APPS048-Codensity T408™ and T432 ™ YUVbypass Application Note

V2.1

10/09/2021

Table of Contents

1 Table of Abbreviations

Abbreviation:	Full Form:
HW	Hardware
AI	Artificial Intelligence
API	Application Programming Interface
AUD	Access Unit Delimiters
AV	Audio Video
AV1	Alliance Open Media Video 1 Codec
CABAC	Context Adaptive Binary Arithmetic Coding
CBR	Constant Bit Rate
CPU	Central Processing Unit
CRF	Constant Rate Factor
DSP	Digital Signal Processing
EOF	End Of File
FIXQB	Fixed Quantization Parameter
FF	Form Factor
FPS	Frames Per Second
GOP	Group of pictures
HDR	High Dynamic Range
HHHL	Half Height, Half Length
HHFL	Half Height, Full Length
HLG	Hybrid Log Gamma
HRD	Hypothetical Reference Decoder
IE	Inference Engine
Mbps	Mega Bit per second
MBps	Mega Byte per second
NLP	Natural Language Processing
NPU	Neural Processing Unit
NVMe	Non-Volatile Memory express
PCIe	Peripheral Component Interconnect express
PPS	Picture Parameter Set
QP	Quoted Printable
RGB	Red Green Blue
RGBA	Red Green Blue Alpha
RDO	Optimization
ROI	Region of interest
SDK	Software Development Kit
SEI	Supplemental Enhancement Information
SRIOV	Single Root I/O Virtualization
SPS	Sequence Parameter Set
VBR	Variable Bit Rat
VCL	Video Coding Layer

VPS	Video Parameter Set
VUI	Video Usability Information

2 References

Technical Notes:
PPT document - Hardware descriptor based TranscodingQuadra.pptx by Philip Tang.
Specifications:

3 Background

This document describes YUV bypass support in the NETINT T408 transcoder/encoder/decoder.

3.1 Intended Audience

This document is intended to help engineers/technicians/developers integrating NETINT transcoders to better understand YUV bypass flow.

3.2 Compatibility

Software Compatibility

This guide is intended to be used with NETINT Codensity T408 Video Transcoder software Release TBD.

Hardware Compatibility

Release TBD supports NETINT Codensity T408 Video Transcoder hardware.

Operation system

N/A.

4 YUVbypass

4.1 Terminology

- AVFrame
  - FFmpeg term for structure that carries any decoded frame information
  - Stands for Audio/Video frame
  - We do not work with the audio side so ignore that part
- SWAVFrame
  - At NETINT we refer to this as an AVFrame that stores all raw decoded video information within a buffer in host memory
- HWAVFrame or HWFrame
  - At NETINT we refer to this as an AVFrame that is using a HWdescriptor in place of raw decoded frame information.
  - In FFmpeg, it is technically just an AVFrame with a pixel format that is hwaccelerator based
  - We sometimes use this term interchangeably with hwdescriptor
- Hwdescriptor
  - Token referring to an address in T408 memory that holds the raw YUV info
  - Gets passed up to host during decoder read
  - Gets sent down to encoder during encoder write
- Hwdownload
  - A filter operation that converts a HWAVFrame to a SWAVframe
- Hwupload
  - A filter operation that converts a SWAVframe to HWAVframe
- YUVbypass
  - The event where downstream block (encoder engine) receives a Hwdescriptor as input rather than raw YUV. (yuv is 'bypassing' travelling to host and back)

4.2 Intro

Traditional transcoding flow is "*Write encoded stream to decoder -> Decode -> Read raw YUV from decoder -> Write raw YUV to encoder -> Encode -> Read encoded bitstream*" after setup decoder and encoder. Transcoding flow with HW descriptors is "*Write encoded stream to decoder -> Decode -> Read descriptor -> Write descriptor to encoder -> Encode -> Read encoded bitstream*". This is an example of YUVbypass. The HWdescriptor is only 24 bytes. So YUVbypass will reduce the bandwidth usage of PCIE, especially for 1 input multi output.

4.3 Data structure

The struct for HWdescriptor is defined as:

```
// 24 bytes
typedef struct _ni_hwframe_surface
{
  int32_t device_handle;

  int8_t i8FrameIdx;
  int8_t i8InstID;
  uint16_t ui16SessionID;
```

```
    uint16_t ui16width;
    uint16_t ui16height;

    int8_t bit_depth; //1 ==8bit per pixel, 2 ==10
    int8_t encoding_type; //h264/265
    int8_t seq_change;
    int8_t rsvd;
    int32_t device_handle_ext; // high 32-bit device handle
    int8_t rsv[4];
} ni_hwframe_surface_t;
```

4.4 New APIs

4.4.1 Introduction

The following table lists the libxcoder API functions related to YUVbypass:

API function	Description
ni_device_session_init_framepool	Setup framepool for hwupload
ni_device_session_copy	Copies existing decoding session params for hw frame usage
ni_device_handle_map_SN	Queries device Serial number
ni_device_session_hwdl	Reads YUV data with hw descriptor from the card
ni_device_session_hwup	Writes YUV data to the card and reads back hw descriptor
ni_frame_buffer_alloc_hwenc	Allocate memory for the frame buffer based on provided parameters taking into account pic line size and extra data. Applicable to YUV420p AVFrame for hw only.
ni_decode_buffer_free	Recycle a frame buffer on card
ni_frame_buffer_alloc_v4	Allocate memory for the frame buffer based on provided parameters taking into account width, height, format, stride, alignment, and extra data.

4.4.2 Additional API information

ni_device_session_init_framepool

This function sends frame pool setup info to device.
Input: p_ctx - Pointer to a caller allocated ni_session_context_t struct
Input: pool_size - framepool size in the card
Return value: On success - returns NI_RETCODE_SUCCESS
On failure - returns Netint Error Code(<0)

```
LIB_API int ni_device_session_init_framepool(ni_session_context_t *p_ctx,
                                             uint32_t pool_size);
```

ni_device_session_copy

This function copies existing decoding session params for hw frame usage.
Input: src_p_ctx - Pointer to a caller allocated source ni_session_context_t struct
Output: dst_p_ctx - Pointer to a caller allocated destination ni_session_context_t struct
Return value: On success - returns NI_RETCODE_SUCCESS
On failure - returns Netint Error Code(<0)

```
LIB_API ni_retcode_t ni_device_session_copy(ni_session_context_t *src_p_ctx,
                                            ni_session_context_t *dst_p_ctx);
```

ni_device_handle_map_SN

This function queries device Serial number. In yuvbypass mode, encoder device must keep consistent with uploader or decoder device check the serial number here to confirm this.
Input: device_handle - Device handle obtained by calling ni_device_open()
Output: serial_num - Pointer to a caller allocated ni_serial_num_t struct, 20 bytes.
Return value: On success - returns NI_RETCODE_SUCCESS
On failure - returns Netint Error Code(<0)

```
LIB_API ni_retcode_t ni_device_handle_map_SN(ni_device_handle_t device_handle,
                                             ni_serial_num_t *p_serial_num);
```

ni_device_session_hwdl

This function reads YUV data with hw descriptor from the card.
Input: p_ctx - Pointer to a caller allocated ni_session_context_t struct
Output: p_data - Pointer to a caller allocated ni_session_data_io_t struct which contains either a ni_frame_t data frame or ni_packet_t data packet.
Intput: hwdesc – Pointer to a caller allocated ni_hwframe_surface_t struct.
Return value: On success - returns Total number of bytes read
On failure - returns Netint Error Code(<0)

```
LIB_API int ni_device_session_hwdl(ni_session_context_t* p_ctx,
                                   ni_session_data_io_t *p_data,
                                   ni_hwframe_surface_t* hwdesc);
```

ni_device_session_hwup

This function writes YUV data to the card and reads back hw descriptor.
Input: p_ctx - Pointer to a caller allocated ni_session_context_t struct
Intput: p_src_data - Pointer to a caller allocated ni_session_data_io_t struct which contains either a ni_frame_t data frame or ni_packet_t data packet.
Output: hwdesc – Pointer to a a caller allocated ni_hwframe_surface_t struct.
Return value: On success - returns Total number of bytes read
On failure - returns Netint Error Code(<0)

```
LIB_API int ni_device_session_hwup(ni_session_context_t* p_ctx,
                                   ni_session_data_io_t *p_src_data,
                                   ni_hwframe_surface_t* hwdesc);
```

ni_frame_buffer_alloc_hwenc

This function allocate memory for the frame buffer based on provided parameters taking into account pic line size and extra data. Applicable to YUV420p AVFrame for hw only..
Output: p_frame - Pointer to a caller allocated ni_frame_t struct
Input: video_width - Width of the video frame
Input: video_height - Height of the video frame
Input: linesize - Picture line size
Input: alignment - Allignment requirement
Input: extra_len - Extra data size (incl. meta data)
Return value: On success - returns Total number of bytes read
On failure - returns Netint Error Code(<0)

```
LIB_API ni_retcode_t ni_frame_buffer_alloc_hwenc(ni_frame_t *pframe,
                                                 int video_width,
                                                 int video_height,
                                                 int extra_len);
```

ni_decode_buffer_free

This function recycle a frame buffer on card. When the YUV data is finished, send recycle command to the card.
Input:
Input: surface – Pointer to a a caller allocated ni_hwframe_surface_t struct.
Input: device_handle - Device handle obtained by calling ni_device_open().
Input: device_handle – Windows Event handle obtained by calling ni_create_event().

```
LIB_API ni_retcode_t ni_decode_buffer_free(ni_hwframe_surface_t* surface,
                                           ni_device_handle_t device_handle,
                                           ni_event_handle_t event_handle);
```

ni_frame_buffer_alloc_v4

This function allocates memory for the frame buffer based on provided parameters taking into account width, height, format, stride, alignment, and extra data. Used when HW download.
Output: pframe – Pointer to caller allocated ni_frame_t
Input: pixel_fromat - pixel_format
Input: video_width - Width of the video frame
Input: video_height - Height of the video frame
Input: linesize - horizontal stride
Input: alignment - apply a 16 pixel height alignmen
Input: extra_len - meta data size
Return value: On success - returns Total number of bytes read
On failure - returns Netint Error Code(<0)

```
LIB_API ni_retcode_t ni_frame_buffer_alloc_v4(ni_frame_t *pframe,
                                              int pixel_format,
                                              int video_width,
                                              int video_height,
                                              int linesize[],
                                              int alignment,
                                              int extra_len);
```

4.5 Usage

4.5.1 HW Transcode

There are two way to enable YUVbypass, with parameter -xcoder-params “out=hw” or -hwframes 1. For sequence changes, YUVbypass must add “-noautoscale” parameter.

Examples:

Traditional:

```
sudo ffmpeg -vsync 0 -c:v h265_ni_dec -i input.h265 -c:v h264_ni_enc -xcoder-params "RcEnable=1:bitrate=7500000" output.h264 –y
```

Hwframes ON (YUVbypass ON):

```
sudo ffmpeg -vsync 0 -c:v h265_ni_dec -dec 0 -xcoder-params “out=hw” -i input.h265 -c:v h264_ni_enc -enc 0 -xcoder-params "RcEnable=1:bitrate=7500000" output.h264 -y
```

or

```
sudo ffmpeg -vsync 0 -c:v h265_ni_dec -dec 0 -hwframes 1 -i input.h265 -c:v h264_ni_enc -enc 0 -xcoder-params "RcEnable=1:bitrate=7500000" output.h264 -y
```

Hwframes OFF (Explicit, same as traditional):

```
sudo ffmpeg -vsync 0 -c:v h265_ni_dec -dec 0 -xcoder-params “out=sw” -i input.h265 -c:v h264_ni_enc -enc 0 -xcoder-params "RcEnable=1:bitrate=7500000" output.h264 -y
```

or

```
sudo ffmpeg -vsync 0 -c:v h265_ni_dec -dec 0 -hwframes 0 -i input.h265 -c:v h264_ni_enc -enc 0 -xcoder-params "RcEnable=1:bitrate=7500000" output.h264 -y
```

Hwframes ON with Sequence Change:

```
sudo ffmpeg -vsync 0 -c:v h265_ni_dec -dec 0 -xcoder-params “out=hw” -i input.h265 -noautoscale -c:v h264_ni_enc -enc 0 -xcoder-params "RcEnable=1:bitrate=7500000" output.h264 -y
```

4.5.2 HWupload

Hwupload filter can upload the YUV to the card, and receive HWdescriptor from the card.
The “device_name” is arbitrary name, must match name used for filter_hw_device.
Uploader Device ID (0 here) is similar to dec/enc ID.

Single Upload Example:

```
sudo ffmpeg -init_hw_device ni=device_name:0 -pix_fmt yuv420p -s:v 1920x1080 -r 30 -i input.yuv -filter_hw_device device_name -vf 'format=yuv420p,hwupload' -c:v h265_ni_enc -enc 0 -xcoder-params " RcEnable=1:bitrate=7500000" output.265 -y
```

This HWdescriptor can be used for multi encoding. Upload to card, encode with several encoders using same input pool, while traditional, each encoder requires its own source frame pool.

Upload Example with Spilt Filter:

```
sudo ffmpeg -init_hw_device ni= device_name:0 -pix_fmt yuv420p -s:v 1920x1080 -r 30 -i input.yuv -filter_hw_device device_name -filter_complex 'format=yuv420p,hwupload,split=2[out1][out2]' -map '[out1]' -c:v h265_ni_enc -enc 0 -xcoder-params " RcEnable=1: bitrate=3000000" dinner_upload_split_low.265 -y -map '[out2]' -c:v h265_ni_enc -enc 0 -xcoder-params "RcEnable=1:bitrate=6000000" dinner_upload_split_high.265 -y
```

Upload Example with Scale Filter:

```
sudo ffmpeg -init_hw_device ni= device_name:0 -pix_fmt yuv420p -s:v 1920x1080 -r 60 -i input.yuv -filter_hw_device device_name -vf scale=1280:720,format=yuv420p,hwupload -c:v h265_ni_enc -xcoder-params 'RcEnable=1:bitrate=2000000' output.h265
```

4.5.3 HWdownload

With HWdescriptor, HWdownload filter can get YUV from the card.
HWdownload is used for some specific cases. Actually, we can not benifit from it compared with traditional way.

Download YUV Example:

```
sudo ffmpeg -vsync 0 -c:v h264_ni_dec -xcoder-params 'out=hw' -dec 0 -i input.264 -vf hwdownload,format=yuv420p -c:v rawvideo output.yuv -y
```

Download with Crop Filter in Transcoder:

```
sudo ffmpeg -y -hide_banner -nostdin -vsync 0 -xcoder-params 'out=hw' -dec 0 -c:v h264_ni_dec -i input.264 -vf hwdownload,format=yuv420p,crop=640:360 -c:v h265_ni_enc -enc 0 output.h265
```

5 Revision History

Version	Change	Create Date	Author
1.0	Document Created	Oct 09, 2021	Roy Liu
2.0	Updated to new app note template	December 7, 2021	Balwinder Sidhu
2.1	Add introduction to new APIs	Jan 10, 2022	Roy Liu

6 Legal Notice