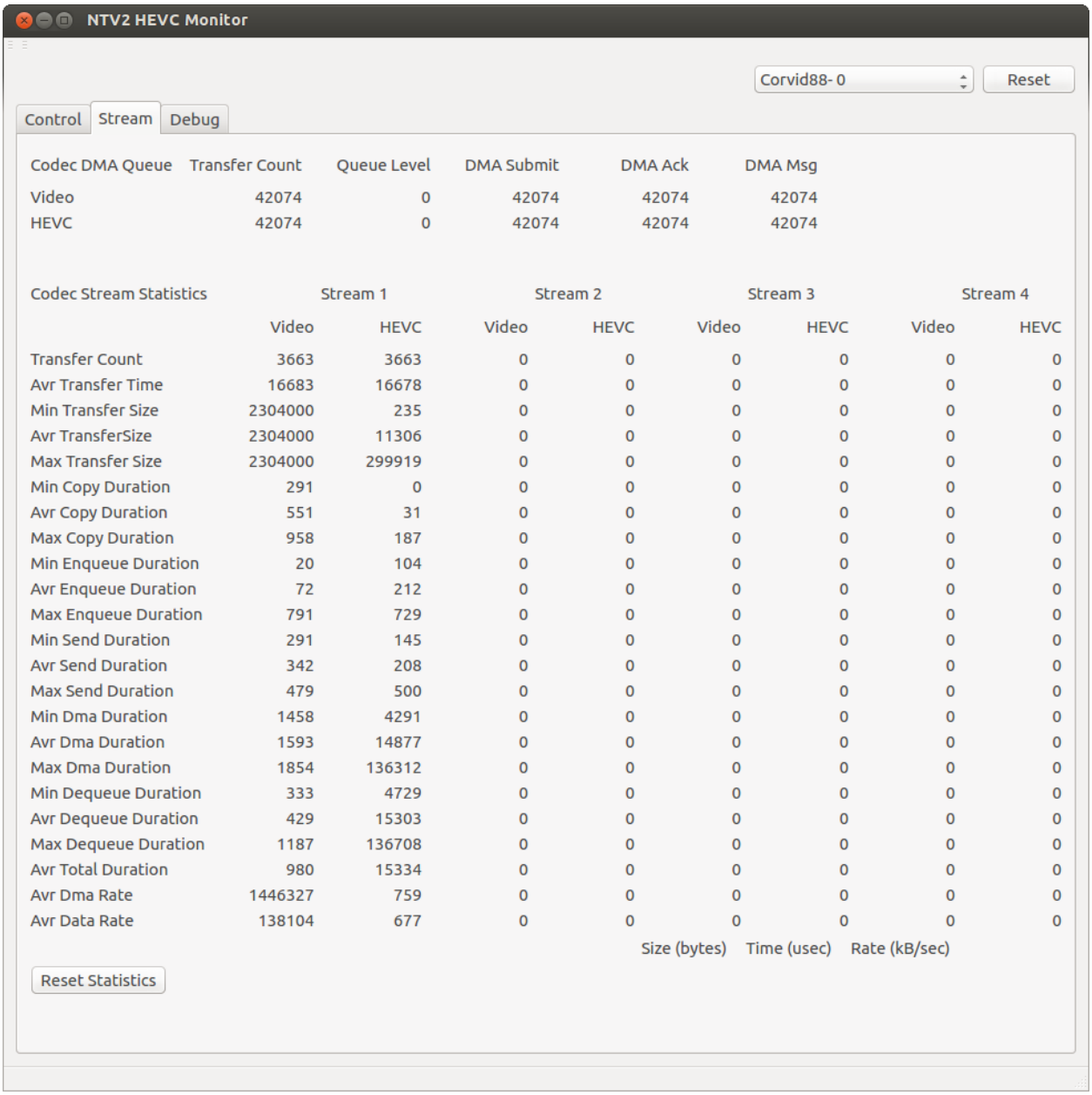


Figure 6 — HEVC Monitor Stream Tab

The “Codec DMA Queue” interface displays the current state of the NTV2 driver codec queues. There is a raw video queue and a compressed HEVC queue. The transfer count is the number of DMA transfers operations completed by each queue. The queue level is the current number of transfers in the driver queue. The DMA submit value is the current number of DMA transfers issued to the codec. The DMA Ack value is the current number of acknowledge interrupts received from the codec. The DMA Msg value is the current number of DMA completion message interrupts received from the codec. For more information concerning the codec transactions see the codec documentation.

The “Codec Stream Statistics” contains counts and timing information for each of the codec streams. Sizes are displayed in bytes, time in microseconds and data rates in kilobytes per second.

- Transfer Count — The number of transfers completed for this stream.
- Avr Transfer Time — The average time between transfers.
- Min Transfer Size — The minimum, average and maximum size of a video transfer.
- Avr Transfer Size
- Max Transfer Size
- Min Copy Duration — The minimum, average and maximum duration of a
- Avr Copy Duration — video frame bounce buffer copy. This is the software time
- Max Copy Duration — required to copy the video frame.
- Min Enqueue Duration — The minimum, average and maximum time between adding
- Avr Enqueue Duration — a DMA transfer to the queue and sending the DMA request
- Max Enqueue Duration — to the codec.
- Min Send Duration — The minimum, average and maximum time between sending
- Avr Send Duration — a DMA request to the codec and the acknowledge interrupt
- Max Send Duration — from the codec.
- Min DMA Duration — The minimum, average and maximum time between the codec
- Avr DMA Duration — acknowledge interrupt and completion message interrupt. This
- Max DMA Duration — is the hardware time required for the DMA transfer.
- Min Dequeue Duration — The minimum, average and maximum time between the adding
- Avr Dequeue Duration — a transfer to the DMA queue and releasing the queue entry. This
- Max Dequeue Duration — is the software time required for the DMA transfer.
- Avr Total Duration — The total software time required for the video transfer.
- Avr DMA Rate — The average hardware DMA data rate.
- Avr Data Rate — The average stream data rate.

The DMA duration of the HEVC stream does not reflect the actual hardware DMA time. It appears the codec stalls this DMA operation until a frame is available. This also affects the calculation of the HEVC stream DMA rate.

There is a “Reset Statistics” button that can be used to restart the statistics tracking at any time.

Debug Tab

The “Debug” tab provides driver and firmware revision information and can enable/disable driver debug output to the system log file.

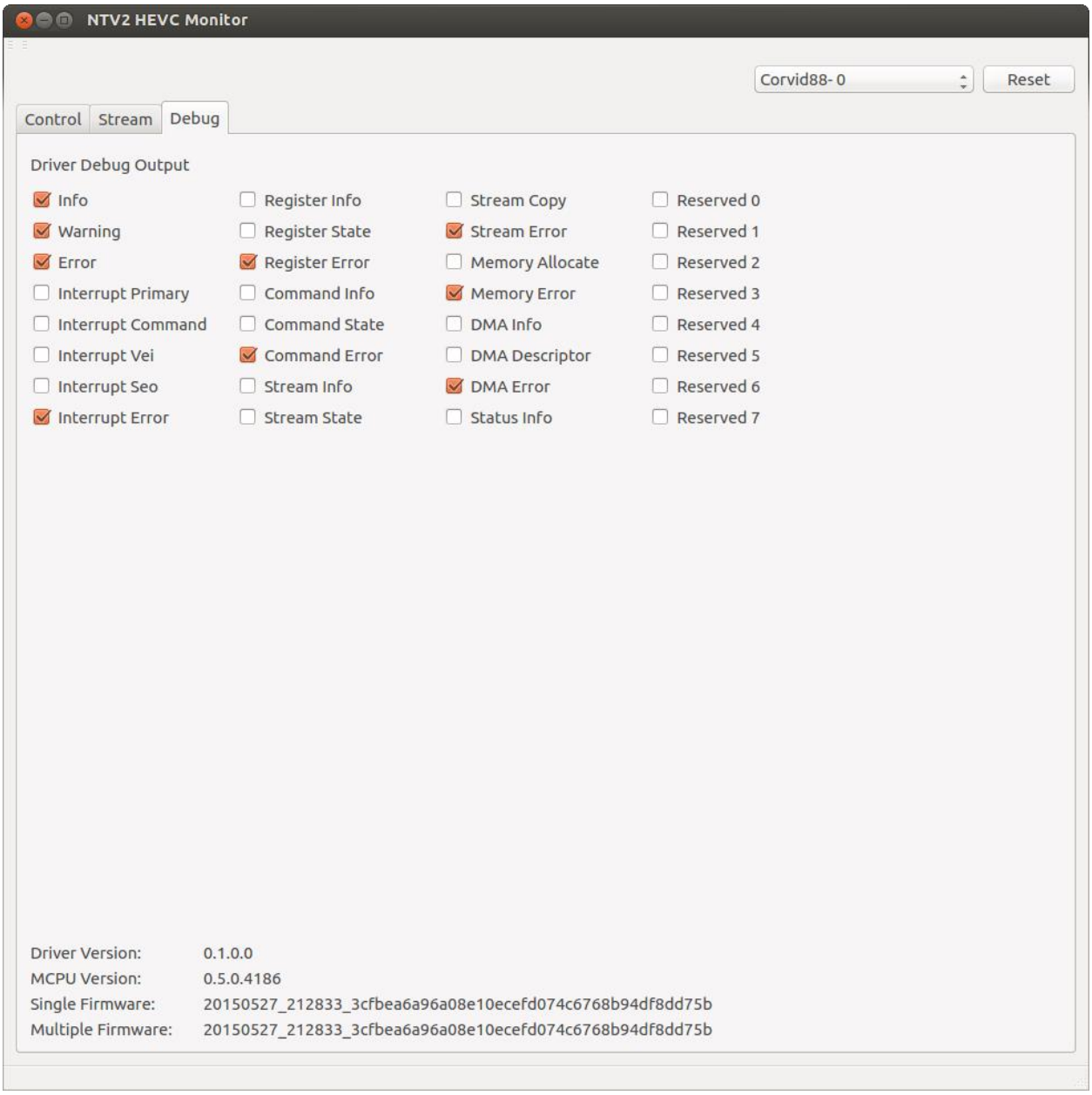


Figure 7 — HEVC Monitor Debug Tab

The “Driver Debug Output” check boxes enable various debug messages from the NTV2 driver. The messages are written to the system driver debug output log. In Linux, messages can be monitored using:

$ sudo tail –f /var/log/syslog

In Windows, use the dbgview application from the Microsoft Technet website.

Driver debug messages can be very useful in debugging all types of software and hardware issues. The driver debug information is categorized into several subsystems. The info messages

are usually one per driver call for monitoring how the driver is being driven by the application or the operating system. The state messages contain more detail on how the driver performs operations. The error messages are warnings and errors that usually require developer attention.

The initial info, warning and error information relates mostly to driver loading and unloading. The interrupt category messages are reported from the interrupt routines. The command, vei (raw video) and seo (encodec video) information relate to the interrupt driver requests to the codec. The register category will log all codec register access. The command and stream categories are for monitoring the codec command and streaming transfer operations. The DMA category messages detail the setup of the DMA registers. There are also categories for bounce buffer allocation and driver status information requests.

The debug tab also contains version information for the NTV2 codec driver and internal codec software and firmware.

8 — NTV2EncodeHEVC Demo Application

Overview

The NTV2 encode HEVC application demonstrates capture of live video from an SDI source to a raw HEVC disk file using an AJA Corvid HEVC board. Encode HEVC programs the hardware using the NTV2 classes library including new extensions to encode video. The hardware is managed by the NTV2 driver which provides the functions to manage streaming video to the classes library.

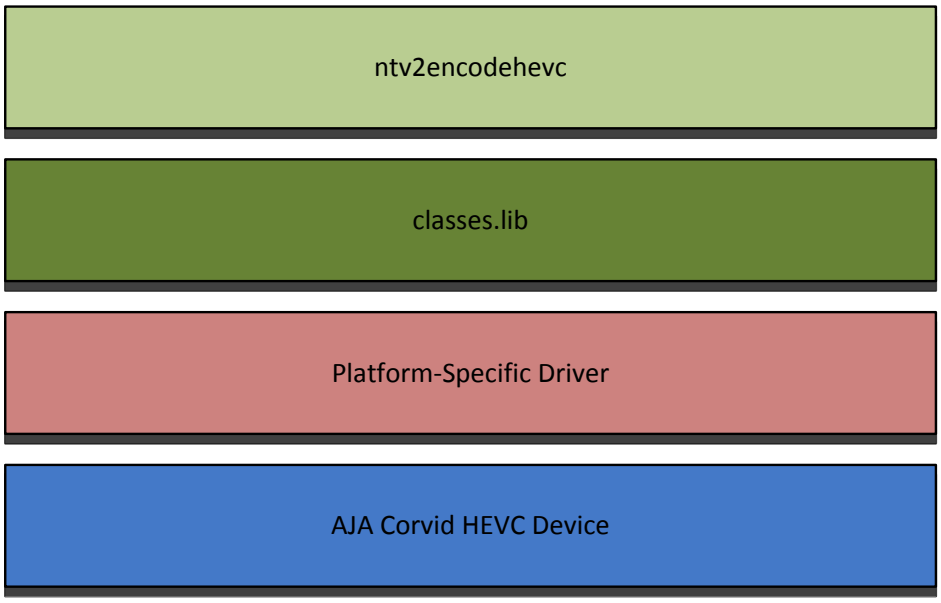


Figure 8 — NTV2EncodeHEVC Application software stack

Encode HEVC is a command line application with options to control the NTV2 and HEVC configuration. While in operation, the NTV2 components can be monitored using the NTV2 Watcher and NTV2 Cables applications. The HEVC codec can be monitored using the NTV2 HEVC monitor application.

Build and Run

The Encode HEVC application is located in the demoapps folder of the NTV2 distribution. It uses the NTV2EncodeHEVC class from the democlasses folder to do the video setup and capture. The application has standard build files for each supported platform. The ntv2encodehevc executable is built to the standard bin folder. See the NTV2 SDK installation guide for more platform specific build information.

The Encode HEVC application should be run from the command line. Without optional parameters it will detect the input format of the SDI input video and if a matching preset is available, compress it in 8 bit 420 format. By default, the codec will be configured in single stream mode. If the input is UHD, use the –q option to indicate that the 4 – SDI inputs should be treated as a single UHD stream.

```
$ ./ntv2encodehevc
Capture: M31_FILE_1280X720_420_8_5994p

           Capture  Capture
   Frames   Frames   Buffer
Processed  Dropped    Level
     3595        0        1

Capture last frame number 3663
Output last frame number 3663
```

The Encode HEVC application can also compress multiple streams of various formats. Run an instance of the application for each stream and specify the –c[1-4] option to indicate which stream to configure. For instance "`$ ./ntv2encodehevc -c3`" configures the codec to compress using stream 3.

The format of the encoded stream can be set using the –f option. Use the –lf option to list the supported formats.

```
$ ./ntv2encodehevc -lf
M31 Formats
0:  8 Bit YCbCr 420 Planar
1:  10 Bit YCbCr 420 Planar
2:  8 Bit YCbCr 422 Planar
3:  10 Bit YCbCr 422 Planar
```

The codec preset used to encode the stream can be set using the –p option. Use the –lp option to list the supported presets. This option will override the automatic stream format detection.

```
$ ./ntv2encodehevc -lp
M31 Presets
0:  FILE 1280x720 420 Planar 8 Bit 50p
1:  FILE 1280x720 420 Planar 8 Bit 59.94p
2:  FILE 1280x720 422 Planar 10 Bit 50p
3:  FILE 1280x720 422 Planar 10 Bit 59.94p
4:  FILE 1920x1080 420 Planar 8 Bit 50p
5:  FILE 1920x1080 420 Planar 8 Bit 59.94p
6:  FILE 1920x1080 422 Planar 10 Bit 50p
```

7: FILE 1920x1080 422 Planar 10 Bit 59.94p
8: FILE 3840x2160 420 Planar 8 Bit 50p
9: FILE 3840x2160 420 Planar 8 Bit 59.94p
10: FILE 3840x2160 420 Planar 10 Bit 50p
11: FILE 3840x2160 420 Planar 10 Bit 59.94p
12: FILE 3840x2160 422 Planar 8 Bit 50p
13: FILE 3840x2160 422 Planar 8 Bit 59.94p
14: FILE 3840x2160 422 Planar 10 Bit 50p
15: FILE 3840x2160 422 Planar 10 Bit 59.94p

The Encode HEVC application can output audio capture from the SDI input stream in the file raw.aiff by specifying the –a[1-16] option. The file format is 16 bit 48 kHz PCM with 1-16 channels.

The HEVC codec supports picture data added to the HEVC stream. Use the –i option to activate this feature. The data contains a sequence identifier, presentation time and optional data that can include supplemental enhancement information (SEI). The codec also outputs stream information that contains the picture sequence identifier, presentation time, etc. The stream information for each frame is written to the raw.txt file.

The –t option adds two timecode burns to the video. The top burn is based on the count of frames compressed. The bottom burn is the SMPTE-12 (RP-188) timecode from the SDI stream.

Encode HEVC captures the SDI video input, encode it as HEVC and writes the file raw.hevc to the default directory. If the codec is configured for multistream compression the file will be raw_1.hevc, raw_2.hevc, etc. To stop the capture type ctrl-c. The raw.hevc file can be viewed using ffplay (install from the internet). For HD video use:

```
$ ffplay raw.hevc
```

For UHD streams the video display can be scaled using:

```
$ ffplay –vf “scale=1920:1080” raw.hevc
```

HEVC playback is a CPU intensive process and may not run in real-time. Use the s-key during ffplay to enter single frame mode. The space-key resumes normal playback.

Design

NTV2 Encode HEVC captures uncompressed video from the SDI input and writes an HEVC compressed file. The current data flow captures the uncompressed video to system memory to allow video processing by the application before being HEVC encoded.

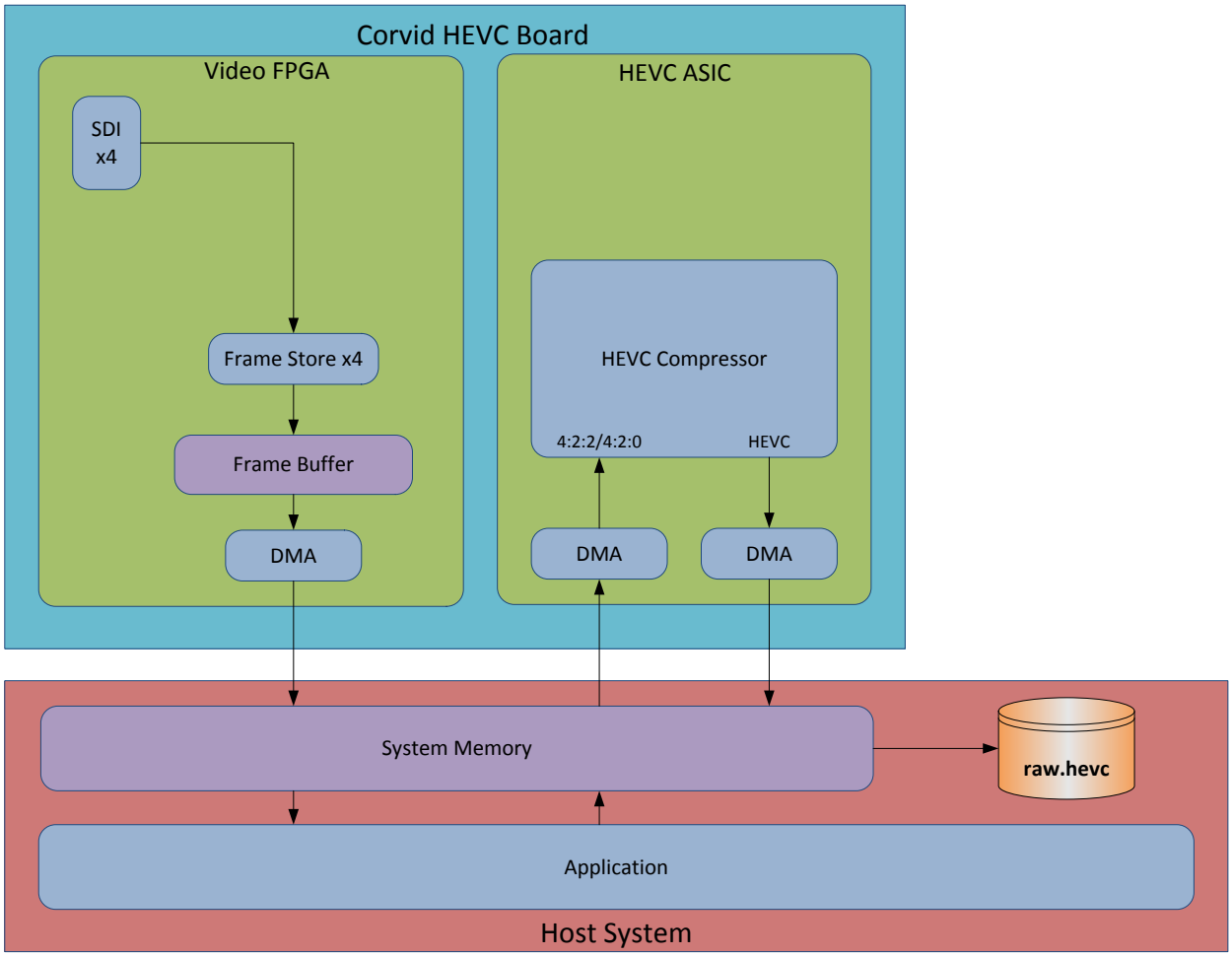


Figure 9 — Data flow

The Encode HEVC application configures the video FPGA and HEVC codec based on the specified codec preset. This determines the NTV2 video format and the frame buffer format used to capture the video. It also determines the raster and pixel format expected by the codec. The preset also configures the characteristics of the video compression. Code could be added to modify these codec parameters before processing starts to change bit rates, etc.

Thread Model

Encode HEVC uses a simple threading model to move the captured SDI video from the NTV2 frame buffer to system buffers, the HEVC codec and finally the output file. The threads move data to and from the HEVC board using DMA.

The video input thread uses NTV2TransferWithAutoCirculate() to move each captured video frame from the FPGA frame buffer to the video input system memory ring.

The video process thread represents some arbitrary video processing controlled by the application. This could CPU, GPU, etc. processing and for this application produces a result in the same video and pixel format as the original. The example currently does a simple data copy of each video frame from the video input ring to the video raw ring.

The codec raw thread uses NTV2 RawTransfer() to move each raw video frame from the video raw ring to the hardware codec where the video is compressed.

The codec HEVC thread uses NTV2 EncTransfer() to move each compressed video frame from the hardware codec to the video HEVC ring.

The file writer thread uses standard C library file functions to write each compressed video frame from the video HEVC ring to the raw.hevc output file.

Startup and Shutdown

The Encode HEVC application configures the HEVC codec to run in asynchronous mode. This makes stream startup simple as the codec waits for data to begin compression. However the codec is particular about how it is shutdown to avoid an error state. This is not as important when encoding a single stream since the codec can be reset between shutdown and subsequent startups. However this will not be possible when encoding multiple streams that do not all start and stop at the same time.

The current model used to stop the codec is to mark the last frame to be encoded and have each thread monitor its progress through the pipeline. After processing the last frame each thread discontinues further frame processing and the pipeline starves. So far, it does not appear that the codec requires raw frames to the flush the encoded data after the last frame has been received. Once the last HEVC frame has been transferred from the codec, it can be moved cleanly to the stop state.

ToDo

The first version of the NTV2 encode HEVC application demonstrates encoding of a single video stream in various HD and UHD formats and pixel resolutions. Future work could include:

- More tested formats such as NTSC, PAL, 1080i…
- More tested frame rates

- Audio and ancillary data
- Multistream (running multiple copies of the application)
- Support for encoding from the codec SDI interface (vif)
- HEVC parameter settings
- Non SDI rasters

Multistream would demonstrate independent control of codec streams. Using the codec SDI inputs would support the current data flow model and also allow the pixel format of the system based video processing to be different than the encoded pixel format. HEVC parameter changes would include setting the desired HEVC data rate.