APPS0054- Codensity T408 & T432 GStreamer Support Application Note

V2.0

Table of Contents

1 Table of Abbreviations

Abbreviation:	Full Form:
API	Application Programming Interface
AV	Audio Video
EOF	End Of File
GOP	Group of pictures
NVMe	Non-Volatile Memory express
PCIe	Peripheral Component Interconnect express
SDK	Software Development Kit

2 References

Guides

1. Codensity T408 & T432 Quickstart Guide
2. Codensity T408 & T432 Integration & Programming Guide

Links:

1. Getting started with GStreamer's gst-build: https://www.collabora.com/news-and-blog/blog/2020/03/19/getting-started-with-gstreamer-gst-build/
2. gst-build: https://gitlab.freedesktop.org/gstreamer/gst-build.git

3 Background

This document describes how to integrate Netint transcoding capability into GStreamer to add hardware based transcode support.

3.1 Intended Audience

This document is intended for T408/T432 users that require GStreamer support.

3.2 Compatibility

Software Compatibility

This guide is applicable to all T408 and T432 releases.

Hardware Compatibility

This guide is applicable to all T408 and T432 hardware.

Operation system

This guide is specific to Linux. The OS version is not critical. This guide was developed using Ubuntu 18.04 but should extend to other Linux versions and distributions that are supported by T408, T432 and GStreamer.

4 Overview

FFmpeg/libav is a library that contains many kinds of codecs, supports various container formats and filters. NETINT transcoding capability has been fully integrated into FFmpeg.

GStreamer is one level above that and a generic multimedia framework, based on pipeline graphs. It allows to put together arbitrary multimedia processing pipelines built from different elements that execute different tasks.

The gst-libav module of GStreamer contains a GStreamer plugin for using FFmpeg/libav's encoders, decoders, muxers and demuxers. By building the gst-libav module and linking to the NETINT-enabled libav libraries, GStreamer user is able to take full advantage of NETINT codecs while working in the familiar GStreamer environment.

5 Build and installation

All the software components involved on the host system shall be rebuilt to make proper linkage. In general, the NETINT enabled FFmpeg/libxcoder libraries should be built and installed first, then the GStreamer. Follow the steps below to build various components of the system.

Ubuntu 18.04 host is used for the installation.

5.1 FFmpeg/libxcoder

NETINT transcoder card and firmware installation/upgrade shall be done following the instructions in the Codensity T408 & T432 Quickstart Guide.

Note: it is important that this version of NETINT enabled FFmpeg (and currently FFmpeg version 4.3.1 only) is the only one installed on the host system. To make sure, check /usr/local/lib and /usr/local/include folders and uninstall/clean-up any pre-existing FFmpeg versions.

FFmpeg download and NETINT patch applying shall be done following the instructions in the Quickstart guide as well. After the patch has been applied and libxcoder has been built and installed, build FFmpeg with the option of shared components and NETINT GStreamer support (and any other desired options), using the following script and command line option:

```
$ bash build_ffmpeg.sh --shared --gstreamer
```

Install FFmpeg with the following command:

```
$ sudo make install
```

5.2 GStreamer

We use GStreamer 1.18 and gst-build as an example of GStreamer build and installation.

As preparation for setup, install all the necessary build tools on the host which may include **meson**, **ninja** and all those shown missing when using *meson* to configure build environment in the next steps.

Download the GStreamer v1.18 gst-build repo:

```
$ git clone -b 1.18 https://gitlab.freedesktop.org/gstreamer/gst-build.git
```

Configure the build:

```
$ cd gst-build; meson build
```

Apply NETINT's patch to GStreamer file subprojects/gst-libav/ext/libav/gstavcodecmap.c. This is a workaround to set pixel format in the NETINT decoder context so that it can handle 8 and 10 bits per pixel format input automatically.

```
$ cp gst-libav-1.18.0_netint_v3.0.0.diff subprojects/gst-
libav/ext/libav/

$ cd subprojects/gst-libav/ext/libav/; patch -p0 < gst-libav-
1.18.0_netint_v3.0.0.diff
```

Build the gst-build from its root folder:

```
$ ninja -C build
```

5.3 Setup verification

Follow the steps in the Quickstart guide for checking the NETINT transcoder card status on the host system. When they are ready, initialize NETINT transcoder device resource pool on the host by the following command:

```
$ init_rsrc_logan
```

Test the gst-build by launching an environment with right environment variables set:

```
$ ninja -C build devenv
```

From here can try a number of things including checking the NETINT encoder/decoder elements:

```
$ [gst-1.18] $ gst-inspect-1.0 |grep _ni

libav:  avenc_h264_ni_logan_enc: libav H.264 NetInt Logan encoder
v259R1E09 encoder
libav:  avenc_h265_ni_logan_enc: libav H.265 NetInt Logan encoder
v259R1E09 encoder
libav:  avdec_h264_ni_logan_dec: libav H.264 NetInt Logan decoder
v259R1E09 decoder
libav:  avdec_h265_ni_logan_dec: libav H.265 NetInt Logan decoder
v259R1E09 decoder
```

Encoding with NETINT H.264 encoder and encode parameters specified:

```
$  gst-launch-1.0 videotestsrc num-buffers=10 ! video/x-
raw,width=1920,height=1080,format=I420 ! avenc_h264_ni_logan_enc
keep-alive-timeout=15 xcoder-
params=gopPresetIdx=2:frameRate=25:RcEnable=1:bitrate=500000:intr
aPeriod=250:lowDelay=1  ! filesink location=/tmp/test-
ni264enc.264

Setting pipeline to PAUSED ...
Pipeline is PREROLLING ...
```

```
Pipeline is PREROLLED ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
Got EOS from element "pipeline0".
Execution ended after 0:00:00.141376568
Setting pipeline to NULL ...
Encoder HW[0] INST[127]-average usage:22%
Freeing pipeline ...
```

For details of encoding configuration parameters, check the Codensity T408 & T432 Integration & Programming Guide.

6 GStreamer working with NETINT transcoder

6.1 Encoding

Section 5.3 gives an example pipeline of GStreamer working with NETINT H.264 encoder:

```
$ gst-launch-1.0 videotestsrc num-buffers=10 ! video/x-
raw,width=1920,height=1080,format=I420 ! avenc_h264_ni_logan_enc
keep-alive-timeout=15 xcoder-
params=gopPresetIdx=2:frameRate=25:RcEnable=1:bitrate=500000:intr
aPeriod=250:lowDelay=1  ! filesink location=/tmp/test-
ni264enc.264
```

The highlighted parts of this pipeline show that the NETINT encoder accepts YUV420p raw data, its element name being **avenc_h264_ni_logan_enc**, and associated element properties .

Another example of encoding a raw YUV file to HEVC output:

```
gst-launch-1.0 filesrc location=akiyo_352x288p25.yuv ! videoparse
width=352 height=288 format=i420 framerate=25/1 !
avenc_h265_ni_logan_enc ! filesink location=/tmp/aki-enc-ni.265
```

6.1.1 NETINT encoder element properties

Most of the NETINT encoder element properties as shown by gst-inspect-1.0 tool are generic codec options and have no effect. The element properties that do have effect are those private codec options. The mapping between NETINT encoder element private codec options and FFmpeg libavcodec NETINT encoder codec options is shown in Table 1. For details of those codec options especially the encoding parameters, consult the Codensity T408 & T432 Integration & Programming Guide.

GStreamer NETINT encoder element property	FFmpeg NETINT encoder codec option
enc	enc
iosize	iosize
keep-alive-timeout	keep_alive_timeout
xcoder	xcoder
xcoder-gop	xcoder-gop
xcoder-params	xcoder-params

Table 1 Mapping of NETINT encoder element properties and codec options

6.2 Decoding

Usage of NETINT decoder element in GStreamer is similar. Here is an example of decoding an HEVC source into raw YUV output.

```
gst-launch-1.0 filesrc location=720p.h265 ! h265parse !
avdec_h265_ni_logan_dec ! filesink location=/tmp/720p.yuv
```

6.3 Transcoding

Here is an example of transcoding from AVC to HEVC.

```
gst-launch-1.0 filesrc location=1080p.h264 ! h264parse !
avdec_h264_ni_logan_dec ! avenc_h265_ni_logan_enc ! filesink
location=/tmp/1080p.265
```

6.4 List of Currently Unsupported NETINT transcode features

This is a list of NETINT transcoder features that the GStreamer does not currently support. Support for these is either in development or could be added in the future if required.

- Decoding command option setting
- Run-time libav/libxcoder log level setting

6.5 Known issues

As listed in 6.4.

7 Appendix A – gst multiqueue example

When using gstreamer for transcoding single instance to multiple outputs, the 'multiqueue' option is helpful. Following example uses gstreamer, and Netint decoder and encoder to transcode an H.264 1080p30 to 32 outputs.

```
gst-launch-1.0 multiqueue sync-by-running-time=TRUE max-size-bytes=0
max-size-buffers=0 max-size-time=0 name=mq filesrc
location=dinner_1920x1080p30_300_b3.h264 ! h264parse !
avdec_h264_ni_logan_dec ! \
tee name=t ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink
location=s1.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s2.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s3.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s4.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s5.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s6.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s7.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s8.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s9.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s10.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s11.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s12.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s13.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s14.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s15.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s16.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s17.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s18.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s19.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s20.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s21.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s22.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s23.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s24.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s25.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s26.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s27.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s28.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s29.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s30.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s31.264 \
t. ! mq. mq. ! avenc_h264_ni_logan_enc ! filesink location=s32.264
```

8 Appendix B – Multiqueue with scaling example

When using gstreamer for transcoding and ladder encoding with multiple outputs, the `multiqueue` option will help reduce CPU usage.

```
gst-launch-1.0 multiqueue sync-by-running-time=TRUE max-size-
bytes=10485760 max-size-buffers=5 max-size-time=2000000000 name=mq
filesrc location=football_19201080p60_7200_H264.mp4 ! qtdemux !
h264parse ! avdec_h264_ni_logan_dec ! videorate max-rate=60 ! tee
name=t ! mq. mq. ! avenc_h264_ni_logan_enc xcoder-
params='RcEnable=1:bitrate=8000000:RcInitDelay=2000:gopPresetIdx=1:lowD
elay=1' ! fpsdisplaysink video-sink=fakesink  -v \
t. ! mq. mq. ! videoscale ! video/x-raw,width=1920,height=1080 !
avenc_h264_ni_logan_enc xcoder-
params='RcEnable=1:bitrate=4000000:RcInitDelay=2000:gopPresetIdx=2:lowD
elay=1' ! fpsdisplaysink video-sink=fakesink  -v \
t. ! mq. mq. ! videoscale ! video/x-raw,width=1280,height=720 !
avenc_h264_ni_logan_enc xcoder-
params='RcEnable=1:bitrate=5000000:RcInitDelay=2000:gopPresetIdx=1:lowD
elay=1' ! fpsdisplaysink video-sink=fakesink  -v \
t. ! mq. mq. ! videoscale ! video/x-raw,width=1280,height=720 !
avenc_h264_ni_logan_enc xcoder-
params='RcEnable=1:bitrate=2500000:RcInitDelay=2000:gopPresetIdx=2:lowD
elay=1' ! fpsdisplaysink video-sink=fakesink  -v \
t. ! mq. mq. ! videoscale ! video/x-raw,width=848,height=480 !
avenc_h264_ni_logan_enc xcoder-
params='RcEnable=1:bitrate=2400000:RcInitDelay=2000:gopPresetIdx=1:lowD
elay=1' ! fpsdisplaysink video-sink=fakesink  -v \
t. ! mq. mq. ! videoscale ! video/x-raw,width=848,height=480 !
avenc_h264_ni_logan_enc xcoder-
params='RcEnable=1:bitrate=1200000:RcInitDelay=2000:gopPresetIdx=2:lowD
elay=1' ! fpsdisplaysink video-sink=fakesink  -v \
t. ! mq. mq. ! videoscale ! video/x-raw,width=640,height=360 !
avenc_h264_ni_logan_enc xcoder-
params='RcEnable=1:bitrate=1800000:RcInitDelay=2000:gopPresetIdx=1:lowD
elay=1' ! fpsdisplaysink video-sink=fakesink  -v \
t. ! mq. mq. ! videoscale ! video/x-raw,width=640,height=360 !
avenc_h264_ni_logan_enc xcoder-
params='RcEnable=1:bitrate=900000:RcInitDelay=2000:gopPresetIdx=2:lowDe
lay=1' ! fpsdisplaysink video-sink=fakesink  -v
```

9 Revision History

Version	Change	Date	Author Name
0.1	Draft Document Created	May 4, 2022	Zhong Wang
0.2	Added encoding example and decoding support.	May 24, 2022	Zhong Wang
0.3	Formatting and patch name	July 19, 2022	Steven Zhou
0.4	Codec names update	July 19, 2022	Zhong Wang
1.0	Issued version	November 17, 2022	Balwinder Sidhu
2.0	Added Appendix B	Jan 26 2023	K. Robinson

10 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 rights1 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.