ARC® H.264 Encoder

ARC Video Subsystem Family C Application Program Interface (API)

Reference

5883-006

ARC® H.264 Encoder Reference

ARC® International

European Headquarters
ARC International,
Verulam Point,
Station Way,
St Albans, Herts, AL1 5HE, UK
Tel. +44 (0) 1727 891400
Fax. +44 (0) 1727 891402

North American Headquarters
3590 N. First Street, Suite 200
San Jose, CA 95134 USA
Tel. +1 408.437.3400
Fax +1 408.437.3401

www.arc.com

Confidential Information

Notice

U.S. Government Restricted Rights Legend

Trademark Acknowledgments

5883-006 April 2008

Contents

List of Tables

Chapter 1 — Introduction

Overview

This document describes the software interface for the ARC® H264 Encoder for the ARC Video Subsystem Family. The following topics are covered:

- Data Types
- Exported Functions
- Imported Function Requirements

The descriptions of data structure definitions and function call are intended for use by developers needing to interface to the encoder; the descriptions of the contents and usage of the data structures are intended for use by anyone wishing to take control of the encoding parameters.

Further Reading

The following are recommended to supplement the information in this document:

- *ARC Video Subsystem Family Encoders API Programmer's Guide*
- ITU-T H.264 | ISO/IEC 14496-10 *Information technology: Coding of audio-visual objects: Part 10: Advanced Video Coding*

Key Encoder Features

- Support for Baseline Profile
- ¼ MV accuracy, +/- 15.75 search range, 6-step motion estimation search
- Inter 16x16, 16x8, 8x16 and 8x8 macroblock prediction
- 16x16 luma intra mode prediction (modes 0-2)
- 8x8 chroma intra mode prediction (modes 0-2)
- VBR/CBR rate control methods
- Switchable deblocking filter
- Dynamic encoding algorithm to balance encoding quality with processor bandwidth

Dynamic Encoding

The ARC H.264 Encoder incorporates procedures to optimize the video compression performance while minimizing power requirements.

Video encoding algorithms comprise a range of different coding techniques, each of which takes a differing amount of computation to use. To a large extent, the nature of the incoming, raw video stream dictates the amount of processing required. As this nature is not known to the encoder a priori, it has been designed to adapt the processing to the response of the encoder to the input. For example, a static camera sequence with a small proportion of the image changing with time (such as a one-to-one video conference), is much easier to encode than an action-packed feature film, as the former is much more predictable than the latter.

The Dynamic Encoding Process (DEP) interrogates the encoding process at regular intervals to understand how the encoding process is progressing against a set time budget. If the process is going well and ahead of schedule in a frame, then the DEP will switch on more coding modes than were previously on, with the consequence that the compressed data will be better quality, whereas if there is a difficult sequence to encode, the process will gradually constrain the coding possibilities in order to catch up – at the expense of quality.

The elements that are switchable include the following:

- Macroblock coding modes (e.g., in H.264 macroblock types)
- Coding sub-modes (as used in H.264 Intra and Inter macroblocks)
- Coding decision thresholds (enabling early decisions that avoid other tests)
- Motion estimation parameters (for example motion vector range and precision)

These elements are ranked inside the encoder in order of desirability (in terms of minimizing the effect on the quality of the output compressed video data) and are gracefully switched on or off in turn.

This process is performed on a per frame basis, and such that the time taken for each frame attempts to be within the budget calculated for the frame based on the input parameters and the target Megahertz operation of the processor, which is input via the *quality_level* setting. This value is only the target and it should be noted that, depending on the complexity of the input video and the operational parameters selected, the processor speed target may not be attainable.

Similarly, should the processor speed target be greatly in excess of that needed for the sequence and parameters input, the dynamic encoding mechanism may not be utilized. Dynamic encoding is performed on a per frame basis and in that instance the period of timer interrupt may be greater than the time taken to process the frame.

H.264SetupParams

Structure Definition

```
typedef struct {
  IS_ARGTYPE;
  u32          frame_width;
  u32          frame_height;
  bool         fixed_quantiser;
  u32          fixed_quantiser_value;
  u32          target_bitrate;
  u32          intra_picture_frequency;
  u32          disable_deblocking_filter_idc;
  bool         code_all_pictures;
  u32          target_picture_period;
  u32          input_picture_period;
  u32          intra_refresh_rate;
  bool         real_time_rate_control;
  s8           chroma_qp_index_offset;
  bool         constrained_intra_pred_flg;
  u32          num_slice_grps_P_pictures;
  u32          maximum_packet_size;
  u32          reaction_multiplier;
  u32          quality_level;
  u32          dynamic_control;

  h264_vui_parameters* vui_parameters;
} H264SetupParams;
```

Type Code

The type code is ENC_ARGTYPE_H264SETUP.

Description

This structure contains the parameters used to initialize an encoding session.

Table 1 H.264 Encoding Setup Parameters

Parameter	Description
frame_width	Number of pixels in the luminance in the input YUV file. The value should be a multiple of 16.
frame_height	Number of luminance lines in the input YUV data. The value should be a multiple of 16
fixed_quantiser	Binary value, with 1 indicating fixed quantizer sand 0 indicating a constant bit-rate.

Parameter	Description
fixed_quantiser_value	If *fixed_quantiser* is 1, the value of the quantizer to use. Disregarded if *fixed_quantiser* is `0`. Legal values for the quantizer range from 12 to 42.
target_bitrate	Required bitrate for the encoding in bits per second.
intra_refresh_rate	Number of macroblocks to be intra refreshed in each frame. Normally `0`, but positive numbers might be beneficial in error-prone environments.
intra_picture_frequency	Frequency at which intra frames are transmitted. `0`: only the first frame is intra. `1`: every frame is intra. `12`: every 12th frame is intra. Etc. Video-telephony applications normally use `0`.
disable_deblocking_filter_idc	This parameter controls the de-blocking process. It can take 3 values: *0*: Normal de-blocking *1*: No de-blocking *2*: De-blocking at slice boundaries Usually set to `0` (de-blocking is performed).
code_all_pictures	Binary parameter to indicates whether all input pictures should be coded. This is beneficial for off-line coding and for codec-comparison tests. In real-time, network-constrained encoding this option is usually switched off as frame skips might be needed to cope with the content and network conditions.
target_picture_period	Required output picture rate (number of units assigned to one frame). This value *must* be a multiple of the input picture period, and should never be less than the input picture period. It is used in calculating effective bitrate. Specify in units of 1/90000 of a second.
input_picture_period	Time units that are nominally in the input YUV file (number of units assigned to one frame). Specify in units of 1/90000 of a second. A typical value is 3000, representing 30-Hz input. If the target picture period is less than the input picture period, the API skips frames as needed.
real_time_rate_control	In case of rate control, indicates whether the encoder is running in real-time (true) or non-real-time (false). In case of real time, rate control is performed using actual measurements of buffer fill, whereas in non-real-time mode, the encoder simulates a real output bit buffer by simulating time by counting source pictures and knowing the source picture rate.

Parameter	Description
constrained_intra_pred_flag	This parameter indicates whether to use constrained intra mode: true indicates intra prediction from inter coded pixels is not performed. This parameter corresponds to the parameter of the same name in the H.264 standard. It is used to increase error resilience in the bit stream, but using it means less efficient encoding of intra data.
num_slice_grps_P_pictures	Number of slice groups for P pictures. Currently unsupported.
maximum_packet_size	Byte value representing the maximum number of bytes allowed in a packet for the network being used. For example, Ethernet is 1500. If this parameter is not set, a value of 40,000 is used to avoid using different slices for single frames.
reaction_multiplier	Period over which the bit-rate requested should be applied. The units are in $1/30^{th}$ of a second. The default value is 30. For applications where network considerations are not relevant, this parameter can be set to a larger value.
quality_level	If *dynamic_control* is off , this takes a value of between `0` and `10`, inclusive. If no value is specified, the default is `10`, i.e., all coding tools are switched on, regardless of power consumption required. A value of `0` means that every effort is made to reduce computation by switching all optional modes off. This results in lower-quality compressed output. A typical value of this parameter is `5`.
	If *dynamic_control* is enabled, this field is interpreted differently and has a different range. In this case, this field is taken to indicate the number of MegaHerz that should be used to encode the data. E.g. 200 would indicate that it should be attempted to be encoded using 200 MHz processor speed, although it should be noted that the other parameters selected and/or the nature of the input video may mean that this value is not attained. When *dynamic_control* is enabled a value of zero for this parameter sets a default of 200 MHz.
dynamic_control	This parameter takes three possible values: 0, 1 and 2.
	If zero is selected then the dynamic control feature is not applied and the coding tools that are selected by the *quality_level* are applied to the entire sequence without change.
	If the value 1 is selected, then this switches on the dynamic control feature which will try to perform the encoding within the power settings selected by the *quality_level* parameter. In this case only the processing time for the picture is included in this calculation, the output or storing of compressed data is not included.
	If the value 2 is selected, then this also switches on the dynamic control algorithm, but now the time spent saving or outputting of the compressed stream is included in the calculation.

Parameter	Description
chroma_qp_index_offset	This parameter has a range of (-12, 12) and is an offset to the luma quantizer to be used for chroma. A lower value means that chroma is coded more accurately (using more bits), and a higher value means chroma is coded less accurately (using fewer bits). The use of this parameter is as described in the H.264 standard.

h264_vui_parameters

Structure Definition

```
typedef struct h264_vui_parameters_tag
{
  // The following parameters should be set by the application
  bool aspect_ratio_info_present_flag;
  int aspect_ratio_idc;
  int sar_width;
  int sar_height;
  bool overscan_info_present_flag;
  bool overscan_appropriate_flag;
  bool video_signal_type_present_flag;
  int video_format;
  bool video_full_range_flag;
  bool colour_description_present_flag;
  int colour_primaries;
  int transfer_characteristics;
  int matrix_coefficients;
  bool chroma_loc_info_present_flag;
  int chroma_sample_loc_type_top_field;
  int chroma_sample_loc_type_bottom_field;
  bool timing_info_present_flag;
  unsigned int num_units_in_tick;
  unsigned int time_scale;
  bool fixed_frame_rate_flag;

  // The following parameters are set by the encoder
  bool nal_hrd_parameters_present_flag;
  int nal_hrd_cpb_cnt_minus1;
  int nal_hrd_bit_rate_scale;
  int nal_hrd_cpb_size_scale;
  unsigned int nal_hrd_bit_rate_value_minus1[32];
  unsigned int nal_hrd_cpb_size_value_minus1[32];
  bool nal_hrd_cbr_flag[32];
  int nal_hrd_initial_cpb_removal_delay_length_minus1;
  int nal_hrd_cpb_removal_delay_length_minus1;
  int nal_hrd_dpb_output_delay_length_minus1;
  int nal_hrd_time_offset_length;
  bool vcl_hrd_parameters_present_flag;
  int vcl_hrd_cpb_cnt_minus1;
  int vcl_hrd_bit_rate_scale;
  int vcl_hrd_cpb_size_scale;
  unsigned int vcl_hrd_bit_rate_value_minus1[32];
```

```
  unsigned int vcl_hrd_cpb_size_value_minus1[32];
  bool vcl_hrd_cbr_flag[32];
  int vcl_hrd_initial_cpb_removal_delay_length_minus1;
  int vcl_hrd_cpb_removal_delay_length_minus1;
  int vcl_hrd_dpb_output_delay_length_minus1;
  int vcl_hrd_time_offset_length;
  bool low_delay_hrd_flag;
  bool pic_struct_present_flag;
  bool bitstream_restriction_flag;
  bool motion_vectors_over_pic_boundaries_flag;
  unsigned int max_bytes_per_pic_denom;
  unsigned int max_bits_per_mb_denom;
  int log2_max_mv_length_horizontal;
  int log2_max_mv_length_vertical;
  int num_reorder_frames;
  unsigned int max_dec_frame_buffering;
} h264_vui_parameters;
```

Type Code

This is not a first-class API type. It is passed through the API only as a member of another type, and so has no code.

Description

The VUI parameters are as described in Annex E of the H.264 Specification. These parameters represent the visual user interface, and provide details of the final display of the sequence for precise reproduction of the original. They have no bearing on the encoding process. The encoding application may set the parameters shown in VUI Parameters Set by Encoding Application.

Table 2 VUI Parameters Set by Encoding Application

Parameter	Description
aspect_ratio_info_present_flag	Indicates whether aspect-ratio information is to be included in the encoded bitstream, with the following parameters.
aspect_ratio_idc	Indicates the value of the sample aspect ratio of the luma samples as in Table E.1 of the standard. *1*=square *2*=PAL(4:3) *3*=NTSC(4:3)
sar_width	Horizontal size of the sample aspect ratio (in arbitrary units). Used when `aspect_ratio_idc=255`.
sar_height	Vertical size of the sample aspect ratio (in the same arbitrary units as `sar_width`). Used when `aspect_ratio_idc=255`.
overscan_info_present_flag	Indicates whether `overscan_appropriate_flag` is to be included in the encoded bitstream.
overscan_appropriate_flag	Indicates whether the decoded pictures should be displayed with overscan.

Parameter	Description
video_signal_type_present_flag	Indicates whether the next three parameters are to be included in the encoded bitstream.
video_format	Format of pictures before encoding as in Table E.2 of the standard. *0*=Component *1*=PAL *2*=NTSC
video_full_range_flag	Black level and range of the luma and chroma signals.
colour_description_present_flag	Indicates whether the next three parameters are to be included in the encoded bitstream.
colour_primaries	Chromaticity coordinates of the source primaries as specified in Table E.3 of the standard.
transfer_characteristics	Opto-electronic transfer characteristic of the source picture as specified in Table E.4 of the standard.
matrix_coefficients	Describes the matrix coefficients used in deriving luma and chroma signals from the green, blue, and red primaries, as specified in Table E.5 of the standard.
chroma_loc_info_present_flag	Indicates whether the next two parameters are to be included in the encoded bitstream.
chroma_sample_loc_type_top_field	Indicates the location of chroma samples for the top field as shown in Figure E.1 of the standard.
chroma_sample_loc_type_bottom_field	Indicates the location of chroma samples for the bottom field as shown in Figure E.1 of the standard.
timing_info_present_flag	Indicates whether the next three parameters are to be included in the encoded bitstream.
int *num_units_in_tick*	Number of time units of a clock operating at the frequency `time_scale` Hz that corresponds to one increment (called a clock tick) of a clock-tick counter. A clock tick is the minimum interval of time that can be represented in the coded data (i.e., a field or frame period, depending on which encoding tools are used).
time_scale	Number of time units in one second. If `time_scale` is `30000`, `num_units_in_tick` can be set as follows to indicate the given frequencies: `1200`=25Hz, `1001`=29.97Hz, `1250`=24Hz.
fixed_frame_rate_flag	Indicates whether the temporal distance between the output times of any two consecutive pictures in output order is constrained.

The remaining parameters are set by the encoder.

Table 3 Encoding Parameters Set by Encoder

Parameter	Description
nal_hrd_parameters_present_flag	Always false
nal_hrd_cpb_cnt_minus1	Unused
nal_hrd_bit_rate_scale	Unused
nal_hrd_cpb_size_scale	Unused
nal_hrd_bit_rate_value_minus1	Unused
nal_hrd_cpb_size_value_minus1	Unused
nal_hrd_cbr_flag	Unused
nal_hrd_initial_cpb_removal_delay_length_minus1	Unused
nal_hrd_cpb_removal_delay_length_minus1	Unused
nal_hrd_dpb_output_delay_length_minus1	Unused
nal_hrd_time_offset_length	Unused
vcl_hrd_parameters_present_flag	Always false.
vcl_hrd_cpb_cnt_minus1	Unused
vcl_hrd_bit_rate_scale	Unused
vcl_hrd_cpb_size_scale	Unused
vcl_hrd_bit_rate_value_minus1	Unused
vcl_hrd_cpb_size_value_minus1	Unused
vcl_hrd_cbr_flag	Unused
vcl_hrd_initial_cpb_removal_delay_length_minus1	Unused
vcl_hrd_cpb_removal_delay_length_minus1	Unused
vcl_hrd_dpb_output_delay_length_minus1	Unused
vcl_hrd_time_offset_length	Unused
low_delay_hrd_flag	True if `code_all_pictures` in the setup parameters is false; false otherwise.
pic_struct_present_flag	Always false
bitstream_restriction_flag	Always true
motion_vectors_over_pic_boundaries_flag	Always true
max_bytes_per_pic_denom	Always 0
max_bits_per_mb_denom	Always 0
log2_max_mv_length_horizontal	Always 13
log2_max_mv_length_vertical	Set according to level.
num_reorder_frames	Always 0
max_dec_frame_buffering	Set to the number of reference frames according to the current SPS.

H.264EncodingParams

Structure Definition

```
typedef struct {
  IS_ARGTYPE;
  h264_encoder_results* results;
  RawFrame*    decoded_frames;
  u32          max_decoded_frames;
  bool         statistics_valid;
  H264EncoderStats* cumulative_stats;
} H264EncodingParams;
```

Type Code

This type code is ENC_ARGTYPE_H264ENCPARAMS.

Description

Table 4 H.264 Encoding Parameters

Parameter	Description
results	Pointer to the encoding result data structure native to the encoder. See below for details on this structure.
decoded_frames	A null array of frames of reconstructed data. This array is produced only if the encoder and API have been built with the DEBUG_OUTPUT preprocessor symbol defined, and the output has been enabled by the **ENC_FN_SET_OUTPUTDEBUG()** control function. By default, reconstructed output is disabled, and this array is not altered by a call to **h264_encode()**.
max_decoded_frames	The length of the decoded_frames array
statistics_valid	Set to true by the API if statistics collection is enabled and the encoder has generated a valid set of statistics; false otherwise.
cumulative_stats	A pointer to a set of running statistics built up during an encoding session; null if statistics generation is disabled.

h264_encoder_results

Structure Definition

```
typedef struct      h264_encoder_results_tag
{
       int bit_count;
       bool source_picture_finally_encoded;
       bool field_coding;
       int intra_4x4_count;
       int intra_16x16_count;
       int inter_16x16_count;
       int inter_16x8_count;
       int inter_8x16_count;
       int inter_8x8_count;
```

```
        int skipped_count;
        int forced_skip_count;
        int skip_8x8_count;
        int cycles_in_frame;
        int direct_count;
        int buffer_level;  // Only set for non-real-time encoding

        unsigned int timestamp;
        #ifdef STATISTICS
        double average_quant;
        #endif // STATISTICS
        int number_of_output_pictures;

        unsigned char  *    pFrameY;
        unsigned char  *    pFrameU;
        unsigned char  *    pFrameV;
        void                  * pEncodedPicture;
        void                  * pSourcePicture;
        void                  * pOutputPicture[H264_ENCODER_MAX_OUTPUT_PICTURES];

}       h264_encoder_results;
```

Type Code

This is not a first-class API type - it is passed through the API only as a member of another type, and so has no code.

Description

Table 5 H.264 Encoding Results

Parameter	Description
bit_count	Number of bits produced by encoding the picture.
sp_picture	True if the picture was encoded with SP slices, false otherwise.
source_picture_stored_for_later	When true indicates that the source frame will be encoded later (i.e., out of source order), but that this can not happen until the next (future) key frame is encoded. The source frame is stored for later encoding and the function returns immediately.
source_picture_finally_encoded	When true, indicates that the encoding of a stored (out of source order) frame is completed.
intra_4x4_count	Number of macro blocks of the picture encoded as intra 4x4 macroblocks.
intra_16x16_count	Number of macroblocks of the picture encoded as intra 16x16 macroblocks.
inter_16x16_count	Number of macroblocks of the picture encoded as inter 16x16 macroblocks.
inter_16x8_count	Number of macroblocks of the picture encoded as inter 16x8 macroblocks.
inter_8x16_count	Number of macroblocks of the picture encoded as inter 8x16 macroblocks.

Parameter	Description
inter_8x8_count	Number of macroblocks of the picture encoded as inter 8x8 macroblocks.
skipped_count	Number of macroblocks of the picture encoded as skipped macroblocks.
direct_count;	Number of macroblocks of the picture encoded as direct macroblocks (not counting those that were skipped).
buffer_level	In the case of non-real-time encoding, this parameter indicates the fullness in bits of the simulated bit buffer. Not used otherwise.
timestamp	Timestamp of the source picture.
average_quant	Average quantizer index used to encode the picture. Only included if statistics are enabled.
number_of_output_pictures	Number of output pictures in the array `pOutputPicture`.
pFrameY	Pointer to the Y data of the encoded source picture.
pFrameU	Pointer to the U data of the encoded source picture.
pFrameV	Pointer to the V data of the encoded source picture.
pEncodedPicture	Pointer to the encoded source picture.
pSourcePicture	Pointer to the source picture.
pOutputPicture	An array of pointers to the output pictures. The output might be a single picture equal to the encoded source picture, but not in the case of encoding pictures out of source order. This parameter is a pointer to pictures in output (display) order rather than coding order. These pictures can be used to show the progress of encoding as encoded pictures on a monitor. (Note that this information is also available in the **decoded_frames** member of **H264EncodingParam**s).
control_list	Pointer to the internal control list used to encode the picture. You can use this list to generate switching-stream SP pictures. The structure of the control list is not given in this document.

H264EncoderStats

Structure Definition

```
typedef struct {
  IS_ARGTYPE;
  u32          tot_elapsed_ticks;
  u32          num_pics_encoded;
  u32          total_bits;
  double       y_snr;
  double       u_snr;
  double       v_snr;
  u32          num_MB_force_skipped;
  u32          num_8x8_skipped;
  char*        stats_buf;
```

```
  u32          stats_buf_len;
} H264EncoderStats;
```

Type Code

This type code is ENC_ARGTYPE_H264STATS.

Description

Table 6 H.264 Encoding Statistics

Parameter	Description
tot_elapsed_ticks	The running total of system ticks spent in the encoder.
num_pics_encoded	The running total of frames encoded.
total_bits	The running total of bits output.
y_snr	The running total of Y component SNR, for later averaging (to get the average SNR, divide this by num_frames_encoded).
u_snr	The running total of U component SNR, for later averaging.
v_snr	The running total of V component SNR, for later averaging.
num_MB_force_skipped	The number of macro blocks encoded as skipped macroblocks.
num_8x8_skipped	The number of 8x8 macro blocks encoded as skipped macroblocks.
stats_buf	A buffer used by the **ENC_FN_H264_CUMULATIVESTATS()** control function to contain a formatted string of statistics.
stats_buf_len	The length of stats_buf.

CAUTION Generation of statistics severely impairs run-time performance. Its use is discouraged other than for test purposes.

Chapter 3 — Exported Functions

The ARC H.264 Encoder API implements a set of exported functions compliant with the ARC Video Subsystem Family Encoders API definition. Refer to the *ARC Video Subsystem Family Encoders API Programmer's Guide* for full details about the base API definition. In the following discussion, the functions are introduced using the generic names given in the base API.

init()

Initializes the encoder prior to processing data.

Synopsis

```
s32 h264_enc_init(u32              encoder_handle,
                  u32              coding_type,
                  StreamDescriptor *stream_desc,
                  EncArgument      *extra_data);
```

Parameters

encoder_handle	Instance ID to be used for this session.
coding_type	Coding type; must be `ENC_TYPE_H264`.
stream_desc	Details of the stream being encoded.
extra_data	Encoder setup parameters; must of type `H264SetupParams`.

Example

```
H264SetupParams setupParams; RawFrame inputFrame; StreamDescriptor strDesc; u32
handle = 1;

// Initialize setup parameters
SETUP_ENCARGUMENT(setupParams, ENC_ARGTYPE_H264SETUP, sizeof(H264SetupParams)));
setupParams.vui_parameters = malloc(sizeof(h264_vui_parameters));
get_h264_parameters_from_whereever(&setupParams);

// Set up stream descriptor
strDesc.width = setupParams.frame_width; strDesc.height =
setupParams.frame_height; strDesc.frames = 0; strDesc.current_frame = 0;
strDesc.interlaced = 0;

// Initialize encoder
if (h264_enc_init(handle,
                  ENC_TYPE_H264
                  &strDesc,
                  &(ENC_ARG(setupParams))) != ENC_ERR_NONE)
{
    fprintf(stderr,"Error initialising encoder\n");
    exit(-1);
}
```

uninit()

Tears down the current encoding session.

Synopsis

```
s32 h264_enc_uninit(u32 encoder_handle);
```

Parameters

encoder_handle	Encoder instance ID.

Example

```
u32 handle;

if (h264_enc_uninit(handle) != ENC_ERR_NONE)
{
	fprintf(stderr,"Error uninitialising encoder\n");
}
```

encode()

Encode a frame of data, and output one or more packets of encoded data.

Synopsis

```
s32 h264_enc_encode(u32         encoder_handle,
                    RawFrame    **in_frames,
                    u32         *num_frames,
                    EncPacket   **out_packets,
                    u32         *num_packets,
                    u32         *status,
                    EncArgument *extra_data);
```

Parameters

encoder_handle	Encoder instance ID
in_frames	An array of frames of YUV 4:2:0 data for encoding. May contain zero elements if no further input is available but output must still be produced. The function consumes only the first frame of data in the array. If necessary, frame skipping is be performed by the API.
num_frames	Points to a variable which, on call, contains the number of input frames submitted in *in_frames*, and on return, contains the number of frames consumed.
out_packets	A pointer to a data packet array populated in the correct output order by the encoder. We recommend this array be at least 6 elements in size.
num_packets	Points to a variable which, on call, contains the number of packets that can be fit into the data packet array `out_packets`, and on return, contains the number of packets actually stored.
status	Unused
extra_data	Encoding parameters; must be of type `H264EncodingParams`.

Example

```
#define MAX_OUTPUT_PACKETS 6

StreamDescriptor strDesc;
H264EncodingParams encParams;
RawFrame inputFrame;
RawFrame* inputArray[2];
EncPacket* outputArray[MAX_OUTPUT_PACKETS];
u32 frames;

frames = 0;
SETUP_ENCARGUMENT(encParams,
                  ENC_ARGTYPE_H264ENCPARAMS,
                  sizeof(H264EncodingParams)))

while (more_to_encode)
{
    u32 result;
    u32 status = 0;
    u32 numin;
    u32 numout;
    int i;
    int size = (setupParams.frame_width * setupParams.frame_height * 3)/2;

    numin = get_input_frame(&inputFrame,size);

    if (numin == 0)
    {
       if (verbose) fprintf(stderr,"End of file\n");
       break;
    }

    inputFrame.info.frame_num = frames++;

    inputArray[0] = &inputFrame;
    inputArray[1] = 0;
    numin = 1;

    do
    {
         numout =  MAX_OUTPUT_PACKETS;
         if ((result = h264_enc_encode (handle,
                                        inputArray,
                                        &numin,
                                        outputArray,
                                        &numout,
                                        &status,
                                        &(ENC_ARG(encParams)))
                     !=  ENC_ERR_NONE)
         {
             if (result != ENC_ERR_OUTOFOPS)
             {
                 if (result == ENC_ERR_EOF)
```

```
                    more_to_encode = 0;
                else
                {
                    fprintf(stderr,"Fatal error during encoding\n");
                    exit(-1);
                }
            }
        }


        for (i = 0; i < numout; i++)
        {
            // Dispose of the data
            output_encoded_data(outputArray[i]);
            // Release the packet back to API
            outputArray[i].unused = true;
        }
    }
    while (result == ENC_ERR_OUTOFOPS);
}
```

control()

Perform a maintenance or configuration function on the encoder.

Synopsis

```
s32 h264_enc_control(u32         encoder_handle,
                     u32         function,
                     EncArgument *extra_data);
```

Parameters

encoder_handle	Encoder instance ID
function	Function request code
extra_data	Additional data specific to the function requested

Supported Function Codes

Table 7 h264_enc_control() Function Request Codes

Code	Function	Argument type
ENC_FN_H264_SETPARAMS	Change encoding parameters	H264SetupParams
ENC_FN_H264_CUMULATIVESTATS	Generate cumulative statistics	H264EncodingParams
ENC_FN_COLL_STATS	Get statistics generation state	ArgFundamental
ENC_FN_OUTPUTDEBUG	Get debug output state	ArgFundamental
ENC_FN_SET_COLL_STATS	Toggle statistics collection	ArgFundamental
ENC_FN_SET_OUTPUTDEBUG	Toggle debug out (reconstructed YUV data)	ArgFundamental
ENC_FN_H264_GETSTATSHDR	Get a formatted header line for the statistics	ArgCharstar

CAUTION Debug and statistics output severely impair run-time performance. Their use is discouraged other than for test purposes.

Example

```
SETUP_ENCARGUMENT(fund,ENC_ARGTYPE_FUNDAMENTAL,sizeof(ArgFundamental));
fund.array = malloc(sizeof(AnyFundamental));
fund.num_entries = 1;

if ((h264_enc_control(1,ENC_FN_COLL_STATS,&(ENC_ARG (fund)))) != ENC_ERR_NONE)
{
    fprintf(stderr,"Error getting statistics collection state\n");
}
printf("Statistics generation is %s\n",
       (fund.array[0].boolval == false ? "off" : "on"));
```

Chapter 4 — Imported Function Requirements

The H.264 encoder requires two imported functions to be provided:

- allocate()
- release()

allocate()

The encoder has a core requirement for three types of dedicated memory, two of which will be requested via the allocate function:

- Encoded output data is placed into a number of small buffers that are allocated on demand during encoding. The encoded data is placed into these buffers by DMA. Start code sequences are placed into separate buffers that are interleaved with the encoded data. The start codes are copied into place by the processor. The buffers should be allocated on 4-byte aligned addresses.
- Reference frame data is held internally in a pair of frame buffers requested by the encoder at start-up. These need to be capable of being used for DMA, and allocated on 8-byte aligned addresses. If output of reconstructed data is requested, pointers to these buffers will be returned over the API.

The third type of dedicated memory is used to input the YUV 4:2:0 data for encoding. This is expected to be allocated and managed by the encoding application, and passed into the encoder by the h264_encode API function.

release()

Memory obtained via **allocate()** is released as follows:

- After being written, the output data buffers are passed back to the encoding application in the `out_packets` array passed to the **API h264_encode()** function call. The application should return control of each buffer back to the API by setting the unused flag in the `EncPacket` object enclosing the buffer. After the flag is set, the API triggers the release function to free the packet.
- The reference frame buffers are released by the encoder when an encoding session is terminated.