APPS017-Libxavcodec API Examples Application Note

V5.0

2021-12-29

Table of Contents

1 Table of Abbreviations

Abbreviation:	Full Form:
GOP	Group Of Pictures
ES	Elementary stream
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:
Specifications:

3 Background

This document describes three examples to show the usage of libavcodec API.

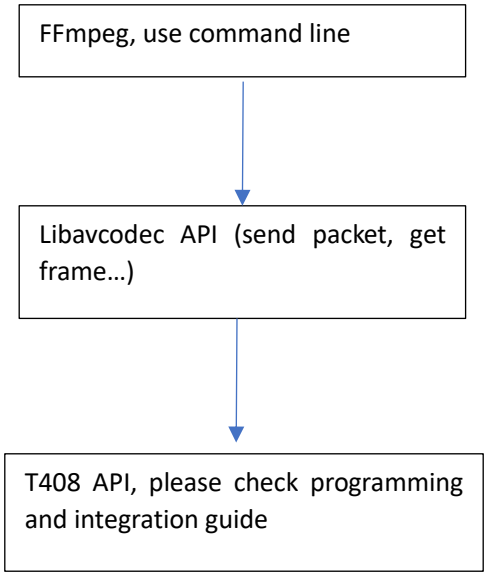

Figure 1 Different layers of FFmpeg

Normally, user can use FFmpeg command line.
For advanced customer, they can call the libavcodec API to perform decoding or encoding.
Only for sophisticated customers who has enough knowledge of low layer details of controlling T408 transcoder card, they can use T408 API. The details of T408 API is in the programming and integration guide document provided by Netint.
It is recommended to use ffmpeg command line or libavcodec API. This document and examples are about libavcodec API.

3.1 Intended Audience

This document is intended to help engineers/technicians/developers operating NETINT transcoders better understand the libxcoder API.

3.2 Compatibility

Software Compatibility

This guide is intended to be used with NETINT Codensity T408 Video Transcoder software Release 2.0.0 or newer.

Hardware Compatibility

Release 2.0.0 or newer supports NETINT Codensity T408 Video Transcoder hardware.

Operation system

All OS supporting T408 Video Transcoder.

4 libavcodec API

4.1 Intro

There are two examples to show you how to use T408 API
ni_demuxing_decode_video: an example to show you how to demux and decode an encoded file.
ni_encode_video: an example to show you how to encode a YUV file to ES.

4.2 Compile the examples without using FFmpeg make file

This method uses an independent make file to compile the two examples. If you don't want to build ffmpeg project, please use this method.

4.2.1 Prepare files

The files should be copied to: Tools/apiexample

- demuxing_decoding.c ------ this is an example provided by FFmpeg. Doesn't require T408. Put here as a reference
- Makefile ---- make file for building executables
- ni_demuxing_decoding.c ----- this is an example modified from demuxing_decoding.c, it use T408 to decode 264 or 265 input.
- ni_encode_video.c---- this is an example by using T408 to do encoding YUV file to 265 or 264 ES file.
- testdecoding.sh ---- this is a test bash file for decoding. It decodes the .264 and .265 files saved in libxcoder/test folder and decode to .YUV file.
    - 1280x720p_Basketball.264
    - 1920x1080p_ParkScene.264
    - 1920x1080p_Pedestrian.264
    - akiyo_352x288p25.264
    - akiyo_352x288p25.265

  There 5 files will be decoded with 3 methods to YUV:
  Demuxing_decoding
  FFmpeg (will automatically pick a decoder)
  Ni_demuxing_decoding (will use T408)
  If the test is passed, all three methods should generate the same .YUV. You can use "diff" command to compare them.
- testencoding.sh
  This test must run after testdecoding.sh since it requires the .yuv file generated by decoding test.
  It will encode 4 .YUV files to H265 ES file by using 2 methods:
  FFmpeg and T408
  Ni_encode_video
  Since the bitrate and parameter settings are exactly the same, they should generate exactly same result. User can use "diff" command to compare them.

4.2.2 Build examples

You need to build libxcoder and ffmpeg first. Please follow T408 quick start guide on how to build ffmpeg. This API example doesn't need ffmpeg but building ffmpeg is an easy way to prepare all the libraries.
When you build ffmpeg with "build_ffmpeg.sh", for some unknow reason, you need to run with:

Build_ffmpeg.sh --shared
After ffmpeg is built, you need to run;
Sudo make install to install all libraries.
After ffmpeg is built, you can try run_ffmpeg.sh and go through 1~9 test to make sure the T408 is initialized and working properly.
Go to tools/apiexample and run:
make
It should compile and link, generate 3 executables for each of the 3 .c files.
Note: before building these examples, you must have libavcodec and other libraries installed first.

```
FFMPEG_LIBS=    libavdevice                     \
                libavformat                     \
                libavfilter                    \
                libavcodec                      \
                libswresample                   \
                libswscale                      \
                libavutil                      \
```

As mentioned about, an easy way to get all these libraries installed is to download a ffmpeg project and compile it.

4.2.3 Test

You need to test decoding first.

4.2.3.1 testdecoding.sh

sudo bash testdecoding.sh

After this test is finished, we have these files in the folder;

```
1280x720p_Basketball.264.ffmpeg.yuv
1280x720p_Basketball.264.ni.yuv
1920x1080p_ParkScene.264.ffmpeg.yuv
1920x1080p_ParkScene.264.ni.yuv
1920x1080p_Pedestrian.264.ffmpeg.yuv
1920x1080p_Pedestrian.264.ni.yuv
akiyo_352x288p25.264.ffmpeg.yuv
akiyo_352x288p25.264.ni.yuv
akiyo_352x288p25.265.ffmpeg.yuv
akiyo_352x288p25.265.ni.yuv
demuxing_decoding
demuxing_decoding.c
demuxing_decoding.o
files.txt
Makefile
ni_demuxing_decoding
ni_demuxing_decoding.c
ni_demuxing_decoding.o
ni_encode_video
ni_encode_video.c
ni_encode_video.o
```

testdecoding.sh
testencoding.sh

The .ffmpeg.yuv files are generated by FFmpeg. The ni.yuv files are generated by ni_demuxing_decoding. They should be exactly the same.

4.2.3.2 testencoding.sh

Encoding test will use .ni.yuv files generated in the testdecoding.sh.
sudo bash testencoding.sh
After it finish, we will have these files in the folder;
1280x720p_Basketball.264.ffmpeg.yuv
1280x720p_Basketball.264.ni.yuv
1280x720p_Basketball.264.ni.yuv_ffmpeg.265
1280x720p_Basketball.264.ni.yuv_niexample.265
1920x1080p_ParkScene.264.ffmpeg.yuv
1920x1080p_ParkScene.264.ni.yuv
1920x1080p_ParkScene.264.ni.yuv_ffmpeg.265
1920x1080p_ParkScene.264.ni.yuv_niexample.265
1920x1080p_Pedestrian.264.ffmpeg.yuv
1920x1080p_Pedestrian.264.ni.yuv
1920x1080p_Pedestrian.264.ni.yuv_ffmpeg.265
1920x1080p_Pedestrian.264.ni.yuv_niexample.265
akiyo_352x288p25.264.ffmpeg.yuv
akiyo_352x288p25.264.ni.yuv
akiyo_352x288p25.264.ni.yuv_ffmpeg.265
akiyo_352x288p25.264.ni.yuv_niexample.265
akiyo_352x288p25.265.ffmpeg.yuv
akiyo_352x288p25.265.ni.yuv
demuxing_decoding
demuxing_decoding.c
demuxing_decoding.o
Makefile
ni_demuxing_decoding
ni_demuxing_decoding.c
ni_demuxing_decoding.o
ni_encode_video
ni_encode_video.c
ni_encode_video.o
testdecoding.sh
testencoding.sh

All the _ffmpeg.265 files are generated by ffmpeg. All the _niexample.265 files are generated by ni_encode_video. They should be exactly the same.
Here is a list of all the generated files and their size (note: for .265 files, the size may change with different release).

```
-rw-r--r-- 1 root root  692582400 Jul  7 15:01 1280x720p_Basketball.264.ffmpeg.yuv
-rw-r--r-- 1 root root  692582400 Jul  7 15:01 1280x720p_Basketball.264.ni.yuv
-rw-r--r-- 1 root root       2523514 Jul  7 15:02 1280x720p_Basketball.264.ni.yuv_ffmpeg.265
-rw-r--r-- 1 root root       2523514 Jul  7 15:02 1280x720p_Basketball.264.ni.yuv_niexample.265
-rw-r--r-- 1 root root  746496000 Jul  7 15:01 1920x1080p_ParkScene.264.ffmpeg.yuv
-rw-r--r-- 1 root root  746496000 Jul  7 15:01 1920x1080p_ParkScene.264.ni.yuv
-rw-r--r-- 1 root root       2322270 Jul  7 15:02 1920x1080p_ParkScene.264.ni.yuv_ffmpeg.265
-rw-r--r-- 1 root root       2322270 Jul  7 15:02 1920x1080p_ParkScene.264.ni.yuv_niexample.265
-rw-r--r-- 1 root root 1166400000 Jul  7 15:01 1920x1080p_Pedestrian.264.ffmpeg.yuv
-rw-r--r-- 1 root root 1166400000 Jul  7 15:01 1920x1080p_Pedestrian.264.ni.yuv
-rw-r--r-- 1 root root       3730877 Jul  7 15:02 1920x1080p_Pedestrian.264.ni.yuv_ffmpeg.265
-rw-r--r-- 1 root root       3730877 Jul  7 15:02 1920x1080p_Pedestrian.264.ni.yuv_niexample.265
-rw-r--r-- 1 root root      45619200 Jul  7 15:01 akiyo_352x288p25.264.ffmpeg.yuv
-rw-r--r-- 1 root root      45619200 Jul  7 15:01 akiyo_352x288p25.264.ni.yuv
-rw-r--r-- 1 root root        264576 Jul  7 15:02 akiyo_352x288p25.264.ni.yuv_ffmpeg.265
-rw-r--r-- 1 root root        264576 Jul  7 15:02 akiyo_352x288p25.264.ni.yuv_niexample.265
-rw-r--r-- 1 root root      45619200 Jul  7 15:01 akiyo_352x288p25.265.ffmpeg.yuv
-rw-r--r-- 1 root root      45619200 Jul  7 15:01 akiyo_352x288p25.265.ni.yuv
```

5 Revision History

Version	Change	Create Date	Author
1.0	Document Created	May 28,2020	Tom Han
2.0	Add independent make file and test method	June 25,2020	Tom Han
3.0	Fix a few typo	July 6	Tom Han
4.0	Remove compile with ffmpeg method and add details in building examples	July 7	Tom Han
5.0	Updated as per new app note template	December 9, 2021	Balwinder Sidhu

6 Legal Notice

Information in this document is provided in connection with NETINT products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in NETINT's terms and conditions of sale for such products, NETINT assumes no liability whatsoever and NETINT disclaims any express or implied warranty, relating to sale and/or use of NETINT products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right.

A "Mission Critical Application" is any application in which failure of the NETINT Product could result, directly or indirectly, in personal injury or death. Should you purchase or use NETINT's products for any such mission critical application, you shall indemnify and hold NETINT and its subsidiaries, subcontractors and affiliates, and the directors, officers, and employees of each, harmless against all claims costs, damages, and expenses and reasonable attorney's fees arising out of, directly or indirectly, any claim of product liability, personal injury, or death arising in any way out of such mission critical application, whether or not NETINT or its subcontractor was negligent in the design, manufacture, or warning of the NETINT product or any of its parts.

NETINT may make changes to specifications, technical documentation, and product descriptions at any time, without notice. The information here is subject to change without notice. Do not finalize a design with this information. The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications.

NETINT, Codensity, and NETINT Logo are trademarks of NETINT Technologies Inc. All other trademarks or registered trademarks are the property of their respective owners.