Quadra™

Integration & Programming Guide

V5.2

2025/03/13

Table of Contents

1. 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.

2. Quadra Video Processing Unit

NETINT is a supplier of high-performance, low-latency, real-time video processing units (VPUs) for x86 and ARM servers.

NETINT provides multiple stream transcoding functions and services directly to video content providers, and Transcoding as a Service (TaaS) providers, for integration into their video streaming systems and services.

NETINT's functions and services can be used for highly efficient Video-on-Demand file transcoding, as well as real-time live video streaming applications.

This guide provides an overview of NETINT Quadra video transcoding solution parameters, and the ways in which they can be used when integrating and managing transcoding into a customer's transcoding workflow.

3. Introducing Quadra Video Processing Units

Quadra has 4 encoder cores that support H.2semiplanar64/H.265/AV1 and JPEG. There are also 4 decoder cores that support H.264/H.265/VP9 and JPEG.

A PCIe Gen4 x4 interface provides the high speed connectivity to the host PC, with up to 6 GB/s of bidirectional data. This allows the host to transfer to Quadra up to 10 bit YUV 8k@60fps. Low speed connectivity is provided by 2x I2C, SPI, UART, GPIO, and JTAG.

Quadra also has a 2D Engine that contains 4 GPUs. Each GPU is capable of many types of operation on raw video input. Some supported operations are

- Scaling
- Cropping
- Video overlay
    - Video format conversion
    - Draw box
    - Rotation

An audio block is available within Quadra, comprising of 2x DSP audio processors. These are now capable of additional, general-purpose processing, to enhance transcoding performance.

Quadra also has 2 Deep Neural Network processing blocks each containing an AI engine. The AI engine is capable of 18 Trillion Operations Per Second (TOPs) for typical DNN operations, such as

- Object detection
- Classification
- Segmentation

One application of the AI engine is to provide Region of Interest (ROI) information to the encoder. The ROI is a dynamic, AI selected area of a frame, that can be used by the encoder to improve image quality. Another Quadra AI use case is background replacement. This is where the background of a video is replaced with a different, static background.

4. Intended Audience

This document is intended to assist developers when integrating Quadra into their own media systems and workflows. It is also a reference guide for customers who are directly using NETINTs video utility programs and servers.

5. Compatibility

Software Compatibility

This guide is intended to be used with Quadra Video Transcoder Release 5.2.

Hardware Compatibility

Quadra Release 5.2, and this guide supports all Quadra Video Transcoder hardware, including

- T1M (M.2)
- T1U (U.2)
- T1A (AIC)
- T2A (AIC)

System Compatibility

The Recommended System for Quadra is detailed in the **Quadra Quick Start Guide**, see section **2.4 Hardware Installation.**

6. Protocol Stack

The following diagram is a high-level block view of the entire Quadra software stack, and its connection to the device firmware via the NVMe driver.

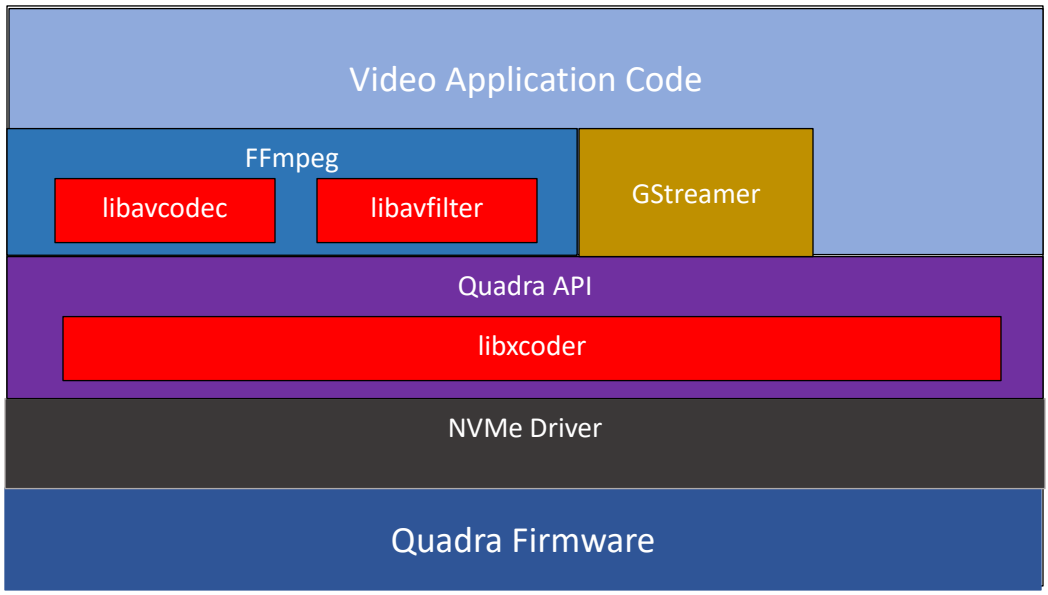


Figure 1 Quadra Software Block Diagram

Application software can access Quadra's video encoding, decoding and 2D processing services, through either high level FFmpeg plugins (libavcodec and libavfilter), through the GStreamer framework, or directly interfacing to the device via NETINT's low-level API library called ***libxcoder***.

The libxcoder library also provides an API to the Quadra Deep Neural Network services.

7. FFmpeg NETINT Command Options

7.1 Decoding

The list of FFmpeg NETINT command options for decoding can be displayed with this command:

```
ffmpeg -help decoder=<ni_dec_name>
```

<ni_dec_name> options are:

- ***h264_ni_quadra_dec***
- ***h265_ni_quadra_dec***
- ***vp9_ni_quadra_dec***
- ***jpeg_ni_quadra_dec***

The above names are for NETINTs AVC, HEVC, VP9 and JPEG decoding respectively.

Example:

```
$ ffmpeg -hide_banner -help decoder=h264_ni_quadra_dec


Decoder h264_ni_quadra_dec [H.264 NetInt Quadra decoder v---6ADEV]:
    General capabilities: delay avoidprobe hardware
    Threading capabilities: none
    Supported hardware devices: ni_quadra
    Supported pixel formats: yuv420p nv12 yuv420p10le p010le ni_quadra

h264_ni_quadra_dec AVOptions:

  -xcoder            <string>     .D.V...... Select which XCoder card to use. (default
"bestmodelload")

     bestmodelload                .D.V...... Pick the least model load XCoder/decoder
available.

     bestload                     .D.V...... Pick the least real load XCoder/decoder
available.

  -dec               <int>        .D.V...... Select which decoder to use by index.First
is 0, second is 1, and so on. (from -1 to INT_MAX) (default -1)

  -decname           <string>     .D.V...... Select which decoder to use by NVMe block
device name, e.g. /dev/nvme0n1.

  -user_data_sei_passthru <boolean>    .D.V...... Enable user data unregistered SEI
passthrough. (default false)

  -custom_sei_passthru <int>        .D.V...... Specify a custom SEI type to
passthrough. (from -1 to 254) (default -1)

  -xcoder-params     <string>     .D.V...... Set the XCoder configuration using a :-
separated list of key=value parameters.

  -keep_alive_timeout <int>        .D.V...... Specify a custom session keep alive
timeout in seconds. (from 1 to 100) (default 3)

  -low_delay         <int>        .D.V...... Enable low delay decoding mode for 1 in, 1
out decoding sequence. set 1 to enable low delay mode. Should be used only for streams
that are in sequence. (from 0 to 1) (default 0)
```

Arguments:

xcoder: specifies type of load used to determine the XCoder card to use, **model load** or **real load**

Default: **bestmodelload** instructs the system to use the decoder with the least model load

dec: specific decoder index to assign to the decoding instance

Default: -1 instructs the system to use the least loaded decoder

decname: NVMe block device name, e.g. /dev/nvme0n1 to assign to the decoding instance. When specified this takes precedence over ***dec***. It has no default value and validity check is present.

user_data_sei_passthru: enables user data unregistered SEI passthrough. See the **User data Unregistered SEI passthrough Application Note** for more details.

custom_sei_passthru: specifies a custom SEI payload to passthrough. See the **Custom SEI passthrough Application Note** for more details.

xcoder-params: specifies the decoding configuration, using a separated list of key=value parameters. See **Section 9 Decoder** for more details.

keep_alive_timeout specifies a session keep-alive timeout. This is a periodic request/response between libxcoder and the XCoder firmware, when this times out, the decoding instance on the decoder will be terminated by the XCoder firmware. Valid range is from 1-100, inclusive. This option is overwritten if a ***keepAliveTimeout*** option is specified in the xcoder-params.

low_delay enables low delay decoding mode for 1 in, 1 out, decoding sequence. This should only be used for streams that are encoded in sequence. If out-of-sequence stream is paired with low delay, longer input buffering may occur and output display order will also be out-of-sequence.

Decoding command example with decoder index specified as 0:

```
ffmpeg -y -hide_banner -nostdin -vsync 0 -c:v h264_ni_quadra_dec -dec
0 -i ../libxcoder/test/akiyo_352x288p25.264 -c:v rawvideo output_5.yuv
```

7.2 Encoding

FFmpeg NETINT command options for encoding can be seen using the following command:

ffmpeg -help encoder=<ni_enc_name>

<ni_enc_name> options are:

- *h264_ni_quadra_enc*
- *h265_ni_quadra_enc*
- *av1_ni_quadra_enc*
- *jpeg_ni_quadra_enc*

These names are for NETINT AVC, HEVC, AV1 or JPEG encoder respectively.

Example:

```
$ ffmpeg -hide_banner -help encoder=h265_ni_quadra_enc

Encoder h265_ni_quadra_enc [H.265 NETINT Quadra encoder v---6rzDV]:
    General capabilities: delay
    Threading capabilities: none
    Supported hardware devices: ni_quadra ni_quadra ni_quadra ni_quadra
ni_quadra
    Supported pixel formats: yuv420p yuvj420p yuv420p10le nv12 p010le
ni_quadra
h265_ni_quadra_enc AVOptions:
  -xcoder           <string>     E..V...... Select which XCoder card to
use. (default "bestmodelload")
     bestmodelload                E..V...... Pick the least model load
XCoder/encoder available.
     bestload                     E..V...... Pick the least real load
XCoder/encoder available.
  -enc              <int>        E..V...... Select which encoder to use
by index. First is 0, second is 1, and so on. (from -1 to INT_MAX)
(default -1)
  -ni_enc_idx       <int>        E..V...... Select which encoder to use
by index. First is 0, second is 1, and so on. (from -1 to INT_MAX)
(default -1)
  -ni_enc_name      <string>     E..V...... Select which encoder to use
by NVMe block device name, e.g. /dev/nvme0n1.
  -encname          <string>     E..V...... Select which encoder to use
by NVMe block device name, e.g. /dev/nvme0n1.
  -iosize           <int>        E..V...... Specify a custom NVMe IO
transfer size (multiples of 4096 only). (from -1 to INT_MAX) (default -1)
  -xcoder-params    <string>     E..V...... Set the XCoder configuration
using a :-separated list of key=value parameters.
```

```
  -xcoder-gop        <string>     E..V...... Set the XCoder custom gop
using a :-separated list of key=value parameters.
  -keep_alive_timeout <int>        E..V...... Specify a custom session
keep alive timeout in seconds. (from 1 to 100) (default 3)
  -gen_global_headers <int>        E..V...... Generate SPS and PPS headers
during codec initialization. (from -1 to 1) (default off)
     auto            -1           E..V......
     off             0            E..V......
     on              1            E..V......
```

Arguments:

xcoder: specifies which type of load is used to determine the XCoder card to use, model load or real load. **Default**: bestmodelload, instructs the system to use the encoder with the least model load

enc/ni_enc_idx assigns the encoding instance to a specific encoder by its index. The default value is -1. If the3.1.1 input is a hardware frame, the encoding instance will be placed on the same device as the hardware frame. If the input is a software frame the encoding instance will be placed on the least loaded encoder.

encname/ni_enc_name assigns the encoding instance to a specific decoder by its NVMe block device name, e.g. /dev/nvme0n1. When specified, this takes precedence over ***enc***. It has no default value and validity check is present.

iosize specifies a custom NVMe I/O transfer size.

xcoder-params specifies the encoding configuration using a separated list of key=value parameters. See section 8.4 for more details.

xcoder-gop specifies a custom GOP for encoding using a separated list of key=value parameters. See section 8.4 for details.

keep_alive_timeout specifies a session keep alive timeout value. This is a periodical request/response between libxcoder and XCoder firmware that when timed out, the encoding instance on the encoder will be terminated by the XCoder firmware. Valid range is from 1-100, inclusive. This option is overridden if the ***keepAliveTimeout*** option is specified in the xcoder-params.

gen_global_headers specifies whether to generate headers to extradata in advance of first frame arriving at encoder. For H.264/HEVC, the extradata includes the VPS/SPS/PPS headers, while for AV1, it includes the sequence header and related structures at the container level. This is useful for containers such as mkv, hls, asf, and flv which typically require this extradata to ensure that players and decoders are with the video parameters before actual video frames are processed.

Encoding command example using default least model load encoder placement:

```
ffmpeg -y -hide_banner -nostdin -f rawvideo -pix_fmt yuv420p -
s:v 352x288 -r 25 -i ../libxcoder/test/akiyo_352x288p25.yuv -
c:v h264_ni_quadra_enc output_7.h264
```

Transcoding example using default least model load decoder and encoder placement:

```
ffmpeg -y -hide_banner -nostdin -vsync 0 -c:v
h264_ni_quadra_dec -
i ../libxcoder/test/1280x720p_Basketball.264 -c:v
h265_ni_quadra_enc  output_9.h265
```

7.3 Filters

FFmpeg NETINT command options for filtering are shown with the following command:

```
ffmpeg -help filter=<ni_filter_name>
```

<ni_filter_name> options for NETINT filters are:

- *ni_quadra_scale*
- *ni_quadra_overlay*
- *ni_quadra_split*
- *ni_quadra_crop*
- *ni_quadra_pad*
- *ni_quadra_hwupload*
- *ni_quadra_roi*
- *ni_quadra_xstack*
- *ni_quadra_bg*
- *ni_quadra_rotate*
- *ni_quadra_drawbox*
- *ni_quadra_drawtext*
- *ni_quadra_ai_pre*
- *ni_quadra_delogo*
- *ni_quadra_merge*
- *ni_quadra_flip*
- *ni_quadra_hvsplus*

Example:

```
$ ffmpeg -hide_banner -help filter=ni_quadra_scale

Filter ni_quadra_scale
  NetInt Quadra video scaler v---64DEV
    Inputs:
       #0: default (video)
    Outputs:
       #0: default (video)
ni_scale AVOptions:
  w                 <string>     ..FV..... Output video width
  width             <string>     ..FV..... Output video width
  h                 <string>     ..FV..... Output video height
  height            <string>     ..FV..... Output video height
  force_original_aspect_ratio <int>        ..FV..... decrease
or increase w/h if necessary to keep the original AR (from 0 to
2) (default disable)
     disable                      ..FV.....
     decrease                     ..FV.....
     increase                     ..FV.....
  format            <int>        ..FV..... set_output_format
(from 0 to 13) (default auto)
     yuv420p          0           ..FV.....
     yuyv422          1           ..FV.....
     uyvy422          2           ..FV.....
     nv12             3           ..FV.....
     argb             4           ..FV.....
     rgba             5           ..FV.....
     abgr             6           ..FV.....
     bgra             7           ..FV.....
     yuv420p10le      8           ..FV.....
     nv16             9           ..FV.....
     bgr0             10          ..FV.....
     p010le           11          ..FV.....
     bgrp             12          ..FV.....
     auto             13          ..FV.....
  force_divisible_by <int>        ..FV..... enforce that the
output resolution is divisible by a defined integer when
force_original_aspect_ratio is used (from 1 to 256) (default 1)
  filterblit        <boolean>    ..FV..... filterblit enable
(default false)
  keep_alive_timeout <int>        ..FV..... Specify a custom
session keep alive timeout in seconds. (from 1 to 100) (default
3)
```

Arguments:

w is width to scale up/down to.

h is height to scale up/down to.

force_original_aspect_ratio maintains the input video's aspect ratio when scaling.

format sets the pixel format of the output video.

force_divisible_by will force the output resolution to be divisible by the defined integer

filterblit when true enables an FIR filter, it will produce better quality but at a lower perfromance than the default modified **Bresenham** filter.

keep_alive_timeout specifies a session keep-alive timeout value. This is a periodic request/response between the libxcoder and the Quadra firmware. When this times-out, the firmware will terminate the filtering instance on the scaler. The Valid range is from 1 to 100 (inclusive).

Filter command example:

```
ffmpeg -c:v h264_ni_quadra_dec -dec 0 -xcoder-params "out=hw" -
i input1080p.264 -vf ni_quadra_scale=1280:720 -c:v
h265_ni_quadra_enc -enc -1 -xcoder-params
"RcEnable=1:bitrate=1000000" output720p.265
```

7.4 Default FFmpeg Parameters

Some FFmpeg parameters may also override explicit libxcoder parameters. Try to avoid using multiple parameters that control the same behavior.

7.4.1 Bitrate

If the FFmpeg -b argument is specified on the command line then this will set the bitrate, and will override any libxcoder bitrate parameter.

When using FFmpeg, it is recommended the FFmpeg -b:v bitrate parameter be used instead of the libxcoder parameter.

There are 4 situations for bitrate, these are

1. No bitrate is specified on the command line, therefore the default bitrate will be used. This is 200,000 – see the **RCEnable** parameter in **Section 9.3**. Note this only applies when RCEnable =1.
2. If only the FFmpeg -b command is specified then this will set the bitrate.
3. If only the libxcoder command -xcoder-params is specified then this sets the bitrate.
4. If both the FFmpeg -b and the xcoder parameters are specified then the FFmpeg parameter will take priority and set the bitrate.

7.5 New FFmpeg Parameters

The following FFmpeg parameters are designed to simplify the integration of Quadra with FFmpeg.

7.5.1 New advanced per-file options

force_nidec: This parameter forces the selection of Netint HW decoders

Supported values:

logan

quadra

The format of the input streams does not need to be specified as FFmpeg will automatically detect all the input stream formats.

Command Example:

Force select Quadra h265_ni_quadra_dec decoder

```
ffmpeg -y -hide_banner -nostdin -vsync 0 -force_nidec quadra -
xcoder-params "out=sw" -i input.265 -c:v rawvideo output.yuv
```

7.5.2 Parameters for updating PMT

The PMTs provide information on each program present in the transport stream, including the program_number, and list the elementary streams that comprise the MPEG-2 program. Quadra provides a feature to update maximum bitrate descriptors in PMT:

mpegts_max_bitrate: The value indicates an upper bound of the total bitrate, including transport overhead, that will be encountered in this program element or program.

mpegts_max_video_bitrate: The value indicates an upper bound of the video bitrate.

mpegts_max_audio_bitrate[N]: The value indicates an upper bound of the bitrate for audio track N. ***[N]*** is an audio track number and should be replaced with the supported range between 0 and 9. (i.e. mpegts_max_audio_bitrate0 needs to be used for audio track 0.)

Supported values:

Up to 1600000000 in the unit of bit per second

Command Example:

Set PMT with maximum bitrate to 32 Mbps, maximum video bitrate to 16 Mbps, maximum audio bitrate of track 0 to 64000 bps, maximum audio bitrate of track 1 to 56000 bps, and maximum audio bitrate of track 2 to 48000 bps if source.ts contains 1 video stream and 3 audio streams.

```
ffmpeg -y -hide_banner -nostdin -vsync 0 -i source.ts -map 0:v
-c:v copy -map 0:a:0 -c:a:0 copy -map 0:a:1 -c:a:1 copy -map
0:a:2 -c:a:2 copy -mpegts_max_bitrate 32000000 -
mpegts_max_video_bitrate 16000000 -mpegts_max_audio_bitrate0
64000 -mpegts_max_audio
_bitrate1 56000 -mpegts_max_audio_bitrate2 48000 -f mpegts
test.ts
```

8. Encoder

The following NETINT encoders all use Quadra hardware to perform encoding.

- h264_ni_quadra_enc
- h265_ni_quadra_enc
- av1_ni_quadra_enc
- jpeg_ni_quadra_enc

The encoder supports 8 or 10 bit YUV 420 for encoding in planar and semi-planar. The encoder also supports encoding 8 bit with a 10 bit input. Note that JPEG supports 8 bit baseline encoding only. Lossless and progressive encoding are not supported. JPEG encoding accepts only inputs with full color range

8.1 Encode Options

The following Quadra encoder options affect compression efficiency (quality) and performance

- **RDOQ** tunes every non-zero coefficient to find a better tradeoff between rate and distortion. It can therefore improve the compression efficiency (quality) but with a penalty on performance. RDOQ is enabled when **enableRdoQuant** is set to **1**.
- **RDO Level.** When the RDO level increases, the encoder selects more candidates during each mode selection stage, with an RD cost. Therefore, the overall quality will be increased but with a penalty on performance. Please refer to section "**8.4 - Encoding Parameters**" for a description of the **rdoLevel** parameter.

The following table shows the performance penalty when using RDQQ and RDO level. Please note that this table only provides a general guideline, the actual performance impact will vary depending on the encoder load (resolution, parallel jobs, etc.)

	RDO L1 as an anchor	Performance cycles ratio
h265	rdo l1	1x
	rdo l2	2x
	rdo l3	3.4x
	rdoq+rdo l1	1.5x
	rdoq+rdo l3	5x
h264	rdo l1	1x
	rdoq + rdo l1	1.7x

The following table shows RDOQ and RDO Level availability for each codec

	2-Pass	RDO Level	RDOQ
AVC	supported	Fixed level 1	Supported
HEVC	supported	1, 2, 3	Supported
AV1	supported	1, 2, 3	No Supported

- **Lookahead** improves encoding quality by 2-pass encode, the 1st pass (lookahead pass) analyzes the cost and reference dependency at a macroblock level to improve 2nd pass encode compression efficiency (quality), with a penalty on performance. Lookahead is enabled when the **lookaheadDepth** parameter is set to a value >= 4, please refer to section "**8.4 - Encoding Parameters**" for a description of lookaheadDepth parameter.

- **Multicore Joint Mode** improves the high resolution encode performance by utilizing all 4 hardware cores in parallel to encode. Please note when running multiple encoding instances in parallel, that this mode will not improve performance, since parallel instances already utilize multiple hardware cores. This affects rate control negatively because rate control will not be able to make any adjustment until 4 frames later (encoded in parallel), and therefore this may have a penalty on the compression efficiency (quality). Multicore joint mode is enabled when the **multicoreJointMode** parameter is set to 1.

- **GOP preset** specifies the GOP structure used for encoding. GOP preset is selected by setting the **gopPresetIdx** parameter, please refer to section "8.4 - Encoding Parameters" for a list of GOP preset indices. For a low delay application, the recommended GOP preset is 9 (all P-frames). For the highest quality, the recommended GOP preset is –1 (adaptive GOP, where the firmware dynamically adjusts the GOP pattern), this is also the default GOP preset setting. Please refer to section "**8.4.3 - Gop Pattern Setting**" for details of each GOP preset index. The user is also allowed to specify a customized GOP pattern by setting the GOP preset 0 (custom GOP) with a set of GOP structure syntax, please refer to section "**8.4.3.1 - Custom Gop Structure**". It is generally not required and not recommended to use a customized GOP pattern, since GOP preset patterns are usually a better choice. Custom Gop patterns should only be used if you cannot find an appropriate GOP preset for you application

- **CBR (Constant Bitrate)** rate control mode is enabled when the VBV buffer size is a non-zero value (default VBV buffer size is 3000), please refer to section "**8.4 - Encoding Parameters**" for a description of **vbvBufferSize** parameter. In CBR mode, rate control is bound by the VBV buffer constraint, and therefore the instant bitrate will be limited and more stable.
    - Please also note when cuLevelRCEnable is 1 (enable block level rate control), and lookaheadDepth is 0 (no lookahead), rate control handles user specified bitrate as maximum bitrate or average bitrate depending on bitrateMode parameter, please refer to bitrateMode parameter descriptions for details.

- **ABR (Average Bitrate)** rate control mode is enabled when VBV buffer size and vbvMaxRate are both set to 0, please refer to section "**8.4 - Encoding Parameters**" for description of **vbvBufferSize** and **vbvMaxRate** parameter. In ABR mode, rate control

maintains average bitrate to match target bitrate, but is not constrained by VBV buffer, and therefore instant bitrate may have more fluctuations compared to CBR. On the other hand, ABR may produce bitrate more closely matching the target bitrate.

- **Constrained VBR (Constrained Variable Bitrate)** rate control mode is enabled when both the VBV buffer size and VBV max rate are non-zero value (default VBV buffer size is 3000, default VBV max rate is 0 (disabled)), please refer to section "**8.4 - Encoding Parameters**" for description of **vbvBufferSize** and **vbvMaxRate** parameters. Compared to CBR mode, Constrained VBR mode allows higher instant bitrate, and therefore may produce higher quality at the cost of higher peak rate.

- **CRF (Constant Rate Factor)** mode is enabled when **crf** parameter is set to value ranging from 0 to 51, and the recommended value is 23. Please refer to section "**8.4 - Encoding Parameters**" for description of CRF parameter. Different from CBR rate control, in CRF mode the encoder varies the bitrate to maintain constant subjective quality. Therefore, it is not possible to predict the bitrate a CRF encode will come out to. CRF mode is generally used to encode video for offline file storage.

- **Capped CRF** mode is enabled when **crf** parameter is set together with **bitrate, vbvBufferSize, vbvMaxRate** (optional), **vbvMinRate** (optional) to ensure a consistent quality level, while the maximum (and minimum) bitrate cap prevents the bitrate from exceeding upper bound (and lower bound) threshold, please refer to section "**8.4 - Encoding Parameters**" for description of these parameters. Capped CRF mode incorporates CRF with maximum (and minimum) limit bitrate limit and VBV buffer constraint. Therefore, although the encoder attempts to maintain consistent subjective quality, it is also required to cap bitrate; therefore consistent quality is no longer guaranteed. On the other hand, Capped CRF brings the benefit of bitrate control on top of CRF mode. Capped CRF mode is generally used for live streaming. Please refer to Section "8.3.4 CRF & Capped CRF Descriptions and Examples" for more details.

- **HVS** improves subjective quality by adjusting the macroblock level quantization parameter, according to human visual heuristics, but usually this reduces the objective quality. HVS is enabled when **hvsQPEnable** parameter is set to 1.

- **CU Level Rate Control** is finer granularity macroblock level rate control, which may improve compression efficiency (quality). CU level rate control is enabled when **cuLevelRCEnable** parameter is set to 1.

- **Tolerance of Rate Control for Inter/ Intra** defines how much tolerance is given to CU level rate control to match target picture size, please refer to section "**8.4 - Encoding Parameters**" for description of tolCtbRcInter / tolCtbRcIntra parameters. A smaller value means less tolerance, in which case CU level rate control will make drastic adjustments at macroblock level trying to match target size. It is recommended to keep the default tolerance value.

- **Rate Control QP Delta Range** defines the range of quantization parameters CU level rate control is allowed to operate with, please refer to section "**8.4 - Encoding Parameters**" for description of rcQpDeltaRange parameter. A larger value means CU level rate is allowed to increase or decrease QP by larger delta value at macroblock level. It is recommended to keep the default QP delta range value.
- **Bitrate Window** is the window of frames in which rate control attempts to match the target bitrate, please refer to section "**8.4 - Encoding Parameters**" for a description of the bitrateWindow parameter. By default, the bitrate window is equal to the intra period, or 150 if the intra period is 0 (only encode first frame as I-frame). It is recommended to keep the default bitrate window value.
- **CTB Row QP Step** defines the maximum accumulated QP adjustment step per CTB Row allowed by CU level rate control, please refer to section "**8.4 - Encoding Parameters**" for description of ctbRowQpStep parameter. A larger value means CU level rate is allowed to increase or decrease QP by larger delta value per MB / CTB row. It is recommended to keep the default QP step value.

8.2 Block Level adjustment for Subjective Quality and/or Rate Control

Subjective video quality can be enhanced by tuning QP at the block level, this can be enabled/disabled by the **hvsQPEnable** parameter. The block is the CTB (coding tree block) for H.265, MB (macroblock) for H.264 and the superblock for AV1. Block level rate control can also be enabled/disabled by the **cuLevelRCEnable** parameter. Applications can enable both parameters to enable subjective video quality adjustment, and also the block level rate control, at the same time.

hvsQPEnable = 0 & **cuLevelRCEnable** = 0: Default settings. Disable CTB QP adjustment.

hvsQPEnable = 1: Block level QP adjustment for Subjective Quality only

HVS QP will adjust the QP according to the block complexity of the input image. To follow the sensitivity of the Human Eye system for high complex blocks, the QP will have a small increase; for flat content blocks, the QP will have a small decrease. The method may cause a small decrease in objective quality, or much deviation on the final encoded bit rate.

cuLevelRCEnable = 1: Block Level QP adjustment for Stable Bit Rate Control only

In this mode, the rate control will try to control current frame bits within the target range by adjusting the block QP. This mode will have a more stable bit rate.

hvsQPEnable = 1 & **cuLevelRCEnable** = 1: Block level QP adjustment for both Subjective Quality and Bit Rate Control

8.3 Objective Quality vs Performance

There is a trade-off between Objective Quality and Encoder Performance. Using the **rdoLevel** and **enableRdoQuant** parameters this trade-off can be adjusted.

The **rdoLevel** specifies the number of candidates to use for Rate Distortion Optimization. RDO is a method for improving video quality during compression. Lower RDO values will generate lower quality, but performance is improved. Higher RDO values will generate higher quality, but again, this will cause lower performance. Similarly, **enableRdoQuant** specifies whether to enable or disable the RDO Quantization.

8.4 Encoding Parameters

Syntax and Conformance

level

Sets the level for encoding. If level=0 the encoder will automatically determine the level based on picture size, frame rate and bitrate. If specified, the level will be used.

When a non-zero level is specified, the encoder will use it regardless of the encoder parameters.

Not Applicable: JPEG

Supported Values:

Decimal values from 0 to 9.9 in 0.1 increments

H.264 levels: 1, 1.1, 1.2, 1.3,
2, 2.1, 2.2,
3, 3.1, 3.2,
4, 4.1, 4.2,
5, 5.1, 5.2,
6, 6.1, 6.2

H.265 levels: 1, 2, 2.1,
3, 3.1,
4, 4.1,
5, 5.1, 5.2,
6, 6.1, 6.2

AV1 levels: 2.0, 2.1,
3.0, 3.1,
4.0, 4.1,
5.0, 5.1

Default: 0

profile

Sets the profile for encoding. The valid profiles for H.264 and H.265 are shown below.

Any profile can be used for 8 bit encoding, but only the 10 bit profiles (main10 for H.265 and high10 for h.264) may be used for 10 bit encoding.

NOTE – For the H.264 baseline, the GOP must not contain any B frames, therefore the only supported values for **gopPresetIdx** are 1, 9, 10 or 0 (custom GOP with **picType** != 3)

Not Applicable: JPEG

Supported Values:

H.265:
1=main (8 bit default)
2= main10 (10 bit default)

H.264:
1=baseline (not compatible with B frames)
2=main
4=high (8 bit default)
5= high10 (10 bit default)

AV1:
1=main (8 bit and 10 bit default)

Default:

H.265:
1=main (8 bit default)
2= main10 (10 bit default)

H.264:
4=high (8 bit default)
5= high10 (10 bit default)

AV1:
1=main (8 bit and 10 bit default)

high-tier

Sets the tier for AV1 and H.265 encoding.

NOTE – High tier only takes effect if the current level supports it.

Not Applicable: JPEG and H.264

Supported Values:

0: Main tier
1: High tier

Default:

0 for H.265
1 for AV1

hrdEnable

Enable hypothetical reference decoder (HRD) compliance.

When enabled this parameter will:

1. Update the HRD parameters in the VUI
2. Send the pic timing sei messages in the bitstream
3. Send the buffering period sei messages in the bitstream
4. Automatically set **RcEnable** = 1 if **RcEnable** is not already enabled

NOTE – **hrdEnable** is automatically enabled when **dolbyVisionProfile = 5**

NOTE – Please note that encoder cannot guarantee HRD conformance when **multicoreJointMode** is set to 1, so it is recommended not to enable multicoreJointMode when strict adherence to HRD conformance is required.

Not Applicable: JPEG and AV1

Supported Values:

0: Disable
1: Enable

Default:

0: Disable

enableAUD

Specifies whether to include Access Unit Delimiters (AUD) or not.

Required For:

AUDs are required for **DolbyVision** compatibility, or when placing bitstreams in transport streams.

Not Applicable: JPEG and AV1

Supported Values:

0: Disable
1: Enable

Default:
0: Disable

dolbyVisionProfile

Configures the Netint encoder as a 3rd party encoder with the **Dolby Encoding Engine**. Overrides VUI colour parameters as per **DolbyVision** profile 5 requirements (**video_format**=1 (full-range), **colour_primaries**=**transfer_characteristics**=**matrix_coeffs**=2 (unspecified), **chroma_loc_info_present_flag**=0) and enables AUD and HRD.

NOTE – When **dolbyVisionProfile** is enabled:

1. Automatically set **RcEnable** = 1
2. Automatically set **hrdEnable** = 1
3. Automatically set default **vbvBufferSize** = 3000
4. Automatically set **enableAUD** = 1

Applicable:

H.265 only
FFmpeg version 4.3.1 and above

Not Applicable:

JPEG, AV1 and H.264
FFmpeg version 4.2.1 and earlier

Supported Values:

0 or 5

Default:

0: Disabled

maxCLL

Specifies parameters for HDR Content Light Level Info as per CEA 861.3. Specified as a string with format “%hu,%hu” where %hu are unsigned 16 bit integers. The first value is the maximum content light level (or 0 if no maximum is indicated), the second value is the maximum picture average light level (or 0). For example, for **MaxCLL**=1000 nits, **MaxFALL**=400 nits: max-cll =“1000,400”. Note that this string value will need to be escaped or quoted to protect against shell expansion on many platforms. When this parameter is specified, it will be used in preference to content light level info side data of FFmpeg AVFrame.

Not Applicable: JPEG and AV1
Default: None – Use side data values if present

masterDisplay

Specifies parameters for the HDR Mastering Display Colour Volume SEI as per SMPTE ST 2086. Specified as a string with format:
“G(%hu,%hu)B(%hu,%hu)R(%hu,%hu)WP(%hu,%hu)L(%u,%u)” where %hu are unsigned 16bit integers and %u are unsigned 32bit integers. The SEI includes X,Y display primaries for RGB channels and white point (WP) in units of 0.00002 and max,min luminance (L) values in units of 0.0001 nits
For example for a P3D65 1000-nits monitor, where G(x=0.265, y=0.690), B(x=0.150, y=0.060), R(x=0.680, y=0.320), WP(x=0.3127, y=0.3290), L(max=1000, min=0.0001):
“G(13250,34500)B(7500,3000)R(34000,16000)WP(15635,16450)L(10000000,1)”
Note that this string value will need to be escaped or quoted to protect against shell expansion on many platforms. When this parameter is specified, it will be used in preference to mastering display info side data of FFmpeg AVFrame.

Not Applicable: JPEG and AV1
Default: None – Use side data values if present

enableAllSeiPassthru

All custom SEI types will be passed through if this is enabled. Also, when enabled, the firmware SEI will be disabled.

Note – If the **enableAllSeiPassthru** parameter is enabled (set to 1) for decoding, then the **enableAllSeiPassthru** parameter for encoding must also be enabled (set to 1).

Supported Values:

0: Disable
1: Enable

Default: 0

useLowDelayPocType

When enabled, the encoder will use **picture_order_count_type**=2 in the H.264 SPS, this tells the decoders all frames are in sequence, this typically results in a lower decoding delay. This feature is only supported for H.264 when all frames are reference frames, i.e, when using gop presets gopPresetIdx=1, 3, 7, and 9. For custom gop (gopPresetIdx 0), this feature is only supported for custom gop size 1. By default, this feature is disabled, and the encoder uses **picture_order_count_type**=0, which is compatible with all gops.

Not Applicable: JPEG, H265, and AV1

Supported Values:

0: Disable
1: Enable

Default : 0: Disabled

confWinTop

Conformance top window size. This is the number of pixel rows at the top of the picture that should not be displayed when decoding. This is in addition to any cropping info that may already be included on the AVFrame to be encoded.

Not Applicable: AV1 and JPEG
Supported Values: 0 to 8192
Default : 0

confWinBot

Conformance bottom window size. This is the number of pixel rows at the bottom of the picture that should not be displayed when decoding. This is in addition to any cropping info that may already be included on the AVFrame to be encoded.

Not Applicable: AV1 and JPEG
Supported Values: 0 to 8192
Default: 0

confWinLeft

Conformance left window size. This is the number of pixel columns at the left side of the picture that should not be displayed when decoding. This is in addition to any cropping info that may already be included on the AVFrame to be encoded

Not Applicable: AV1 and JPEG
Supported Values: 0 to 8192
Default : 0

confWinRight

Conformance right window size. This is the number of pixel columns at the right side of the picture that should not be displayed when decoding. This is in addition to any cropping info that may already be included on the AVFrame to be encoded

Not Applicable: AV1 and JPEG
Supported Values: 0 to 8192
Default : 0

repeatHeaders

Specifies whether the encoder repeats the VPS/SPS/PPS headers on all I frames. Or if intra refresh is enabled, headers are repeated on P frames at IntraRefreshMinPeriod.

For HDR/HDR10+ streams, the static HDR SEIs (content light level info, mastering display colour volume, and alternative transfer characterics) if present are also repeated. Repeated headers permit a bitstream to be decoded mid-stream. It also provides error resilience, in case packets were lost during transmission.

Not Applicable: JPEG and AV1
Supported Values:
0: disable
1: enable
Default: 1

prefTRC

Specifies the HLG preferred transfer characteristics value. If this parameter is present, the encoder will include an alternative transfer characteristics metadata in the bitstream with the preferred transfer characteristics field set to the value of this parameter. If the parameter is not present, the SEI will not be present. The alternative transfer characteristics metadata is required by ETSI for HLG and specifies an alternative transfer characteristic from that provided in the VUI.

Not Applicable: JPEG
Supported Values: 0 to 255
Default : Alternative Transfer Charactersictics metadata is not present

sliceMode

Enable multi-slice encoding. Must be used in conjunction with parameter **sliceArg**.

Not Applicable: JPEG and AV1
Supported Values:
0: single slice per frame
1: multiple slices per frame
Default: 0

sliceArg

If **sliceMode** is enabled, this represents the number of CTU/MB rows in each slice. Value must be between 1 and number of CTU/MB rows in the picture. The number of rows is calculated by height/64 (H.265) or by height/16 (H.264).

NOTE – sliceNum is number of slices in one frame.
sliceNum = (ctbPerCol + sliceArg - 1) / sliceArg.
need to keep sliceNum <= 256, if sliceNum > 256, sliceArg will increase one automatically until sliceNum <= 256

Not Applicable: JPEG and AV1
Supported Values: 1 to minimum value between number of CTU/MB rows and 127
Default: 0 (invalid)

entropyCodingMode

Selects the entropy coding mode used in the encoding process. Note that CABAC is only compatible with H.264 Main, High, and High10 profiles and is disabled for other profiles.

Not Applicable: JPEG, AV1, and H.265
Supported Values:
0: CAVLC
1: CABAC
Default: 1

frameRate

The numerator of the frame rate. This works in conjunction with **frameRateDenom** to support fractional framerates. The **frameRate** is used by the encoder for rate control (when enabled) and to set the VUI timing information. If not specified, the **frameRate** passed in from FFmpeg (or 3rd party application) is used. This parameter is intended for integrating directly with libxcoder.

Supported Values: 1 to 2^{16}-1
Default: FFmpeg Value

frameRateDenom

The encoder frame rate denominator that supports fraction frame rate together with **frameRate**. The frame rate would then be **frameRate** / **frameRateDenom**, e.g. **frameRate**=30000 and **frameRateDenom**=1001 represents frame rate of 30000/1001=29.97. This parameter is intended for integrating directly with libxcoder.

Supported Values: 1 to 2^{16}-1
Default: 1

colorPri

Specifies one of the VUI color description parameters: color_primaries. The supported values are defined as AVColorPrimaries in FFmpeg, and ni_color_primaries_t in libxcoder. If colorPri, colorTrc, or colorSpc is not equal to 2 (unspecified), the relevant VUI parameters are added to the sequence header.

Not Applicable: JPEG
Supported Values: 0-12, 22
Default: FFmpeg value

colorTrc

Specifies one of the VUI color description parameters: transfer_characteristics. The supported values are defined as AVColorTransferCharacteristic in FFmpeg, and ni_color_transfer_characteristic_t in libxcoder. If colorPri, colorTrc, or colorSpc is not equal to 2 (unspecified), the relevant VUI parameters are added to the sequence header.

Not Applicable: JPEG
Supported Values: 0-18
Default: FFmpeg value

colorSpc

Specifies one of the VUI color description parameters: matrix_coeffs. The supported values are defined as AVColorSpace in FFmpeg, and ni_color_space_t in libxcoder. If colorPri, colorTrc, or colorSpc is not equal to 2 (unspecified), the relevant VUI parameters are added to the sequence header.

Not Applicable: JPEG
Supported Values: 0-14
Default: FFmpeg value

sarNum, sarDenom

Specifies the VUI sample aspect ratio aspect_ratio_idc, as sarNum : sarDenom. When sarNum is 0, aspect_ratio_idc is not included in VUI. When sarNum is greater than 0, the aspect_ratio_idc is included in VUI. This parameter is intended for integrating directly with libxcoder.

NOTE – A user must use the matching sarNum and sarDenom to the actual picture. Otherwise, there will be a discrepancy between the value in the VUI and the actual.

Not Applicable: JPEG
Supported Values: 0 to max integer
Default: FFmpeg value

videoFullRangeFlag

Specifies the VUI video_full_range_flag parameter value. When it is 1, the relevant VUI parameters are added to the sequence header.

NOTE – A user must use the matching value to the actual picture. Otherwise, there will be a discrepancy between the value in the VUI and the actual.

Not Applicable: JPEG
Supported Values: 0 or 1
Default: FFmpeg value

av1ErrorResilientMode

Specifies whether AV1 error resilient mode is enabled. AV1 error resilient mode allows the syntax of a frame to be parsed independently of previously decoded frames. Note that enabling AV1 error resilient mode may reduce compression efficiency. Also note that error resilient mode is only applicable to AV1, enabling this mode has no effect on other codecs.

NOTE – Please note that enabling AV1 error resilient mode may reduce compression efficiency. Our test results showed that enabling AV1 error resilient mode can cause up to more than 5% of BD rate loss, and the average BD rate loss is around ~2.42%

Not Applicable: JPEG, H.264, and H.265
Supported Values:
0: Disable
1: Enable
Default: 0: Disabled

temporalLayersEnable

Enables temporal scalability with SVC (Scalable Video Coding). When enabled, temporal layers related syntax is added to bitstream. For temporal IDs assignments for each preset gop pattern, please refer to section *8.4.3.2 - Pre-defined GOP Structure*.

NOTE – For custom gop pattern (gopPresetIdx 0), SVC temproal layers syntax is added to bitstream when temporal IDs specified in Custom Gop Parameters (xcoder-gop) are greater than 0, regardless of temporalLayersEnable. Please refer to section *8.4.3.1 - Custom Gop Structure* for information regarding Custom Gop Parameters

Not Applicable: JPEG
Supported Values:
0: Disable
1: Enable
Default: 0: Disabled

ppsInitQp

Specifies h.264 pic_init_qp_minus26 or h.265 init_qp_minus26 in Picture Parameter Sets. When specified, every Picture Parameter Set in the h.264 or h.265 bitstream will signal the specified ppsInitQp value for init_qp_minus26 syntax - instead of setting init_qp_minus26 according to the initial QP value decided by firmware, which is the default behavior when user does not specify ppsInitQP.

NOTE – When ppsInitQp parameter is not specified, firmware sets PPS init_qp_minus26 according to the initial QP, which is automatically decided by the firmware

NOTE – ppsInitQp parameter is not applicable to AV1 and JPEG

Not Applicable: AV1 and JPEG
Supported Values: 0 to 51
Default: FW automatically decided initial QP value

spatialLayers

Specifies the number of spatial layers in AV1 encoding. Setting spatialLayers > 1 enables AV1 spatial scalability with multiple spatial layers. Quadra supports up to 4 spatial layers.

NOTE – Please note that Quadra does not support spatial layers with different resolutions, all spatial layers must have the same resolution. Also, Quadra expects input frames to be in spatial layers order, E.g. first input frame is in Baser Layer, second input frame is in Enhancement Layer 1, third input frame is in Enhancement Layer 2, and so on

NOTE – Please note that when AV1 spatial scalability is enabled (by setting spatialLayers > 1), cross layers referencing is enabled, and the reference hierarchy or gop pattern is no longer decided by gopPresetIdx. Specifying gopPresetIdx with multiple spatialLayers is prohibited. By default, each EL (Enhancement Layer) frame refers to the frame one layer lower. If **spatialLayersRefBaseLayer** is set to 1, each EL frame will refer to the BL (Base Layer) frame instead of the lower layer frame. Please refer to section “8.4.3.4 - AV1 Spatial Layers Gop Patterns” for illustrations of cross layer reference hierarchies when AV1 spatial scalability is enabled.

NOTE – Please note that intraPeriod must be divisible to the number of spatial layers. For example, the default intraPeriod 120 is compatible to spatialLayers 2, 3, and 4, while intraPeriod 100 would not be compatible to spatialLayers 3. Setting intraPeriod 0 (disable periodical I-frame insertion) is also compatible with all spatialLayers settings

NOTE – Please note that lookahead is not supported when multiple spatial layers

NOTE – Please note that spatial layers is only supported for AV1

Not Applicable: H.264, H.265, JPEG
Supported Values:
1 ~ 4: Support up 4 spatial layers. Set 1 to disable AV1 spatial scalability, > 1 to enable AV1 spatial scalability
Default: 1: AV1 spatial scalability disabled

spatialLayersRefBaseLayer

When spatialLayersRefBaseLayer is enabled, each EL (Enhancement Layer) frame will refer to the BL (Base Layer) frame instead of the lower layer frame. Please refer to section “8.4.3.4 - AV1 Spatial Layers Gop Patterns” for illustrations of cross layer reference hierarchies when spatialLayersRefBaseLayer is enabled.

NOTE – spatialLayersRefBaseLayer only takes effects when **spatialLayers** is greater than 2

Not Applicable: H.264, H.265, JPEG
Supported Values:
0: Disable. When disabled, each Enhancement Layer frame references the lower layer frame
1: Enable. When enabled, each Enhancement Layer frame references the Base layer frame
Default: 0: Disabled

enableTimecode

When enabled, set the pic_struct_present flag in VUI to 1. This is needed for properly inserting timecode data into the picture timing SEI in H264. See section 6.6.7.6 in the libxcoder API integration guide for more details on how to insert timecode data into SEI.

NOTE – This option should only be enabled when the ni_enc_insert_timecode function (from ni_av_codec.h in libxcoder) is called to insert timecode data in SEI for each frame. If only enabling this option, the pic_struct_present flag in VUI will be set to 1 but there won’t be any pic_timing SEI added. On the other hand, if only ni_enc_insert_timecode function is called, pic_timing SEI will be added but the pic_struct_present flag in VUI will remain 0, which will cause the timecode data in pic_timing SEI to be ignored.

NOTE – This option is only valid for H264. It does not have any effect when using other codecs.

Not Applicable: H265, AV1 and JPEG
Supported Values:
0: Disable
1: Enable
Default: 0: Disabled

avccHvcc

When this flag is disabled, the encoder produces a byte stream where NALUs are sent sequentially, as specified by Annex B. Each NALU is preceded by a Start Code Prefix, which is not allowed to appear within the NALU itself.

When it is enabled, the firmware encoder generates the bitstream in AVCC/HVCC. The encoder generates [Codec]DecoderConfigurationBox followed by NALUs, where [Codec] can be either ‘AVC’ or ‘HEVC’. Refer to the table below for the syntax details. Each NALU, following this configuration box, is preceded by its length rather than a Start Code Prefix.

Field of Box	Description	Length
size	Length of the whole box including ‘size’ field	4 bytes
type	‘avcC’ or ‘hvcC’	4 bytes
[Codec]DecoderConfigurationRecord	Refer to ISO 14496-15	variable

NOTE – The primary purpose of this option is to facilitate the construction of bitstream within a container, such as MP4, without the need to convert from AnnexB to AVCC/HVCC. It is important to note that the generated bitstream is not final and requires further processing into a container.

NOTE – This option is applicable only for generating elementary stream formats of H.264 and H.265. When using an FFMPEG command-line operation, it is essential to use 'h264' or 'h265' and avoid using container file extensions in the output file name.

NOTE – This option is not supported on FFmpeg 7.1.

Not Applicable: AV1 and JPEG
Supported Values:
0: Disable
1: Enable
Default: 0: Disabled

Libxcoder Application Features

lowDelay

Specifies whether or not to enable the low latency mode in encoding. When enabled, libxcoder increases its rate of polling the encoder and only permits buffering of a single frame to minimize the delay.

NOTE – That when enabled, the **gopPresetIdx** must be 1, 3, 7, 9, 10, or 0 with a consecutive order gop pattern, **lookaheadDepth** must be 0, and **multicoreJointMode** must be 0.

NOTE – That in libxcoder encoder send/receive multi-thread mode, when enabled, its value can be a positive integer value in milliseconds for threads synchronization. It represents the time the sending thread waits before deciding it's in a deadlock and has to continue without waiting for receiving thread to signal.

Not Applicable: JPEG
Supported Values:
0: disable
Positive integer: enable
Default: 0

minFramesDelay

Specifies whether to enable the minimum encoding delay frames feature. When enabled, libxcoder increases its rate of polling the encoder and only permits buffering of the minimum frames to minimize the delay.

NOTE – The minimum encoding delay frames feature supports all gopPresetIdx values while also supporting lookaheadDepth. The number of minimum encoding frames depends on gopsize and lookaheadDepth. For example, when gopPresetIdx is 5 and lookaheadDepth is 4, the number of minimum encoding delay frames is 7, which is obtained by adding gopsize and lookahead Depth, then subtracting one.

Not Applicable: JPEG
Supported Values:
0: disable
1: enable
Default: 0

keepAliveTimeout

Specifies a session keep alive timeout value. This is a periodic request/response between libxcoder and XCoder firmware that when timed out, the session instance will be terminated by XCoder firmware. If this option is used in conjunction with FFmpeg command line option keep_alive_timeout then keepAliveTimeout overrides keep_alive_timeout.

Supported Values: Integer in the range 1 to 100
Default: 3

zeroCopyMode

Enable or disable libxcoder zero copy feature. When zero copy is disabled, libxcoder copies YUV data from input frame buffer(s) to consecutive internal buffer before transferring data to device. This reduces overhead by triggering data transfer only once. When zero copy is enabled, libxcoder transfers data directly from input frame buffer(s) (luma / chroma buffers can be inconsecutive) to device, which improves performance when encoding high resolution / large frames.
The default setting is auto mode, in which libxcoder decides whether to enable or disable zero copy based on input resolution – zero copy is enabled if input resolution >= 1080p, and disabled if input resolution < 1080p. Auto mode's zero copy enable / disable is based on performance test results, and therefore is the suggested setting.

NOTE – For applications based on libxcoder instead of calling FFMpeg libavcodec functions, please refer to libxcoder_API_IntegrationGuide section 6.6.2 - “Input YUV Frame Preparation” regarding the libxcoder APIs required to use the zero copy feature. Otherwise zero copy will not take effect regardless of the zeroCopyMode setting.

NOTE – Zero copy only applies to input frames which meet the following requirements

- YUV420 8 or 10 bit planar/semi-planar pixel format, or RGBA / BGRA / ABGR / ARGB
- Input frame width and height are 2-pixels aligned
- Input frame linesizes (aka strides) of luma / chroma buffers are 2-bytes aligned
- Input frame resolution >= 144x128
- Each input frame’s linesize shall be the same throughout the entire encoding session

Supported Values:

0: Disable zero copy (For any input resolution)
1: Enable zero copy (For any input resolution)
-1: Auto mode (Enable zero copy if input resolution >= 1080p,
disable zero copy if input resolution < 1080p)

Default: 1 (Enable zero copy for all input resolutions)

ddrPriorityMode

Specifies the DDR memory priority mode. Only need set once at beginning, and it will reset to default automatic after current process finish.

NOTE – This is a global setting, it will influence all running processes. It is best to only use it when there is only one process. If there are multiple processes, other processes fps performance may influence by this parameter.

Supported Values:

0: set default ddr mode

1:increase ddr priority for decoder and encoder

2:increase ddr priority for scaler

3:increase ddr priority for ai

Default: -1

statisticOutputLevel

Specify the information output for each frame, including the maximum and minimum MV of the macroblocks in the current frame, as well as the number of inter/intra macroblocks in the current frame and the frame size of the current frame.

NOTE – This feature will decrease the performance because collecting this information is time-consuming.

Supported Values:

0: disable
1:enable

Default : 0

disableAdaptiveBuffers

Specifies whether to disable adaptive buffers when bitstream sequence is changed. When **disableAdaptiveBuffers**=1 and width/height of pictures is different from previous width/height in sequence change, it will take a little time to re-configure encoder with new width/height.

Note – If this option is used and the output type of decoder is **hw** in transcoding , the **disableAdaptiveBuffers** parameter of quadra decoder should be used at same time. If there are many sequence changes in customer's bitstream, suggest to enable this parameter.
Not Applicable: JPEG or VP9.
Supported Values:
0: Disable
1: enable
Default: 0: Disable

Encode Algorithm and Features

rdoLevel

Specifies the number of candidates to use for Rate Distortion Optimization. RDO is a method for improving video quality during compression. Lower values mean a lower quality but better performance, higher values mean more quality but less performance

Not Applicable: JPEG
Supported Values:
H.264: 1
H.265: 1 to 3
AV1: 1 to 3
Default: 1

EnableRdoQuant

Specifies whether to enable or disable RDO Quantization. RDOQ provides optimized quantization to further improve video quality, with the cost of lower performance

Applicable: H.264 and H.265
Not Applicable: AV1 and JPEG
Supported Values:
0: Disable
1: Enable
Default: 0

tuneBframeVisual

Specifies whether to enable or disable B-frame visual tuning. B-frame visual tuning helps mitigate subjective quality fluctuation, at the cost of lowered compression efficiency. When enabled, there are 2 modes available – setting tuneBframeVisual to 1 enables MEDIUM tuning mode, in which encoder tunes B-frame visual quality with minor loss in compression efficiency, and setting tuneBframeVisual to 2 enables HIGH tuning mode, in which encoder applies stronger tuning to B-frame visual quality with more compression efficiency loss

NOTE – Please also note that MEDIUM tuning mode is only applicable to H.264 and H.265, and is only available when lookahead (enabled by **lookaheadDepth**) or CRF (enabled by **crf**) encode is enabled

Not Applicable: JPEG (MEDIUM tuning mode is not applicable to AV1)
Supported Values:

0: Disable
1: MEDIUM (tune B-frame visual quality with minor loss in compression efficiency)
2: HIGH (stronger B-frame quality tuning with more compression efficiency loss)

Default: 0

enable2PassGop

This parameter will affect the group of picture pattern for 2-pass encoding.
By default, it is disabled. Encoder has its own GOP for 2-pass mode when lookahead > 0. GOP will be different from what is described in **gopPresetIdx**.
When it is enabled, the 2-pass encoding's GOP is consistent with GOP specified by **gopPresetIdx**.
For more details, please refer to **Section 8.4.3.2 Pre-defined GOP Structure**.
Not Applicable:JPEG
Supported Values:

0: Disable
1: Enable

Default: 0

lookAheadDepth

Number of frames to lookahead while encoding. Lookahead can increase encoder quality at the expense of performance (approximately 25%) and delay. Note that the input video pixel width must be at least 288 pixels wide.

Not Applicable: JPEG
Supported Values

0: Disable lookahead
4 to 40: Enable lookahead with lookahead depth in frames (4 to 40 frames).

Note: Due to resource constraints, when transcoding a 8k(7680x4320) 10 bitstream, the maximum value of lookahead is 16.

Default: 0

gopPresetIdx

Defines the **G**roup **O**f **P**icture pattern.

By default, encoder uses Adaptive Gop, for which the encoder dynamically adjusts gop pattern while encoding based on statistics collected from previous frames (1-pass) or lookahead (2-pass).

If both coding parameters **gopPresetIdx** and **lookaheadDepth** are set, then the gop pattern is different from setting only the coding parameter **gopPresetIdx**. For details, please refer to **Section 8.4.3 Custom Gop Structure**.

For 2-pass encoding, the GOP is not only affected by this parameter, but also affected by parameter **enable2PassGop**. Please refer to it for more details.

Not Applicable: JPEG
Supported Values:

-1: Adaptive Gop (default)
0 : Custom Gop
1 : I-I-I-I,..I (all intra, gop_size=1)
3 : I-B-B-B,…B (consecutive B, gop_size=1)
4 : I-B-P-B-P,… (gop_size=2)
5 : I-B-B-B-P,… (gop_size=4)
7 : I-B-B-B-B,… (consecutive B, gop_size=4)
8 : I-B-B-B-B-B-B-B-B,… (random access, gop_size=8)
9 : I-P-P-P,… P (consecutive P, gop_size=1)
10 : I-P-P-P-P,… (hierarchical P, gop_size=4)

Default: -1 (Adaptive Gop)

intraPeriod

Key frame / IDR frame interval.

Supported Values:
0 to 1024
0 implies an infinite period (disables periodic IDR frames)

Default : 120

intraQP

Specifies intra frame quantization parameter.
NOTE – The intraQP only takes effect when rate control is disabled e.g. **rcEnable** = 0

When used for JPEG, the encoder scales the quantization table in the JPEG standard to produce bitstreams of different video quality.

Supported Values: 0 to 51

Default : 22

intraQPDelta

Delta value added to the Intra frame QP. Can be used to lower the Intra Picture encoded size (higher QP) or to increase Intra quality relative to the Inter Pictures (lower QP) to get rid of intra flashing. Recommended value range is –12 to 12.

NOTE – The intraQpDelta only takes effect when rate control is enabled, eg. rcEnable = 1

After intraQpDelta is applied, the intra frame QP is further adjusted by rate control, which means that the encoded bitstream may not contain the exact value of the user-defined delta depending on the source.

Supported Values: -51 to 51

Default : -2

qLevel

JPEG only. Specifies the quantization scale to produce the different video qualities by scaling the quantization table. Higher values produce better quality.

Not Applicable: H264, H265, and AV1

Supported Values: 0 to 9

minQP

Min QP for rate control.

Supported Values: 0 to 51

Default : 8

maxQP

Max QP for rate control.

Supported Values: 0 to 51

Default : 51

roiEnable

ROI (Region of interest), is a feature of the encoder that permits the quality of some regions to be improved at the expense of other regions. If rate control is disabled, the QPs are used directly for encoding, if rate control is enabled, the encoder scales the QPs as necessary to meet the bitrate target. When ROI is enabled, the ROI map can be updated, enabled, or disabled on a frame by frame basis.

For ROI map configuration, please refer to ni_quadra_roi in section **10 – Filters**.

NOTE – The ROI has no effect for 2-pass (lookaheadDepth > 0) encode

Not Applicable: JPEG

Supported Values:

0: disable
1: enable

Default : 0

RoiDemoMode

Enables the ROI demo mode. When ROI is enabled (**roiEnable**=1), **ROIDemoMode** permits the ROI feature to be demonstrated using the standard FFmpeg command line, without additional application development. ROI demo mode is currently only supported on FFmpeg 3.4.2 and above. Demo mode 1 has QP=40 for the center 1/3, and QP=10 for outer 2/3. Demo mode 2 has opposite QP. In both cases ROI is enabled at frame 90 and disabled at frame 300.
NOTE – ROI takes no effects for 2-pass (lookaheadDepth > 0) encode

Applicable: FFmpeg 3.4.2 and above
Not Applicable: JPEG
Supported Values: 0 to 2
Default: 0: disabled

cacheRoi

Enables caching of an ROI map. When ROI is enabled, the ROI map can be changed on a frame by frame basis. When **cacheRoi** is enabled at the same time, the currently available ROI map is cached and applied to the subsequent frames until a new map is supplied. It is only valid if ROI is enabled.
NOTE – ROI takes no effects for 2-pass (lookaheadDepth > 0) encode

Not Applicable: JPEG

Supported Values:
0: disable
1: enable

Default: 0

intraRefreshMode

Selects intra refresh macroblock update method. This method is not supported for 2-pass encode and the command will be rejected.

Not Applicable: JPEG

Supported Values:
0: no intra refresh
1: row – works in conjunction with next parameter

Default: 0

intraRefreshArg

Specifies the number of consecutive CTB or MB rows refreshed (to be encoded as Intra) per frame

NOTE – Only takes effects if intraRefreshMode = 1

NOTE – For h.264, each MB row is 16 pixels or more in height

For h.265 / AV1, each CTU row is 64 pixels or more in height

Please note if intra refresh cycle is 1 (meaning intra refresh must be completed in 1 frame), this defeats the purpose of intra refresh and therefore the encode command will be rejected. Intra refresh cycle can be calculated by

Intra refresh cycle = input picture height / (intraRefreshArg * 16 for h.264, or * 64 for h.265 and AV1)

For example, if h.265 input resolution is 640x480 and intraRefreshArg is 8, intra refresh cycle = ceil(480 / (64 * 8)) = 1, which is invalid setting and will be rejected

Not Applicable: JPEG

Supported Values:

0: intra refresh is disabled
Nonzero: number of consecutive CTB / MB rows to be encoded as Intra

Default: 0

IntraRefreshMinPeriod

Intra refresh cycle starts at intra period, which is specified by **intraPeriod** or **intraRefreshMinPeriod**. If both **intraRefreshMinPeriod** and **intraPeriod** are specified, **intreaRereshMinPeriod** has higher priority.

Not Applicable: JPEG

Default: 120

IntraRefreshResetOnForceIDR

Intra refresh cycle will reset to restart on force IDR frame

Note this only takes effects if intraRefreshMode = 1

Not Applicable: JPEG
Supported Values:
0: intra refresh on force IDR frame is disabled
1: intra refresh on force IDR frame is enabled
Default: 0

longTermReferenceEnable

Enables the long term reference (LTR) feature. With long term reference enabled, an application can

1. Set LTR interval by longTermReferenceInterval, and/or
2. Set any frame as LTR on a frame by frame basis. *For more detail refer to QUADRA libxcoder API Guide*
3. Set number of LTRs (1 or 2) by longTermReferenceCount

LTR is only supported by low delay gop (i.e. all frames in sequence), namely gopPresetIdx=1, 3, 7, 9, or 0 (if custom gop has consecutive order frames). Also, LTR is not supported for 2-pass encode and the command will be rejected.

NOTE – When the longTermReferenceEnable is enabled Quadra preserves up to 2 LTRs. The number of LTRs (1 or 2) can be configured by longTermReferenceCount. The application does not need to specify the LTR index as Quadra maintains the LTR index internally.

NOTE – The occurrence of an IDR frame will clear any existing LTRs (referencing frames prior to an IDR is not allowed in the standards).

Not Applicable: JPEG
Supported Values:
0: disable
1: enable
Default: 0

longTermReferenceInterval

When longTermReferenceEnable is enabled, assign longTermReferenceInterval > 0 will set frames as LTR periodically upon specified frame interval. If longTermReferenceInterval = 0, only IDR frames will be set as LTR

Note the longTermReferenceInterval only takes effect if longTermReferenceEnable = 1

Not Applicable: JPEG

Default: 0

longTermReferenceCount

When longTermReferenceEnable is enabled, the number of LTRs (1 or 2) can be configured by longTermReferenceCount.

By default, Quadra preserves up to 2 LTRs. However, due to Quadra h.264 encoder limitation, some decoders may not be able to decode h.264 bitstream Quadra encoded with 2 LTRs, in which case longTermReferenceInterCount should be set to 1

Not Applicable: JPEG

Supported Values: 1 to 2

Default: 2

multicoreJointMode

Enables encoder multi-core mode where all 4 cores work together in parallel (a.k.a. joint mode). When disabled (default) the encoder instances use a single video core to encode. When enabled, encoder instances use all 4 video encoding cores in parallel. Recommended only for high resolution encoding with less than 4 instances e.g. 8K encode. When more than 4 encoding instances are used enabling this feature will lower performance due to extra synchronization overhead.

When multicoreJointMode is enabled, lowDelay and picSkip must be 0.

NOTE – Please note that encoder cannot guarantee HRD conformance when **multicoreJointMode** is set to 1, so it is recommended not to enable multicoreJointMode when strict adherence to HRD conformance is required.

Not Applicable: JPEG
Supported Values:
0: Disable
1: Enable

Default: 0: Disable

enableSSIM

When enabled, XCoder firmware will return SSIM values for Y, U, and V of each encoded frame in the ni_metadata_enc_bstream_t structure. The SSIM values are 4 decimal places multiplied by 10000. Divide by 10000 to get the original value.

NOTE – SSIM measures video quality by calculating Structural Similarity. Range 0 to 1.

NOTE – By default, SSIM values are logged in debug mode. To change this to info level at compile time use `-m, --with-info-level-ssim-log` when compiling libxcoder with `libxcoder/build.sh` script.

Not Applicable: JPEG and AV1

Supported Values:
0: Disable
1: Enable

Default : 0

getPsnrMode

Specify whether to output PSNR. Hardware only support output psnr on y component for HEVC codec. For other codec or for UV component, the psnr should calculate on the SW. Enable this feature will reduce performance.

NOTE – Currently PSNR is supported for 8-bit YUV420p only, except for h.265's PSNR of Y component (getPsnrMode=2) which is supported for 8-bit YUV420p planar, semi-planar, and 10-bit YUV420p planar, semi-planar pixel formats. Also note that getPsnrMode is implicitly disabled (getPsnrMode=3) when user enables PSNR with unsupported codec and/or pixel format

NOTE – Currently PSNR is not supported when cropWidth and cropHeight are set. getPsnrMode is implicitly disabled (getPsnrMode=3) when user enables PSNR along with cropWidth and cropHeight parameters

Not Applicable : JPEG

Supported Values:

0 – acquire psnr for both Y and UV component
1 – acquire psnr for UV component only
2 – acquire psnr for Y component only
3 – donot acquire psnr

Default: 3

intervalOfPsnr

Specify the interval frame numbers for outputting PSNR. Its function is to reduce performance consumption, and this params affects the input frames. Therefore, if it is a non continuous frame output, the output interval may be slightly different from the value of intervalOfPsnr, depending on the gop size in the encoding.

Not Applicable : JPEG

Supported Values: > 0

Default: 1

getReconstructedMode

Specify whether to get reconstructed frame from Hardware. Currently get reconstructed frame is supported for 8-bit YUV420p only. The workflow is like getPsnrMode.

Not Applicable : JPEG

Supported Values:

0 – donot acquire reconstructed frame
1 – acquire reconstructed frame for 10-bit YUV component

Default: 0

ReconfDemoMode

Specifies demo mode. When enabled the specified demo mode reconfigures the encoding parameters while encoding is in progress. Must be used with **ReconfFile.** For example, “ReconfDemoMode=1:ReconfFile=reconf.txt”. The types of parameters that can be reconfigured by the demo mode are listed in Supported Values. The example command below shows how to enable the bitrate reconfiguration demo:

```
ffmpeg -vsync 0 -c:v h264_ni_quadra_dec -dec 0 -y -i
Dinner_1920x1080p30_300.h264 -c:v h264_ni_quadra_enc -xcoder-params
"bitrate=1000000:RcEnable=1:ReconfDemoMode=1:ReconfFile=reconf.txt" -enc 0
out.h264
```

The Long Term Reference (LTR) and the reference invalidation features are explained in detail in sections 8.4.1 Long Term Reference and 8.4.2 Reference Invalidation, respectively.

Not Applicable: JPEG

Supported Values:

1: Bitrate
2: Intra Period
3: VUI
4: Long Term Reference (LTR) frame setting
6: Max & Min qp setting
7: Long Term Reference (LTR) interval setting
8: Frame reference invalidation
9: Framerate
10: maxFrameSize_Bytes
14: CRF
15: CRF (floating point value)
16: VBV value
17: maxFrameSize_Byte by ratio
18: sliceArg

Default: 0 (Disabled)

ReconfFile

Specifies the name of the reconf file from which the reconfiguration demo mode will read key and value information. Must be used with **ReconfDemoMode**. For example, “ReconfDemoMode=1:ReconfFile=reconf.txt”. The reconf file must be created in the same directory where the command is running. The file contains one line or multiple lines of key:value pairs.

NOTE - The **ReconfFile** command only supports using the **MaxFrameSize_Bytes.**

The content of the reconf file for the different demo mode is shown in the examples below:

Bitrate (**ReconfDemoMode 1**)

Key: frame number to change bitrate
Value: bitrate
File content example: 100:1000000

At frame number 100, bitrate changes to 1000000 bps. Add multiple lines of key:value to change the bitrate over multiple frames.

Intra Period (**ReconfDemoMode 2**)

Key: frame number to change intra period
Value: intra period
File content example: 100:30

At frame number 100, force IDR frame (aka key frame) and changes intra period to 30. Add multiple lines of key:value pairs to change intra period over multiple frames.

NOTE - reconfigure intra period is not allowed if intraRefreshMode is enabled or if gopPresetIdx is 1

VUI (**ReconfDemoMode 3**)

Key: frame number to change VUI
Value: colorDescPresent, colorPrimaries, colorTrc, colorSpace, aspectRatioWidth, aspectRatioHeight, videoFullRange
File content example: 100:1,255,255,255,16,11,1

A VUI containing the 7 listed parameter changes at frame number 100.

Long Term Reference (LTR) frame setting (**ReconfDemoMode 4**)

Refer to the description of **longTermReferenceEnable**
Key: frame number to be set as LTR
Value: 1 (set as LTR)
File content example: 10:1

frame 10 is set to LTR.

Max & Min qp setting (**ReconfDemoMode 6**)

Key: frame number to change qp

Value: minQpI, maxQpI, maxDeltaQp, minQpPB, maxQpPB

Note that maxDeltaQp is currently ignored

File content example: 100:8,51,51,8,51

At frame 100, change max & min qp

Long Term Reference (LTR) interval setting (**ReconfDemoMode 7**)

Refer to the description of **longTermReferenceInterval**

Key: frame number

Value: LTR interval

File content example: 25:5

At frame 25, change LTR interval to 5

Frame reference invalidation (**ReconfDemoMode 8**)

Key: frame number

Value: reference frame to invalidate

File content example: 24:21

At frame 24, invalidate all references with frame number >= 21

Framerate (**ReconfDemoMode 9**)

Key: frame number to change framerate

Value: framerate numerator, denominator

File content example: 100:25,1

At frame 100, change framerate to 25

MaxFrameSize parameter setting (**ReconfDemoMode 10)**

NOTE - The **MaxFrameSize** parameter has been deprecated and should NOT be used. Instead, use the more explicit **MaxFrameSize_Bytes** parameter (see below), which is equivalent of **MaxFrameSize**.
Using the new **MaxFrameSize_Bytes** parameter will also make the Quadra application code easier to understand.

Refer to the description of **MaxFrameSize** for required settings

Key: frame number to change MaxFrameSize

Value: maximum frame size in bytes

File content example: 100:200000

At frame 100, change MaxFrameSize to 200000 bytes

MaxFrameSize_Bytes parameter setting (**ReconfDemoMode 10**)

Refer to the description of **MaxFrameSize_Bytes** for required settings

Key: frame number to change MaxFrameSize_Bytes
Value: maximum frame size in bytes
File content example: 100:200000
At frame 100, change MaxFrameSize_Bytes to 200000

CRF parameter setting (**ReconfDemoMode 14**)
Key: frame number to change crf setting
Value: crf value (ranging from 0 – 51)
File content example: 100:27
At frame 100, change crf value to 27

NOTE – Reconfigure CRF parameter is valid only when CRF mode is enabled (CRF mode enabled by setting **crf** parameter when encoding session started)

NOTE – Run-time disabling CRF (reconfigure **crf** to -1) is not allowed

Not Applicable: JPEG

Supported Values: name string that has length < 10 excluding file extension

Default: null

VBV parameter setting (**ReconfDemoMode 16**)
Key: frame number to change vbv buffer size and vbv max rate setting
Value: vbvBufferSize
vbvMaxRate(0 (disable), or any value >= bitrate)
File content example: 100:2000000,2500
At frame 100, change vbvBufferSize to 2500, vbvMaxRate to 2000000

Not Applicable: JPEG

Supported Values: for vbvBufferSize(–1, 0 or [min_vbv_size ~ 3000].), The min_vbv_size is defined as 1000/framerate (one frame time) but not less than 10m
for vbvMaxRate 0 (disable), or any value >= bitrate

Default: null

MaxFrameSizeRatio parameter setting (**ReconfDemoMode 17**)

Key: frame number to change MaxFrameSizeRatio
Value: maximum frame size in ratio
File content example: 100:200
At frame 100, change MaxFrameSizeRatio to 200

NOTE – Reconfigure MaxFrameSizeRatio parameter is valid only when low_delay_mode is enabled, MaxFrameSizeRatio need to > 0, and the final MaxFrameSize will < NI_MAX_FRAME_SIZE

SliceArg parameter setting (**ReconfDemoMode 18**)
Key: frame number to change sliceArg setting
Value: sliceArg value
File content example: 100:20
At frame 100, change sliceArg value to 20

Not Applicable: JPEG, AV1, VP9

Supported Values: The minimum value is 1, The maximum value is (height+ ctu_mb_size-1)/ctu_mb_size, height is video height, ctu_mb_size is 16 for h264, 64 for h265

Default: null

enableAIEnhance

Embedded image enhancement models are utilized to improve the visual quality of videos during the transcoding process. Upon decoding the YUV hardware frames, they are transmitted to the Quadra AI Inference Engine, which generates enhanced frames of identical size and format, before passing them on to the encoder. Generally, this approach results in videos with superior visual quality.

NOTE – This feature only support 8bit with following resolution:

1280x720,1920x1080,3840x2160,720x1280,1080x1920,2496x1080

Not Applicable: JPEG

Supported Values:

= 0: disable the feature.
=1: enable the feature.

Default: 0

NOTE – Specify AIEnhanceLevel when enabling this parameter.

AIEnhanceLevel

Specifies the adjust level of AI engine to AI enhance optimization. The higher means the AI engine would change more about the original picture and the PSNR may be lower.

NOTE – This feature is only effective when enableAIEnhance=1.

Not Applicable: JPEG

Supported Values:

= 1: Lowest
=2: Medium
=3: Highest

Default: 3

enableHVSPlus

HVS+ utilizes Quadra's internal AI engine to enhance video quality during encoding by mimicking the human visual system. It optimizes video content through perceptual optimization, achieving a balanced performance between PSNR (Peak Signal-to-Noise Ratio) and VMAF (Video Multi-Method Assessment Fusion). This ensures that both objective and subjective quality metrics are met. HVS+ offers two preconfigured settings tailored to different types of video content, providing optimal enhancement for various genres.
NOTE – This feature only support 8bit with following resolution:

1280x720,1920x1080,3840x2160

Not Applicable: JPEG

Supported Values:
= 0: disable the feature.
=1: enable the feature.

Default: 0

NOTE – Specify HVSPlusLevel when enabling this parameter. And this parameter cannot work with enableAIEnhance at same time.

HVSPlusLevel

Level 1 preserves the PSNR better in general. Level 2 is more towards improving the subject quality.
NOTE – This feature is only effective when enableHVSPlus=1.

Not Applicable: JPEG

Supported Values:
= 1: Better PSNR
= 2: Better subject quality

Default: 1

cropWidth

Specifies cropping width, must be even.

NOTE – When specified, cropWidth + horOffset must be less than input picture width
NOTE – When specified, cropHeight must also be specified a valid value

Not Applicable: JPEG

Supported Values: 144 pixels to input picture width

Default: 0

cropHeight

Specifies cropping height, must be even.

NOTE – When specified, cropHeight + verOffset must be less than input picture height
NOTE – When specified, cropWidth must also be specified a valid value

Not Applicable: JPEG

Supported Values: 128 pixels to input picture height

Default: 0

horOffset

Specifies horizontal cropping offset, must be even.

NOTE – horOffset is only effective when cropWidth and cropHeight are specified
NOTE – When specified, cropWidth + horOffset must be less than input picture width

Not Applicable: JPEG

Supported Values: 0 to input picture width

Default: 0

verOffset

Specifies vertical cropping offset, must be even.

NOTE – verOffset is only effective when cropWidth and cropHeight are specified
NOTE – When specified, cropHeight + verOffset must be less than input picture height

Not Applicable: JPEG

Supported Values: 0 to input picture height

Default: 0

motionContrainedMode

Enables Motion Constrained mode 1 (performance mode) or mode 2 (quality mode). When Motion Constrained mode is enabled, the encoder will encode specific blocks in intra mode to prevent motion vectors from crossing over picture boundary, allowing the encoded frame to be converted to HEVC tile and stitched together with other tiles to form multi-tiles bitstream, without causing artifacts when the bitstream is decoded. One use case of this feature is for multiview application which displays multiple video sequences simultaneously on the display device. Please also note that Motion Constrained mode has negative impacts on performance and/or compression efficiency due to the constraint of motion compensation. Motion Constrained mode 1 (performance mode) does not lower performance, but lowers compression efficiency, while mode 2 (quality mode) lowers performance, but has much better quality compared to mode 1 at the same bitrate. Currently motionConstrainedMode is only supported in HEVC/h.265 encode.

NOTE – When motionConstrainedMode is 1 (performance mode) or 2 (quality mode), rdoLevel must be set to 1

NOTE – When motionConstrainedMode is 2 (quality mode), input resolution (or **cropWidth**, **cropHeight**) must be 64x64 aligned

NOTE – When motionConstrainedMode is 2 (quality mode), multicoreJointMode must be set to 0 (disabled)

Not Applicable: H264, AV1, JPEG

Supported Values: 0 (disabled), 1 (performance mode), 2 (quality mode)

Default: 0

Rate Control

RcEnable

Enables or disables rate control.

Supported Values:

0: Disable
1: Enable

Default: 0

bitrate

Used when **RcEnable** is enabled (set to 1) or **crf** value is specified (crf value >= 0), otherwise it is ignored. The encoding bitrate is in bps.

NOTE – Please also refer to **bitrateMode** parameter, which is used to control whether rate control handles user specified **bitrate** as the maximum bitrate or the average bitrate **RcEnable** is 1 (ABR or CBR mode), **cuLevelRCEnable** is 1 (enable block level rate control), and **lookaheadDepth** is 0 (no lookahead.

Supported Values:

The range is 10000 to 800000000. As an example, set 3000000 for 3 Mbps.

Default: 200000

bitrateMode

Decides rate control behavior when **RcEnable=**1, **cuLevelRCEnable=**1, and **lookaheadDepth**=0 (NOTE - lookaheadDepth default value is 0).

When **bitrateMode** is 0 (default), encoder perceives **bitrate** as the maximum bitrate allowed, and therefore may produce bitrate lower than **bitrate** to achieve higher compression efficiency. When **bitrateMode** is 1, encoder perceives **bitrate** as the average bitrate target, and shall produce bitrate close to the target.

NOTE – **bitrateMode** mainly affects low delay gop patterns, including **gopPresetIdx** 3, 7, 9. For other gop presets, the effects of **bitrateMode** may not be noticeable.

NOTE – Setting **bitrateMode**=1 may cause more instantaneous bitrate fluctuations and/or bitrate spikes compared to **bitrateMode** 0. Therefore, it is recommended to use **bitrateMode** 0 for applications which need to prevent bitrate spike.

Applicable: This parameter is valid when **RcEnable** is 1, **cuLevelRCEnable** is 1 (enable block level rate control), and **lookaheadDepth** is 0 (no lookahead). Please also note this parameter does not affect CRF or Capped CRF mode (refer to **crf** parameter).
Not Applicable: JPEG

Supported Values:

0: Maximum bitrate
1: Average bitrate

Default: 0 (Maximum bitrate)

enableVFR

Specifies the variable framerate(VFR) encoding is support or not.

The VFR feature is disabled by default and the encoder rate control assumes that the framerate is constant. When enabled, the VFR feature uses the video timestamps to calculate the average framerate every second. Encoding a VFR stream without VFR enabled will likely result in the encoded bitrate being wrong.

For example, the framerate reported in a video container may be 25 fps, while the average framerate from the timestamps could be 20fps. Without VFR enabled, the average encoded bitrate in this case would be only 80% of the target.

Enabling the VFR feature also causes h.264 fixed_frame_rate_flag syntax to be set to 0.

Supported Values:

0: Disable
1: Enable

Default: 0

vbvBufferSize

Specifies the size of the VBV (CPB) buffer in milliseconds. This parameter determines if HRD (Hypothetical Reference Decoder) compliance is followed when **RcEnable** is 1, or if peak bitrate is constrained when **crf** >= 0. Setting vbvBufferSize to 0 will disable HRD compliance or peak bitrate constraint.

NOTE – When vbvBufferSize is not specified (or when vbvBufferSize is set to -1), the default setting varies depending on the rate control mode. For CBR mode (**RcEnable** = 1), default vbvBufferSize is 3000. For CRF mode (**crf** >= 0), default vbvBufferSize is 0, which means peak bitrate is not constrained.

NOTE – Please also note that **bitrate** must also be specified along with vbvBufferSize, including Capped CRF mode and Constrained VBR mode in which the **vbvMaxRate** might have already been specified. This is because both **bitrate** and **vbvBufferSize** parameters are required to calculate the actual buffer size, and vbvMaxRate does not serve as replacement in afore mentioned calculation. Please refer to Section “8.3.4 CRF & Capped CRF Descriptions and Examples” for more details.

Applicable: This value is valid if **RcEnable** = 1 or **crf** >= 0
Not Applicable: JPEG
Supported Values:

The value of vbvBufferSize is in unit of millisecond; for example, vbvBufferSize 3000 sets the buffer size to (bitrate * 3 seconds).
The range of supported values is –1, 0 or [min_vbv_size ~ 3000].
min_vbv_size is 1000/framerate (one frame time) but not less than 10ms
Value 0 disables HRD compliance and peak bitrate constraint.
Value –1 (default) is the auto mode in which the default VBV buffer size is determined by rate control mode (3000 in CBR mode, 0 (no VBV buffer constraint) in CRF mode).

Default: 3000 in CBR mode **(RcEnable** = 1)
0 in CRF mode **(crf** >= 0)

vbvBufferReencode

When vbvBufferReencode is set to 1, the encoder will re-encode frame(s) if their encoded bit count exceeds the VBV buffer limit, ensuring compliance with VBV (also known as HRD) constraints. Please note that re-encoding increases latency, so it is recommended only when strict adherence to VBV (HRD) conformance is required.

NOTE – Please note that encoder cannot guarantee HRD conformance when **multicoreJointMode** is set to 1, so it is recommended not enable vbvBufferReencode in multicoreJointMode.

NOTE – For AV1 encoding, if multicoreJointMode is set to 1, vbvBufferReencode is not supported.

Not Applicable: JPEG

Supported Values:

0: Disable
1: Enable

Default: 0

vbvMaxRate

The vbvMaxRate parameter sets the maximum bitrate limit during encoding in Constrained VBR mode and Capped CRF mode. Compared to ABR mode, in which **vbvBufferSize** and **vbvMaxRate** are both set to 0, and CBR mode, in which **vbvBufferSize** is set to valid non-zero values and **vbvMaxRate** is set to 0, Constrained VBR mode requires **vbvBufferSize** and **vbvMaxRate** both set to valid non-zero values. In Capped CRF mode, encoder implicitly takes **bitrate** as the maximum bitrate limit if **vbvMaxRate** is not set; otherwise, encoder will take **vbvMaxRate** as the maximum bitrate limit when it is set. Please refer to Section "8.3.4 CRF & Capped CRF Descriptions and Examples" for more details.

NOTE – In Constrained VBR mode, **bitrate** parameter specifies the average bitrate, while **vbvMaxRate** sets the maximum bitrate limit. The instantaneous (per frame) bitrate fluctuation, however, is constrained by vbvBufferSize – larger VBV buffer may result in higher instantaneous bitrate fluctuation.

Constrained VBR mode sample FFmpeg command –

```
ffmpeg -c:v h264_ni_quadra_dec -i crowdrun_1920x1080.264 -c:v
h264_ni_quadra_enc -xcoder-params
"RcEnable=1:bitrate=3000000:vbvMaxRate=6000000:vbvBufferSize=1
000" quadra.h264
```

NOTE – In Constrained VBR mode, **bitrate** parameter specifies the average bitrate, while **vbvMaxRate** sets the maximum bitrate limit. The instantaneous (per frame) bitrate fluctuation, however, is constrained by vbvBufferSize – larger VBV buffer may result in higher instantaneous bitrate fluctuation.

NOTE – In Constrained VBR mode, **vbvBufferSize** must be >= average bits per frame, which can be calculated by 1000 / framerate (in unit of ms). Also note that encoder will reject configuration if vbvBufferSize < (1000 / framerate)

NOTE – In Constrained VBR mode, **vbvMaxRate** must be >= bitrate. Also note that encoder will reject configuration if vbvMaxRate < bitrate

NOTE – Please also note that **bitrate** must also be specified along with vbvBufferSize, including Capped CRF mode and Constrained VBR mode in which the **vbvMaxRate** might have already been specified. This is because both **bitrate** and **vbvBufferSize** parameters are required to calculate the actual buffer size, and vbvMaxRate does not serve as replacement in afore mentioned calculation.

Not Applicable: JPEG
Supported Values: 0 (disable), or any value >= bitrate. Encoder will reject configuration if vbvMaxRate < bitrate.

Default: 0

NOTE – the following table provides a brief overview of min, max, average runtime bitrate as handled and reported by various supported bitrate controls

Types of bitrate control	Brief Definition	vbvBufferSize	vbvMaxRate	Runtime bitrate variation compared to bitrate setting
ABR (Average Bit Rate)	In ABR mode, rate control maintains average bitrate to match target bitrate, but is not constrained by VBV buffer, and therefore instant bitrate can fluctuate more compared to CBR. On the other hand, in ABR mode, the average bitrate will be more closely matching the target bitrate.	0	0	Average bitrate ~= **bitrate** parameter value
CBR (Constant Bit Rate)	In CBR mode, rate control is bound by the VBV buffer constraint, and therefore the instant bitrate will be limited and more stable.	10-3000	0	Instant bitrate (per second) is constrained by **bitrate** parameter value Average bitrate <= **bitrate** parameter value (NOTE – bitrate can be lower than target due to bits saving when bitrateMode = 0, please refer to **bitrateMode** parameter description for more details)
Constrained VBR (Constrained Variable Bit Rate)	Compared to CBR mode, Constrained VBR mode allows higher instant bitrate, and therefore may produce higher quality at the cost of higher peak rate	10-3000	value >= bitrate setting	Instant bitrate (per second) is constrained by **vbvMaxRate** parameter value Average bitrate <= **bitrate** parameter value (NOTE – bitrate can be lower than target due to bits saving when bitrateMode = 0, please refer to **bitrateMode** parameter description for more details)

Types of bitrate control	Brief Definition	vbvBufferSize	vbvMaxRate	Runtime bitrate variation compared to bitrate setting
Capped CRF (Capped Constant Rate Factor)	In Capped CRF mode, the encoder attempts to maintain constant subjective quality, while bitrate is capped by a maximum bitrate limit. (The user may also add a minimum bitrate limit if needed)	10-3000	value >= bitrate setting (NOTE – if vbvMaxRate is not set, maximum bitrate limit is implicitly set to be equal to `bitrate` parameter)	Instant bitrate (per second) is constrained by **bitrate** parameter value Average bitrate <= **bitrate** parameter value

fillerEnable

When enabled, the HRD compliance check algorithm can add filler data when the VBV (CPB) buffer underflows. This parameter enables rate control and HRD if not already enabled. i.e. **RcEnable** and **vbvBufferSize** will become 1 and 3000 respectively.

Enabling **fillerEnable** will:

1. Automatically set **RcEnable** = 1
2. Automatically set default **vbvBufferSize** 3000

NOTE – Enabling **fillerEnable** can slow down the encoder depending on the amount of filler data added.

Not Applicable: AV1 and JPEG

Supported Values:

0: Disable
1: Enable

Default : 0

picSkip

When enabled, the HRD algorithm can skip pictures when the VBV (CPB) buffer overflows. HRD rate control is enabled when **rcEnable** is 1 and **vbvBufferSize** > 0.

If **picSkip** is enabled, and the maximum output size has been specified with either **MaxFrameSize(Deprecated)**, **MaxFrameSize_Bits** or **MaxFrameSize_Bytes**, then the encoder will discard any frames whose output size exceeds the specified size.

NOTE – When enabled, the **gopPresetIdx** must be 1, 3, 7, 9, 10 (or 0 if the custom gop encodes in consecutive order), lookaheadDepth and multicoreJointMode must be 0

NOTE – picSkip can cause loss of SEI packets, do not enable picSkip if the application requires SEI in the bitstream

Not Applicable: JPEG

Supported Values:

0: Disable
1: Enable

Default : 0

skipFrameEnable

When enabled, the encoded frame will be forced to be a skip frame, meaning that all macroblocks are encoded as skip mode macroblocks when VBV buffer overflows.

NOTE – When enabled, the gopPresetIdx must be 1, 3, 7, 9, 10(or 0 if the customer gop encodes in consecutive order). Otherwise, the video will have ghosting visual effects.

NOTE – This parameter can be used with **maxConsecutiveSkipFrameNum,** which can be used to limit the maximum number of consecutive skip frames. This parameter can be used with **skipFrameInterVal,** which can be used to force the insertion of a skip frame every skiFrameInterVal frames.

NOTE – Although it can limit the bit rate, there may be additional unknown issues. Please use this parameter when you are clear about what you are doing.

Not Applicable: JPEG
Supported Values:

0: Disable
1: Enable

Default : 0

maxConsecutiveSkipFrameNum

Specify the maximum number of consecutive skip frames. This parameter only takes effect when the skipFrameEnable is enabled.

NOTE – A value that is too large may cause the video to appear sluggish. It is recommended that the value be less than 5.

Not Applicable: JPEG
Supported Values:
Values greater than or equal to 0

Default : 1

skipFrameInterVal

Specify the insertion of a skip frame every skipFrameInterVal frame interval. This parameter only takes effect when the skipFrameEnable is enabled.

This parameter may improve the image quality of certain videos, but it may not have a significant effect on optimizing the quality of certain videos. Also, if the encoding frame rate is low or if the skipFrameInterVal is too small, there may be lag phenomenon. It is recommended that the value in the range of 7 to 10 for 30fps encoding.

Not Applicable: JPEG
Supported Values: 0 - 255

Default : 0

stillImageDetectLevel

Specify the level of detection for static frames. Setting different levels is because the definition of static is sometimes very subjective, and different scenarios have slightly different requirements for static.

The levels are divided into two categories: 1-3 and 4-6. Levels 1-3 do not require additional resources and do not reduce performance, but their accuracy is not as high as levels 4-6.

The larger the value of level, the stricter the judgment of whether it is a static frame. When the level value is large, only when the image is completely still, it is judged as a static frame. When the level value is small, if there are very small changes in the image (such as a change area that may only be 1%-3% of the entire image), it will also be considered a still image.

Note: The detection of still Image is after encoding one frame. Currently the param is only supported with gopPresetIdx=9.

Not Applicable: JPEG
Supported Values: 0 – 6

Default : 0

sceneChangeDetectLevel

Specify the level of detection for scene change. Setting different levels is because the definition of static is sometimes very subjective, and different scenarios have slightly different requirements for scene change.

The levels are divided into two categories:1-5 and 6-10. Levels 1-5 do not require additional resources and do not reduce performance, but their accuracy is not as high as Levels 6-10.

The same as the above stillImageDetectLevel, the larger the value of level, the stricter the judgment of whether it is a scene change. When the level value is large, only the current frame is completely different from the before frame, will be considered a scene change. When the level is small, there is a significant difference between the current frame and the previous frame, it may be considered a scene change.

Note: The detection of scene Change is after encoding one frame. Currently the param is only supported with gopPresetIdx=9.

Not Applicable: JPEG
Supported Values: 0 – 10

Default : 0

maxFrameSize -- DEPRECATED

NOTE – The **maxFrameSize** parameter has been **deprecated** and should NOT be used. Use the more explicit **maxFrameSize_Bytes** parameter instead (see below), which is equivalent of the **maxFrameSize**.
Using the new **maxFrameSize_Bytes** parameter will also make the Quadra application code easier to understand.

Specifies the maximum allowable output byte count for any frame (please note the unit of **maxFrameSize** is in ***bytes***). While encoding, the encoder will re-encode the frame with higher QP if the output byte count exceeds the **maxFrameSize**. This process will continue until the output byte count becomes less than **maxFrameSize**. If the QP has been increased to the **maxQP** but the output byte count still exceeds **maxFrameSize**, the encoder will either produce the output, or if **picSkip** is enabled, it will discard the output (see the **picSkip** parameter description).

Note that **maxFrameSize** is only valid if **lowDelay** is enabled. If **lowDelay** is disabled, **maxFrameSize** parameter will be rejected. If **lowDelay** is enabled, but **maxFrameSize** is not specified in the command line, libxcoder will set **maxFrameSize** to the default value, which is ½ of the input frame size.

This feature can also be used to limit the maximum frame size to a multiple of the average frame size, using the **ratio** keyword. For example, **maxFrameSize=ratio[8]** will limit the maximum frame size to be no more than 8 times the average frame size calculated by the libxcoder.

NOTE – If the bitrate or framerate is reconfigured via the libxcoder API, the encoder will automatically adjust the **maxFrameSize** to adapt to the new bitrate or framerate. User applications may also reconfigure the **maxFrameSize** via the libxcoder API to override the encoder auto adjusted **maxFrameSize** value (see Libxcoder API Integration Guide).

Not Applicable: JPEG
Supported Values: Acceptable values range from the minimum of bitrate divided by framerate, to the maximum of the input frame size
Default: ½ of input frame size

NOTE Quadra only supports using one of the 3 **maxFrameSize** parameters at a time. For example, only use either the **maxFrameSize** (**Deprecated**), or the **maxFrameSize_Bits** or the **maxFrameSize_Bytes** within the same command. It is recommended to use the **maxFrameSize_Bytes** only.

maxFrameSize_Bytes

NOTE – The **maxFrameSize_Bytes** parameter is equivalent of **maxFrameSize** which has been **deprecated**. The user should use this more explicit **maxFrameSize_Bytes** parameter rather than **maxFrameSize** to make the Quadra application code more readable.

Specifies the maximum allowable output byte count for any frame. The size specified with **maxFrameSize_Bytes** is *bytes*. While encoding, the encoder will re-encode the frame with higher QP if the output byte count exceeds **maxFrameSize_Bytes**. This process will continue until the output byte count becomes less than **maxFrameSize_Bytes**. If the QP has been increased to the **maxQP** but the output byte count still exceeds **maxFrameSize_Bytes**, the encoder will either produce the output, or if **picSkip** is enabled, it will discard the output (see the **picSkip** parameter description).

Note that **maxFrameSize_Bytes** is only valid if **lowDelay** is enabled. If **lowDelay** is disabled, the **maxFrameSize_Bytes** parameter will be rejected. If **lowDelay** is enabled, but **maxFrameSize_Bytes** is not specified in the command line, the libxcoder will set **maxFrameSize_Bytes** to the default value, which is ½ the input frame size.

This feature can also be used to limit the maximum frame size to a multiple of the average frame size, using the **ratio** keyword. For example, **maxFrameSize_Bytes =ratio[8]** will limit the maximum frame size to be no more than 8 times the average frame size calculated by the libxcoder.

NOTE – If specified **maxFrameSize_bytes** is less than (bitrate/framerate), libxcoder will adjust maxFrameSize_bytes to (bitrate/framerate).

NOTE – If the bitrate or framerate is reconfigured via the libxcoder API (see the **Libxcoder API Integration Guide**), then the encoder will automatically adjust the **maxFrameSize_Bytes** to adapt to the new bitrate or framerate. User applications may also reconfigure the **maxFrameSize_Bytes** via the libxcoder API to override the encoder auto adjusted **maxFrameSize_Bytes** (again refer to the **Libxcoder API Integration Guide**).

Not Applicable: JPEG
Supported Values: Acceptable values range from ((bitrate / 8) / framerate) (min value, in bytes), to input frame size (max value, in bytes)
Default: ½ of input frame size in bytes

NOTE Quadra only supports using one of the 3 **maxFrameSize** parameters at a time. For example, only use either the **maxFrameSize** (**Deprecated**), or the **maxFrameSize _Bits** or the **maxFrameSize_Bytes** within the same command. It is recommended to use **maxFrameSize_Bytes** only.

maxFrameSize_Bits

Specifies the maximum allowable output bit count for any frame (please note the unit of **maxFrameSize_Bits** is in ***bits***). While encoding, the encoder will re-encode the frame with a higher QP if the output bit count exceeds the **maxFrameSize_Bits**. This process will continue until the output bit count becomes less than the **maxFrameSize_Bits**. If the QP has been increased to the **maxQP** but the output bit count still exceeds **maxFrameSize_Bits**, the encoder will either produce the output, or if **picSkip** is enabled it will discard the output (see the **picSkip** parameter description).

Note that **maxFrameSize_Bits** is only valid if **lowDelay** is enabled. If **lowDelay** is disabled, **maxFrameSize_Bits** parameter will be rejected. If **lowDelay** is enabled, but **maxFrameSize_Bits** is not specified in the command line, libxcoder will set **maxFrameSize_Bits** to the default value, which is ½ of the input frame size.

This feature can also be used to limit the maximum frame size to a multiple of the average frame size, using the **ratio** keyword. For example, **maxFrameSize_Bits =ratio[8]** will limit the maximum frame size to be no more than 8 times the average frame size calculated by the libxcoder.

NOTE – If the bitrate or framerate is reconfigured via the libxcoder API (please refer to the Libxcoder API Integration Guide), then the encoder will automatically adjust the **maxFrameSize_Bits** to adapt to the new bitrate or framerate. User applications may also reconfigure the **maxFrameSize_Bits** via the libxcoder API to override the encoder auto adjusted **maxFrameSize_Bits** (refer to Libxcoder API Integration Guide).

Not Applicable: JPEG
Supported Values: Acceptable values range from bitrate divided by framerate (min) to input frame size (max)
Default: ½ of input frame size in bits

NOTE Quadra only supports using one of the 3 **maxFrameSize** parameters at a time. For example, only use either the **maxFrameSize** (**Deprecated**), or the **maxFrameSize _Bits** or the **maxFrameSize_Bytes** within the same command.

It is recommended to use **maxFrameSize_Bytes** only.

crf

This parameter enables Constant Rate Factor mode. CRF is a constant subjective quality mode as opposed to normal rate control which adjusts quality to maintain constant bitrate (CBR). On the other hand, it is not possible to predict the bitrate a CRF encode will come out to. CRF values range from 0 to 51, lower value indicates higher quality and higher value indicates lower quality. The recommended CRF value is 23.

CRF can also be used together with **bitrate**, **vbvBufferSize, vbvMaxRate, vbvMinRate** parameters to maintain a consistent quality level, while the maximum (and minimum) bitrate cap prevents the bitrate from exceeding a certain threshold – this mode is referred to as Capped CRF mode. For more details and examples of Capped CRF encode parameters, please refer to Section 8.3.4 "**CRF & Capped CRF Examples**".

NOTE – When CRF mode is enabled, **rcEnable** must be 0

NOTE – CRF mode can be enabled with lookahead (l**ookaheadDepth** 4~40) or without lookahead (**lookaheadDepth** 0), the supported GOP patterns in each mode are described below

- CRF with **lookaheadDepth** 0 – supports gopPresetIdx -1 (adaptive gop), 0 (custom gop), 1, 4, 5, 8, 9
- CRF with **lookaheadDepth** > 0 – supports gopPresetIdx -1 (adaptive gop), 0 (custom gop), 4, 5, 8, 9

Not Applicable: JPEG

Supported Values: 0-51

Default: -1 (disabled)

crfFloat

This parameter enables Constant Rate Factor mode. Supports floating-point numbers with 2 decimal places. CRF is a constant subjective quality mode as opposed to normal rate control which adjusts quality to maintain constant bitrate (CBR). On the other hand, it is not possible to predict the bitrate a CRF encode will come out to. CRF values range from 0.00 to 51.00, lower value indicates higher quality and higher value indicates lower quality. The recommended CRF value is 23.00.

CRF can also be used together with **bitrate**, **vbvBufferSize, vbvMaxRate, vbvMinRate** parameters to maintain a consistent quality level, while the maximum (and minimum) bitrate cap prevents the bitrate from exceeding a certain threshold – this mode is referred to as Capped CRF mode. For more details and examples of Capped CRF encode parameters, please refer to Section 8.3.4 "**CRF & Capped CRF Examples**".

NOTE When CRF mode is enabled, **rcEnable** must be 0

NOTE CRF mode can be enabled with lookahead (l**ookaheadDepth** 4~40) or without lookahead (**lookaheadDepth** 0), the supported GOP patterns in each mode are described below

- CRF with **lookaheadDepth** 0 – supports gopPresetIdx -1 (adaptive gop), 0 (custom gop), 1, 4, 5, 8, 9
- CRF with **lookaheadDepth** > 0 – supports gopPresetIdx -1 (adaptive gop), 0 (custom gop), 4, 5, 8, 9

Not Applicable: JPEG
Supported Values: 0.00-51.00
Default: -1.00 (disabled)

crfMaxIframeEnable

The **crfMaxIframeEnable** parameter allows encoder to produce higher quality I-frames in Capped CRF mode to improve quality. The side effect is that I-frame bit count will be increased, which may not be ideal in scenarios where bitrate spike is unacceptable.

Not Applicable: JPEG

Supported Values:

0: Disable
1: Enable

Default : 0

enableSmoothCrf

The **enableSmoothCrf** parameter determines whether the size curve of the encoded stream will be smoothed when different crfFloat values are set to foloating-point type.Without enabling this parameter, the size of the encoded stream size may increase as the crfFloat value increases. If this parameter is enabled, the size of the encoded stream decreases as the crfFloat value increases.

NOTE –This parameter will only take effect when crfFloat is set.

Not Applicable: JPEG

Supported Values: 0 (disable), 1.

Default: 0

vbvMinRate

The **vbvMinRate** parameter sets the minimum bitrate limit during encoding in Capped CRF mode. (NOTE – unlike **vbvMaxRate** parameter which takes effect in both Constrained VBR mode and Capped CRF mode, **vbvMinRate** only takes effect in Capped CRF mode.)

In Capped CRF mode, if **vbvMinRate** is not set, the encoder will not constrain bitrate by a minimum bitrate limit (only limit maximum bitrate). If **vbvMinRate** is set, the encoder will produce more bits to meet the minimum bitrate, which may also raise quality to be higher than the specified consistent quality level (which depends on **crf** parameter value) when bits required at specified quality level is lower than the minimum bitrate. Please refer to Section “8.3.4 CRF & Capped CRF Descriptions and Examples” for more details.

Capped CRF mode sample FFmpeg command with vbvMinRate –

```
ffmpeg -f rawvideo -pix_fmt yuv420p -s 1920x1080 -r 30.0 -i
input1920x1080.yuv -c:v h265_ni_quadra_enc -xcoder-params
"gopPresetIdx=5:RcEnable=0:crf=23:lookAheadDepth=10:vbvBufferS
ize=1000:bitrate= 3200000:vbvMinRate=1000000"
output1920x1080.h265
```

NOTE – **vbvMinRate** must be <= bitrate. Also note that encoder will reject configuration if vbvMinRate > bitrate

Not Applicable: JPEG

Supported Values: 0 (disable), or any value <= bitrate. Encoder will reject configuration if vbvMaxRate > bitrate.

Default: 0

qcomp

This parameter affects how much bitrate is given to important scenes as opposed to unimportant scenes for subjective quality. Only effective in CRF mode.

NOTE – qcomp takes effects only if **crf** is set

Not Applicable: JPEG

Supported Values: 0.0 to 1.0

Default: 0.6

ipRatio

Modifies average I-frames bit count increase as compared to P-frames. Higher values increase the quality of I-frames generated. Only effective in CRF mode.
NOTE – ipRatio takes effects only if **crf** is set. If want to enable ipRatio in CBR or VBR mode, should set enableipRatio to 1.

Not Applicable: JPEG
Supported Values: 0.01 to 10.0
Default: 1.40

enableipRatio

Enable or disable the ipRatio when in CBR and VBR mode. This parameter only affect ipRatio in CBR and VBR mode. If crf is set, ipRatio always works, regardless of whether enableipRatio is enabled or not.
NOTE – If you donnot want to using ipRatio in CBR or VBR, please also not setting enableipRatio either.

Not Applicable: JPEG
Supported Values: 0 to 1
Default: 0

iFrameSizeRatio

Specifies the ratio of I frame size to default I frame size. The ratio of the frame size is not exactly the same as the specified ratio, they are just approximate. This is influenced by different bitrate control algorithms.
NOTE – Please do not use ipRatio and iFrameSizeratio together.

Not Applicable: JPEG
Supported Values: integer 1 to 300
Default: 100

pbRatio

Modifies average B-frames bit count decrease as compared to P-frames. Higher values decrease the quality of B-frames generated. Only effective in CRF mode.

NOTE – pbRatio takes effects only if **crf** is set

Not Applicable: JPEG
Supported Values: 0.01 to 10.0
Default: 1.30

cplxDecay

This parameter affects how fast the past frames complexity decays when encoder decides current frame quantization based on complexity. Only effective in CRF mode.

NOTE – cplxDecay takes effects only if **crf** is set

Not Applicable: JPEG
Supported Values: 0.1 to 1.0
Default: 0.5

chromaQpOffset

Offset of Cb/Cr chroma quantization from the luma quantization selected by rate control. This is a general way to specify encoder spending more or less bits on the chroma channel.

Not Applicable: JPEG
Supported Values: -12 to 12
Default: 0

hvsQPEnable

Enable or disable block level QP adjustment for subjective quality enhancement.

Not Applicable: JPEG
Supported Values:
0: Disable
1: Enable
Default: 0

hvsBaseMbComplexity

Base complexity of block level QP adjustment for subjective quality. A higher base complexity value will result in higher subjective quality, at the cost of increased bit count. In low target bitrate use case, increasing base complexity value may cause actual bitrate to be higher than the target bitrate. Default base complexity value is 15.

NOTE – hvsBaseMbComplexity only takes effects if **hvsQPEnable** is set to 1

Not Applicable: JPEG
Supported Values: 0 to 31
Default: 15

CuLevelRCEnable

Enable or disable coding unit level rate control.

NOTE – If this flag is 1 and intraPeriod is small (i.e. <30), I frame flickering for static scene can be observed. If this happens, either set tolCtbRcIntra=-1 to disable CU level rate control of I-frames, or increase tolCtbRcIntra>1 so that CU level rate control will give I-frame bits large tolerance to work around this issue.

Applicable: This value is valid when **RCEnable** is 1 (ABR or CBR mode), or when **crf**, **bitrate**, and **vbvBufferSize** are set (Capped CRF mode)
Not Applicable: JPEG
Supported Values:
0: Disable
1: Enable
Default: 0

tolCtbRcInter

Tolerance of CU level rate control for P-frames and B-frames (aka INTER frames). CU level rate control will enforce INTER frame bits limit within the range of: [ExpectedBits * (1 – tolCtbRcInter), ExpectedBits * (1 + tolCtbRcInter)], except for -1 and 0 as described below:
If tolCtbRcInter == -1, CU level rate control is disabled for INTER frames
If tolCtbRcInter == 0, CU level rate control adjust INTER frames bits according to the current VBV buffer level. The total VBV buffer size can be configured by **vbvBufferSize.**

NOTE – tolCtbRcInter only takes effects if **cuLevelRCEnable** is set to 1

Not Applicable: JPEG

Supported Values:

=-1: CU level rate control is disabled for P/B-frames

= 0: CU level rate control adjust P/B-frame bits according to current VBV buffer level

> 0: CU level rate control adjust P/B-frames bits according to the tolerance setting, which is applied to the target picture size as described above

Default: 0.1

tolCtbRcIntra

Tolerance of CU level rate control for I-frames (aka INTRA frames). CU level rate control will enforce INTRA frame bits limit within the range of: [ExpectedBits * (1 – tolCtbRcIntra), ExpectedBits * (1 + tolCtbRcIntra)], except for -1 and 0 as described below:
If tolCtbRcIntra == -1, CU level rate control is disabled for I-frames
If tolCtbRcIntra == 0, CU level rate control adjust I-frames bits according to the current VBV buffer level. The total VBV buffer size can be configured by **vbvBufferSize**.

NOTE – tolCtbRcIntra only takes effects if **cuLevelRCEnable** is set to 1

Not Applicable: JPEG
Supported Values:

= -1: CU level rate control is disabled for I-frames

= 0: CU level rate control adjust I-frames bits according to the current VBV buffer level

> 0: CU level rate control adjust I-frames bits according to the tolerance setting, which is applied to the target picture size as described above
Default: 0.1

rcQpDeltaRange

Max absolute value of CU/MB QP delta relative to frame QP in CTB RC.
It controls maximum range CU/MB QP is allowed to vary within a frame.

NOTE – rcQpDeltaRange only takes effects if **cuLevelRCEnable** = 1 or **hvsQPEnable** = 1.

Not Applicable: JPEG

Supported Values:

0 to 15
0 = effectively disables cuLevelRCEnable and/or hvsQPEnable

Default: 10

bitrateWindow

Specifies the bitrate window length in frames. Rate control allocates bits for one window and tries to match the target bitrate at the end of each window. This enforces an average target bit rate within the window interval, allowing encoder to adapt to changes of complexity on video input and variations on compression efficiency between frames. Typically window begins with an intra frame, but this is not mandatory.

By default bitrateWindow = intraPeriod

If intraPeriod = 0 (infinite period), bitrateWindow = 150

Not Applicable: JPEG

Supported Values: 1 to 300

Default: equals intraPeriod

ctbRowQpStep

The maximum accumulated QP adjustment step per CTB Row allowed by CU level rate control. Increasing ctbRowQpStep helps reduce instant bitrate spike / overshoot, but it may also reduce compression efficiency. CU level rate control QP step per CTB is calculated by

QP_step_per_CTB = MIN((ctbRowQpStep / Ctb_per_Row),4).

If ctbRowQpStep == 0, Quadra will use pre-defined ctbRowQpStep value [4] for H.264 and [16] for HEVC.

NOTE – The maximum effective value of ctbRowQpStep = CTB per Row * 4. For example, if input resolution is 1920x1080, CTB per row = 1920 / 64 = 30, and maximum effective ctbRowQpStep = 30 * 4 = 120. A larger value is also accepted, but will have the same effects as the maximum effective value.

NOTE – ctbRowQpStep only takes effects if **cuLevelRCEnable** is set to 1

Not Applicable: JPEG

Supported Values: 0 to 500, setting 0 will cause Quadra to use pre-defined ctbRowQpStep value [4] for H.264 and [16] for HEVC

Default: 0

enableAcqLimit

Whether to enable encoder to limit its upstream's output buffering count. Enabling this option could prevent the upstream instance outputting more buffers than encoder needs.

NOTE – enableAcqLimit is enabled by default for FFmpeg-n6.1 and FFmpeg-n7.1.

Supported Values: 0 and 1. 0: disabled. 1: enabled.
Default: 0

customizeQpLevel

Specifies the level of the qp map. When customizeQpLevel > 0. The qp_info in roi block indicates the row number it chooses in customizeQpMapFile. The qp_info range from 0 to 10. And it will choose qp in customizeQpMapFile table after rate control, the qp after rate control indicate the column number and qp_info indicates the row number. It will take effect when ROI is enabled (**roiEnable**=1) and customizeQpMapFile is provided. customizeQpLevel is currently only supported on FFmpeg 3.4.2 and above. It supports qp level 0 and 1

Applicable: FFmpeg 3.4.2 and above

Not Applicable: JPEG

Supported Values: 0: disable, 1: enable

Default: 0

customizeQpMapFile

Specifies the name of the qp map file. It will take effect when ROI is enabled (**roiEnable**=1) and customizeQpLevel is not 0. customizeQpMap is currently only supported on FFmpeg 3.4.2 and above. It supports qp level from 1 to 10, supports qp value from 0 to 63. The qp map file must be created in the same directory where the command is running. The file contains one line or multiple lines of qp value map
NOTE – 1. ROI takes no effects for 2-pass (lookaheadDepth > 0) encode.

Applicable: FFmpeg 3.4.2 and above
Not Applicable: JPEG

enableCompensateQp

It only takes effect when maxframesize > 0. XCoder firmware will record the previous frame first qp and ssim when previous frame trigger re-encode because of maxframesize. And re-encode with lower qp for next frame if its ssim different from last time is greater than the threshold
NOTE – After re-encode, it will keep frame size < maxframesize, and if maxframesize is too low, it will re-encode repeated, fps will drop and psnr won't improve

Not Applicable: JPEG and AV1

Supported Values:

0: Disable
1: Enable

Default: 0

8.4.1 Long Term Reference

The Libxcoder encoder configuration parameter **longTermReferenceEnable** is used to enable the Long Term Reference (LTR) feature when opening an encoder instance.

The Long Term Reference is enabled by setting **longTermReferenceEnable=1.**

When **longTermReferenceEnable**=1, the IDR frames are automatically set as LTR (regardless of **longTermReferenceInterval** or other settings).

When **longTermReferenceEnable**=1, the user can set the LTR by either one or both of these methods:

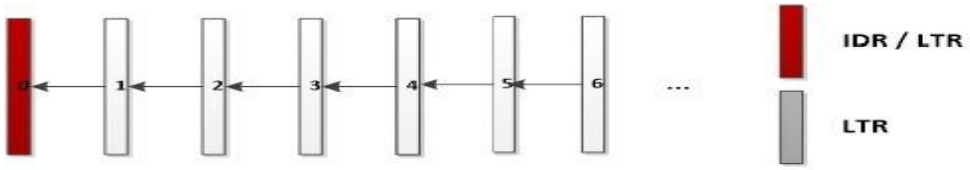
1. Set the LTR interval at configuration by **longTermReferenceInterval**, and/or change LTR interval during run-time by **libxcoder API**

2. Set any frame as LTR during runt-time by **libxcoder API**

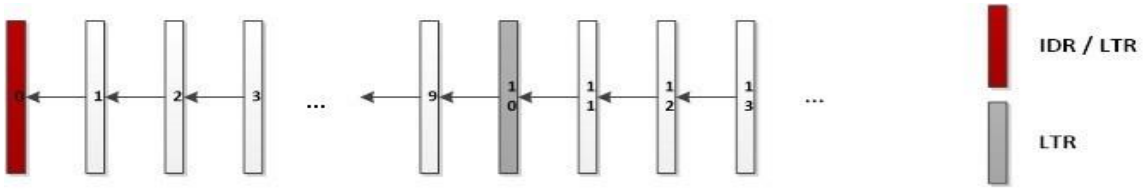
For more details regarding libxcoder API integration, please refer to the Libxcoder API Integration Guide, or **Application Note APPS528 - Long Term Reference & Reference Invalidation Application Note**

When the **longTermReferenceEnable**=1, the user can also set the number of LTRs (1 or 2) using the l**ongTermReferenceCount** parameter**.**

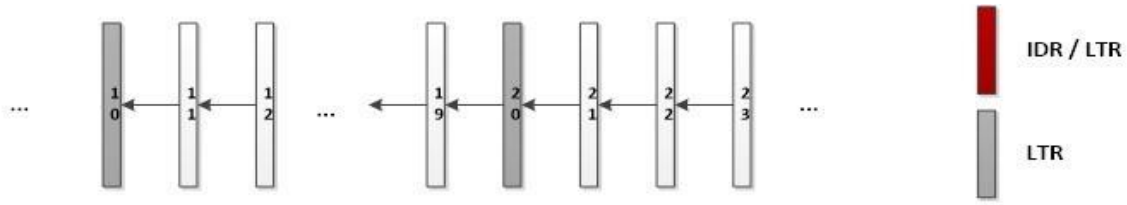
Example 1: longTermReferenceEnable=1 or longTermReferenceEnable=1:longTermReferenceInterval=0 will cause the IDR frame to be a long term reference frame.



Example 2:
longTermReferenceEnable=1:longTermReferenceInterval=10:longTermReferenceCount=2 will cause the IDR frame to be the long term reference frame, and every 10th frame to also be a long term reference frame.



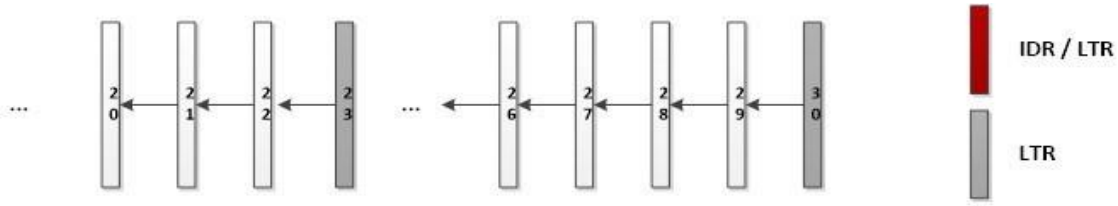
Up to 2 LTPs to be preserved – E.g. Frame 20 replaces the oldest LTR frame 0, so that frame 10 and frame 20 are the current LTR frames.



Example 3:

longTermReferenceEnable=1:longTermReferanceInterval=10:longTermReferenceCount=2, using the libxcoder API and setting frame 23 as LTR will cause up to 2 LTRs to be preserved – e.g frame 23 replaces the oldest LTR frame 10, so that frame 20 and frame 23 are the current LTR frames.

The next LTR will be frame 30 at LTR interval, replacing the oldest LTR frame 20, so that frame 23 and frame 30 are the current LTR frames.



NOTE - I-Frame clears all existing LTRs and resets LTR interval counter. I-Frame itself is automatically set as the new LTR.

For example, if longTermReferenceInterval = 4, longTermReferenceCount = 2, intraPeriod = 15

- Frame 15 will be set as LTR1, Frame 19 will be set as LTR2

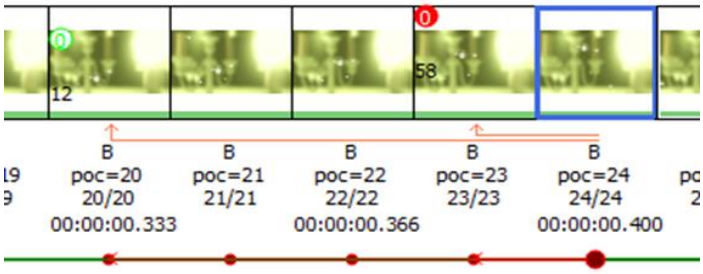
8.4.2 Reference Invalidation

Reference invalidation is used by the receiver of the bitstream to invalidate and change the encoding frames' references when the received packet is corrupted (and therefore cannot be referenced).

Since the need to invalidate reference only arises when the receiving end detects corrupted frame during run-time, reference invalidation requires **libxcoder API** integration.

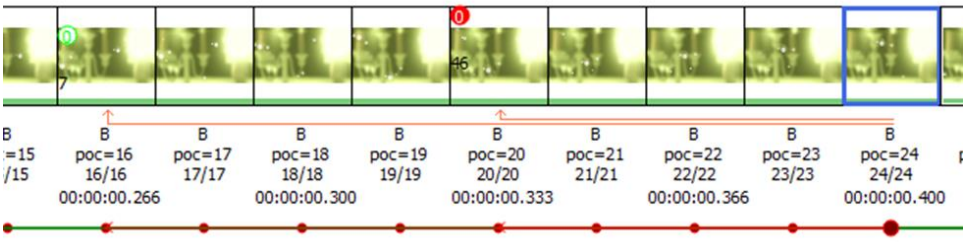
For more details regarding libxcoder API integration, please refer to Libxcoder API Integration Guide, or Application Note APPS528 - Long Term Reference & Reference Invalidation Application Note

Reference Invalidation is illustrated below. Without reference invalidation Frame 24 refers to Frames 23 and 20:



In the following example, encoding Frame 24 invalidates all references with frame number >= 21. Since reference frame 23 is invalidated, Frame 24 now refers to Frames 20 and 16

NOTE – The frame number should be incremented for every frame sent to the encoder (regardless of whether the encoder drops the output due to picSkip / MaxFrameSize / MaxFrameSize_Bytes or MaxFrameSize_Bits). Also note that unlike poc, the frame number should not be reset to 0 upon I-frame.



8.4.3 Gop Pattern Settings

Quadra supports three ways to set the gop patterns. The first method is the default method, where gopPresetIdx = -1 or is not set. By default, encoder uses Adaptive Gop, for which the encoder dynamically adjusts gop pattern while encoding. The second method is setting custom gop structures via xcoder-gop parameters, where gopPresetIdx must be set to 0. The third method is the preset gop structure, where gopPresetIdx can be set from 1 to 10.

Now we introduce the custom gop structure and the predefined gop structures in detail. Some commonly used gop patterns are shown below as examples.

8.4.3.1 Custom Gop Structure

The following table lists the custom gop parameters, xcoder-gop.

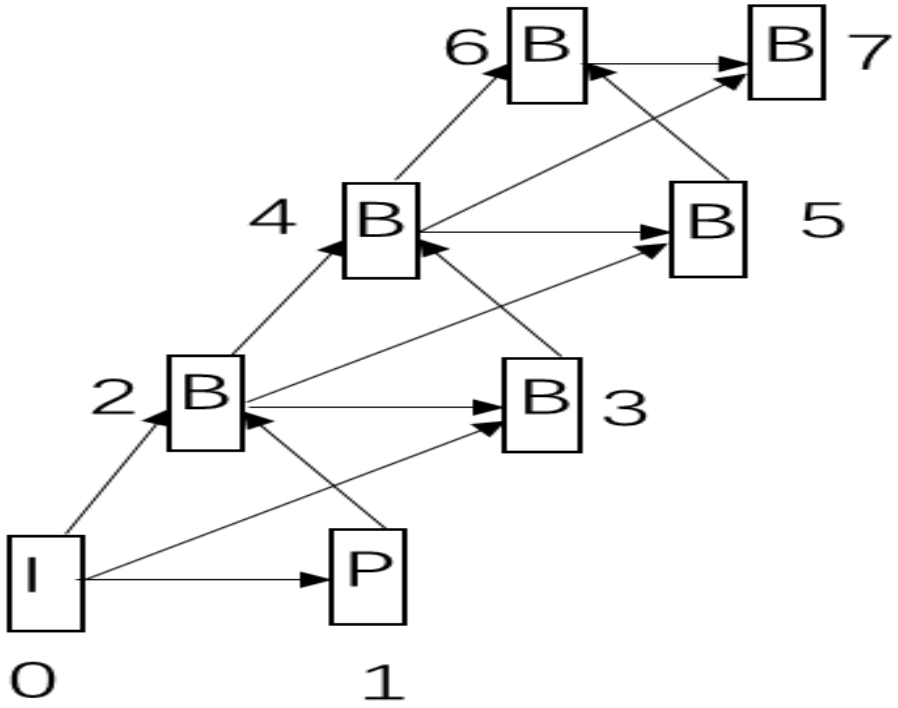
NOTE - The order of the custom GOP parameters does not matter. E.g. g2numRefPics can be placed before g0numRefPics

Quadra Custom Gop Parameters (xcoder-gop)

Parameter	Values	Description
customGopSize	1 to 8	Specifies size of custom GOP pattern. **NOTE** - in following frame parameters, prefix g0 means this setting corresponds to the 1st frame in the gop structure, g1 corresponds to the 2nd frame in gop structure, and so on **NOTE** - Number of frames in gop structure (e.g. g0, g1, …) must match **customGopSize**
Frame parameters		
g0pocOffset	1 to gop size	POC (display order) of the frame within a GOP, ranging from 1 to gop size
g0QpOffset	-51 to 51	Delta QP, will be added to the frame QP to set the final QP
g0temporalId	0 to highest temporal layer in the gop	temporal layer ID
g0picType	1: P frame 2: B frame	Frame type, can be either P or B frame
g0numRefPics	1 to gop size	Number of reference pictures kept for this frame, including references for current and future pictures. **NOTE** - in the following Reference List parameters, prefix g0refPic0 means the setting corresponds to 1st reference frame, g0refPic1 corresponds to 2nd reference frame, and so on **NOTE** – if a reference frame will be referenced by future frame(s) but is not referenced by the current frame, it would need to be included in current frame reference list as well. For example, in hierarchical-P GOP pattern (see Custom GOP Example 3), Frame 3 only references Frame 2, but its reference list must also include Frame 0 for Frame 4 to reference; therefore g2numRefPics=2 **NOTE** - Number of Reference List parameters must match **g0numRefPics** **NOTE** - The maximum number of reference frames referenced by the current picture is 2

Reference List parameters		
g0refPic0	- (gop size) to gop size 0 is not allowed	Delta POC of the reference picture, relative to the POC of the current frame **NOTE** - g0refPic0 cannot be 0, since a picture cannot refer to itself **NOTE** – the **index** of reference frames (E.g. g0refPic**0**, g0refPic**1**, etc.) in the reference list must follow order described below 1. Forward references (past) must precede Backward references (future) 2. Forward references (past) must be in poc order High to Low 3. Backward references (future) must be in poc order Low to High The purpose is to assign lower indices to reference frames closet to the current frame in display order, because these are the reference frames which are most often used for reference, and lower reference index helps to reduce bit count For example, in hierarchical-P GOP pattern (see Custom GOP Example 3), Frame 3 reference list must assign index 0 (g2refPic0) to Frame 2 (g2refPic0=-1), and assign index 1 (g2refPic1) to Frame 0 (g2refPic1=-3)
g0refPic0Used	0: Not referenced by current frame 1: Referenced by current frame	Specifies whether each reference frame in reference list is used in current (1) or future (0)

Custom GOP Example 1 (GOP size 2, low delay):



ffmpeg -y -f rawvideo -pix_fmt yuv420p -s:v 1440x1080 -r 24 -i input1440x1080.yuv -c:v h264_ni_quadra_enc -vframes 50 -xcoder-params 'profile=2:level=6:RcEnable=1:bitrate=10000000:intraPeriod=0:gopPresetIdx=0' -xcoder-gop "customGopSize=2:g0pocOffset=1:g0QpOffset=0:g0temporalId=0:g0picType=2:g0numRefPics=2:g0refPic0=-1:g0refPic0Used=1:g0refPic1=-3:g0refPic1Used=1:g1pocOffset=2:g1QpOffset=0:g1temporalId=0:g1picType=2:g1numRefPics=2:g1refPic0=-1:g1refPic0Used=1:g1refPic1=-2:g1refPic1Used=1" -enc 0 output1440x1080.h264

Custom GOP Example 2 (Hierarchical-B with GOP size 4):

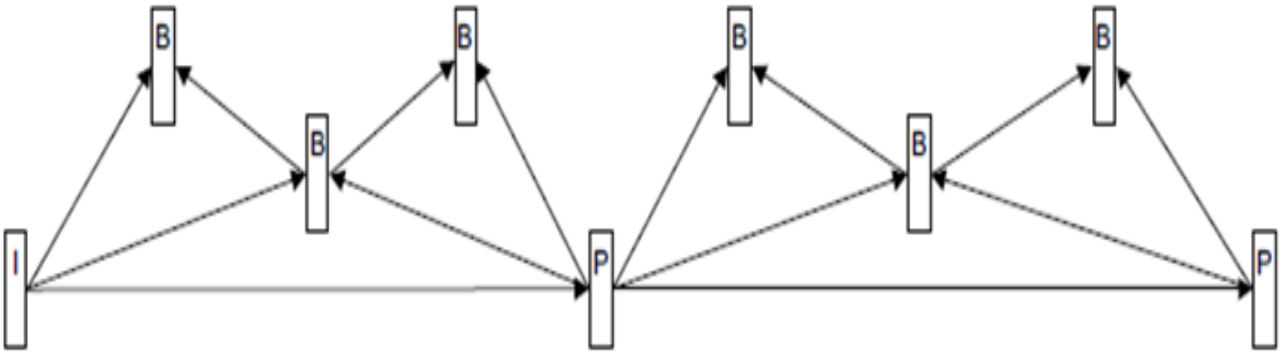


Figure 1. Hierarchical-B With Gop-Size = 4

ffmpeg -y -f rawvideo -pix_fmt yuv420p -s:v 1440x1080 -r 24 -i input1440x1080.yuv -c:v h264_ni_quadra_enc -vframes 50 -xcoder-params 'profile=2:level=6:RcEnable=1:bitrate=10000000:intraPeriod=0:gopPresetIdx=0' -xcoder-gop "customGopSize=4:g0pocOffset=4:g0QpOffset=0:g0temporalId=0:g0picType=1:g0numRefPics=1:g0refPic0=-4:g0refPic0Used=1:g1pocOffset=2:g1QpOffset=0:g1temporalId=0:g1picType=2:g1numRefPics=2:g1refPic0=-2:g1refPic0Used=1:g1refPic1=2:g1refPic1Used=1:g2pocOffset=1:g2QpOffset=0:g2temporalId=0:g2picType=2:g2numRefPics=3:g2refPic0=-1:g2refPic0Used=1:g2refPic1=1:g2refPic1Used=1:g2refPic2=3:g2refPic2Used=0:g3pocOffset=3:g3QpOffset=0:g3temporalId=0:g3picType=2:g3numRefPics=2:g3refPic0=-1:g3refPic0Used=1:g3refPic1=1:g3refPic1Used=1" -enc 0 output1440x1080.h264

Custom GOP Example 3 (Hierarchical-P with GOP size 4):

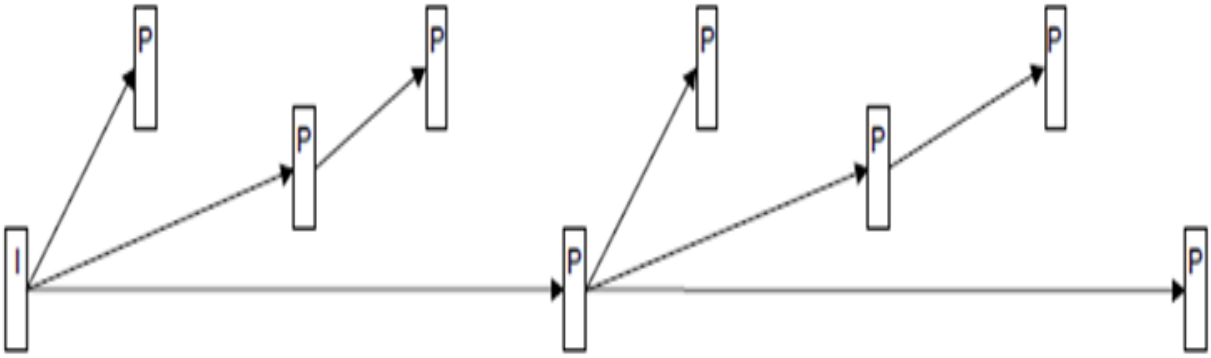


Figure 2. Hierarchical-P With Gop-Size = 4

ffmpeg -y -f rawvideo -pix_fmt yuv420p -s:v 1440x1080 -r 24 -i input1440x1080.yuv -c:v h264_ni_quadra_enc -vframes 50 -xcoder-params 'profile=2:level=6:RcEnable=1:bitrate=10000000:intraPeriod=0:gopPresetIdx=0' -xcoder-gop " customGopSize=4:g0pocOffset=1:g0QpOffset=0:g0temporalId=0:g0picType=1:g0numRefPics=1 :g0refPic0=-1:g0refPic0Used=1:g1pocOffset=2:g1QpOffset=0:g1temporalId=0:g1picType=1:g1numRefPics= 1:g1refPic0=-2:g1refPic0Used=1:g2pocOffset=3:g2QpOffset=0:g2temporalId=0:g2picType=1:g2numRefPics= 2:g2refPic0=-1:g2refPic0Used=1:g2refPic1=-3:g2refPic1Used=0:g3pocOffset=4:g3QpOffset=0:g3temporalId=0:g3picType=1:g3numRefPics= 1:g3refPic0=-4:g3refPic0Used=1" -enc 0 output1440x1080.h264

8.4.3.2 Pre-defined GOP Structure

The gopPresetIdx are pre-configured gop patterns that fit most applications without the complexity of configuring a custom gop.

According to the different values set for the **gopPresetIdx**, **lookAheadDepth**, and **enable2PassGop** parameters, the gop structure can be divided into two types:

1. 1-pass, lookaheadDepth = 0, enable2PassGop takes no effects, the gop structure is determined by gopPresetIdx.
2. 2-pass, lookaheadDepth > 0, the gop structure is determined by gopPresetIdx and enable2PassGop.

Here are the predefined GOP structures for H.264 / H.265 / AV1.

Preset GOP Patterns for AVC / HEVC / AV1 1-pass encode (lookaheadDepth = 0)

NOTE – temporal ID field only takes effects when parameter* **temporalLayersEnable *is set to 1. When temporalLayersEnable is not specified (default 0), all frames are assigned temporal ID 0*

gopPresetIdx	frame	Type	POC	QP offset	num of ref	Reference List	used_by cur	temporal ID *
-1	Adaptive Gop(default)							
0	Custom Gop							
1	1	I	1	1	0	X	X	0
2	obsolete							
3	1	B	1	1	2	-1 -2	1	0
4	1	P	2	1	1	-2	1	0
	2	B	1	3	2	-1 1	1 1	1
5	1	P	4	1	1	-4	1	0
	2	B	2	3	2	-2 2	1 1	1
	3	B	1	5	3	-1 1 3	1 1 0	2
	4	B	3	5	2	-1 1	1 1	2
6	obsolete							
7	1	B	1	5	2	-1 -5	1 1	0
	2	B	2	3	2	-1 -2	1 1	0
	3	B	3	5	2	-1 -3	1 1	0
	4	B	4	1	2	-1 -4	1 1	0
8	1	B	8	1	2	-8 -16	1 1	0
	2	B	4	3	2	-4 4	1 1	1
	3	B	2	5	3	-2 2 6	1 1 0	2
	4	B	1	7	4	-1 1 3 7	1 1 0 0	3
	5	B	3	7	4	-1 -3 1 5	1 0 1 0	3
	6	B	6	5	3	-2 -6 2	1 0 1	2
	7	B	5	7	4	-1 -5 1 3	1 0 1 0	3
	8	B	7	7	3	-1 -7 1	1 0 1	3
9	1	P	1	1	1	-1	1	0
10	1	P	1	5	1	-1	1	2
	2	P	2	3	1	-2	1	1
	3	P	3	5	2	-1 -3	1 0	2
	4	P	4	1	1	-4	1	0

Preset GOP Patterns for AVC / HEVC / AV1 2-pass encode (lookaheadDepth > 0 and enable2PassGop = 0)

NOTE – temporal ID field only takes effects when parameter* **temporalLayersEnable *is set to 1. When temporalLayersEnable is not specified (default 0), all frames are assigned temporal ID 0*

*NOTE – 2-pass encode algorithm adjusts QP. Therefore, picQp may not be reflected in coded frame

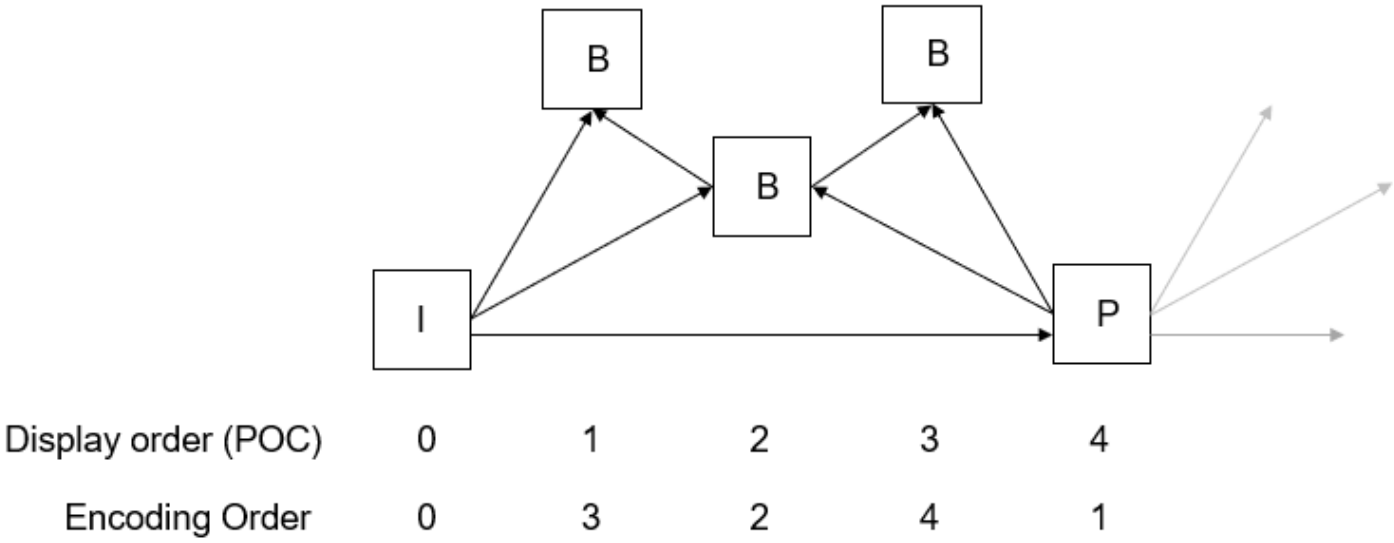
gopPresetIdx	frame	Type	POC	QP offset	num of ref	Reference List	used_by cur	temporal ID *
-1	Adaptive Gop(default)							
0	Custom Gop							
1	Not supported for 2-pass							
2	obsolete							
3	Not supported for 2-pass							
4	1	B	2	1	2	-2 -4	1 1	0
	2	B	1	3	2	-1 1	1 1	1
5	1	B	4	1	2	-4 -8	1 1	0
	2	B	2	3	2	-2 2	1 1	1
	3	B	1	5	3	-1 1 3	1 1 0	2
	4	B	3	5	3	-1 -3 1	1 0 1	2
6	obsolete							
7	Not supported for 2-pass							
8	1	B	8	1	2	-8 -16	1 1	0
	2	B	4	3	2	-4 4	1 1	1
	3	B	2	5	3	-2 2 6	1 1 0	2
	4	B	1	7	4	-1 1 3 7	1 1 0 0	3
	5	B	3	7	4	-1 -3 1 5	1 0 1 0	3
	6	B	6	5	3	-2 -6 2	1 0 1	2
	7	B	5	7	4	-1 -5 1 3	1 0 1 0	3
	8	B	7	7	3	-1 -7 1	1 0 1	3
9	1	P	1	1	1	-1	1	0
10	Not supported for 2-pass							

Preset GOP Patterns for AVC / HEVC / AV1 2-pass encode (lookaheadDepth > 0 and enable2PassGop=1)

gopPresetIdx	frame	Type	POC	QP offset	num of ref	Reference List	used_by cur	temporal ID *
-1	Adaptive Gop(default)							
0	Custom Gop							
1	Not supported for 2-pass							
2	obsolete							
3	Not supported for 2-pass							
4	1	P	2	1	1	-2	1	0
	2	B	1	3	2	-1 1	1 1	1
5	1	P	4	1	1	-4	1	0
	2	B	2	3	2	-2 2	1 1	1
	3	B	1	5	3	-1 1 3	1 1 0	2
	4	B	3	5	2	-1 1	1 1	2
6	obsolete							
7	Not supported for 2-pass							
8	1	P	8	1	1	-8	1	0
	2	B	4	3	2	-4 4	1 1	1
	3	B	2	5	3	-2 2 6	1 1 0	2
	4	B	1	7	4	-1 1 3 7	1 1 0 0	3
	5	B	3	7	3	-1 1 5	1 1 0	3
	6	B	6	5	2	-2 2	1 1	2
	7	B	5	7	3	-1 1 3	1 1 0	3
	8	B	7	7	2	-1 1	1 1	3
9	1	P	1	1	1	-1	1	0
10	Not supported for 2-pass							

8.4.3.3 Description of GOP Patterns

The following is the gop structure for when gopPresetIdx=5, lookaheadDepth = 0 or for when gopPresetIdx = 5, lookahead > 0, enable2Pass=1.



The four frames in this gop are described below in decoding order.

- Frame 1 is a P frame with POC 4, referencing one frame with POC 0. The reference frame is defined by delta POC value relative to current frame. In this case it is -4.
- Frame 2 is a B frame with POC 2, referencing two frames with POC 0 and 4 respectively. So its reference frames are listed as -2 and 2.
- Frame 3 is a B frame with POC 1, referencing two frames with POC 0 and 2 respectively. It also needs to keep the frame with POC 4 as a reference frame to be used in future. So its reference frame list is -1, 1 and 3.
- Frame 4 is a B frame with POC 3, referencing two frames with POC 2 and 4 respectively. Its reference list is -1 and 1.

The corresponding GOP structure table is show below, where:

- **QP Offset** will be added to the QP parameter to set the final QP;
- **Used by Current Frame** specifies whether each reference frame in Reference List is used in current(1) or future(0).

Frame	Type	POC	QP Offset	Number of References	Reference List	used_by_cur
1	P	4	1	1	-4	1
2	B	2	3	2	-2 2	1 1
3	B	1	5	3	-1 1 3	1 1 0
4	B	3	5	2	-1 1	1 1

The diagrams of all gop patterns for 1 pass (lookAheadDepth=0) are shown below. P and B pictures can have one to multiple reference frames as illustrated by the arrows. The inter frame immediately after I-frame is encoded as P-frame, although it appears in the table as B-frame because only one reference frame is available:

Diagram of gopPresetIdx=1

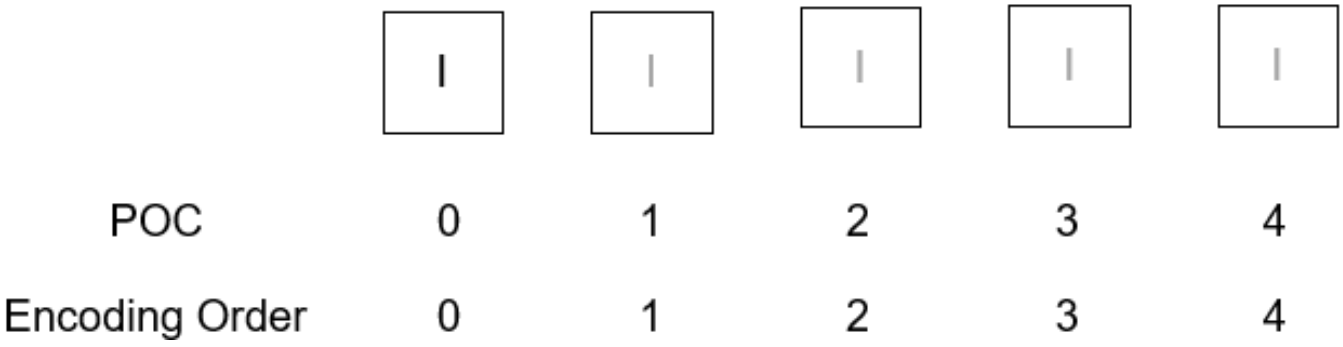


Diagram of gopPresetIdx=3

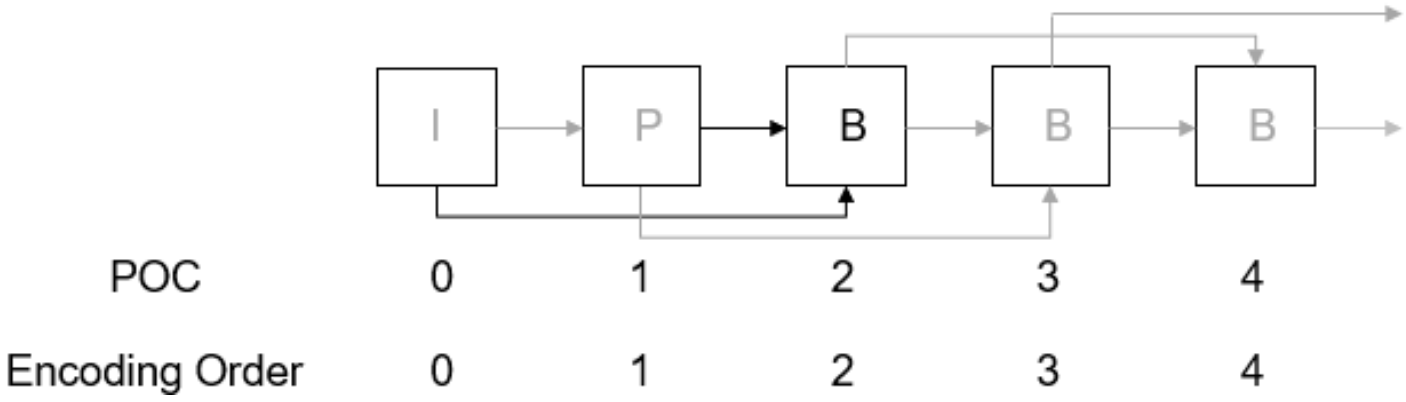
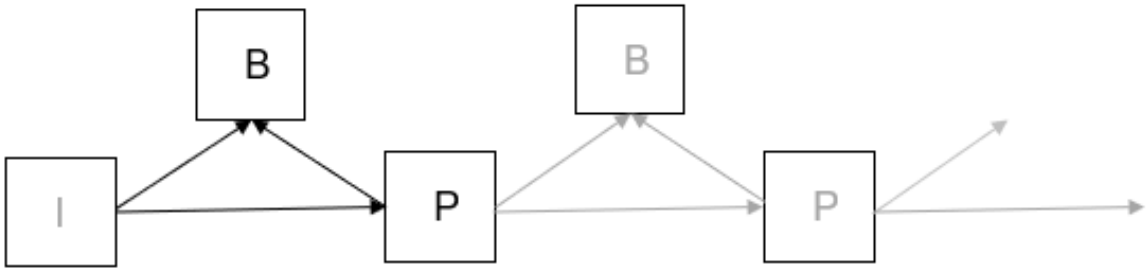
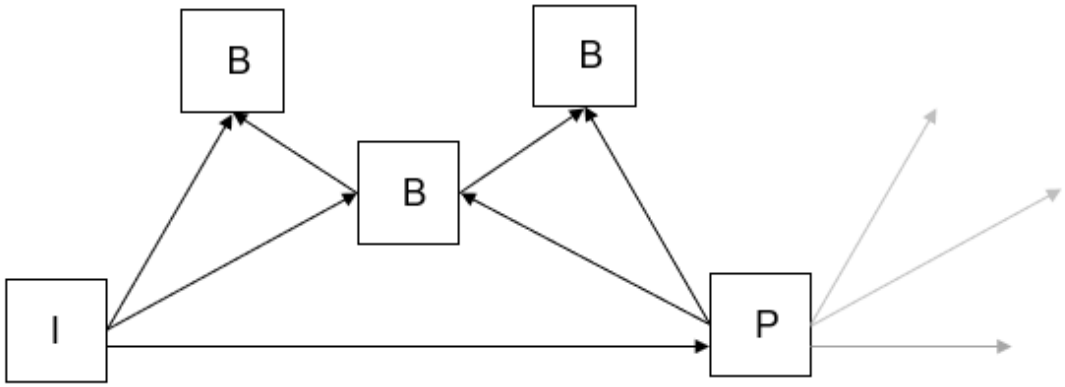


Diagram of gopPresetIdx=4



POC	0	1	2	3	4
Encoding Order	0	2	1	4	3

Diagram of gopPresetIdx=5



Display order (POC)	0	1	2	3	4
Encoding Order	0	3	2	4	1

Diagram of gopPresetIdx=7

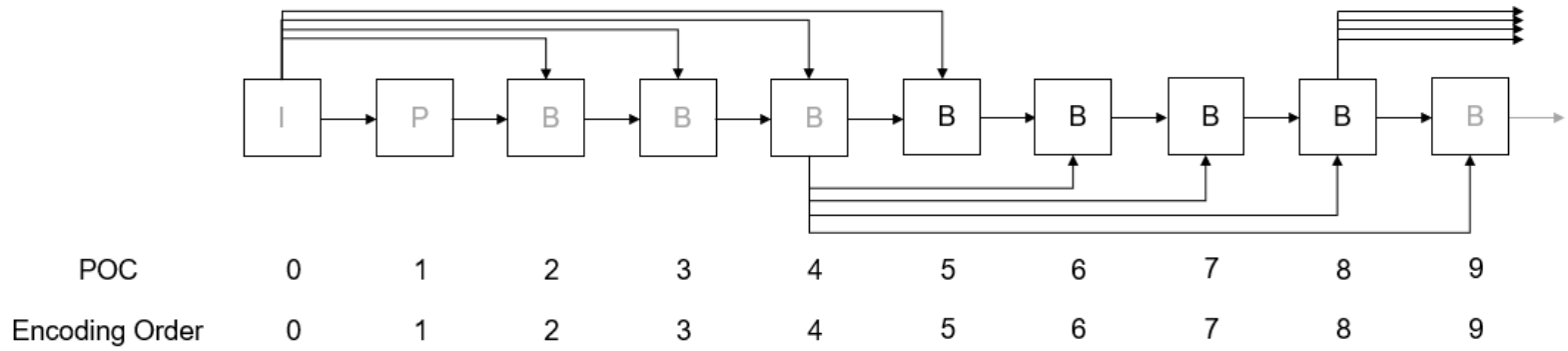


Diagram of gopPresetIdx=8

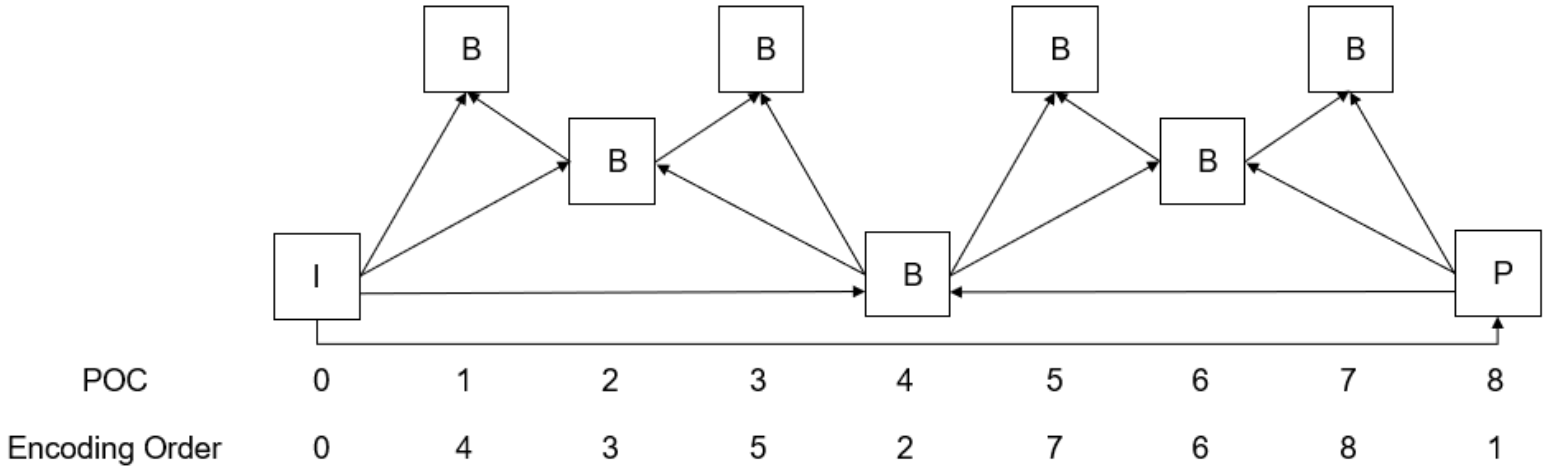
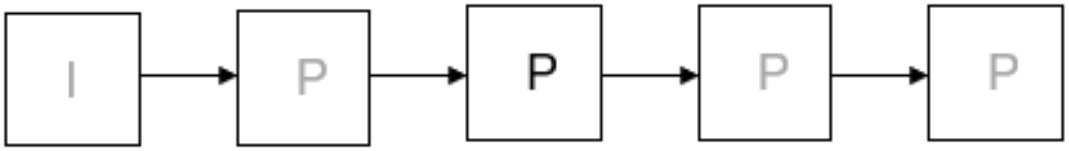
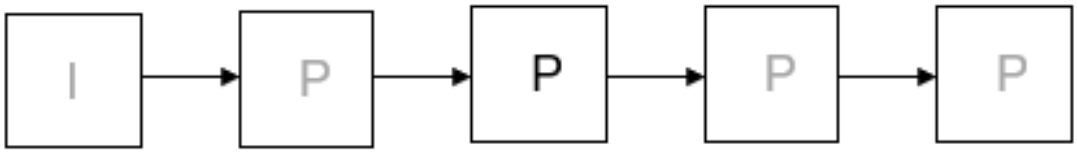


Diagram of gopPresetIdx=9



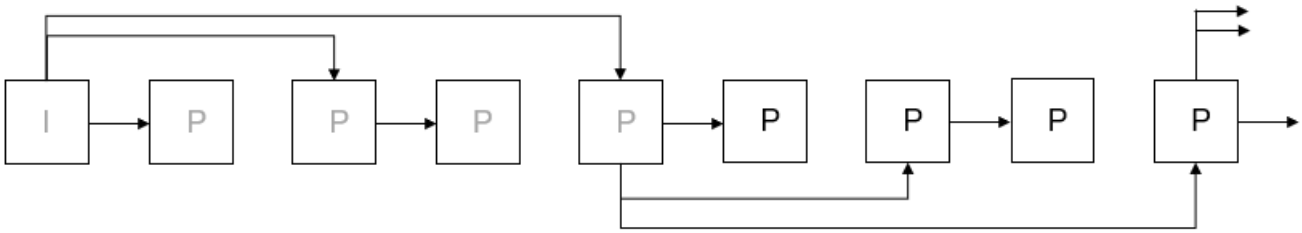
POC	0	1	2	3	4
Encoding Order	0	1	2	3	4

Note that frame number count is relative to adjacent frames. For example, in the gopPresetIdx=9 diagram above, if the user sees encoding order frame number 2 as frame 0, the adjacent frame numbers should change accordingly. The following diagram shows the case when the user sees encoding order frame number 2 as 0 in the diagram above. This helps to better interpret the indexes on the table.



POC	-2	-1	0	1	2
Encoding Order	-2	-1	0	1	2

Diagram of gopPresetIdx=10



POC	0	1	2	3	4	5	6	7	8
Encoding Order	0	1	2	3	4	5	6	7	8

The diagrams of gop patterns for 2 pass (2-pass & enable2PassGop=0) are shown below:

Diagram of gopPresetIdx=4 for 2 pass

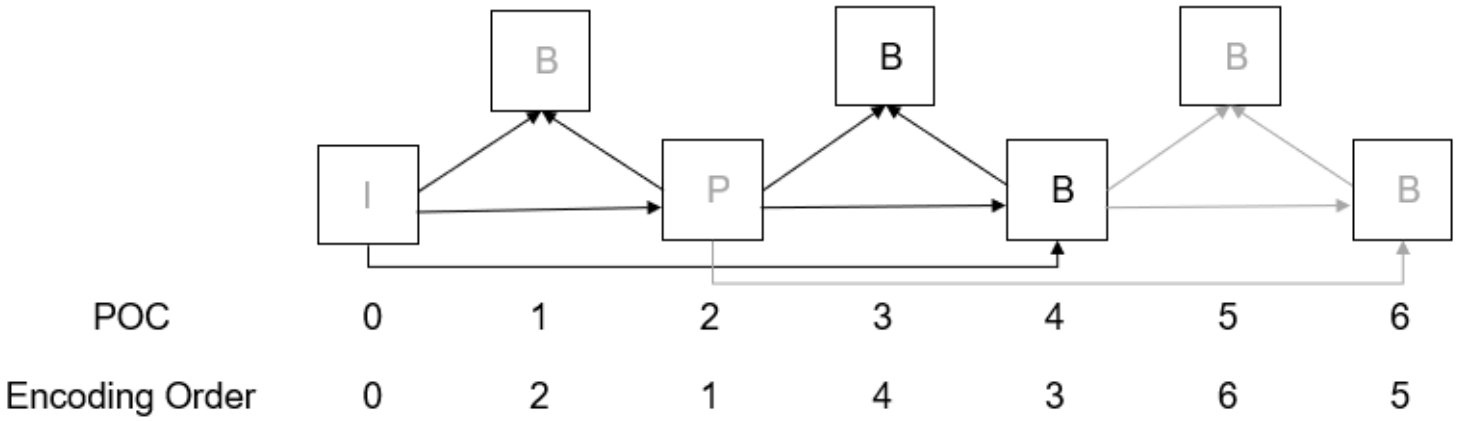


Diagram of gopPresetIdx=5 for 2 pass

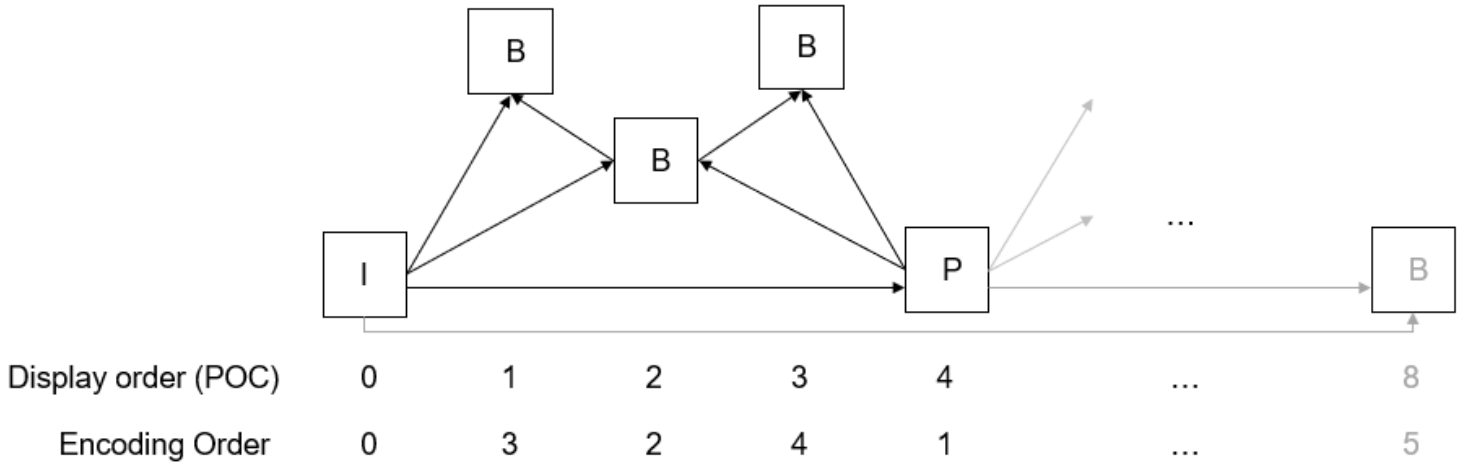
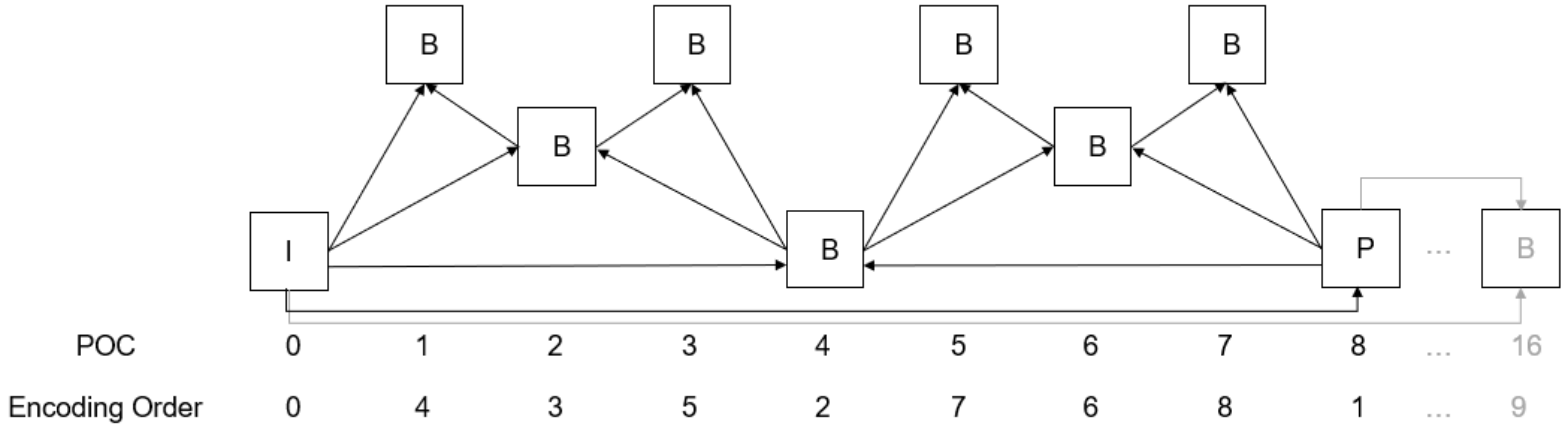


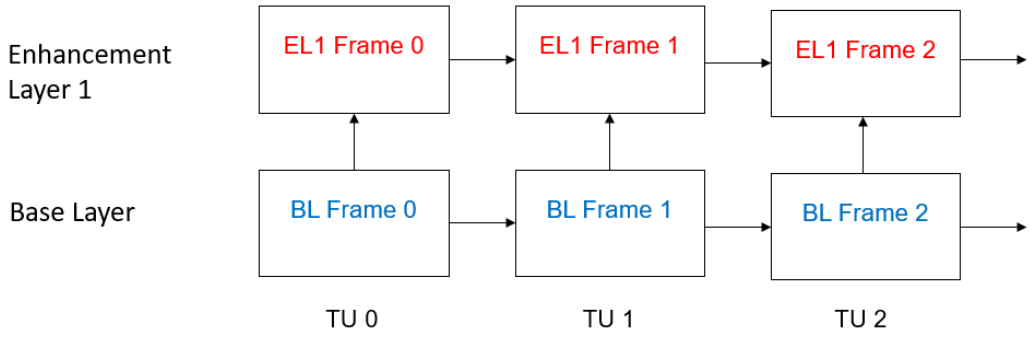
Diagram of gopPresetIdx=8 for 2 pass



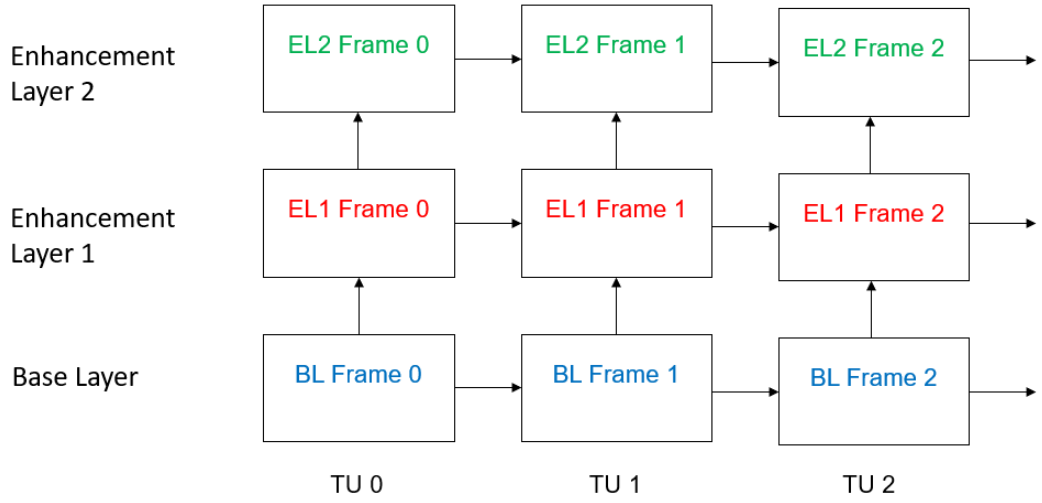
8.4.3.4 AV1 Spatial Layers Gop Patterns

The following charts illustrate the gop structures for AV1 spatial scalability, enabled by **spatialLayers** (and optional **spatialLayersRefBaseLayer**) parameters

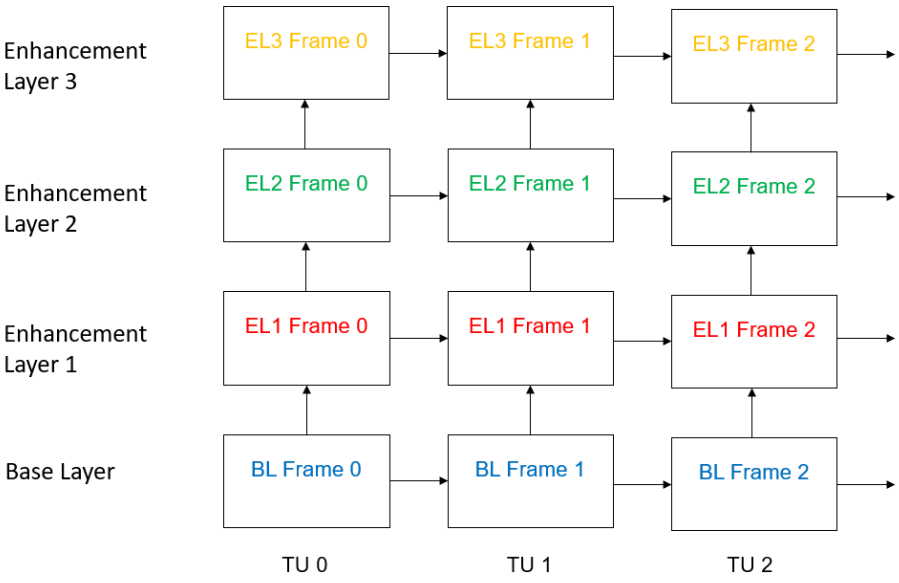
2 spatial layers (spatialLayers=2)



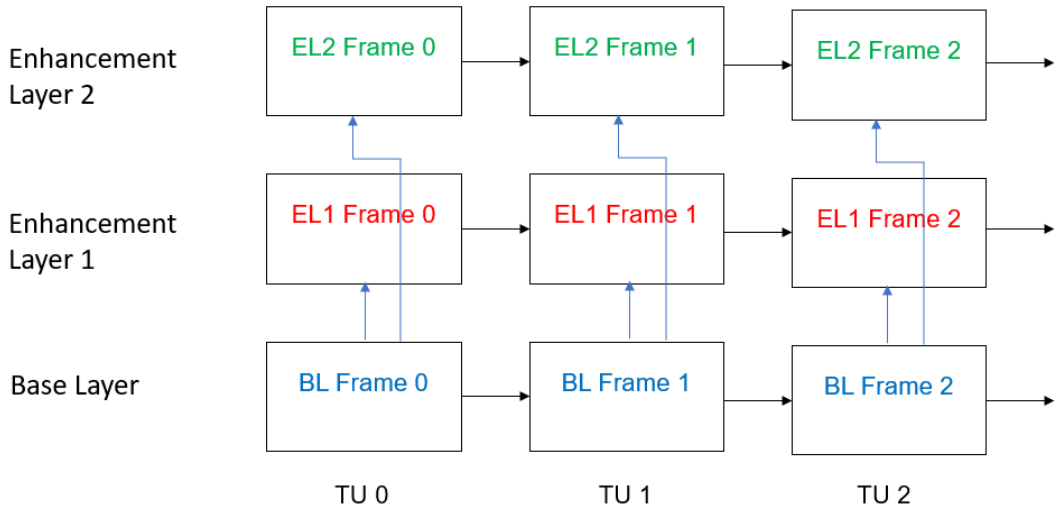
3 spatial layers (spatialLayers=3)



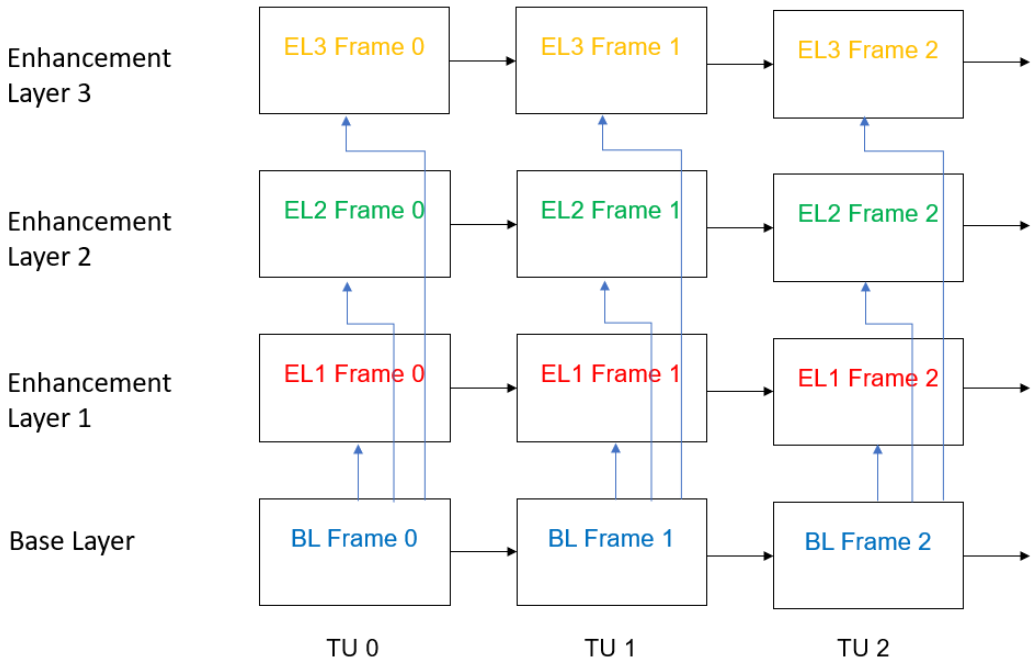
4 spatial layers (spatialLayers=4)



3 spatial layers, with each EL referencing BL (spatialLayers=3, spatialLayersRefBaseLayer=1)



4 spatial layers, with each EL referencing BL (spatialLayers=4, spatialLayersRefBaseLayer=1)



8.4.4 CRF & Capped CRF Descriptions & Examples

CRF: Constant Rate Factor Mode, enabled by setting the rate factor parameter **crf**. With CRF the encoder varies the bitrate to maintain constant subjective quality.

CRF Sample Command

```
ffmpeg -f rawvideo -pix_fmt yuv420p -s 1920x1080 -r 30.0 -i input1920x1080.yuv -c:v h265_ni_quadra_enc -xcoder-params "gopPresetIdx=5:RcEnable=0:crf=23:lookAheadDepth=10" output1920x1080.h265
```

Capped CRF: Capped Constant Rate Factor Mode is enabled by setting the rate factor parameter **crf** together with **bitrate, vbvBufferSize, vbvMaxRate** (optional), **vbvMinRate** (optional). Capped CRF combines the CRF mode with a maximum bitrate limit or caprate. In this approach, CRF is used to specify the target quality level, while the caprate prevents the average bitrate from exceeding the target bitrate. This combination is particularly useful in scenarios where a specific quality level is desired and it's also crucial to constraint bitrate to meet bandwidth or storage requirements. Please also note that in Capped CRF mode, if encoder is restricted by caprate and cannot produce enough bits, it may not be able to maintain the quality level specified by the **crf** parameter. The following descriptions provide more details about these parameters -

- **crf** determines the quality of encoded frames; generally speaking, the better the quality, the more bits each frame will consume. These bits cause the encoder buffer level to rise and can be considered as the "input rate" of the encoder buffer. Please note the quality will be limited by caprate, therefore setting low **crf** value (high quality) with low **bitrate** or **vbvMaxRate** is not recommended

- **bitrate** or **vbvMaxRate** is the caprate. If **vbvMaxRate** is not specified, encoder will take **bitrate** as the caprate. If both **bitrate** and **vbvMaxRate** are specified, the encoder will instead take **vbvMaxRate** as the caprate. The caprate is also the "draining rate" of the encoder buffer; the higher the caprate, the faster the encoder buffer is drained. If user sets low **crf** value (high quality & high input rate) together with low caprate (low draining rate), the encoder will need to suppress bits and may not be able to maintain the high quality level specified by the user

- **vbvBufferSize** determines the buffer size. The buffer size determines "tolerance" of peak rate. While **bitrate** or **vbvMaxRate** determines the maximum average bitrate, the instantaneous peak rate is decided by both the caprate and buffer size, e.g. peak rate <= caprate + buffer size. Please note that **bitrate** is required even if **vbvMaxRate** is already specified, because encoder needs both bitrate and vbvBufferSize parameters to calculate the actual buffer size: buffer size (bits) = bitrate (bps) * vbvBufferSize (ms) / 1000

Besides the maximum bitrate limit, the user may also set a minimum bitrate limit, so that encoder will produce more bits to achieve higher quality in low volatile scenes which would otherwise have produced bits less than the minimum bitrate limit.

Note that in Capped CRF mode, although the encoder attempts to maintain consistent subjective quality, it is also required to adjust quality to meet the bitrate limit, and therefore consistent quality is no longer guaranteed. More details -

- Capped CRF requires **bitrate** and **vbvBufferSize**, these parameters together define the size of the VBV buffer, which is used to constrain bitrate. Please note that **bitrate** is required even if **vbvMaxRate** is already specified, because encoder needs both **bitrate** and **vbvBufferSize** parameters to calculate the actual buffer size: buffer size (bits) = bitrate (bps) * vbvBufferSize (ms) / 1000
- If **vbvMaxRate** is not set, encoder will take **bitrate** as the maximum bitrate limit. If **vbvMaxRate** is set, the encoder will take **vbvMaxRate** as the maximum bitrate limit instead.
- If **vbvMinRate** is not set, the encoder will not constrain bitrate by a minimum bitrate limit. If **vbvMinRate** is set, the encoder will produce more bits to meet the minimum bitrate, which may also raise quality to be higher than the specified consistent quality level (which depends on **crf** parameter value) when bits required at specified quality level is lower than the minimum bitrate.
- Please also refer to **bitrate**, **vbvBufferSize, vbvMaxRate, vbvMinRate** descriptions in Section 8.4 “Encoding Parameters”

Capped CRF Sample Command (without vbvMaxRate and vbvMinRate)

```
ffmpeg -f rawvideo -pix_fmt yuv420p -s 1920x1080 -r 30.0 -i input1920x1080.yuv -c:v h265_ni_quadra_enc -xcoder-params "gopPresetIdx=5:RcEnable=0:crf=23:lookAheadDepth=10:vbvBufferSize=1000:bitrate=3200000" output1920x1080.h265
```

Capped CRF Sample Command (with vbvMaxRate)

```
ffmpeg -f rawvideo -pix_fmt yuv420p -s 1920x1080 -r 30.0 -i input1920x1080.yuv -c:v h265_ni_quadra_enc -xcoder-params "gopPresetIdx=5:RcEnable=0:crf=23:lookAheadDepth=10:vbvBufferSize=1000:bitrate=3200000:vbvMaxRate=4000000" output1920x1080.h265
```

Capped CRF Sample Command (with vbvMinRate)

```
ffmpeg -f rawvideo -pix_fmt yuv420p -s 1920x1080 -r 30.0 -i input1920x1080.yuv -c:v h265_ni_quadra_enc -xcoder-params "gopPresetIdx=5:RcEnable=0:crf=23:lookAheadDepth=10:vbvBufferSize=1000:bitrate=3200000:vbvMinRate=1000000" output1920x1080.h265
```

Capped CRF Sample Command (with vbvMaxRate and vbvMinRate)

```
ffmpeg -f rawvideo -pix_fmt yuv420p -s 1920x1080 -r 30.0 -i input1920x1080.yuv -c:v h265_ni_quadra_enc -xcoder-params "gopPresetIdx=5:RcEnable=0:crf=23:lookAheadDepth=10:vbvBufferSize=1000:bitrate=3200000:vbvMaxRate=4000000:vbvMinRate=1000000" output1920x1080.h265
```

8.4.5 Encoder Limitations

General Restrictions:

Horizontal stride for luma and chroma needs to be 128-byte aligned, if not, the input needs to be padded until it meets the requirement. Height needs to be even.

H.265 and AV1 2-Pass encode have the following restrictions:

1. If the input resolution width and height are not both aligned to 32-pixels, H.265 and AV1 2-Pass encode output may vary (encode output may mismatch)

AV1 encode has the following restrictions:

1. Requires FFmpeg version 4.0 or above
2. HW limitation:
    - Due to Quadra HW limitation, AV1 non-8x8-aligned input resolution will be cropped
    - If width is not 8-pixel aligned, width will be cropped
        - E.g. Input resolution 854x480 -> output resolution will be cropped to 848x480
    - If height is not 8-pixel aligned, height will be cropped
        - E.g. Input resolution 960x540 -> output resolution will be cropped to 960x536
3. Input resolution restriction:

    Minimum resolution supported by encoder

    - width >= 144 pixels
    - height >= 128 pixels
        - If the input resolution is less than the minimum resolution, the encoder will encode according to the minimum resolution and also encode the crop information to ensure that when decode the bitstream encoded by encoder, the decoded YUV resolution is the same as the input resolution.

    Maximum resolution supported by encoder

    - width <= 4096 pixels
    - height <= 4352 pixels
    - width * height <= (4096 * 2304) pixels

9. Decoder

The NETINT decoders h264_ni_quadra_dec, h265_ni_quadra_dec, vp9_ni_quadra_dec, and jpeg_ni_quadra_dec all use Quadra hardware for decoding. The Quadra decoder has 3 post processors which can be enabled to perform cropping, scaling, and 10 to 8 bit conversion. We refer to their outputs as Out0, Out1, and Out2. Out 0 is always enabled and Out1 and Out 2 may be enabled as required. The order of operations in the post processor is cropping followed by scaling.

The Quadra decoder supports decoding images from 144x144 to 8192x8192. For JPEG the range is 48x48 to 8192x8192.

The limit of the input can be determined by the following formula:

Number of reference frames * (Picture width * Picture Height * 1.5 * ((Bit depth + 7)/8)) <= 989.4 MB

The NETINT decoder supports hardware and software AVFrames for output depending on the *out* parameter. If out=sw (default), the decoder will return a software AVFrame. The software AVFrame will require the YUV data to be transferred to the host. For software frames, only the first decoder output is available. If out=hw, the decoder will use hardware frames and not return the YUV data to the host but instead leave the YUV data on the hardware and return instead a handle to the YUV frame. If multiple outputs are enabled, the firmware will return multiple handles. This AVFrame will not contain pointers to a YUV frame data, but will point to the hardware descriptors. A hardware frame can contain handles for up to 3 outputs of 8 or 10 bit compressed YUV. In order to use a hardware AVFrame with an FFmpeg filter or soft encoder, the YUV frame must be transferred to the host as a software frame by calling the FFmpeg hwdownload filter. **Note that most NETINT filters require a hardware frame**.

See also **APPS548 Codensity Quadra Software and Hardware Frames (YUVbypass) Application Note** to learn more about hardware frames.

JPEG supports 8 bit baseline only. There is no support for lossless or progressive decoding.

Like the T408, the Quadra decoder will return any SEI payloads that we support to libxcoder as part of the metadata. The SEIs currently supported for decoding are t35 (close captions and HDR10+ metadata), mastering display colour volume, content light level info, and alternative transfer characteristics for HDR, pic timing and buffering period for HRD.

It should be noted that the character limit for an expression for decoder scaling or cropping is twenty characters.

9.1 Decoder Parameters

out

Specifies the output type of decoder output Out0. Specifies whether a hardware or software frame is returned.

Supported Values:
sw: Software
hw: Hardware
Default: sw: Software

enableOut1

Enables Decoder output 1. Note that Decoder output 0 is always enabled. Out1 if enabled is always returned as a hardware frame.

Supported Values:
0: Disable
1: Enable
Default: 0: Disable

enableOut2

Enables Decoder output 2. Note that Decoder output 0 is always enabled. Out2 if enabled is always returned as a hardware frame.

Supported Values:
0: Disable
1: Enable
Default: 0: Disable

force8Bit0

When enabled, 10 bit video on output 0 is converted to 8 bit output by shifting down by 2 bits.

Not Applicable: JPEG, semiplanar0 = 2 with 10-bit input
Supported Values:
0: Disable
1: Enable
Default: 0: Disable

force8Bit1

When enabled, 10 bit video on output 1 is converted to 8 bit output by shifting down by 2 bits.

Not Applicable: JPEG, semiplanar1 = 2 with 10-bit input
Supported Values:
0: Disable
1: Enable
Default: 0: Disable

force8Bit2

When enabled, 10 bit video on output 2 is converted to 8 bit output by shifting down by 2 bits.

Not Applicable: JPEG, semiplanar2 = 2 with 10-bit input
Supported Values:
0: Disable
1: Enable
Default: 0: Disable

semiplanar0

When set to 0, the yuv420 format is used, main perk is that it is most compatible with other HW blocks or SW.
When set to 1, yuv output 0 format will be in nv12 or p010le semiplanar format depending on source bit-depth. Improves performance when memory usage is high.
When set to 2, yuv output 0 format will be in a special compressed tiled format. This format greatly improves performance when memory usage is high. Ideal for large resolution 10-bit inputs. Must have HW frames enabled, VP9 not supported, final decoder output resolution (after any crop or scale) must align to multiples of 4, and only compatible with co-located encoder, ni_quadra_scale and ni_quadra_overlay filters for downstream usage.

Supported Values:
0: Disable
1: Enable
2: Tiled mode
Default: 0: Disable

semiplanar1

When set to 0, the yuv420 format is used, main perk is that it is most compatible with other HW blocks or SW.
When set to 1, yuv output 1 format will be in nv12 or p010le semiplanar format depending on source bit-depth. Improves performance when memory usage is high.
When set to 2, yuv output 1 format will be in a special compressed tiled format. This format greatly improves performance when memory usage is high. Ideal for large resolution 10-bit inputs. Must have HW frames enabled, VP9 not supported, final decoder output resolution (after any crop or scale) must align to multiples of 4, and only compatible with co-located encoder, ni_quadra_scale and ni_quadra_overlay filters for downstream usage.

Supported Values:

0: Disable
1: Enable
2: Tiled mode

Default: 0: Disable

semiplanar2

When set to 0, the yuv420 format is used, main perk is that it is most compatible with other HW blocks or SW.
When set to 1, yuv output 2 format will be in nv12 or p010le semiplanar format depending on source bit-depth. Improves performance when memory usage is high.
When set to 2, yuv output 2 format will be in a special compressed tiled format. This format greatly improves performance when memory usage is high. Ideal for large resolution 10-bit inputs. Must have HW frames enabled, VP9 not supported, final decoder output resolution (after any crop or scale) must align to multiples of 4, and only compatible with co-located encoder, ni_quadra_scale and ni_quadra_overlay filters for downstream usage.

Supported Values:

0: Disable
1: Enable
2: Tiled mode

Default: 0: Disable

cropMode0

Specifies the crop mode for output 0. When set to auto, the cropping parameters are determined from the bitstream. When set to manual, the cropping parameters are specified by the crop parameters below.

Supported Values:
auto
manual
Default: auto

cropMode1

Specifies the crop mode for output 1. When set to auto, the cropping parameters are determined from the bitstream. When set to manual, the cropping parameters are specified by the crop parameters below.

Supported Values:
auto
manual
Default: auto

cropMode2

Specifies the crop mode for output 2. When set to auto, the cropping parameters are determined from the bitstream. When set to manual, the cropping parameters are specified by the crop parameters below.

Supported Values:
auto
manual
Default: auto

crop0

Cropping parameters for output 0. When manual mode is selected, specifies the x and y coordinates where cropping begins and the width and height for cropping. Existing header cropping info ignored. Note that the cropping dimensions must be even, and the minimum cropped size is 48x48. Out of bounds offsets will be resized to fit with corresponding width and height values.

Supported Values: W,H,X,Y.
Default: decoded picture width, decoded picture height, 0, 0

crop1

Cropping parameters for output 1. specifies the x and y coordinates where cropping begins and the width and height for cropping. Existing header cropping info ignored. Note that the cropping dimensions must be even, and the minimum cropped size is 48x48. Out of bounds offsets will be resized to fit with corresponding width and height values.

Supported Values: W,H,X,Y.
Default: decoded picture width, decoded picture height, 0, 0

crop2

Cropping parameters for output 2. When manual mode is selected, specifies the x and y coordinates where cropping begins and the width and height for cropping. Existing header cropping info ignored. Note that the cropping dimensions must be even, and the minimum cropped size is 48x48. Out of bounds offsets will be resized to fit with corresponding width and height values.

Supported Values: W,H,X,Y.
Default: decoded picture width, decoded picture height, 0, 0

scale0

Specifies the width (w) and height (h) to scale decoder output 0. If not specified, then no scaling is done. The width and height may be set to no larger than the decoded output size. Note that scaling is applied after any cropping is done. Note that the scaling dimensions must be even, and the minimum scaling size is 2x2.

Supported Values: W(Width)x H(Height)
Default: Disabled

scale1

Specifies the width (w) and height (h) to scale decoder output 1. If not specified, then no scaling is done. The width and height may be set to no larger than the decoded output size. Note that scaling is applied after any cropping is done. Note that the scaling dimensions must be even, and the minimum scaling size is 2x2. Values are W,X,H.

Supported Values: W(Width)x H(Height)
Default: Disabled

scale2

Specifies the width (w) and height (h) to scale decoder output 2. If not specified, then the no scaling is done. The width and height may be set to no larger than the decoded output size. Note that scaling is applied after any cropping is done. Note that the scaling dimensions must be even, and the minimum scaling size is 2x2.

Supported Values: W(Width)x H(Height)
Default: Disabled

multicoreJointMode

Enables decoder multi-core mode where all 4 cores work together in parallel (a.k.a. joint mode). When disabled (default), the decoder instances only use a single video core to decode. When enabled, decoder instances uses all 4 video decoding cores in parallel. Recommended only for high resolution decoding with less than 4 instances, e.g. 8K decode. When more than 4 decoding instances are used, enabling this feature will lower performance due to extra synchronization overhead.

Not Applicable: JPEG or VP9.
Supported Values:
0: Disable
1: Enable
Default: 0: Disable

lowDelay

Specifies whether to enable the low latency mode in decoding. When enabled, libxcoder uses a different query method that returns upon frame ready to reduce polling. This method only permits buffering of a single frame to minimize delay and therefore will not work with non-sequentially decoded inputs. If improper input is provided, the query will timeout within 4 seconds and self-disable the low delay mode and decode as normal.

Note – If non-sequential input (B frames) is provided, this feature will disable. Input also requires POC type = 2

Note that in libxcoder decoder send/receive multi-thread mode, when enabled, its value can be a positive integer value in milliseconds for threads synchronization. It represents the time the sending thread waits before deciding it's in a deadlock and has to continue without waiting for receiving thread to signal.

Not Applicable: JPEG or VP9.
Supported Values:
0: Disable
Positive integer: Enable
Default: 0: Disable

forceLowDelay

By Default, decoder low delay mode will be cancelled automatically if there's frame drop, i.e, a stream packet sent to decoder is not able to output as a frame. With this option enabled, decoder won't exit lowDelay mode when frame drop happens. Instead, it will accumulate the number of dropped frames and continue decoding in low delay mode. This option should only be applied when it's certain that the input stream can be decoded without reordering, else enabling it will break the display order of frames.
Note – If non-sequential input (B frames) is provided, this feature will break the display order of frames.

Not Applicable: JPEG or VP9.
Supported Values:
0: Disable
1: Enable
Default: 0: Disable

enableLowDelayCheck

This function is used to detect the presence of B-frames in the input stream. This function will only take effect when low-delay mode is enabled. If B frames are detected in low delay mode, the low delay function will be turned off.
Note – If it can be determined that the decoding order of the input stream is consistent with the display order, do not enable this function.

Not Applicable: JPEG or VP9.
Supported Values:
0: Disable
1: Enable
Default: 0: Disable

keepAliveTimeout

Specifies a session keep alive timeout value. This is a periodical request/response between libxcoder and XCoder firmware that when timed out, the session instance will be terminated by XCoder firmware. If this option is used in conjunction with FFmpeg command option keep_alive_timeout then keepAliveTimeout overrides keep_alive_timeout.

Supported Values: Integer in the range 1 to 100
Default: 3

customSeiPassthru

Specify a custom SEI type to passthrough.

Supported Values: Integer in the range -1 to 254

Default: -1

enableAllSeiPassthru

All custom SEI types will be passed through if this is enabled. Also, when enabled, the firmware SEI will be disabled.

Note – If the **enableAllSeiPassthru** parameter is enabled (set to 1) for decoding, then the **enableAllSeiPassthru** parameter for encoding must also be enabled (set to 1).

Supported Values:

0: Disable
1: Enable

Default: 0

enableUserDataSeiPassthru

Enable user data unregistered SEI passthrough.

Supported Values: Integer of 0 (false) or 1 (true)

Default: 0 (false)

svctDecodingLayer

Specifies the maximum temporal layer ID of frames to be decoded. The decoder will discard frames with temporal layer ID higher than the value specified by this parameter.

Note – this is only for H.264 SVC-T (temporal scalable video coding) decoding. Default value is -1. By default, it will decode all the frames.

Supported Values: Integer in the range -1 to max integer

Default: -1

ddrPriorityMode

Specifies the ddr priority mode. Only need set once at beginning, and it will reset to default automatic after current process finish.

Note – this is a global setting, it will influence all running processes. It is best to only use it when there is only one process. If there are multiple processes, other processes fps performance may influence by this parameter.

Supported Values:

0: set default ddr mode
1:increase ddr priority for decoder and encoder
2:increase ddr priority for scaler
3:increase ddr priority for ai

Default: -1

ecErrThreshold

Specifies the threshold of broken ratio of a decoder output picture. If a decoded picture has larger broken part than the threshold, it would be considered as a broken picture and won't be used as a reference picture.

Note – if ecPolicy is set to limited_error, picture with larger broken part than threshold would be dropped.

Supported Values: Integer in the range 0 to 100

0: a good picture should have no broken part
100: all pictures are considered good

Default: 10

ecPolicy

Specifies the error concealment policy that should be used by the decoder when it encounters a broken bitstream. If a frame was not completely decoded because of bitstream errors it can still be used as a reference in H.264/HEVC/VP9 codecs, this will cause lingering artifacts in the subsequent frames that reference one of the frames with decoding errors. The **ecPolicy** controls how corrupted reference frames are handled during decoding.

Note - Applicable H.264/HEVC/VP9 to only. Not supported for tiled outputs.

Supported values:

skip: If no replacement is available, skip decoding of all frames until the next I-frame.

limited_error: Try to replace corrupted reference frames other frames that were previously decoded and are kept in the decoded pictures buffer. If no replacement is available then decoding of all frames will be skipped until the next I-frame. If broken ratio of the output picture is larger than threshold(default 10%), the picture would be dropped.

tolerant: Try to replace corrupted reference frames other frames that were previously decoded and are kept in the decoded pictures buffer. If no replacement is available then decoding of all frames will be skipped until the next I-frame.

best_effort: Try to replace corrupted reference frames with other frames that were previously decoded and are kept in the decoded picture buffer. If no replacement frame is available in the decoded pictures buffer, keep using the original reference frame despite the decoding errors.

ignore: Ignore any decoding errors in reference frames and keep using them as is.

best_effort_out_dc: It's very similar to best_effort. Compared with best_effort, it will drop frames whose error_ratio is larger than error_ratio_threshold, and it won't inherit reference frames' error_ratio if no replacement frame is available.

Default: best_effort

enableAdvancedEc

If a frame has not been completely decoded due to bitstream errors, or if a partially decoded reference frame was used to decode the frame, the decoder will conceal the errors in the output pictures. With **enableAdvancedEc**=1 (the default), part of a previous frame will be copied over the broken part of a partially decoded frame. If no good frame is available or **enableAdvancedEc**=0, then a solid green color fill will be used to show the error. With **enableAdvancedEc**=2, part of a previous frame will be copied over the broken part of a partially decoded frame and the additional memory will be kept in whole decoding life.

Note - Only applicable to H.264/HEVC/VP9. Not supported for tiled outputs.

Supported values:

0: Always use a solid green color to conceal partially decoded frames.
1 (default): Try to use the last good frame to conceal any picture errors
2: Try to use the last good frame to conceal any picture errors in whole decoding life.

in the partially decoded frames. With **enableAdvancedEc**=1, when a broken bitstream is encountered, the decoder will allocate additional memory to hold the last good frame so it can be used for concealment. That is, a slightly higher memory usage must be expected when a corrupted stream is being fed into the decoder, with this option enabled. With **enableAdvancedEc**=2, when the first good frame is decoded, the decoder will allocate additional memory to hold the last good frame so it can be used for concealment. That is, a slightly higher memory usage must be expected whether the bitstream is a good or corrupted.

Default: 1 (enabled)

enablePpuScaleAdapt

This parameter is used to enable the PPU scale long and short edge adaptation.

If long edge adaptation is enabled, the resolution of the input stream changes, causing the resolution of the output stream to be recalculated based on the scaling ratio between the short edge and the long edge. For example, if the initial input stream is 1920x1080, and the **ppuscale** is set to 960x360, and long edge adaptation is enabled, then the long side is 1920 and the short side is 1080. The output short side remains unchanged at 360. The long side is calculated to be

1920x360/1080 = 640

Which means the output resolution will be 640x360. When the input stream is changed to 1080x1920 the output stream will be changed to 360x640. If the subsequent input stream is changed to 1080x720, the output stream will be changed to 540x360. Therefore, when we set long edge adaptation and set the output resolution to 960x360, the resolution change of the input stream is 1920x1080->1080x1920->1080x720, and the output resolution is 640x360->360x640->540x360.

If short edge adaptation is enabled and other conditions remain unchanged, the output resolution is 960x540->540x960->960x640.

Note – If the **enablePpuScaleAdapt** parameter is enabled, the resolution of the PPU output is aligned downwards by 2 (by default). We do not recommend using this parameter. We recommend using **scale0LongShortAdapt** instead.

Supported Values:

0: set default disable PpuScaleAdapt
1: set PpuScaleAdapt as adapt to long edge
2: set PpuScaleAdapt as adapt to short edge

Default: 0

Scale0LongShortAdapt

This parameter is used to enable PPU scale0 long and short edge adaptation. You can check the detailed usage instructions in Chapter 9.2.

Supported Values:

0: set default disable PpuScale0Adapt
1: set PpuScale0Adapt as adapt to long edge
2: set PpuScale0Adapt as adapt to short edge

Default: 0

Scale1LongShortAdapt

This parameter is used to enable PPU scale1 long and short edge adaptation. You can check the detailed usage instructions in Chapter 9.2.

Supported Values:

0: set default disable PpuScale1Adapt
1: set PpuScale1Adapt as adapt to long edge
2: set PpuScale1Adapt as adapt to short edge

Default: 0

Scale2LongShortAdapt

This parameter is used to enable PPU scale2 long and short edge adaptation. You can check the detailed usage instructions in Chapter 9.2.

Supported Values:

0: set default disable PpuScale2Adapt
1: set PpuScale2Adapt as adapt to long edge
2: set PpuScale2Adapt as adapt to short edge

Default: 0

Scale0ResCeil

This parameter sets the resolution of the ppu scale0 output to be rounded, and the input parameter must be an even number. For example, [2, 4, 8, 16]. The default is 2.

Supported Values: even number

Default: 2

Scale1ResCeil

This parameter sets the resolution of the ppu scale1 output to be rounded, and the input parameter must be an even number. For example, [2, 4, 8, 16]. The default is 2.

Supported Values: even number

Default: 2

Scale2ResCeil

This parameter sets the resolution of the ppu scale2 output to be rounded, and the input parameter must be an even number. For example, [2, 4, 8, 16]. The default is 2.

Supported Values: even number

Default: 2

Scale0Round

This parameter is used to set whether the resolution of the ppu scale0 is rounded up or down, with default rounding up.

Supported Values: [up, down]

Default: up

Scale1Round

This parameter is used to set whether the resolution of the ppu scale1 is rounded up or down, with default rounding up.

Supported Values: [up, down]

Default: up

Scale2Round

This parameter is used to set whether the resolution of the ppu scale2 is rounded up or down, with default rounding up.

Supported Values: [up, down]

Default: up

enablePpuScaleLimit

Enabling this parameter causes the output resolution of the PPU Scale to be compared to the input stream resolution. If the PPU Scale is upward then an error will be reported via exit.
For example, when the input stream is 720x360, and if the output resolution is 2560x1440, then the scaling will exit with an error. If however, enablePpuScaleLimit=0 then it will not exit with an error, the smallest resolution between the input and output will be chosen as the actual output resolution.

Supported Values:

0: Set disabled
1: Set enabled

Default: 0

skipPtsGuess

When enable this parameter, it will pass through the decoder pts and skip guessing correct pts in libxcoder. By default, this parameter is disabled.

Supported Values:
0: Disable
1: Enable
Default: 0: Disable

minPacketsDelay

Specifies whether to enable the minimum decoding delay packets feature. When enabled, libxcoder increases its rate of polling the decoder and only permits buffering of the minimum packets to minimize the delay.

Note – The minimum decoding delay packets is calculated according to the related bitstream header SPS/PPS/VPS. If enable this option in multi-core mode(multicoreJointMode), the decoding performance will decrease to non-multi-core mode level.

Not Applicable: JPEG or VP9.

Supported Values:
0: Disable
1: Enable

Default: 0: Disable

enableFollowIFrame

If “-force_key_frame source” and “intraPeriod=0” are specified in the command line for transcoding, IDR frames are inserted at the same locations as in the source, excluding non-IDR I-frames. For more details, see Section 12.6, ‘IDR Frame Forcing.’

Setting enableFollowIFrame to 1 causes the decoder to mark both non-IDR and IDR I-frame locations in the source, enabling the encoder to insert IDR frames at these marked locations. The example command is shown below:

```
ffmpeg -c:v h264_ni_quadra_dec -xcoder-params enableFollowIFrame=1 -i input.264 -force_key_frames source -c:v h265_ni_quadra_enc -xcoder-params “intraPeriod=0:RcEnable=1:bitrate=7500000” output.265
```

Not Applicable: JPEG.
Supported Values:
0: Disable
1: Enable
Default: 0: Disable

disableAdaptiveBuffers

Specifies whether to disable adaptive buffers when bitstream sequence change. When **disableAdaptiveBuffers**=1 and width/height of pictures is different from previous width/height in sequence change, the picture buffers will re-allocate. It takes a little time to re-configure decoder and save memory space if width/height is from large to small.

Note – If this option is used and the output type of decoder is **hw** in transcoding , the **disableAdaptiveBuffers** parameter of quadra encoder should be used at same time. If there are many sequence changes in customer’s bitstream, suggest to enable this parameter.
Not Applicable: JPEG or VP9.
Supported Values:
0: Disable
1: enable
Default: 0: Disable

surviveStreamErr

Specifies whether to treat unsupported or corrupted value in stream headers as non-fatal errors. When surviveStreamErr=1, if the decoder encounters an unsupported value in stream headers, the following frames would be dropped until the next supported header is received. Then the decoder will continue decoding the supported frames.

This feature is useful when decoding inputs that may suffer from random packet loss (e.g. livestream through UDP). It makes sure that the decoding process does not terminate when corruptions occur in the stream headers.

Not Applicable: JPEG.

Supported Values:

0: Disable
1: enable

Default: 0: Disable

reduceDpbDelay

Specifies whether to enable reduceDpbDelay mode for H264 bitstream. When **reduceDpbDelay** is equal to 1 and the flag bitstream_restriction_flag of vui_parameters is equal to 0 or **reduceDpbDelay** is equal to 1 and there is no vui_parameter, SW will get decoded picture after two packets are decoded in the decoding beginning. If FW found there is out of order picture, FW will increase the number of packets which need to be decoded and drop out of order picture.

Note – This feature is an 'experimental' feature with risks which may drop some out of order pictures and there may be frame skip in the decoding beginning. The feature will obviously decrease the delay of the first decoded picture, because in some cases, the delay of the first decoded picture is very big like 17.

Not Applicable: JPEG or VP9 or HEVC.

Supported Values:

0: Disable
1: enable

Default: 0: Disable

The following is a Quadra decoder example using the postprocessor to do the scaling. Since the cropping mode defaults to auto, any required cropping will be applied. For example, since the picture height for 1080p is not divisible by 16, it must be padded to 1088 before H.264 encoding thus the bitstream will contain cropping info to remove this padding back to a picture size of 1920x1080. The hwdownload filter will use our callback function to fetch the YUV data to write to the output file. The downloaded YUV will already be scaled and so will require less CPU to transfer it. If we do a transcode to a NETINT encoder, the YUV will not need downloading at all.

```
ffmpeg -y -c:v h264_ni_quadra_dec -dec 0 -xcoder-params "out=hw:scale0=1280x720:force8Bit0=1" -i inputs/h264/Marketplace_1920x1080p30_300_10bit.h264 -vf hwdownload,format=yuv420p -c:v rawvideo temp.yuv
```

Below is a Quadra decoder example using the postprocessor to do manual cropping. In this case we want to change the aspect ratio of the output from 16:9 to 4:3. This is accomplished by cropping the 1920x1088 decoded output to 1440x1080. We have started the cropping at x=240 and y=0 which removes 240 pixels from both the right and left sides of the picture and the bottom 8 bits of padding from the bottom of the picture. Since hw is not specified in this case we default to outputting a 10 bit software YUV frame.

```
ffmpeg -c:v h264_ni_quadra_dec -dec 0    -xcoder-params "cropMode0=manual:crop0=1440,1080,240,0"    -i input1080p10bit.264 -c:v rawvideo output1440x1080-10bit.yuv
```

9.2 PPU Scale Adaptive

We support three configuration methods for ppu scale adaptation, which can be configured through parameter " scale0LongShortAdapt", parameter " enablePpuScaleAdapt", and “scale0=0xheight or Widthx0”. We do not recommend using the parameter "enablePpuScaleAdapt" because after enabling this parameter, it defaults to rounding down 2. And after enabling this parameter, it will synchronously apply to ppu0, ppu1, ppu2. The other two configuration methods are rounded up by 2 by default, and ppu0 ppu1 ppu2 can be configured separately through scale0LongShortAdapt scale1LongShortAdapt and scale2LongShortAdapt (scale0=0xheight, scale1=0xheight, scale2=0xheight).

This will be introduced with parameter “scale0LongShortAdapt”. It must be used with scale0=widthxheight. It can set 3 values, 0, 1, 2. The default value is 0, 1 is to enable long edge resolution adaptation, while short edge resolution remains unchanged. 2 is to enable short edge resolution adaptation while keeping long edge resolution unchanged. We will use in_w, in_h represents the input stream resolution, out_w, out_h represents the final output stream resolution, scale0_w. scale0_h represents the expected ppu output resolution set.

With “scale0LongShortAdapt = 0,scale0=scale0_w X scale0_h”, If scale0_w is larger than in_w or scale0_h is larger than in_h, then the output resolution is consistent with the input resolution, (out_w = in_w, out_h = in_h). otherwise the output resolution is the set resolution, (out_w = scale0_w, out_h = scale0_h). And if “scale0=0x0”, the output resolution is consistent with the input resolution, If “scale0=0xheight”, it is same as “scale0LongShortAdapt=1”, If “scale0=widthx0”,it is same as “scale0LongShortAdapt=2”.

For example, we set scale0LongShortAdapt = 1, if the initial input stream is 1920x1080, scale0 is set to 960x360. At this point, the long side is 1920 and the short side is 1080, so the output short side remains unchanged and is 360. The long side is calculated to be 1920x360/1080=640, and the output resolution is 640x360. Then, when the input stream is changed to 1080x1920, the output stream will be changed to 360x640. If the subsequent input stream is changed to 1080x720, the output stream will be changed to 540x360. Therefore, when we set long edge adaptation, we set the output resolution to 960x360, The resolution change of the input stream is 1920x1080->1080x1920->1080x720, and the output resolution is 640x360->360x640->540x360. If short edge adaptation is enabled and other conditions remain unchanged, the output resolution is 960x540->540x960->960x640.

Note- If scale0LongShortAdapt = 1 or 2, when scale0=0x0, 0xh, or wx0, it will return error code from libxocoder. And if “scale0LongShortAdapt = 1 or 2” or “scale0=0xh, scale0=wx0”, we will check the area of input stream and scale stream. For example, the input stream is 1280x720, and scale0=3000x360, due to 3000x360 is larger than 1280x720, so the output stream is 1280x720.

We will also check the output width and height is larger than the input stream. For example, the input stream is 640x360, scale0=0x720, in this case, the resolution of the short side remains unchanged, and when calculating the resolution of the long side, the calculated resolution is 1280x720. Because we do not support expansion on either side, the final resolution is 640x360.

```
ffmpeg -c:v h264_ni_quadra_dec -xcoder-params out=hw:scale0=720x1280:scale0LongShortAdapt=0
-I input.h264 -c:v h265_ni_quadra_enc output.h265
```

10. Filters

Libxcoder supports hardware filters in Quadra that make use of the hardware scaling, cropping, padding, and overlay features of the 2D engine, provide access to the multiple outputs of the decoder and to support transferring software AVFrames to the hardware so they can be used as hardware AVFrames.

The table below lists the NETINT hardware filters, all of which are implemented using features of the 2D and AI engines (except for ni_quadra_split and ni_quadra_hwupload).

Ni_quadra_split is used to provide access to the 2nd and 3rd decoder output. The FFmpeg split function can also be used on hardware AVFrames but will only split the first output.

Ni_quadra_hwupload transfers the YUV data from a software frame to the hardware and generates a hardware frame as an output.

See also APPS548 Codensity Quadra Software and Hardware Frames (YUVbypass) Application Note to learn more about hardware frames.

Quadra libxcoder filters

Filter name	Description	Resources Used
ni_quadra_scale	NETINT Scaling Filter	2D Engine
ni_quadra_overlay	NETINT Overlay filter	2D Engine
ni_quadra_split	NETINT Split filter (gives access to 2nd and 3rd decoder outputs)	
ni_quadra_crop	NETINT Crop filter	2D Engine
ni_quadra_pad	NETINT Pad filter	2D Engine
ni_quadra_hwupload	NETINT Hardware Upload Filter (transfers a software YUV or RGBA frame to the hardware for encode or filtering)	
ni_quadra_roi	NETINT ROI filter	AI Engine, 2D Engine
ni_quadra_bg	NETINT background removal filter	AI Engine, 2D Engine
ni_quadra_xstack	NETINT Stacking filter	2D Engine
ni_quadra_rotate	NETINT Rotate Filter	2D Engine
ni_quadra_drawbox	NETINT Draw box Filter	2D Engine
ni_quadra_drawtext	NETINT Draw text Filter	2D Engine
ni_quadra_delogo	NETINT Delogo Filter	2D Engine
ni_quadra_merge	NETINT Merge Filter	2D Engine
ni_quadra_flip	NETINT Flip Filter	2D Engine
ni_quadra_hvsplus	NETINT HVS+ Filter	AI Engine, 2D Engine

The NetInt 2D Engine filters are hardware-assisted filters to provide the application the ability to scale, crop, pad, overlay, stack, rotate, and draw video inside the hardware without the need to transfer YUV data to and from the host. For example if we decode video in 16:9 aspect ratio, scale, crop, then re-encode to video with a 4:3 aspect ratio, we can do all of this in the hardware without needing to transfer any YUV using hardware frames and the Netint filters. If we were to use FFmpeg's native scale and crop filters we would need to transfer the decoded YUV to the host to scale and crop and then transfer back to the hardware to encode.

The ni_quadra_scale, ni_quadra_overlay, ni_quadra_crop, ni_quadra_pad, ni_quadra_xstack, ni_quadra_rotate, ni_quadra_drawtext, ni_quadra_drawbox, ni_quadra_delogo, ni_quadra_merge and ni_quadra_flip filters only work with hardware frames. To use these NETINT filters with software frames, you must first use ni_quadra_hwupload to upload the software frame to the Quadra device to create a hardware frame. Similarly, FFmpeg has a native hwdownload filter to retrieve the YUV data from the hardware, converting a hardware frame to a software frame.

The ni_quadra_split filter may be used on software frames though the behavior will default to what the FFmpeg native split filter will do. The ni_quadra_roi and ni_quadra_hvsplus filters work with hardware and software frames.

The 2D Engine supports raster YUV input and output as well as RGBA format for overlay. It requires input pictures to have even width and height for RGBA. For YUV the stride must be a multiple of 128 bytes for both the luma and chroma planes. The minimum picture height is 16 pixels and the minimum picture width is 16 pixels.

The maximum resolution for ARGB, ABGR, RGBA, BGRA is 7040x7040. The maximum resolution for other pixel formats is 8192x8192.

For ni_quadra_overlay, ni_quadra_crop, ni_quadra_pad, ni_quadra_drawtext, ni_quadra_drawbox, ni_quadra_delogo, ni_quadra_merge and ni_quadra_flip the output format is the same as input format. It depends on the input formats supported by ni_quadra_hwupload (see section 10.1.6).

10.1 ni_quadra_scale

The ni_quadra_scale filter provides up or down scaling to any picture size. It works just like the FFmpeg software scale filter but uses the hardware to do the scaling. Quadra does not support the FFmpeg scale interlacing mode parameter or the libswscale flags and parameters. Scaling is supported for yuv420p, yuv420p10le, nv12, p010le, rgba, argb, abgr, bgra, bgr0, nv16, yuyv422 and uyvy422. The bgrp format cannot be scaled but can be used as an output in the *format* parameter.

ni_quadra_scale only supports hardware AVFrames as input and output. To scale a software frame, use ni_quadra_hwupload to upload the frame to the Quadra device. If the hardware frame is on the same device as the scaler, it can be accessed directly.

The ni_quadra_scale filter only supports limited color range video (also known as “TV” range). While full range video (or “PC” range) can be passed through the filter, the ni_quadra_scale filter will always convert it to limited color range video. This conversion will cause a slight color shift in the output video.

The scaling of the decoder PPU supports both full/PC or limited/tv range video. When using the ni_quadra_scale filter to convert between rgb and yuv, only bt709 and bt2020 are supported.

Parameters

Below are the parameters for the ni_quadra_scale filter:

width, w
height, h

Set the output video dimension.

If the width or w value is 0, the input width is used for the output. If the height or h value is 0, the input height is used for the output.

If one and only one of the values is -n with n >= 1, the ni_quadra_scale filter will use a value that maintains the aspect ratio of the input image, calculated from the other specified dimension. After that it will, however, make sure that the calculated dimension is divisible by n and adjust the value if necessary.

If both values are -n with n >= 1, the behavior will be identical to both values being set to 0 as previously detailed.

size, s

Set the video size using an FFmpeg abbreviation.

Supported Values:

ntsc (720x480), pal (720x576), qntsc (352x240), qpal (352x288), sntsc (640x480), spal (768x576), film (352x240), ntsc-film (352x240), sqcif (128x96), qcif (176x144), cif (352x288), 4cif (704x576), 16cif (1408x1152), qqvga (160x120), qvga (320x240), vga (640x480), svga (800x600), xga (1024x768), uxga (1600x1200), qxga (2048x1536), sxga (1280x1024), qsxga (2560x2048), hsxga (5120x4096), wvga (852x480), wxga (1366x768), wsxga (1600x1024), wuxga (1920x1200), woxga (2560x1600), wqsxga (3200x2048), wquxga (3840x2400), whsxga (6400x4096), whuxga (7680x4800), cga (320x200), ega (640x350), hd480 (852x480), hd720 (1280x720), hd1080 (1920x1080), 2k (2048x1080), 2kflat (1998x1080), 2kscope (2048x858), 4k (4096x2160), 4kflat (3996x2160), 4kscope (4096x1716), nhd (640x360), hqvga (240x160), wqvga (400x240), fwqvga (432x240), hvga (480x320), qhd (960x540), 2kdci (2048x1080), 4kdci (4096x2160), uhd2160 (3840x2160), uhd4320 (7680x4320)

Default: none

force_original_aspect_ratio

Enable decreasing or increasing output video width or height to maintain the original aspect ratio. If a user knows in advance the maximum allowed resolution for a device, then this option can be used to limit the output video to that resolution, while also retaining the aspect ratio.

For example, Device A allows 1280x720 playback and your video source is 1920x800. Enabling this option with the value **decrease**, and specifying 1280x720 on the command line, will cause the output resolution to be 1280x534.

Note this option is different to specifying -1 for the w or h, because for this option the output resolution must also be specified.

Supported Values:

decrease
increase

Default: disable

force_divisible_by

Ensures that the output resolution is divisible by the given integer when used with force_original_aspect_ratio. This option respects the value set for force_original_aspect_ratio and will increase or decrease the resolution accordingly.
This option is useful if you want to have a video fit within a defined resolution using force_original_aspect_ratio but have encoder restrictions when it comes to width or height.

Supported Values: Integer in the range 1 to 256
Default: 1

format

Changes the output pixel format.

Supported Values:

auto: use same pixel format as input
yuv420p: change output format to 8-bit yuv420 planar
nv12 – change output format to 8-bit yuv420 semi-planar
yuv420p10le: change output format to 10-bit yuv420 planar
p010le: change output format to 10-bit yuv420 semi-planar
rgba – change output format to 32-bit rgba
argb – change output format to 32-bit argb
abgr – change output format to 32-bit abgr
bgra – change output format to 32-bit bgra
bgr0 – change output format to 32-bit bgr0
bgrp – change output format to 24-bit bgrp
nv16 – change output format to 64-bit YUV422 semi-planar
yuyv422 – change output format to 8-bit YUV422
uyvy422 – change output format to 8-bit YUV422

Default: auto

keep_alive_timeout

Specifies a session keep alive timeout value. This is a periodic request/response between libxcoder and XCoder firmware that when timed out, terminates the session instance in the XCoder firmware.

Supported Values: Integer in the range 1 to 100

Default: 3

buffer_limit

Whether to limit the number of output buffers. If enabled, the buffering count of the filter is its buffer pool size(default 4) plus encoding buffering count if its downstream is encoder. It's enabled by default in FFmpeg-n6.1 and FFmpeg-n7.1.

Supported Values: Integer 0 and 1.

Default: 0

filterblit

Specifies the scaling algorithm. The default is a simple blit function that uses an algorithm similar to the nearest neighbor algorithm. When the filterblit parameter is set to 1 then the filterblit algorithm will be used for scaling. The filterblit function uses a FIR filter algorithm that is similar in quality to the bicubic algorithm. When the filterblit parameter is set to 2, a bicubic algorithm is used.

Supported Values: Integer in the range 0 to 2

Default: 0

autoselect

It selects the filterblit mode automatically according to the output resolution when it is enabled. It sets filterblit parameter to 2 when resolution is larger than 540p and sets filterblit parameter to 1 in other cases.

Supported Values: Integer in the range 0 to 1

Default: 0

in_color_matrix

Set the input YCbCr color space type. Only applicable to YUV formats. Used when converting YCbCr to RGB.

Supported Values: bt709 and bt2020

Default: bt709

out_color_matrix

Set the output YCbCr color space type. Only applicable to YUV formats. Used when converting RGB to YCbCr.

Supported Values: bt709 and bt2020
Default: bt709

is_p2p

Specifies if output buffer of the filter is p2p buffer. When is_p2p is set to 1, the output buffer of the filter is set as p2p buffer and can be read out by p2p read. Else the output buffer is normal and cannot perform p2p.

Supported Values: Bool 0 and 1.
Default: 0

auto_skip

Specifies if to skip the hardware scale when input is the same as the output. When auto-skip is set to 1, frames will pass through the scale filter and no hardware operations will be executed. When auto_skip is set to 0, hardware scale will always be executed regardless of whether the input is the same as the output.

NOTE: auto-skip is only supported with FFmpeg version > 3.4.2. If the FFmpeg version is lower than 3.4.2 then the hardware operation will always be skipped when the input is the same as the output.

Supported Values: Bool 0 and 1.

Default: 0

param_b

Set parameter B of bicubic.This param only effect when filterblit is set to 2.
NOTE: Please try to use the same value on the same device. Otherwise, it may have some impact on FW load

Supported Values: Double 0 to 1.

Default: 0

param_c

Set parameter C of bicubic.This param only effect when filterblit is set to 2.
NOTE: Please try to use the same value on the same device. Otherwise, it may have some impact on FW load

Supported Values: Double 0 to 1.

Default: 0.75

The following example uses the NETINT ni_quadra_scale filter with all processing on the same device with no YUV transfers. The decoder is on device 0 and outputs hardware frames. The ni_quadra_scale filter automatically co-locates with the decoder and outputs hardware frames on the same device. Since the encoder device ID is -1, the encoder will be co-located with its scaled hardware frame input. If the encoder were to be placed on a different device, then the scaled frame would be automatically transferred to the host and then to the encoder device which is sub-optimal.

```
ffmpeg -c:v h264_ni_quadra_dec -dec 0 –xcoder-params "out=hw" -i input1080p.264 -vf
ni_quadra_scale=1280:720 -c:v h265_ni_quadra_enc -enc -1 -xcoder-params
"RcEnable=1:bitrate=1000000" output720p.265
```

10.2 ni_quadra_overlay

The ni_quadra_overlay filter mixes two streams together with or without alpha blending, the main image and the overlay image. Overlays are typically used to overlay broadcast streams with logos or text. These typically use alpha blending, where the logo background would be transparent, or partially transparent to blend with the main image. Overlays can also be used for picture in picture which will typically not use alpha blending. An overlay can be a moving picture or a single still frame. Alpha defines how the weighting should be given to the color components during the blending process. The ni_quadra_overlay filter is supported for yuv420p, yuv420p10le, nv12, p010le, rgba, argb, abgr, bgra, bgr0, nv16, yuyv422 and uyvy422.

The ni_quadra_overlay filter only supports limited color range video (also known as “TV” range). While full range (or “PC” range) video is accepted, the ni_quadra_overlay filter will always convert it to limited color range video. This conversion will cause a slight color shift in the output video.

Prerequisites

The ni_quadra_overlay filter is unique in that it requires two input frames to produce an output. Both inputs must be hardware frames on the same device. This imposes the following rules:

1. If sources are from two decoding sessions:
   Decoder must have explicit decode ID ie. (-c:v h264_ni_quadra_dec -dec 3 -xcoder-params 'out=hw' -i input1.h264 -c:v vp9_ni_quadra_dec -dec 3 -xcoder-params 'out=hw' -i input2.ivf …)

2. If sources are from upload and decoding session:
   Decoder and upload must have same ID ie. (-c:v h264_ni_quadra_dec -dec 1 -xcoder-params 'out=hw' -i input1.h264 -c:v h264_ni_quadra_dec -dec -1 -xcoder-params 'out=sw' -i input2.h264 -filter_complex '[1:v]ni_quadra_hwupload=1[in2]; [0:v][in2]ni_quadra_overlay=0:0[out];…)

3. Multiple upload instances with ni_quadra_hwupload will also require the upload parameter to have matching device ID.

Parameters

The ni_quadra_overlay parameters are mostly identical to the FFmpeg overlay but it uses the hardware to do the overlay. The input and output from the ni_quadra_overlay filter is a NETINT hardware AVFrame and the underlying pixel format of the output will be the same as the main image. There is a special case where the output is changed to nv12 if the main is compressed 8-bit tiled 4x4 and overlay is rgb.

Note that the eof_action parameter defaults to repeat, this permits a single image to be overlaid for the entire duration of the main video stream.

Below are the parameters for the ni_quadra_overlay filter:

x y

Set the expression for the x and y coordinates of the overlaid video on the main video. In case the expression is invalid, it is set to a big value (meaning that the overlay will not be displayed within the output visible area). The x and y expressions can contain the following parameters, main_w, main_h, width and height of main image, overlay_w, overlay_h, width and height of overlay, hsub, vsub, horiz and vert chroma subsample values of the output, t (timestamp expressed in seconds).

Supported Values: Integer in the range 0 to 8192
Default: 0 for both expressions

eof_action

The action to take when EOF is encountered on the secondary input. Repeat, repeats the last frame, endall ends both streams, pass continues with just the main stream.

Supported Values:
endall
pass
repeat
Default: repeat

shortest

Force the output to terminate when the shortest input terminates.

Supported Values:
0: Disable
1: Enable
Default: 0: Disable

repeatlast

Force the filter to extend the last frame of secondary streams until the end of the primary stream.

Supported Values:
0: Disable
1: Enable
Default: 1: Disable

alpha

Set format of alpha of the overlaid video, it can be straight or premultiplied. If the overlaid video is YUV, the overlay will completely overwrite the background because there is no alpha channel. If the overlaid video is an RGBA icon, then the overlay will blend with the background as per the alpha value of each pixel.

Supported Values:
straight
premultiplied
Default: straight

inplace

Perform an **in-place** overlay. The *ni_quadra_overlay* filter normally makes a copy of the background frame and applies the overlay to the copied frame. This copy operation can have a performance penalty on the Quadra device. To improve performance, the **inplace** parameter will apply the overlay immediately to the background frame. Although this improves performance, this restricts how you can use FFmpeg. You **must not** use the *split* or *ni_quadra_split* filter before the *ni_quadra_overlay* filter. Doing so may cause the video of other entities using the *split* to include the overlay. Also, if the overlay image is too large, performance can actually **decrease**.

is_p2p

Specifies if output buffer of the filter is p2p buffer. When is_p2p is set to 1, the output buffer of the filter is set as p2p buffer and can be read out by p2p read. Else the output buffer is normal and cannot perform p2p. This option won't work when **inplace** is set to 1.

Supported Values: Bool 0 and 1.
Default: 0

keep_alive_timeout

Specifies a session keep alive timeout value. This is a periodic request/response between libxcoder and XCoder firmware that when timed out, terminates the session instance in the XCoder firmware.

Supported Values: Integer in the range 1 to 100
Default: 3

Buffer_limit

Whether to limit the number of output buffers. If enabled, the buffering count of the filter is its buffer pool size(default 4) plus encoding buffering count if its downstream is encoder. It's enabled by default in FFmpeg-n6.1 and FFmpeg-n7.1.

Supported Values: Integer 0 and 1.
Default: 0

The following is a Quadra overlay example. The decoder on device 0 decodes the background frames from the mp4 file. The icon overlay is a PNG file that is converted to RGBA, scaled, then transferred to the same hardware device using ni_quadra_hwupload. This icon will be overlaid on every frame because the default **eof_action** is **repeat**. The ni_quadra_overlay output is sent to the encoder which is also on device 0.

```
ffmpeg -dec 0 -c:v h264_ni_quadra_dec -xcoder-params "out=hw" -i WorldFootball-2min.mp4 -i cbctrans.png -filter_complex "[1:v]format=rgba,ni_quadra_hwupload=0[a];[0:v][a]ni_quadra_overlay=main_w-overlay_w-25:main_h-overlay_h-25[b]"    -c:a copy -map "[b]" -enc 0 -c:v h265_ni_quadra_enc -xcoder-params "RcEnable=1:bitrate=2000000" -map "0:a" WorldFootball-overlay.mp4
```

10.3 ni_quadra_split

The ni_quadra_split filter provides the same functionality as the FFmpeg software split filter but it also provides a way to access the 2nd and 3rd decoder outputs. A Netint hardware frame can contain up to 3 outputs. The first one is directly accessible through the hardware frame, the other 2 can only be accessed through ni_quadra_split.

Parameters

The following are the parameters for the ni_quadra_split filter. Output0 corresponds to the default decoder output frame while output1 and output2 are the extra outputs on multi-output decoding. The integer value provided to each parameter corresponds to the number of copies of the specified output to produce. Note that the ni_quadra_split filter does not actually do any processing on the hardware, it simply adds references to output buffers and gives access to the multiple outputs of the decoder.

output0

Specifies number of copies of output 0 to output as HWAVFrame.

Supported Values: Integers from 0 to 128
Default: 2

output1

Specifies number of copies of output 1 to output as HWAVFrame.

Supported Values: Integers from 0 to 128
Default: 0

output2

Specifies number of copies of output 2 to output as HWAVFrame.

Supported Values: Integers from 0 to 128
Default: 0

The following example uses multiple decoder outputs with scaling in the decoder post processor. Note that the decoder scales can only scale down. Using the scalers in the decoder output is an alternative to the 2D Engine which is a shared resource and could affect performance if more heavily used. ni_quadra_split is used to select the 1st, 2nd, and 3rd decoder output for mapping to the 3 encoders. The encoders are located on the same device as the decoder (-enc -1) to avoid all YUV transfers.

```
ffmpeg -c:v h264_ni_quadra_dec -dec 0    -xcoder-params
"out=hw:enableOut1=1:scale1=1280x720:enableOut2=1:scale2=854x480" -i
demo_1920x1080p30.h264 -filter_complex '[0:v]ni_quadra_split=1:1:1[out1][out2][out3]' -map
'[out1]' -c:v h265_ni_quadra_enc -enc -1 -xcoder-params
"RcEnable=1:vbvBufferSize=3000:bitrate=10000000" 1080p.265 -map '[out2]' -c:v
h265_ni_quadra_enc -enc -1 -xcoder-params "RcEnable=1:vbvBufferSize=3000:bitrate=4000000"
720p.265 -map '[out3]' -c:v h265_ni_quadra_enc -enc -1 -xcoder-params
"RcEnable=1:vbvBufferSize=3000:bitrate=1000000" 480p.265
```

The following is an example using ni_quadra_split and ni_quadra_scale on Quadra. Ni_quadra_split is used to select 3 copies of output 0 and create new hardware AVFrames that can be used to access this output and feed them to each 2D engine. The ni_quadra_scale filters are collocated on the same hardware as the decoder as are the encoders (-enc -1) to avoid all YUV transfers. Using the below method we can get different scale quality.

```
ffmpeg -c:v h264_ni_quadra_dec -dec 0 -xcoder-params "out=hw" -i demo_1920x1080p30.h264 -
filter_complex
'[0:v]ni_quadra_split=3:0:0[out1][in2][in3];[in2]ni_quadra_scale=1280:720[out2];[in3]ni_quadra_scal
e=854:480[out3]' -map '[out1]' -c:v h265_ni_quadra_enc -enc -1 -xcoder-params
"RcEnable=1:vbvBufferSize=3000:bitrate=10000000"    1080p.265 -map '[out2]' -c:v
h265_ni_quadra_enc -enc -1 -xcoder-params "RcEnable=1:vbvBufferSize=3000:bitrate=4000000"
720p.265 -map '[out3]' -c:v h265_ni_quadra_enc -enc -1 -xcoder-params
"RcEnable=1:vbvBufferSize=3000:bitrate=1000000" 480p.265
```

Another example using PPU 0/1/2, ni_quadra_split, for generating more than 3 ladders

```
ffmpeg    -c:v h264_ni_quadra_dec -dec 0 -xcoder-params
out=hw:lowDelay=1:enableOut1=1:enableOut2=1:scale1=1280x720:scale2=960x540    -i
input.list -filter_complex
"[0:v]ni_quadra_split=2:1:2[out1][out2][out3][out4][in5];[in5]ni_quadra_scale=640:360[out5]
" -map [out1] -xcoder-params gopPresetIdx=9:RcEnable=1:bitrate=5000000:intraPeriod=60 -
c:v h264_ni_quadra_enc -enc 0 -f null - -map [out2] -xcoder-params
gopPresetIdx=9:RcEnable=1:bitrate=3500000:intraPeriod=60 -c:v h264_ni_quadra_enc -enc 0
-f null - -map [out3] -xcoder-params
gopPresetIdx=9:RcEnable=1:bitrate=2000000:intraPeriod=60 -c:v h264_ni_quadra_enc -enc 0
-f null - -map [out4] -xcoder-params
gopPresetIdx=9:RcEnable=1:bitrate=1000000:intraPeriod=60 -c:v h264_ni_quadra_enc -enc 0
-f null - -map [out5] -xcoder-params
gopPresetIdx=9:RcEnable=1:bitrate=600000:intraPeriod=60 -c:v h264_ni_quadra_enc -enc 0 -f
null -
```

10.4 ni_quadra_crop

The ni_quadra_crop filter provides cropping similar to the FFmpeg soft crop filter but uses the hardware to do the cropping. There are several applications for cropping. One is to handle the cropping specified in the bitstream (i.e. to remove padding added for hardware or codec alignment). The other is to apply cropping specified by the user for applications such as changing the aspect ratio of a picture, i.e. cropping a 16:9 widescreen image to 4:3.

Cropping information in a bitstream is typically added by the encoder, if it needs pads the input picture size to meet the height and width alignment requirements of the codec, for instance H.264 encodes in 16x16 pixel macroblocks.

Another application for cropping is when you need to change the aspect ratio of a video for example when changing a 16:9 widescreen image to the older 4:3 format, the right and left edges of the picture need to be cropped.

Another method of changing aspect ratio is to use scaling and add letterboxing to the top and bottom of the image. Letterboxing requires the pad filter which is described next. Sometimes a combination of both is used. The ni_quadra_crop filter is supported for yuv420p, yuv420p10le, nv12, p010le, rgba, argb, abgr, bgra, bgr0, nv16, yuyv422 and uyvy422.

The ni_quadra_crop filter only supports limited color range video (also known as “tv” range). While full range (or “PC” range) video is accepted by the filter, the ni_quadra_crop will always convert it to limited color range video. This will cause a slight color shift in the output video.

Parameters

The following are the ni_quadra_crop parameters:

w, out_w

The width of the output video. This expression is evaluated only once during the filter configuration, or when the ‘w’ or ‘out_w’ command is sent.

h, out_h

The height of the output video. This expression is evaluated only once during the filter configuration, or when the ‘h’ or ‘out_h’ command is sent.

Supported Values: Integers from 2 to 8192
Default: ih (input height)

x

The horizontal position, in the input video, of the left edge of the output video. This expression is evaluated per-frame.

Supported Values: Integers from 2 to 8192
Default: (in_w-out_w)/2

y

The vertical position, in the input video, of the top edge of the output video. This expression is evaluated per-frame.

Supported Values: Integers from 2 to 8192
Default: (in_h-out_h)/2

keep_aspect

Force the output display aspect ratio to be the same of the input, by changing the output sample aspect ratio. It defaults to 0.

Supported Values:
0: change aspect ratio
1: keep aspect ratio
Default: 0: change aspect ratio

is_p2p

Specifies if output buffer of the filter is p2p buffer. When is_p2p is set to 1, the output buffer of the filter is set as p2p buffer and can be read out by p2p read. Else the output buffer is normal and cannot perform p2p.

Supported Values: Bool 0 and 1.
Default: 0

keep_alive_timeout

Specifies a session keep alive timeout value. This is a periodic request/response between libxcoder and XCoder firmware that when timed out terminates the session instance.

Supported Values: Integer in the range 1 to 100
Default: 3

buffer_limit

Whether to limit the number of output buffers. If enabled, the buffering count of the filter is its buffer pool size(default 4) plus encoding buffering count if its downstream is encoder. It's enabled by default in FFmpeg-n6.1 and FFmpeg-n7.1.

Supported Values: Integer 0 and 1.
Default: 0

The out_w, out_h, x, y parameters are expressions containing the following constants:

x, y

The computed values for x and y. They are evaluated for each new frame.

in_w, in_h, **iw, ih**

The input width and height

out_w, out_hw, ow, oh

The output (cropped) width and height.

a

Same as iw/ih

sar

input sample aspect ratio

dar

input display aspect ratio, it is the same as (iw / ih) * sar

hsub, vsub

horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" hsub is 2 and vsub is 1.

auto_skip

Specifies if to skip the hardware crop when input is the same as the output. When auto-skip is set to 1, frames will pass through the scale filter and no hardware operations will be executed. When auto_skip is set to 0, hardware crop will always be executed regardless of whether the input is the same as the output.

NOTE: auto-skip is only supported with ffmpeg version > 3.4.2. If ffmpeg version < 3.4.2 the hardware operation will always be skipped when the input is the same as output.

Supported Values: Bool 0 and 1.
Default: 0

The following shows an example using ni_quadra_crop. The input is an 852x480 yuv420p frame that is uploaded as a hardware frame using ni_quadra_hwupload and then cropped to 640x480 starting at (0,0) and then encoded to H.265.

```
ffmpeg -hide_banner -f rawvideo -pix_fmt yuv420p -s:v 852x480 -r 60 -i input852x480.yuv -vf
"format=yuv420p,ni_quadra_hwupload=0,ni_quadra_crop=640:480:0:0" -enc -1 -c:v
h265_ni_quadra_enc -xcoder-params "RcEnable=1:bitrate=10000000" output640x480.265
```

10.5 ni_quadra_pad

The ni_quadra_pad filter works like the FFmpeg software pad filter but uses the hardware to do the padding.

There are a couple of applications for padding. One is to pad an image so that the width and height are aligned as required by an encoder. This is done automatically by the encoder. The other is to change the aspect ratio of an image by letterboxing, i.e. adding black bars to the top and bottom or left and right of an image. The latter application is what the padding filter will normally be used for.

As with scaling, the 2D Engine can pad an image that already contains padding. The ni_quadra_pad filter is supported for yuv420p, yuv420p10le, nv12, rgba, argb, abgr, bgra, bgr0, nv16. The pixel formats yuyv422 and uyvy422 are not supported inputs like the FFmpeg pad filter.

The ni_quadra_pad filter only supports limited color range video (also known as “TV range”). While full range (or “PC” range) video is accepted by the filter, it will convert the video to limited color range video. This conversion will cause a slight color shift in the output video.

Parameters

The following are the ni_quadra_pad parameters:

width, w
height, h

Specify an expression for the size of the output image with the paddings added. If the value for width or height is 0, the corresponding input size is used for the output.

x / y

Specifies the offsets to place the input image within the padded area, with respect to the top/left border of the output image. The x expression can reference the value set by the y expression, and vice versa. If x or y evaluate to a negative number, they'll be changed so the input image is centered on the padded area.

Supported Values: -8192 to 8192
Default: 0

color

Specify the color of the padded area. It can be a color name or a hex value. For example, black is 0x000000. For the complete syntax of this option, check the FFmpeg help: https://ffmpeg.org/ffmpeg-utils.html#color-syntax

Default: 0

aspect

Pad to aspect ratio instead of a resolution.

is_p2p

Specifies if output buffer of the filter is p2p buffer. When is_p2p is set to 1, the output buffer of the filter is set as p2p buffer and can be read out by p2p read. Else the output buffer is normal and cannot perform p2p.

Supported Values: Bool 0 and 1.
Default: 0

keep_alive_timeout

Specifies a session keep alive timeout value. This is a periodic request/response between libxcoder and XCoder firmware that when timed out, terminates the session instance in the XCoder firmware.

Supported Values: Integer in the range 1 to 100
Default: 3

buffer_limit

Whether to limit the number of output buffers. If enabled, the buffering count of the filter is its buffer pool size(default 4) plus encoding buffering count if its downstream is encoder. It's enabled by default in FFmpeg-n6.1 and FFmpeg-n7.1.

Supported Values: Integer 0 and 1.
Default: 0

The value for the width, height, x, and y options are expressions containing the following constants:

in_w, in_h, iw, ih

The input video width and height.

out_w, out_hw, ow, oh

The output video width and height.

x, y

The x and y offsets as specified by the x and y expressions, or NAN if not yet specified.

a

same as iw / ih

sar

input sample aspect ratio

dar

input display aspect ratio, it is the same as (iw / ih) * sar

hsub, vsub

The horizontal and vertical chroma subsample values. For example for the pixel format "yuv422p" hsub is 2 and vsub is 1.

auto_skip

Specifies if to skip the hardware pad when input is the same as the output. When auto-skip is set to 1, frames will pass through the pad filter and no hardware operations will be executed. When auto_skip is set to 0, hardware scale will always be executed regardless of whether the input is the same as the output.

NOTE: auto-skip is only supported with ffmpeg version > 3.4.2. If ffmpeg version < 3.4.2 the hardware operation will always be skipped when the input is the same as output.

Supported Values: Bool 0 and 1.
Default: 0

The following shows an example using ni_quadra_pad to change the aspect ratio from ntsc 480p (640x480) to 16:9 480p (852x480). The raw YUV420 planar file input640x480 is uploaded to device 0 as a hardware frame using the ni_quadra_hwupload filter. The ni_quadra_pad filter then pads to 852x480 and puts the original frame at position x=106, y=0. The padding is set to black. The encoder is also collocated on device 0 using -enc -1.

```
ffmpeg -y -hide_banner -f rawvideo -pix_fmt yuv420p -s:v 640x480 -r 60 -i input640x480.yuv -vf
"format=yuv420p,ni_quadra_hwupload=0,ni_quadra_pad=852:480:106:0:0x000000" -enc -1 -c:v
h265_ni_quadra_enc -xcoder-params "RcEnable=1:bitrate=10000000" output852x480.265
```

10.6 ni_quadra_hwupload

The FFmpeg hwupload filter transfers YUV image data from a software frame to the hardware and outputs a hardware frame. The ni_quadra_hwupload filter does the same thing but adds a device ID parameter to specify which card the image data should be transferred to. Using ni_quadra_hwupload is required if we need to process a software frame with an Netint filter that only supports hardware frames. It is also useful if we need to feed the same software frame to multiple inputs on the same card (say multiple encoders and the 2D Engine) as it saves additional YUV transfers.

12 pixel formats are supported for the ni_quadra_hwupload filter: yuv420p, yuv420p10le, nv12, p010le, rgba, argb, abgr, bgra, bgr0, nv16, yuyv422 and uyvy422.

Parameters

The following are the ni_quadra_hwupload parameters:

device

Device ID of the hardware to upload to. A value of –1 or no entry means to upload to the device with the lowest *upload pixel rate.* If there are not enough resources to open the upload instance, the upload will return with failure.

The *upload pixel rate* is calculated from the input video given to the filter. For example, the ffmpeg command

ffmpeg -r 30 –s:v 1920x1080 –i input.yuv –vf ni_quadra_hwupload=1 ...

uploads to device 1. An *upload pixel rate* of 1920 x 1080 x 30 = 62,208,000 pixels/second will be added to device 1. If another ffmpeg command is run, say

ffmpeg –r 25 –s:v 1280x720 –i input2.yuv -vf ni_quadra_hwupload=-1 ...

the device with the lowest *upload pixel rate* is chosen. A device other than 1 will be selected and will have an *upload pixel rate* of 1280 x 720 x 25 = 23,040,000 pixels/second added to it.

Supported Values: -1, 0-255
Default: Resource monitor assigns the device

devname

Device name of the hardware to upload to. The device name should be like /dev/nvmeXnY as ni_rsrc_mon shows.

keep_alive_timeout

Specifies a session keep alive timeout value. This is a periodic request/response between libxcoder and XCoder firmware that when timed out, terminates the session instance by XCoder firmware.

Supported Values: Integer in the range 1 to 100
Default: 3

The following example shows ni_quadra_hwupload being used with ni_quadra_scale to avoid the multiple YUV transfers. The input MPEG2 bitstream is decoded using the FFmpeg soft decoder which outputs software frames. ni_quadra_hwupload is then used to upload the YUV software frame to device 0 returning a hardware frame. Then the FFmpeg split is used to split the hardware frame into 3. The first split output [out1] is fed directly to a Netint h.265 encoder collocated with the hardware frame (-enc -1). The second split output [in2] is scaled to 720p on [out2] using ni_quadra_scale and then encoded on a second collocated encoder. The third split output [in3] is scaled to 540p using ni_quadra_scale and fed to a third collocated encoder. All decoding, scaling, and encoding is done on a single device with

```
ffmpeg -y -i input1080p-mpeg2.ts -filter_complex
'[0:v]ni_quadra_hwupload=device=0,split=3[out1][in2][in3];

[in2]ni_quadra_scale=1280:720[out2];[in3]ni_quadra_scale=854:480[out3]' -map '[out1]'

-c:v h265_ni_quadra_enc -enc -1 -xcoder-params
"RcEnable=1:vbvBufferSize=3000:bitrate=10000000"

1080p.265 -map '[out2]' -c:v h265_ni_quadra_enc -enc -1 -xcoder-params

"RcEnable=1:vbvBufferSize=3000:bitrate=4000000" 720p.265

-map '[out3]' -c:v h265_ni_quadra_enc -enc -1 -xcoder-params

"RcEnable=1:vbvBufferSize=3000:bitrate=1000000 " 480p.265
```

```
ffmpeg –y –i input.mp4 -vf
ni_quadra_hwupload=devname=/nvme0n1,ni_quadra_scale=1280:720    -c:v
h264_ni_quadra_enc    output.mp4
```

10.7 ni_quadra_roi

The ni_quadra_roi filter analyzes input video to detect faces and generate Region of Interest (ROI) data to improve the encoding quality of the faces. It does this by inferencing the input frames using the AI Engine, identifies the objects' coordinates and classes in the images and creates ROI side data to be used by the encoder.

The ni_quadra_roi filter loads a yolov4 object detection model down to the AI Engine device specified. It also defines its input and output dimensions. For yolov4 models, the input requires a tensor with shape 416x416 and pixel format BGRPlanar. This means that the filter needs to scale these input images and do the format conversion. The filter supports both hardware frames and software frames. For software frames, only YUV420P is supported for now and the soft FFmpeg scaler is used to do the scaling. Since FFmpeg doesn't support BGRPlanar, the filter first scales the input image to a 416x416 RGB24 AVFrame, then rearranges the frame to a compact RGRPlanar tensor, and transfers the tensor down to AI module to do inference. For hardware frames, the 2D Engine converts the RGB24 to BGRPlanar and then does the scaling. The scaled hardware frames are then passed to the AI module to do inferences.

Before the inference begins, an output buffer will be preset to the model. when the inference completes, the output tensor will be saved in the output buffer. The filter then fetches the output tensor from firmware. It runs post processing with the output tensor to compute the objects' boxes and class labels in the images. if there are any objects, their boxes will be converted to the four coordinates relative to the up left zero point of the images. Each objects' coordinates will be put into the ROI side data including a preset QP offset. All the ROI side data in an image will be appended to and passed down to the encoder along with the images.

Parameters

The following are the ni_quadra_roi parameters:

nb

Specifies the AI module (network_binary_yolov4_head.nb) network binary file path including the file name. This field is required for the filter to work.
Supported Values: String
Default: Null

qpoffset

Specifies the ROI QP offset so that encoder can set the specific QP in these regions based on the QP offset. A negative QP offset will increase the quality in the ROI regions detected by the filter. A positive QP offset will decrease the quality of the ROI regions. A QP offset of 0 means no change in quality. The QP offset range of -1 to +1 translates to an actual QP offset of -25 to +25 in the encoder.
Not Applicable: Not applicable for FFmpeg 4.2.0 or before
Supported Values: -1.0 to 1.0
Default: 0.0

devid

Specifies the Device ID of Quadra hardware to use when software frames are used as an input. When a hardware frame is used the filter will be collocated to the same device as the frame.
Supported Values: 0 to max device ID
Default: 0

obj_thresh

Specifies the yolov4 post processing object threshold. Each region would have its score. When it reaches the obj_thresh, it will be taken as an object. The higher the threshold means the harder to be taken as an object.
Supported Values: 0 to 1.0
Default: 0.25

nms_thresh

Specifies the yolov4 post processing NMS IOU threshold. The gold is to select the bounding boxes with the highest detection probability and eliminate all the bounding boxes whose intersection over union (IOU) value is higher than given IOU threshold.
Supported Values: floating point value
Default: 0.45

keep_alive_timeout

Specifies a session keep alive timeout value. This is a periodic request/response between libxcoder and XCoder firmware that when timed out, terminates the session instance in the XCoder firmware.
Supported Values: Integer in the range 1 to 100
Default: 3

buffer_limit

Whether to limit the number of output buffers. If enabled, the buffering count of the filter is its buffer pool size(default 4) plus encoding buffering count if its downstream is encoder. It's enabled by default in FFmpeg-n6.1 and FFmpeg-n7.1.
Supported Values: Integer 0 and 1.
Default: 0

The following example shows the ni_quadra_roi filter using hardware frames:

```
ffmpeg -y -vsync 0 -dec 0 -c:v h264_ni_quadra_dec -xcoder-params 'out=hw' -i cr7_1920x1080.h264  –-vf 'ni_quadra_roi=nb=./network_binary_yolov4_head.nb:qpoffset=-0.6' -enc 0 -c:v h264_ni_quadra_enc -xcoder-params 'roiEnable=1:RcEnable=1:bitrate=500000' -an cr7_1080p_roi_b500000.h264
```

The following example below shows the ni_quadra_roi filter using software frames.

```
ffmpeg -y -vsync 0 -dec 0 -c:v h264_ni_quadra_dec -i cr7_1920x1080.h264 -vf 'ni_quadra_roi=nb=./network_binary_yolov4_head.nb:qpoffset=-0.6' -enc 0 -c:v h264_ni_quadra_enc -xcoder-params 'roiEnable=1:RcEnable=1:bitrate=500000' -an cr7_1080p_roi_b500000.h264
```

Please note that this filter is available with FFmpeg-n4.2.1 and higher version since the ROI API was introduced in FFmpeg-n4.2.

10.8 ni_quadra_bg

The ni_quadra_bg background removal filter analyses input frames, inferences these images using the AI module (segm32), segments the foreground and background of the input images, and then removes the background.

The filter takes a segm_32 object segmentation model as input, down to the AI module specified by the user. The imported model will be unfolded and initialized in memory. As for the segm32 model, the dimensions of the input and output are as defined. The segm32 model requires a tensor with shape 256 x 144 (width x height) and a pixel format of BGR Planar. In this case, the filter needs to scale the input frames and do the format translation to adapt to the different resolutions. Currently the filter only supports hardware frames. The 2D Engine is used to do the scaling and format conversion and then the result is passed to the AI module to perform the inferences.

When the inference completes the output tensor is then retrieved by the filter which then performs some post-processing to segment the foreground and background. The result of the post-processing is the alpha data needed to mix the new background which is converted to RGBA format to be mixed with the original image using the 2D Engine overlay feature.

The ni_quadra_bg filter only supports limited color range video (also known as “TV” range). While full range (or “PC” range) video is accepted by the filter, the ni_quadra_bg filter will always convert to limited color range video. This conversion will cause a slight color shift in the output video.

Parameters

The following are the ni_quadra_bg parameters:

nb

Specifies the AI module (segm32.nb) network binary file path including the file name. This field is required for the filter to work.
Supported Values: String
Default: Null

bg_img

Specifies the background image full path, including the file name. This field is required for the filter to work, and has no default value.
Supported Values: String
Default: Null

use_default_bg

Specifies whether to use the default background image. When the value is set to 0, the background of input video will be replaced by the customized bg_img. If set to 1, the default background will be used. By default the customized bg_img is used.
Supported Values: 0 or 1
Default: 0

skip

Specifies the number of frames to skip between inference. When set to 0, all frames are inferenced. When set to 1, every 2nd frame is inferenced. When set to 2, every 3rd frame is inferenced.

Supported Values: 0 or higher
Default: 0

is_p2p

Specifies if output buffer of the filter is p2p buffer. When is_p2p is set to 1, the output buffer of the filter is set as p2p buffer and can be read out by p2p read. Else the output buffer is normal and cannot perform p2p.

Supported Values: Bool 0 and 1.
Default: 0

The following example shows using the ni_quadra_bg filter using hardware frames with custom background image:

```
ffmpeg -y -loglevel debug -vsync 0 -dec 0 -c:v h264_ni_quadra_dec -xcoder-params 'out=hw' -i ./bg_1920x1080.h264    -vf 'ni_quadra_bg=nb=./segm32.nb:bg_img=./bg.png:use_default_bg=0' -enc 0 -c:v h264_ni_quadra_enc ./bg_1920x1080.h264
```

The following is an example input before and after background replacement:

Before:

After:

10.9 ni_quadra_xstack

The ni_quadra_xstack filter mixes up to 50 streams together with or without alpha blending. The stacking of inputs is typically used to create a grid for broadcast streams used in applications like Zoom or MS Teams.

Prerequisites

The ni_quadra_xstack filter is unique in that it requires multiple input frames to produce an output. All inputs must be hardware frames on the same device. This imposes the following rules:

1. If sources are from several decoding sessions:
   Decoder must have explicit decode ID ie. (-c:v h264_ni_quadra_dec -dec 3 -xcoder-params 'out=hw' -i input1.h264 -c:v vp9_ni_quadra_dec -dec 3 -xcoder-params 'out=hw' -i input2.ivf …)

2. If sources are from upload and decoding sessions:
   Decoder and upload must have same ID ie. (-c:v h264_ni_quadra_dec -dec 1 -xcoder-params 'out=hw' -i input1.h264 -c:v h264_ni_quadra_dec -dec -1 -xcoder-params 'out=sw' -i input2.h264 -filter_complex '[1:v]ni_quadra_hwupload=1[in2]; [0:v][in2]ni_quadra_xstack=0:0[out];…)

3. Multiple upload instances with ni_quadra_hwupload will also require the upload parameter to have matching device ID.

4. Any input YUV (Y’CbCr) video should be limited range. Full range video will be accepted; however, the output video will be limited range and will have a slight color shift.

Parameters

The ni_quadra_xstack parameters are mostly identical to the FFmpeg xstack but it uses the hardware to do the stacking. The inputs and output from the ni_quadra_xstack filter is a NETINT hardware AVFrame and the output frame rate will be the same as the frame rate as the first input except when explicitly specified by the **sync** parameter below. All inputs to the ni_quadra_xstack filters must have the same pixel format. For RGB pixel formats with alpha, the alpha must be straight alpha and not premultiplied (associated) alpha.

Below are the parameters for the ni_quadra_xstack filter:

inputs

Set the number of inputs that the filter will receive.

Supported Values: Integer in the range 2 to 50
Default: 2

layout

Specifies the layout for the x and y coordinates of the input videos on the output video. This option requires the desired layout configuration to be explicitly set by the user. This sets the position of each video input in output. Each input is separated by a '|'. The first number represents the column, and the second number represents the row. Numbers start at 0 and are separated by a '_'. Optionally one can use wX and hX, where X is video input from which to take width or height. Multiple values can be used when separated by a '+'. In such case values are summed together. Odd values are rounded up to the nearest even integer.

Note that if inputs are of different sizes gaps may appear, as not all of the output video frame will be filled. Similarly, videos can overlap with each other if their position doesn't leave enough space for the full frame of adjoining videos.

Supported Values: Integer in the range 0 to 8192

Default: For 2 inputs, a default layout of 0_0|w0_0 is set. In all other cases, a layout must be set by the user.

size

Set the expression for the width and height of the input videos on the output video. If the input video is a different size than the output size specified by this parameter, then it is scaled to the output size. Each input is separated by a '|'. The first number represents the width, and the second number represents the height. Numbers start at 1 and are separated by a '_'. Odd values are rounded up to the nearest even number.

Supported Values: Integer in the range 1 to 8192
Default: Use input video width and height

shortest

Force the output to terminate when the shortest input terminates.

Supported Values:
0: Disable
1: Enable
Default: 0: Disable

fill

Fill the background of the output video with this specified color value

Supported Values:
RGBA in hex or an FFMpeg color name
Default: black

sync

Use the specified input source as the primary source to synchronize all the other inputs with and generate an output with the same FPS as the specified input.

Supported Values: Integer in the range 0 to 49. However, if the value specified is greater than or equal to the inputs parameter above, then input 0 will be used.

Default: 0

is_p2p

Specifies if output buffer of the filter is p2p buffer. When is_p2p is set to 1, the output buffer of the filter is set as p2p buffer and can be read out by p2p read. Else the output buffer is normal and cannot perform p2p.

Supported Values: Bool 0 and 1.
Default: 0

keep_alive_timeout

Specifies a session keep alive timeout value. This is a periodic request/response between libxcoder and XCoder firmware that when timed out, terminates the session instance in the XCoder firmware.

Supported Values: Integer in the range 1 to 100
Default: 3

buffer_limit

Whether to limit the number of output buffers. If enabled, the buffering count of the filter is its buffer pool size(default 4) plus encoding buffering count if its downstream is encoder. It's enabled by default in FFmpeg-n6.1 and FFmpeg-n7.1.

Supported Values: Integer 0 and 1.
Default: 0

Note: When using ni_quadra_xstack in ffmpeg command line, if some of inputs frame rate are different, need to set **–vsync 2** in ffmpeg command line to compatible with different frame rate outputs. For muxer with this extension they will select video sync as VSYNC_VFR by default

.avi .mkv .webm .flv .ts .m2t .m2ts .mts webp .apng .gif

The following is a Quadra xstack example. The decoder on device 0 decodes the h264 file *Crowdrun_3840x2160p30_300.h264* and outputs hardware frames. The decoder on device 0 decodes the file *Dinnerscene_1906x984p60_300.h264* and outputs hardware frames.

The two decoder outputs are inputs to the ni_quadra_xstack filter. The ni_quadra_xstack filter will re-size the first input to 320x240 and the second input to 320x240, as defined by the *size* parameter. After re-sizing both inputs, the ni_quadra_xstack filter will place the first input at position 0,0 and the second input at position 0,240 as defined by the *layout* parameter.

Finally, the output of the ni_quadra_stack filter is given to the NetInt h264 encoder which generates the file xstack_02.mkv

```
ffmpeg -y -loglevel debug -c:v h264_ni_quadra_dec -dec 0 -xcoder-params "out=hw" -i Crowdrun_3840x2160p30_300.h264 -c:v h264_ni_quadra_dec -dec 0 -xcoder-params "out=hw" -i Dinnerscene_1906x984p60_300.h264 -filter_complex "[0:v][1:v]ni_quadra_xstack=inputs=2:layout=0_0|0_h0:size=320_240|320_240[out]" -map "[out]" -c:v h264_ni_quadra_enc -enc 0 xstack_02.mkv
```

Please note that this filter is only available with FFmpeg-n4.1.3 and higher version since the native FFmpeg xstack filter was introduced in FFmpeg-n4.1.

10.10 ni_quadra_rotate

The ni_quadra_rotate filter rotates a picture/video. It works just like the FFmpeg software rotate filter but uses the hardware to do the rotation. Rotation is supported for yuv420p.

ni_quadra_rotate only supports hardware AVFrames as input and output. To rotate a software frame, use ni_quadra_hwupload to upload the frame to the Quadra device. If the hardware frame is on the same device as the scaler, it can be accessed directly.

The ni_quadra_rotate filter only supports limited color range video (also known as “TV” range). While full range (or “PC” range) video is accepted by the filter, the ni_quadra_bg filter will always convert to limited color range video. This conversion will cause a slight color shift in the output video.

Parameters

The following are the ni_quadra_rotate parameters:

angle, a

Angle to clockwise rotate the input in radians.

Supported Values: 0, PI/2, PI, 3*PI/2

Default: 0

out_w, ow

Set the output width expression.

Default: “iw”

out_h, oh

Set the output height expression.

Default: “ih”

fillcolor, c

Set the color used to fill the output area not covered by the rotated image.

Default: “black”

is_p2p

Specifies if output buffer of the filter is p2p buffer. When is_p2p is set to 1, the output buffer of the filter is set as p2p buffer and can be read out by p2p read. Else the output buffer is normal and cannot perform p2p.

Supported Values: Bool 0 and 1.
Default: 0

auto_skip

Specifies if to skip the hardware rotate when input is the same as the output. When auto-skip is set to 1, frames will pass through the rotate filter and no hardware operations will be executed. When auto_skip is set to 0, hardware scale will always be executed regardless of whether the input is the same as the output.

NOTE: auto-skip is only supported with ffmpeg version > 3.4.2. If ffmpeg version < 3.4.2 the hardware operation will always be skipped when the input is the same as output.

Supported Values: Bool 0 and 1.
Default: 0

The following example decodes a H.264 input using the h264_ni_quadra_dec and rotates the video by 180 degrees before passing it to h264_ni_quadra_enc for encoding.

```
ffmpeg -vsync 0 -c:v h264_ni_quadra_dec -xcoder-params 'out=hw' -i input.h264 -vf
ni_quadra_drawbox

'ni_quadra_rotate=PI' -c:v h264_ni_quadra_enc -c:a copy output.h264
```

10.11 ni_quadra_drawbox

The ni_quadra_drawbox filter can draw multiple same coloured boxes on a picture/video. It works just like the FFmpeg software drawbox filter but uses the hardware to do the drawing. Quadra does not support the FFmpeg drawbox AVOptions of thickness or replace. It can only draw with one pixel thickness. Drawbox is supported for rgba, argb, abgr and bgra.

ni_quadra_drawbox only supports hardware AVFrames as input and output. To draw a software frame, use ni_quadra_hwupload to upload the frame to the Quadra device. If the hardware frame is on the same device as the scaler, it can be accessed directly.

Because of HW limitations, there is chromatic aberration when box y position is even value. This should be avoided when drawing boxes.

Parameters

The following are the ni_quadra_drawbox parameters:

x / y

The expressions which specify the top left corner coordinates of the box. It defaults to 0.

width, w
height, h

The expressions which specify the width and height of the box; It defaults to 0.

color, c

Specify the color of the box to write.

Default: “black”

x1 y1 w1 h1

Those expressions are the same as x y w and h, but for box 1.

x2 y2 w2 h2

Those expressions are the same as x y w and h, but for box 2.

x3 y3 w3 h3

Those expressions are the same as x y w and h, but for box 3.

x4 y4 w4 h4

Those expressions are the same as x y w and h, but for box 4.

is_p2p

Specifies if output buffer of the filter is p2p buffer. When is_p2p is set to 1, the output buffer of the filter is set as p2p buffer and can be read out by p2p read. Else the output buffer is normal and cannot perform p2p.

Supported Values: Bool 0 and 1.
Default: 0

The following example shows how to use drawbox. The decoder decodes a H.264 input using the h264_ni_quadra_dec, ni_quadra_scale converts yuv420p to rgba, ni_quadra_drawbox draws a red box at x=100, y=100, w=1720, h=880 and ni_quadra_scale converts rgba to yuv420p before passing it to h264_ni_quadra_enc for encoding.

```
ffmpeg -vsync 0 -c:v h264_ni_quadra_dec -xcoder-params 'out=hw' -i input.h264 -vf '
ni_quadra_scale=iw:ih:format=rgba,ni_quadra_drawbox=x=100:y=100:h=1720:w=880:c=red:x
1=200:y1=100:w1=300:h1=200,ni_quadra_scale=iw:ih:format=yuv420p ' -c:v
h264_ni_quadra_enc -c:a copy output.h264
```

10.12 ni_quadra_drawtext

The ni_quadra_drawtext filter is based on FFmpeg's native drawtext filter. It draws text on top of a video frame using the libfreetype library and works with NETINT hardware frame only. The ni_quadra_drawtext filter parameters are identical to those of FFmpeg drawtext filter when drawing only one drawtext. Reference FFmpeg documentation of drawtext parameters for details.

The ni_quadra_drawtext filter only supports NETINT hardware AVFrame as input and output.

The ni_quadra_drawtext filter only accepts limited color range video (also known as "tv" range). While full range video (also known as "pc" range) will be accepted; the ni_quadra_drawtext filter will convert to limited color range video. This conversion will cause a slight color shift in the output video.

The drawtext filters compilation has dependency of freetype and fontconfig libraries. On Linux hosts these libraries can be installed by the following commands:

- sudo apt-get install libfreetype6-dev
- sudo apt-get install libfontconfig1-dev

In addition, building of FFmpeg/libav needs to have those libraries enabled. A configurable option (disabled by default) is added into *build_ffmpeg.sh* to enable this:

```
build_ffmpeg.sh --nidrawtext
```

NOTE: This filter supports a maximum input resolution of 7040x7040.
Altering parameters via commands is only supported for FFmpeg 4.3 and above.

Parameters

The following are the ni_quadra_drawtext parameters when drawing multiple texts simultaneously:

text/(t0-t31)

Specify the text to write.

Default: NULL

x, y/(x0-x31), (y0-y31)

The expressions which specify the top left corner coordinates of the text. It
Default: 0”

font/(f0-f31)

Specify the font of the text to write.

Default: “Sans”

fontcolor/(fc0-fc31)

Specify the color of the text to write.

Default: “black”

fontsize/(fs0-fs31)

Specify the size of the text to write.

Default: 36

fontcolor_expr/(fc_expr0 -fc_expr31)

Specify the color of the text to write by RGB.

Default:NULL

NOTE: If fontcolor_expr was set, it will overwrite the corresponding fontcolor setting.

boxcolor/(bc0-bc31)

Specify the color of the text box.

Default: white

boxborderw/(bb0-bb31)

Specify the box border width (padding)

allowed formats: "all", "vert|oriz", "top|right|bottom|left.

Default: 0

Commands

ni_quadra_drawtext filter supports altering parameters via commands, but it only supports for ffmpeg4.3 and above.

reinit

Alter existing filter parameters.

Syntax for the argument is the same as for filter invocation, e.g.

fontsize=56:fontcolor=green:text='Hello World'

If the entire argument can’t be parsed or applied as valid values then the filter will continue with its existing parameters.

The following example shows how to print a time stamp onto a transcoded H.264 stream using ni_quadra_drawtext, and can draw up to 32 texts simultaneously:

```
ffmpeg -y -c:v h264_ni_quadra_dec -xcoder-params "out=hw" -i input1080p.h264 -
filter_complex
"ni_quadra_drawtext=font=Sans:fontcolor=White:text='%{pts¥:localtime¥:1456007118}':x=1050
:y=680[out]" -map [out] -c:v h264_ni_quadra_enc output.h264
```

```
ffmpeg -y -c:v h264_ni_quadra_dec -xcoder-params "out=hw" -i input1080p.264    -vf
"ni_quadra_drawtext=f0=Sans:fc_expr0=#0000ff:fs0=36:t0=hello:x0=100:y0=100:f1=Sans:t1=hell
o1:fc_expr1=#00ff00:fs1=36:x1=300:y1=100" -c:v h264_ni_quadra_enc output2.h264
```

```
ffmpeg -y -c:v h264_ni_quadra_dec -xcoder-params "out=hw" -i input1080p.h264 -
filter_complex
"ni_quadra_drawtext=f0=Sans:fs0=36:t0=hello:x0=100:y0=100:t1=hello1:fs1=36:x1=300:y1=100:
t2=hello2:fs2=36:x2=500:y2=100:t3=hello3:fs3=36:x3=700:y3=100:t4=hello4:fs4=36:x4=900:y4=
100:t5=hello5:fs5=36:x5=1100:y5=100:t6=hello6:fs6=36:x6=1300:y6=100:t7=hello7:fs7=36:x7=1
500:y7=100:t8=hello8:fs8=36:x8=1700:y8=100:t9=hello9:fs9=36:x9=100:y9=200:t10=hello10:fs1
0=36:x10=300:y10=200:t11=hello11:fs11=36:x11=500:y11=200:t12=hello12:fs12=36:x12=700:y1
2=200:t13=hello13:fs13=36:x13=900:y13=200:t14=hello14:fs14=36:x14=1100:y14=200:t15=hell
o15:fs15=36:x15=1300:y15=200:t16=hello16:fs16=36:x16=1500:y16=200:t17=hello17:fs17=36:x
17=1700:y17=200:t18=hello18:fs18=36:x18=100:y18=500:t19=hello19:fs19=36:x19=300:y19=50
0:t20=hello20:fs20=36:x20=500:y20=500:t21=hello21:fs21=36:x21=700:y21=500:t22=hello22:fs
22=36:x22=900:y22=500:t23=hello23:fs23=36:x23=1100:y23=500:t24=hello24:fs24=36:x24=130
0:y24=500:t25=hello25:fs25=36:x25=1500:y25=500:t26=hello26:fs26=36:x26=1700:y26=500:t27
=hello27:fs27=36:x27=100:y27=900:t28=hello28:fs28=36:x28=300:y28=900:t29=hello29:fs29=36
:x29=500:y29=900:t30=hello30:fs30=36:x30=700:y30=900:t31=hello31:fs31=36:x31=900:y31=90
0[out]" -map [out] -c:v h264_ni_quadra_enc output.h264
```

10.13 ni_quadra_bgr

The ni_quadra_bgr background removal filter analyses input frames, inferences these images using the AI module (segm32), segments the foreground and background of the input images, and then removes the background. The output is kept in RGBA format to allow later overlay to judge fore and background based on value of alpha channel.

The filter takes a segm_32 object segmentation model as input, down to the AI module specified by the user. The imported model will be unfolded and initialized in memory. As for the segm32 model, the dimensions of the input and output are as defined. The segm32 model requires a tensor with shape 256 x 144 (width x height) and a pixel format of BGR Planar. In this case, the filter needs to scale the input frames and do the format translation to adapt to the different resolutions. Currently the filter only supports hardware frames. The 2D Engine is used to do the scaling and format conversion and then the result is passed to the AI module to perform the inferences.

When the inference completes the output tensor is then retrieved by the filter which then performs some post-processing to segment the foreground and background. The result of the post-processing is the alpha data is posted to the alpha channel of the RGBA output. This is then returned back to Quadra HW in the form of a hwupload such that the filter retains HWframe pixel format.

The ni_quadra_bgr filter only supports limited color range video (also known as “tv” range). While full range can be passed through the filter, the ni_quadra_bgr filter will always convert to limited color range output. This will cause a slight color shift in the output video.

Parameters

The following are the ni_quadra_bgr parameters:

nb

Specifies the AI module (segm32.nb) network binary file path including the file name. This field is required for the filter to work.
Supported Values: String
Default: Null

skip

Specifies the number of frames to skip between inference. When set to 0, all frames are inferenced. When set to 1, every 2nd frame is inferenced. When set to 2, every 3rd frame is inferenced.

Supported Values: 0 or higher
Default: 0

The following example shows using the ni_quadra_bgr filter to overlay input stream on top of background image.

```
ffmpeg –vsync 2 -i theatre.bmp ¥
-c:v h264_ni_quadra_dec -dec 0 -xcoder-params 'out=hw:scale0=274x154' -i ken0.h264 ¥
-c:v h264_ni_quadra_dec -dec 0 -xcoder-params 'out=hw:scale0=274x154' -i ken1.h264 ¥
-c:v h264_ni_quadra_dec -dec 0 -xcoder-params 'out=hw:scale0=274x154' -i ken2.h264 ¥
-filter_complex "[0:v]format=rgba,ni_quadra_hwupload=0[a0]; ¥
[1:v]ni_quadra_bgr=nb=segm32_tflite.nb:skip=1[a1]; ¥
[2:v]ni_quadra_bgr=nb=segm32_tflite.nb:skip=1[a2]; ¥
[3:v]ni_quadra_bgr=nb=segm32_tflite.nb:skip=1[a3]; ¥
[a0][a1][a2][a3]ni_quadra_xstack=inputs=4: ¥
layout=0_0 ¥
|600_558|810_558|1026_558: ¥
size=1920_1080 ¥
|274_154|274_154|274_154, ¥
ni_quadra_scale=format=yuv420p[out]" ¥
-map "[out]" ¥
-c:v h264_ni_quadra_enc -enc 0 xstack_3_ppl_theatre.mkv
```

10.14 ni_quadra_ai_pre

The ni_quadra_ai_pre AI pre-processing provides a filter to do picture pre-processing by AI model. The pixel format and resolution depend on the AI model and the input of picture.

The filter takes a segm_32 object segmentation model as input, down to the AI module specified by the user. The imported model will be unfolded and initialized in memory. As for the segm32 model, the dimensions of the input and output are as defined. The segm32 model requires a tensor with custom designed and should match the input resolution.

The input frames can be SW/HW frames. The output frame must be HW frames.

Parameters

The following are the ni_quadra_ai_pre parameters:

nb

Specifies the AI model (segm32.nb) network binary file path including the file name. This field is required for the filter to work.
Supported Values: String
Default: Null

devid

Specifies the device index of card. This parameter should be set when the input is software frame.
Supported Values: Integer
Default: 0

mode

Specifies the AI processing mode. mode=0 means the AI model will process the whole YUV, while mode=1 means the AI model will only process Y channel and keep U and V unchanged. Users should specify network binary file according to the mode.
Supported Values: 0(YUV), 1(Y only)
Default: 0

keep_alive_timeout

Specifies a session keep alive timeout value. This is a periodic request/response between libxcoder and XCoder firmware that when timed out, terminates the session instance in the XCoder firmware.
Supported Values: Integer in the range 1 to 100
Default: 10

buffer_limit

Whether to limit the number of output buffers. If enabled, the buffering count of the filter is its buffer pool size (default 4) plus encoding buffering count if its downstream is encoder. It is enabled by default in FFmpeg-n6.1 and FFmpeg-n7.1.
Supported Values: Integer 0 and 1.
Default: 0

width

Specify the output frame width. For some nb model the input and output frame width is not same, in this case, the width should be set so that filter can't know and check the output from AI engine.
Supported Values: Integer 0 and 8192.
Default: -1

height

Specify the output frame height. For some nb model the input and output frame height is not same, in this case, the width should be set so that filter can't know and check the output from AI engine.
Supported Values: Integer 0 and 8192.
Default: -1

timeout

Specifies timeout value for processing one frame. This is the timeout checking in ffmpeg's software. It reminds developers the AI processing may take too much time and should increase this timeout and all the keep_alive_timeout on the pipeline.
Supported Values: Integer in the range 1 to 100
Default: 3

The following example shows using the ni_quadra_ai_pre filter to do picture pre-processing on 1080p clip transcode.

```
ffmpeg -y -loglevel info -c:v h264_ni_quadra_dec -dec 0 -xcoder-params "out=hw" -i Dinner_1920x1080p30_300.h264 -vf ni_quadra_ai_pre=nb=usm_1080P_level1.nb -c:v h265_ni_quadra_enc -enc -1 -xcoder-params "RcEnable=1:bitrate=1000000" ai_1080p_300.h264
```

NOTE:

The ai_pre_process filter supports SW/HW input frames and HW output frames, and also has different modes. So, there are various combinations of parameters. The setting of these parameters mainly depends on the AI models.

The resolution of the input frame must match the input resolution of the AI model. If they do not match, report an “init” error.
If the resolutions of input frames and output frames are not the same, the width/height parameters must be used to specify the resolution of the output frames. For example, the input resolution of the AI model "hw_bicubic_network_binary.nb" is 1080p, and the output resolution is 720p.
Mode 0 is a normal mode. Mode 1, however, is a special mode that modifies only the Y channel of HW frames and must be used in conjunction with a specific AI model. For example, the AI model "network_binary_no_noise_ml.nb" modifies the Y channel of HW frames, while the U/V channels are transmitted through the FW.

10.15 ni_quadra_delogo

The ni_quadra_delogo filter allows you to blur the area you selected as delogo do. You can select the start point and area to blur. ni_quadra_delogo only supports hardware AVFrames as input and output.

The ni_quadra_delogo filter only supports limited color range video (also known as “tv” range). While full-range video (also known as “pc” range) is accepted, the ni_quadra_delogo filter will always convert to limited color range. This will cause a slight color shift in the output video.

Parameters

The following are the ni_quadra_delogo parameters:

x / y

The expressions which specify the top left corner coordinates of the delogo area. It defaults to 0.

w / h

The expressions which specify the width and height of the delogo area;

buffer_limit

Whether to limit the number of output buffers. If enabled, the buffering count of the filter is its buffer pool size(default 4) plus encoding buffering count if its downstream is encoder. It’s enabled by default in FFmpeg-n6.1 and FFmpeg-n7.1.
Supported Values: Integer 0 and 1.
Default: 0

The following example shows how to use delogo. The decoder decodes a H.264 input using the h264_ni_quadra_dec, ni_quadra_delogo blurs an area with x=600, y=400, w=300, h=200 and pass it to h265_ni_quadra_enc for encoding. It does not support iw and ih input parameters.

```
ffmpeg -y -loglevel info -c:v h264_ni_quadra_dec -dec 0 -xcoder-params "out=hw" -i Dinner_1920x1080p30_300.h264 -vf ni_quadra_delogo=600:400:300:200 -c:v h265_ni_quadra_enc –enc 0 delogo.h265
```

10.16 ni_quadra_merge

The ni_quadra_merge filter will merge decoder ppu0 and ppu1 as one frame. The inputs are from decoder output directly with hardware AVFrames. The filter uses ppu0 Y as Y data and ppu1 UV as UV data, scale the Y size to UV size and merge into one. The ppu1 AVFrame cannot be used as Mult inputs because the Y data is changed by merge filter. The ppu0 and ppu1 must have the same format. The inputs are only support YUV420P, NV12 YUV420P10LE and P010LE format. ni_quadra_merge only supports hardware AVFrames as input and output.

The ni_quadra_merge filter only accepts limited color range video (also knowns as “tv” range). Full range video (or “pc” range) is accepted; however, the ni_quadra_merge filter will always convert to limited color range video. This will cause a slight color shift in the output video.

Parameters

The following are the ni_quadra_merge parameters:

Filterblit

Specifies the merge algorithm. The default is a simple blit function that uses an algorithm similar to the nearest neighbor algorithm. When the filterblit parameter is set to 1 then the filterblit function will be used for scaling. The filterblit function uses an algorithm that is similar in quality to the bicubic algorithm. When the filterblit parameter is set to 2, a bicubic algorithm is used. The parameter only applies ppu0 Y data.

Supported Values: Integer in the range 0 to 2
Default: 0

buffer_limit

Whether to limit the number of output buffers. If enabled, the buffering count of the filter is its buffer pool size(default 4) plus encoding buffering count if its downstream is encoder. It’s enabled by default in FFmpeg-n6.1 and FFmpeg-n7.1.

Supported Values: Integer 0 and 1.
Default: 0

param_b

Set parameter B of bicubic.This param only effect when filterblit set to 2.
NOTE: Please try to use the same value on the same device. Otherwise, it may have some impact on FW load

Supported Values: Double 0 to 1.

Default: 0

param_c

Set parameter C of bicubic.This param only effect when filterblit set to 2.
NOTE: Please try to use the same value on the same device. Otherwise, it may have some impact on FW load

Supported Values: Double 0 to 1.

Default: 0.75

The following example shows how to use merge. The decoder decodes a H.264 input using the h264_ni_quadra_dec, ni_quadra_merge will scale ppu0 Y to the same size as ppu1, merge ppu0 Y and ppu1 UV and pass it to h265_ni_quadra_enc for encoding.

```
ffmpeg -c:v h264_ni_quadra_dec -dec 0 -xcoder-params
out=hw:semiplanar0=1:enableOut1=1:scale1=1280x720:semiplanar1=1 -i test.h264 -vf
“ni_quadra_merge=filterblit=2”    -c:v h265_ni_quadra_enc –enc 0 merge.h265
```

10.17 ni_quadra_flip

The ni_quadra_flip filter flips the input video horizontally or vertically according to flip_type parameter. ni_quadra_flip only supports hardware AVFrames as input and output.

The ni_quadra_flip filter only accepts limited color range (also known as “tv” range). Full range video (also known as “pc” range) is accepted; however, the ni_quadra_flip filter will always convert to limited range. This will cause a slight color shift in the video output.

Parameters

The following are the ni_quadra_flip parameters:

flip_type

The expressions specify the flip type of picture. 0 for horizontally flipping and 1 for vertically flipping. It defaults to 0.

buffer_limit

Whether to limit the number of output buffers. If enabled, the buffering count of the filter is its buffer pool size(default 4) plus encoding buffering count if its downstream is encoder. It’s enabled by default in FFmpeg-n6.1 and FFmpeg-n7.1.

Supported Values: Integer 0 and 1.
Default: 0

The following example shows how to use a flip filter. The decoder decodes an H.264 input using the h264_ni_quadra_dec, ni_quadra_flip flips the input video horizontally and passes it to h265_ni_quadra_enc for encoding.

```
ffmpeg -y -c:v h264_ni_quadra_dec –dec 0 -xcoder-params "out=hw" -i test.h264 -vf
ni_quadra_flip=flip_type=0 -c:v h265_ni_quadra_enc -enc 0 test.h265
```

10.18 ni_quadra_hvsplus

The ni_quadra_hvsplus filter utilizes Quadra's internal AI engine to enhance video quality by mimicking the human visual system at the pre-processing stage before encoding, thereby improving VMAF (Video Multi-Method Assessment Fusion) scores. It optimizes video content through perceptual optimization, achieving a balanced performance between PSNR and VMAF. This filter offers two preconfigured levels tailored to different types of video content, providing optimal enhancement for various genres.

Limitations

- The supported resolution for this filter is up to 4k (3840 x 2160).

Parameters

The following are the ni_quadra_hvsplus parameters:

devid

Specifies the Device ID of Quadra hardware to use when software frames are used as an input. When a hardware frame is used the filter will be collocated to the same device as the frame.
Supported Values: 0 to max device ID
Default: 0

level

Level 1 preserves PSNR better while improving VMAF. Level 2 is more aggressive in improving VMAF.
Supported Values: Integer 1 and 2.
Default: 2

The following example shows how to use a hvsplus filter. The hvsplus filter performs preprocessing of the input frames in the AI engine and sends them to h264_ni_quadra_enc for encoding. It is using the default level of 2. To change the level to 1, replace ‘ni_quadra_hvsplus’ to ‘ni_quadra_hvsplus=level=1’.

```
ffmpeg -y -i Football_1080p30.y4m -filter_complex '[0:v]ni_quadra_hvsplus,format=yuv420p' -c:v h264_ni_quadra_enc -xcoder-params "RcEnable=1:bitrate=2000000" -enc 0 test.mp4
```

11. Supported Versions of FFmpeg

The FFmpeg versions below are supported on Linux

- 3.1.1
- 3.4.2
- 4.1.3
- 4.2.1
- 4.3
- 4.3.1*
- 4.4
- 4.4.2
- 5.0
- 5.1.2
- 6.1
- 7.1

FFmpeg version 4.3.1 is also supported on Android and MacOS. FFmpeg version 4.3.1 and FFmpeg version 4.2.1 are supported on Windows. This version has been fully validated for Windows. Note that co-existence with Logan is also supported with 4.3.1.

NOTE: FFmpeg label their release versions like this

n4.3
n4.3-dev
n4.3.1
n4.3.2
n4.3.3
n4.3.4
n4.3.5
n4.3.6
n4.3.7
n4.4

Note : Not all Quadra features are supported on all versions of FFmpeg. Some features are not supported on the older versions of FFmpeg. FFMpeg 3.1.1 has the following limitations:

- Regular Linux and Kernel version support only; No support for Windows, Android, MacOS or Docker container
- Supports SW frames only, so does not support HW frames
- H.264/H.265 decode/encode/transcode only. No support for other codecs or NETINT filters. For example, HW frame upload/download, and scalers etc. are not supported
- HDR10 user-specified mastering display color volume and content light level values only
- HDR10+ or DolbyVision is not supported

The feature descriptions throughout this document will state which FFmpeg versions are supported. If no FFmpeg version is listed, the feature will be supported in **all** FFmpeg versions **on Linux**.

The following table lists all features supported on each FFmpeg version.

Feature		3.1.1	3.4.2	4.1.3	4.2.1 4.3	4.3.1	4.4 5.0	4.4.2 5.1.2	6.1	7.1
Decoder	H264	√	√	√	√	√	√		√	√
	H265	√	√	√	√	√	√		√	√
	VP9	×	√	√	√	√	√		√	√
	JPEG	×	√	√	√	√	√		√	√
Encoder	H264	√	√	√	√	√	√		√	√
	H265	√	√	√	√	√	√		√	√
	AV1	×	×	√	√	√	√		√	√
	JPEG	×	√	√	√	√	√		√	√
Dec Xcode	All xcoder params	√	√	√	√	√	√		√	√
Enc Xcode	dolbyVisionProfile	×	×	×	×	√	√		√	√
	Others xcoder Params	√	√	√	√	√	√		√	√
Filters	ni_quadra_scale	√	√	√	√	√	√		√	√
	ni_quadra_overlay	√	√	√	√	√	√		√	√
	ni_quadra_split	√	√	√	√	√	√		√	√
	ni_quadra_crop	√	√	√	√	√	√		√	√
	ni_quadra_pad	√	√	√	√	√	√		√	√
	ni_quadra_hwupload	√	√	√	√	√	√		√	√
	ni_quadra_roi	√	×	×	√	√	√		√	√
	ni_quadra_bg	√	√	√	√	√	√		√	√
	ni_quadra_bgr	√	√	√	√	√	√		√	√
	ni_quadra_xstack	×	×	√	√	√	√		√	√
	ni_quadra_rotate	√	√	√	√	√	√		√	√
	ni_quadra_drawbox	√	√	√	√	√	√		√	√
	ni_quadra_drawtext	√	√	√	√	√	√		√	√
	ni_quadra_ai_pre	√	√	√	√	√	√		√	√
	ni_quadra_delogo	√	√	√	√	√	√		√	√
	ni_quadra_merge	√	√	√	√	√	√		√	√
	ni_quadra_flip	√	√	√	√	√	√		√	√
	ni_quadra_hvsplus	√	√	√	√	√	√		√	√
Advanced Feature	HDR	Refer to 12.1, dolbyVisionProfile is supported from 4.3.1.								
	ROI	Refer to 12.2, support from 4.2.1								
	Closed Captions	√	√	√	√	√	√		√	√
	Rate Control	√	√	√	√	√	√		√	√
	User Data,UnregSEIPassthrough	√	√	√	√	√	√		√	√
	IDR Frame Forcing	√	√	√	√	√	√		√	√
	SCTE 35	×	×	×	×	×	×		√	√
Others	AV1 tile	×	×	×	√	√	√		√	√
	HEVC tile	×	×	×	√	√	√		√	√

12. Advanced Feature Support

12.1 HDR

Quadra completely supports 3 HDR standards, HLG, HDR10 and HDR10+ for H.264, H.265, and AV1. These standards all use 10 bit color for greater dynamic range, a wider range of colors as per ITU-R BT.2020. For Dolby Vision, Quadra supports a compatibility mode such that the Dolby Encoding Engine can use the Quadra encoder for single base layer profile 5 Dolby Vision encoding with H.265. This mode is enabled by setting the encoding parameter dolbyVisionProfile=5.

HDR10/10+ uses a Perceptual Quantization transfer curve as per SMPTE ST 2084 that supports a much larger range of brightness but is not backwards compatible with standard dynamic range (SDR). The colors of HDR10/10+ content played back on an SDR monitor will appear very faded. HLG on the other hand uses the ARIB STD-B67 transfer curve which provides greater dynamic range at high brightness and is backward compatible with the SDR gamma curve at low brightness and so an HLG stream can be played on both SDR and HDR monitors.

The HDR colour information is carried in the VUI for H.264 and H.265 and in the metadata OBU for AV1.

The following 3 standards specify the HDR colour information:

Standard	**VUI Color Information**
HLG ATSC A/341	color_primaries=9 (ITU-R BT.2020-2 Wide Gamut Color) transfer_characteristics=18 (ARIB STD-B67 HLG Transfer Curve) matrix_coeffs=9 (ITU-R BT.2020-2 Non-constant Luminance)
HLG ETSI ETSI TS 101 154	color_primaries=9 (ITU-R BT.2020-2 Wide Gamut Color) transfer_characteristics=14 (ITU-R BT.2020-2 Functionally equivalent to BT.709) matrix_coeffs=9 (ITU-R BT.2020 Non-constant Luminance)
HDR10/10+	color_primaries=9 (ITU-R BT.2020-2 Wide Gamut Color) transfer_characteristics=16 (SMPTE ST2084 PQ Transfer Curve) matrix_coeffs=9 (ITU-R BT.2020-2 Non-constant Luminance)

HDR bitstreams may also carry static metadata containing the parameters of the mastering display using content light level info, and the mastering display color volume.

HDR10+ adds dynamic metadata that can update the color information on a frame by frame basis. This metadata is stored in T35 payloads as per SMPTE 2094-40. The HDR metadata is stored using SEIs for H.264 and H.265 or equivalent metadata OBU types for AV1.

There are no special commands to enable HDR transcoding. The Quadra decoder will pass HDR color information and HDR metadata up to FFmpeg if the bitstream contains it, and the Quadra encoder will insert the HDR color information and metadata in the encoded bitstream if supplied by FFmpeg. Transcoding a compliant HDR10 bitstream will result in a compliant HDR10 bitstream. The same for HDR10+ and HLG.

FFmpeg supports specifying the color information on the command line with 3 parameters that map to the VUI color parameters as follows. These parameters may be specified in the input or output sections of the FFmpeg command line. If the color information is specified on the command line, it will replace any color information that is contained in the input media.

These will be properly set by the decoder if transcoding.

FFmpeg Color Parameter	**VUI Color Parameter**
color_primaries	color_primaries
color_trc	transfer_characteristics
colorspace	matrix_coeffs

The following is an example FFmpeg command line to encode a 10 bit HLG YUV file to H.265 as per ATSC requirements:

```
ffmpeg -f rawvideo -pix_fmt yuv420p10le -s:v 3840x2160 -r 60 -color_primaries 9 -color_trc 18 -colorspace 9 -i Input_3840x2160_10bit_le.yuv -enc 0 -c:v h265_ni_quadra_enc -xcoder-params "RcEnable=1:bitrate=20000000" outputATSCHlgT408.265
```

Note: While ETSI specifies transfer characteristics=14 for HLG in the VUI, they also specify the inclusion of an alternative transfer characteristics SEI that specifies a preferred transfer characteristics of 18. The Netint decoder will return the preferred transfer characteristics instead of the VUI transfer characteristics if this SEI is present. The NETINT Encoder has a parameter (**prefTRC**) to specify the inclusion of this SEI, and to set its value.
For example, the following command line to encode a 10 bit HLG YUV file to H.265 as per ETSI requirements is as follows:

```
ffmpeg -f rawvideo -pix_fmt yuv420p10le -s:v 3840x2160 -r 60 -color_primaries 9 -color_trc 14 -colorspace 9 -i Input_3840x2160_10bit_le.yuv -enc 0 -c:v h265_ni_quadra_enc -xcoder-params "RcEnable=1:bitrate=20000000:prefTRC=18" outputETSIHlgT408.265
```

Note: FFmpeg does not currently support specifying the static and dynamic metadata for HDR10/10+.

An example of HDR transcoding between H.265 to H.264 is as follows. If the input is 10 bits, then the output will be 10 bits. Any HDR VUI color information from the input bitstream will be transferred to the output bitstream. Any static or dynamic HDR10/10+ metadata from the input bitstream will be transferred to the output bitstream. When a ETSI HLG bitstream is decoded, the preferred transfer characteristics will be used in the VUI of the output bitstream.

```
ffmpeg -c:v h265_ni_quadra_dec -dec 0 -i inputHDR.ts -c:a copy -enc 0 -c:v h264_ni_quadra_enc -xcoder-params "RcEnable=1:bitrate=20000000" outputHDR.ts
```

If an ETSI compliant output bitstream is required then the VUI transfer characteristics can be overwritten on the command line and the preferred transfer characteristics specified.

```
ffmpeg -c:v h265_ni_quadra_dec -dec 0 -i inputHDR.ts -c:a copy -color_trc 14 -c:v h264_ni_quadra_enc -enc 0 -xcoder-params "RcEnable=1:bitrate=20000000: prefTRC=18" outputHDR.ts
```

Note: HLG is supported in all supported versions of FFmpeg. HDR10 is supported in FFmpeg version 4.1.3 or higher, while HDR10+ and Dolby Vision compatibility are supported in FFmpeg 4.2.1 or higher.

12.2 Region of Interest (ROI)

ROI is a feature of the encoder that permits the quality of some regions to be improved at the expense of other regions. This is achieved by specifying an ROI map containing the QP (0-51) for each 16x16 pixel block for H.264, and 32x32 pixel block for H.265, or 64x64 pixel block of AV1.

A higher QP means lower quality, a lower QP means higher quality. If rate control is disabled, the QPs are used directly for encoding, if rate control is enabled, the encoder scales the QPs as necessary to meet the bitrate target. When ROI is enabled, the ROI map can be updated, enabled, or disabled on a frame by frame basis.

As of version FFmpeg 4.2.1, FFmpeg supports an API for ROI that permits a number of rectangular ROI regions to be defined along with a QP Offset in the range of -1 to +1. The FFmpeg QP Offset corresponds to a QP Offset of -25 to +25 on the encoder. As of version FFmpeg 4.3.1, FFmpeg supports an ROI filter (addroi) that permits a number of ROI regions to be specified on the command line. Unfortunately, this filter is fairly limited since it does not permit the ROI regions to be updated on a frame by frame basis. For more detail see the Region of Interest application note.

The FFmpeg Region Of Interest (ROI) filter inferences from input frames using the in-built AI module in Quadra. It identifies the bounding coordinates of chosen objects and classes within images, and then wraps the coordinates into ROI side data.

The filter loads a YOLOv4 object detection model into the user specified AI module. This model will then be unfolded and initialized in memory. The input and output dimensions are also defined.

YOLOv4 models require a tensor with shape 416x416 and pixel format BGRPlanar as the input. This means that the filter needs to scale the input images and perform the format conversion.

The filter supports both hardware frames and software frames.

12.2.1 Software Frame

For software frames, only YUV420P is supported as the input. The SWS library in FFmpeg is used to perform software scale. FFmpeg doesn't support BGRPlanar, and so the filter first scales the input image to a 416x416 RGB24 AVFrame, it then rearranges the frame to a compact RGRPlanar tensor, and then transfers the tensor down to the AI module to perform the inference.

12.2.2 Hardware Frame

For hardware frames, because Quadra 2D supports BGRPlanar output in the format conversion, the filter passes hardware frames directly to the 2D engine to perform scaling. The scaled image is stored in another hardware frame. Once scaling completes, it passes the scaled hardware frame over to the AI module for inference.

Before inference begins, an output buffer is preset in the model. When the inference is complete the output tensor will be saved in the output buffer. The filter then fetches the output tensor from firmware. It executes post processing on the output tensor to compute the object boxes and class labels inside the images. If there are any objects detected, their bounding boxes are converted to four coordinates relative to the upper left zero point of the images. Each objects' coordinates will be put into the ROI side data including a preset QP offset. All ROI side data within an image is appended to, then passed down to the encoder along with the actual images themselves.

12.2.3 Parameters

The table below defines the parameters supported by the **ni_roi** filter.

Table 1: Quadra ni_roi Parameters

Parameter	Values	Description
nb	String Default: NULL	Specify the full filename and path of the AI modules network binary. Without this required field, this filter won't work.
qpoffset	[-1.0, 1.0] Default: 0.0	Specify the ROI QP offset so that the encoder can set the specific QP in these regions based on the QP offset. ROI side data structure is defined in FFmpeg/ libavutil/frame.h, named struct AVRegionOfInterest.
devid	[-1, 2147483647] Default: 0	Specify the Device ID of the Quadra device to use. A value of -1 will collocate the instance on the same device as the input YUV hardware frame. If there are not enough resources, or if the device ID is not specified then the resource monitor will decide where to place the instance.
obj_thresh	[0,1.0] Default: 0.25	Specify the YOLOv4 post processing object threshold. Each region has a score. When the score reaches the obj_thresh, it will be specified as an object. The higher the threshold, the harder it is to be defined as an object.
nms_thresh	[0,1.0] Default: 0.45	Specify the YOLOv4 post processing NMS IOU threshold. The goal is to select the bounding boxes with the highest detection probability and eliminate all the bounding boxes whose intersection Over Union value is higher than a given IOU threshold.

12.2.4 Examples

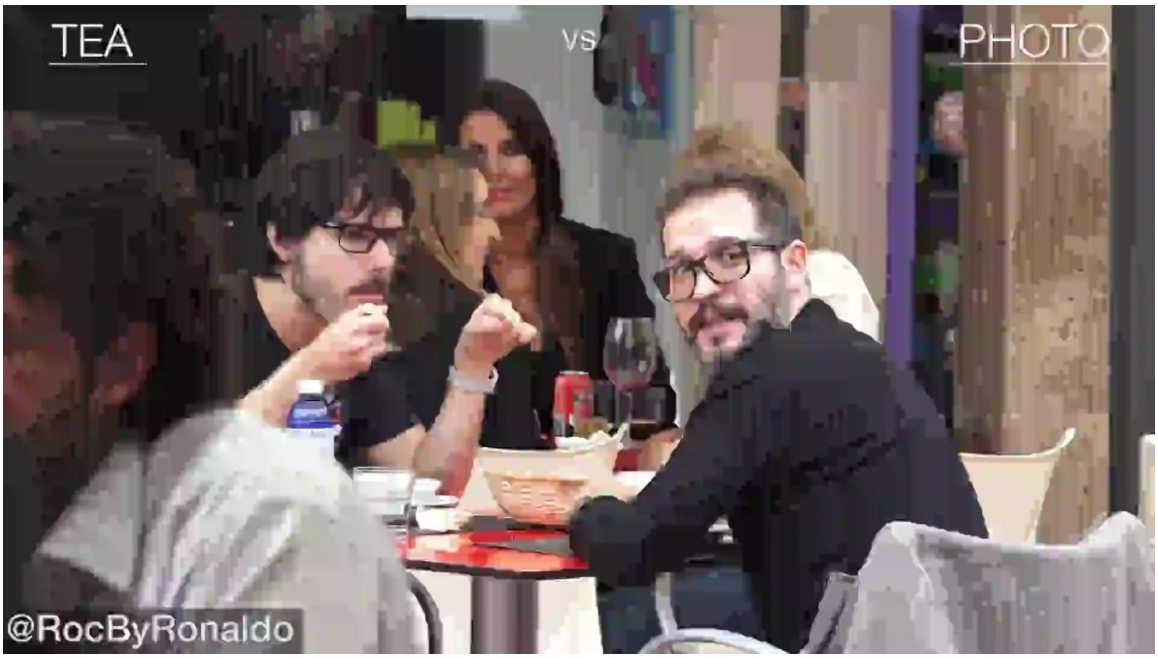
The command line below shows an example using the ROI filter during transcoding with hardware frame mode.

ffmpeg -y -vsync 0 -init_hw_device ni=foo:0 -dec 0 -c:v h264_ni_quadra_dec -xcoder-params 'out=hw' -i cr7_1920x1080.h264 -filter_hw_device foo -vf 'ni_roi =nb=./network_binary_yolov4_head.nb:qpoffset=-0.6' -enc 0 -c:v h264_ni_quadra_enc -xcoder-params 'roiEnable=1:RcEnable=1:bitrate=500000' -an cr7_1080p_roi_b500000.h264

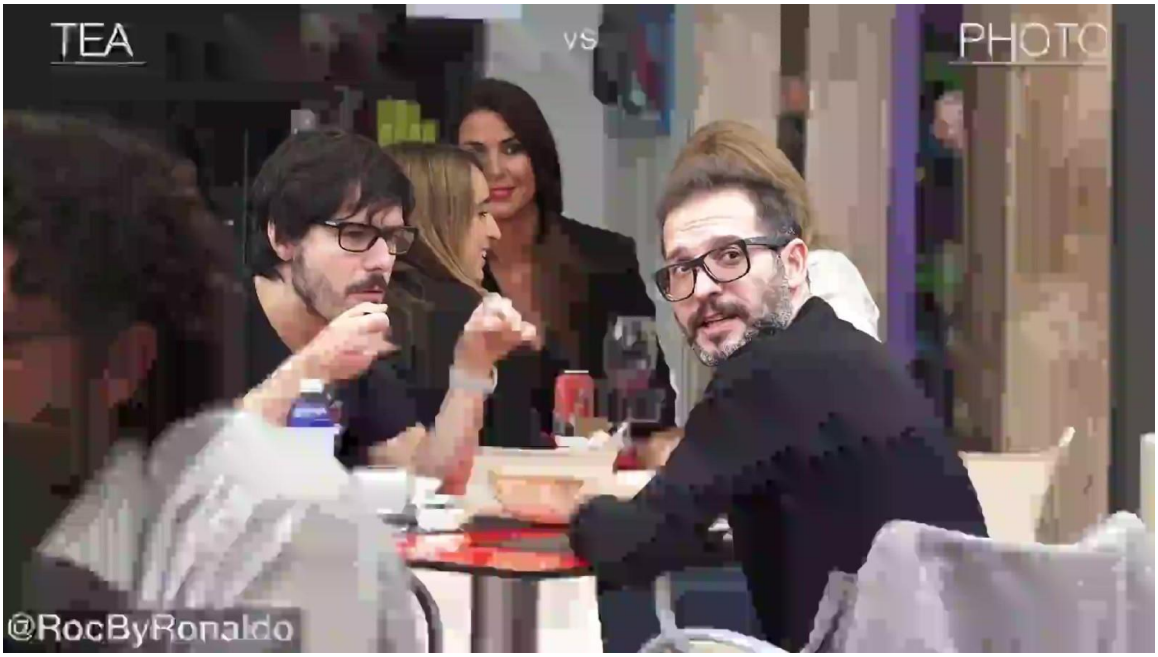
The following command line shows a software frame mode example.

ffmpeg -y -vsync 0 -dec 0 -c:v h264_ni_quadra_dec -i cr7_1920x1080.h264 -vf 'ni_roi =nb=./network_binary_yolov4_head.nb:qpoffset=-0.6' -enc 0 -c:v h264_ni_quadra_enc -xcoder-params 'roiEnable=1:RcEnable=1:bitrate=500000' -an cr7_1080p_roi_b500000.h264

The following comparison illustrates the improvements when using an ROI filter. The top picture is without an ROI filter, and the bottom picture is with an ROI filter with a bitrate of 500kbp. The human faces in picture have a mosaic effect just like the other regions in the screenshot



The image below, with an ROI, has no mosaic effect on the faces



This image is zoomed in from a screenshot with ROI to show the QP of the human face and areas surrounding it.

The **qpoffset** is set as '-0.6', which means the QP of the ROI has an offset value of -15 from its preset QP.

The non-human face MBs have a QP value of 32, while the human face MBs have a QP value of 17.

12.3 Closed Captions

Quadra supports EIA CEA-708 closed captions for H.264, H.265, and AV1. There are no special encoder parameters to set, the Quadra decoder automatically passes closed captions up to FFmpeg if present in the bitstream and the Quadra encoder will automatically insert closed captions in the encoded bitstream if they are present in the incoming stream to encoder. FFmpeg stores CE708 closed captions as ATSC A53 Part 4 Closed Captions side data. Closed captions are stored in the encoded bitstreams as T.35 payloads formatted according to CEA-708.

12.4 Rate Control

There are 5 rate control modes supported by the NETINT encoder:

CQP: Constant QP mode, enabled by setting RCEnable=0, uses a fixed QP specified by “intraQP” for I-frames plus an offset defined in the GOP structure for other frames. This mode is usually used for encoder quality evaluation and is not recommended to achieve the best encoding efficiency. By default, “RcEnable” parameter is 0 which means CQP mode.

CRF: Constant Rate Factor Mode, enabled by setting the rate factor parameter crf. With CRF the encoder varies the bitrate to maintain constant subjective quality.

Capped CRF: Capped Constant Rate Factor Mode, enabled by setting the rate factor parameter crf together with bitrate, vbvBufferSize , vbvMaxRate (optional), vbvMinRate (optional). Capped CRF adds bitrate constraint on top of CRF. Please refer to Section “8.3.4 CRF & Capped CRF Descriptions and Examples” for more details.

CBR: Constant Bitrate Mode, enabled by setting RCEnable=1 and vbvBufferSize>0, varies the QP on a frame by frame basis to maintain bitrate as set by “bitrate” parameter and to constrain instant bitrate by video buffering verifier as set by “vbvBufferSize” parameter. In this mode, the encoder buffers up an amount of bitstream as specified by the vbvBufferSize parameter to perform rate control. This buffer is typically known as a video buffering verifier. The larger it is, the better for rate control, but this comes with an increase in delay and less constrained instant bitrate.

- Please also note when cuLevelRCEnable=1 (enable block level rate control), and lookaheadDepth=0 (no lookahead), encoder perceives bitrate parameter as maximum bitrate or average bitrate depending on the bitrateMode parameter - please refer to bitrateMode parameter descriptions for details.

ABR: Average Bitrate Mode, enabled by setting RCEnable=1 and vbvBufferSize=0, varies the QP on a frame by frame basis to maintain an average bitrate as set by the “bitrate” parameter. In ABR mode, rate control maintains average bitrate to match target bitrate, but is not constrained by VBV buffer, and therefore instant bitrate may have more fluctuations compared to CBR. On the other hand, ABR may produce bitrate more closely matching the target bitrate.

Constrained VBR Mode: Constrained Variable Bitrate Mode, enabled by setting RCEnable=1, vbvBufferSize > 0, and vbvMaxRate > 0, allows higher instant bitrate, while still maintaining average bitrate close to target bitrate. Compared to CBR mode, Constrained VBR mode may produce higher quality, at the cost of higher peak rate.

12.5 User Data Unregistered SEI Passthrough

Quadra supports passthrough of user data unregistered SEI payloads during transcoding for H.264 and H.265. This can be enabled by specifying the decoder codec parameter user_data_sei_passthru as per the following example:

```
ffmpeg -c:v h264_ni_quadra_dec –user_data_sei_passthru 1 -i input.264 -c:v
h265_ni_quadra_enc output.265
```

This feature is intended for passing through small user data unregistered SEI messages up to 1024 bytes in size. For more details on user data passthrough please see the Application Note.

12.6 IDR Frame Forcing

The QUADRA encoder supports forcing IDR frames at any point. Forcing an IDR is useful for several reasons:

- When doing commercial substitution, an I-frame is required in the bitstream upon returning from the commercial. This frame will likely not coincide with the intra period and so a forced IDR frame can be used.
- Another application is to force IDRs in the transcoded bitstream at the same period as in the input bitstream.

FFmpeg supports the forcing of IDRs using the -force_key_frames parameter. This parameter can accept a list of frame numbers or times for forcing. It also supports regular expressions in the form of -force_key_frames ‘expr:gte(t,n_forced*REFRESH_PERIOD)’ where REFRESH_PERIOD is the refresh period in seconds (ex. 1,2,etc). The period can also be specified in frames using -force_key_frames ‘expr:gte(n,n_forced*REFRESH_FRAMES)’ where REFRESH_FRAMES is the refresh period in frames.

Note: These forced IDR frames are generated in addition to any generated by a non-zero intraPeriod parameter.

An example FFmpeg command line to encode a 1920x1080 YUV420 video to H.265 and force IDR pictures every 2 seconds (-force_key_frames). The intraPeriod parameter is set to zero so that the only I frames are the forced ones:

```
ffmpeg -f rawvideo -pix_fmt yuv420p -s:v 1920x1080 -r 30 -i input.yuv -force_key_frames
‘expr:gte(t,n_forced*2)’ -c:v h265_ni_quadra_enc
-xcoder-params “intraPeriod=0:RcEnable=1:bitrate=7500000” output.265
```

The force_key_frames parameter can also be used while transcoding to force IDR I-frames at the same positions as in the source file as shown in the following example:

```
ffmpeg -c:v h264_ni_quadra_dec -i input.264 -force_key_frames source
-c:v h265_ni_quadra_enc -xcoder-params “intraPeriod=0:RcEnable=1:bitrate=7500000”
output.265
```

To force the insertion of IDR I-frames at the locations of both IDR and non-IDR I-frames, use the enableFollowIFrame decoder parameter, as described in Section 9.1.

See the FFmpeg documentation for more information on the –force_key_frames parameter.

12.7 Sequence change

Sequence change is a feature that allows on-the-fly resolution change in the input stream.

12.7.1 Decoder

The Quadra decoder handles the sequence change automatically. The decoder detects the resolution change and reports the new resolution to upper layer (libxcoder).

The libxcoder handles this change by reallocating a data buffer based on the new picture size, it receives YUV data at the new resolution accordingly.

12.7.2 Encoder

When receiving YUV frames at different resolutions within the sequence change scenario, the default behavior of all FFmpeg versions is to auto-scale to the original resolution and then pass them on to the encoder. In this case, the Quadra encoder would proceed as normal since the received picture size does not change.

12.7.3 FFmpeg autoscale command line option

An output option **autoscale** is available (enabled by default), as a NETINT and FFmpeg patch (see References for details). If autoscale is disabled, FFmpeg won't auto insert a scale filter in the filter graph to force scaling the whole decoded stream into the same size as that of the first frame. If **noautoscale** is used, then when a sequence change is detected **nienc** will close the current Quadra encoding session, and will then start a new one at the new resolution.

An example of disabling auto scaling is as follows:

```
ffmpeg –hide_banner -vsync 0 -c:v h265_ni_quadra_dec -i sc.265 -
noautoscale -c:v h264_ni_quadra_enc sc-265-no-autoscale.264
```

The **autoscale** option is enabled by default so there is no need to specify it on the command line if its required to be enabled. If **autoscale** however needs to be disabled, then specify it as

-noautoscale

on the command line.

12.7.4 Reconfig Decoder Ppu Output Resolution

Quadra support reconfig ppu output resolution by adding new packet side data with a new header packet. This side data type is **AV_PKT_DATA_PPU_CONFIG.** When you want to reconfig the ppu output resolution, you can add the side data to the first packet which the input stream has a new resolution. And then the decoder can output the new resolution which you reconfiged.

This has a struct to reconfig the ppu resolution. It is **AVNIPpuConfig**.

Parameters:

The following are the **AVNIPpuConfig** parameters:

out_enabled:

This is enable ppu reconfig. It default to 0. If enabled, it must be set 1.

Default: 0

ppu_w ppu_h:

This is for reconfig the ppu output resolution. You can set the ppu_w to reconfig ppu output width, and ppu_h to reconfig ppu output height.

Note: This feature only supports **h264/h265.**

12.8 SCTE 35 Cue Out and Cue In

Based on the SCTE 35 2023r1 standard from Digital Program Insertion Cueing Message — SCTE, FFmpeg 6.1 has been extended to support

1.Decoding of SCTE 35 data to force keyframes on Cue Out and Cue In

2.SCTE 35 markup in HTTP Live Streaming (HLS) playlist using EXT-X-SCTE35

Decoding logic is currently limited to the following:
- splice_insert()
  - splice_event_cancel_indicator 0
  - out_of_network_indicator 0 or 1
  - duration_flag 0 or 1
    - If duration_flag is set for CUE OUT, a CUE IN will be added based on auto_return and duration and any subsequent CUE IN will be ignored
  - program_splice_flag 1 and splice_immediate_flag 0

12.9 Color space support

Color space information is stored in the encoded bitstream as metadata.

> For H.264 and H.265, color space information is stored in the *Video Usability Information* (VUI) within the *Stream Parameter Set* (SPS) data of the bitstream.
> For AV1, color space information is stored in the *Sequence Header Open Bitstream Unit* (OBU).
> For VP9, color space information is stored in the *Color Config* field of the uncompressed frame header.

When decoding and encoding YUV ($Y'C_bC_r$), the Quadra decoder and encoder is independent of color space. The Quadra VPU only needs to read or write the metadata and does not need to change color space information. The video playback device uses color space information in the bitstream to determine how to interpret the YUV as color to be displayed.

The table below lists some of the common supported color spaces (specifically, the color primaries) as defined in

ITU-T H.264 Specification (08/2021)
ITU-T H.265 Specification (V10)
VP9 Bitstream Specification v0.7-20170222 draft
AV1 Bitstream & Decoding Process Specification (V1.00 with Errata 1)

Color Space	H.264	H.265	AV1	VP9
BT.709	Y	Y	Y	Y
BT.470M	Y	Y	Y	Y
BT.470BG, BT.601-625	Y	Y	Y	Y
BT.601-525, SMPTE 170M, SMPTE 240M	Y	Y	Y	Y
FILM	Y	Y	Y	N
BT.2020	Y	Y	Y	Y
SMPTE 428	Y	Y	Y	N
SMPTE 431-2	Y	Y	Y	N
SMPTE 432-1	Y	Y	Y	N
JEDEC P22	Y	Y	N	N
EBU TECH 3213	Y	Y	Y	N

When using the 2D engine filters, all color spaces are supported. However, when dealing with YUV, the 2D engine filters will only work correctly with limited range YUV. Full range will be accepted but there will be a small color shift in its output. The 2D engine will only output limited range YUV.

12.9.1 RGB/YUV conversion

The Quadra encoder is capable of converting 8-bit RGB to YUV (R'G'B' to Y'C_bC_r) prior to encoding. The Quadra encoder only supports the BT.709 color matrix and the resulting YUV will always be full range.

The Quadra 2D engine is also capable of converting between RGB and YUV. Only BT.709 and BT.2020 color matrices are supported. When converting from YUV to RGB, the YUV must be limited range. If converting from RGB to YUV, the Quadra 2D engine will only output limited range YUV.

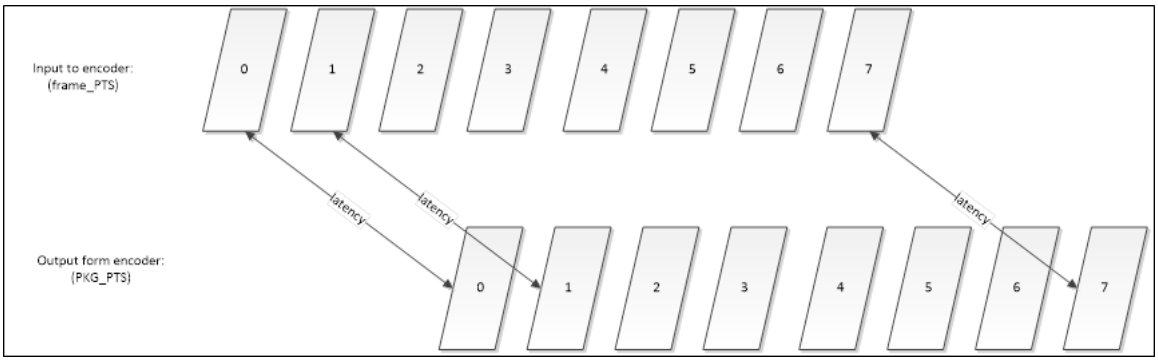
13. Performance

There are many methods for improving performance when using Quadra. This section will describe each recommended method.

13.1 Low Latency Mode

13.1.1 Encoder

Video encode latency is defined as the delay between a video frame input to the encoder and the SAME frame output from the encoder.



In the above encode latency diagram we can see the order of display frames is increasing sequentially. When there is a B frame, the DTS and PTS may be in different order.

13.1.2 GOP Requirements to Minimize Encoder Latency

As discussed above, when B frames are used the frames may need to be encoded out of order, this will increase latency. Therefore, to minimize latency we need to use a low delay GOP. A low delay GOP is one in which all the frames are encoded in sequence. On Quadra, the low delay GOPs are **gopPresetIdx**=1, 3, 7, and 9, or a custom GOP with in-sequence frames, for example where the **pocOffset** increments by 1 for each frame.

The table below lists the details of the Quadra GOP Presets.

gopPresetIdx	Description	GOP Size	Ref Frames	Encode Order	Max Frames out of Order	Low GOP
1	All I	1	2	I0-I1-I2…	0	✓
3	All B	1	2	B0-B1-B2…	0	✓
4	BP	2	2	B1-P0-B3-P2…	1	
5	BBBP	4	2	B2-B1-B3-P0	3	
7	Consecutive BBBB	4	2	B0-B1-B2-B3…	0	✓
8	BBBBBBBB	8	2	B3-B2-B4-B1-B6-B5-B7-B0…	7	
9	All P	1	1	P0-P2-P2	0	✓

You can see above that the low delay GOPs have all frames in-sequence. If a low delay GOP is not used then the encoder must buffer the incoming frames in order to encode them out of sequence, which adds to the encoder latency.

13.1.3 Encoder Low Latency Mode

The second aspect of encode latency is the buffering of input frames for better performance. The encoder allocates additional source frame buffers to the minimum, specified in the above Table. This is so that libxcoder can be downloading the next frame, while the encoder is encoding the previous frame. While this does increase performance, it adds to the latency. Therefore, we have defined a special low latency mode for the encoder. libxcoder will only send a single frame to the encoder at a time, and does not send the next frame until it receives an encoded frame. While this does reduce performance somewhat, it does ensure the lowest possible encoding latency.

The actual latency is primarily determined by the length of time to encode each frame and the number of streams being encoded. This is determined mainly by the picture size and the load on Quadra.

Quadra has 4 encoder cores, enabling encoding with 4 instances, without any increase in latency, 1 instance per encoder core. For example, if Quadra can encode 32 1080p30 streams in real time, then it takes roughly 1s/30/32≈1msec to encode a single frame. The latency will increase linearly with each group of 4 encoding instances. For example the latency for instances 1-4 would be the same, the latency for instances 5-8 would be double, etc. The encoder low latency mode is enabled with encode parameter lowDelay=1.
For example, here is a H.264 to H.265 transcode with low latency mode enabled in the encoder and a low delay GOP (gopPresetIdx=3):

```
ffmpeg -vsync 0 -c:v h264_ni_quadra_dec -xcoder-params “out=hw” -
i input.264 -c:v h265_ni_quadra_enc -xcoder-params
"gopPresetIdx=3:lowDelay=1:RcEnable=1:bitrate=4000000"
output.265
```

If we set **lowDelay=0**, or omit it from the command above, the encoder will encode using normal delay mode, which is the default.

Note that low delay mode requires a low delay GOP and if enabled with a high delay GOP, an error message will be returned.

Normally, FFmpeg polls the encoder at a 200us interval when a new YUV frame is sent to the encoder, it polls until a frame is available to read. The frame is then sent to the encoder at the interval of input FPS. For example, for 50Hz input, the interval is 20ms. This way a frame sent to the encoder for encoding, will only be available after 20ms. So the latency is longer than the interval of frames. The encoder low latency mode changes this behavior. If enabled, it polls the encoder only once, and this poll request is blocking, until a frame is available to read.

Note that the latency of the first frame can be a little higher than other frames due to additional time for buffer allocation. Also note that the latency can vary from frame to frame slightly based on the complexity of the frame.

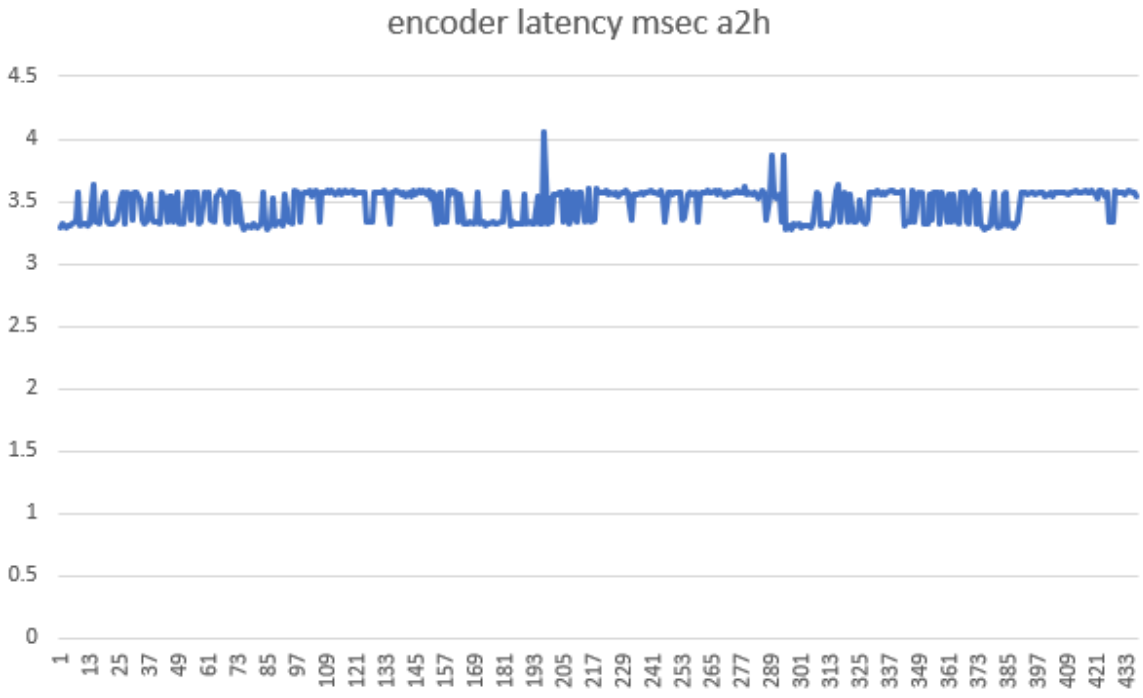
The graph below shows the measured latency for encoding a single 1080p H.265 bitstream with low delay mode enabled. The source stream is in H.264, thus the abbreviation a2h, which represents **A**VC to **H**EVC transcoding.

The command used

```
ffmpeg -y -nostdin -hide_banner -f concat -c:v h264_ni_quadra_dec
-xcoder-params out=hw -i list.txt -c:v h265_ni_quadra_enc -
xcoder-params
gopPresetIdx=3:lowDelay=1:RcEnable=1:bitrate=4000000 -f null -
2>&1 |ts '%.s''
```

This Guide describes the encoder **lowDelay** parameter as follows:

> Specifies whether or not to enable the low latency mode in encoding. When enabled, libxcoder only permits buffering of a single frame to minimize the delay.
>
> Note that when enabled, the gopPresetIdx must be 1, 3, 7, 9, 10, or 0 with a consecutive order GOP pattern, lookaheadDepth must be 0, and multicoreJointMode must be 0.
>
> Note that in libxcoder encoder send/receive multi-thread mode, when enabled, its value can be a positive integer value in milliseconds for threads synchronization. It represents the time the sending thread waits before deciding it's in a deadlock and has to continue without waiting for receiving thread to signal.



13.1.4 Decoder

Video decode latency is defined as the delay between a video frame input to the decoder, and the same frame output from the decoder.

13.1.5 GOP Requirements to Minimize Decoder Latency

To minimize decode latency, a low delay GOP stream must be used in the input bitstream. A low delay GOP has all the frames encoded in sequence. See the above table for examples of such a GOP that can be generated by the Quadra encoder.

13.1.6 Decode Low Latency Mode

A special decoder low latency mode has been defined such that libxcoder only sends a single frame to the decoder at a time for decoding. The libxcoder does not send the next frame until it receives the decoded frame. While this reduces performance somewhat, it ensures the lowest decoding latency possible.

Below is a command used for decoding a stream encoded in sequence:

```
ffmpeg -vsync 0 -c:v h264_ni_quadra_dec -xcoder-params lowDelay=1
-i input.264 -c:v h265_ni_quadra_enc output.265
```

This Guide describes the decoder **lowDelay** parameter as follows:

> Specifies whether to enable the low latency mode in decoding. When enabled, libxcoder uses a different query method that returns upon frame ready to reduce polling. This method only permits buffering of a single frame to minimize delay and therefore will not work with non-sequentially decoded inputs. User must be aware to enable lowDelay decode only on streams whose frames are in sequence. If improper input is provided, the frames will be decoded and returned out of order.
>
> Note that in libxcoder decoder send/receive multi-thread mode, when enabled, its value can be a positive integer value in milliseconds for threads synchronization. It represents the time the sending thread waits before deciding it's in a deadlock and has to continue without waiting for receiving thread to signal.

13.1.7 Summary for Minimizing Latency

To minimize decoder/encoder latency do the following:

1. Use low delay GOP with in sequence frames for decoding/encoding

2. For H.264 encode, using libxcoder parameter **useLowDelayPocType=1** ensures that the encoded bitstream can be decoded without delay.

3. Enable low delay mode in the Netint decoder or encoder

13.2 Measuring Latency

Libxcoder can be compiled to generate latency information for both the decoder and encoder. The information can be output to the FFmpeg log. This can then be used to plot latency as shown in the graph above.

Note the following section describes the encoder latency measurement only, the decoder latency measurement can be created using a similar method.

The libXcoder Latency reporting mechanism reports per frame latency for the frame's duration through the Quadra hardware. It prints logs to stderr.

13.2.1 Compiling the Latency Reporting Mechanism

To enable latency reporting, compile the **libXcoder** with the '**--with-latency-display**' flag**:**

```
cd libxcoder
./build.sh -p
```

Next compile FFmpeg, libAVcodec, or any custom application that is integrated with Quadra.

See the "**Build FFmpeg with NETINT Codec Library**" section in the **QuickStartGuideQuadra_*.pdf** for more instructions on the installation of the FFmpeg application.

13.2.2 Running FFmpeg with low-delay mode encoder

To provide the best throughput of frames, libXcoder's default operation mode allows frames to be buffered in the decoder/encoder input queue on the Quadra device itself. If the lowest latency is desired the **lowDelay** xcoder parameter can be used. This will cause libXcoder to attempt to retrieve a processed frame before sending the next frame.

An example FFmpeg command with the **lowDelay** xcoder-param:

```
ffmpeg -y -nostdin -hide_banner -vsync 0 -stream_loop 1 -f
rawvideo -pix_fmt yuv420p -s:v 1920x1080 -r 25 -i
/mnt/ramdisk/dinner_1920x1080p30_300_br7500_b3__B265d.yuv -c:v
h265_ni_quadra_enc -enc 0 -xcoder-params
gopPresetIdx=3:lowDelay=1:RcEnable=1:bitrate=4000000 -f null -
2>&1
```

13.2.3 Latency Logs

When decoding or transcoding through Quadra, latency report messages will appear in the stderr of the terminal thus:

```
1642192725.181572 DTS:323,DELTA:8008256,eLAT:6166089;
1642192725.181599 DTS:324,DELTA:7805161,eLAT:5912997;
1642192725.181626 DTS:325,DELTA:7884774,eLAT:6053380;
1642192725.181652 DTS:326,DELTA:7990645,eLAT:6149894;
1642192725.181679 DTS:327,DELTA:7730706,eLAT:5903105;
1642192725.181705 DTS:328,DELTA:7895577,eLAT:6055641;
1642192725.181732 DTS:329,DELTA:7685357,eLAT:5832137;
1642192725.181758 DTS:330,DELTA:7506154,eLAT:5675062;
1642192725.181786 DTS:331,DELTA:7538778,eLAT:5747147;
1642192725.181812 DTS:332,DELTA:7834258,eLAT:6022893;
```

They describe:

DTS	Decoding time stamp of frame in the timebase transmitted to libXcoder
DELTA	Frame period between this frame and previous frame in nanoseconds
dLAT	HW decoder latency for frame of this DTS in nanoseconds
eLAT	HW encoder latency for frame of this DTS in nanoseconds

13.2.4 Interpreting Latency Results

The measurement mechanism exists at the lowest level of libXcoder, immediately before any frames/packets are sent to, or received from, Quadra Hardware. The latency measured does not account for any time spent within the FFmpeg, libAVcodec, or other parts of the libXcoder call stack. Though this time is typically very low for functions using the APIs.

As the measurement mechanism is in libXcoder it is also dependent on higher level APIs calling the function to retrieve frame from the decoder/encoder output buffer. The processing rate control mechanisms such as FFmpeg's "-re" can cause the minimum measured latency to be higher than the frame period.

Also, be aware that the hardware sided buffering mechanisms can also cause latency. If a plot of the measured latency appears to show any linear increase in latency from the beginning of a stream, then the decoder/encoder buffer is likely working. These buffers allow frames to be sent to the hardware before they are processed and sent back to the libXcoder. The "**lowDelay**" encoder parameter can be used to restrict this buffering behavior on the encoder. The "**-re**" ffmpeg mechanism to limit the processing frame rate can be used to regulate the frame send-rate to the decoder, in order to minimize hardware buffering.

13.2.5 Encoder Latency Measurement

13.2.6 Scope

This section will demonstrate the collection of latency measurement data for Quadra.
The Quadra and FFmpeg environment configuration and usage is not in the scope of this section.
The document assumes a Linux host installed with a Quadra card and with libxcoder and FFmpeg successfully compiling and Quadra demonstrated for video transcoding operation.

13.2.7 Using libxcoder latency logs

This method includes enabling print outs for encoder latency time and then parsing the log data to determine the latency measurement.

13.2.8 Build libxcoder with-p flag

Run the libxcoder build with the -p flag for latency patch.

```
sh build.sh -p
sudo make install
```

Build FFmpeg-n4.2.1, for example, as usual.

13.2.9 Collect eLAT data

Run the following ffmpeg encode command with a yuv file input. It will generate an output log file.

```
ffmpeg -y -nostdin -hide_banner -vsync 0 -stream_loop 1 -f
rawvideo -pix_fmt yuv420p -s:v 1920x1080 -r 25 -i
/mnt/ramdisk/dinner_1920x1080p30_300_br7500_b3__B265d.yuv -c:v
h265_ni_quadra_enc -enc 0 -xcoder-params
gopPresetIdx=3:lowDelay=1:RcEnable=1:bitrate=4000000 -f null -
2>&1 |ts '%.s' > output-y2h-0-0-0.log
```

The output log file will have extra log messages as follows.

```
1642193165.553249 DTS:1,DELTA:6965127,eLAT:5511326;
1642193165.553276 DTS:2,DELTA:8132467,eLAT:5629519;
1642193165.553303 DTS:3,DELTA:9577126,eLAT:5670137;
1642193165.553330 DTS:4,DELTA:9994202,eLAT:5658198;
1642193165.553357 DTS:5,DELTA:7654552,eLAT:5642386;
1642193165.553384 DTS:6,DELTA:10671531,eLAT:5939394;
1642193165.553411 DTS:7,DELTA:10420093,eLAT:6033683;
```

where,

DTS Decoding time stamp of frame in the timebase transmitted to libXcoder
DELTA Frame period between this frame and previous frame in nanoseconds
eLAT HW encoder latency for frame of this DTS in nanoseconds

13.2.10 Measure Latency

Parse the output log file to collect the eLAT time stamps.

```
cat output-y2h-0-0-0.log | sed 's/;/ /g' | grep "eLAT:" | cut -
d"," -f3 | sed 's/eLAT:/ /g'> encoder-out_time-y2h-0-0-0.log
```

Collate the parsed eLAT data to a csv file.

The parsed data will be as sample below:

5511326
5629519
5670137
5658198
5642386
5939394
6033683

Calculate the eLAT / 1000000 to get the latency msec.

13.2.11 Using ffmpeg-debug_ts

This method can be used for collecting latency measurements for T408 and any 3rd party GPU like Nvidia or AMD.
Please note that the method measures latency approximately and it is suggested not to use it on Quadra because a more accurate method was provided in the previous sections.

1. Run ffmpeg command to perform encoding operation. Use -c:v parameter to specify video encoder to use like h264_ni_quadra_enc or h265_ni_quadra_enc for NETINT encoder.

T408 example:

```
ffmpeg -y -nostdin -hide_banner -vsync 0 -debug_ts -stream_loop 1
-f rawvideo -pix_fmt yuv420p -s:v 1920x1080 -r 25 -i
/mnt/ramdisk/dinner_1920x1080p30_300_br7500_b3__B265d.yuv -c:v
h265_ni_quadra_enc -enc 0 -xcoder-params
gopPresetIdx=3:lowDelay=1:RcEnable=1:bitrate=4000000 -f null -
2>&1 |ts '%.s' > output-y2h-0-0-0.log
```

The output log file, for T408 as example, will have extra log messages as follows.

```
1642199011.985473 muxer <- type:video pkt_pts:1 pkt_pts_time:0.04
pkt_dts:-6 pkt_dts_time:-0.24 size:964
1642199011.987290 demuxer -> ist_index:0 type:video
next_dts:80000 next_dts_time:0.08 next_pts:80000
next_pts_time:0.08 pkt_pts:2 pkt_pts_time:0.08 pkt_dts:2
pkt_dts_time:0.08 off:0 off_time:0
1642199011.987346 demuxer+ffmpeg -> ist_index:0 type:video
pkt_pts:2 pkt_pts_time:0.08 pkt_dts:2 pkt_dts_time:0.08 off:0
off_time:0
1642199011.987362 decoder -> ist_index:0 type:video frame_pts:2
frame_pts_time:0.08 best_effort_ts:2 best_effort_ts_time:0.08
keyframe:1 frame_type:1 time_base:1/25
1642199011.987376 filter -> pts:2 pts_time:0.08 exact:2.000008
time_base:1/25
1642199011.987390 encoder <- type:video frame_pts:2
frame_pts_time:0.08 time_base:1/25
1642199011.993528 encoder -> type:video pkt_pts:2
pkt_pts_time:0.08 pkt_dts:-5 pkt_dts_time:-0.2
1642199011.993600     Last message repeated 1 times
```

2] Filter and record latency data from output log

```
cat output-y2h-0-0-0.log | grep "encoder <-" | cut -d" " -f1 >
encoder-in_time-0-0.log && cat output-$tag.log | grep "encoder -
>" | cut -d" " -f1 > encoder-out_time-0-0-0.log
```

3] Collate the encoder in time and encoder out time stamps to a csv file.

Sample encoder in times:
1642199011.987390
1642199011.997031
1642199012.007143
1642199012.015154
1642199012.025613
1642199012.035669
1642199012.043303
1642199012.053758
Sample encoder out times:
1642199011.993528
1642199012.003550
1642199012.013914
1642199012.021652
1642199012.032136
1642199012.042110
1642199012.049788
1642199012.060334

4] Calculate the output log encoder out and encoder in time stamps diff * 1000 to get the latency msec.

14. GStreamer NETINT Plugins

In the Quadra release V4.7 and onwards, NETINT provide one version of GStreamer 1.22 support for Quadra transcode and filters.

14.1 Gstreamer-1.22.2 & Gstreamer-1.24.4

Based on the gstreamer-1.22.2 and gstreamer-1.24.4, the plugins directly use the NETINT libxcoder interface.

14.1.1 Decoding

The list of gstreamer NETINT command options for decoding can be shown with this command:

```
gst-inspect-1.0 --plugin niquadra | grep decoder
```

The result of <decoder name> is as following:

- ***niquadrah264dec: NetInt NIQUADRA H264 decoder***
- ***niquadrah265dec: NetInt NIQUADRA H265 decoder***
- ***niquadrajpegdec: NetInt NIQUADRA JPEG decoder***
- ***niquadravp9dec: NetInt NIQUADRA VP9 decoder***

Get more details by following command:

```
gst-inspect-1.0 <decoder name>
```

Example:

```
gst-inspect-1.0 niquadrah264dec
```

14.1.2 Encoding

The list of gstreamer NETINT command options for decoding can be shown with this command:

```
gst-inspect-1.0 --plugin niquadra | grep encoder
```

The result of <encoder name> is as following:

- ***niquadraav1enc: NetInt QUADRA AV1 encoder***
- ***niquadrah264enc: NetInt QUADRA H264 encoder***
- ***niquadrah265enc: NetInt QUADRA H265 encoder***
- ***niquadrajpegenc: NetInt QUADRA JPEG encoder***

Get more details by following command:

```
gst-inspect-1.0 <encoder name>
```

Example:

```
gst-inspect-1.0 niquadrah264enc
```

14.1.3 Filters

The list of gstreamer NETINT command options for filters is as following:

- *niquadrabgr: NIBGR Netint element*
- *niquadracrop: NICROP Netint element*
- *niquadradrawbox: NIDRAWBOX Netint element*
- *niquadrahwdownload: HWDownload Netint element*
- *niquadrahwupload: HWUpload Netint element*
- *niquadraoverlay: Overlay NETINT element*
- *niquadrapad: NIPAD Netint element*
- *niquadraroi: NIROI Netint element*
- *niquadrarotate: NIROTATE Netint element*
- *niquadrascale: NISCALE Netint element*
- *niquadrastack: Stack NETINT element*

14.1.4 Known issues.

The following is a complete list of all known issues.

1. Gstreamer with NETINT encoding produces different results from FFmpeg
The FFmpeg encoder does not contain the aspect ratio in VUI by default. Gstreamer will add an aspect ratio in the VUI with the default value ”1/1”. See the below pipeline for an example. Change the default value by using the pixel-aspect-ratio property.

$ gst-launch-1.0 filesrc location=/opt/work/FFmpegXcoder/libxcoder/test/akiyo_352x288p25.yuv ! videoparse width=352 height=288 format=i420 framerate=25/1 pixel-aspect-ratio=1/2 ! niquadrah265enc ! filesink location=~/aki-gstreamer-enc-ni.265

The FFmpeg encoder does not contain any color description in the VUI by default. If it is not explicitly specified choose a default colorimetry for GStreamer. The default colorimetry will be different for different resolutions. The test video is 355x288 which is SD, so the first BT601 is chosen. See the pipeline example below, this overwrites the default value by adding “video/x-raw,colorimetry=bt709”

$ gst-launch-1.0 filesrc location=/opt/work/FFmpegXcoder/libxcoder/test/akiyo_352x288p25.yuv ! videoparse width=352 height=288 format=i420 framerate=25/1 ! video/x-raw,colorimetry=bt709 ! niquadrah265enc ! filesink location=~/aki-gstreamer-enc-ni.265

2. Gstreamer will not support wmv containers with h264 format
3.Add limit to the processing speed of Gstreamer for 8k 10bit input video
Due to the limited memory pool size on the firmware, the processing speed of the filters with multiple HW frame inputs like overlay/xstack is generally controlled, especially for high resolution like 8k-10bit input video. For example we can set the property “max-size-buffers” and “max-size-bytes” for queue element to control the processing speed for the following test command:

$ gst-launch-1.0 filesrc location=/home/test/Shanghai_7680x4320p30_450_10bit.h264 ! h264parse ! niquadrah264dec xcoder-params='out=hw' dec=0 ! niquadraoverlay x=0 y=0 name=overlay ! tee name=t ! queue max-size-buffers=1 max-size-bytes=663552000 ! niquadrah264enc enc=0 ! h264parse ! filesink async=false location=/home/test/test1.h264 t. ! queue max-size-buffers=1 max-size-bytes=663552000 ! niquadrascale width=959 height=539 ! niquadrah264enc enc=0 ! h264parse ! filesink async=false location=/home/test/test2.h264 t. ! queue max-size-buffers=1 max-size-bytes=663552000 ! niquadrascale width=480 height=270 ! niquadrah264enc enc=0 ! h264parse ! filesink async=false location=/home/test/test3.h264 filesrc location=/home/test/Crowdrun_3840x2160p30_300.h264 ! h264parse ! niquadrah264dec xcoder-params='out=hw' dec=0 ! niquadrascale width=960 height=540 ! overlay.

If the limited isn’t added, this test case will report “insufficient resource” issue because of insufficient memory.

14.1.5 Supported Features of Gstreamer

Gstreamer is supported on Linux, Windows, Android and MacOS with version 1.22.x. This version has been fully validated for Linux.

The following table lists all features supported on gstreamer.

Feature		1.22
Decoder	H264	√
	H265	√
	VP9	√
	JPEG	√
Encoder	H264	√
	H265	√
	AV1	√
	JPEG	√
Decoder Xcoder	All xcoder parameters	× (multi-output is not supported)
Encoder Xcoder	dolbyVisionProfile	√
	Others xcoder Parameters	√
Filters	ni_quadra_scale	√
	ni_quadra_overlay	√
	ni_quadra_split	× (No need for gstreamer)
	ni_quadra_crop	√
	ni_quadra_pad	√
	ni_quadra_hwupload	√
	ni_quadra_roi	√
	ni_quadra_bg	√
	ni_quadra_xstack	√
	ni_quadra_rotate	√
	ni_quadra_drawbox	√
	ni_quadra_drawtext	×
	ni_quadra_ai_pre	×
	ni_quadra_delogo	×
	ni_quadra_merge	×
	ni_quadra_flip	×
Advanced Feature	HDR	
	ROI	
	Closed Captions	√
	Rate Control	√

	User Data Unregistered SEI Passthrough	√
	IDR Frame Forcing	×
Others	AV1 tile	×
	HEVC tile	×

15. Resource Management

A resource management mechanism is in place on the NETINT server for the management of video transcoding resources. It provides query/allocation of transcoding resources in the form of utility programs. A C language library and API are ready to integrate with third party application software packages. such as FFmpeg.

15.1 Transcoding Resources

The transcoding resources on a host are hardware transcoder cards and decoder/encoder chips inside those cards. Each decoder/encoder has a certain processing capacity that can handle a limited number of video streams based on resolution and frame rate. The resource management's tasks are to present inventory and status on available resources and enable resource distribution. User applications can build their own resource management schemes on top of this resource pool or leave this task to the NETINT server for some default simplified resource distribution scheme.

15.2 Device Load and Software Transcoding Instance

At system run time, device firmware maintains a value for each hardware codec representing the processing load currently on the codec. This number is obtained by accumulating clock cycles spent decoding, encoding, and filtering streams and dividing it by the maximum number of cycles available during a period of time. This reflects how heavy the codec is being used for the stream processing. This is called the real load.

For each stream being decoded or encoded, a software decoding or encoding instance is created on the hardware codec. The number of active software transcoding instances on a hardware instance is another measure of load on transcoding resources.

The firmware also tracks the model load for each card. The model load is calculated by width*height*FPS. The model load will be increased from the firmware side once an instance is created successfully, either for encoder or decoder. When an instance is closed successfully the model load will be deducted as well.

15.3 Resource Distribution Strategy

Users may query at run time the load numbers and create resources on specific devices by specifying the device ID directly for encoders and decoders as well as for ni_quadra_hwupload and ni_quadra_roi. Other Netint filters which use hardware frames are always collocated with their input.

See also APPS548 Codensity Quadra YUVbypass Application Note to learn more about hardware frames.

If the device ID is not specified, then decoders and encoders will be created on the least loaded device, either as modeled (by default), or by real load. If the input to the encoder is a hardware frame it will be collocated with its input.

15.3.1 Examples

The following example allocates the H.264 decoder to the least model load device by default.

```
ffmpeg -c:v h264_ni_quadra_dec -i input.264 output.yuv
```

The following example allocates the decoder to device 0 and the encoder to device 1. This is not optimal since the decoder YUV frames must be transferred from device 0 to the host and then back to device 1. Note that the use of hardware frames does not help when the encoder and decoder are on different devices.

```
ffmpeg -c:v h264_ni_quadra_dec -dec 0 -i input.264 -c:v h265_ni_quadra_enc -enc 1
output.265
```

15.4 NETINT Command-Line Interface (CLI)

A few utility programs are provided to list and monitor resource usage. Running the utility /usr/local/bin/ni_rsrc_list produces results showing capabilities of cards on the host. Another utility is /usr/local/bin/ni_rsrc_mon, that actively monitors the resource usage on the server and initializes resources.

15.4.1 ni_rsrc_mon

A sample output is shown below:

```
ni_rsrc_mon
************************************************
1 devices retrieved from current pool at start up
Fri Jan  3 11:58:40 2025 up 00:00:00 v---6rxDV
INDEX    TEMP     POWER    FR       SN
0        40       6922mW   ---6rxDV Q1A10BA11FC060-0044
Num decoders: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE
0     0    0          0    0    0         0       /dev/nvme1n1
Num encoders: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE
0     0    0          0    0    0         0       /dev/nvme1n1
Num scalers: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE
0     0    0          0    0    0         0       /dev/nvme1n1
Num AIs: 1
INDEX LOAD MODEL_LOAD INST MEM  SHARE_MEM P2P_MEM DEVICE
0     0    0          0    0    0         0       /dev/nvme1n1
************************************************
```

The number of devices having encoders, decoders, scalers (2D Engines), and AI engines are listed.

The output format also shows the real-time temperature, power, current Firmware revision and Serial number.

To see the number of uploaders and firmware/system load by subsystem, try the full output format.

The simple output format shows the maximum firmware/system load amongst the subsystems on the Quadra device.

A help text with description of how to use the program can be accessed with the command:

-n	Specify reporting interval in one second interval. If 0 or no selection, report only once. Default: 0
-R	Specify if refresh devices on host in each monitor interval. Default: 1
-o	Output format. [text, simple, full, json, json1, json2, extra]. Default: text
-D	Dump firmware logs to current directory. Default: 0(not dump fw log).
-k	Specify to dump which card's firmware logs. Default: -1(dump fw log of all cards).
-r	Initialize Quadra device regardless firmware release version to libxcoder version compatibility. Default: only init cards with compatible firmware version.
-t	Set timeout time in seconds for device polling. Program will exit with failure if timeout is reached without finding at least one device. If 0 or no selection, poll indefinitely until a Quadra device is found. Default: 0
-S	Skip init_rsrc.
-d	Print detailed infomation for decoder/encoder in text and json formats.
-l	Set loglevel of libxcoder API. [none, fatal, error, info, debug, trace] Default: info
-h	Open this help message.
-v	Print version info.

Interpreting load values

`ni_rsrc_mon` utilizes 3 card load mechanisms when reporting:

1. Current load - the most recent VPU load in 1 second. Reported under LOAD/VPU
2. FW model load - FW calculates "estimated" VPU load based on the number of instances, resolutions, encoding parameters, etc. Reported under MODEL_LOAD
3. FW load - the most recent CPU load in 1 second. FW calculates CPU load by (busy cycle count / total cycle count). Reported under FW_LOAD/FW
    - ni_rsrc_mon reported LOAD is the higher value between 1. current load and 2. fw load to reflect the factor that is likely to become the bottleneck on the card, e.g.
        - when running 540p multiple instances encoding, CPU load is usually higher, so ni_rsrc_mon will report FW load as the LOAD, which may be the reason why we are seeing different LOAD value reported by ni_rsrc_mon.
        - For reference, ni_rsrc_mon printout can be found in ni_rsrc_mon.c print_text function, which prints LOAD value with the following condition -
          `sessionCtxt->load_query.current_load > sessionCtxt->load_query.fw_load) ? sessionCtxt->load_query.current_load : sessionCtxt->load_query.fw_load`

To print all the stats (including FW load) via ni_rsrc_mon specify `-o` option with `json, json1 or full`

Reporting columns for text output format

INDEX	index number used by resource manager to identify the resource
LOAD	realtime load given in percentage. This value is max of VPU and FW load reported in full output format
MODEL_LOAD	estimated load given in percentage based on framerate and resolution
INST	number of job instances
MEM	usage of memory given in percentage by the subsystem
SHARE_MEM	usage of memory shared across subsystems on the same device
P2P_MEM	usage of memory by P2P
DEVICE	path to NVMe block device file handle

Additional reporting columns for full output format

VPU	same as LOAD in JSON outputs
FW	system load
TOTAL	same as MEM
CRITICAL	usage of memory considered critical
L_FL2V	last ran firmware loader 2 version
N_FL2V	nor flash firmware loader 2 version
FR	current firmware revision
N_FR	nor flash firmware revision

The INST column also shows the maximum supported number of job instances in the form of INST/MAX_INST.

Additional reporting columns for full JSON formats

LOAD	VPU load
FW_LOAD	system load

Additional reporting columns for JSON2 formats

TEMP	realtime temperature given in degrees Celsius
POWER	realtime power(mw), print N/A when query power not supported

Reporting columns for extra output format

TEMP	realtime temperature given in degrees Celsius
POWER	realtime power(mw), print N/A when query power not supported
DEVICE	path to NVMe block device file handle

Reporting columns for detail output

Use ni_rsrc_mon -d

For Decoder

INDEX	index number used by resource manager to identify the resource
AvgCost	estimated load given in percentage based on framerate and resolution
FrameRate	framerate of current session
IDR	number of IDR frame it processes
InFrame	number of frames send to the decoder
OutFrame	number of frames get from the decoder
Width	width of decoder
Height	height of decoder
SID	session id
DEVICE	path to NVMe block device file handle
NAMESPACE	path to namespace block device file handle

For Encoder

INDEX	index number used by resource manager to identify the resource
AvgCost	estimated load given in percentage based on framerate and resolution
FrameRate	framerate of current session
IDR	number of IDR frame it processes
UserIDR	number of force IDR frame set by user
InFrame	number of frames send to the decoder
OutFrame	number of frames get from the decoder
BR	instantaneous bitrate of current encoder session
AvgBR	average bitrate of current encoder session
Width	width of decoder
Height	height of decoder
Format	input frame format send to encoder
SID	session id
DEVICE	path to NVMe block device file handle
NAMESPACE	path to namespace block device file handle

15.4.2 ni_rsrc_list

This tool is to display information for NETINT hardware.

A help text with description of how to use the program can be accessed with the command:

-a Print includes info for uninitialized cards.

-h Display this help and exit.

-v Print version info.

-o Output format. [text, full, json]

Default: text

-l Set loglevel of libxcoder API.

[none, fatal, error, info, debug, trace]

Default: info

15.5 NVMe SMART Log

To obtain the Quadra device NVMe SMART log, use the following command:

```
sudo nvme smart-log /dev/nvme0
```

Here is an example of the output:

```
Smart Log for NVME device:nvme0 namespace-id:ffffffff
critical_warning                    : 0
temperature                         : 38 C
available_spare                     : 100%
available_spare_threshold           : 20%
percentage_used                     : 0%
data_units_read                     : 0
data_units_written                  : 0
host_read_commands                  : 0
host_write_commands                 : 0
controller_busy_time                : 0
power_cycles                        : 0
power_on_hours                      : 0
unsafe_shutdowns                    : 0
media_errors                        : 0
num_err_log_entries                 : 0
Warning Temperature Time            : 0
Critical Composite Temperature Time : 0
Temperature Sensor 1                : 38 C
Temperature Sensor 2                : 41 C
Thermal Management T1 Trans Count   : 0
Thermal Management T2 Trans Count   : 0
Thermal Management T1 Total Time    : 0
Thermal Management T2 Total Time    : 0
```

Below is an explanation of the sensors :

Temperature Sensor 1 represents the board temperature.
Temperature Sensor 2 represents the on die temperature.

temperature represents the composite temperature.

When determining throttling behavior, the composite temperature is used.

15.6 Device Temperature

Quadra uses temperature sensors to protect the device from excessive temperature.

15.6.1 Warning Temperature and Throttling

Throttling will be activated once the device composite temperature reaches or exceeds 70°C. Throttling will be deactivated once the composite temperature drops again below 70°C.

Device performance will be reduced during throttling. The following component clock speeds will be reduced during throttling

- CPU
- Encoder Engine
- Decoder Engine
- 2D Engine/Scaler

Note that ni_rsrc_mon LOAD values may appear doubled or halved for a transient period after switching between throttling and non-throttling states due to pre-throttle/unthrottle load stats being measured against current clock.

15.6.2 Critical Temperature and Device Reset

To protect the Quadra device, if the composite temperature of 80°C or more is detected, the device will reset. This will result in a loss of all current workloads.

15.7 Resource Pool Management

A number of files with NI prefix names are created (e.g. in folder /dev/shm on Linux) during resource initialization running either *init_rsrc* or *ni_rsrc_mon* for the very first time after system reboot. They are used by libxcoder and the *ni_rsrc_mon* utility to identify all Quadra cards in the system.

The host resource pool management starts with utility program (*init_rsrc*, or *ni_rsrc_mon*) scanning the available NETINT transcoder cards on the host. It collects NVMe devices by looking at */dev/nvmeX* device files and issuing NVMe identify requests. It checks the response and identifies NETINT devices by matching with NETINT vendor ID 0x1D82. It further identifies transcoder cards by checking the vendor specific data section for a flag named xcoder_support. T4xx card sets this value to 1, and Quadra card 2. All other NVMe devices with this value set to 0 would be considered a non NETINT transcoder device and ignored.

When the NETINT Quadra cards are identified, their information including character device file name *(/dev/nvmeX*), block device file name *(/dev/nvmeXnY*) and capabilities numbers are retrieved, records and locks (referred to as host resource pool files) are created and saved at */dev/shm* on Linux – a temporary file storage file system that uses RAM for the backing store, and functions as a shared memory for applications to access for transcode resource allocation. On Windows and Android systems the host resource pool files will be generated similarly though the actual file location and mechanism may be different.

A utility program *ni_rsrc_list* can be run to display the information of cards available on that host. An additional command line argument, -a, when specified, displays general information about cards that were not initialized by *init_rsrc*.

A utility program *ni_rsrc_mon* is usually run to print out the status of cards on a host such as load, number of active instance and memory usage. A continued running of this program also keeps updating the host resource pool files in the background, that is cards pulled from or inserted into the system will be detected by this program and host resource records will be updated accordingly.

When an application needs to run a transcoding task, it can either explicitly designate a card to use by its index or its block device name, as presented by *ni_rsrc_list*, or leave it to the resource management to automatically pick a card with the least load to accomplish load balancing on a host. This is done in the transcode session opening.

15.8 Thread Management and Keep Alive

Note that each opened NetInt transcoding session maintains a keep-alive thread. This thread sends a regular heartbeat from the libxcoder int the host application, to the firmware running on the transcoding card. When integrating the libxcoder into any third party frameworks, care should be taken to ensure that the transcoding sessions opening thread (and the keep-alive thread spawn by it), shall have the fairness of thread scheduling (e.g. a round-robin scheduling policy). This will allow the heartbeat to be sent within the specified time interval, thus keeping the channel open. For Linux macos and Android platforms, change the name of this thread to KAT+hw_id+session_ id, such as hw_id=0, session_id=0x3ef, thread name is KAT0003ef.

If a session heartbeat is not been received by the firmware within the dedicated timeframe, then the session will be terminated.

16. Debugging

16.1 NETINT Codec Library Debug Log

The NETINT Codec Library (including libxcoder) provides full logging of event sequences and information. This includes run time timestamps, for troubleshooting and debugging purposes.

When using the NETINT Codec Library, libxcoder uses the same logging level as specified by FFmpeg's command line option "**-loglevel**". Please reference the FFmpeg manual page for details. The default level is "-loglevel none" and the highest level is "-loglevel trace"

If your application imports libxcoder directly, the logging level may be set by importing ni_utils.h and calling the ni_log_set_level() function. Please refer to the code excerpt below from ni_util.h for enumerations and functions relevant to libxcoder logging.

```
typedef enum
{
  NI_LOG_NONE  = 0,
  NI_LOG_FATAL = 1,
  NI_LOG_ERROR = 2,
  NI_LOG_INFO  = 3,
  NI_LOG_DEBUG = 4,
  NI_LOG_TRACE = 5
} ni_log_level_t;

void ni_log_set_level(ni_log_level_t level);
ni_log_level_t ni_log_get_level(void);
ni_log_level_t ff_to_ni_log_level(int fflog_level);
```

17. Deprecated Parameters

This section lists all the deprecated parameters in Quadra's SDK.

NETINT **strongly recommends** that ALL deprecated parameters are no longer used, and that their replacement parameters are used instead. Each deprecated parameter has a replacement parameter or a description of a replacement strategy listed next to it.

Please discuss with your NETINT support representative, or a member of our NETINT FAE team if you require any further guidance on any parameters.

17.1 Backward Compatibility

All Quadra releases from **4.0.0** guarantee backward compatibility.

Backward compatibility means that any application code (or command lines), developed for release 4.0.0 and onwards, will work with any other subsequent releases (for example 4.4.0). Any firmware from 4.0.0 and onwards will work with any software component from 4.0.0 and onwards, and vice versa.

Parameters are deprecated as opposed to being removed to ensure that each release maintains this backward compatibility for all users. Therefore all deprecated parameters will always continue to work, with any future releases. No deprecated parameters ever break backward compatibility.

17.2 List of Deprecated Parameters

Deprecated Parameter Name : MaxFrameSize
Replacement Parameter Name : MaxFrameSize_Bytes
See section : Encoding Parameters

Notes
MaxFrameSize has been deprecated and should not be used.

Instead use the more explicit **maxFrameSize_Bytes** parameter which is equivalent of **maxFrameSize**. Using the new **maxFrameSize_Bytes** parameter will also make the Quadra application code easier to understand.

18. Troubleshooting

The following section lists some ideas for troubleshooting the device or the host side software.

NETINT's support team are always here to provide help and support so please ask if needed, but this list could also help to solve an issue quickly.

18.1 Performance Is Lower than expected

Performance can be affected by the following, please check to make sure everything is working as expected.

1. Make sure the device is running at PCIe Gen 4 speeds
2. Make sure the device has the correct firmware running on it
3. Make sure the software is installed correctly on the host
4. Make sure the device is not throttling. Throttling occurs when the device is too hot. Adequate cooling is needed, and the higher the load on the device the hotter the device will get over time.

   See the section on Throttling for more information.

   Talk to your NETINT representative for information on cooling solutions.

18.2 Compilation Failures

If the host side software, or you application is failing to compile/link, see if any of the listed help can resolve your issue.

18.2.1 FFMpeg Compilation fails with Quadra and CUDA

Some customers have tried to compile the latest CUDA (version 12) with FFMpeg 4.4 and Quadra.

The command for this is expected to be

bash build_ffmpeg.sh --quadra --ffnvcodec

The default for FFmpeg compilation is for it to use the Shader Modules “30” – hence the **compute_30 unsupported GPU architecture message**. Running the above command may result in this failure

```
[root@junk FFmpeg]# bash build_ffmpeg.sh --quadra –ffnvcodec

nvcc -gencode arch=compute_30,code=sm_30 -O2 -m64 -ptx -c -o
/tmp/ffconf.lHNCpvwM/test.o /tmp/ffconf.lHNCpvwM/test.cu

nvcc fatal : Unsupported gpu architecture 'compute_30'

ERROR: failed checking for nvcc.
```

To tell FFmpeg 4.4 to compile for a more recent and supported Shader Model, here for example the NVidia Turing architecture (SM75), set the following custom flags

```
bash build_ffmpeg.sh --quadra --ffnvcodec --custom_flags '--nvccflags=-
gencode=arch=compute_75,code=compute_75'
```

19. Abbreviations

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
CTB	Coding Tree Block
CU	Coding Unit
DMA	Direct Memory Access
DSP	Digital Signal Processing
EOF	End Of File
FPS	Frames Per Second
GOP	Group Of Pictures
HDR	High Dynamic Range
HLG	Hybrid Log Gamma
HRD	Hypothetical Reference Decoder
IE	Inference Engine
Mbps	Mega Bit per second
MB	Macroblock
PPS	Picture Parameter Set
P2P	Peer-to-Peer DMA
QP	Quantization Parameter

RGB	Red Green Blue
RGBA	Red Green Blue Alpha
RDO	Rate Distortion Optimization
ROI	Region of interest
SEI	Supplemental Enhancement Information
SPS	Sequence Parameter Set
VBR	Variable Bit Rate
VCL	Video Coding Layer
VPS	Video Parameter Set
VUI	Video Usability Information