APPS009-Codensity T4xx ROI Application Note

V.3.0

02/04/2021

Table of Contents

1 Table of Abbreviations

Abbreviation:	Full Form:
ROI	Region of Interest
MB	Macro Block
QP	Quantization Parameter
CU	Coding Unit
CTU	Coding Tree Unit
AI	Artificial Intelligence
API	Application Programming Interface
AUD	Access Unit Delimiters
AV	Audio Video
AV1	Alliance Open Media Video 1 Codec
CABAC	Context Adaptive Binary Arithmetic Coding
CBR	Constant Bit Rate
CPU	Central Processing Unit
CRF	Constant Rate Factor
DSP	Digital Signal Processing
EOF	End Of File
FIXQB	Fixed Quantization Parameter
FF	Form Factor
FPS	Frames Per Second
GOP	Group of pictures
HDR	High Dynamic Range
HHHL	Half Height, Half Length
HHFL	Half Height, Full Length
HLG	Hybrid Log Gamma
HRD	Hypothetical Reference Decoder
IE	Inference Engine
Mbps	Mega Bit per second
MBps	Mega Byte per second
NLP	Natural Language Processing
NPU	Neural Processing Unit
NVMe	Non-Volatile Memory express
PCIe	Peripheral Component Interconnect express
PPS	Picture Parameter Set
QP	Quoted Printable
RGB	Red Green Blue
RGBA	Red Green Blue Alpha

RDO	Optimization
ROI	Region of interest
SDK	Software Development Kit
SEI	Supplemental Enhancement Information
SRIOV	Single Root I/O Virtualization
SPS	Sequence Parameter Set
VBR	Variable Bit Rat
VCL	Video Coding Layer
VPS	Video Parameter Set
VUI	Video Usability Information

2 References

Technical Notes: T408 & T432 Integration & Programming Guide

Specifications:

3 Background

This document describes Region of Interest (ROI) feature and its usage in operation/integrating with the NETINT T4XX transcoder.

3.1 Intended Audience

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

3.2 Compatibility

Software Compatibility

This guide is intended to be used with NETINT Codensity T4XX Video Transcoder software Release 1.6.0.

Hardware Compatibility

Release 1.6.0 supports NETINT Codensity T4XX Video Transcoder hardware.

Operation system

All OS supporting T4XX Video Transcoder.

4 ROI

4.1 Intro

ROI stands for Region of Interest and is a feature that permits some regions of the picture to be encoded with better quality than other regions. This makes sure that the compressed bits are being put to the best use possible.

To use the ROI feature on the T4xx, the following steps are required:

1. The feature must be enabled in libxcoder with **roiEnable=1**
2. An ROI QP map specifying QP values and other information need to be supplied with each YUV frame. The ROI values can change and ROI can be enabled/disabled on a frame by frame basis. There are a number of ways to provide this QP map as explained in the following sections.

4.2 FFmpeg ROI API

FFmpeg 4.2.1 and higher support an ROI API by providing an ROI struct definition and a way of passing that data to encoder by using sidedata in AVFrame. FFmpeg defines sidedata for YUV frames in libavutil/frame.h to specify ROI regions:

```
/**
* Regions Of Interest, the data is an array of AVRegionOfInterest
type, the number of
* array element is implied by AVFrameSideData.size /
AVRegionOfInterest.self_size.
*/

typedef struct AVRegionOfInterest {
        uint32_t self_size;
        int top;
        int bottom;
        int left;
        int right;

/**
    * Quantisation offset.
    *
    * Must be in the range -1 to +1.  A value of zero indicates
no quality
    * change.  A negative value asks for better quality (less
quantisation),
```

```
    * while a positive value asks for worse quality (greater
quantisation).
    */
        AVRational qoffset;
 } AVRegionOfInterest;
```

The following code shows an example of applying ROI to a frame: for a 720p frame, the first ROI defines the real region of interest (a window in the middle of image) with highest quality, and the second ROI sets the whole image to lowest quality.

```
AVBufferRef *roi_ref = av_buffer_alloc(sizeof(AVRegionOfInterest) * 2);
AVRegionOfInterest *my_roi = (AVRegionOfInterest*)roi_ref->data;
my_roi[0].self_size = sizeof(AVRegionOfInterest);
my_roi[0].top = 240;
my_roi[0].bottom = 480;
my_roi[0].left = 320;
my_roi[0].right = 960;
my_roi[0].qoffset.num = -1;
my_roi[0].qoffset.den = 1;

my_roi[1].self_size = sizeof(AVRegionOfInterest);
my_roi[1].top = 0;
my_roi[1].bottom = 720;
my_roi[1].left = 0;
my_roi[1].right = 1280;//320;
my_roi[1].qoffset.num = 1;
my_roi[1].qoffset.den = 1;

AVFrameSideData *my_side_data = av_frame_new_side_data_from_buf(frame,
                                                                AV_FRAME_DATA_REGIONS_OF_INTEREST,
                                                                roi_ref);
```

Note that the ROIs are specified in the order of decreasing importance, so the end result is shown in the analyzer as follows:

Note that with the FFmpeg's method of ROI, the ROI map is initialized to the equivalent of qoffset=0. So in the example above, if only my_roi[0] were specified, then the background QP would be the default QP of the encoder. Also the average QP is hardcoded to be 26 which means that when qoffset = -1, we are reducing the QP by 25 and when qoffset = +1, we are increasing the QP by 25.

The formula for how the encoder applies the ROI is as follows:

The QP of an ROI block (16x16 MB for H.264 or 32x32 Sub-CTU for H.265) is:

```
QP = (ROI_QP – ROI_AVG_QP) + QP_BY_RC
```

where:

ROI_QP is the QP specified for the ROI block (1-51) in the ROI map

ROI_AVG_QP is the average of all the QPs of all ROI blocks

QP_BY_RC is the QP determined by the rate control algorithm.

So if ROI_QP is larger than average, the rate control QP is increased (poorer quality); if it's smaller than average, the rate control QP is decreased (better quality).

4.2.1 NetInt *cacheRoi* encoder option

The FFmpeg ROI API is designed to specify the ROI on a single frame and needs to be specified frame by frame. There is an NetInt encoder configuration option *cacheRoi* that can change this behavior. By using this option, user can specify the ROI on a frame and can expect it to stay in effect, i.e. cached and applied to the subsequent frames, until a new one is supplied. Its usage is shown in the following examples.

If we have a video encoding program encode_video that integrates with libavcodec, and uses NetInt encoder nienc to encode video using libxcoder encoder parameters specified in T408 & T432 Integration & Programming Guide, then a command with *cacheRoi* option enabled would be like this:

```
sudo  encode_video -s 352x288 -
i ../libxcoder/test/akiyo_352x288p25.yuv -c h265_ni_enc     -x
"bitrate=3000000:roiEnable=1:cacheRoi=1" -o aki-roiCache.265
```

4.3 FFmpeg ROI Filter

FFmpeg 4.3.1 and higher support an ROI filter called addroi that permits a number of ROI regions to be specified on the command line. Unfortunately, the ROI filter does not currently support being changed on a frame by frame basis and so it is only useful for hardcoded ROI regions or for demo purposes.

The definition of the ffmpeg ROI filter can be found here: https://ffmpeg.org/ffmpeg-filters.html#addroi.

The FFmpeg ROI filter is called addroi and has the following parameters:

- x,y: coordinates of top left corner of ROI region
- w,h: width and height of ROI region
- qoffset: quantization offset of the region from -1 to +1. 0 means no change, negative offset is better quality, positive offset is worse quality
- clear: if true, removes any previously set regions before setting this one

Coordinates and width may be set using variables iw and ih which are input picture dimensions.

For example to specify an ROI region that sets the centre ¼ of the frame as a region of interest with a -0.1 qoffset to improve its quality, you could use the following ROI filter command:

- `addroi=iw/4:ih/4:iw/2:ih/2:-1/10`

Internally, libxcoder maps the FFmpeg qoffset range of -1 to +1 to the QP range of 1 to 51. So QP = (qoffset/2 * 50) +26. So a qoffset= -1 corresponds to a QP=1. Any regions not specified will get qoffset=0 which corresponds to QP=26. If you encode with rate control off, the QP of the center region will be 20 with 26 in the background. If you enable rate control, the QPs will vary but the centre region will still be encoder with a smaller QP.

Note that while FFmpeg permits ROI regions to be specified to pixels accuracy, the Netint encoder supports regions of 16x16 pixels for H.264 and 32x32 for H.265. Libxcoder automatically rounds up the regions to the next largest size. Also note that when multiple regions are specified the earlier regions take priority over later regions if they overlap.

An FFmpeg command for the above example is as follows with rate control disabled so the QP's may be easily observed:

```
sudo  ffmpeg  -vsync 0 -dec 0 -c:v h264_ni_dec -i
~/FFmpegXcoder/libxcoder/test/1280x720p_Basketball.264 -vf
addroi=iw/4:ih/4:iw/2:ih/2:-1 -enc 0 -c:v h264_ni_enc -xcoder-
params "roiEnable=1:RcEnable=0" 720p-roiFilter.264 -y
```

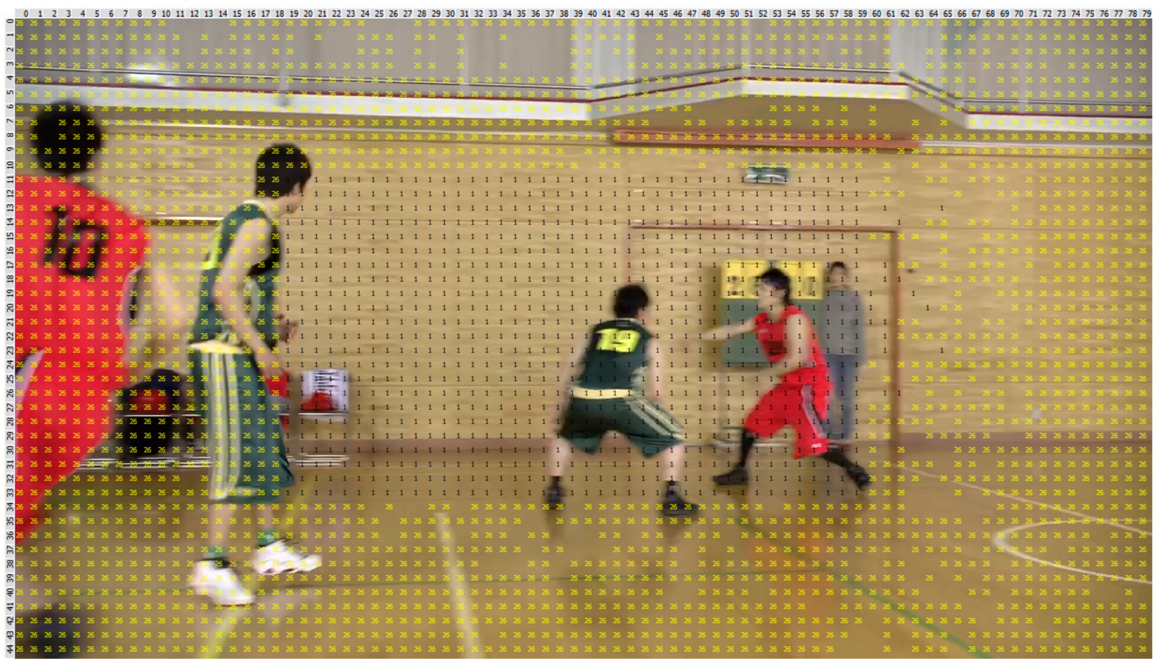
The filter can also be specified in a separate file as follows:

```
sudo  ffmpeg  -vsync 0 -dec 0 -c:v h264_ni_dec -i
~/FFmpegXcoder/libxcoder/test/1280x720p_Basketball.264 -
filter_script roifilter.txt -enc 0 -c:v h264_ni_enc -xcoder-
params "roiEnable=1:RcEnable=0" 720p-roiFilter.264 -y
```

The contents of roifilter.txt in this example would be:

```
addroi=enable=iw/4:ih/4:iw/2:ih/2:-1
```

You can see the QP's in the analyzer output below with QP=1 in the centre region and QP=26 in the surrounding area:



Note that extreme values of ROI such as qoffset=-1 are used as an example only and may overload the rate control and cause the bitrate to be higher than specified. ROI should be used conservatively.

4.4 ROI demo mode

FFmpeg versions earlier than 4.2.1 do not have any built-in support for ROI. Customers wishing to implement ROI must provide the QP map directly to libXcoder themselves. As an example, we support an ROI demo mode in FFmpeg 3.4.2 that provides some hardcoded ROI maps to the encoder.

ROI initialization is done by means of FFmpeg command line options for the T4XX. The resulting command line option looks something like this:

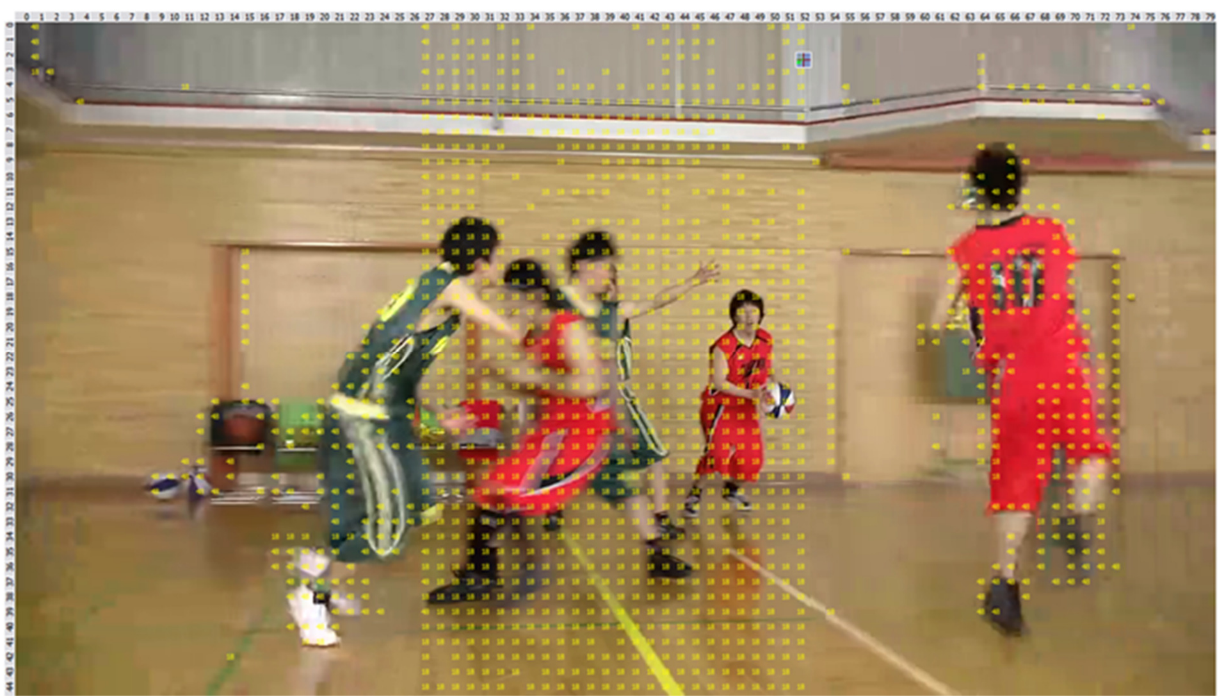
```
sudo ./ffmpeg -c:v h264_ni_dec -i input.264 -c:v h265_ni_enc -
xcoder-params "roiEnable=1: RoiDemoMode=1:RcEnable=1" output.265
```

Note that ROI can be used with or without rate control enabled. When rate control is not enabled the QP values are used as is. When rate control is enabled, the QP values are scaled up and down as necessary to meet the rate control objectives.

There are two demo modes to showcase the usage of ROI without additional application development. Mode 1 sets QP for center 1/3 (vertically) of the picture to a higher value (40) which represents lower quality, and the rest of picture to lower value (10) which is of higher quality. Mode 2 is reverse of mode 1, i.e. the center 1/3 of the picture has the lower importance and the rest the higher. Demo mode takes effect between source video frames number 90 and number 300. Note that this ROI demo mode is supported only in FFmpeg 3.4.2, but the ROI feature is supported in all firmware versions.

This image shows the QPs as selected by the rate control when ROI is disabled:

This image shows the QP when ROI is used with high quality in the centre ½ (QP=10) and low quality on the outer 2/3 (QP=40) of the image.



4.5 Custom map and average QP

The structure and size of the ROI custom map is different for H.264 and H.265. H.264 supports 16x16 pixel ROI regions, while H.265 supports 32x32 pixel sub-CTU regions, 4 to a CTU. Each QP entry is 6 bits in size. There is some other data in the map data structure that we are not using and should be initialized to 0.

The QPs of each block in the custom map should be set directly, and the average of the QPs need to also be calculated.

The QP for each block can vary from 0 to 51 with 0 being the best quality and 51 being the worst. The actual QP numbers themselves are not that important, it is the difference between the numbers that is important. When rate control is enabled, the encoder calculates the average QP for the picture and scales the QPs so that the average is correct.

The QP map is different for H.264 and H.265. The H.264 QP map consists of single byte entries for each 16x16 pixel macroblock (MB) containing the QP (in the top 6 bits), while the H.265 QP map consists of 8 byte entries for each 64x64 coding tree unit (CTU) containing the QP for each of 4 32x32 sub CTU blocks.

The description and sample code given in the following sections can be found in NetInt's libxcoder library source file libxcoder/source/ni_device_api.h, and libavcodec's NetInt encoder source code libavcodec/nienc.c.

4.5.1 H.264

The H.264 custom map entry is defined as follows (1 MB = 8 bits):

```
typedef union _ni_enc_avc_roi_custom_map
{
  struct
  {
    uint8_t  mb_force_mode  :  2; // [ 1: 0]
    uint8_t  mb_qp         :  6; // [ 7: 2]
  } field;
} ni_enc_avc_roi_custom_map_t;
```

Each 16x16 pixel MB has one such entry. There is a single QP for the MB. The number of MBs in each row and column is rounded up. The custom map size is calculated and map memory is allocated as follows:

```
uint32_t mbWidth = (width + 16 - 1) >> 4;

uint32_t mbHeight = (height + 16 - 1) >> 4;
```

```
customMapSize = sizeof(ni_enc_avc_roi_custom_map_t) * mbWidth *
mbHeight;

ni_enc_avc_roi_custom_map_t *avc_roi_map =
(ni_enc_avc_roi_custom_map_t*)calloc(1, customMapSize);
```

Then ROI MBs QPs are copied into custom map, their average is calculated as follows:

```
// copy roi MBs QPs into custom map, calculate average
sumQp = 0;
for (i = 0; i < numMbs; i++)
{
  if ((i % mbWidth > mbWidth/3) && (i % mbWidth < mbWidth*2/3))
  {
     avc_roi_map[i].field.mb_qp = importanceLevelCentre;
  }
  else
  {
     avc_roi_map[i].field.mb_qp = importanceLevelRest;
  }
  sumQp += avc_roi_map[i].field.mb_qp;
}
roi_avg_qp = (sumQp + (numMbs>>1)) / numMbs; // round off
```

4.5.2 H.265

The H.265 custom map entry is defined as follows (1 CTU = 64bits):

```
typedef union _ni_enc_hevc_roi_custom_map
{
  struct
  {
    uint32_t  ctu_force_mode  :  2; //[ 1: 0]
    uint32_t  ctu_coeff_drop  :  1; //[    2]
    uint32_t  reserved       :  5; //[ 7: 3]
    uint32_t  sub_ctu_qp_0    :  6; //[13: 8]
    uint32_t  sub_ctu_qp_1    :  6; //[19:14]
    uint32_t  sub_ctu_qp_2    :  6; //[25:20]
    uint32_t  sub_ctu_qp_3    :  6; //[31:26]

    uint32_t  lambda_sad_0    :  8; //[39:32]
    uint32_t  lambda_sad_1    :  8; //[47:40]
    uint32_t  lambda_sad_2    :  8; //[55:48]
    uint32_t  lambda_sad_3    :  8; //[63:56]
```

```
  } field;
} ni_enc_hevc_roi_custom_map_t;
```

ROI for H.265 is specified for 32x32 pixel subCTU blocks – 4 subCTU QPs are stored in each custom CTU map entry. The approach taken here is to first load the 32x32 subCTU entries into hevc_sub_ctu_roi_buf, then load from there into the final custom map hevc_roi_map, as follows:

```
// number of CTUs in each row and column
ctuWidth = (width + 64 -1) >> 6;
ctuHeight = (height + 64 - 1) >> 6;
// number of sub CTUs in each row and column
subCtuWidth = ctuWidth * 2;
subCtuHeight = ctuHeight * 2;
numSubCtus = subCtuWidth * subCtuHeight;
hevc_sub_ctu_roi_buf = (uint8_t *)malloc(numSubCtus);
for (i = 0; i < numSubCtus; i++)
{
    if ((i % subCtuWidth > subCtuWidth/3) &&
        (i % subCtuWidth < subCtuWidth*2/3))
    {
       hevc_sub_ctu_roi_buf[i] = importanceLevelCentre;
    }
    else
    {
       hevc_sub_ctu_roi_buf[i] = importanceLevelRest;
    }
}
hevc_roi_map = (ni_enc_hevc_roi_custom_map_t *)
        calloc(1, sizeof(ni_enc_hevc_roi_custom_map_t) * ctuWidth
* ctuHeight);
sumQp = 0;
for (i = 0; i < ctuHeight; i++)
{
   uint8_t *ptr = &hevc_sub_ctu_roi_buf[subCtuWidth * i * 2];
   for (j = 0; j < ctuWidth; j++, ptr += 2)
   {
      ctu = i * ctuWidth + j;
      hevc_roi_map[ctu].field.sub_ctu_qp_0 = *ptr;
      hevc_roi_map[ctu].field.sub_ctu_qp_1 = *(ptr + 1);
      hevc_roi_map[ctu].field.sub_ctu_qp_2 = *(ptr + subCtuWidth);
      hevc_roi_map[ctu].field.sub_ctu_qp_3 = *(ptr + subCtuWidth +
1);
```

```
      sumQp += (hevc_roi_map[ctu].field.sub_ctu_qp_0 +
hevc_roi_map[ctu].field.sub_ctu_qp_1 +
                          hevc_roi_map[ctu].field.sub_ctu_qp_2 +
hevc_roi_map[ctu].field.sub_ctu_qp_3);
   }
}
roi_avg_qp = (sumQp + (numSubCtus>>1)) / numSubCtus; // round off
```

5 Revision History

Version	Change	Create Date	Author
1.0	Document Created	Feb 18,2020	Zhong Wang
2.0	FFmpeg ROI API	Oct 01, 2020	Zhong Wang
2.1	ROI Filter	Feb 4, 2021	Neil Gunn
2.2	cacheRoi	Apr 16, 2021	Zhong Wang
3.0	Updated to new app note template	November 30, 2021	Balwinder Sidhu

6 Legal Notice

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

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

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

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