Hantro

Application Programming Interface 1.0

7280 MPEG-4/H.263 Encoder

User Manual Version 1.1

Copyright Information

Hantro Products Oy
Kiviharjunlenkki 1
90220 Oulu
FINLAND
www.hantro.com

Glossary

720p	High Definition resolution of 1280x720 progressive video
API	Application Programming Interface
bps	Bits per second
CIF	Common Interchange Format (352x288 pixels)
CIR	Cyclic Intra Refresh
fps	Frames per second
GOB	Group of Macroblocks
GOV	Group of Video Object Plane
HEC	Header Extension Code
H.263	Video Coding Standard for low bit rate communication (ITU-T)
HD	High Definition
MB	Macroblock (16x16 pixel)
mbps	Mega (1000000) bits per second
MPEG-4	Motion Picture Experts Group standard 4 (ISO / IEC 14496-2)
PAL	Phase Alternate Line (resolution 720x576 pixels)
QCIF	Quarter CIF (176x144 pixels)
QVGA	Quarter Video Graphics Array (320x240 pixels)
QP	Quantization parameter
QQVGA	Quarter QVGA (160x120 pixels)
RVLC	Reversible Variable Length Coding
sub-QCIF	Another video resolution (128x96 pixels)
SVH	Short Video Header
SXGA	Super eXtended Graphics Array (1280x1024 pixels)
VBV	Video Buffering Verifier, a part of MPEG-4 standard
VGA	Video Graphics Array (resolution 640x480 pixels)
VLC	Variable Length Coding
VO	Video Object, MPEG-4 Video stream layer
VOL	Video Object Layer, MPEG-4 Video stream layer
VOP	Video Object Plane, MPEG-4 Video stream layer
VOS	Video Object Sequence, MPEG-4 Video stream layer
VP	Video Packet, MPEG-4 Video stream layer
VS	Video Stabilization

Table of Contents

1 Introduction

This document presents the Application Programming Interface (API) of the Hantro 7280 MPEG-4 and H.263 hardware based encoder. The encoder is able to encode MPEG-4 standard [1] main profile level 4 with simple profile tools and H.263 standard [2] baseline profile compatible video streams.

The encoder conforms to the MPEG-4 Simple profile and H.263 baseline profile and can encode streams up to a maximum picture size of 1280x1024 (SXGA). The encoder can produce 16 Mbps streams at 30 fps but the real maximum performance is very much system dependent (CPU, bus, memory speed, etc.).

The usage of the API is described in chapter 2. Chapter 3 gives guidelines to video frame storage format. The detailed function interfaces and data types are introduced in chapter 4. Chapter 5 gives application examples. References are presented in the end of the document.

In the document all functions, parameters, data types and code are described in `Courier new (syntax style)` font. Notes and filenames are written in *italic* expression.

This document assumes that the reader understands the fundamentals of C-language and the MPEG-4 and H.263 standards.

2 API Version History

Table 1 describes the released API versions, any changes introduced.

TABLE 1 API VERSION HISTORY

API version	Changes/Comments
1.0	Original version
1.0	User Manual version 1.1: fixed Encoded picture size for **MPEG4_ADV_SIMPLE_PROFILE_LEVEL_5** in Table 4 to 1620 MB

3 Usage of the Encoder

The block diagram of a typical encoding process is depicted in Figure 1. The figure shows the main steps of the encoder usage: the initialization, the configuration (optional), the stream producing with its own sub steps (start, frame encoding and end) and finally the release of the encoder.

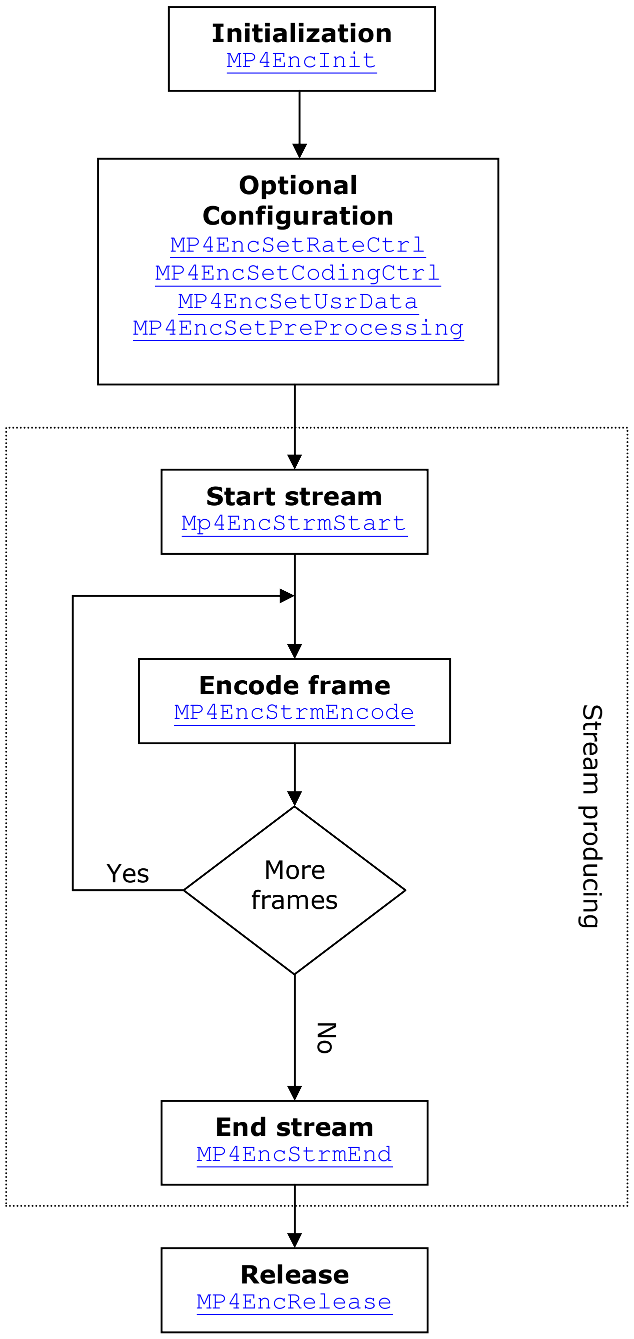

FIGURE 1 ENCODING PROCESS BLOCK DIAGRAM

3.1 Encoder limitations

Depending on the application needs the 7280 encoder hardware can be scaled down to support only lower resolutions (720p, D1, VGA, CIF, etc.). The encoder software will always check what the maximum supported resolution is and fail to initialize if too big picture size is used.
The video stabilization block can be left out also from the design, in which case the pre-processor will fail to initialize with the stabilization turned on.

NOTE! Check the exact specifications of the encoder version in use.

3.2 Initialization and release of the encoder

In order to be able to use the encoder, it has to be properly initialized first. The initialization is done by calling `MP4EncInit`. The call will allocate all the resources needed by the encoder and will execute all the necessary setups in order to have a fully operational encoder. If successful, the initialization call will return a new instance of the encoder, which will be used as an identifier for all the subsequent encoder operations.

The software has support for multiple encoder instances. This means that it is possible to initialize two different encoders and use them for encoding two different streams at the same time. The only limitation is that the hardware can encode only one frame at a time.

The initialization call will return `ENC_OK` for a successful initialization. The error code returned in case of a failure will give hints about what went wrong during the initialization process.

The encoder needs a certain number of parameters when initializing. These parameters are the most significant ones, parameters that can't be changed during the whole encoding process.

Stream profile and level indication

This value is defined by both MPEG-4 and H.263 standards and it will force restrictions to all of the encoder's parameters. For a more friendly approach, the supported profile and level codes are enumerated.
As mentioned above by specifying a certain profile and level, the user sets restrictions to many of the encoder parameters. Table 2 and Table 3 present the typical encoding values used with each profile and level. Table 4 and Table 5 present the most important restrictions set by each profile and level.

TABLE 2 MPEG-4 PROFILES AND LEVELS AND THEIR TYPICAL USAGE

Profile&Level	**Encoded Picture Size**	**Frame Rate [fps]**	**Bit Rate [kbps]**
MPEG4_SIMPLE_PROFILE_LEVEL_0	sub-QCIF	15	64
	QQVGA	15	64
	QCIF	15	64
MPEG4_SIMPLE_PROFILE_LEVEL_0B	sub-QCIF	15	128
	QQVGA	15	128
	QCIF	15	128
MPEG4_SIMPLE_PROFILE_LEVEL_1	sub-QCIF	30	64
	QQVGA	15	64
	QCIF	15	64
MPEG4_SIMPLE_PROFILE_LEVEL_2	QQVGA	30	128
	QCIF	30	128
	CIF	15	128
MPEG4_SIMPLE_PROFILE_LEVEL_3	QCIF	30	384
	CIF	30	384
MPEG4_SIMPLE_PROFILE_LEVEL_4A	VGA	30	4000
MPEG4_SIMPLE_PROFILE_LEVEL_5	PAL	25	8000
MPEG4_SIMPLE_PROFILE_LEVEL_6	720p	30	12000
MPEG4_ADV_SIMPLE_PROFILE_LEVEL_3	CIF	30	768
MPEG4_ADV_SIMPLE_PROFILE_LEVEL_4	352x576	15	3000
MPEG4_ADV_SIMPLE_PROFILE_LEVEL_5	PAL	25	8000
MPEG4_MAIN_PROFILE_LEVEL_4	1280x720	30	16000
MPEG4_MAIN_PROFILE_LEVEL_4	SXGA	30	16000

TABLE 3 H.263 PROFILES AND LEVELS AND THEIR TYPICAL USAGE

Profile&Level	**Encoded Picture Size**	**Frame Rate [fps]**	**Bit Rate [kbps]**
H263_PROFILE_0_LEVEL_10	sub-QCIF	~15	64
	QCIF	~15	64
H263_PROFILE_0_LEVEL_20	sub-QCIF	~30	128
	QCIF	~30	128
	CIF	~15	128
H263_PROFILE_0_LEVEL_30	sub-QCIF	~30	384
	QCIF	~30	384
	CIF	~30	384
H263_PROFILE_0_LEVEL_40	CIF	~30	2048
H263_PROFILE_0_LEVEL_50	CIF	~30	4096
H263_PROFILE_0_LEVEL_60	720*288	~30	4096

H263_PROFILE_0_LEVEL_70	PAL	~25	4096

NOTE! The H.263 baseline profile level 10 to 40 streams support just the standard CIF, QCIF and sub-QCIF encoded picture sizes. For a custom size a higher level has to be specified.

NOTE! Within the MPEG-4 Advanced Simple and Main Profiles just Simple Video Objects are supported. This are marked accordingly in the stream headers.

TABLE 4 MPEG-4 PROFILES AND LEVELS AND THEIR MAXIMAL VALUES

Profile&Level	Encoded Picture Size [MB]	Macroblock Rate [MB/s]	Bit Rate [kbit/s]	VP size
MPEG4_SIMPLE_PROFILE_LEVEL_0	99	1485	64	2048
MPEG4_SIMPLE_PROFILE_LEVEL_0B	99	1485	64	2048
MPEG4_SIMPLE_PROFILE_LEVEL_1	99	1485	64	2048
MPEG4_SIMPLE_PROFILE_LEVEL_2	396	5940	128	4096
MPEG4_SIMPLE_PROFILE_LEVEL_3	396	11880	384	8192
MPEG4_SIMPLE_PROFILE_LEVEL_4A	1200	36000	4000	16384
MPEG4_SIMPLE_PROFILE_LEVEL_5	1620	40500	8000	16384
MPEG4_SIMPLE_PROFILE_LEVEL_6	3600	108000	12000	16384
MPEG4_ADV_SIMPLE_PROFILE_LEVEL_3	396	11880	768	4096
MPEG4_ADV_SIMPLE_PROFILE_LEVEL_4	792	23760	3000	8192
MPEG4_ADV_SIMPLE_PROFILE_LEVEL_5	1620	23760	8000	16384
MPEG4_MAIN_PROFILE_LEVEL_4	16320	489600	38400	16384

TABLE 5 H.263 PROFILES AND LEVELS AND THEIR MAXIMAL VALUES

Profile&Level	Encoded Picture Size	Frame Rate [fps]	Bit Rate [kbps]
H263_PROFILE_0_LEVEL_10	QCIF	(30 000)/2002	64
H263_PROFILE_0_LEVEL_20	CIF	(30 000)/2002(1 (30 000)/1001(2	128
H263_PROFILE_0_LEVEL_30	CIF	(30 000)/1001	384
H263_PROFILE_0_LEVEL_40	CIF	(30 000)/1001	2048
H263_PROFILE_0_LEVEL_50	CIF	(30 000)/1001	4096
H263_PROFILE_0_LEVEL_60	720*288	(30 000)/1001	8192
H263_PROFILE_0_LEVEL_70	PAL	(30 000)/1001	16384

(1 CIF resolution
(2 QCIF and sub-QCIF resolution

The standards define the limitations in a more complex way, so for more in detail information, please consult the relevant papers [1], [2].

The profile and level selection cannot be changed after the initialization.

Encoded picture size

The width and height of the encoded picture in pixels has to be specified when initializing an encoder instance. Notice the difference between the input picture size as captured by a camera and the final encoded image size (See Video preprocessor usage). The input image size can be different than the encoded one if before the encoding process the input image is cropped. The main limitations of the encoded picture's size are set by the selected profile and level. See tables 1 and 2 for more details. Also some implementation specific limitations are in force: the encoded picture width has to be multiple of 4, the height a multiple of 2. The smallest encoded picture size is 96x96 and the maximum is 1280x1024 (or 1024x1280 when rotated) pixels. Even though the encoded picture width can be a 4 multiple, the horizontal scanline (stride) has to be a 16 multiple, meaning that the memory offset form the start of a pixel row to the beginning of the next pixel row is always a 16 multiple. This assumption can be seen in the initial value of the pre-processor parameters, where the input source image width is the rounded up 16 multiple of the encoded with. There is no such limitation on the height of the picture. Figure 2 shows the described limitation regarding the input picture horizontal scanline. Check also chapter 4.1 Frame size limitations.
The encoded size cannot be altered after the initialization.

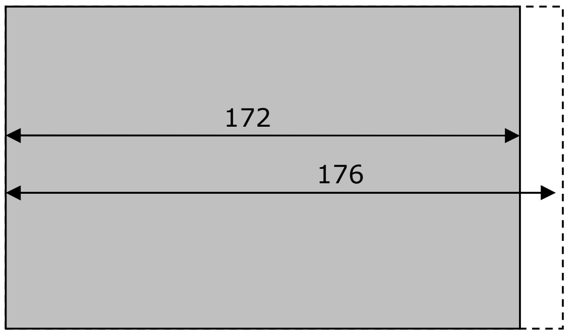

FIGURE 2. ENCODED PICTURE SIZE AND ASSUMED VIRTUAL INPUT PICTURE SIZE

When the encoded picture size is obtained by rotating the original input picture, the size set at the initialization phase reflects the final rotated pictures dimensions.

Frame rate descriptor

In order to be able to efficiently control the bit rate the encoder needs a target frame rate. A so-called time resolution or frame rate numerator and a frame rate denominator specify the frame rate. The time resolution sets the number of equal subintervals, called ticks, within a second. The frame rate is computed as a division of the frame rate numerator (i.e. time resolution) and the frame rate denominator. This way subunit frame rates can be specified, for instance a 7.5 fps frame rate can be used by setting the time resolution to 15 and the denominator to 2.
The time resolution is important also from another point of view. When encoding a video frame, the time stamp of this is specified in the same tick units, which were defined for the time resolution. More details about the time stamp will be given in the Stream producing chapter.

NOTE! Keep in mind that this is just a target frame rate. The time stamps of the encoded video frames will determine the real frame rate. In other words, even if a 30 fps frame rate (i.e. time resolution 30 and frame rate denominator 1) is targeted but the time between two consecutive frames is always 2 ticks, the final stream will have a 15 fps frame rate.

In H.263 streams the frame rate is relative to the (30 000)/1001 frame rate. So, to avoid rounding errors please use '30 000' for the time resolution (numerator) and multiples of 1001 for the frame rate denominator.

Stream type indication

Several stream types can be chosen. Depending on the target environment certain stream types will have to be generated. There are several types of MPEG-4 compatible streams and also H.263 compatible streams. Different stream types imply the usage of certain encoding tools. It is important that the user will carefully select the correct type in such a way that the final stream will meet its target audience expectations. In other words it should be always kept in mind the stream will have to be decoded at some point and the decoder will have to support all the tools in the encoding process.
More details about the different stream types can be seen in the API Reference chapter, description for `MP4EncInit` call. Ultimately the best description of the stream properties can be found in the relevant MPEG-4 and H.263 standard papers [1], [2].

TABLE 6 STREAM TYPES AND THE RELEVANT ENCODING TOOLSTHEY USE

Stream type	Video packages	Data partitioning	RVLC	4MV
MPEG4_PLAIN_STRM	OFF	OFF	OFF	ON
MPEG4_VP_STRM	ON	OFF	OFF	ON
MPEG4_VP_DP_STRM	ON	ON	OFF	ON
MPEG4_VP_DP_RVLC_STRM	ON	ON	ON	ON
MPEG4_SVH_STRM	OFF	OFF	OFF	OFF
H263_STRM	OFF	OFF	OFF	OFF

NOTE! Other limitations of the tools in use set by the selected profile and level might be in force.

NOTE! Mpeg4 stream with data partitioning (DP) requires higher software processing power so use it just when truly necessary and be aware of the processing requirements.

Default encoder initial values

- The output bit rate is set to the typical bit rate for the selected frame size and stream profile and level (See Table 1 and 2).
- VOP and MB based rate control is enabled. Frame skipping is disabled. VBV model is enabled.
- Pre-processor (video stabilization, color conversion, rotation and cropping) is disabled.
- HEC is enabled.
- GOB insertion is disabled.
- The default video package size is set to 400 bits.

Encoder release

At the end of the encoding process the encoder in use has to be released. Use MP4EncRelease to perform a safe release of all the resources allocated when the encoder was initialized. If a release fails most probably the encoder instance was corrupted and there is no safe way to assure that all the encoder's resources were freed.

3.3 Configuration of the encoder

There are several encoding parameters that can be updated after the encoder has been initialized. The configuration API calls take as parameter an encoder instance and the address of a specific data structure that contains the new values. For most parameters it is possible to read the current values in use by the encoder. This provides an easy way of changing just some of the values from a bunch of parameters grouped together in a structure. Some of the encoder parameters (stream user data) can be changed just before a stream is started and others can be altered at any time between two frame encoding.

Bit rate control configuration

As mentioned in the chapter about the encoder initialization, the rate control (RC) is configured initially to produce the maximum bit rate allowed within the selected stream profile and level. The bit rate and some other rate control parameters can be updated after the initialization of the encoder using MP4EncSetRateCtrl. The current parameters can be retrieved using MP4EncGetRateCtrl. The output bit rate is controlled by changing the quantization parameter (QP) or in extreme cases by skipping entire frames from being encoded.

The rate control parameters value can be changed at any time during the encoding sequence. A description of the different rate control settings is given below.

- **VOP based rate control** – this is a frame based rate control algorithm, which can be turned ON or OFF. If ON the RC can adjust the QP between frames.
- **MB based rate control** – this is a macroblock based rate control, which can be turned ON or OFF. If ON the RC can adjust the QP inside a frame.
- **VOP skipping** – when the output rate cannot be adjusted just via the QP changes frame skipping can help control it; it can be turned ON or OFF. Note that if VBV is enabled it may skip VOPs even if VOP skipping is disabled.
- **Default QP** – this is the default or initial QP in use by the encoder. This will be used for the first encoded VOP when the rate control is turned ON or during the whole encoding sequence if the rate control is turned OFF.
- **Min QP** – this is the minimum value QP that can be used by the encoder and is relevant just when the RC is turned ON
- **Max QP** – this is the maximum QP which can be set by the encoder; also in use just if the RC is turned ON
- **Output bit rate** – this is the target bit rate for the output stream; its maximum value is set by the profile and level of the stream
- **Video rate buffer verifier –** This is an algorithm for checking a bit stream with its bit rate, to verify that the amount of rate buffer memory required in a decoder is less than the stated buffer size. This can be turned ON or OFF but by turning it OFF the output stream might not be 100% standard compatible.
- **GOP length** – GOP is a group of successive pictures within a video stream. GOP usually contains an Intra frame (I-frame) and several Inter frames (Predicted

frame). GOP length tells the distance between two Intra frames. Intra frames make video more easily seek-able and editable, but they use more bits. Rate control uses the GOP length to match the average bit rate of each GOP to the target bit rate. Recommended setting for rate control GOP length is the distance between two Intra frames.

VBR and CBR video

Variable Bit Rate (**VBR**) is best choice for locally stored video. More bits are allocated for complex sections and less bits for simple sections. VBR has better quality versus space ratio than CBR video. In Hantro encoder VBR is constrained so that rate control tries to reach set average bit rate over the time. Refer to Table 4 and the MPEG-4 standard [1] for maximum bit rate limits.

Constant Bit Rate (**CBR**) is useful for streaming video over constant bandwidth channel. CBR is not good choice for storage, since it doesn’t allocate enough bits for complex sections and wastes bits on simple sections. In a real time encoder a way to achieve CBR is to change QP of macroblocks inside the frame during encoding. This might have the adverse effect of lower part of the frame becoming blurry.

Using Hantro encoder in **VBR** mode:

vopRc = 1
mbRc = 0
vbv = 0

Using Hantro encoder in **CBR** mode:

vopRc = 1
mbRc = 1
vbv = 1

Inserting user data into the stream

User data can be inserted just in MPEG-4 streams. There are four types of user data considering the places where these are inserted.

1. Visual Object Sequence (VOS) header user data – this is inserted when the VOS header is generated
2. Visual Object (VO) header user data – this is inserted when the VO header is generated
3. Video Object Layer (VOL) header user data – this is inserted when the VOL header is generated
4. Group of VOP (GOV) header user data – this is inserted when a GOV header is generated

The first three types are inserted when the stream is started by a call to Mp4EncStrmStart. So, these can be set at any time between the encoder initialization and the start of a stream. The GOV user data will be inserted in the next GOV header (See MP4EncSetCodingCtrl). Several GOV headers can be generated during an encoding process. A previously set user cannot be retrieved from the encoder.

The user will specify the address of a data buffer and the size of it in bytes. The encoder will make an internal copy of this and it will insert it into the stream at a proper time. This way the data buffer at the user side can be released after the setting. Any new setting will override the previously set user data. Also after insertion into the stream the internal copy is released, so for GOV user data, this will be inserted just once and not every time a GOV header is generated. Keep in mind to adjust the output buffer's size, which is passed as parameter for all the stream generator API calls, to reflect the amount of user data inserted.

Example:

```
MP4EncInst encoder;
const char * mydata = “My custom user data”;
MP4EncUsrDataType type = MPEG4_VOS_USER_DATA;
u32 bytes = strlen(mydata);

/* encoder is already initialized and we have an instance */

/* insert some VOS user data */
if( MP4EncSetUsrData(encoder, mydata, bytes, type) != ENC_OK )
{
      /* handle error */
}
```

Coding control settings

There are several coding control parameters that can be altered using MP4EncSetCodingCtrl. Current settings can be retrieved using MP4EncGetCodingCtrl. The recommended way of changing any of the coding control parameters is first to retrieve the current settings and then change the desired ones.

The coding control parameter values can be changed at any time during the encoding sequence between frames. A description of the different coding control parameters are given below.

Overview of the available options

- HEC (Header Extension Code) insertion into a video packaged MPEG-4 stream. By enabling the HEC insertion the stream will contain more data that will help the error recovery at the decoder side when the decoded stream is corrupted. The insertion can be turned ON/OFF.
- Setting the video package size. When video packaged stream is produced the size of a VP can be specified. Video packages also help in the error recovery of the erroneous streams and their role is to split the one frame data into smaller independent packages. The size is specified in bits. Note that the real video package size will be bigger than the target size. This is due to the fact that a full macroblock has to be encoded before a video package can be ended.
- GOV header insertion. GOV headers can be inserted at any time between frames into an MPEG-4 stream. The only rule is that the first encoded frame after a GOV header was inserted is INTRA coded. Here the GOV insertion is just enabled or disabled, the insertion is done by calling MP4EncStrmEncode.
- GOB header insertion. GOB headers are the counter part of the video package headers for H.263 and MPEG-4 SVH streams. They have the same role to insert more resynchronization points into the stream, which could help in the error recovery. The insertion can be turned ON/OFF.

Example:

```
MP4EncInst encoder;
MP4EncCodingCtrl codingCfg;

/* encoder is already initialized and we have an instance */

if(MP4EncGetCodingCtrl(encoder, &codingCfg) != ENC_OK)
{
      /* handle error */
}
else
{
      codingCfg.insGOV = 1; /* enable GOV header insertion */

      if(MP4EncSetCodingCtrl(encoder, &codingCfg) != ENC_OK)
      {
            /* handle error */
      }
}
```

Other stream parameters

These are set using MP4EncSetStreamInfo.

- Video Range
  Specifies the range of the encoded video samples. If not set correctly the video range dynamic might suffer degradation on the decoder side.

- Pixel Aspect Ratio
  Specifies the pixel aspect ratio of the encoded picture. By default a square pixel is assumed.

3.4 Stream producing

After a successful initialization and an optional configuration, the encoder is ready to produce an MPEG-4 or H.263 compliant stream. There are three separate phases for producing a stream:

1. **Starting the stream** by calling `Mp4EncStrmStart`
2. **Encoding video frames** by calling `MP4EncStrmEncode`
3. **Ending the stream** by calling `MP4EncStrmEnd`

Starting the stream

A stream has to be started by a call to `Mp4EncStrmStart`. The user has to provide an output buffer where any possible stream header data will be written. The output buffer is specified with a pointer to the buffer and the size of the buffer in bytes. All the user data except the GOV user data are inserted into the stream and cannot be set afterwards. If successful the call will return `ENC_OK` and the size of the generated stream in bytes.

Encoding video frames

A video sequence can be encoded after a stream has been started. Normally a video sequence is encoded frame by frame by calling `MP4EncStrmEncode`. The main inputs for this function are a buffer, which contains the video frame to be encoded, and another buffer where the output stream will be written.

The supported formats of the input video are: planar YCbCr 4:2:0, semiplanar YCbCr 4:2:0 or interleaved YCbYCr 4:2:2. The format in use has been selected during the initialization. Independently of what input format is used the memory area where the buffers are located has to be 64-bit aligned, linear and hardware accessible. With other words the addresses, which are passed as parameters to the `MP4EncStrmEncode` function, are bus addresses and will be given to the hardware unmodified and they will be used to read the input video frame.

The output buffer is used by the software to construct the final stream. The pointer to the buffer and the size of the buffer will be passed to the encode function. The size of the buffer has to be correlated with the target bit rate and frame rate, for instance, when the target bit rate is 3 Mbps and the frame rate is 30 fps the average frame size is 100 000 bits. Keep in mind that this is an average value and single encoded frames can take many times more space. Optimal value for output buffer size is the size of the VBV buffer. This is specified by the standard and a single frame should not be larger than the buffer. If the encoder reaches the end of the output buffer, it will discard the current frame. This is even more likely when the rate control is disabled. Even if the current frame is lost the encoding process can continue with a new frame, which will be INTRA coded just to assure that all the possible errors are not perpetrated.

One other important parameter for every video frame is its time stamp. This has to be specified at every call of `MP4EncStrmEncode`. The time stamp is relative to the previously encoded video frame and it is specified in ticks (time units). Please, remember the time resolution set when the encoder was initialized. The time resolution has specified the number of ticks per second. The first encoded frame normally has a time stamp zero. For instance, if the time resolution was set to 30 ticks and the frame rate is 30 fps than the relative time between frames is 1 tick.

After a frame has been encoded (i.e. MP4EncStrmEncode returns ENC_VOP_READY) the absolute time stamp of the video frame is returned to the user. By absolute time we mean the time since the first frame was encoded. Be careful, this time is the one encoded in the stream and is not the computing time or any other time related to the encoding process. The time format is: hours, minutes, seconds, time resolution and time increment. The time increment divided by the time resolution will give the subunits of a second.

When video packaged stream is produced the individual video package sizes can be known also. The user has to provide a buffer where the sizes of different video packages are returned. This buffer has to be big enough to fit one u32 value for each video package produced. The theoretical maximum number of video packages produced is equal to the number of macroblocks per picture. If the end of the VP size buffer is not reached after the last video package, a zero will follow the last significant value. For the maximum supported resolution (SXGA) the maximum needed VP size buffer has 20484 bytes (when `sizeof(u32) = 4`).

NOTE! There is a limit in place for the number of consecutive predicted frames generated (INTER VOP). Because DCT calculation errors can accumulate in time and cause picture degradation, this limit is set at 128 consecutive predictions. This means that after 128 consecutive INTER frames the encoder will generate an INTRA frame.

GOV header insertion

The GOV header insertion is enabled with a call of MP4EncSetCodingCtrl. After this the actual stream will be produced when MP4EncStrmEncode is called. This call will return ENC_GOV_READY and in the output stream buffer will contain the header data. The header is inserted just once after it was enabled.

Ending the stream

A stream will be ended by a call to MP4EncStrmEnd. The user has to provide an output buffer where any possible stream end header data will be written. If successful the call will return ENC_OK and the size of the generated stream in bytes.

3.5 Video pre-processor usage

The encoder includes the following pre-processing blocks:

1. YCbYCr 4:2:2 to YCbCr 4:2:0 conversion
2. Picture rotation
3. Picture cropping
4. Video stabilization

By default the encoder input is considered to be in planar YUV 4:2:0 format. Cropping, rotation and stabilization are disabled.

3.5.1 Color conversion

The conversion block reads the input picture in interleaved YCbCr 4:2:2 format. The picture is converted into planar YCbCr 4:2:0 format by sub-sampling the chrominance samples vertically and rearranging the samples. The conversion is configured at the initialization phase and it will affect every encoded picture.

3.5.2 Picture rotation

The rotation block rotates the picture 90 degrees clockwise or counter-clockwise. The rotation is configured at the initialization phase and it will affect every encoded picture. Figure 3 illustrates the picture dimensions when rotating the input.

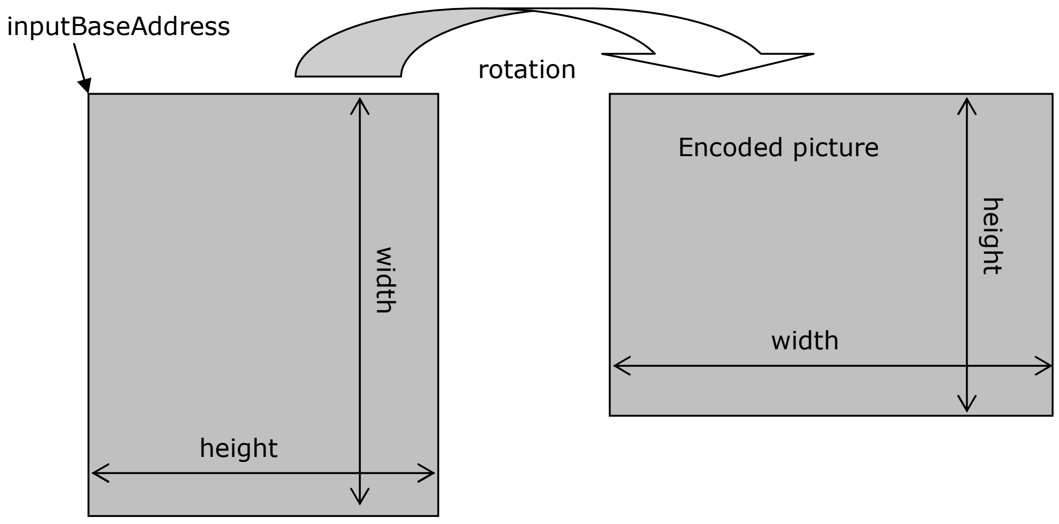

FIGURE 3 PICTURE DIMENSIONS WITH ROTATION

3.5.3 Picture cropping

The picture cropping provides a way to read and encode a smaller selected part of a bigger picture. The cropping is controlled by setting the size of the encoded picture and the top-left corner coordinates of the encoded picture relative to the top-left corner of the input picture with `MP4EncSetCrop`. `MP4EncGetCrop` retrieves the current status of the block.

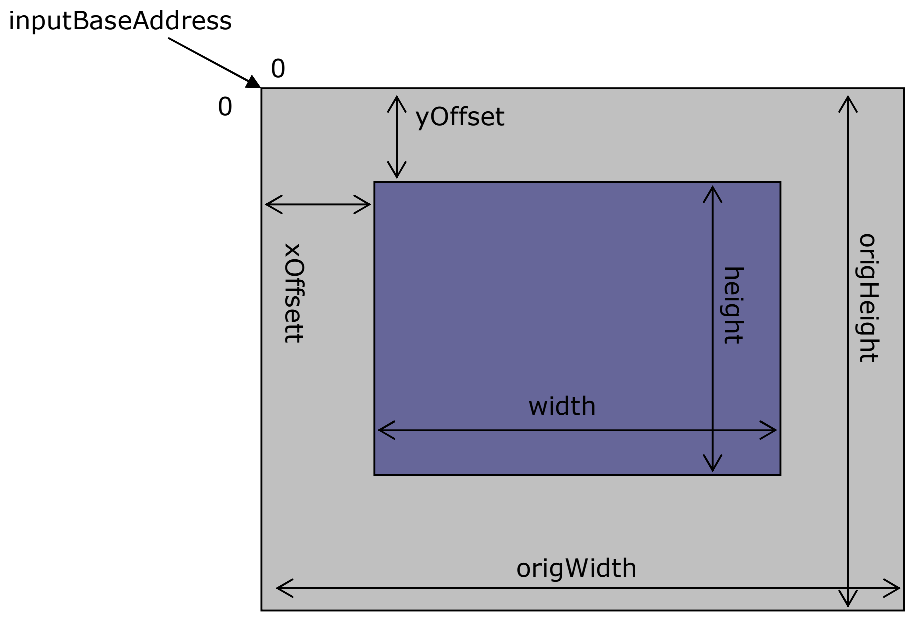

FIGURE 4 RELATIONS BETWEEN THE FULL INPUT IMAGE'S AND THE ENCODED IMAGE'S DIFFERENT SIZES

Figure 4 illustrates the relations between picture parameters. The grey picture is the input picture as captured by a camera and the blue picture is the one encoded by the encoder. Below are the conditions that have to be satisfied in order to enable the cropping block:

$$origWidth >= (width + 15) \& (\sim 15)$$
$$origWidth - xOffset - width \geq 0$$
$$origHeight - yOffset - height \geq 0$$

The input source picture horizontal stride (in pixels) has to take always 16 multiple values. This means that when for example, the input width is 170 pixels the horizontal stride has to be 176.

Offset of pixel *n* in the picture buffer:

$$pixel\_offset = (n/width)*((width+15) \& (\sim 15)) + (n\%width))$$

3.5.4 Video stabilization

The encoder can perform a video stabilization function, which will compensate for any unwanted movement introduced in the video sequence by a shaky capturing device. In order to enable the stabilization the input picture has to be bigger than the encoded picture by at least 8 pixels (both dimensions). The stabilization function will determine the best place from where to read the encoded picture within the boundaries of the input picture. The stabilization uses the cropping function of the encoder and because of this no other application specified cropping can be performed.
In this mode the encoder will always process 2 pictures at a time. One will be encoded and the other will be stabilized. The currently stabilized picture will be encoded next time.

4 Video Frame Storage Format

The input picture format of the encoder is planar YCbCr 4:2:0, semiplanar YCbCr 4:2:0 or interleaved YCbCr 4:2:2. Figure 5 describes how the planar YCbCr 4:2:0 picture is stored in external memory. The luminance and both chrominance components are stored in three buffers and they must be located in a linear and physically contiguous memory block.

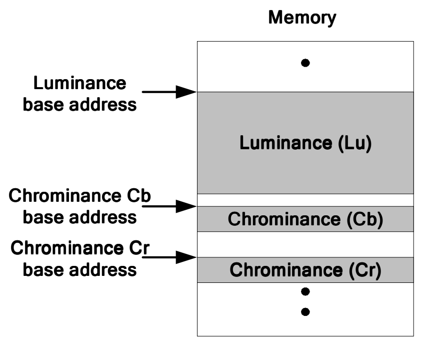

FIGURE 5. PLANAR YCBCR 4:2:0 PICTURE STORAGE IN EXTERNAL MEMORY

Figure 6 describes the format of semiplanar YCbCr 4:2:0 picture. The luminance is equal to planar format but the Cb and Cr chrominance components are interleaved together.

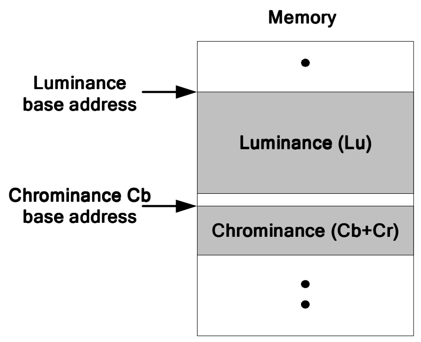

FIGURE 6. SEMIPLANAR YCBCR 4:2:0 PICTURE STORAGE IN EXTERNAL MEMORY

Figure 7 describes the format of interleaved YCbCr 4:2:2 picture where all the three components are interleaved. The sample order can be either Y-Cb-Y-Cr or Cb-Y-Cr-Y.

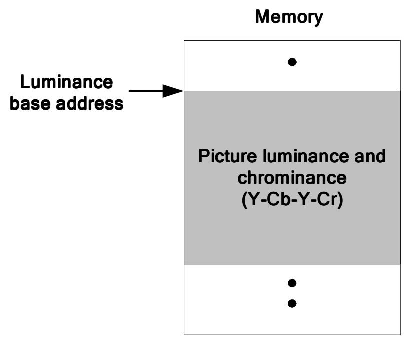

FIGURE 7. INTERLEAVED YCBCR 4:2:2 PICTURE STORAGE IN EXTERNAL MEMORY

The input picture data endianess is set when the encoder software is compiled and can't be changed during run-time. The sizes of the different components are based on the picture's dimensions and the input format:

Planar YCbCr 4:2:0	Y = width x height [bytes]
	Cb = width/2 x height/2 [bytes]
	Cr = width/2 x height/2 [bytes]

Semiplanar YCbCr 4:2:0	Y = width x height [bytes] Cb+Cr = width x height/2 [bytes]
Interleaved YCbCr 4:2:2	Y+Cb+Cr = width x height x 2 [bytes]

4.1 Frame size limitations

YCbCr 4:2:0 limitations regarding the input picture size and the encoded picture size:

encoded_width % 4 = 0
encoded_height % 2 = 0
input_width >= (encoded_width + 15) & (~15)
input_width % 16 = 0
input_height >= encoded_height
input_height % 2 = 0

YCbCr 4:2:2 limitations regarding the input picture size and the encoded picture size:

encoded_width % 4 = 0
encoded_height % 2 = 0
input_width >= (encoded_width + 15) & (~15)
input_width % 8 = 0
input_height >= encoded_height
input_height % 2 = 0

Where:

%	- remainder operator
&	- bitwise AND operator
~	- bitwise NOT operator

5 Interface Functions

This chapter describes the API declared in *mp4encapi.h*.

The following common numeric data types are declared in *basetype.h*:
u8 – unsigned 8 bits integer value
i8 – signed 8 bits integer value
u16 – unsigned 16 bits integer value
i16 – signed 16 bits integer value
u32 – unsigned 32 bits value
i32 – signed 32 bits value

5.1 MP4EncGetApiVersion

Syntax

```
MP4EncApiVersion MP4EncGetApiVersion(void)
```

Purpose

Returns the encoder's API version information. Does not require the encoder initialization.

Parameters

None.

Return value

`MP4EncApiVersion`
A structure containing the API's major and minor version number.

Description

```
typedef struct MP4EncApiVersion
{
    u32 major;
    u32 minor;
} MP4EncApiVersion;
```

`major`
API major version number.

`minor`
API minor version number.

5.2 MP4EncGetBuild

Syntax

`MP4EncBuild` `MP4EncGetBuild(void)`

Purpose

Returns the hardware and software build information of the encoder. Does not require encoder initialization.

Parameters

None.

Return value

`MP4EncBuild`

A structure containing the encoder‛s build information

```
typedef struct
{
    u32 swBuild;
    u32 hwBuild;
} MP4EncBuild;
```

`swBuild`

The internal release version of the 7280 MPEG-4 software.

`hwBuild`

The internal release version of the 7280 hardware.

5.3 MP4EncInit

Syntax

```
MP4EncRet MP4EncInit(const MP4EncCfg *pEncCfg,
                           MP4EncInst *instAddr )
```

Purpose

Call this function to init the encoder and get an instance of it.

Parameters

`MP4EncCfg` `*pEncCfg`

Points to a structure that contains the encoder’s initial configuration parameters.

`MP4EncInst` `*instAddr`

Points to a space where the new encoder instance pointer will be stored.

Return value

`ENC_OK` – initialization successful
`ENC_NULL_ARGUMENT` – error, a pointer argument had an invalid NULL value
`ENC_INVALID_ARGUMENT` – error, one of the arguments was invalid
`ENC_MEMORY_ERROR` – error, the encoder was not able to allocate memory
`ENC_EWL_ERROR` – error, the encoder's system interface failed to initialize
`ENC_EWL_MEMORY_ERROR` – error, the system interface failed to allocate memory

Description

```
typedef struct
{
    MP4EncProfileAndLevel profileAndLevel;
    MP4EncStrmType strmType;
    u32 width;
    u32 height;
    u32 frmRateNum;
    u32 frmRateDenom;
} MP4EncCfg;
```

`profileAndLevel`

Specifies according to the standards the profile and level of the generated stream. The possible values are:

```
MPEG4_SIMPLE_PROFILE_LEVEL_0
MPEG4_SIMPLE_PROFILE_LEVEL_0B
MPEG4_SIMPLE_PROFILE_LEVEL_1
MPEG4_SIMPLE_PROFILE_LEVEL_2
MPEG4_SIMPLE_PROFILE_LEVEL_3
MPEG4_SIMPLE_PROFILE_LEVEL_4A
MPEG4_SIMPLE_PROFILE_LEVEL_5
MPEG4_SIMPLE_PROFILE_LEVEL_6
MPEG4_ADV_SIMPLE_PROFILE_LEVEL_3
MPEG4_ADV_SIMPLE_PROFILE_LEVEL_4
MPEG4_ADV_SIMPLE_PROFILE_LEVEL_5
MPEG4_MAIN_PROFILE_LEVEL_4
H263_PROFILE_0_LEVEL_10
H263_PROFILE_0_LEVEL_20
H263_PROFILE_0_LEVEL_30
H263_PROFILE_0_LEVEL_40
H263_PROFILE_0_LEVEL_50
H263_PROFILE_0_LEVEL_60
H263_PROFILE_0_LEVEL_70
```

For more information check the chapter about Initialization and release of the encoder.

`strmType`

Specifies the type of the stream generated. This will set the main coding tools in use. The possible values are:
`MPEG4_PLAIN_STRM` **–** The encoder will produce plain MPEG-4 stream. No error resilience tools will be in use.
`MPEG4_VP_STRM` **–** The encoder will produce MPEG-4 stream with resynchronization markers, which define the boundaries of so called video packages (VP). The size of a video package can be changed after the initialization of the encoder. The resynchronization markers will help in the error recovery of a decoder that receives erroneous MPEG-4 stream.
`MPEG4_VP_DP_STRM` – The encoder will produce MPEG-4 stream with resynchronization markers and data partitioning (DP). By partitioning the different data in the stream a better error recovery rate can be achieved. The maximum size of video packages when DP is used is set by the stream profile and level specified by *profileAndLevel*.
`MPEG4_VP_DP_RVLC_STRM` – The encoder will produce MPEG-4 stream with resynchronization markers, using data partitioning and reversible VLC (RVLC) code words. The produced stream will give all the advantages of video packages and data partitioning and it will further increase the error resilience by using RVLC. This will allow an MPEG-4 decoder to decode an erroneous stream in reverse stream order.
`MPEG4_SVH_STRM` **–** The encoder will produce MPEG-4 stream that is using short video header (SVH) format for the encoded video frames. This format is equivalent with the H.263 baseline stream format. The stream will have MPEG-4 top stream headers that make it incompatible with pure H.263 decoders. Picture resolution has to be one of: sub-QCIF, QCIF, CIF or 4QCIF.
`H263_STRM` – The encoder will produce a stream compatible with the H.263 standard. Up to PAL resolution supported.

`width`

Valid value range: [96, 1280] for MPEG-4 or [96, 720] for H.263
The width of the encoded image in pixels. This has to be multiple of 4. Horizontal stride is presumed to be the next 16 multiple equal or bigger than this value.

`height`

Valid value range: [96, 1024] for MPEG-4 or [96, 720] for H.263
The height of the encoded image in pixels. This has to be multiple of 2.

`frmRateNum`

Valid value range: [1, 65535]
The numerator part of the input frame rate. The frame rate is defined by the `frmRateNum`/`frmRateDenom` ratio. This value is also used as the time resolution or the number of subunits (ticks) within a second.

`frmRateDenom`

Valid value range: [1, 65535]
The denominator part of the input frame rate. This value has to be equal or less than the numerator part `frmRateNum`.

5.4 MP4EncRelease

Syntax

MP4EncRet `MP4EncRelease(` **MP4EncInst** `inst )`

Purpose

Releases an encoder instance. This will free all the resources allocated at the encoder initialization phase.

Parameters

MP4EncInst `inst`

The encoder instance to be released. This instance was created earlier with an `MP4EncInit` call.

Return value

`ENC_OK` – release successful
`ENC_NULL_ARGUMENT` – error, a pointer argument had an invalid NULL value
`ENC_INSTANCE_ERROR` – error, the encoder instance to be released is invalid

5.5 MP4EncSetStreamInfo

Syntax

MP4EncRet `MP4EncSetStreamInfo(` **MP4EncInst** `inst,`
const **MP4EncStreamInfo** `*pStreamInfo)`

Purpose

Sets some stream header parameters before starting the stream.

Parameters

MP4EncInst `inst`

The instance that defines the encoder in use.

MP4EncStreamInfo `*pStreamInfo`

Pointer to a structure that contains the stream parameters.

Return value

`ENC_OK` – the setting was successful
`ENC_NULL_ARGUMENT` – error, a pointer argument had an invalid NULL value
`ENC_INVALID_ARGUMENT` – error, one of the arguments was invalid
`ENC_INSTANCE_ERROR` – error, an invalid encoder instance was specified
`ENC_INVALID_STATUS` – error, the stream was already started.

Description

```
typedef struc
{
    u32 videoRange;
    u32 pixelAspectRatioWidth;
    u32 pixelAspectRatioHeight;
} MP4EncStreamInfo;
```

`videoRange`

Valid values: 0 or 1
0 = Range [16, 235] for luminance and [16, 240] for chrominance. Default value.
1 = Range [0, 255] for both luminance and chrominance.
Specifies the black level and range of luminance and chrominance signals. Default value 0.

`pixelAspectRatioWidth`

Valid range: [0, 255]
Horizontal size of the pixel aspect ratio. A zero value will set the pixel aspect ratio to the default square, 1:1 ratio. Default value 0.

`pixelAspectRatioHeight`

Valid range: [0, 255]
Vertical size of the pixel aspect ratio. A zero value will set the pixel aspect ratio to default square, 1:1 ratio. Default value 0.

When the pixel aspect ration is not square the stream headers will contain the extended pixel aspect ration information with the exact values for the width and height.

5.6 MP4EncGetStreamInfo

Syntax

```
MP4EncRet MP4EncGetStreamInfo( MP4EncInst inst,
                                MP4EncStreamInfo *pStreamInfo)
```

Purpose

Retrieves the current stream header parameters.

Parameters

MP4EncInst `inst`

The instance that defines the encoder in use.

MP4EncStreamInfo `*pStreamInfo`

Pointer to a structure where the current header parameters will be saved.

Return value

`ENC_OK` – the retrieving was successful
`ENC_NULL_ARGUMENT` **–** error, a pointer argument had an invalid NULL value
`ENC_INSTANCE_ERROR` – error, an invalid encoder instance was specified

Description

For more info about `MP4EncGetStreamInfo` see the description for `MP4EncSetStreamInfo`.

5.7 MP4EncSetCodingCtrl

Syntax

```
MP4EncRet MP4EncSetCodingCtrl( MP4EncInst inst,
                               const MP4EncCodingCtrl *pCodeParams )
```

Purpose

Sets the encoder’s coding parameters.

Parameters

`MP4EncInst` `inst`
The instance that defines the encoder in use.

`MP4EncCodingCtrl` `*pCodeParams`
Pointer to a structure that contains the encoder’s coding parameters.

Return value

`ENC_OK` – the setting was successful
`ENC_NULL_ARGUMENT` **–** error, a pointer argument had an invalid NULL value
`ENC_INVALID_ARGUMENT` – error, one of the arguments was invalid
`ENC_INSTANCE_ERROR` – error, an invalid encoder instance was specified

Description

```
typedef struct
{
    u32 insHEC;
    u32 insGOB;
    u32 insGOV;
    u32 vpSize;
} MP4EncCodingCtrl;
```

`insHEC`
If different than zero then it will enable the Header Extension Code insertion when MPEG-4 video packaged stream is produced. Disabled by default.

`insGOB`

This a bit mask for inserting Group of Macroblocs (GOB) header into an H.263 or an MPEG-4 short video header stream. Bit 0 (LSB) corresponds to GOB number 0, bit 1 to GOB 1 and so on. Default value is 0.

`insGOV`

If different than zero then a Group of VOPs (GOV) header will be inserted in an MPEG-4 stream. After a GOV header is inserted the following video frame is INTRA coded. GOV is inserted just once. Default value is 0.

`vpSize`

Valid value range: [1, 60000]
Sets the target size in bits of a video package when MPEG-4 video packaged stream is produced. The VP size is limited by standard when the MPEG-4 stream is also data partitioned. Default value is 400.

5.8 MP4EncGetCodingCtrl

Syntax

```
MP4EncRet MP4EncGetCodingCtrl( MP4EncInst inst,
                               MP4EncCodingCtrl *pCodeParams )
```

Purpose

Retrieves the current coding parameters in use by the encoder.

Parameters

`MP4EncInst` `inst`

The instance that defines the encoder in use.

`MP4EncCodingCtrl` `*pCodeParams`

Pointer to a structure where the encoder's coding parameters will be saved.

Return value

`ENC_OK` – the retrieving was successful
`ENC_NULL_ARGUMENT` – error, a pointer argument had an invalid NULL value
`ENC_INSTANCE_ERROR` – error, an invalid encoder instance was specified

Description

For more info about `MP4EncGetCodingCtrl` see the description for `MP4EncSetCodingCtrl`.

5.9 MP4EncSetRateCtrl

Syntax

```
MP4EncRet MP4EncSetRateCtrl(  MP4EncInst inst,
                              const MP4EncRateCtrl *pRateCtrl )
```

Purpose

Sets the rate control parameters of the encoder.

Parameters

MP4EncInst `inst`

The instance that defines the encoder in use.

MP4EncRateCtrl `*pRateCtrl`

Pointer to a structure where the new rate control parameters are set.

Return value

`ENC_OK` – the setting was successful
`ENC_NULL_ARGUMENT` **–** error, a pointer argument had an invalid NULL value
`ENC_INVALID_ARGUMENT` – error, one of the arguments was invalid
`ENC_INSTANCE_ERROR` – error, an invalid encoder instance was specified

Description

```
typedef struct
{
    u32 vopRc;
    u32 mbRc;
    u32 vopSkip;
    i32 qpHdr;
    u32 qpMin;
    u32 qpMax;
    u32 bitPerSecond;
    u32 vbv;
    u32 gopLen;
} MP4EncRateCtrl;
```

`vopRc`

If different than zero it will enable the VOP based rate control. The QP will be changed between VOPs. Enabled by default.

`mbRc`

If different than zero it will enable the macroblock based rate control. The QP will be adjusting also inside a VOP. Enabled by default.

`vopSkip`

If different than zero then frame skipping is enabled, i.e. the rate control is allowed to skip frames if needed. When VBV is enabled, the rate control may have to skip frames despite of this value. Disabled by default.

`qpHdr`

Valid value range: -1 or [1, 31]
The default quantization parameter used by the encoder. If the rate control is enabled then this value is used just at the beginning of the encoding process. When the rate control is disabled then this QP value is used all the time. -1 lets RC calculate initial QP. Default value is 10.

`qpMin`

Valid value range: [1, 31]
The minimum QP that can be used by the RC in the stream. Default value is 1.

`qpMax`

Valid value range: [1, 31]
The maximum QP that can be used by the RC in the stream. Default value is 31.

`bitPerSecond`

The target bit rate as bits per second (bps) when the rate control is enabled. The rate control is considered enabled when any of the VOP or MB based rate control is enabled. Default value is the minimum between the maximum bitrate allowed for the selected profile&level and 4mbps.

`vbv`

If different than zero it will enable the VBV model. Value 1 represents the video buffer size defined by the standard. Otherwise this value represents the video buffer size in units of 16384 bits. Enabled by default with size set by standard.

`gopLen`

Valid value range: [1, 150]
Length of group of pictures. Rate control calculates bit reserve for this GOP length. Recommended value is same as the INTRA frame rate.

5.10 MP4EncGetRateCtrl

Syntax

```
MP4EncRet MP4EncGetRateCtrl(  MP4EncInst inst,
                              MP4EncRateCtrl *pRateCtrl )
```

Purpose

Retrieves the current rate control parameters in use by the encoder.

Parameters

MP4EncInst `inst`

The instance that defines the encoder in use.

MP4EncRateCtrl `*pRateCtrl`

Pointer to a structure where the new rate control parameters will be saved.

Return value

`ENC_OK` – the retrieving was successful
`ENC_NULL_ARGUMENT` **–** error, a pointer argument had an invalid NULL value
`ENC_INSTANCE_ERROR` – error, an invalid encoder instance was specified

Description

For more info about `MP4EncRateCtrl` see the descriptions for `MP4EncSetRateCtrl`.

5.11 MP4EncSetUsrData

Syntax

```
MP4EncRet MP4EncSetUsrData(   MP4EncInst inst,
                              const u8 *pBuf,
                              u32 length,
                              MP4EncUsrDataType type )
```

Purpose

Sets a user data information to be inserted into an MPEG-4 stream.

Parameters

MP4EncInst `inst`
The instance that defines the encoder in use.

const u8 `*pBuf`
Pointer to a buffer that contains the user data to be inserted. The data will be copied internally so it is safe to release it after the call returns.

u32 `length`
The length in bytes of the data contained in the buffer. Zero disables user data.

MP4EncUsrDataType `type`
Defines the MPEG-4 header, which the user data will belong to. Possible values:
`MPEG4_VOS_USER_DATA` – the user data will be inserted into the Visual Object Sequence header
`MPEG4_VO_USER_DATA` – the user data will be inserted into the Visual Object header
`MPEG4_VOL_USER_DATA` – the user data will be inserted into the Video Object Layer header
`MPEG4_GOV_USER_DATA` – the user data will be inserted into the Group of Video Object Plane header

Return value

`ENC_OK` – the setting was successful
`ENC_NULL_ARGUMENT` **–** error, a pointer argument had an invalid NULL value
`ENC_INVALID_ARGUMENT` – error, one of the arguments was invalid
`ENC_INSTANCE_ERROR` – error, an invalid encoder instance was specified
`ENC_MEMORY_ERROR` – error, failed to allocate memory for user data buffer

Description

All the user data can be set at any time before stating the stream (see `Mp4EncStrmStart`). After the stream was started just the `MPEG4_GOV_USER_DATA` can be set. For the latest to be inserted, one has to enable also the whole GOV header insertion by calling `MP4EncSetCodingCtrl`. For several consecutive settings of the same user data type only the last setting is valid.

5.12 MP4EncStrmStart

Syntax

```
MP4EncRet MP4EncStrmStart( MP4EncInst inst,
                           const MP4EncIn *pEncIn,
                           MP4EncOut *pEncOut )
```

Purpose

Starts a new MPEG-4 or H.263 stream.

Parameters

MP4EncInst `inst`
The instance that defines the encoder in use.

MP4EncIn `*pEncIn`
Pointer to a structure where the input parameters are provided.

MP4EncOut `*pEncOut`
Pointer to a structure where the output parameters will be saved.

Return value

`ENC_OK` – the setting was successful
`ENC_NULL_ARGUMENT` **–** error, a pointer argument had an invalid NULL value
`ENC_INVALID_ARGUMENT` – error, one of the arguments was invalid
`ENC_INSTANCE_ERROR` – error, an invalid encoder instance was specified
`ENC_INVALID_STATUS` – error, a stream was already started
`ENC_OUTPUT_BUFFER_OVERFLOW` – error, the output buffer's size was too small to fit the generated stream. Allocate a bigger buffer and try again.

Description

```
typedef struct
{
    u32 busLuma;
    u32 busChromaU;
    u32 busChromaV;
    u32 *pOutBuf;
    u32 outBufBusAddress;
    u32 outBufSize;
    MP4EncVopType vopType;
    u32 timeIncr;
    u32 *pVpBuf;
    u32 vpBufSize;
    u32 busLumaStab;
} MP4EncIn;
```

This data structure is common for all the API calls, which generate stream. Not all the fields are used when starting a stream. Here is the description of the relevant ones:

`pOutBuf`

This is a pointer to the output buffer, which will be used to store the generated stream.

`outBufSize`

This is the size in bytes of the stream buffer described above. When some user data is inserted in the stream start headers then the size of these has to be considered when allocating the stream buffer. Minimum value is 32.

```
typedef struct
{
    TimeCode timeCode;
    MP4EncVopType vopType;
    u32 strmSize;
} MP4EncOut;
```

This data structure is common for all the API calls, which generate stream. Not all fields are used in starting a stream. Here is the description of the relevant ones:

`strmSize`

The actual size of the generated stream in bytes is returned here. If the call was unsuccessful then this value is not relevant at all and it shall be ignored.

5.13 MP4EncStrmEncode

Syntax

```
MP4EncRet MP4EncStrmEncode( MP4EncInst inst,
                            const MP4EncIn *pEncIn,
                            MP4EncOut *pEncOut )
```

Purpose

Encodes a video frame.

Parameters

MP4EncInst `inst`

The instance that defines the encoder in use.

MP4EncIn `*pEncIn`

Pointer to a structure where the input parameters are provided.

MP4EncOut `*pEncOut`

Pointer to a structure where the output parameters will be saved.

Return value

`ENC_VOP_READY` – a frame encoding was finished
`ENC_GOV_READY` – the GOV header was generated. Call again to continue with the frame encoding.
`ENC_NULL_ARGUMENT` – error, a pointer argument had an invalid NULL value
`ENC_INVALID_ARGUMENT` – error, one of the arguments was invalid
`ENC_INSTANCE_ERROR` – error, an invalid encoder instance was specified
`ENC_INVALID_STATUS` – error, stream was not started yet
`ENC_OUTPUT_BUFFER_OVERFLOW` – error, the output buffer's size was too small to fit the generated stream. The whole frame is lost. New frame encoding has to be started.
`ENC_HW_TIMEOUT` – error, the wait for a hardware finish has timed out. The current frame is lost. New frame encoding has to be started.
`ENC_HW_DATA_ERROR` – error. This is caused by invalid data detected in the hardware output. New frame encoding has to be started.
`ENC_HW_BUS_ERROR` – error. This can be caused by invalid bus addresses, addresses that push the encoder to access an invalid memory area. New frame encoding has to be started.
`ENC_HW_RESET` – error. Hardware was reset by external means. New frame encoding has to be started.
`ENC_HW_RESERVED` – error. Hardware could not be reserved for exclusive access.
`ENC_SYSTEM_ERROR` – error. Error in a system calls. Encoder has to be released.

Description

```
typedef struct
{
    u32 busLuma;
    u32 busChromaU;
    u32 busChromaV;
    u32 *pOutBuf;
    u32 outBufBusAddress;
    u32 outBufSize;
    MP4EncVopType vopType;
    u32 timeIncr;
    u32 *pVpBuf;
    u32 vpBufSize;
    u32 busLumaStab;
} MP4EncIn;
```

This data structure is common for all the API calls, which generate stream. Not all the fields are used when encoding a stream. Here is the description of the relevant ones:

`busLuma`
The bus address of the buffer where the input picture's luminance component is located.

`busChromaU`
The bus address of the buffer where the input picture's first chrominance component (Cb or U) is located.

`busChromaV`
The bus address of the buffer where the input picture’s second chrominance component (Cr or V) is located.

`pOutBuf`
This is a pointer to the output buffer, which will be used to store the generated stream.

`outBufBusAddress`
The bus address of the output buffer, required by the hardware operations. Not needed when video packaged stream is generated.

`outBufSize`
This is the size in bytes of the stream buffer described above. When GOV user data is inserted in the stream the size of the user data has to be considered when allocating the stream buffer. Minimum value is 4096.

`vopType`
The encoding type for specified picture. It can take the following values:
`INTRA_VOP` – the picture will be INTRA coded.
`PREDICTED_VOP` – the picture is wanted to be INTER coded, using the previous picture as a predictor. The actual coding type is selected by the encoder.

`timeIncr`
Valid value range: [0, 127*time resolution]
The time stamp of the frame relative to the last encoded frame. This is given in ticks. A zero value is usually used for the very first frame.

`pVpBuf`
Pointer to a buffer where the individual video package’s sizes will be returned. Set to NULL if the information is not needed. This buffer has to have `sizeof(u32)` bytes for each macroblock of the encoded picture. A zero value is written after the last relevant VP size (when buffer big enough). Maximum size of 20484 bytes for SXGA resolution (when `sizeof(u32) = 4`).

`vpBufSize`
This is the size in bytes of the video package size buffer described above.

`busLumaStab`
The bus address of the luminance component buffer of the picture to be stabilized. Used only when video stabilization is enabled.

```
typedef struct
{
    u32 hours;
    u32 minutes;
    u32 seconds;
    u32 timeIncr;
    u32 timeRes;
} TimeCode;
```

`hours`
The hour part of the time since the first frame was encoded.

`minutes`

The minute part of the time since the first frame was encoded.

`seconds`
The second part of the time since the first frame was encoded.

`timeIncr`
The subunits of the last second of the time since the first frame was encoded.

`timeRes`
The defined number of subunits in a second.

```
typedef struct
{
    TimeCode timeCode;
    MP4EncVopType vopType;
    u32 strmSize;
} MP4EncOut;
```

`timeCode`
The encoded stream time since the first frame was encoded.

`vopType`
The encoding type of the last frame as it was encoded into the stream. This can be different from the type requested by the user. It can take the following values:
`INTRA_VOP` – the frame was INTRA coded.
`PREDICTED_VOP` – the frame was INTER coded
`NOTCODED_VOP` – the frame was not coded at the rate control's request.

`strmSize`
The size of the generated stream in bytes is returned here. If the call was unsuccessful then this value is not relevant at all and it shall be ignored.

5.14 MP4EncStrmEnd

Syntax

```
MP4EncRet MP4EncStrmEnd ( MP4EncInst inst,
                          const MP4EncIn *pEncIn,
                          MP4EncOut *pEncOut )
```

Purpose

Ends a started stream.

Parameters

MP4EncInst `inst`
The instance that defines the encoder in use.

MP4EncIn `*pEncIn`
Pointer to a structure where the input parameters are provided.

MP4EncOut `*pEncOut`

Pointer to a structure where the output parameters will be saved.

Return value

`ENC_OK` – the setting was successful
`ENC_NULL_ARGUMENT` **–** error, a pointer argument had an invalid NULL value
`ENC_INVALID_ARGUMENT` – error, one of the arguments was invalid
`ENC_INSTANCE_ERROR` – error, an invalid encoder instance was specified
`ENC_INVALID_STATUS` – error, the stream was not started or the current frame was not fully encoded.
`ENC_OUTPUT_BUFFER_OVERFLOW` – error, the output buffer's size was too small to fit the generated stream. Allocate a bigger buffer and try again.

Description

```
typedef struct
{
    u32 busLuma;
    u32 busChromaU;
    u32 busChromaV;
    u32 *pOutBuf;
    u32 outBufBusAddress;
    u32 outBufSize;
    MP4EncVopType vopType;
    u32 timeIncr;
    u32 *pVpBuf;
    u32 vpBufSize;
    u32 busLumaStab;
} MP4EncIn;
```

This data structure is common for all the API calls, which generate stream. Not all the fields are used when ending a stream. Here is the description of the relevant ones:

`pOutBuf`
This is a pointer to the output buffer, which will be used to store the generated stream.

`outBufSize`
This is the size in bytes of the buffer described above. Minimum value is 32.

```
typedef struct
{
    TimeCode timeCode;
    MP4EncVopType vopType;
    u32 strmSize;
} MP4EncOut;
```

This data structure is common for all the API calls, which generate stream. Not all the fields are used in ending a stream though. Here is the description of the relevant ones:

`strmSize`
The actual size in bytes of the generated stream is returned here. If the call was unsuccessful then this value is not relevant at all and it shall be ignored

5.15 MP4EncSetPreProcessing

Syntax

```
MP4EncRet MP4EncSetPreProcessing( MP4EncInst inst,
                                  const MP4EncPreProcessingCfg *pPreProcCfg )
```

Purpose

Sets the preprocessing block's parameters.

Parameters

MP4EncInst `inst`
The instance that defines the encoder in use.

MP4EncPreProcessingCfg `*pPreProcCfg`
Pointer to a structure where the new parameters are set.

Return value

`ENC_OK` – the setting was successful.
`ENC_NULL_ARGUMENT` **–** error, a pointer argument had an invalid NULL value.
`ENC_INVALID_ARGUMENT` – error, one of the arguments was invalid.
`ENC_INSTANCE_ERROR` – error, an invalid encoder instance was specified.
`ENC_SYSTEM_ERROR` – error. Error in a system calls. Encoder has to be released.

Description

```
typedef struct
{
    u32 origWidth;
    u32 origHeight;
    u32 xOffset;
    u32 yOffset;
    MP4EncPictureYuvType yuvType;
    MP4EncPictureRotation rotation;
    u32 videoStabilization;
} MP4EncPreProcessingCfg;
```

`origWidth`
This is the input image's full width. This size has to be equal or bigger than the encoded image's width specified at the encoder initialization phase (See `MP4EncInit`). A zero value disables the cropping. It is also restricted depending on the input image format (See Video Frame Storage Format). Horizontal stride is presumed to be the next 16 multiple equal or bigger than this value.
Valid value range: [96, 1920]

`origHeight`

This is the input image's full height. This size has to be equal or bigger than the encoded image's height specified at the encoder initialization phase (See `MP4EncInit`). A zero value disables the cropping.
Valid value range: [96, 1920]

`xOffset`

This is the horizontal offset from the top-left corner of the input image to the top-left corner of the encoded image. Keep in mind that the minimum encoded picture width is 96.
Valid value range: [0, 1824]

`yOffset`

This is the vertical offset from the top-left corner of the input image to the top-left corner of the encoded image. Keep in mind that the minimum encoded picture height is 96.
Valid value range: [0, 1824]

`yuvType`

Specifies the input YUV picture format. The formats are described in Video Frame Storage Format chapter. The possible values are:
`ENC_YUV420_PLANAR`
`ENC_YUV420_SEMIPLANAR`
`ENC_YUV422_INTERLEAVED_YUYV`
`ENC_YUV422_INTERLEAVED_UYVY`

`rotation`

Specifies the YUV picture rotation before encoding. The possible values are:
`ENC_ROTATE_0` – no rotation.
`ENC_ROTATE_90R` – rotates the picture clockwise by 90 degrees before the encoding.
`ENC_ROTATE_90L` – rotates the picture counter-clockwise by 90 degrees before the encoding.

`videoStabilization`

Enables or disables the video stabilization function. Set to a non-zero value will enable the stabilization. The input image's dimensions (`origWidth, origHeight`) have to be at least 8 pixels bigger than the final encoded image's. Also when enabled the cropping offset (`xOffset, yOffset`) values are ignored.

For more information check the Video preprocessor usage chapter. If any invalid configuration value will be set the pre-processing block will keep its old configuration.

5.16 MP4EncGetPreProcessing

Syntax

```
MP4EncRet MP4EncGetPreProcessing( MP4EncInst inst,
                                  MP4EncPreProcessingCfg *pPreProcCfg )
```

Purpose

Retrieves the current preprocessing parameters in use by the encoder.

Parameters

MP4EncInst `inst`

The instance that defines the encoder in use.

MP4EncPreProcessingCfg `*pPreProcCfg`

Pointer to a structure where the retrieved parameters are saved

Return value

`ENC_OK` – the retrieving was successful
`ENC_NULL_ARGUMENT` **–** error, a pointer argument had an invalid NULL value
`ENC_INSTANCE_ERROR` – error, an invalid encoder instance was specified

Description

For more info about `MP4EncGetPreProcessing` see the descriptions for `MP4EncSetPreProcessing`. When the video stabilization is in use the `xOffset` and `yOffset` values hold the latest stabilization result.

6 Application Examples

The source code given in this chapter is just an example and it is not guaranteed to compile as such.

6.1 Encode 300 frames

```
#include "MP4EncApi.h"
#include <stdlib.h>

MP4EncRet ret;
MP4EncInst encoder;
MP4EncCfg cfg;
MP4EncIn encIn;
MP4EncOut encOut;

/* Output buffer:
    Optimal size is the size of the VBV buffer, for example
     Simple Profile Level 0: 20 kB
     Simple Profile Level 3: 80 kB
     Simple Profile Level 5: 224 kB
*/

u32 outbuf_size = 224 * 1024;
u32 outbuf_bus_address = 0;
u32 *outbuf_virt_address = NULL;

u32 picture_size;
u32 pict_bus_address = 0;
u32 pict_bus_address_stab = 0;

/* Step 1: Initialize an encoder instance */

/* 30 fps frame rate */
cfg.frmRateDenom = 1;
cfg.frmRateNum   = 30; /* this is the time resolution also */

/* VGA resolution */
cfg.width        = 640;
cfg.height       = 480;

cfg.strmType     = MPEG4_PLAIN_STRM; /* setup the stream type */

/* and finally the profile&level of the stream */
cfg.profileAndLevel = MPEG4_SIMPLE_PROFILE_LEVEL_5;

if((ret = MP4EncInit(&cfg, &encoder)) != ENC_OK)
{
    /* Handle here the error situation */
    goto end;
}

/* Step 2: Optional extra encoder configuration
   See the next example code for how to change the
   default encoder parameters
*/
```

```
/* Step 3: Allocate linear memory resources. This implementation
       is OS specific and not part of this example. */
    AllocDMAMemory(&outbuf_virt_address, &output_bus_address, &outbuf_size);
    AllocDMAMemory(NULL, &pict_bus_address, &picture_size);

    encIn.pOutBuf    = outbuf_virt_address;
    encIn.busOutBuf  = output_bus_address;
    encIn.outBufSize = outbuf_size; /* bytes */

    /* Step 4: Start the stream */

    ret = MP4EncStrmStart(encoder, &encIn, &encOut);
    if(ret != ENC_OK)
    {
        /* Handle here the error situation */
        goto close;
    }
    else
    {
        SaveStream(encIn.pOutBuf, encOut.strmSize);
    }

    /* Step 5: Encode the video frames */

    ret = ENC_VOP_READY;
    next = 0;
    last = 300; /* encode 300 frames */

    while( (next <= last) &&
          (ret == ENC_VOP_READY ||
           ret == ENC_OUTPUT_BUFFER_OVERFLOW ))
    {

        if(next == 0)
            encIn.timeIncr = 0; /* start time is 0 */
        else
            encIn.timeIncr = 1; /* time units between frames */

        /* Select VOP type */
        if(next == 0)
            encIn.vopType = INTRA_VOP; /* first frame has to be INTRA coded */
        else
            encIn.vopType = PREDICTED_VOP;

        /* Read next frame */
        ReadVideoFrame(pict_bus_address);

        /* we use one linear buffer for the input image */
        encIn.busLuma = pict_bus_address;
        encIn.busChromaU = encIn.busLuma + cfg.width*cfg.height;
        encIn.busChromaV = encIn.busChromaU + (cfg.width/2) * (cfg.height/2);

        /* Loop until one VOP is encoded */
        do
        {
            ret = MP4EncStrmEncode(encoder, &encIn, &encOut);

            switch (ret)
            {
            case ENC_VOP_READY:
                SaveStream(encIn.pOutBuf, encOut.strmSize);
                break;
            case ENC_GOV_READY:
                /* if the GOV insertion was enabled */
                SaveStream(encIn.pOutBuf, encOut.strmSize);
                break;

            default:
                /* All the others are ERROR codes
                   The next frame will be forced to INTRA coded
                   by the encoder
                */
                break;
            }
        }
        while(ret == ENC_GOV_READY);

        next++
    }   /* End of main encoding loop */

    /* Step 6: End stream */
    ret = MP4EncStrmEnd(encoder, &encIn, &encOut);
    if(ret != ENC_OK)
    {
        /* Handle here the error situation */
        goto close;
    }
    else
    {
        SaveStream(encIn.pOutBuf, encOut.strmSize);
    }

    /* Free all resources */
    FreeDMAMemory(outbuf_virt_address, output_bus_address, outbuf_size);
    FreeDMAMemory(NULL, pict_bus_address, picture_size);

close:

    /* Last Step: Release the encoder instance */
    if((ret = MP4EncRelease(encoder)) != ENC_OK)
    {
        /*
           There is nothing much to do for an error here,
           just a notification message. There has to be a fault
           somewhere else that caused a failure of the
           encoder release
        */
    }

end: /* nothing else to do */
```

6.2 Optional encoder settings

```
#include "MP4EncApi.h"
#include <string.h>

MP4EncRet ret;
MP4EncInst encoder;

MP4EncCodingCtrl codingCfg;
MP4EncRateCtrl rcCfg;

/* encoder is already initialized and we have an instance */

/* Rate control */

/* get the current settings */
if((ret = MP4EncGetRateCtrl(encoder, &rcCfg)) != ENC_OK)
{
    /* Handle here the error situation */
}
else
{
    rcCfg.bitPerSecond = 2000*1000 /* 2000kbps */

    /* keep all the other parameters default */

    /* update the new settings */
    if((ret = MP4EncSetRateCtrl(encoder, &rcCfg)) != ENC_OK)
    {
        /* Handle here the error situation */
    }
}

/* Coding control */

/* get the current settings */
if((ret = MP4EncGetCodingCtrl(encoder, &codingCfg)) != ENC_OK)
{
    /* Handle here the error situation */
}
else
{
    codingCfg.vpSize = 600; /* new VP size */
    codingCfg.insHEC = 1; /* enable HEC */
    codingCfg.insGOB = 0; /* disable GOB */

    /* keep all the other parameters default */

    /* update the new settings */
    if((ret = MP4EncSetCodingCtrl(encoder, &codingCfg)) != ENC_OK)
    {
        /* Handle here the error situation */
    }
}
```

```
/* Set some user data */
{
    MP4EncUsrDataType type = MPEG4_VOS_USER_DATA;
    const char * data = "TEST USER DATA";
    u32 length = (u32)strlen(data);

    if((ret = MP4EncSetUsrData(encoder, data, length, type))) != ENC_OK)
    {
        /* Handle here the error situation */
    }
}
```

6.3 Encode with video stabilization

```
#include "MP4EncApi.h"
#include "basetype.h"
#include <stdlib.h>

MP4EncRet ret;
MP4EncInst encoder;
MP4EncCfg cfg;
MP4EncPreProcessingCfg preProcCfg;
MP4EncIn encIn;
MP4EncOut encOut;

i32 last = 0;

u32 picture_size;
u32 pict_bus_address = 0;
u32 pict_bus_address_stab = 0;

u32 outbuf_size;
u32 outbuf_bus_address = 0;
u32 *outbuf_virt_address = NULL;

/* Step 1: Initialize an encoder instance */

/* 30 fps frame rate */
cfg.frameRateDenom = 1;
cfg.frameRateNum   = 30; /* this is the time resolution also */

/* VGA resolution */
cfg.width        = 640;
cfg.height       = 480;

/* Output buffer size */
outbuf_size = cfg.width*cfg.height;

/* Stream type */
cfg.strmType     = MPEG4_PLAIN_STRM; /* setup the stream type */

/* and finally the profile&level of the stream */
cfg.profileAndLevel = MPEG4_SIMPLE_PROFILE_LEVEL_5;

if((ret = MP4EncInit(&cfg, &encoder)) != ENC_OK)
{
    /* Handle here the error situation */
    goto end;
}
```

```
/* Step 2: Setup video stabilization */
if((ret = MP4EncGetPreProcessing(encoder, &preProcCfg)) != ENC_OK)
{
    /* Handle here the error situation */
    goto close;
}
preProcCfg.videoStabilization = 1;
preProcCfg.yuvType = ENC_YUV420_PLANAR;
preProcCfg.rotation = ENC_ROTATE_0;


/* extra 16 pixels for stabilization */
preProcCfg.origWidth = cfg.width + 16;
preProcCfg.origHeight = cfg.height + 16;

if((ret = MP4EncSetPreProcessing(encoder, &preProcCfg)) != ENC_OK)
{
    /* Handle here the error situation */
    goto close;
}

/* picture buffer size YUV 420 */
picture_size = (preProcCfg.origWidth * preProcCfg.origHeight * 3) / 2;

/* Step 3: Allocate linear memory resources. This implementation
   is OS specific and not part of this example. */
AllocDMAMemory(&outbuf_virt_address, &output_bus_address, &outbuf_size);
AllocDMAMemory(NULL, &pict_bus_address, &picture_size);
AllocDMAMemory(NULL, &pict_bus_address_stab, & picture_size);

encIn.pOutBuf    = outbuf_virt_address;
encIn.busOutBuf  = output_bus_address;
encIn.outBufSize = outbuf_size; /* bytes */

/* Step 4: Start the stream */

ret = MP4EncStrmStart(encoder, &encIn, &encOut);
if(ret != ENC_OK)
{
    /* Handle here the error situation */
    goto close;
}
else
{
    /* Save the generated stream. This implementation is OS
       Specific and not part of this example. */
    SaveStream(encIn.pOutBuf, encOut.streamSize);
}

/* Step 5: Encode the video frames */

/* Read first frame and get the bus address. This implementation
   is OS specific and not part of this example. */
ReadYuv420PlanarFrame(pict_bus_address);

encIn.timeIncrement = 0; /* start time is 0 */
encIn.codingType = INTRA_VOP; /* First frame must be intra */
```

```
while( !last )
{

   /* Read next frame to be stabilized */
    ReadYuv420PlanarFrame(pict_bus_address_stab);

    /* one linear buffer for the input image in planar YCbCr 4:2:0 format */
    encIn.busLuma = pict_bus_address;
    encIn.busChromaU = encIn.busLuma + cfg.width*cfg.height;
    encIn.busChromaV = encIn.busChromaU + (cfg.width/2) * (cfg.height/2);

    /* stabilization input, only luminance data used */
    encIn.busLumaStab = pict_bus_address_stab;

    /* encode and stabilize in one run */
    ret = MP4EncStrmEncode(encoder, &encIn, &encOut);

    switch (ret)
    {
        case ENC_VOP_READY:
            SaveStream(encIn.pOutBuf, encOut.streamSize);
            break;

        default:
            /* stop for any error */
            last = 1;
     break;
    }

    /* stabilization result if needed */
    MP4EncGetPreProcessing(encoder, &preProcCfg)

    /* stabilized image will be encoded next (swap picture buffers) */
    {
        u32 tmp = pict_bus_address;
        pict_bus_address = pict_bus_address_stab;
        pict_bus_address_stab = tmp;
    }

    encIn.timeIncrement = 1; /* time units between frames */
    encIn.codingType = PREDICTED_VOP; /* use prediction during encoding */

}   /* End of main encoding loop */

/* Step 6: End stream */
ret = MP4EncStrmEnd(encoder, &encIn, &encOut);
if(ret != ENC_OK)
{
    /* Handle here the error situation */
    goto close;
}

SaveStream(encIn.pOutBuf, encOut.streamSize);

close:

/* Free all resources */
FreeDMAMemory(outbuf_virt_address, output_bus_address, outbuf_size);
FreeDMAMemory(NULL, pict_bus_address, picture_size);
FreeDMAMemory(NULL, pict_bus_address_stab, picture_size);
```

```
/* Last Step: Release the encoder instance */
if((ret = MP4EncRelease(encoder)) != ENC_OK)
{
    /*
       There is nothing much to do for an error here,
       just a notification message. There has to be a fault
       somewhere else that caused a failure of the
       encoder release.
    */
}

end: /* nothing else to do */
```

References

[1] ISO/IEC 14496-2, Third edition 2004-06-01, Information technology – Coding of audio-visual objects – Part 2: Visual

[2] ITU-T Recommendation H.263 (02/98), Video coding for low bit rate communication

VISIBLY BETTER

Headquarters:
Hantro Products Oy
Kiviharjunlenkki 1, FI-90220 Oulu, Finland
Tel: +358 207 425100, Fax: +358 207 425299

Hantro Finland	Hantro Germany	Hantro USA	Hantro Japan	Hantro Taiwan	Hantro Korea
Lars Sonckin kaari 14	Ismaninger Str. 17-19	1762 Technology Drive	Yurakucho Building 11F	6F, No. 331	#518 ho, Dongburoot, 16-2
FI-02600	81675	Suite 202. San Jose	1-10-1, Yurakucho	Fu-Hsing N. Rd.	Sunaedong, Bundanggu,
Espoo	Munich	CA 95110,	Chiyoda-ku, Tokyo	Taipei	Seongnamsi, Kyunggido,
Finland	Germany	USA	100-0006, Japan	Taiwan	463-825 Korea
Tel: +358 207 425100	Tel: +49 89 4130 0645	Tel: +1 408 451 9170	Tel: +81 3 5219 3638	Tel: +886 2 2717 2092	Tel: +82 31 718 2506
Fax: +358 207 425298	Fax: +49 89 4130 0663	Fax: +1 408 451 9672	Fax: +81 3 5219 3639	Fax: +886 2 2717 2097	Fax: +82 31 718 2505

Email: sales@hantro.com
WWW.HANTRO.COM