WAVE420L HEVC Codec IP

API Reference Manual

Version 1.8.14

WAVE420L HEVC Codec IP: API Reference Manual

Version 1.8.14

Revision History

Date	Revision	Change
2021/7/28	1.8.14	Release version

Proprietary Notice

Address and Phone Number

Chips&Media
NC Tower1, 7~8th FL, 509, Teheran-ro, Gangnam-gu, 125-880
Seoul, Korea
Tel: +82-2-568-3767
Fax: +82-2-568-3768

Homepage: *http://www.chipsnmedia.com*

Table of Contents

Chapter 3. API DEFINITIONS

List of Figures

List of Tables

Preface

1. About This Document

This document is the API reference manual for WAVE420L Video Codec IP.

2. Intended audience

This document has been written for experienced software engineers who want to develop video applications by using the APIs.

3. Scope

This document introduces data types, structures, API functions of VPU which are used in our reference software we provide.

4. Typographical conventions

The following typographical conventions are used in this document:

bold	Highlights signal names within text, and interface elements such as menu names. May also be used for emphasis in descriptive lists where appropriate.
italic	Highlights cross-references in blue, file names, and citations.
`typewriter`	Denotes example source codes and dumped character or text.

5. Further reading

This section lists documents which are related to this product.

- *WAVE420L Datasheet*
- *WAVE420L Programmer's Guide*
- *WAVE420L Verification Guide*

Chapter 1
VPU API Overview

This section describes the basic architecture of video processing unit (VPU) APIs and decoder and encoder operation flow using the API functions.

1.1. Basic Architecture

The VPU API consists of three types of APIs: Control API, Encoder API, and Decoder API.

- **Control API**: API functions for general control of VPU. An initialization function, VPU_Init(), is a good example of the control API.
- **Encoder API**: API functions for encoder operation including VPU_EncOpen(), VPU_EncStartOneFrame(), etc.
- **Decoder API**: API functions for decoder operation including VPU_DecOpen(), VPU_DecStartOneFrame(), etc.

The VPU API functions are based on a frame-by-frame picture processing scheme. So in order to run the decoder or encoder, application should call the proper API function, and after completion of one picture processing, they can check the result of it.

To support multi-instance decoding/encoding, VPU API functions use a handle specifying a certain instance, and the handle for each instance will be provided when application has created a new decoder instance. If application wants to give a command to a specific instance, the corresponding handle should be used in every API function call for that instance.

1.2. Encoder Operation Flow

To encode a bitstream, application must follow the below procedure.

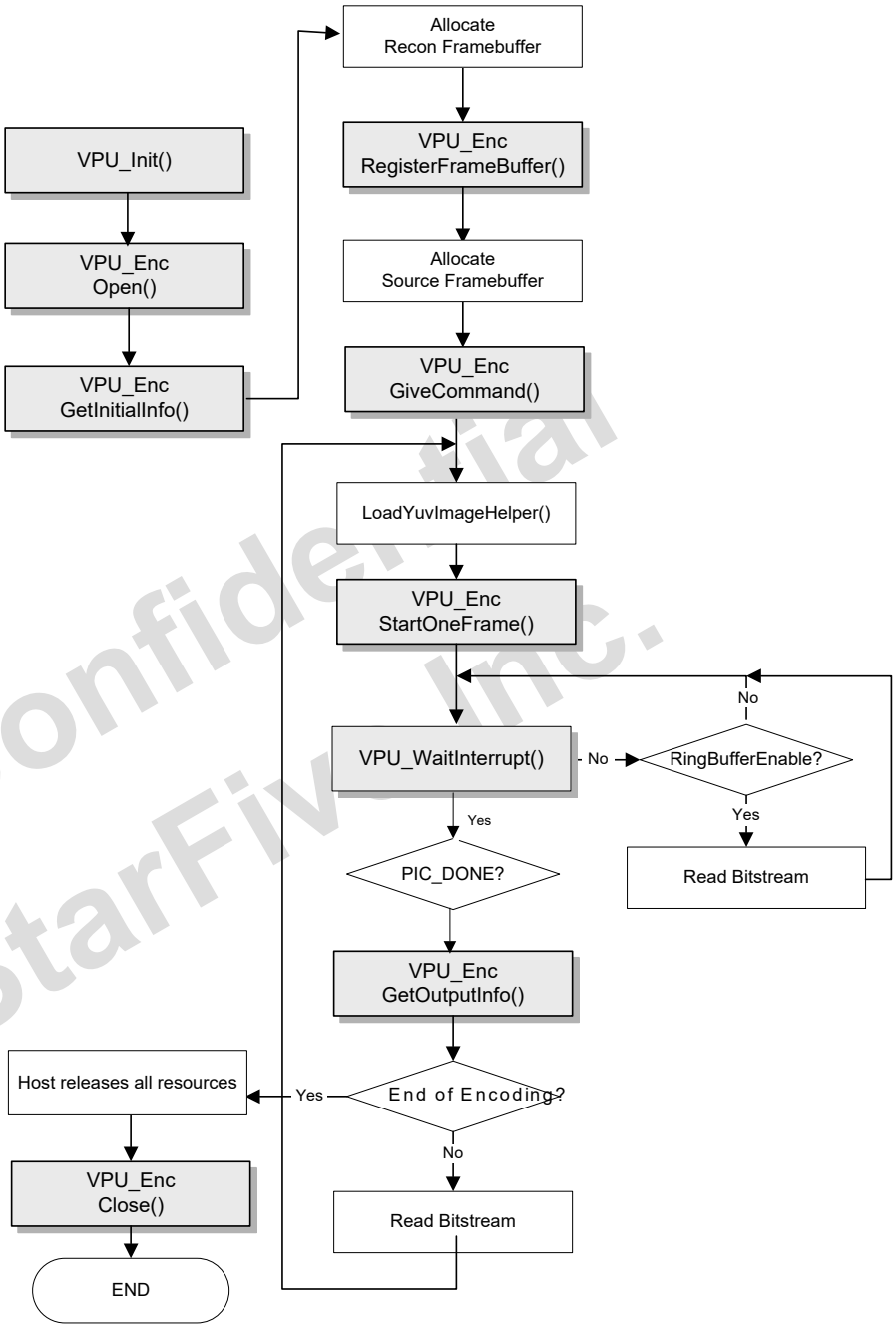


Figure 1.1. Encoder Operation Flow

1. VPU_Init() loads VPU firmware whose path is defined at BIT_CODE_FILE_PATH in config.h file and begins to boot up.

2. Application should open an encoder instance by using VPU_EncOpen().

3. Before starting picture encoding, application should know some crucial parameters for encoder operation such as required frame buffer size, etc. with VPU_EncGetInitialInfo().

4. By using the returned frame buffer requirement, application should allocate proper size of frame buffers, and inform it to the VPU : VPU_EncRegisterFrameBuffer().
5. Host allocates source frame buffer.
6. Generate high-level header syntaxes : VPU_EncGiveCommand().
7. Start picture encoder operation on a picture by picture basis: VPU_EncStartOneFrame().
8. Check the completion of picture encoder operation: VPU_WaitInterrupt().
   a. Host should read bitstream buffer if stream full occurs during encoding operation.
9. After encoding a frame, application checks the results of encoder operation: VPU_EncGetOutputInfo().
10. If there are more frames to encode, it goes back the step 7 VPU_EncStartOneFrame() and keeps proceeding. Else, it goes to the next step.
11. Terminate the sequence operation by closing the instance: VPU_EncClose().

1.3. Decoder Operation Flow

To decode a bitstream, application must follow the below procedure.

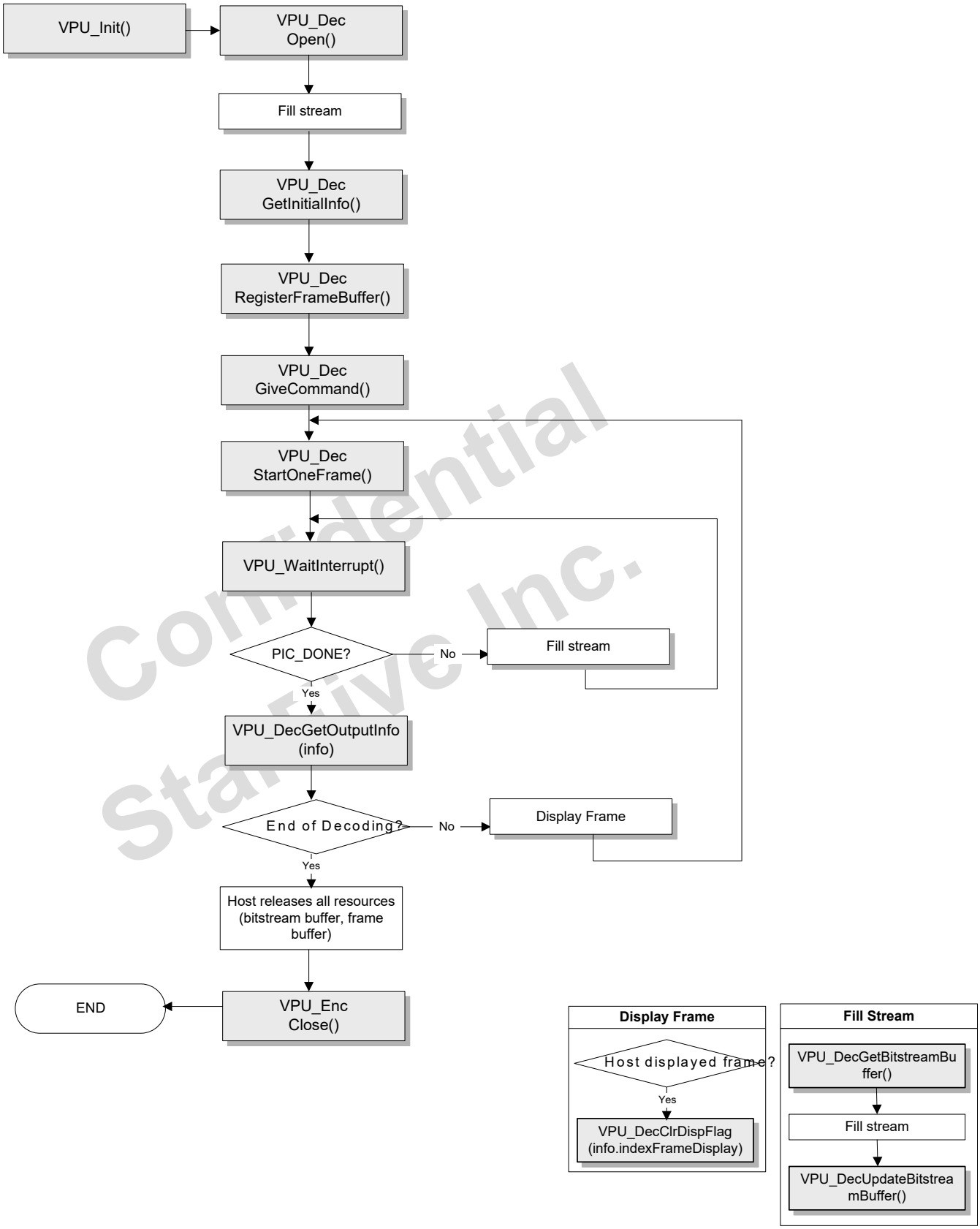


Figure 1.2. Decoder Operation Flow

1. VPU_Init() loads VPU firmware whose path is defined at BIT_CODE_FILE_PATH in config.h file and begins to boot up.

2. VPU_DecOpen() enables application to open a decoder instance.

3. Fill stream step is associated with a few function calls and bistream buffer fill-up. VPU_DecGetBitstreamBuffer() returns how much bitstream is read or written in the buffer and available buffer size. This function is used for application to know status of the buffer before filling bitstream buffer. After filling bitstream in the buffer, application should inform VPU of the amount of bits transferred into bitstream buffer with VPU_DecUpdateBitstreamBuffer().

4. VPU_DecGetInitialInfo() or a pair of VPU_DecIssueSeqInit() and VPU_DecCompleteSeqInit() decodes a header of video sequence and returns some crucial sequence parameters for decoder operations such as picture size, frame rate, required frame buffer number, etc.

5. Application should allocate an appropriate size of frame buffers and inform VPU of it by calling VPU_DecRegisterFrameBuffer().

6. Application can give a special command to VPU such as use of Secondary AXI or user data report by using VPU_DecGiveCommand().

7. VPU_DecStartOneFrame() starts picture decoder operation on a picture by picture basis.

8. VPU_WaitInterrupt( )

9. VPU_IsBusy() checks the completion of picture decoder operation.

   a. Fill more bitstream buffer if there is stream empty during decoding operation.

10. VPU_DecGetOutputInfo() returns the result of picture decoding operation.

11. Go back to 7. to go on decoding until the entire sequence has completed with decoding.

12. Display the decoded frame and call VPU_DecClrDispFlag() to clear a buffer display use flag.

13. Terminate the sequence operation by closing the instance with VPU_DecClose().

Chapter 2
DATA TYPE DEFINITIONS

This section describes the common data types used in VPU(Video Processing Unit) API functions.

2.1. Data Types

Uint8

```
typedef uint8_t         Uint8;
```

Description

This type is an 8-bit unsigned integral type, which is used for declaring pixel data.

Uint32

```
typedef uint32_t        Uint32;
```

Description

This type is a 32-bit unsigned integral type, which is used for declaring variables with wide ranges and no signs such as size of buffer.

Uint16

```
typedef uint16_t        Uint16;
```

Description

This type is a 16-bit unsigned integral type.

Int8

```
typedef int8_t          Int8;
```

Description

This type is an 8-bit signed integral type.

Int32

```
typedef int32_t         Int32;
```

Description

This type is a 32-bit signed integral type.

Int16

```
typedef int16_t         Int16;
```

Description

This type is a 16-bit signed integral type.

PhysicalAddress

```
typedef Uint32 PhysicalAddress;
```

Description

This is a type for representing physical addresses which are recognizable by VPU. In general, VPU hardware does not know about virtual address space which is set and handled by host processor. All these virtual addresses are translated into physical addresses by Memory Management Unit. All data buffer addresses such as stream buffer and frame buffer should be given to VPU as an address on physical address space.

BYTE

```
typedef unsigned char   BYTE;
```

Description

This type is an 8-bit unsigned integral type.

VpuHandle

```
typedef struct CodecInst* VpuHandle;
```

Description

This is a dedicated type for handle returned when a decoder instance or a encoder instance is opened.

DecHandle

```
typedef struct CodecInst* DecHandle;
```

Description

This is a dedicated type for decoder handle returned when a decoder instance is opened. A decoder instance can be referred to by the corresponding handle. CodecInst is a type managed internally by API. Application does not need to care about it.

Note This type is vaild for decoder only.

EncHandle

```
typedef EncInst * EncHandle;
```

Description

This is a dedicated type for encoder handle returned when an encoder instance is opened. An encoder instance can be referred by the corresponding handle. EncInst is a type managed internally by API. Application does not need to care about it.

Note This type is vaild for encoder only.

2.2. Eumerations

CodStd

```
typedef enum {
    STD_AVC,
    STD_VC1,
    STD_MPEG2,
    STD_MPEG4,
    STD_H263,
    STD_DIV3,
    STD_RV,
    STD_AVS,
    STD_THO = 9,
    STD_VP3,
    STD_VP8,
    STD_HEVC,
    STD_VP9,
    STD_AVS2,
    STD_MAX
} CodStd;
```

Description

This is an enumeration for declaring codec standard type variables. Currently, VPU supports many different video standards such as H.265/HEVC, MPEG4 SP/ASP, H.263 Profile 3, H.264/AVC BP/MP/HP, VC1 SP/MP/AP, Divx3, MPEG1, MPEG2, AVS, RealVideo 8/9/10, AVS Jizhun/Guangdian profile, Theora, VP3, VP8, and VP9.

Note MPEG-1 decoder operation is handled as a special case of MPEG2 decoder. STD_THO must be always 9.

SET_PARAM_OPTION

```
typedef enum {
    OPT_COMMON          = 0,
    OPT_CUSTOM_GOP      = 1,
    OPT_CUSTOM_HEADER   = 2,
    OPT_VUI             = 3,
    OPT_CHANGE_PARAM    = 16
} SET_PARAM_OPTION;
```

Description

This is an enumeration for declaring SET_PARAM command options. Depending on this, SET_PARAM command parameter registers have different settings.

Note This is only for WAVE encoder IP.

OPT_COMMON
SET_PARAM command option for encoding sequence

OPT_CUSTOM_GOP
SET_PARAM command option for setting custom GOP

OPT_CUSTOM_HEADER
SET_PARAM command option for setting custom VPS/SPS/PPS

OPT_VUI
SET_PARAM command option for encoding VUI

OPT_CHANGE_PARAM
SET_PARAM command option for parameters change (WAVE520 only)

C7_SET_PARAM_OPTION

```
typedef enum {
    C7_OPT_COMMON       = 16,
    C7_OPT_SET_PARAM    = 17,
    C7_OPT_PARAM_CHANGE = 18,
} C7_SET_PARAM_OPTION;
```

Description

This is an enumeration for declaring SET_PARAM command options. (CODA7Q encoder only) Depending on this, SET_PARAM command parameter registers have different settings.

DEC_PIC_HDR_OPTION

```
typedef enum {
    INIT_SEQ_NORMAL     = 0x01,
    INIT_SEQ_W_THUMBNAIL = 0x11,
} DEC_PIC_HDR_OPTION;
```

Description

This is an enumeration for declaring the operation mode of DEC_PIC_HDR command. (WAVE decoder only)

INIT_SEQ_NORMAL
It initializes some parameters (i.e. buffer mode) required for decoding sequence, performs sequence header, and returns information on the sequence.

INIT_SEQ_W_THUMBNAIL
It decodes only the first I picture of sequence to get thumbnail.

DEC_PIC_OPTION

```
typedef enum {
    DEC_PIC_NORMAL      = 0x00,
    DEC_PIC_W_THUMBNAIL = 0x10,
    SKIP_NON_IRAP       = 0x11,
    SKIP_NON_RECOVERY   = 0x12,
    SKIP_NON_REF_PIC    = 0x13,
    SKIP_TEMPORAL_LAYER = 0x14,
} DEC_PIC_OPTION;
```

Description

This is an enumeration for declaring the running option of DEC_PIC command. (WAVE decoder only)

DEC_PIC_NORMAL
It is normal mode of DEC_PIC command.

DEC_PIC_W_THUMBNAIL
It handles CRA picture as BLA picture not to use reference from the previously decoded pictures.

SKIP_NON_IRAP
It is thumbnail mode (skip non-IRAP without reference reg.)

SKIP_NON_RECOVERY
It skips to decode non-IRAP pictures.

SKIP_NON_REF_PIC
It skips to decode non-reference pictures which correspond to sub-layer non-reference picture with MAX_DEC_TEMP_ID. (The sub-layer non-reference picture is the one whose nal_unit_type equal to TRAIL_N, TSA_N, STSA_N, RADL_N, RASL_N, RSV_VCL_N10, RSV_VCL_N12, or RSV_VCL_N14.)

SKIP_TEMPORAL_LAYER
It decodes only frames whose temporal id is equal to or less than MAX_DEC_TEMP_ID.

RetCode

```
typedef enum {
    RETCODE_SUCCESS,                    /* 0  */
    RETCODE_FAILURE,
    RETCODE_INVALID_HANDLE,
    RETCODE_INVALID_PARAM,
    RETCODE_INVALID_COMMAND,
    RETCODE_ROTATOR_OUTPUT_NOT_SET,     /* 5  */
    RETCODE_ROTATOR_STRIDE_NOT_SET,
    RETCODE_FRAME_NOT_COMPLETE,
    RETCODE_INVALID_FRAME_BUFFER,
    RETCODE_INSUFFICIENT_FRAME_BUFFERS,
    RETCODE_INVALID_STRIDE,             /* 10 */
    RETCODE_WRONG_CALL_SEQUENCE,
    RETCODE_CALLED_BEFORE,
    RETCODE_NOT_INITIALIZED,
    RETCODE_USERDATA_BUF_NOT_SET,
    RETCODE_MEMORY_ACCESS_VIOLATION,    /* 15 */
    RETCODE_VPU_RESPONSE_TIMEOUT,
    RETCODE_INSUFFICIENT_RESOURCE,
    RETCODE_NOT_FOUND_BITCODE_PATH,
    RETCODE_NOT_SUPPORTED_FEATURE,
    RETCODE_NOT_FOUND_VPU_DEVICE,       /* 20 */
    RETCODE_CP0_EXCEPTION,
    RETCODE_STREAM_BUF_FULL,
    RETCODE_ACCESS_VIOLATION_HW,
    RETCODE_QUERY_FAILURE,
    RETCODE_QUEUEING_FAILURE,
    RETCODE_VPU_STILL_RUNNING,
    RETCODE_REPORT_NOT_READY,
#ifdef AUTO_FRM_SKIP_DROP
    RETCODE_FRAME_DROP,
#endif
} RetCode;
```

Description

This is an enumeration for declaring return codes from API function calls. The meaning of each return code is the same for all of the API functions, but the reasons of non-successful return might be different. Some details of those reasons are briefly described in the API definition chapter. In this chapter, the basic meaning of each return code is presented.

RETCODE_SUCCESS
This means that operation was done successfully.

RETCODE_FAILURE
This means that operation was not done successfully. When un-recoverable decoder error happens such as header parsing errors, this value is returned from VPU API.

RETCODE_INVALID_HANDLE
This means that the given handle for the current API function call was invalid (for example, not initialized yet, improper function call for the given handle, etc.).

RETCODE_INVALID_PARAM
This means that the given argument parameters (for example, input data structure) was invalid (not initialized yet or not valid anymore).

RETCODE_INVALID_COMMAND
This means that the given command was invalid (for example, undefined, or not allowed in the given instances).

RETCODE_ROTATOR_OUTPUT_NOT_SET
This means that rotator output buffer was not allocated even though postprocessor (rotation, mirroring, or deringing) is enabled.

RETCODE_ROTATOR_STRIDE_NOT_SET
This means that rotator stride was not provided even though postprocessor (rotation, mirroring, or deringing) is enabled.

RETCODE_FRAME_NOT_COMPLETE
This means that frame decoding operation was not completed yet, so the given API function call cannot be allowed.

RETCODE_INVALID_FRAME_BUFFER
This means that the given source frame buffer pointers were invalid in encoder (not initialized yet or not valid anymore).

RETCODE_INSUFFICIENT_FRAME_BUFFERS
This means that the given numbers of frame buffers were not enough for the operations of the given handle. This return code is only received when calling VPU_DecRegisterFrameBuffer() or VPU_EncRegisterFrameBuffer() function.

RETCODE_INVALID_STRIDE
This means that the given stride was invalid (for example, 0, not a multiple of 8 or smaller than picture size). This return code is only allowed in API functions which set stride.

RETCODE_WRONG_CALL_SEQUENCE
This means that the current API function call was invalid considering the allowed sequences between API functions (for example, missing one crucial function call before this function call).

RETCODE_CALLED_BEFORE
This means that multiple calls of the current API function for a given instance are invalid.

RETCODE_NOT_INITIALIZED
This means that VPU was not initialized yet. Before calling any API functions, the initialization API function, VPU_Init(), should be called at the beginning.

RETCODE_USERDATA_BUF_NOT_SET
This means that there is no memory allocation for reporting userdata. Before setting user data enable, user data buffer address and size should be set with valid value.

RETCODE_MEMORY_ACCESS_VIOLATION
This means that access violation to the protected memory has been occurred.

RETCODE_VPU_RESPONSE_TIMEOUT
This means that VPU response time is too long, time out.

RETCODE_INSUFFICIENT_RESOURCE
This means that VPU cannot allocate memory due to lack of memory.

RETCODE_NOT_FOUND_BITCODE_PATH
This means that BIT_CODE_FILE_PATH has a wrong firmware path or firmware size is 0 when calling VPU_InitWithBitcode() function.

RETCODE_NOT_SUPPORTED_FEATURE
This means that HOST application uses an API option that is not supported in current hardware.

RETCODE_NOT_FOUND_VPU_DEVICE
This means that HOST application uses the undefined product ID.

RETCODE_CP0_EXCEPTION
This means that coprocessor exception has occurred. (WAVE only)

RETCODE_STREAM_BUF_FULL
This means that stream buffer is full in encoder.

RETCODE_ACCESS_VIOLATION_HW
This means that GDI access error has occurred. It might come from violation of write protection region or spec-out GDI read/write request. (WAVE only)

RETCODE_QUERY_FAILURE
This means that query command was not successful (WAVE5 only)

RETCODE_QUEUEING_FAILURE
This means that commands cannot be queued (WAVE5 only)

RETCODE_VPU_STILL_RUNNING
This means that VPU cannot be flushed or closed now. because VPU is running (WAVE5 only)

RETCODE_REPORT_NOT_READY
This means that report is not ready for Query(GET_RESULT) command (WAVE5 only)

RETCODE_FRAME_DROP
This means that frame is dropped. HOST application don't have to wait INT_BIT_PIC_RUN. (CODA9 only)

CodecCommand

```
typedef enum {
    ENABLE_ROTATION,
    DISABLE_ROTATION,
    ENABLE_MIRRORING,
    DISABLE_MIRRORING,
    SET_MIRROR_DIRECTION,
    SET_ROTATION_ANGLE,
    SET_ROTATOR_OUTPUT,
    SET_ROTATOR_STRIDE,
    DEC_GET_SEQ_INFO,
    DEC_SET_SPS_RBSP,
    DEC_SET_PPS_RBSP,
    DEC_SET_SEQ_CHANGE_MASK,
    ENABLE_DERING,
    DISABLE_DERING,
```

```
    SET_SEC_AXI,
    SET_DRAM_CONFIG,    //coda960 only
    GET_DRAM_CONFIG,    //coda960 only
    ENABLE_REP_USERDATA,
    DISABLE_REP_USERDATA,
    SET_ADDR_REP_USERDATA,
    SET_VIRT_ADDR_REP_USERDATA,
    SET_SIZE_REP_USERDATA,
    SET_USERDATA_REPORT_MODE,
    // WAVE4 CU REPORT INFTERFACE
    ENABLE_REP_CUDATA,
    DISABLE_REP_CUDATA,
    SET_ADDR_REP_CUDATA,
    SET_SIZE_REP_CUDATA,
    SET_CACHE_CONFIG,
    GET_TILEDMAP_CONFIG,
    SET_LOW_DELAY_CONFIG,
    GET_LOW_DELAY_OUTPUT,
    SET_DECODE_FLUSH,
    DEC_SET_FRAME_DELAY,
    DEC_SET_WTL_FRAME_FORMAT,
    DEC_GET_FIELD_PIC_TYPE,
    DEC_GET_DISPLAY_OUTPUT_INFO,
    DEC_ENABLE_REORDER,
    DEC_DISABLE_REORDER,
    DEC_SET_AVC_ERROR_CONCEAL_MODE,
    DEC_FREE_FRAME_BUFFER,
    DEC_GET_FRAMEBUF_INFO,
    DEC_RESET_FRAMEBUF_INFO,
    ENABLE_DEC_THUMBNAIL_MODE,
    DEC_SET_DISPLAY_FLAG,
    DEC_SET_TARGET_TEMPORAL_ID,
    DEC_SET_BWB_CUR_FRAME_IDX,
    DEC_SET_FBC_CUR_FRAME_IDX,
    DEC_SET_INTER_RES_INFO_ON,
    DEC_SET_INTER_RES_INFO_OFF,
    DEC_FREE_FBC_TABLE_BUFFER,
    DEC_FREE_MV_BUFFER,
    DEC_ALLOC_FBC_Y_TABLE_BUFFER,
    DEC_ALLOC_FBC_C_TABLE_BUFFER,
    DEC_ALLOC_MV_BUFFER,
    ENC_ADD_PPS,
    ENC_SET_ACTIVE_PPS,
    ENC_GET_ACTIVE_PPS,
    //vpu put header stream to bitstream buffer
    ENC_PUT_VIDEO_HEADER,
    ENC_PUT_MP4_HEADER,
    ENC_PUT_AVC_HEADER,
    //host generate header bitstream
    ENC_GET_VIDEO_HEADER,
    ENC_SET_INTRA_MB_REFRESH_NUMBER,
    ENC_ENABLE_HEC,
    ENC_DISABLE_HEC,
    ENC_SET_SLICE_INFO,
    ENC_SET_GOP_NUMBER,
    ENC_SET_INTRA_QP,
    ENC_SET_BITRATE,
    ENC_SET_FRAME_RATE,

    ENC_SET_PARA_CHANGE,
    ENABLE_LOGGING,
```

```
    DISABLE_LOGGING,
    ENC_SET_SEI_NAL_DATA,
    DEC_GET_QUEUE_STATUS,
    ENC_GET_BW_REPORT,  // wave520 only
    ENC_SET_MIN_MAX_QP,
#if defined(RC_PIC_PARACHANGE) && defined(RC_CHANGE_PARAMETER_DEF)
    ENC_SET_PIC_CHANGE_PARAM,
#endif
    ENC_GET_FRAMEBUF_IDX,
    CMD_END
} CodecCommand;
```

Description

This is a special enumeration type for some configuration commands which can be issued to VPU by HOST application. Most of these commands can be called occasionally, not periodically for changing the configuration of decoder or encoder operation running on VPU. Details of these commands are presented in *the section called "VPU_DecGiveCommand()"*.

ENC_SET_MIN_MAX_QP
This command sets a maximum delta QP and maximum/minimum QP values of intra picture for rate control.

AVCErrorConcealMode

```
typedef enum {
    AVC_ERROR_CONCEAL_MODE_DEFAULT                                  = 0,
    AVC_ERROR_CONCEAL_MODE_ENABLE_SELECTIVE_CONCEAL_MISSING_REFERENCE  = 1,
    AVC_ERROR_CONCEAL_MODE_DISABLE_CONCEAL_MISSING_REFERENCE        = 2,
    AVC_ERROR_CONCEAL_MODE_DISABLE_CONCEAL_WRONG_FRAME_NUM          = 4,
} AVCErrorConcealMode;
```

Description

This is an enumeration type for representing error conceal modes. (H.264/AVC decoder only)

AVC_ERROR_CONCEAL_MODE_DEFAULT
basic error concealment and error concealment for missing reference frame, wrong frame_num syntax (default)

AVC_ERROR_CONCEAL_MODE_ENABLE_SELECTIVE_CONCEAL_MISSING_REFERENCE
error concealment - selective error concealment for missing reference frame

AVC_ERROR_CONCEAL_MODE_DISABLE_CONCEAL_MISSING_REFERENCE
error concealment - disable error concealment for missing reference frame

AVC_ERROR_CONCEAL_MODE_DISABLE_CONCEAL_WRONG_FRAME_NUM
error concealment - disable error concealment for wrong frame_num syntax

CbCrOrder

```
typedef enum {
    CBCR_ORDER_NORMAL,
    CBCR_ORDER_REVERSED
} CbCrOrder;
```

Description

This is an enumeration type for representing the way of writing chroma data in planar format of frame buffer.

CBCR_ORDER_NORMAL
Cb data are written in Cb buffer, and Cr data are written in Cr buffer.

CBCR_ORDER_REVERSED
Cr data are written in Cb buffer, and Cb data are written in Cr buffer.

MirrorDirection

```
typedef enum {
   MIRDIR_NONE,
   MIRDIR_VER,
   MIRDIR_HOR,
   MIRDIR_HOR_VER
} MirrorDirection;
```

Description

This is an enumeration type for representing the mirroring direction.

FrameBufferFormat

```
typedef enum {
   FORMAT_420                = 0  ,      /* 8bit */
   FORMAT_422                ,           /* 8bit */
   FORMAT_224                ,           /* 8bit */
   FORMAT_444                ,           /* 8bit */
   FORMAT_400                ,           /* 8bit */

                                         /* Little Endian Perspective     */
                                         /*     | addr 0  | addr 1  |     */
   FORMAT_420_P10_16BIT_MSB = 5 ,        /* lsb | 00xxxxx |xxxxxxxx | msb */
   FORMAT_420_P10_16BIT_LSB ,            /* lsb | xxxxxxx |xxxxxx00 | msb */
   FORMAT_420_P10_32BIT_MSB ,            /* lsb |00xxxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
   FORMAT_420_P10_32BIT_LSB ,            /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxxx00| msb */

                                         /* 4:2:2 packed format */
                                         /* Little Endian Perspective     */
                                         /*     | addr 0  | addr 1  |     */
   FORMAT_422_P10_16BIT_MSB ,            /* lsb | 00xxxxx |xxxxxxxx | msb */
   FORMAT_422_P10_16BIT_LSB ,            /* lsb | xxxxxxx |xxxxxx00 | msb */
   FORMAT_422_P10_32BIT_MSB ,            /* lsb |00xxxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
   FORMAT_422_P10_32BIT_LSB ,            /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxxx00| msb */

   FORMAT_YUYV               ,
   FORMAT_YUYV_P10_16BIT_MSB,            /* lsb | 000000xxxxxxxxxx | msb */
   FORMAT_YUYV_P10_16BIT_LSB,            /* lsb | xxxxxxxxxx000000 | msb */
   FORMAT_YUYV_P10_32BIT_MSB,            /* lsb |00xxxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
   FORMAT_YUYV_P10_32BIT_LSB,            /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxxx00| msb */
   FORMAT_YVYU               ,
   FORMAT_YVYU_P10_16BIT_MSB,            /* lsb | 000000xxxxxxxxxx | msb */
   FORMAT_YVYU_P10_16BIT_LSB,            /* lsb | xxxxxxxxxx000000 | msb */
   FORMAT_YVYU_P10_32BIT_MSB,            /* lsb |00xxxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
   FORMAT_YVYU_P10_32BIT_LSB,            /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxxx00| msb */
   FORMAT_UYVY               ,
   FORMAT_UYVY_P10_16BIT_MSB,            /* lsb | 000000xxxxxxxxxx | msb */
   FORMAT_UYVY_P10_16BIT_LSB,            /* lsb | 000000xxxxxxxxxx | msb */
   FORMAT_UYVY_P10_32BIT_MSB,            /* lsb |00xxxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
   FORMAT_UYVY_P10_32BIT_LSB,            /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxxx00| msb */
   FORMAT_VYUY               ,
   FORMAT_VYUY_P10_16BIT_MSB,            /* lsb | 000000xxxxxxxxxx | msb */
```

```
    FORMAT_VYUY_P10_16BIT_LSB,          /* lsb | xxxxxxxxxx000000 | msb */
    FORMAT_VYUY_P10_32BIT_MSB,          /* lsb |00xxxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
    FORMAT_VYUY_P10_32BIT_LSB,          /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxxx00| msb */
    FORMAT_MAX,
} FrameBufferFormat;
```

Description

This is an enumeration type for representing chroma formats of the frame buffer and pixel formats in packed mode.

FORMAT_YUYV
8bit packed format : Y0U0Y1V0 Y2U1Y3V1 …

FORMAT_YUYV_P10_16BIT_LSB
10bit packed(YUYV) format(1Pixel=2Byte)

FORMAT_YUYV_P10_32BIT_MSB
10bit packed(YUYV) format(1Pixel=2Byte)

FORMAT_YUYV_P10_32BIT_LSB
10bit packed(YUYV) format(3Pixel=4Byte)

FORMAT_YVYU
10bit packed(YUYV) format(3Pixel=4Byte) 8bit packed format : Y0V0Y1U0 Y2V1Y3U1 …

FORMAT_YVYU_P10_16BIT_LSB
10bit packed(YVYU) format(1Pixel=2Byte)

FORMAT_YVYU_P10_32BIT_MSB
10bit packed(YVYU) format(1Pixel=2Byte)

FORMAT_YVYU_P10_32BIT_LSB
10bit packed(YVYU) format(3Pixel=4Byte)

FORMAT_UYVY
10bit packed(YVYU) format(3Pixel=4Byte) 8bit packed format : U0Y0V0Y1 U1Y2V1Y3 …

FORMAT_UYVY_P10_16BIT_LSB
10bit packed(UYVY) format(1Pixel=2Byte)

FORMAT_UYVY_P10_32BIT_MSB
10bit packed(UYVY) format(1Pixel=2Byte)

FORMAT_UYVY_P10_32BIT_LSB
10bit packed(UYVY) format(3Pixel=4Byte)

FORMAT_VYUY
10bit packed(UYVY) format(3Pixel=4Byte) 8bit packed format : V0Y0U0Y1 V1Y2U1Y3 …

FORMAT_VYUY_P10_16BIT_LSB
10bit packed(VYUY) format(1Pixel=2Byte)

FORMAT_VYUY_P10_32BIT_MSB
10bit packed(VYUY) format(1Pixel=2Byte)

FORMAT_VYUY_P10_32BIT_LSB
10bit packed(VYUY) format(3Pixel=4Byte)

FORMAT_MAX
10bit packed(VYUY) format(3Pixel=4Byte)

ScalerImageFormat

```
typedef enum{
    YUV_FORMAT_I420,
    YUV_FORMAT_NV12,
    YUV_FORMAT_NV21,
    YUV_FORMAT_I422,
    YUV_FORMAT_NV16,
    YUV_FORMAT_NV61,
    YUV_FORMAT_UYVY,
    YUV_FORMAT_YUYV,
} ScalerImageFormat;
```

Description

This is an enumeration type for representing output image formats of down scaler.

YUV_FORMAT_I420
This format is a 420 planar format, which is described as forcc I420.

YUV_FORMAT_NV12
This format is a 420 semi-planar format with U and V interleaved, which is described as fourcc NV12.

YUV_FORMAT_NV21
This format is a 420 semi-planar format with V and U interleaved, which is described as fourcc NV21.

YUV_FORMAT_I422
This format is a 422 planar format, which is described as forcc I422.

YUV_FORMAT_NV16
This format is a 422 semi-planar format with U and V interleaved, which is described as fourcc NV16.

YUV_FORMAT_NV61
This format is a 422 semi-planar format with V and U interleaved, which is described as fourcc NV61.

YUV_FORMAT_UYVY
This format is a 422 packed mode with UYVY, which is described as fourcc UYVY.

YUV_FORMAT_YUYV
This format is a 422 packed mode with YUYV, which is described as fourcc YUYV.

InterruptBit

```
typedef enum {
    INT_BIT_INIT            = 0,
    INT_BIT_SEQ_INIT        = 1,
    INT_BIT_SEQ_END         = 2,
    INT_BIT_PIC_RUN         = 3,
    INT_BIT_FRAMEBUF_SET    = 4,
    INT_BIT_ENC_HEADER      = 5,
    INT_BIT_DEC_PARA_SET    = 7,
```

```
    INT_BIT_DEC_BUF_FLUSH   = 8,
    INT_BIT_USERDATA        = 9,
    INT_BIT_DEC_FIELD       = 10,
    INT_BIT_DEC_MB_ROWS     = 13,
    INT_BIT_BIT_BUF_EMPTY   = 14,
    INT_BIT_BIT_BUF_FULL    = 15
} InterruptBit;
```

Description

This is an enumeration type for representing interrupt bit positions for CODA series.

Wave4InterruptBit

```
typedef enum {
    INT_WAVE_INIT           = 0,
    INT_WAVE_DEC_PIC_HDR    = 1,
    INT_WAVE_ENC_SETPARAM   = 2,
    INT_WAVE_FINI_SEQ       = 2,
    INT_WAVE_DEC_PIC        = 3,
    INT_WAVE_ENC_PIC        = 3,
    INT_WAVE_SET_FRAMEBUF   = 4,
    INT_WAVE_FLUSH_DECODER  = 5,
    INT_WAVE_GET_FW_VERSION = 8,
    INT_WAVE_QUERY_DECODER  = 9,
    INT_WAVE_SLEEP_VPU      = 10,
    INT_WAVE_WAKEUP_VPU     = 11,
    INT_WAVE_CHANGE_INST    = 12,
    INT_WAVE_CREATE_INSTANCE= 14,
    INT_WAVE_BIT_BUF_EMPTY  = 15,   /* Decoder */
    INT_WAVE_BIT_BUF_FULL   = 15,   /* Encoder */
} Wave4InterruptBit;
```

Description

This is an enumeration type for representing interrupt bit positions.

Wave5InterruptBit

```
typedef enum {
    INT_WAVE5_INIT_VPU          = 0,
    INT_WAVE5_WAKEUP_VPU        = 1,
    INT_WAVE5_SLEEP_VPU         = 2,
    INT_WAVE5_CREATE_INSTANCE   = 3,
    INT_WAVE5_FLUSH_INSTANCE    = 4,
    INT_WAVE5_DESTORY_INSTANCE  = 5,
    INT_WAVE5_INIT_SEQ          = 6,
    INT_WAVE5_SET_FRAMEBUF      = 7,
    INT_WAVE5_DEC_PIC           = 8,
    INT_WAVE5_ENC_PIC           = 8,
    INT_WAVE5_ENC_SET_PARAM     = 9,
    INT_WAVE5_DEC_QUERY         = 14,
    INT_WAVE5_BSBUF_EMPTY       = 15,
    INT_WAVE5_BSBUF_FULL        = 15,
} Wave5InterruptBit;
```

Description

This is an enumeration type for representing interrupt bit positions.

PicType

```
typedef enum {
    PIC_TYPE_I            = 0,
    PIC_TYPE_P            = 1,
    PIC_TYPE_B            = 2,
    PIC_TYPE_REPEAT       = 2,
    PIC_TYPE_VC1_BI       = 2,
    PIC_TYPE_VC1_B        = 3,
    PIC_TYPE_D            = 3,
    PIC_TYPE_S            = 3,
    PIC_TYPE_VC1_P_SKIP   = 4,
    PIC_TYPE_MP4_P_SKIP_NOT_CODED = 4,
    PIC_TYPE_IDR          = 5,
    PIC_TYPE_MAX
}PicType;
```

Description

This is an enumeration type for representing picture types.

PIC_TYPE_I
I picture

PIC_TYPE_P
P picture

PIC_TYPE_B
B picture (except VC1)

PIC_TYPE_REPEAT
Repeat frame (VP9 only)

PIC_TYPE_VC1_BI
VC1 BI picture (VC1 only)

PIC_TYPE_VC1_B
VC1 B picture (VC1 only)

PIC_TYPE_D
D picture in MPEG2 that is only composed of DC coefficients (MPEG2 only)

PIC_TYPE_S
S picture in MPEG4 that is an acronym of Sprite and used for GMC (MPEG4 only)

PIC_TYPE_VC1_P_SKIP
VC1 P skip picture (VC1 only)

PIC_TYPE_MP4_P_SKIP_NOT_CODED
Not Coded P Picture in mpeg4 packed mode

PIC_TYPE_IDR
H.264/H.265 IDR picture

PIC_TYPE_MAX
No Meaning

AvcNpfFieldInfo

```
typedef enum {
```

```
    PAIRED_FIELD          = 0,
    TOP_FIELD_MISSING     = 1,
    BOT_FIELD_MISSING     = 2,
}AvcNpfFieldInfo;
```

Description

This is an enumeration type for H.264/AVC NPF (Non Paired Field) information.

FrameFlag

```
typedef enum {
    FF_NONE                 = 0,
    FF_FRAME                = 1,
    FF_FIELD                = 2,
}FrameFlag;
```

Description

This is an enumeration type for specifying frame buffer types when tiled2linear or wtlEnable is used.

FF_NONE
Frame buffer type when tiled2linear or wtlEnable is disabled

FF_FRAME
Frame buffer type to store one frame

FF_FIELD
Frame buffer type to store top field or bottom field separately

BitStreamMode

```
typedef enum {
    BS_MODE_INTERRUPT,
    BS_MODE_RESERVED,
    BS_MODE_PIC_END,
}BitStreamMode;
```

Description

This is an enumeration type for representing bitstream handling modes in decoder.

BS_MODE_INTERRUPT
VPU returns an interrupt when bitstream buffer is empty while decoding. VPU waits for more bitstream to be filled.

BS_MODE_RESERVED
Reserved for the future

BS_MODE_PIC_END
VPU tries to decode with very small amount of bitstream (not a complete 512-byte chunk). If it is not successful, VPU performs error concealment for the rest of the frame.

SWResetMode

```
typedef enum {
    SW_RESET_SAFETY,
```

```
    SW_RESET_FORCE,
    SW_RESET_ON_BOOT
}SWResetMode;
```

Description

This is an enumeration type for representing software reset options.

SW_RESET_SAFETY
It resets VPU in safe way. It waits until pending bus transaction is completed and then perform reset.

SW_RESET_FORCE
It forces to reset VPU without waiting pending bus transaction to be completed. It is used for immediate termination such as system off.

SW_RESET_ON_BOOT
This is the default reset mode that is executed since system booting. This mode is actually executed in VPU_Init(), so does not have to be used independently.

ProductId

```
typedef enum {
    PRODUCT_ID_980,
    PRODUCT_ID_960  = 1,
    PRODUCT_ID_950  = 1,    // same with CODA960
    PRODUCT_ID_7503,
    PRODUCT_ID_320,
    PRODUCT_ID_410,
    PRODUCT_ID_4102,
    PRODUCT_ID_420,
    PRODUCT_ID_412,
    PRODUCT_ID_7Q,
    PRODUCT_ID_420L,
    PRODUCT_ID_510,
    PRODUCT_ID_512,
    PRODUCT_ID_515,
    PRODUCT_ID_520,
    PRODUCT_ID_NONE,
}ProductId;
```

Description

This is an enumeration type for representing product IDs.

TiledMapType

```
typedef enum {
    LINEAR_FRAME_MAP            = 0,
    TILED_FRAME_V_MAP           = 1,
    TILED_FRAME_H_MAP           = 2,
    TILED_FIELD_V_MAP           = 3,
    TILED_MIXED_V_MAP           = 4,
    TILED_FRAME_MB_RASTER_MAP   = 5,
    TILED_FIELD_MB_RASTER_MAP   = 6,
    TILED_FRAME_NO_BANK_MAP     = 7,   // coda980 only
    TILED_FIELD_NO_BANK_MAP     = 8,   // coda980 only
    LINEAR_FIELD_MAP            = 9,   // coda980 only
    CODA_TILED_MAP_TYPE_MAX     = 10,
```

```
    COMPRESSED_FRAME_MAP        = 10,  // WAVE4 only
    ARM_COMPRESSED_FRAME_MAP      = 12,  // AFBC enabled WAVE decoder
    TILED_MAP_TYPE_MAX
} TiledMapType;
```

Description

This is an enumeration type for representing map types for frame buffer.

LINEAR_FRAME_MAP
Linear frame map type

> **Note** Products earlier than CODA9 can only set this linear map type. Linear frame map type

TILED_FRAME_V_MAP
Tiled frame vertical map type (CODA9 only)

TILED_FRAME_H_MAP
Tiled frame horizontal map type (CODA9 only)

TILED_FIELD_V_MAP
Tiled field vertical map type (CODA9 only)

TILED_MIXED_V_MAP
Tiled mixed vertical map type (CODA9 only)

TILED_FRAME_MB_RASTER_MAP
Tiled frame MB raster map type (CODA9 only)

TILED_FIELD_MB_RASTER_MAP
Tiled field MB raster map type (CODA9 only)

TILED_FRAME_NO_BANK_MAP
Tiled frame no bank map. (CODA9 only)

TILED_FIELD_NO_BANK_MAP
Tiled field no bank map. (CODA9 only)

LINEAR_FIELD_MAP
Linear field map type. (CODA9 only)

COMPRESSED_FRAME_MAP
Compressed frame map type (WAVE only)

ARM_COMPRESSED_FRAME_MAP
AFBC(ARM Frame Buffer Compression) compressed frame map type

FramebufferAllocType

```
typedef enum {
    FB_TYPE_CODEC,
    FB_TYPE_PPU,
} FramebufferAllocType;
```

Description

This is an enumeration for declaring a type of framebuffer that is allocated when VPU_DecAllocateFrameBuffer() and VPU_EncAllocateFrameBuffer() function call.

FB_TYPE_CODEC
A framebuffer type used for decoding or encoding

FB_TYPE_PPU
A framebuffer type used for additional allocation of framebuffer for postprocessing(rotation/mirror) or display (tiled2linear) purpose

ChangeCommonParam

```
typedef enum {
    // COMMON parameters
    ENC_PIC_PARAM_CHANGE                = (1<<1),
    ENC_INTRA_PARAM_CHANGE              = (1<<3),
    ENC_CONF_WIN_TOP_BOT_CHANGE         = (1<<4),
    ENC_CONF_WIN_LEFT_RIGHT_CHANGE      = (1<<5),
    ENC_FRAME_RATE_CHANGE               = (1<<6),
    ENC_INDEPENDENT_SLICE_CHANGE        = (1<<7),
    ENC_DEPENDENT_SLICE_CHANGE          = (1<<8),
    ENC_INTRA_REFRESH_CHANGE            = (1<<9),
    ENC_PARAM_CHANGE                    = (1<<10),
    ENC_RC_INIT_QP                      = (1<<11),
    ENC_RC_PARAM_CHANGE                 = (1<<12),
    ENC_RC_MIN_MAX_QP_CHANGE            = (1<<13),
    ENC_RC_TARGET_RATE_LAYER_0_3_CHANGE = (1<<14),
    ENC_RC_TARGET_RATE_LAYER_4_7_CHANGE = (1<<15),
    ENC_RC_INTRA_MIN_MAX_QP_CHANGE      = (1<<16),
    ENC_NUM_UNITS_IN_TICK_CHANGE        = (1<<18),
    ENC_TIME_SCALE_CHANGE               = (1<<19),
    ENC_RC_TRANS_RATE_CHANGE            = (1<<21),
    ENC_RC_TARGET_RATE_CHANGE           = (1<<22),
    ENC_ROT_PARAM_CHANGE                = (1<<23),
    ENC_NR_PARAM_CHANGE                 = (1<<25),
    ENC_NR_WEIGHT_CHANGE                = (1<<26),
    ENC_CHANGE_SET_PARAM_ALL            = (0xFFFFFFFF),
} ChangeCommonParam;
```

Description

This is an enumeration for encoder parameter change. (WAVE4 encoder only)

Wave5ChangeParam

```
typedef enum {
    // COMMON parameters which are changed frame by frame.
    ENC_SET_PPS_PARAM_CHANGE                = (1<<1),
    ENC_SET_RC_FRAMERATE_CHANGE             = (1<<4),
    ENC_SET_INDEPEND_SLICE_CHANGE           = (1<<5),
    ENC_SET_DEPEND_SLICE_CHANGE             = (1<<6),
    ENC_SET_RDO_PARAM_CHANGE                = (1<<8),
    ENC_SET_RC_TARGET_RATE_CHANGE           = (1<<9),
    ENC_SET_RC_PARAM_CHANGE                 = (1<<10),
    ENC_SET_RC_MIN_MAX_QP_CHANGE            = (1<<11),
    ENC_SET_RC_BIT_RATIO_LAYER_CHANGE       = (1<<12),
    ENC_SET_BG_PARAM_CHANGE                 = (1<<18),
    ENC_SET_CUSTOM_MD_CHANGE                = (1<<19),
} Wave5ChangeParam;
```

Description

This is an enumeration for encoder parameter change. (WAVE5 encoder only)

Mp4HeaderType

```
typedef enum {
    VOL_HEADER,
    VOS_HEADER,
    VIS_HEADER
} Mp4HeaderType;
```

Description

This is a special enumeration type for MPEG4 top-level header classes such as visual sequence header, visual object header and video object layer header. It is for MPEG4 encoder only.

VOL_HEADER
Video object layer header

VOS_HEADER
Visual object sequence header

VIS_HEADER
Video object header

AvcHeaderType

```
typedef enum {
    SPS_RBSP,
    PPS_RBSP,
    SPS_RBSP_MVC,
    PPS_RBSP_MVC,
    SVC_RBSP_SEI,
    END_OF_SEQUENCE,
    END_OF_STREAM
} AvcHeaderType;
```

Description

This is a special enumeration type for AVC parameter sets such as sequence parameter set and picture parameter set. It is for AVC encoder only.

SPS_RBSP
Sequence parameter set

PPS_RBSP
Picture parameter set

SPS_RBSP_MVC
Subset sequence parameter set

PPS_RBSP_MVC
Picture parameter set for dependent view

SVC_RBSP_SEI
SEI for SVC

END_OF_SEQUENCE
End of sequence nal for H.264/AVC encoder (CODA7Q only)

END_OF_STREAM
End of stream nal for H.264/AVC encoder (CODA7Q only)

HevcHeaderType

```
typedef enum {
    CODEOPT_ENC_VPS             = (1 << 2),
    CODEOPT_ENC_SPS             = (1 << 3),
    CODEOPT_ENC_PPS             = (1 << 4),
} HevcHeaderType ;
```

Description

This is a special enumeration type for explicit encoding headers such as VPS, SPS, PPS. (WAVE encoder only)

CODEOPT_ENC_VPS
A flag to encode VPS nal unit explicitly

CODEOPT_ENC_SPS
A flag to encode SPS nal unit explicitly

CODEOPT_ENC_PPS
A flag to encode PPS nal unit explicitly

ENC_PIC_CODE_OPTION

```
typedef enum {
    CODEOPT_ENC_HEADER_IMPLICIT = (1 << 0),
    CODEOPT_ENC_VCL             = (1 << 1),
} ENC_PIC_CODE_OPTION;
```

Description

This is a special enumeration type for NAL unit coding options.

CODEOPT_ENC_HEADER_IMPLICIT
A flag to encode (a) headers (VPS, SPS, PPS) implicitly for generating bitstreams conforming to spec.

CODEOPT_ENC_VCL
A flag to encode VCL nal unit explicitly

GOP_PRESET_IDX

```
typedef enum {
    PRESET_IDX_CUSTOM_GOP       = 0,
    PRESET_IDX_ALL_I            = 1,
    PRESET_IDX_IPP              = 2,
    PRESET_IDX_IBBB             = 3,
    PRESET_IDX_IBPBP            = 4,
    PRESET_IDX_IBBBP            = 5,
    PRESET_IDX_IPPPP            = 6,
    PRESET_IDX_IBBBB            = 7,
    PRESET_IDX_RA_IB            = 8,
    PRESET_IDX_T0S              = 16,
    PRESET_IDX_T1S              = 17,
    PRESET_IDX_T1L              = 18,
    PRESET_IDX_T2ST1S           = 19,
    PRESET_IDX_T2ST1L           = 20,
} GOP_PRESET_IDX;
```

Description

This is a special enumeration type for defining GOP structure presets.

PRESET_IDX_CUSTOM_GOP
User defined GOP structure

PRESET_IDX_ALL_I
All Intra, gopsize = 1

PRESET_IDX_IPP
Consecutive P, cyclic gopsize = 1

PRESET_IDX_IBBB
Consecutive B, cyclic gopsize = 1

PRESET_IDX_IBPBP
gopsize = 2

PRESET_IDX_IBBBP
gopsize = 4

PRESET_IDX_IPPPP
Consecutive P, cyclic gopsize = 4

PRESET_IDX_IBBBB
Consecutive B, cyclic gopsize = 4

PRESET_IDX_RA_IB
Random Access, cyclic gopsize = 8

PRESET_IDX_T0S
Bitmain preset1

PRESET_IDX_T1S
Bitmain preset2

PRESET_IDX_T1L
Bitmain preset3

PRESET_IDX_T2ST1S
Bitmain preset4

PRESET_IDX_T2ST1L
Bitmain preset5

2.3. Data Structures

ProductInfo

```
typedef struct {
    Uint32 productId;      //W4_RET_PRODUCT_ID
    Uint32 fwVersion;      //W4_RET_FW_VERSION
    Uint32 productName;      //W4_RET_PRODUCT_NAME
    Uint32 productVersion;      //W4_RET_PRODUCT_VERSION
    Uint32 customerId;      //W4_RET_CUSTOMER_ID
    Uint32 stdDef0;      //W4_RET_STD_DEF0
    Uint32 stdDef1;      //W4_RET_STD_DEF1
    Uint32 confFeature;      //W4_RET_CONF_FEATURE
    Uint32 configDate;      //W4_RET_CONFIG_DATE
    Uint32 configRevision;      //W4_RET_CONFIG_REVISION
    Uint32 configType;      //W4_RET_CONFIG_TYPE
    Uint32 configVcore[4];      //W4_RET_CONF_VCORE0
}ProductInfo;
```

Description

This is data structure of product information. (WAVE only)

productId
The product id

fwVersion
The firmware version

productName
VPU hardware product name

productVersion
VPU hardware product version

customerId
The customer id

stdDef0
The system configuration information

stdDef1
The hardware configuration information

confFeature
The supported codec standard

configDate
The date that the hardware has been configured in YYYYmmdd in digit

configRevision
The revision number when the hardware has been configured

configType
The define value used in hardware configuration

configVcore
VCORE Configuration Information

TiledMapConfig

```
typedef struct {
    // gdi2.0
    int xy2axiLumMap[32];
    int xy2axiChrMap[32];
    int xy2axiConfig;

    // gdi1.0
    int xy2caMap[16];
    int xy2baMap[16];
    int xy2raMap[16];
    int rbc2axiMap[32];
    int xy2rbcConfig;
    unsigned long tiledBaseAddr;

    // common
    int mapType;
    int productId;
    int tbSeparateMap;
    int topBotSplit;
    int tiledMap;
    int convLinear;
} TiledMapConfig;
```

Description

This is a data structure of tiled map information.

Note WAVE4 does not support tiledmap type so this structure is not used in the product.

DRAMConfig

```
typedef struct {
    int rasBit;
    int casBit;
    int bankBit;
    int busBit;
} DRAMConfig;
```

Description

This is a data structure of DRAM information. (CODA960 and BODA950 only). VPUAPI sets default values for this structure. However, HOST application can configure if the default values are not associated with their DRAM or desirable to change.

rasBit
This value is used for width of RAS bit. (13 on the CNM FPGA platform)

casBit
This value is used for width of CAS bit. (9 on the CNM FPGA platform)

bankBit
This value is used for width of BANK bit. (2 on the CNM FPGA platform)

busBit
This value is used for width of system BUS bit. (3 on CNM FPGA platform)

FrameBuffer

```
typedef struct {
    PhysicalAddress bufY;
    PhysicalAddress bufCb;
    PhysicalAddress bufCr;
    PhysicalAddress bufYBot;     // coda980 only
    PhysicalAddress bufCbBot;    // coda980 only
    PhysicalAddress bufCrBot;    // coda980 only
    int cbcrInterleave;
    int nv21;
    int endian;
    int myIndex;
    int mapType;
    int stride;
    int width;
    int height;
    int size;
    int lumaBitDepth;
    int chromaBitDepth;
    FrameBufferFormat   format;
    int  sourceLBurstEn;
    int srcBufState;
    int sequenceNo;
    BOOL updateFbInfo;
} FrameBuffer;
```

Description

This is a data structure for representing frame buffer information such as pointer of each YUV component, endian, map type, etc.

All of the 3 component addresses must be aligned to AXI bus width. HOST application must allocate external SDRAM spaces for those components by using this data structure. For example, YCbCr 4:2:0, one pixel value of a component occupies one byte, so the frame data sizes of Cb and Cr buffer are 1/4 of Y buffer size.

In case of CbCr interleave mode, Cb and Cr frame data are written to memory area started from bufCb address. Also, in case that the map type of frame buffer is a field type, the base addresses of frame buffer for bottom fields - bufYBot, bufCbBot and bufCrBot should be set separately.

bufY

It indicates the base address for Y component in the physical address space when linear map is used. It is the RAS base address for Y component when tiled map is used (CODA9). It is also compressed Y buffer or ARM compressed framebuffer (WAVE).

bufCb

It indicates the base address for Cb component in the physical address space when linear map is used. It is the RAS base address for Cb component when tiled map is used (CODA9). It is also compressed CbCr buffer (WAVE)

bufCr

It indicates the base address for Cr component in the physical address space when linear map is used. It is the RAS base address for Cr component when tiled map is used (CODA9).

bufYBot

It indicates the base address for Y bottom field component in the physical address space when linear map is used. It is the RAS base address for Y bottom field component when tiled map is used (CODA980 only).

bufCbBot
It indicates the base address for Cb bottom field component in the physical address space when linear map is used. It is the RAS base address for Cb bottom field component when tiled map is used (CODA980 only).

bufCrBot
It indicates the base address for Cr bottom field component in the physical address space when linear map is used. It is the RAS base address for Cr bottom field component when tiled map is used (CODA980 only).

cbcrInterleave
It specifies a chroma interleave mode of frame buffer.

- 0 : CbCr data is written in their separate frame memory (chroma separate mode).
- 1 : CbCr data is interleaved in chroma memory (chroma interleave mode).

nv21
It specifies the way chroma data is interleaved in the frame buffer, bufCb or bufCbBot.
- 0 : CbCr data is interleaved in chroma memory (NV12).
- 1 : CrCb data is interleaved in chroma memory (NV21).

endian
It specifies endianess of frame buffer.
- 0 : little endian format
- 1 : big endian format
- 2 : 32 bit little endian format
- 3 : 32 bit big endian format
- 16 ~ 31 : 128 bit endian format

Note For setting specific values of 128 bit endiness, please refer to the *WAVE Datasheet*.

myIndex
A frame buffer index to identify each frame buffer that is processed by VPU.

mapType
A map type for GDI inferface or FBC (Frame Buffer Compression). NOTE: For detailed map types, please refer to *the section called "TiledMapType"*.

stride
A horizontal stride for given frame buffer

width
A width for given frame buffer

height
A height for given frame buffer

size
A size for given frame buffer

lumaBitDepth
Bit depth for luma component

chromaBitDepth
Bit depth for chroma component

format
A YUV format of frame buffer

sourceLBurstEn
It enables source frame data with long burst length to be loaded for reducing DMA latency (CODA9 encoder only).
- 0 : disable the long-burst mode.
- 1 : enable the long-burst mode.

srcBufState
It indicates a status of each source buffer, whether the source buffer is used for encoding or not.

sequenceNo
A sequence number that the frame belongs to. It increases by 1 every time a sequence changes in decoder.

updateFbInfo
If this is TRUE, VPU updates API-internal framebuffer information when any of the information is changed.

FrameBufferAllocInfo

```
typedef struct {
    int mapType;
    int cbcrInterleave;
    int nv21;
    FrameBufferFormat format;
    int stride;
    int height;
    int size;
    int lumaBitDepth;
    int chromaBitDepth;
    int endian;
    int num;
    int type;
} FrameBufferAllocInfo;
```

Description

This is a data structure for representing framebuffer parameters. It is used when framebuffer allocation using VPU_DecAllocateFrameBuffer() or VPU_EncAllocateFrameBuffer().

mapType
the section called "TiledMapType"

cbcrInterleave
CbCr interleave mode of frame buffer

nv21
1 : CrCb (NV21) , 0 : CbCr (NV12). This is valid when cbcrInterleave is 1.

format
the section called "FrameBufferFormat"

stride
A stride value of frame buffer

height
A height of frame buffer

size
A size of frame buffer

lumaBitDepth
A bit-depth of luma sample

chromaBitDepth
A bit-depth of chroma sample

endian
An endianess of frame buffer

num
The number of frame buffer to allocate

type
the section called "FramebufferAllocType"

VpuRect

```
typedef struct {
    Uint32 left;
    Uint32 top;
    Uint32 right;
    Uint32 bottom;
} VpuRect;
```

Description

This is a data structure for representing rectangular window information in a frame.

In order to specify a display window (or display window after cropping), this structure is provided to HOST application. Each value means an offset from the start point of a frame and therefore, all variables have positive values.

left
A horizontal pixel offset of top-left corner of rectangle from (0, 0)

top
A vertical pixel offset of top-left corner of rectangle from (0, 0)

right
A horizontal pixel offset of bottom-right corner of rectangle from (0, 0)

bottom
A vertical pixel offset of bottom-right corner of rectangle from (0, 0)

ThoScaleInfo

```
typedef struct {
    int frameWidth;
    int frameHeight;
    int picWidth;
    int picHeight;
    int picOffsetX;
    int picOffsetY;
} ThoScaleInfo;
```

Description

This is a data structure of picture size information. This structure is valid only for Theora decoding case. When HOST application allocates frame buffers and gets a displayable picture region, HOST application needs this information.

frameWidth
This value is used for width of frame buffer.

frameHeight
This value is used for height of frame buffer.

picWidth
This value is used for width of the displayable picture region.

picHeight
This value is used for height of the displayable picture region.

picOffsetX
This value is located at the lower-left corner of the displayable picture region.

picOffsetY
This value is located at the lower-left corner of the displayable picture region.

Vp8ScaleInfo

```
typedef struct {
    unsigned hScaleFactor   : 2;
    unsigned vScaleFactor   : 2;
    unsigned picWidth       : 14;
    unsigned picHeight      : 14;
} Vp8ScaleInfo;
```

Description

This is a data structure of picture upscaling information for post-processing out of decoding loop. This structure is valid only for VP8 decoding case and can never be used by VPU itself. If HOST application has an upsampling device, this information is useful for them. When the HOST application allocates a frame buffer, HOST application needs upscaled resolution derived by this information to allocate enough (maximum) memory for variable resolution picture decoding.

hScaleFactor
This is an upscaling factor for horizontal expansion. The value could be 0 to 3, and meaning of each value is described in below table.

Table 2.1. Upsampling Ratio by Scale Factor

h/vScaleFactor	Upsampling Ratio
0	1
1	5/4
2	5/3
3	2/1

vScaleFactor
This is an upscaling factor for vertical expansion. The value could be 0 to 3, meaning of each value is described in above table.

picWidth
Picture width in units of sample

picHeight
Picture height in units of sample

LowDelayInfo

```
typedef struct {
    int lowDelayEn;
    int numRows;
} LowDelayInfo;
```

Description

The data structure to enable low delay decoding.

lowDelayEn
This enables low delay decoding. (CODA980 H.264/AVC decoder only)

If this flag is 1, VPU sends an interrupt to HOST application when numRows decoding is done.

- 0 : disable
- 1 : enable

When this field is enabled, reorderEnable, tiled2LinearEnable, and the post-rotator should be disabled.

numRows
This field indicates the number of mb rows (macroblock unit).

The value is from 1 to height/16 - 1. If the value of this field is 0 or picture height/16, low delay decoding is disabled even though lowDelayEn is 1.

SecAxiUse

```
typedef struct {
    union {
        struct {
            int useBitEnable;
            int useIpEnable;
            int useDbkYEnable;
            int useDbkCEnable;
            int useOvlEnable;
            int useBtpEnable;
            int useMeEnable;
            int useScalerEnable;
        } coda9;
        struct {
            // for Decoder
            int useBitEnable;
            int useIpEnable;
            int useLfRowEnable;
            // for Encoder
            int useEncImdEnable;
            int useEncLfEnable;
            int useEncRdoEnable;
        } wave4;
    } u;
} SecAxiUse;
```

Description

This is a data structure for representing use of secondary AXI for each hardware block.

useBitEnable
This enables AXI secondary channel for prediction data of the BIT-processor.

useIpEnable
This enables AXI secondary channel for row pixel data of IP.

useDbkYEnable
This enables AXI secondary channel for temporal luminance data of the de-blocking filter.

useDbkCEnable
This enables AXI secondary channel for temporal chrominance data of the de-blocking filter.

useOvlEnable
This enables AXI secondary channel for temporal data of the the overlap filter (VC1 only).

useBtpEnable
This enables AXI secondary channel for bit-plane data of the BIT-processor (VC1 only).

useMeEnable
This enables AXI secondary channel for motion estimation data.

useScalerEnable
This enables AXI secondary channel for scaler temporal data.

useLfRowEnable
This enables AXI secondary channel for loopfilter.

useEncImdEnable
This enables AXI secondary channel for intra mode decision.

useEncLfEnable
This enables AXI secondary channel for loopfilter.

useEncRdoEnable
This enables AXI secondary channel for RDO.

CacheSizeCfg

```
typedef struct {
    unsigned BufferSize     : 8;
    unsigned PageSizeX      : 4;
    unsigned PageSizeY      : 4;
    unsigned CacheSizeX     : 4;
    unsigned CacheSizeY     : 4;
    unsigned Reserved       : 8;
} CacheSizeCfg;
```

Description

This is a data structure for representing cache rectangle area for each component of MC reference frame. (CODA9 only)

BufferSize
This is the cache buffer size for each component and can be set with 0 to 255. The unit of this value is fixed with 256byte.

PageSizeX
This is the cache page size and can be set as 0 to 4. With this value(n), 8*(2^n) byte is requested as the width of a page.

PageSizeY

This is the cache page size and can be set as 0 to 7. With this value(m), a page width*(2^m) byte is requested as the rectangle of a page.

CacheSizeX

This is the component data cache size, and it can be set as 0 to 7 in a page unit. Then there can be 2^n pages in x(y)-direction. Make sure that for luma component the CacheSizeX + CacheSizeY must be less than 8. For chroma components, CacheSizeX + CacheSizeY must be less than 7.

CacheSizeY

This is the component data cache size, and it can be set as 0 to 7 in a page unit. Then there can be 2^n pages in x(y)-direction. Make sure that for luma component the CacheSizeX + CacheSizeY must be less than 8. For chroma components, CacheSizeX + CacheSizeY must be less than 7.

MaverickCacheConfig

```
typedef struct {
    struct {
        union {
            Uint32 word;
            CacheSizeCfg cfg;
        } luma;
        union {
            Uint32 word;
            CacheSizeCfg cfg;
        } chroma;
        unsigned Bypass         : 1;
        unsigned DualConf       : 1;
        unsigned PageMerge      : 2;
    } type1;
    struct {
 unsigned int CacheMode;
    } type2;
} MaverickCacheConfig;
```

Description

This is a data structure for cache configuration. (CODA9 only)

cfg

the section called “CacheSizeCfg”

Bypass

It disables cache function.

- 1 : Cache off
- 0 : Cache on

DualConf

It enables two frame caching mode.

- 1 : Dual mode (caching for FrameIndex0 and FrameIndex1)
- 0 : Single mode (caching for FrameIndex0)

PageMerge

Mode for page merging

- 0 : Disable
- 1 : Horizontal

- 2 : Vertical

We recommend you to set 1 (horizontal) in tiled map or to set 2 (vertical) in linear map.

CacheMode

CacheMode represents cache configuration.

- [10:9] : Cache line processing direction and merge mode
- [8:5] : CacheWayShape
  - [8:7] : CacheWayLuma
  - [6:5] : CacheWayChroma
- [4] reserved
- [3] CacheBurstMode
  - 0: burst 4
  - 1: bust 8
- [2] CacheMapType
  - 0: linear
  - 1: tiled
- [1] CacheBypassME
  - 0: Cache enable
  - 1: Cache disable (bypass)
- [0] CacheBypassMC
  - 0: Cache enable
  - 1: Cache disable (bypass)

DecParamSet

```
typedef struct {
    Uint32 * paraSet;
    int     size;
} DecParamSet;
```

Description

This structure is used when HOST application additionally wants to send SPS data or PPS data from external way. The resulting SPS data or PPS data can be used in real applications as a kind of out-of-band information.

paraSet

The SPS/PPS rbsp data

size

The size of stream in byte

AvcVuiInfo

```
typedef struct {
    int fixedFrameRateFlag;
    int timingInfoPresent;
    int chromaLocBotField;
    int chromaLocTopField;
    int chromaLocInfoPresent;
    int colorPrimaries;
    int colorDescPresent;
    int isExtSAR;
    int vidFullRange;
    int vidFormat;
    int vidSigTypePresent;
```

```
    int vuiParamPresent;
    int vuiPicStructPresent;
    int vuiPicStruct;
} AvcVuiInfo;
```

Description

This is a data structure for H.264/AVC specific picture information. Only H.264/AVC decoder returns this structure after decoding a frame. For details about all these flags, please find them in H.264/AVC VUI syntax.

fixedFrameRateFlag

- 1 : It indicates that the temporal distance between the decoder output times of any two consecutive pictures in output order is constrained as fixed_frame_rate_flag in H.264/AVC VUI syntax.
- 0 : It indicates that no such constraints apply to the temporal distance between the decoder output times of any two consecutive pictures in output order

timingInfoPresent

timing_info_present_flag in H.264/AVC VUI syntax

- 1 : FixedFrameRateFlag is valid.
- 0 : FixedFrameRateFlag is not valid.

chromaLocBotField

chroma_sample_loc_type_bottom_field in H.264/AVC VUI syntax. It specifies the location of chroma samples for the bottom field.

chromaLocTopField

chroma_sample_loc_type_top_field in H.264/AVC VUI syntax. It specifiesf the location of chroma samples for the top field.

chromaLocInfoPresent

chroma_loc_info_present_flag in H.264/AVC VUI syntax.

colorPrimaries

chroma_loc_info_present_flag in H.264/AVC VUI syntax

- 1 : ChromaSampleLocTypeTopField and ChromaSampleLoc TypeTopField are valid.
- 0 : ChromaSampleLocTypeTopField and ChromaSampleLoc TypeTopField are not valid. colour_primaries syntax in VUI parameter in H.264/AVC

colorDescPresent

colour_description_present_flag in VUI parameter in H.264/AVC

isExtSAR

This flag indicates whether aspectRateInfo represents 8bit aspect_ratio_idc or 32bit extended_SAR. If the aspect_ratio_idc is extended_SAR mode, this flag returns 1.

vidFullRange

video_full_range in VUI parameter in H.264/AVC

vidFormat

video_format in VUI parameter in H.264/AVC

vidSigTypePresent

video_signal_type_present_flag in VUI parameter in H.264/AVC

vuiParamPresent

vui_parameters_present_flag in VUI parameter in H.264/AVC

vuiPicStructPresent
pic_struct_present_flag of VUI in H.264/AVC. This field is valid only for H.264/AVC decoding.

vuiPicStruct
pic_struct in H.264/AVC VUI reporting (Table D-1 in H.264/AVC specification)

MP2BarDataInfo

```
typedef struct {
    int barLeft;
    int barRight;
    int barTop;
    int barBottom;
} MP2BarDataInfo;
```

Description

This is a data structure for bar information of MPEG2 user data. For more details on this, please refer to *ATSC Digital Television Standard: Part 4:2009*.

barLeft
A 14-bit unsigned integer value representing the last horizontal luminance sample of a vertical pillarbox bar area at the left side of the reconstructed frame. Pixels shall be numbered from zero, starting with the leftmost pixel.

This variable is initialized to -1.

barRight
A 14-bit unsigned integer value representing the first horizontal luminance sample of a vertical pillarbox bar area at the right side of the reconstructed frame. Pixels shall be numbered from zero, starting with the leftmost pixel.

This variable is initialized to -1.

barTop
A 14-bit unsigned integer value representing the first line of a horizontal letterbox bar area at the top of the reconstructed frame. Designation of line numbers shall be as defined per each applicable standard in Table 6.9.

This variable is initialized to -1.

barBottom
A 14-bit unsigned integer value representing the first line of a horizontal letterbox bar area at the bottom of the reconstructed frame. Designation of line numbers shall be as defined per each applicable standard in Table 6.9.

This variable is initialized to -1.

MP2PicDispExtInfo

```
typedef struct {
    Uint32  offsetNum;
    Int16   horizontalOffset1;
    Int16   horizontalOffset2;
    Int16   horizontalOffset3;
    Int16   verticalOffset1;
```

```
    Int16   verticalOffset2;
    Int16   verticalOffset3;
} MP2PicDispExtInfo;
```

Description

This is a data structure for MP2PicDispExtInfo.

Note For detailed information on these fields, please refer to the MPEG2 standard specification.

offsetNum
This is number of frame_centre_offset with a range of 0 to 3, inclusive.

horizontalOffset1
A horizontal offset of display rectangle in units of 1/16th sample

horizontalOffset2
A horizontal offset of display rectangle in units of 1/16th sample

horizontalOffset3
A horizontal offset of display rectangle in units of 1/16th sample

verticalOffset1
A vertical offset of display rectangle in units of 1/16th sample

verticalOffset2
A vertical offset of display rectangle in units of 1/16th sample

verticalOffset3
A vertical offset of display rectangle in units of 1/16th sample

DecOpenParam

```
typedef struct {
    CodStd          bitstreamFormat;
    PhysicalAddress bitstreamBuffer;
    int             bitstreamBufferSize;
    int             mp4DeblkEnable;
    int             avcExtension;
    int             mp4Class;
    int             tiled2LinearEnable;
    int             tiled2LinearMode;
    int             wtlEnable;
    int             wtlMode;
    int             cbcrInterleave;
    int             nv21;
    int             cbcrOrder;
    int             bwbEnable;
    EndianMode      frameEndian;
    EndianMode      streamEndian;
    int             bitstreamMode;
    Uint32          coreIdx;
    vpu_buffer_t    vbWork;
    int             fbc_mode;
    Uint32          virtAxiID;
    BOOL            bwOptimization;
    Uint32          div3Width;
    Uint32          div3Height;
```

```
} DecOpenParam;
```

Description

This data structure is a group of common decoder parameters to run VPU with a new decoding instance. This is used when HOST application calls VPU_Decopen().

bitstreamFormat
A standard type of bitstream in decoder operation. It is one of codec standards defined in CodStd.

bitstreamBuffer
The start address of bitstream buffer from which the decoder can get the next bitstream. This address must be aligned to AXI bus width.

bitstreamBufferSize
The size of the buffer pointed by bitstreamBuffer in byte. This value must be a multiple of 1024.

mp4DeblkEnable
- 0 : disable
- 1 : enable

When this field is set in case of MPEG4, H.263 (post-processing), DivX3 or MPEG2 decoding, VPU generates MPEG4 deblocking filtered output.

avcExtension
- 0 : No extension of H.264/AVC
- 1 : MVC extension of H.264/AVC

mp4Class
- 0 : MPEG4
- 1 : DivX 5.0 or higher
- 2 : Xvid
- 5 : DivX 4.0
- 6 : old Xvid
- 256 : Sorenson Spark

Note This variable is only valid when decoding MPEG4 stream.

tiled2LinearEnable
It enables a tiled to linear map conversion feature for display.

tiled2LinearMode
It specifies which picture type is converted to. (CODA980 only)
- 1 : conversion to linear frame map (when FrameFlag enum is FF_FRAME)
- 2 : conversion to linear field map (when FrameFlag enum is FF_FIELD)

wtlEnable
It enables WTL (Write Linear) function. If this field is enabled, VPU writes a decoded frame to the frame buffer twice - first in linear map and second in tiled or compressed map. Therefore, HOST application should allocate one more frame buffer for saving both formats of frame buffers.

wtlMode
It specifies whether VPU writes in frame linear map or in field linear map when WTL is enabled. (CODA980 only)
- 1 : write decoded frames in frame linear map (when FrameFlag enum is FF_FRAME)
- 2 : write decoded frames in field linear map (when FrameFlag enum is FF_FIELD)

cbcrInterleave
- 0 : CbCr data is written in separate frame memories (chroma separate mode)
- 1 : CbCr data is interleaved in chroma memory. (chroma interleave mode)

nv21

CrCb interleaved mode (NV21).
- 0 : Decoded chroma data is written in CbCr (NV12) format.
- 1 : Decoded chroma data is written in CrCb (NV21) format.

This is only valid if cbcrInterleave is 1.

cbcrOrder

CbCr order in planar mode (YV12 format)
- 0 : Cb data are written first and then Cr written in their separate plane.
- 1 : Cr data are written first and then Cb written in their separate plane.

bwbEnable

It writes output with 8 burst in linear map mode. (CODA9 only)
- 0 : burst write back is disabled
- 1 : burst write back is enabled.

frameEndian

Frame buffer endianness
- 0 : little endian format
- 1 : big endian format
- 2 : 32 bit little endian format
- 3 : 32 bit big endian format
- 16 ~ 31 : 128 bit endian format

Note For setting specific values of 128 bit endiness, please refer to the *WAVE Datasheet*.

streamEndian

Bitstream buffer endianess
- 0 : little endian format
- 1 : big endian format
- 2 : 32 bits little endian format
- 3 : 32 bits big endian format
- 16 ~ 31 : 128 bit endian format

Note For setting specific values of 128 bit endiness, please refer to the *WAVE Datasheet*.

bitstreamMode

When read pointer reaches write pointer in the middle of decoding one picture,
- 0 : VPU sends an interrupt to HOST application and waits for more bitstream to decode. (interrupt mode)
- 1 : Reserved
- 2 : VPU decodes bitstream from read pointer to write pointer. (PicEnd mode)

coreIdx

VPU core index number (0 ~ [number of VPU core] - 1)

vbWork

BIT processor work buffer SDRAM address/size information. In parallel decoding operation, work buffer is shared between VPU cores. The work buffer address is set to this member variable when VPU_Decopen() is called. Unless HOST application sets the address and size of work buffer, VPU allocates automatically work buffer when VPU_DecOpen() is executed.

fbc_mode
It determines prediction mode of frame buffer compression.
- 0x00 : Best Predection (best for bandwidth, but some performance overhead might exist)
- 0x0C : Normal Prediction (good for bandwidth and performance)
- 0x3C : Basic Predcition (best for performance)

virtAxiID
AXI_ID to distinguish guest OS. For virtualization only. Set this value in highest bit order.

bwOptimization
Bandwidth optimization feature which allows WTL(Write to Linear)-enabled VPU to skip writing compressed format of non-reference pictures or linear format of non-display pictures to the frame buffer for BW saving reason.

div3Width
The picture width of div3 stream (CODA7Q only)

div3Height
The picture height of div3 stream (CODA7Q only)

DecInitialInfo

```
typedef struct {
    Int32           picWidth;
    Int32           picHeight;

    Int32           fRateNumerator;
    Int32           fRateDenominator;
    VpuRect         picCropRect;
    Int32           mp4DataPartitionEnable;
    Int32           mp4ReversibleVlcEnable;
    Int32           mp4ShortVideoHeader;
    Int32           h263AnnexJEnable;
    Int32           minFrameBufferCount;
    Int32           frameBufDelay;
    Int32           normalSliceSize;
    Int32           worstSliceSize;
    // Report Information
    Int32           maxSubLayers;
    Int32           profile;
    Int32           level;
    Int32           tier;
    Int32           interlace;
    Int32           constraint_set_flag[4];
    Int32           direct8x8Flag;
    Int32           vc1Psf;
    Int32           isExtSAR;
    Int32           maxNumRefFrmFlag;
    Int32           maxNumRefFrm;
    Int32           aspectRateInfo;
    Int32           bitRate;
    ThoScaleInfo    thoScaleInfo;
    Vp8ScaleInfo    vp8ScaleInfo;
    Int32           mp2LowDelay;
    Int32           mp2DispVerSize;
    Int32           mp2DispHorSize;
    Uint32          userDataHeader;
    Int32           userDataNum;
    Int32           userDataSize;
    Int32           userDataBufFull;
```

```
    //VUI information
    Int32           chromaFormatIDC;
    Int32           lumaBitdepth;
    Int32           chromaBitdepth;
    Int32           seqInitErrReason;
    Int32           warnInfo;
    PhysicalAddress rdPtr;
    PhysicalAddress wrPtr;
    AvcVuiInfo      avcVuiInfo;
    MP2BarDataInfo  mp2BardataInfo;
    Uint32          sequenceNo;
    Int32           numReorderFrames;
} DecInitialInfo;
```

Description

Data structure to get information necessary to start decoding from the decoder.

picWidth
Horizontal picture size in pixel

This width value is used while allocating decoder frame buffers. In some cases, this returned value, the display picture width declared on stream header, should be aligned to a specific value depending on product and video standard before allocating frame buffers.

picHeight
Vertical picture size in pixel

This height value is used while allocating decoder frame buffers. In some cases, this returned value, the display picture height declared on stream header, should be aligned to a specific value depending on product and video standard before allocating frame buffers.

fRateNumerator
The numerator part of frame rate fraction

Note The meaning of this flag can vary by codec standards. For details about this, please refer to *Appendix: FRAME RATE NUMERATORS in programmer's guide*.

fRateDenominator
The denominator part of frame rate fraction

Note The meaning of this flag can vary by codec standards. For details about this, please refer to *Appendix: FRAME RATE DENOMINATORS in programmer's guide*.

picCropRect
Picture cropping rectangle information (H.264/H.265/AVS decoder only)

This structure specifies the cropping rectangle information. The size and position of cropping window in full frame buffer is presented by using this structure.

mp4DataPartitionEnable
data_partitioned syntax value in MPEG4 VOL header

mp4ReversibleVlcEnable
reversible_vlc syntax value in MPEG4 VOL header

mp4ShortVideoHeader
- 0 : not h.263 stream

- 1 : h.263 stream(mpeg4 short video header)

h263AnnexJEnable
- 0 : Annex J disabled
- 1 : Annex J (optional deblocking filter mode) enabled

minFrameBufferCount
This is the minimum number of frame buffers required for decoding. Applications must allocate at least as many as this number of frame buffers and register the number of buffers to VPU using VPU_DecRegisterFrameBuffer() before decoding pictures.

frameBufDelay
This is the maximum display frame buffer delay for buffering decoded picture reorder. VPU may delay decoded picture display for display reordering when H.264/H.265, pic_order_cnt_type 0 or 1 case and for B-frame handling in VC1 decoder.

normalSliceSize
This is the recommended size of buffer used to save slice in normal case. This value is determined by quarter of the memory size for one raw YUV image in KB unit. This is only for H.264.

worstSliceSize
This is the recommended size of buffer used to save slice in worst case. This value is determined by half of the memory size for one raw YUV image in KB unit. This is only for H.264.

maxSubLayers
Number of sub-layer for H.265/HEVC

profile
- H.265/H.264 : profile_idc
- VC1
  - 0 : Simple profile
  - 1 : Main profile
  - 2 : Advanced profile
- MPEG2
  - 3'b101 : Simple
  - 3'b100 : Main
  - 3'b011 : SNR Scalable
  - 3'b10 : SpatiallyScalable
  - 3'b001 : High
- MPEG4
  - 8'b00000000 : SP
  - 8'b00001111 : ASP
- Real Video
  - 8 (version 8)
  - 9 (version 9)
  - 10 (version 10)
- AVS
  - 8'b0010 0000 : Jizhun profile
  - 8'b0100 1000 : Guangdian profile
- VP8 : 0 - 3

level
- H.265/H.264 : level_idc
- VC1 : level
- MPEG2 :
  - 4'b1010 : Low
  - 4'b1000 : Main

  - 4'b0110 : High 1440,
  - 4'b0100 : High
- MPEG4 :
  - SP
    - 4'b1000 : L0
    - 4'b0001 : L1
    - 4'b0010 : L2
    - 4'b0011 : L3
  - ASP
    - 4'b0000 : L0
    - 4'b0001 : L1
    - 4'b0010 : L2
    - 4'b0011 : L3
    - 4'b0100 : L4
    - 4'b0101 : L5
- Real Video : N/A (real video does not have any level info).
- AVS :
  - 4'b0000 : L2.0
  - 4'b0001 : L4.0
  - 4'b0010 : L4.2
  - 4'b0011 : L6.0
  - 4'b0100 : L6.2
- VC1 : level in struct B

tier

A tier indicator
- 0 : Main
- 1 : High

interlace

When this value is 1, decoded stream may be decoded into progressive or interlace frame. Otherwise, decoded stream is progressive frame.

constraint_set_flag

constraint_set0_flag ~ constraint_set3_flag in H.264/AVC SPS

direct8x8Flag

direct_8x8_inference_flag in H.264/AVC SPS

vc1Psf

Progressive Segmented Frame(PSF) in VC1 sequence layer

maxNumRefFrmFlag

This is one of the SPS syntax elements in H.264.
- 0 : max_num_ref_frames is 0.
- 1 : max_num_ref_frames is not 0.

aspectRateInfo
- H.264/AVC : When avcIsExtSAR is 0, this indicates aspect_ratio_idc[7:0]. When avcIsExtSAR is 1, this indicates sar_width[31:16] and sar_height[15:0]. If aspect_ratio_info_present_flag = 0, the register returns -1 (0xffffffff).
- VC1 : This reports ASPECT_HORIZ_SIZE[15:8] and ASPECT_VERT_SIZE[7:0].
- MPEG2 : This value is index of Table 6-3 in ISO/IEC 13818-2.
- MPEG4/H.263 : This value is index of Table 6-12 in ISO/IEC 14496-2.
- RV : aspect_ratio_info
- AVS : This value is the aspect_ratio_info[3:0] which is used as index of Table 7-5 in AVS Part2

bitRate
The bitrate value written in bitstream syntax. If there is no bitRate, this reports -1.

thoScaleInfo
This is the Theora picture size information. Refer to *the section called "ThoScaleInfo"*.

vp8ScaleInfo
This is VP8 upsampling information. Refer to *the section called "Vp8ScaleInfo"*.

mp2LowDelay
This is low_delay syntax of sequence extension in MPEG2 specification.

mp2DispVerSize
This is display_vertical_size syntax of sequence display extension in MPEG2 specification.

mp2DispHorSize
This is display_horizontal_size syntax of sequence display extension in MPEG2 specification.

userDataHeader
Refer to userDataHeader in *the section called "DecOutputExtData"*.

userDataNum
Refer to userDataNum in *the section called "DecOutputExtData"*.

userDataSize
Refer to userDataSize in *the section called "DecOutputExtData"*.

userDataBufFull
Refer to userDataBufFull in *the section called "DecOutputExtData"*.

chromaFormatIDC
A chroma format indicator

lumaBitdepth
A bit-depth of luma sample

chromaBitdepth
A bit-depth of chroma sample

seqInitErrReason
This is an error reason of sequence header decoding. For detailed meaning of returned value, please refer to the *Appendix: ERROR DEFINITION in programmer's guide*.

rdPtr
A read pointer of bitstream buffer

wrPtr
A write pointer of bitstream buffer

avcVuiInfo
This is H.264/AVC VUI information. Refer to *the section called "AvcVuiInfo"*.

mp2BardataInfo
This is bar information in MPEG2 user data. For details about this, please see the document *ATSC Digital Television Standard: Part 4:2009*.

sequenceNo
This is the number of sequence information. This variable is increased by 1 when VPU detects change of sequence.

numReorderFrames
This is num_reorder_frames in in H.264/AVC VUI syntax (CODA7Q only)

DecOutputExtData

```
typedef struct {
    Uint32          userDataHeader;
    Uint32          userDataNum;
    Uint32          userDataSize;
    Uint32          userDataBufFull;
    Uint32          activeFormat;
} DecOutputExtData;
```

Description

The data structure to get result information from decoding a frame.

userDataHeader
This variable indicates which userdata is reported by VPU. (WAVE only) When this variable is not zero, each bit corresponds to the H265_USERDATA_FLAG_XXX.

```
// H265 USER_DATA(SPS & SEI) ENABLE FLAG
#define H265_USERDATA_FLAG_RESERVED_0           (0)
#define H265_USERDATA_FLAG_RESERVED_1           (1)
#define H265_USERDATA_FLAG_VUI                  (2)
#define H265_USERDATA_FLAG_RESERVED_3           (3)
#define H265_USERDATA_FLAG_PIC_TIMING           (4)
#define H265_USERDATA_FLAG_ITU_T_T35_PRE        (5)
#define H265_USERDATA_FLAG_UNREGISTERED_PRE     (6)
#define H265_USERDATA_FLAG_ITU_T_T35_SUF        (7)
#define H265_USERDATA_FLAG_UNREGISTERED_SUF     (8)
#define H265_USERDATA_FLAG_RESERVED_9           (9)
#define H265_USERDATA_FLAG_MASTERING_COLOR_VOL  (10)
#define H265_USERDATA_FLAG_CHROMA_RESAMPLING_FILTER_HINT  (11)
#define H265_USERDATA_FLAG_KNEE_FUNCTION_INFO   (12)
```

Userdata are written from the memory address specified to SET_ADDR_REP_USERDATA, and userdata consists of two parts, header (offset and size) and userdata as shown below.

```
-------------------------------------
| offset_00(32bit) | size_00(32bit) |
| offset_01(32bit) | size_01(32bit) |
|                  ...              | header
|                  ...              |
| offset_31(32bit) | size_31(32bit) |
-------------------------------------
|                                   | data
|                                   |
```

userDataNum
This is the number of user data.

userDataSize
This is the size of user data.

userDataBufFull
When userDataEnable is enabled, decoder reports frame buffer status into the userDataBufAddr and userDataSize in byte size. When user data report mode is 1 and the user data

size is bigger than the user data buffer size, VPU reports user data as much as buffer size, skips the remainings and sets userDataBufFull.

activeFormat
active_format (4bit syntax value) in AFD user data. The default value is 0000b. This is valid only for H.264/AVC and MPEG2 stream.

Vp8PicInfo

```
typedef struct {
    unsigned showFrame      : 1;
    unsigned versionNumber  : 3;
    unsigned refIdxLast     : 8;
    unsigned refIdxAltr     : 8;
    unsigned refIdxGold     : 8;
} Vp8PicInfo;
```

Description

This is a data structure for VP8 specific hearder information and reference frame indices. Only VP8 decoder returns this structure after decoding a frame.

showFrame
This flag is the frame header syntax, meaning whether the current decoded frame is displayable or not. It is 0 when the current frame is not for display, and 1 when the current frame is for display.

versionNumber
This is the VP8 profile version number information in the frame header. The version number enables or disables certain features in bitstream. It can be defined with one of the four different profiles, 0 to 3 and each of them indicates different decoding complexity.

refIdxLast
This is the frame buffer index for the Last reference frame. This field is valid only for next inter frame decoding.

refIdxAltr
This is the frame buffer index for the altref(Alternative Reference) reference frame. This field is valid only for next inter frame decoding.

refIdxGold
This is the frame buffer index for the Golden reference frame. This field is valid only for next inter frame decoding.

MvcPicInfo

```
typedef struct {
    int viewIdxDisplay;
    int viewIdxDecoded;
} MvcPicInfo;
```

Description

This is a data structure for MVC specific picture information. Only MVC decoder returns this structure after decoding a frame.

viewIdxDisplay
This is view index order of display frame buffer corresponding to indexFrameDisplay of DecOutputInfo structure.

viewIdxDecoded
This is view index order of decoded frame buffer corresponding to indexFrameDecoded of DecOutputInfo structure.

AvcFpaSei

```
typedef struct {
    unsigned exist;
    unsigned framePackingArrangementId;
    unsigned framePackingArrangementCancelFlag;
    unsigned quincunxSamplingFlag;
    unsigned spatialFlippingFlag;
    unsigned frame0FlippedFlag;
    unsigned fieldViewsFlag;
    unsigned currentFrameIsFrame0Flag;
    unsigned frame0SelfContainedFlag;
    unsigned frame1SelfContainedFlag;
    unsigned framePackingArrangementExtensionFlag;
    unsigned framePackingArrangementType;
    unsigned contentInterpretationType;
    unsigned frame0GridPositionX;
    unsigned frame0GridPositionY;
    unsigned frame1GridPositionX;
    unsigned frame1GridPositionY;
    unsigned framePackingArrangementRepetitionPeriod;
} AvcFpaSei;
```

Description

This is a data structure for H.264/AVC FPA(Frame Packing Arrangement) SEI. For detailed information, refer to *ISO/IEC 14496-10 D.2.25 Frame packing arrangement SEI message semantics*.

exist
This is a flag to indicate whether H.264/AVC FPA SEI exists or not.
- 0 : H.264/AVC FPA SEI does not exist.
- 1 : H.264/AVC FPA SEI exists.

framePackingArrangementId
0 ~ 2^32-1 : An identifying number that may be used to identify the usage of the frame packing arrangement SEI message.

framePackingArrangementCancelFlag
1 indicates that the frame packing arrangement SEI message cancels the persistence of any previous frame packing arrangement SEI message in output order.

quincunxSamplingFlag
It indicates whether each color component plane of each constituent frame is quincunx sampled.

spatialFlippingFlag
It indicates that one of the two constituent frames is spatially flipped.

frame0FlippedFlag
It indicates which one of the two constituent frames is flipped.

fieldViewsFlag
1 indicates that all pictures in the current coded video sequence are coded as complementary field pairs.

currentFrameIsFrame0Flag
It indicates the current decoded frame and the next decoded frame in output order.

frame0SelfContainedFlag
It indicates whether inter prediction operations within the decoding process for the samples of constituent frame 0 of the coded video sequence refer to samples of any constituent frame 1.

frame1SelfContainedFlag
It indicates whether inter prediction operations within the decoding process for the samples of constituent frame 1 of the coded video sequence refer to samples of any constituent frame 0.

framePackingArrangementExtensionFlag
0 indicates that no additional data follows within the frame packing arrangement SEI message.

framePackingArrangementType
The type of packing arrangement of the frames

contentInterpretationType
It indicates the intended interpretation of the constituent frames.

frame0GridPositionX
It specifies the horizontal location of the upper left sample of constituent frame 0 to the right of the spatial reference point.

frame0GridPositionY
It specifies the vertical location of the upper left sample of constituent frame 0 below the spatial reference point.

frame1GridPositionX
It specifies the horizontal location of the upper left sample of constituent frame 1 to the right of the spatial reference point.

frame1GridPositionY
It specifies the vertical location of the upper left sample of constituent frame 1 below the spatial reference point.

framePackingArrangementRepetitionPeriod
It indicates persistence of the frame packing arrangement SEI message.

AvcHrdInfo

```
typedef struct {
    int cpbMinus1;
    int vclHrdParamFlag;
    int nalHrdParamFlag;
} AvcHrdInfo;
```

Description

This is a data structure for H.264/AVC specific picture information. (H.264/AVC decoder only) VPU returns this structure after decoding a frame. For detailed information, refer to *ISO/IEC 14496-10 E.1 VUI syntax*.

cpbMinus1
cpb_cnt_minus1

vclHrdParamFlag
vcl_hrd_parameters_present_flag

nalHrdParamFlag
nal_hrd_parameters_present_flag

AvcRpSei

```
typedef struct {
    unsigned exist;
    int recoveryFrameCnt;
    int exactMatchFlag;
    int brokenLinkFlag;
    int changingSliceGroupIdc;
} AvcRpSei;
```

Description

This is a data structure for H.264/AVC specific picture information. (H.264/AVC decoder only) VPU returns this structure after decoding a frame. For detailed information, refer to *ISO/IEC 14496-10 D.1.7 Recovery point SEI message syntax*.

exist
This is a flag to indicate whether H.264/AVC RP SEI exists or not.
- 0 : H.264/AVC RP SEI does not exist.
- 1 : H.264/AVC RP SEI exists.

recoveryFrameCnt
recovery_frame_cnt

exactMatchFlag
exact_match_flag

brokenLinkFlag
broken_link_flag

changingSliceGroupIdc
changing_slice_group_idc

H265RpSei

```
typedef struct {
    unsigned exist;
    int recoveryPocCnt;
    int exactMatchFlag;
    int brokenLinkFlag;
} H265RpSei;
```

Description

This is a data structure for H.265/HEVC specific picture information. (H.265/HEVC decoder only) VPU returns this structure after decoding a frame.

exist
This is a flag to indicate whether H.265/HEVC RP SEI exists or not.
- 0 : H.265/HEVC RP SEI does not exist.
- 1 : H.265/HEVC RP SEI exists.

recoveryPocCnt
recovery_poc_cnt

exactMatchFlag
exact_match_flag

brokenLinkFlag
broken_link_flag

H265Info

```
typedef struct {
    int decodedPOC;
    int displayPOC;
    int temporalId;
} H265Info;
```

Description

This is a data structure that H.265/HEVC decoder returns for reporting POC (Picture Order Count).

decodedPOC
A POC value of picture that has currently been decoded and with decoded index. When indexFrameDecoded is -1, it returns -1.

displayPOC
A POC value of picture with display index. When indexFrameDisplay is -1, it returns -1.

temporalId
A temporal ID of the picture

DecOutputInfo

```
typedef struct {
    int indexFrameDisplay;
    int indexFrameDisplayForTiled;
    int indexFrameDecoded;
    int indexInterFrameDecoded;
    int indexFrameDecodedForTiled;
    int nalType;
    int picType;
    int picTypeFirst;
    int numOfErrMBs;
    int numOfTotMBs;
    int numOfErrMBsInDisplay;
    int numOfTotMBsInDisplay;
    BOOL refMissingFrameFlag;
    int notSufficientSliceBuffer;
    int notSufficientPsBuffer;
    int decodingSuccess;
    int interlacedFrame;
    int chunkReuseRequired;
    VpuRect rcDisplay;
    int dispPicWidth;
    int dispPicHeight;
    VpuRect rcDecoded;
    int decPicWidth;
    int decPicHeight;
    int aspectRateInfo;
    int fRateNumerator;
    int fRateDenominator;
    Vp8ScaleInfo vp8ScaleInfo;
```

```
    Vp8PicInfo vp8PicInfo;
    MvcPicInfo mvcPicInfo;
    AvcFpaSei avcFpaSei;
    AvcHrdInfo avcHrdInfo;
    AvcVuiInfo avcVuiInfo;
    H265Info h265Info;
    int vc1NpfFieldInfo;
    int mp2DispVerSize;
    int mp2DispHorSize;
    int mp2NpfFieldInfo;
    MP2BarDataInfo mp2BardataInfo;
    MP2PicDispExtInfo mp2PicDispExtInfo;
    AvcRpSei avcRpSei;
    H265RpSei h265RpSei;
    int avcNpfFieldInfo;
    int avcPocPic;
    int avcPocTop;
    int avcPocBot;
    Uint32 avcTemporalId;
    // Report Information
    int pictureStructure;
    int topFieldFirst;
    int repeatFirstField;
    int progressiveFrame;
    int fieldSequence;
    int frameDct;
    int nalRefIdc;
    int decFrameInfo;
    int picStrPresent;
    int picTimingStruct;
    int progressiveSequence;
    int mp4TimeIncrement;
    int mp4ModuloTimeBase;
    DecOutputExtData decOutputExtData;
    int consumedByte;
    int rdPtr;
    int wrPtr;
    PhysicalAddress bytePosFrameStart;
    PhysicalAddress bytePosFrameEnd;
    FrameBuffer dispFrame;
    int frameDisplayFlag;
    int sequenceChanged;
    // CODA9: [0]   1 - sequence changed
    // WAVEX: [5]   1 - H.265 profile changed
    //        [16]  1 - resolution changed
    //        [19]  1 - number of DPB changed

    int streamEndFlag;
    int frameCycle;
    int errorReason;
    int errorReasonExt;
    int warnInfo;
    Uint32 sequenceNo;
    int rvTr;
    int rvTrB;
    int indexFramePrescan;
#ifdef SUPPORT_REF_FLAG_REPORT
    int frameReferenceFlag[31];
#endif

    Int32   seekCycle;
```

```
    Int32   parseCycle;
    Int32   decodeCycle;
    Int32 ctuSize;
    Int32 outputFlag;
} DecOutputInfo;
```

Description

The data structure to get result information from decoding a frame.

indexFrameDisplay
This is a frame buffer index for the picture to be displayed at the moment among frame buffers which are registered using VPU_DecRegisterFrameBuffer(). Frame data to be displayed are stored into the frame buffer with this index. When there is no display delay, this index is always the same with indexFrameDecoded. However, if display delay does exist for display reordering in AVC or B-frames in VC1), this index might be different with indexFrameDecoded. By checking this index, HOST application can easily know whether sequence decoding has been finished or not.

- -3(0xFFFD) : It is when decoder skip option is on.
- -2(0xFFFE) : It is when decoder have decoded sequence but cannot give a display output due to reordering.
- -1(0xFFFF) : It is when there is no more output for display at the end of sequence decoding.

indexFrameDisplayForTiled
In case of WTL mode, this index indicates a display index of tiled or compressed framebuffer.

indexFrameDecoded
This is a frame buffer index of decoded picture among frame buffers which were registered using VPU_DecRegisterFrameBuffer(). The currently decoded frame is stored into the frame buffer specified by this index.

- -2 : It indicates that no decoded output is generated because decoder meets EOS (End Of Sequence) or skip.
- -1 : It indicates that decoder fails to decode a picture because there is no available frame buffer.

indexInterFrameDecoded
In case of VP9 codec, this indicates an index of the frame buffer to reallocate for the next frame's decoding. VPU returns this information when detecting change of the inter-frame resolution.

indexFrameDecodedForTiled
In case of WTL mode, this indicates a decoded index of tiled or compressed framebuffer.

nalType
This is nal Type of decoded picture. Please refer to nal_unit_type in Table 7-1 - NAL unit type codes and NAL unit type classes in H.265/HEVC specification. (WAVE only)

picType
This is the picture type of decoded picture. It reports the picture type of bottom field for interlaced stream. *the section called “PicType”*.

picTypeFirst
This is only valid in interlaced mode and indicates the picture type of the top field.

numOfErrMBs
This is the number of error coded unit in a decoded picture.

numOfTotMBs
This is the number of coded unit in a decoded picture.

numOfErrMBsInDisplay
This is the number of error coded unit in a picture mapped to indexFrameDisplay.

numOfTotMBsInDisplay
This is the number of coded unit in a picture mapped to indexFrameDisplay.

refMissingFrameFlag
This indicates that the current frame's references are missing in decoding. (WAVE only)

notSufficientSliceBuffer
This is a flag which represents whether slice save buffer is not sufficient to decode the current picture. VPU might not get the last part of the current picture stream due to buffer overflow, which leads to macroblock errors. HOST application can continue decoding the remaining pictures of the current bitstream without closing the current instance, even though several pictures could be error-corrupted. (H.264/AVC BP only)

notSufficientPsBuffer
This is a flag which represents whether PS (SPS/PPS) save buffer is not sufficient to decode the current picture. VPU might not get the last part of the current picture stream due to buffer overflow. HOST application must close the current instance, since the following picture streams cannot be decoded properly for loss of SPS/PPS data. (H.264/AVC only)

decodingSuccess
This variable indicates whether decoding process was finished completely or not. If stream has error in the picture header syntax or has the first slice header syntax of H.264/AVC stream, VPU returns 0 without proceeding MB decode routine.
- 0 : It indicates incomplete finish of decoding process
- 1 : It indicates complete finish of decoding process

interlacedFrame
- 0 : A progressive frame which consists of one picture
- 1 : An interlaced frame which consists of two fields

chunkReuseRequired
This is a flag which represents whether chunk in bitstream buffer should be reused or not, even after VPU_DecStartOneFrame() is executed. This flag is meaningful when bitstream buffer operates in PicEnd mode. In that mode, VPU consumes all the bitstream in bitstream buffer for the current VPU_DecStartOneFrame() in assumption that one chunk is one frame. However, there might be a few cases that chunk needs to be reused such as the following:

- DivX or XivD stream : One chunk can contain P frame and B frame to reduce display delay. In that case after decoding P frame, this flag is set to 1. HOST application should try decoding with the rest of chunk data to get B frame.
- H.264/AVC NPF stream : After the first field has been decoded, this flag is set to 1. HOST application should check if the next field is NPF or not.
- No DPB available: It is when VPU is not able to consume chunk with no frame buffers available at the moment. Thus, the whole chunk should be provided again.

rcDisplay
This field reports the rectangular region in pixel unit after decoding one frame - the region of indexFrameDisplay frame buffer.

dispPicWidth
This field reports the width of a picture to be displayed in pixel unit after decoding one frame - width of indexFrameDisplay frame bufffer.

dispPicHeight
This field reports the height of a picture to be displayed in pixel unit after decoding one frame - height of indexFrameDisplay frame bufffer.

rcDecoded
This field reports the rectangular region in pixel unit after decoding one frame - the region of indexFrameDecoded frame buffer.

decPicWidth
This field reports the width of a decoded picture in pixel unit after decoding one frame - width of indexFrameDecoded frame bufffer.

decPicHeight
This field reports the height of a decoded picture in pixel unit after decoding one frame - height of indexFrameDecoded frame bufffer.

aspectRateInfo
This is aspect ratio information for each standard. Refer to aspectRateInfo of *the section called "DecInitialInfo"*.

fRateNumerator
The numerator part of frame rate fraction. Note that the meaning of this flag can vary by codec standards. For details about this, please refer to *Appendix: FRAME RATE NUMERATORS in programmer's guide*.

fRateDenominator
The denominator part of frame rate fraction. Note that the meaning of this flag can vary by codec standards. For details about this, please refer to *Appendix: FRAME RATE DENOMINATORS in programmer's guide*.

vp8ScaleInfo
This is VP8 upsampling information. Refer to *the section called "Vp8ScaleInfo"*.

vp8PicInfo
This is VP8 frame header information. Refer to *the section called "Vp8PicInfo"*.

mvcPicInfo
This is MVC related picture information. Refer to *the section called "MvcPicInfo"*.

avcFpaSei
This is H.264/AVC frame packing arrangement SEI information. Refer to *the section called "AvcFpaSei"*.

avcHrdInfo
This is H.264/AVC HRD information. Refer to *the section called "AvcHrdInfo"*.

avcVuiInfo
This is H.264/AVC VUI information. Refer to *the section called "AvcVuiInfo"*.

h265Info
This is H.265/HEVC picture information. Refer to *the section called "H265Info"*.

vc1NpfFieldInfo
This field is valid only for VC1 decoding. Field information of display frame index is returned on `indexFrameDisplay`.
- 0 : Paired fields
- 1 : Bottom (top-field missing)
- 2 : Top (bottom-field missing)

mp2DispVerSize
This is display_vertical_size syntax of sequence display extension in MPEG2 specification.

mp2DispHorSize
This is display_horizontal_size syntax of sequence display extension in MPEG2 specification.

mp2NpfFieldInfo
This field is valid only for MPEG2 decoding. Field information of display frame index is returned on `indexFrameDisplay`.
- 0 : Paired fields
- 1 : Bottom (top-field missing)
- 2 : Top (bottom-field missing)

mp2BardataInfo
This is bar information in MPEG2 user data. For details about this, please see the document *ATSC Digital Television Standard: Part 4:2009*.

mp2PicDispExtInfo
For meaning of each field, please see *the section called "MP2PicDispExtInfo"*.

avcRpSei
This is H.264/AVC recovery point SEI information. Refer to *the section called "AvcRpSei"*.

h265RpSei
This is H.265/HEVC recovery point SEI information. Refer to *the section called "H265RpSei"*.

avcNpfFieldInfo
This field is valid only for H.264/AVC decoding. Field information of display frame index is returned on `indexFrameDisplay`. Refer to the *the section called "AvcNpfFieldInfo"*.
- 0 : Paired fields
- 1 : Bottom (top-field missing)
- 2 : Top (bottom-field missing)

avcPocPic
This field reports the POC value of frame picture in case of H.264/AVC decoding.

avcPocTop
This field reports the POC value of top field picture in case of H.264/AVC decoding.

avcPocBot
This field reports the POC value of bottom field picture in case of H.264/AVC decoding.

avcTemporalId
This field reports the target temporal ID of current picture. This is valid in H.264/AVC decoder.

pictureStructure
This variable indicates that the decoded picture is progressive or interlaced picture. The value of pictureStructure is used as below.
- H.264/AVC : MBAFF
- VC1 : FCM
  - 0 : Progressive
  - 2 : Frame interlace
  - 3 : Field interlaced
- MPEG2 : picture structure
  - 1 : TopField
  - 2 : BotField

  - 3 : Frame
- MPEG4 : N/A
- Real Video : N/A
- H.265/HEVC : N/A

topFieldFirst
For decoded picture consisting of two fields, this variable reports

0 : VPU decodes the bottom field and then top field. @ 1 : VPU decodes the top field and then bottom field.

Regardless of this variable, VPU writes the decoded image of top field picture at each odd line and the decoded image of bottom field picture at each even line in frame buffer.

repeatFirstField
This variable indicates Repeat First Field that repeats to display the first field. This flag is valid for VC1, AVS, and MPEG2.

progressiveFrame
This variable indicates progressive_frame in MPEG2 picture coding extention or in AVS picture header. In the case of VC1, this variable means RPTFRM (Repeat Frame Count), which is used during display process.

fieldSequence
This variable indicates field_sequence in picture coding extention in MPEG2.

frameDct
This variable indicates frame_pred_frame_dct in sequence extension of MPEG2.

nalRefIdc
This variable indicates if the currently decoded frame is a reference frame or not. This flag is valid for H.264/AVC only.

decFrameInfo
- H.264/AVC, MPEG2, and VC1
  - 0 : The decoded frame has paired fields.
  - 1 : The decoded frame has a top-field missing.
  - 2 : The decoded frame has a bottom-field missing.

picStrPresent
It indicates pic_struct_present_flag in H.264/AVC pic_timing SEI.

picTimingStruct
It indicates pic_struct in H.264/AVC pic_timing SEI reporting. (Table D-1 in H.264/AVC specification.) If pic_timing SEI is not presented, pic_struct is inferred by the D.2.1. pic_struct part in H.264/AVC specification. This field is valid only for H.264/AVC decoding.

progressiveSequence
It indicates progressive_sequence in sequence extension of MPEG2.

mp4TimeIncrement
It indicates vop_time_increment_resolution in MPEG4 VOP syntax.

mp4ModuloTimeBase
It indicates modulo_time_base in MPEG4 VOP syntax.

decOutputExtData
The data structure to get additional information about a decoded frame. Refer to *the section called “DecOutputExtData”*.

consumedByte
The number of bytes that are consumed by VPU.

rdPtr
A stream buffer read pointer for the current decoder instance

wrPtr
A stream buffer write pointer for the current decoder instance

bytePosFrameStart
The start byte position of the current frame after decoding the frame for audio-to-video synchronization

H.265/HEVC or H.264/AVC decoder seeks only 3-byte start code (0x000001) while other decoders seek 4-byte start code(0x00000001).

bytePosFrameEnd
It indicates the end byte position of the current frame after decoding. This information helps audio-to-video synchronization.

dispFrame
It indicates the display frame buffer address and information. Refer to *the section called "FrameBuffer"*.

frameDisplayFlag
It reports a frame buffer flag to be displayed.

sequenceChanged
This variable reports that sequence has been changed while H.264/AVC stream decoding. If it is 1, HOST application can get the new sequence information by calling VPU_DecGetInitialInfo() or VPU_DecIssueSeqInit().

For H.265/HEVC decoder, each bit has a different meaning as follows.
- sequenceChanged[5] : It indicates that the profile_idc has been changed.
- sequenceChanged[16] : It indicates that the resolution has been changed.
- sequenceChanged[19] : It indicates that the required number of frame buffer has been changed.

streamEndFlag
This variable reports the status of end of stream flag. This information can be used for low delay decoding (CODA980 only).

frameCycle
This variable reports the cycle number of decoding one frame.

errorReason
This variable reports the error reason that occurs while decoding. For error description, please find the *Appendix: Error Definition* in the Programmer's Guide.

errorReasonExt
This variable reports the specific reason of error. For error description, please find the *Appendix: Error Definition* in the Programmer's Guide. (WAVE only)

sequenceNo
This variable increases by 1 whenever sequence changes. If it happens, HOST should call VPU_DecFrameBufferFlush() to get the decoded result that remains in the buffer in the form of DecOutputInfo array. HOST can recognize with this variable whether this frame is in the current sequence or in the previous sequence when it is displayed. (WAVE only)

rvTr
This variable reports RV timestamp for Ref frame.

rvTrB
This variable reports RV timestamp for B frame.

indexFramePrescan
This variable reports the result of pre-scan which is the start of decoding routine for DEC_PIC command. (WAVE4 only) In the prescan phase, VPU parses bitstream and pre-allocates frame buffers.
- -2 : It is when VPU prescanned bitstream(bitstream consumed), but a decode buffer was not allocated for the bitstream during pre-scan, since there was only header information.
- -1 : It is when VPU detected full of framebuffer while pre-scannig (bitstream not consumed).
- >= 0 : It indicates that prescan has been successfully done. This index is returned to a decoded index for the next decoding.

seekCycle
This variable reports the number of cycles in seeking phase on the command queue. (WAVE5 only)

parseCycle
This variable reports the number of cycles in prescan phase on the command queue. (WAVE5 only)

decodeCycle
This variable reports the number of cycles in decoding phase on the command queue. (WAVE5 only)

ctuSize
A CTU size (only for WAVE series)
- 16 : CTU16x16
- 32 : CTU32x32
- 64 : CTU64x64

outputFlag
This variable reports whether the current frame is bumped out or not. (WAVE5 only)

DecGetFramebufInfo

```
typedef struct {
    vpu_buffer_t  vbFrame;
    vpu_buffer_t  vbWTL;
    vpu_buffer_t  vbFbcYTbl[MAX_REG_FRAME];
    vpu_buffer_t  vbFbcCTbl[MAX_REG_FRAME];
    vpu_buffer_t  vbMvCol[MAX_REG_FRAME];
    FrameBuffer   framebufPool[64];
} DecGetFramebufInfo;
```

Description

This is a data structure of frame buffer information. It is used for parameter when host issues DEC_GET_FRAMEBUF_INFO of *the section called "VPU_DecGiveCommand()"*.

vbFrame
The information of frame buffer where compressed frame is saved

vbWTL
The information of frame buffer where decoded, uncompressed frame is saved with linear format if WTL is on

vbFbcYTbl
The information of frame buffer to save luma offset table of compressed frame

vbFbcCTbl
The information of frame buffer to save chroma offset table of compressed frame

vbMvCol
The information of frame buffer to save motion vector collocated buffer

framebufPool
This is an array of *the section called "FrameBuffer"* which contains the information of each frame buffer. When WTL is enabled, the number of framebufPool would be [number of compressed frame buffer] x 2, and the starting index of frame buffer for WTL is framebufPool[number of compressed frame buffer].

DecQueueStatusInfo

```
typedef struct {
    Uint32  instanceQueueCount;
    Uint32  totalQueueCount;
} DecQueueStatusInfo;
```

Description

This is a data structure of queue command information. It is used for parameter when host issues DEC_GET_QUEUE_STATUS of *the section called "VPU_DecGiveCommand()"*. (WAVE5 only)

instanceQueueCount
This variable indicates the number of queued commands of the instance.

totalQueueCount
This variable indicates the number of queued commands of all instances.

EncMp4Param

```
typedef struct {
    int mp4DataPartitionEnable;
    int mp4ReversibleVlcEnable;
    int mp4IntraDcVlcThr;
    int mp4HecEnable;
    int mp4Verid;
} EncMp4Param;
```

Description

This is a data structure for configuring MPEG4-specific parameters in encoder applications. (CODA9 encoder only)

mp4DataPartitionEnable
It encodes with MPEG4 data_partitioned coding tool.

mp4ReversibleVlcEnable
It encodes with MPEG4 reversible_vlc coding tool.

mp4IntraDcVlcThr
It encodes with MPEG4 intra_dc_vlc_thr coding tool. The valid range is 0 - 7.

mp4HecEnable
It encodes with MPEG4 HEC (Header Extension Code) coding tool.

mp4Verid
It encodes with value of MPEG4 part 2 standard version ID. Version 1 and version 2 are allowed.

EncH263Param

```
typedef struct {
    int h263AnnexIEnable;
    int h263AnnexJEnable;
    int h263AnnexKEnable;
    int h263AnnexTEnable;
} EncH263Param;
```

Description

This is a data structure for configuring H.263-specific parameters in encoder applications. (CODA9 encoder only)

h263AnnexIEnable
It encodes with H.263 Annex I - Advanced INTRA Coding mode.

h263AnnexJEnable
It encodes with H.263 Annex J - Deblocking Filter mode.

h263AnnexKEnable
It encodes with H.263 Annex K - Slice Structured mode.

h263AnnexTEnable
It encodes with H.263 Annex T - Modified Quantization mode.

CustomGopPicParam

```
typedef struct {
    int picType;
    int pocOffset;
    int picQp;
    int numRefPicL0;
    int refPocL0;
    int refPocL1;
    int temporalId;
} CustomGopPicParam;
```

Description

This is a data structure for custom GOP parameters of the given picture. (WAVE encoder only)

picType
A picture type of Nth picture in the custom GOP

pocOffset
A POC of Nth picture in the custom GOP

picQp
A quantization parameter of Nth picture in the custom GOP

refPocL0
A POC of reference L0 of Nth picture in the custom GOP

refPocL1
A POC of reference L1 of Nth picture in the custom GOP

temporalId
A temporal ID of Nth picture in the custom GOP

CustomGopParam

```
typedef struct {
    int customGopSize;
    int useDeriveLambdaWeight;
    CustomGopPicParam picParam[MAX_GOP_NUM];
    int gopPicLambda[MAX_GOP_NUM];
    int enTemporalLayerQp;
    int tidQp0;
    int tidQp1;
    int tidQp2;
    int tidPeriod0;
} CustomGopParam;
```

Description

This is a data structure for custom GOP parameters. (WAVE encoder only)

customGopSize
The size of custom GOP (0~8)

useDeriveLambdaWeight
It internally derives a lamda weight instead of using the given lamda weight.

picParam
Picture parameters of Nth picture in custom GOP

gopPicLambda
A lamda weight of Nth picture in custom GOP

enTemporalLayerQp
Enable QP setting of each temporal layers

tidQp0
Specifies a QP value of temporal layer 0

tidQp1
Specifies a QP value of temporal layer 1

tidQp2
Specifies a QP value of temporal layer 2

tidPeriod0
Specifies the period of temporal layer 0

HevcCtuOptParam

```
typedef struct {
    int roiEnable;
    int roiDeltaQp;
    int mapEndian;
```

```
    int mapStride;
    PhysicalAddress addrRoiCtuMap;
    PhysicalAddress addrCtuModeMap;
    int ctuModeEnable;
    PhysicalAddress addrCtuQpMap;
    int ctuQpEnable;
} HevcCtuOptParam;
```

Description

This is a data structure for setting CTU level options (ROI, CTU mode, CTU QP) in H.265/HEVC encoder (WAVE4 encoder only).

roiEnable
It enables ROI map. NOTE: It is valid when rcEnable is on.

roiDeltaQp
It specifies a delta QP that is used to calculate ROI QPs internally.

mapEndian
It specifies endianness of ROI CTU map. For the specific modes, refer to the EndianMode of *the section called "DecOpenParam"*.

mapStride
It specifies the stride of CTU-level ROI/mode/QP map. It should be set with the value as below.

```
(Width  + CTB_SIZE - 1) / CTB_SIZE
```

addrRoiCtuMap
The start buffer address of ROI map

ROI map holds importance levels for CTUs within a picture. The memory size is the number of CTUs of picture in bytes. For example, if there are 64 CTUs within a picture, the size of ROI map is 64 bytes. All CTUs have their ROI importance level (0 ~ 8 ; 1 byte) in raster order. A CTU with a high ROI importance level is encoded with a lower QP for higher quality. It should be given when roiEnable is 1.

addrCtuModeMap
The start buffer address of CTU mode map

The memory size is the number of CTUs of picture in bytes. For example if there are 64 CTUs within a picture, the size of CTU map is 64 bytes. It should be given when ctuModeEnable is 1.

The content of ctuModeMap directly is mapped to the coding mode which is used to encode the CTU. I.e, if (ctuModeMap[k] is 2), it means that CTU(k) is encoded with force intra.
- 0 : Normal
- 1 : CTU skip
- 2 : Force intra

ctuModeEnable
It enables CTU mode map that allows CTUs to be encoded with force intra or to be skipped.

addrCtuQpMap
The start buffer address of CTU qp map

The memory size is the number of CTUs of picture in bytes. For example if there are 64 CTUs within a picture, the size of CTU map is 64 bytes. It should be given when ctuQpEnable is 1.

The content of ctuQpMap directly is mapped to the QP used to encode the CTU. I.e, if (ctuQpMap[k] is 5), it means that CTU(k) is encoded with QP 5.

ctuQpEnable
It enables CTU QP map that allows CTUs to be encoded with the given QPs.

Note | rcEnable should be turned off for this, encoding with the given CTU QPs.

HevcCustomMapOpt

```
typedef struct {
    int roiAvgQp;
    int customRoiMapEnable;
    int customLambdaMapEnable;
    int customModeMapEnable;
    int customCoefDropEnable;
    PhysicalAddress addrCustomMap;
} HevcCustomMapOpt;
```

Description

This is a data structure for setting custom map options in H.265/HEVC encoder. (WAVE5 encoder only).

roiAvgQp
It sets an average QP of ROI map.

customRoiMapEnable
It enables ROI map.

customLambdaMapEnable
It enables custom lambda map.

customModeMapEnable
It enables to force CTU to be encoded with intra or to be skipped.

customCoefDropEnable
It enables to force all coefficients to be zero after TQ or not for each CTU (to be dropped).

addrCustomMap
The address of custom map

HevcVuiParam

```
typedef struct {
    Uint32 vuiParamFlags;
    Uint32 vuiAspectRatioIdc;
    Uint32 vuiSarSize;
    Uint32 vuiOverScanAppropriate;
    Uint32 videoSignal;
    Uint32 vuiChromaSampleLoc;
    Uint32 vuiDispWinLeftRight;
    Uint32 vuiDispWinTopBottom;
} HevcVuiParam;
```

Description

This is a data structure for setting VUI parameters in H.265/HEVC encoder. (WAVE only)

vuiParamFlags
It specifies vui_parameters_present_flag.

vuiAspectRatioIdc
It specifies aspect_ratio_idc.

vuiSarSize
It specifies sar_width and sar_height (only valid when aspect_ratio_idc is equal to 255).

vuiOverScanAppropriate
It specifies overscan_appropriate_flag.

videoSignal
It specifies video_signal_type_present_flag.

vuiChromaSampleLoc
It specifies chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field.

vuiDispWinLeftRight
It specifies def_disp_win_left_offset and def_disp_win_right_offset.

vuiDispWinTopBottom
It specifies def_disp_win_top_offset and def_disp_win_bottom_offset.

HevcSEIDataEnc

```
typedef struct {
    Uint32 prefixSeiNalEnable;
    Uint32 prefixSeiDataSize;
    Uint32 prefixSeiDataEncOrder;
    PhysicalAddress prefixSeiNalAddr;
    Uint32 suffixSeiNalEnable;
    Uint32 suffixSeiDataSize;
    Uint32 suffixSeiDataEncOrder;
    PhysicalAddress suffixSeiNalAddr;
} HevcSEIDataEnc;
```

Description

This is a data structure for setting SEI NALs in H.265/HEVC encoder.

prefixSeiNalEnable
It enables to encode the prefix SEI NAL which is given by HOST application.

prefixSeiDataSize
The total byte size of the prefix SEI

prefixSeiDataEncOrder
A flag whether to encode PREFIX_SEI_DATA with a picture of this command or with a source picture of the buffer at the moment
- 0 : encode PREFIX_SEI_DATA when the current input source picture is encoded.
- 1 : encode PREFIX_SEI_DATA at this command.

prefixSeiNalAddr
The start address of the total prefix SEI NALs to be encoded

suffixSeiNalEnable
It enables to encode the suffix SEI NAL which is given by HOST application.

suffixSeiDataSize
The total byte size of the suffix SEI

suffixSeiDataEncOrder

A flag whether to encode SUFFIX_SEI_DATA with a picture of this command or with a source picture of the buffer at the moment

- 0 : encode SUFFIX_SEI_DATA when the current input source picture is encoded.
- 1 : encode SUFFIX_SEI_DATA at this command.

suffixSeiNalAddr

The start address of the total suffix SEI NALs to be encoded

EncHevcParam

```
typedef struct {
    int profile;
    int level;
    int tier;
    int internalBitDepth;
    int chromaFormatIdc;
    int losslessEnable;
    int constIntraPredFlag;
    int gopPresetIdx;
    int decodingRefreshType;
    int intraQP;
    int intraPeriod;
    int confWinTop;
    int confWinBot;
    int confWinLeft;
    int confWinRight;
    int independSliceMode;
    int independSliceModeArg;
    int dependSliceMode;
    int dependSliceModeArg;
    int intraRefreshMode;
    int intraRefreshArg;
    int useRecommendEncParam;
    int scalingListEnable;
    int cuSizeMode;
    int tmvpEnable;
    int wppEnable;
    int maxNumMerge;
    int dynamicMerge8x8Enable;
    int dynamicMerge16x16Enable;
    int dynamicMerge32x32Enable;
    int disableDeblk;
    int lfCrossSliceBoundaryEnable;
    int betaOffsetDiv2;
    int tcOffsetDiv2;
    int skipIntraTrans;
    int saoEnable;
    int intraInInterSliceEnable;
    int intraNxNEnable;
    int intraQpOffset;
    int initBufLevelx8;
    int bitAllocMode;
    int fixedBitRatio[MAX_GOP_NUM];
    int cuLevelRCEnable;
    int hvsQPEnable;
    int hvsQpScaleEnable;
    int hvsQpScale;
    int hvsMaxDeltaQp;
    int minQp;
```

```
    int maxQp;
    int maxDeltaQp;
    int transRate;
    // CUSTOM_GOP
    CustomGopParam gopParam;
    HevcCtuOptParam ctuOptParam;
    HevcVuiParam vuiParam;
    Uint32 numUnitsInTick;
    Uint32 timeScale;
    Uint32 numTicksPocDiffOne;
    int chromaCbQpOffset;
    int chromaCrQpOffset;
    int initialRcQp;
    Uint32  nrYEnable;
    Uint32  nrCbEnable;
    Uint32  nrCrEnable;
    Uint32  nrNoiseEstEnable;
    Uint32  nrNoiseSigmaY;
    Uint32  nrNoiseSigmaCb;
    Uint32  nrNoiseSigmaCr;
    // ENC_NR_WEIGHT
    Uint32  nrIntraWeightY;
    Uint32  nrIntraWeightCb;
    Uint32  nrIntraWeightCr;
    Uint32  nrInterWeightY;
    Uint32  nrInterWeightCb;
    Uint32  nrInterWeightCr;
    // ENC_INTRA_MIN_MAX_QP
    Uint32 intraMinQp;
    Uint32 intraMaxQp;
    Uint32 enableAFBCD;
    Uint32 useLongTerm;
    int    forcedIdrHeaderEnable;
    // newly added for WAVE520
//#ifdef WAVE520
    Uint32 monochromeEnable;
    Uint32 strongIntraSmoothEnable;

    Uint32 weightPredEnable;
    Uint32 bgDetectEnable;
    Uint32 bgThrDiff;
    Uint32 bgThrMeanDiff;
    Uint32 bgLambdaQp;
    int    bgDeltaQp;
    Uint32 tileNumCols;
    Uint32 tileNumRows;
    Uint32 tileUniformSpaceEnable;

    Uint32 customLambdaEnable;
    Uint32 customMDEnable;
    int    pu04DeltaRate;
    int    pu08DeltaRate;
    int    pu16DeltaRate;
    int    pu32DeltaRate;
    int    pu04IntraPlanarDeltaRate;
    int    pu04IntraDcDeltaRate;
    int    pu04IntraAngleDeltaRate;
    int    pu08IntraPlanarDeltaRate;
    int    pu08IntraDcDeltaRate;
    int    pu08IntraAngleDeltaRate;
    int    pu16IntraPlanarDeltaRate;
```

```
    int    pu16IntraDcDeltaRate;
    int    pu16IntraAngleDeltaRate;
    int    pu32IntraPlanarDeltaRate;
    int    pu32IntraDcDeltaRate;
    int    pu32IntraAngleDeltaRate;
    int    cu08IntraDeltaRate;
    int    cu08InterDeltaRate;
    int    cu08MergeDeltaRate;
    int    cu16IntraDeltaRate;
    int    cu16InterDeltaRate;
    int    cu16MergeDeltaRate;
    int    cu32IntraDeltaRate;
    int    cu32InterDeltaRate;
    int    cu32MergeDeltaRate;
    int    coefClearDisable;
    int    minQpI;
    int    maxQpI;
    int    minQpP;
    int    maxQpP;
    int    minQpB;
    int    maxQpB;
    PhysicalAddress customLambdaAddr;
    PhysicalAddress userScalingListAddr;
//#endif

    // for H.264 on WAVE
    int avcIdrPeriod;
    int rdoSkip;
    int lambdaScalingEnable;
    int transform8x8Enable;
    int avcSliceMode;
    int avcSliceArg;
    int intraMbRefreshMode;
    int intraMbRefreshArg;
    int mbLevelRcEnable;
    int entropyCodingMode;

    int s2fmeDisable;
    Uint32 rcWeightParam;
    Uint32 rcWeightBuf;
}EncHevcParam;
```


Description

This is a data structure for H.265/HEVC encoder parameters.

profile
A profile indicator
- 1 : main
- 2 : main10

level
A level indicator (level * 10)

tier
A tier indicator
- 0 : Main
- 1 : High

internalBitDepth
A bit-depth (8bit or 10bit) which VPU internally uses for encoding

VPU encodes with internalBitDepth instead of InputBitDepth. For example, if InputBitDepth is 8 and InternalBitDepth is 10, VPU converts the 8-bit source frames into 10-bit ones and then encodes them.

chromaFormatIdc
A chroma format indicator (0 for 4:2:0)

losslessEnable
It enables lossless coding.

constIntraPredFlag
It enables constrained intra prediction.

gopPresetIdx
A GOP structure preset option
- 0 : Custom GOP
- Other values : *the section called "GOP_PRESET_IDX"*

decodingRefreshType
The type of I picture to be inserted at every intraPeriod
- 0 : Non-IRAP
- 1 : CRA
- 2 : IDR

intraQP
A quantization parameter of intra picture

intraPeriod
A period of intra picture in GOP size

confWinTop
A top offset of conformance window

confWinBot
A bottom offset of conformance window

confWinLeft
A left offset of conformance window

confWinRight
A right offset of conformance window

independSliceMode
A slice mode for independent slice
- 0 : No multi-slice
- 1 : Slice in CTU number

independSliceModeArg
The number of CTU for a slice when independSliceMode is set with 1

dependSliceMode
A slice mode for dependent slice
- 0 : No multi-slice
- 1 : Slice in CTU number
- 2 : Slice in number of byte

dependSliceModeArg
The number of CTU or bytes for a slice when dependSliceMode is set with 1 or 2

intraRefreshMode
An intra refresh mode

- 0 : No intra refresh
- 1 : Row
- 2 : Column
- 3 : Step size in CTU

intraRefreshArg
The number of CTU (only valid when intraRefreshMode is 3.)

useRecommendEncParam
It uses one of the recommended encoder parameter presets.
- 0 : Custom
- 1 : Recommend encoder parameters (slow encoding speed, highest picture quality)
- 2 : Boost mode (normal encoding speed, moderate picture quality)
- 3 : Fast mode (fast encoding speed, low picture quality)

scalingListEnable
It enables a scaling list.

cuSizeMode
It enables CU(Coding Unit) size to be used in encoding process. Host application can also select multiple CU sizes.
- 3'b001 : 8x8
- 3'b010 : 16x16
- 3'b100 : 32x32

tmvpEnable
It enables temporal motion vector prediction.

wppEnable
It enables WPP (Wave-front Parallel Processing). WPP is unsupported in ring buffer mode of bitstream buffer.

maxNumMerge
Maximum number of merge candidates (0~2)

dynamicMerge8x8Enable
It enables dynamic merge 8x8 candidates.

dynamicMerge16x16Enable
It enables dynamic merge 16x16 candidates.

dynamicMerge32x32Enable
It enables dynamic merge 32x32 candidates.

disableDeblk
It disables in-loop deblocking filtering.

lfCrossSliceBoundaryEnable
It enables filtering across slice boundaries for in-loop deblocking.

betaOffsetDiv2
It enables BetaOffsetDiv2 for deblocking filter.

tcOffsetDiv2
It enables TcOffsetDiv3 for deblocking filter.

skipIntraTrans
It enables transform skip for an intra CU.

saoEnable
It enables SAO (Sample Adaptive Offset).

intraInInterSliceEnable
It enables to make intra CUs in an inter slice.

intraNxNEnable
It enables intra NxN PUs.

intraQpOffset
It specifies an intra QP offset relative to an inter QP. It is only valid when RateControl is enabled.

initBufLevelx8
It specifies encoder initial delay. It is only valid when RateControl is enabled.

```
encoder initial delay = InitialDelay * InitBufLevelx8 / 8
```

bitAllocMode
It specifies picture bits allocation mode. It is only valid when RateControl is enabled and GOP size is larger than 1.
- 0 : More referenced pictures have better quality than less referenced pictures.
- 1 : All pictures in a GOP have similar image quality.
- 2 : Each picture bits in a GOP is allocated according to FixedRatioN.

fixedBitRatio
A fixed bit ratio (1 ~ 255) for each picture of GOP's bit allocation
- N = 0 ~ (MAX_GOP_SIZE - 1)
- MAX_GOP_SIZE = 8

For instance when MAX_GOP_SIZE is 3, FixedBitRatio0, FixedBitRatio1, and FixedBitRatio2 can be set as 2, 1, and 1 repsectively for the fixed bit ratio 2:1:1. This is only valid when BitAllocMode is 2.

cuLevelRCEnable
It enable CU level rate control.

hvsQPEnable
It enable CU QP adjustment for subjective quality enhancement.

hvsQpScaleEnable
It enable QP scaling factor for CU QP adjustment when hvsQPEnable is 1.

hvsQpScale
A QP scaling factor for CU QP adjustment when hvcQpenable is 1

hvsMaxDeltaQp
A maximum delta QP for HVS

minQp
A minimum QP for rate control

maxQp
A maximum QP for rate control

maxDeltaQp
A maximum delta QP for rate control

transRate
A peak transmission bitrate in bps

gopParam
the section called “CustomGopParam”

ctuOptParam
the section called "HevcCtuOptParam"

vuiParam
the section called "HevcVuiParam"

numUnitsInTick
It specifies the number of time units of a clock operating at the frequency time_scale Hz.

timeScale
It specifies the number of time units that pass in one second.

numTicksPocDiffOne
It specifies the number of clock ticks corresponding to a difference of picture order count values equal to 1.

chromaCbQpOffset
The value of chroma(Cb) QP offset (only for WAVE420L)

chromaCrQpOffset
The value of chroma(Cr) QP offset (only for WAVE420L)

initialRcQp
The value of initial QP by HOST application. This value is meaningless if INITIAL_RC_QP is 63. (only for WAVE420L)

nrYEnable
It enables noise reduction algorithm to Y component.

nrCbEnable
It enables noise reduction algorithm to Cb component.

nrCrEnable
It enables noise reduction algorithm to Cr component.

nrNoiseEstEnable
It enables noise estimation for reduction. When this is disabled, noise estimation is carried out ouside VPU.

nrNoiseSigmaY
It specifies Y noise standard deviation if no use of noise estimation (nrNoiseEstEnable is 0).

nrNoiseSigmaCb
It specifies Cb noise standard deviation if no use of noise estimation (nrNoiseEstEnable is 0).

nrNoiseSigmaCr
It specifies Cr noise standard deviation if no use of noise estimation (nrNoiseEstEnable is 0).

nrIntraWeightY
A weight to Y noise level for intra picture (0 ~ 31). nrIntraWeight/4 is multiplied to the noise level that has been estimated. This weight is put for intra frame to be filtered more strongly or more weakly than just with the estimated noise level.

nrIntraWeightCb
A weight to Cb noise level for intra picture (0 ~ 31)

nrIntraWeightCr
A weight to Cr noise level for intra picture (0 ~ 31)

nrInterWeightY
A weight to Y noise level for inter picture (0 ~ 31). nrInterWeight/4 is multiplied to the noise level that has been estimated. This weight is put for inter frame to be filtered more strongly or more weakly than just with the estimated noise level.

nrInterWeightCb
A weight to Cb noise level for inter picture (0 ~ 31)

nrInterWeightCr
A weight to Cr noise level for inter picture (0 ~ 31)

intraMinQp
It specifies a minimum QP for intra picture (0 ~ 51). It is only valid when RateControl is 1.

intraMaxQp
It specifies a maximum QP for intra picture (0 ~ 51). It is only valid when RateControl is 1.

enableAFBCD
It enables AFBCD function that is able to receive AFBC(ARM Frame Buffer Compression) format stream as source input and decode them.

useLongTerm
It enables long-term reference function.

forcedIdrHeaderEnable
It enables every IDR frame to include VPS/SPS/PPS.

minQpI
A minimum QP of I picture for rate control

maxQpI
A maximum QP of I picture for rate control

minQpP
A minimum QP of P picture for rate control

maxQpP
A maximum QP of P picture for rate control

minQpB
A minimum QP of B picture for rate control

maxQpB
A maximum QP of B picture for rate control

avcIdrPeriod
A period of IDR picture (0 ~ 1024). 0 - implies an infinite period

rdoSkip
It skips RDO(rate distortion optimization).

lambdaScalingEnable
It enables lambda scaling using custom GOP.

transform8x8Enable
It enables 8x8 intra prediction and 8x8 transform.

avcSliceMode
A slice mode for independent slice
- 0 : no multi-slice
- 1 : slice in MB number

avcSliceArg
The number of MB for a slice when avcSliceMode is set with 1

intraMbRefreshMode
An intra refresh mode
- 0 : no intra refresh
- 1 : row
- 2 : column
- 3 : step size in CTU

intraMbRefreshArg
It Specifies an intra MB refresh interval. Depending on intraMbRefreshMode, it can mean one of the followings.
- The number of consecutive MB rows for intraMbRefreshMode of 1
- The number of consecutive MB columns for intraMbRefreshMode of 2
- A step size in MB for intraMbRefreshMode of 3

mbLevelRcEnable
It enables MB-level rate control.

entropyCodingMode
It selects the entropy coding mode used in encoding process.

0 : CAVLC 1 : CABAC

s2fmeDisable
It disables s2me_fme (only for AVC encoder).

rcWeightParam
Adjusts the speed of updating parameters to the rate control model. If RcWeightFactor is set with a large value, the RC parameters are updated slowly. (Min: 1, Max: 31)

rcWeightBuf
It is the smoothing factor in the estimation. (Min: 1, Max: 255) This parameter indicates the speed of adjusting bitrate toward fullness of buffer as a reaction parameter. As it becomess larger, the bit-rate error promptly affects the target bit allocation of the following picture.

EncChangeParam

```
typedef struct {
    int changeParaMode;
    int enable_option;
    // ENC_PIC_PARAM_CHANGE
    int losslessEnable;
    int constIntraPredFlag;
    int chromaCbQpOffset;
    int chromaCrQpOffset;
    // ENC_INTRA_PARAM_CHANGE
    int decodingRefreshType;
    int intraPeriod;
    int intraQP;
    // ENC_CONF_WIN_TOP_BOT_CHANGE
    int confWinTop;
    int confWinBot;
    // ENC_CONF_WIN_LEFT_RIGHT_CHANGE
    int confWinLeft;
    int confWinRight;
    // ENC_FRAME_RATE_CHANGE
    int frameRate;
    // ENC_INDEPENDENT_SLICE_CHANGE
```

```
int independSliceMode;
int independSliceModeArg;
// ENC_DEPENDENT_SLICE_CHANGE
int dependSliceMode;
int dependSliceModeArg;
// ENC_INTRA_REFRESH_CHANGE
int intraRefreshMode;
int intraRefreshArg;
// ENC_PARAM_CHANGE
int useRecommendEncParam;
int scalingListEnable;
int cuSizeMode;
int tmvpEnable;
int wppEnable;
int maxNumMerge;
int dynamicMerge8x8Enable;
int dynamicMerge16x16Enable;
int dynamicMerge32x32Enable;
int disableDeblk;
int lfCrossSliceBoundaryEnable;
int betaOffsetDiv2;
int tcOffsetDiv2;
int skipIntraTrans;
int saoEnable;
int intraInInterSliceEnable;
int intraNxNEnable;
// ENC_RC_PARAM_CHANGE
int rcEnable;
int intraQpOffset;
int initBufLevelx8;
int bitAllocMode;
int fixedBitRatio[MAX_GOP_NUM];

int cuLevelRCEnable;
int hvsQPEnable;
int hvsQpScaleEnable;
int hvsQpScale;
int initialDelay;
Uint32 initialRcQp;
// ENC_RC_MIN_MAX_QP_CHANGE
int minQp;
int maxQp;
int maxDeltaQp;
// ENC_TARGET_RATE_CHANGE
int bitRate;
// ENC_TRANS_RATE_CHANGE
int transRate;
// ENC_RC_INTRA_MIN_MAX_CHANGE
int intraMaxQp;
int intraMinQp;
// ENC_ROT_PARAM_CHANGE
int rotEnable;
int rotMode;

// ENC_NR_PARAM_CHANGE
Uint32  nrYEnable;
Uint32  nrCbEnable;
Uint32  nrCrEnable;
Uint32  nrNoiseEstEnable;
Uint32  nrNoiseSigmaY;
Uint32  nrNoiseSigmaCb;
```

```
    Uint32  nrNoiseSigmaCr;
    // ENC_NR_WEIGHT_CHANGE
    Uint32  nrIntraWeightY;
    Uint32  nrIntraWeightCb;
    Uint32  nrIntraWeightCr;
    Uint32  nrInterWeightY;
    Uint32  nrInterWeightCb;
    Uint32  nrInterWeightCr;
    // ENC_NUM_UNITS_IN_TICK_CHANGE
    Uint32  numUnitsInTick;
    // ENC_TIME_SCALE_CHANGE
    Uint32 timeScale;
    // belows are only for WAVE5 encoder
    Uint32 numTicksPocDiffOne;
    Uint32 monochromeEnable;
    Uint32 strongIntraSmoothEnable;
    Uint32 weightPredEnable;
    Uint32 bgDetectEnable;
    Uint32 bgThrDiff;
    Uint32 bgThrMeanDiff;
    Uint32 bgLambdaQp;
    int    bgDeltaQp;
    Uint32 customLambdaEnable;
    Uint32 customMDEnable;
    int    pu04DeltaRate;
    int    pu08DeltaRate;
    int    pu16DeltaRate;
    int    pu32DeltaRate;
    int    pu04IntraPlanarDeltaRate;
    int    pu04IntraDcDeltaRate;
    int    pu04IntraAngleDeltaRate;
    int    pu08IntraPlanarDeltaRate;
    int    pu08IntraDcDeltaRate;
    int    pu08IntraAngleDeltaRate;
    int    pu16IntraPlanarDeltaRate;
    int    pu16IntraDcDeltaRate;
    int    pu16IntraAngleDeltaRate;
    int    pu32IntraPlanarDeltaRate;
    int    pu32IntraDcDeltaRate;
    int    pu32IntraAngleDeltaRate;
    int    cu08IntraDeltaRate;
    int    cu08InterDeltaRate;
    int    cu08MergeDeltaRate;
    int    cu16IntraDeltaRate;
    int    cu16InterDeltaRate;
    int    cu16MergeDeltaRate;
    int    cu32IntraDeltaRate;
    int    cu32InterDeltaRate;
    int    cu32MergeDeltaRate;
    int    coefClearDisable;
    int    seqRoiEnable;
}EncChangeParam;
```

Description

This is a data structure for encoding parameters that have changed.

changeParaMode

- 0 : changes the COMMON parameters.
- 1 : Reserved

enable_option
the section called "ChangeCommonParam"

losslessEnable
It enables lossless coding.

constIntraPredFlag
It enables constrained intra prediction.

chromaCbQpOffset
The value of chroma(Cb) QP offset (only for WAVE420L)

chromaCrQpOffset
The value of chroma(Cr) QP offset (only for WAVE420L)

decodingRefreshType
The type of I picture to be inserted at every intraPeriod
- 0 : Non-IRAP
- 1 : CRA
- 2 : IDR

intraPeriod
A period of intra picture in GOP size

intraQP
A quantization parameter of intra picture

confWinTop
A top offset of conformance window

confWinBot
A bottom offset of conformance window

confWinLeft
A left offset of conformance window

confWinRight
A right offset of conformance window

frameRate
A frame rate indicator (x 1024)

independSliceMode
A slice mode for independent slice
- 0 : no multi-slice
- 1 : Slice in CTU number

independSliceModeArg
The number of CTU for a slice when independSliceMode is set with 1

dependSliceMode
A slice mode for dependent slice
- 0 : no multi-slice
- 1 : Slice in CTU number
- 2 : Slice in number of byte

dependSliceModeArg
The number of CTU or bytes for a slice when dependSliceMode is set with 1 or 2

intraRefreshMode
An intra refresh mode

- 0 : No intra refresh
- 1 : Row
- 2 : Column
- 3 : Step size in CTU

intraRefreshArg
The number of CTU (only valid when intraRefreshMode is 3.)

useRecommendEncParam
It uses a recommended ENC_PARAM setting.
- 0 : Custom
- 1 : Recommended ENC_PARAM
- 2 ~ 3 : Reserved

scalingListEnable
It enables a scaling list.

cuSizeMode
It enables CU(Coding Unit) size to be used in encoding process. Host application can also select multiple CU sizes.
- 0 : 8x8
- 1 : 16x16
- 2 : 32x32

tmvpEnable
It enables temporal motion vector prediction.

wppEnable
It enables WPP (Wave-front Parallel Processing). WPP is unsupported in ring buffer mode of bitstream buffer.

maxNumMerge
Maximum number of merge candidates (0~2)

dynamicMerge8x8Enable
It enables dynamic merge 8x8 candidates.

dynamicMerge16x16Enable
It enables dynamic merge 16x16 candidates.

dynamicMerge32x32Enable
It enables dynamic merge 32x32 candidates.

disableDeblk
It disables in-loop deblocking filtering.

lfCrossSliceBoundaryEnable
It enables filtering across slice boundaries for in-loop deblocking.

betaOffsetDiv2
It enables BetaOffsetDiv2 for deblocking filter.

tcOffsetDiv2
It enables TcOffsetDiv3 for deblocking filter.

skipIntraTrans
It enables transform skip for an intra CU.

saoEnable
It enables SAO (Sample Adaptive Offset).

intraInInterSliceEnable
It enables to make intra CUs in an inter slice.

intraNxNEnable
It enables intra NxN PUs.

rcEnable
- WAVE420
  - 0 : Rate control is off.
  - 1 : Rate control is on.
- CODA9
  - 0 : Constant QP (VBR, rate control off)
  - 1 : Constant Bit-Rate (CBR)
  - 2 : Average Bit-Rate (ABR)
  - 4 : Picture level rate control

intraQpOffset
It specifies an intra QP offset relative to an inter QP. It is only valid when RateControl is enabled.

initBufLevelx8
It specifies encoder initial delay. It is only valid when RateControl is enabled.

```
encoder initial delay = InitialDelay * InitBufLevelx8 / 8
```

bitAllocMode
It specifies picture bits allocation mode. It is only valid when RateControl is enabled and GOP size is larger than 1.
- 0 : More referenced pictures have better quality than less referenced pictures
- 1 : All pictures in a GOP have similar image quality
- 2 : Each picture bits in a GOP is allocated according to FixedRatioN

fixedBitRatio
A fixed bit ratio (1 ~ 255) for each picture of GOP's bit allocation
- N = 0 ~ (MAX_GOP_SIZE - 1)
- MAX_GOP_SIZE = 8

For instance when MAX_GOP_SIZE is 3, FixedBitRatio0, FixedBitRatio1, and FixedBitRatio2 can be set as 2, 1, and 1 repsectively for the fixed bit ratio 2:1:1. This is only valid when BitAllocMode is 2.

cuLevelRCEnable
It enables CU level rate control.

hvsQPEnable
It enables CU QP adjustment for subjective quality enhancement.

hvsQpScaleEnable
It enables QP scaling factor for CU QP adjustment when hvsQPEnable is 1.

hvsQpScale
QP scaling factor for CU QP adjustment when hvcQpenable is 1.

initialDelay
An initial cpb delay in msec

initialRcQp
The value of initial QP by HOST application. This value is meaningless if INITIAL_RC_QP is 63. (only for WAVE420L)

minQp
Minimum QP for rate control

maxQp
Maximum QP for rate control

maxDeltaQp
Maximum delta QP for rate control

bitRate
A target bitrate when separateBitrateEnable is 0

transRate
A peak transmission bitrate in bps

intraMaxQp
It specifies a maximum QP for intra picture (0 ~ 51). It is only valid when RateControl is 1.

intraMinQp
It specifies a minimum QP for intra picture (0 ~ 51). It is only valid when RateControl is 1.

rotEnable
It enables or disable rotation.

rotMode
- [1:0] 90 degree left rotate
  - 1=90'
  - 2=180'
  - 3=270'
- [2] vertical mirror
- [3] horizontal mirror

nrYEnable
It enables noise reduction algorithm to Y component.

nrCbEnable
It enables noise reduction algorithm to Cb component.

nrCrEnable
It enables noise reduction algorithm to Cr component.

nrNoiseEstEnable
It enables noise estimation for reduction. When this is disabled, noise estimation is carried out ouside VPU.

nrNoiseSigmaY
It specifies Y noise standard deviation if no use of noise estimation (nrNoiseEstEnable is 0).

nrNoiseSigmaCb
It specifies Cb noise standard deviation if no use of noise estimation (nrNoiseEstEnable is 0).

nrNoiseSigmaCr
It specifies Cr noise standard deviation if no use of noise estimation (nrNoiseEstEnable is 0).

nrIntraWeightY
A weight to Y noise level for intra picture (0 ~ 31). nrIntraWeight/4 is multiplied to the noise level that has been estimated. This weight is put for intra frame to be filtered more strongly or more weakly than just with the estimated noise level.

nrIntraWeightCb
A weight to Cb noise level for intra picture (0 ~ 31).

nrIntraWeightCr
A weight to Cr noise level for intra picture (0 ~ 31).

nrInterWeightY
A weight to Y noise level for inter picture (0 ~ 31). nrInterWeight/4 is multiplied to the noise level that has been estimated. This weight is put for inter frame to be filtered more strongly or more weakly than just with the estimated noise level.

nrInterWeightCb
A weight to Cb noise level for inter picture (0 ~ 31).

nrInterWeightCr
A weight to Cr noise level for inter picture (0 ~ 31).

numUnitsInTick
It specifies the number of time units of a clock operating at the frequency time_scale Hz.

timeScale
It specifies the number of time units that pass in one second.

AvcPpsParam

```
typedef struct {
    int ppsId;
    int entropyCodingMode;
    int cabacInitIdc;
    int transform8x8Mode;
} AvcPpsParam;
```

Description

This is a data structure for configuring PPS information at H.264/AVC.

ppsId
H.264 picture_parameter_set_id in PPS. This shall be in the range of 0 to 255, inclusive.

entropyCodingMode
It selects the entropy coding method used in the encoding process.
- 0 : CAVLC
- 1 : CABAC
- 2 : CAVLC/CABAC select according to PicType

cabacInitIdc
It specifies the index for determining the initialization table used in the initialisation process for CABAC. The value of cabac_init_idc shall be in the range of 0 ~ 2.

transform8x8Mode
It specifies whether to enable 8x8 intra prediction and 8x8 transform or not.
- 0 : disable 8x8 intra and 8x8 transform (BP)
- 1 : enable 8x8 intra and 8x8 transform (HP)

EncAvcParam

```
typedef struct {
    int constrainedIntraPredFlag;
    int disableDeblk;
    int deblkFilterOffsetAlpha;
```

```
    int deblkFilterOffsetBeta;
    int chromaQpOffset;
    int audEnable;
    int frameCroppingFlag;
    int frameCropLeft;
    int frameCropRight;
    int frameCropTop;
    int frameCropBottom;
    int mvcExtension;
    int interviewEn;
    int parasetRefreshEn;
    int prefixNalEn;
    int svcExtension;
    int level;
    int profile;
    // [START] CODA980 & WAVE320
    int fieldFlag;
    int fieldRefMode;  int chromaFormat400;
    int ppsNum;
    AvcPpsParam ppsParam[MAX_ENC_PPS_NUM];
    // [END] CODA980 & WAVE320
} EncAvcParam;
```

Description

This is a data structure for configuring H.264/AVC-specific parameters in encoder applications.

constrainedIntraPredFlag
- 0 : disable
- 1 : enable

disableDeblk
- 0 : enable
- 1 : disable
- 2 : disable deblocking filter at slice boundaries

deblkFilterOffsetAlpha
deblk_filter_offset_alpha (-6 - 6)

deblkFilterOffsetBeta
deblk_filter_offset_beta (-6 - 6)

chromaQpOffset
chroma_qp_offset (-12 - 12)

audEnable
- 0 : disable
- 1 : enable

If this is 1, the encoder generates AUD RBSP at the start of every picture.

frameCroppingFlag
- 0 : disable
- 1 : enable

If this is 1, the encoder generates frame_cropping_flag syntax at the SPS header.

frameCropLeft
The sample number of left cropping region in a line. See the frame_crop_left_offset syntax in AVC/H.264 SPS tabular form. The least significant bit of this parameter should be always zero.

frameCropRight
The sample number of right cropping region in a line. See the frame_crop_right_offset syntax in AVC/H.264 SPS tabular form. The least significant bit of this parameter should be always zero.

frameCropTop
The sample number of top cropping region in a picture column. See the frame_crop_top_offset syntax in AVC/H.264 SPS tabular form. The least significant bit of this parameter should be always zero.

frameCropBottom
The sample number of bottom cropping region in a picture column. See the frame_crop_bottom_offset syntax in AVC/H.264 SPS tabular form. The least significant bit of this parameter should be always zero.

mvcExtension
MVC extension

interviewEn
MVC extension

parasetRefreshEn
MVC extension

prefixNalEn
MVC extension

svcExtension
SVC extension

level
H.264/AVC level_idc in SPS

profile
H.264 profile_idc parameter is derived from each coding tool usage.
- 0 : Baseline profile
- 1 : Main profile
- 2 : High profile

fieldFlag
It selects the encoding type, progressive or interlaced.
- 0 : Progressive frame encoding.
- 1 : Interlaced field encoding.

fieldRefMode
Reference mode for interlaced field encoding(avc_field_flag is 1).
- 0 : Same parity field referencing
- 1 : Opposite parity field referencing

chromaFormat400
It specifies type of chroma format.
- 0 : YUV 420 format
- 1 : YUV 400 monochrome format

ppsNum
Number of PPS. If a PPS has the same pps_id, the last PPS is available.

EncSliceMode

```
typedef struct{
```

```
    int sliceMode;
    int sliceSizeMode;
    int sliceSize;
} EncSliceMode;
```

Description

This structure is used for declaring an encoder slice mode and its options. It is newly added for more flexible usage of slice mode control in encoder.

sliceMode

- 0 : one slice per picture
- 1 : multiple slices per picture
- 2 : multiple slice encoding mode 2 for H.264 only.

In normal MPEG4 mode, resync-marker and packet header are inserted between slice boundaries. In short video header with Annex K of 0, GOB headers are inserted at every GOB layer start. In short video header with Annex K of 1, multiple slices are generated. In AVC mode, multiple slice layer RBSP is generated.

sliceSizeMode

This parameter means the size of generated slice when sliceMode of 1.

- 0 : sliceSize is defined by the amount of bits
- 1 : sliceSize is defined by the number of MBs in a slice
- 2 : sliceSize is defined by MBs run-length table (only for H.264)

This parameter is ignored when sliceMode of 0 or in short video header mode with Annex K of 0.

sliceSize

The size of a slice in bits or in MB numbers included in a slice, which is specified by the variable, sliceSizeMode. This parameter is ignored when sliceMode is 0 or in short video header mode with Annex K of 0.

MinMaxQpChangeParam

```
typedef struct {
    int maxQpIEnable;
    int maxQpI;
    int minQpIEnable;
    int minQpI;
    int maxQpPEnable;
    int maxQpP;
    int minQpPEnable;
    int minQpP;
} MinMaxQpChangeParam;
```

Description

This data structure is used when HOST wants to change min/max QP values.

maxQpIEnable

A maximum QP enable flag for I picture

maxQpI

A maximum QP of I picture for rate control

minQpIEnable

A minimum QP enable flag for I picture

minQpI
A minimum QP of I picture for rate control

maxQpPEnable
A maximum QP enable flag for P picture

maxQpP
A maximum QP of P picture for rate control

minQpPEnable
A minimum QP enable flag for P picture

minQpP
A minimum QP of P picture for rate control

ChangePicParam

```
 typedef struct{
     int MaxdeltaQp;
     int MindeltaQp;
     int HvsQpScaleDiv2;
     int EnHvsQp;
     int EnRowLevelRc;
     int RcHvsMaxDeltaQp;
     int Gamma;
     int GammaEn;
     int RcInitDelay;
     int SetDqpNum;
     int dqp[8];
     int RcGopIQpOffset;
     int RcGopIQpOffsetEn;
#ifdef AUTO_FRM_SKIP_DROP
     int EnAutoFrmSkip;
     int EnAuToFrmDrop;
     int VbvThreshold;
     int QpThreshold;
     int MaxContinuousFrameSkipNum;
     int MaxContinuousFrameDropNum;
#endif
     int rcWeightFactor;
} ChangePicParam;
```

Description

This data structure is used when HOST wants to change picture level parameter values.

MaxdeltaQp
A maximum delta QP for HVS

MindeltaQp
A minimum delta QP for HVS

HvsQpScaleDiv2
It specifies the scale for MB QP derivation. It is only avilable when EN_HVS_QP is 1.

EnHvsQp
It enables HVS QP.

EnRowLevelRc
It enables row-level rate control.

RcHvsMaxDeltaQp
A maximum delta QP value for HVS considered rate control

RcInitDelay
Time delay in mili-seconds (CODA9 only)

It is the amount of time in ms taken for bitstream to reach initial occupancy of the vbv buffer from zero level.

This value is ignored if rate control is disabled. The value 0 means VPU does not check for reference decoder buffer delay constraints.

SetDqpNum
It is the number of cyclic pictures.

dqp
A delta QP value of frame# within the cyclic pictures

RcGopIQpOffset
An initial QP offset for I picture in GOP (CODA980 only)

rcGopIQpOffset (-4 to 4) is added to an I picture QP value. This value is valid for H.264/AVC encoder and ignored when RcEnable is 0 or RcGopIQpOffsetEn is 0.

RcGopIQpOffsetEn
An enable flag for initial QP offset for I picture in GOP. (CODA980 only)
- 0 : disable (default)
- 1 : enable

This value is valid for H.264/AVC encoder and ignored when RcEnable is 0.

EnAutoFrmSkip
It enables frame skip automatically according to threshold

EnAuToFrmDrop
It enables frame drop automatically according to threshold

VbvThreshold
Vbv buffer threshold value

QpThreshold
Qp buffer threshold value

MaxContinuousFrameSkipNum
The maximum number of continuous frame(s) to skip

MaxContinuousFrameDropNum
The maximum number of confinuous frame(s) to drop

rcWeightFactor
Adjusts the speed of updating parameters to the rate control model. If RcWeightFactor is set with a large value, the RC parameters are updated slowly. (Min: 1, Max: 32, default: 2)

ParamChange

```
typedef struct {
    char pchChangeCfgFileName[1024];
    int  ChangeFrameNum;
    int  paraEnable;
    int  NewGopNum;
    //int  NewIntraQpEn;
```

```
    int  NewIntraQp;
    int  NewBitrate;
    int  NewFrameRate;
    int  NewIntraRefresh;
    MinMaxQpChangeParam minMaxQpParam;
#ifdef RC_PIC_PARACHANGE
    ChangePicParam changePicParam;
#endif
} ParamChange;
```

Description

This data structure is used when HOST wants to apply new RC parameters.

pchChangeCfgFileName
Encoder configuration file name

ChangeFrameNum
A frame number to start applying parameter change

paraEnable
An enable flag for each parameter change

NewGopNum
A new encode GOP number

NewIntraQp
A new quantized step parameter of intra frame picture for encoding process.

NewBitrate
A new target bit rate in kilo bit per seconds (kbps)

NewFrameRate
A new frame rate

NewIntraRefresh
A new intra MB refresh number

minMaxQpParam
the section called "MinMaxQpChangeParam" for Min/Max Qp change

changePicParam
the section called "ChangePicParam"

EncOpenParam

```
typedef struct {
    PhysicalAddress bitstreamBuffer;
    Uint32          bitstreamBufferSize;
    CodStd          bitstreamFormat;
    int             ringBufferEnable;
    int             picWidth;
    int             picHeight;
    int             linear2TiledEnable;
    int             linear2TiledMode;
    int             frameRateInfo;
    // [START] CODA980 & WAVE320
    int             MESearchRangeX;
    int             MESearchRangeY;
    int             rcGopIQpOffsetEn;
    int             rcGopIQpOffset;
```

```
int             MESearchRange;
// [END] CODA980 & WAVE320
// [START] CODA9xx & WAVE320
int             vbvBufferSize;
int             frameSkipDisable;

int             gopSize;
int idrInterval;
int             meBlkMode;
EncSliceMode    sliceMode;
int             intraRefresh;
int             ConscIntraRefreshEnable;
int             CountIntraMbEnable;
int             FieldSeqIntraRefreshEnable;
int             userQpMax;

//h.264 only
int             maxIntraSize;
int             userMaxDeltaQp;
int             userMinDeltaQp;
int             userQpMinI;
int             userQpMinP;
int             userQpMaxI;
int             userQpMaxP;
int             MEUseZeroPmv;
int             intraCostWeight;
//mp4 only
int             rcIntraQp;
int             userGamma;
int             rcIntervalMode;
int             mbInterval;
// [END] CODA9xx & WAVE320
int             bitRate;
int             initialDelay;
int             rcEnable;

union {
    EncMp4Param     mp4Param;
    EncH263Param    h263Param;
    EncAvcParam     avcParam;
    EncHevcParam    hevcParam;
} EncStdParam;

// Maverick-II Cache Configuration
int             frameCacheBypass;
int             frameCacheBurst;
int             frameCacheMerge;
int             frameCacheWayShape;
int             cbcrInterleave;
int             cbcrOrder;
int             frameEndian;
int             streamEndian;
int             sourceEndian;
int             bwbEnable;
int             lineBufIntEn;
int             packedFormat;
int             srcFormat;
int             srcBitDepth;
ParamChange paramChange;

Uint32          coreIdx;
```

```
    int             nv21;


    Uint32 encodeHrdRbspInVPS;
    Uint32 encodeHrdRbspInVUI;
    Uint32 hrdRbspDataSize;
    PhysicalAddress hrdRbspDataAddr;
    Uint32 encodeVuiRbsp;
    Uint32 vuiRbspDataSize;
    PhysicalAddress vuiRbspDataAddr;
    Uint32          virtAxiID;
    BOOL            enablePTS;
    int             coda9RoiEnable;
    int             RoiPicAvgQp;
    char            RoiFile[1024];
    int             set_dqp_pic_num;
    gop_entry_t     gopEntry[MAX_GOP_SIZE *2];
    int             gopPreset;
    int             LongTermPeriod;
    int             LongTermDeltaQp;
    int             dqp[8];
    int             HvsQpScaleDiv2;
    int             EnHvsQp;
    int             EnRowLevelRc;
    int             RcInitialQp;
    int             RcHvsMaxDeltaQp;
#ifdef ROI_MB_RC
    int             roi_max_delta_qp_minus;
    int             roi_max_delta_qp_plus;
#endif
#ifdef AUTO_FRM_SKIP_DROP
    int             enAutoFrmSkip;
    int             enAutoFrmDrop;
    int             vbvThreshold;
    int             qpThreshold;
    int             maxContinuosFrameSkipNum;
    int             maxContinuosFrameDropNum;
#endif
    int             rcWeightFactor;
    int             rcInitDelay;

    int             intraRefreshNum;
} EncOpenParam;
```

Description

This data structure is used when HOST wants to open a new encoder instance.

bitstreamBuffer
The start address of bitstream buffer into which encoder puts bitstream. This address must be aligned to AXI bus width.

bitstreamBufferSize
The size of the buffer in bytes pointed by bitstreamBuffer. This value must be a multiple of 1024. The maximum size is 16383 x 1024 bytes.

bitstreamFormat
The standard type of bitstream in encoder operation. It is one of STD_MPEG4, STD_H263, STD_AVC and STD_MJPG.

ringBufferEnable

- 0 : disable
- 1 : enable

This flag declares the streaming mode for the current encoder instance. Two streaming modes, packet-based streaming with ring-buffer (buffer-reset mode) and frame-based streaming with line buffer (buffer-flush mode), can be configured by using this flag.

When this field is set, packet-based streaming with ring-buffer is used. And when this field is not set, frame-based streaming with line-buffer is used.

picWidth

The width of a picture to be encoded in pixels.

picHeight

The height of a picture to be encoded in pixels.

linear2TiledEnable

It is a linear to tiled enable mode. The map type can be changed from linear to tiled format while reading source frame in the PrP (Pre-Processing) unit.

- 0 : disable linear to tiled-map conversion for getting source image
- 1 : enable linear to tiled-map conversion for getting source image

linear2TiledMode

It can specify the map type of source frame buffer when linear2TiledEnable is enabled. (CODA980 only)

- 1 : source frame buffer is in linear frame map.
- 2 : source frame buffer is in linear field map.

frameRateInfo

The 16 LSB bits, [15:0], is a numerator and 16 MSB bits, [31:16], is a denominator for calculating frame rate. The numerator means clock ticks per second, and the denominator is clock ticks between frames minus 1.

So the frame rate can be defined by (numerator/(denominator + 1)), which equals to (frameRateInfo & 0xffff) /((frameRateInfo >> 16) + 1).

For example, the value 30 of frameRateInfo represents 30 frames/sec, and the value 0x3e87530 represents 29.97 frames/sec.

MESearchRangeX

The horizontal search range mode for Motion Estimation

- 0 : Horizontal(-64 ~ 63)
- 1 : Horizontal(-48 ~ 47)
- 2 : Horizontal(-32 ~ 31)
- 3 : Horizontal(-16 ~ 15)

MESearchRangeY

The vertical search range mode for Motion Estimation

- 0 : Vertical(-48 ~ 47)
- 1 : Vertical(-32 ~ 31)
- 2 : Vertical(-16 ~ 15)

rcGopIQpOffsetEn

An initial QP offset for I picture in GOP.

This setting can lower an I picture QP value down to a user-defined value ranging -4 to 4. This value is valid for AvcEnc and ignored when when RcEnable is 0 or RcGopIQpOff-setEn is 0.

rcGopIQpOffset
An enable flag for initial QP offset for I picture in GOP.
- 0 : disable (default)
- 1 : disable

This value is valid for AvcEnc and ignored when when RcEnable is 0.

MESearchRange
The search range mode for Motion Estimation
- 0 : Horizontal(-128 ~ 127), Vertical(-64 ~ 63)
- 1 : Horizontal(-64 ~ 63), Vertical(-32 ~ 31)
- 2 : Horizontal(-32 ~ 31), Vertical(-16 ~ 15)
- 3 : Horizontal(-16 ~ 15), Vertical(-16 ~ 15)

vbvBufferSize
vbv_buffer_size in bits

This value is ignored if rate control is disabled or initialDelay is 0. The value 0 means the encoder does not check for reference decoder buffer size constraints.

frameSkipDisable
Frame skip indicates that encoder can skip frame encoding automatically when bitstream has been generated much so far considering the given target bitrate. This parameter is ignored if rate control is not used (bitRate is 0).
- 0 : enables frame skip function in encoder.
- 1 : disables frame skip function in encoder.

gopSize
This is the GOP size.
- 0 : only first picture is I
- 1 : all I pictures
- 2 : IPIP…
- 3 : IPPIPP…

The maximum value is 32767, but in practice, a smaller value should be chosen by HOST application for proper error concealment operations. This value is ignored in case of STD_MJPG

idrInterval
An interval of adding an IDR picture

meBlkMode
A block mode enable flag for Motion Estimation. (H.264/AVC only). A HOST can use some combination (bitwise or-ing) of each value under below.
- 4'b0000 or 4'b1111 : Use all block mode
- 4'b0001 : Enable 16x16 block mode
- 4'b0010 : Enable 16x8 block mode
- 4'b0100 : Enable 8x16 block mode
- 4'b1000 : Enable 8x8 block mode

sliceMode
The parameter for slice mode

intraRefresh
- 0 : intra MB refresh is not used.
- Otherwise : at least N MBs in every P-frame is encoded as intra MBs.

This value is ignored in case of STD_MJPG

ConscIntraRefreshEnable
Consecutive intra MB refresh mode

This option is valid only when IntraMbRefresh-Num[15:0] is not 0.
- 0 - ConscIntraRefreshEn is disabled. IntraMbRefreshNumnumber of MBs are encoded as an intra MB at the defined interval by picture size.
- 1 - IntraMbRefreshNumnumber of consec-utive MBs are encoded as an intra MB.

CountIntraMbEnable
Intra MB count enable

If it is enabled, VPU does not include intra MBs from mode decision when counting IntraMbRefreshNum. This option is valid only when IntraMbRefreshNum[15:0] is not 0.
- 0 - intra mb counting is disabled.
- 1 - intra mb counting is enabled.

FieldSeqIntraRefreshEnable
Field sequence intra refresh enable

This option is valid only when IntraMbRefresh-Num[15:0] is not 0 and interlaced encoding.
- 0 - FieldSeqIntraRefreshEn is disabled.
- 1 - Intra mb is sequentially refreshed at each field.

userQpMax
The maximum quantized step parameter for encoding process

In MPEG4/H.263 mode, the maximum value is 31. In H.264 mode, allowed maximum value is 51.

maxIntraSize
The maximum bit size for intra frame or encoding process. It works only in H.264/AVC encoder.

userMaxDeltaQp
The maximum delta QP for encoding process. It works only in the H.264/AVC mode.

userMinDeltaQp
The minimum delta QP for encoding process. It works only in the H.264/AVC mode.

userQpMinI
The minimum quantized step parameter for Intra frame. (H.264/AVC only)

userQpMinP
The minimum quantized step parameter for Inter frame. (H.264/AVC only)

userQpMaxI
The maximum quantized step parameter for Intra frame. (H.264/AVC only)

userQpMaxP
The maximum quantized step parameter for Inter frame. (H.264/AVC only)

MEUseZeroPmv
The PMV option for Motion Estimation. If this field is 1, encoding quality could be worse than when it was zero.
- 0 : Motion Estimation engine uses PMV that was derived from neighbor MV
- 1 : Motion Estimation engine uses Zero PMV

intraCostWeight
Additional weight of intra cost for mode decision to reduce intra MB density

By default, it could be zero. If this register have some value W, and the cost of best intra mode that was decided by Refine-Intra-Mode-Decision is ICOST, the Final Intra Cost FIC is like below,

FIC = ICOST + W

So, if this field is not zero, the Final Intra Cost have additional weight. Then the Intra/Inter mode decision logic tend to make more Inter-Macroblock.

rcIntraQp
The quantization parameter for I frame

When this value is -1, the quantization parameter for I frames is automatically determined by VPU. This value is ignored in case of STD_MJPG

userGamma
A gamma is a smoothing factor in motion estimation. A value for gamma is factor * 32768, the factor value is selected from the range $0 \leq$ factor ≥ 1.
- If the factor value getting close to 0, Qp changes slowly.
- If the factor value getting close to 1, Qp changes quickly.

The default gamma value is 0.75 * 32768

rcIntervalMode
Encoder Rate Control Mode setting
- 0 : Normal mode rate control
- 1 : FRAME_LEVEL rate control
- 2 : SLICE_LEVEL rate control
- 3 : USER DEFINED MB LEVEL rate control

mbInterval
The user defined MB interval value

This value is used only when rcIntervalMode is 3.

bitRate
Target bit rate in kbps

If 0, there is no rate control, and pictures are encoded with a quantization parameter equal to quantParam in EncParam.

initialDelay
Time delay (in mili-seconds)

It takes for the bitstream to reach initial occupancy of the vbv buffer from zero level.

This value is ignored if rate control is disabled. The value 0 means the encoder does not check for reference decoder buffer delay constraints.

rcEnable
- WAVE420
  - 0 : Rate control is off.
  - 1 : Rate control is on.
- CODA9
  - 0 : Constant QP (VBR, rate control off)
  - 1 : Constant Bit-Rate (CBR)
  - 2 : Average Bit-Rate (ABR)
  - 4 : Picture level rate control

mp4Param
The parameters for MPEG4 part 2 Visual

h263Param
The parameters for ITU-T H.263

avcParam
The parameters for ITU-T H.263

hevcParam
The parameters for ITU-T H.265/HEVC

frameCacheBypass
Enable or disable frame buffer cache
- 0 : ME/MC CACHE
- 1 : ME BYPASS/MC CACHE
- 2 : MC BYPASS/ME CACHE
- 3 : ME/MC BYPASS

frameCacheBurst
Cache burst value
- 0 : Burst 4
- 1 : Burst 8

frameCacheMerge
- 0 : Horizontal no merge
- 1 : Horizontal merge
- 2 : Vertical no merge
- 3 : Vertical merge

frameCacheWayShape
Cache way size of luma and chroma. This value range is from 0 to 15.

[0-15] {chroma[1:0], luma[1:0]}
- Luma [0,1 : 64x64, 2,3 : 128x32]
- Chroma Separated [0,1: 32x32, 2,3: 64x16]
- Chroma Interleaved [0,1: 32x64, 2: 64x32, 3: 128x16]

cbcrInterleave
- 0 : CbCr data is written in separate frame memories (chroma separate mode)
- 1 : CbCr data is interleaved in chroma memory. (chroma interleave mode)

cbcrOrder
CbCr order in planar mode (YV12 format)
- 0 : Cb data are written first and then Cr written in their separate plane.
- 1 : Cr data are written first and then Cb written in their separate plane.

frameEndian
Frame buffer endianness
- 0 : little endian format
- 1 : big endian format
- 2 : 32 bit little endian format
- 3 : 32 bit big endian format
- 16 ~ 31 : 128 bit endian format

Note For setting specific values of 128 bit endiness, please refer to the *WAVE4 Datasheet*.

streamEndian
Bistream buffer endianness
- 0 : little endian format
- 1 : big endian format

- 2 : 32 bit little endian format
- 3 : 32 bit big endian format
- 16 ~ 31 : 128 bit endian format

Note For setting specific values of 128 bit endiness, please refer to the *WAVE4 Datasheet*.

sourceEndian

Endianness of source YUV

- 0 : Little endian format
- 1 : Big endian format
- 2 : 32 bit little endian format
- 3 : 32 bit big endian format
- 16 ~ 31 : 128 bit endian format

Note For setting specific values of 128 bit endiness, please refer to the *WAVE4 Datasheet*.

bwbEnable

It writes output with 8 burst in linear map mode.

- 0 : Burst write back is disabled
- 1 : Burst write back is enabled.

Note WAVE4 does not support it.

lineBufIntEn

- 0 : Disable
- 1 : Enable

This flag is used for AvcEnc frame-based streaming with line buffer. If coding standard is not AvcEnc or ringBufferEnable is 1, this flag is ignored.

If this field is set, VPU sends a buffer full interrupt when line buffer is full, and waits until the interrupt is cleared. HOST should read the bitstream in line buffer and clear the interrupt. If this field is not set, VPU does not send a buffer full interrupt when line buffer is full. In both cases, VPU sets RET_ENC_PIC_FLAG register to 1 if line buffer is full.

packedFormat

This flag indicates whether source images are in packed format.

- 0 : Not packed mode
- 1 : Packed mode

srcFormat

This flag indicates a color format of source image which is one among *the section called "FrameBufferFormat"*.

coreIdx

VPU core index number

- 0 ~ number of VPU core - 1

nv21

- 0 : CbCr data is interleaved in chroma source frame memory. (NV12)
- 1 : CrCb data is interleaved in chroma source frame memory. (NV21)

encodeHrdRbspInVPS

It encodes the HRD syntax into VPS.

encodeHrdRbspInVUI

It encodes the HRD syntax into VUI.

hrdRbspDataSize
The bit size of the HRD rbsp data

hrdRbspDataAddr
The address of the HRD rbsp data

encodeVuiRbsp
A flag to encode the VUI syntax in rbsp which is given by HOST application

vuiRbspDataSize
The bit size of the VUI rbsp data

vuiRbspDataAddr
The address of the VUI rbsp data

enablePTS
An enable flag to report PTS(Presentation Timestamp.

coda9RoiEnable
It enables ROI for CODA980.

RoiPicAvgQp
It specifies an ROI average QP.

RoiFile
It specifies the ROI file.

set_dqp_pic_num
It is the number of cyclic pictures.

gopEntry
The entire structure of cyclic pictures

gopPreset
GOP preset value (0~5)

LongTermPeriod
It specifies an interval value of longterm period.

LongTermDeltaQp
It specifies a delta QP value between base(intra) picture and longterm inter picture.

dqp
A delta QP value of frame# within the cyclic pictures

HvsQpScaleDiv2
It specifies the scale for MB QP derivation. It is only avilable when EN_HVS_QP is 1.

EnHvsQp
It enables HVS QP.

EnRowLevelRc
It enables row-level rate control.

RcInitialQp
An initial QP for rate control

RcHvsMaxDeltaQp
A maximum delta QP value for HVS considered rate control

enAutoFrmSkip
It enables frame skip automatically according to threshold

enAutoFrmDrop
It enables frame drop automatically according to threshold

vbvThreshold
Vbv buffer threshold value

qpThreshold
Qp buffer threshold value

maxContinuosFrameSkipNum
It specifies the maximum number of continusous frame skip.

maxContinuosFrameDropNum
It specifies the maximum number of continusous frame drop.

rcWeightFactor
Adjusts the speed of updating parameters to the rate control model. If RcWeightFactor is set with a large value, the RC parameters are updated slowly. (Min: 1, Max: 32, default: 2)

rcInitDelay
Time delay in mili-seconds (CODA9 only)

It is the amount of time in ms taken for bitstream to reach initial occupancy of the vbv buffer from zero level.

This value is ignored if rate control is disabled. The value 0 means VPU does not check for reference decoder buffer delay constraints.

intraRefreshNum
The number of intra MB to be inserted in picture (CODA9 only)
- 0 : intra MB refresh is not used.
- Other value : intraRefreshNum of MBs are encoded as intra MBs in every P frame.

EncInitialInfo

```
typedef struct {
    int             minFrameBufferCount;
    int             minSrcFrameCount;
} EncInitialInfo;
```

Description

This is a data structure for parameters of VPU_EncGetInitialInfo() which is required to get the minimum required buffer count in HOST application. This returned value is used to allocate frame buffers in VPU_EncRegisterFrameBuffer().

minFrameBufferCount
Minimum number of frame buffer

minSrcFrameCount
Minimum number of source buffer

AvcRoiParam

```
typedef struct {
```

```
    int mode;
    int number;
    VpuRect region[MAX_ROI_NUMBER];
    int qp[MAX_ROI_NUMBER];
} AvcRoiParam;
```

Description

This is a data structure for setting ROI. (CODA9 encoder only).

mode

- 0 : It disables ROI.
- 1 : User can set QP value for both ROI and non-ROI region
- 2 : User can set QP value for only ROI region (Currently, not support)

number

Total number of ROI

The maximum value is MAX_ROI_NUMBER. MAX_ROI_NUMBER can be either 10 or 50. In order to use MAX_ROI_NUMBER as 50, SUPPORT_ROI_50 define needs to be enabled in the reference SW.

region

Rectangle information for ROI

qp

QP value for ROI region

EncCodeOpt

```
typedef struct {
    int implicitHeaderEncode;
    int encodeVCL;
    int encodeVPS;
    int encodeSPS;
    int encodePPS;
    int encodeAUD;
    int encodeEOS;
    int encodeEOB;
    int encodeVUI;
    int encodeFiller;
} EncCodeOpt;
```

Description

This is a data structure for setting NAL unit coding options.

implicitHeaderEncode

Whether HOST application encodes a header implicitly or not. If this value is 1, below encode options are ignored.

encodeVCL

A flag to encode VCL nal unit explicitly

encodeVPS

A flag to encode VPS nal unit explicitly

encodeSPS

A flag to encode SPS nal unit explicitly

encodePPS
A flag to encode PPS nal unit explicitly

encodeAUD
A flag to encode AUD nal unit explicitly

encodeEOS
A flag to encode EOS nal unit explicitly. This should be set when to encode the last source picture of sequence.

encodeEOB
A flag to encode EOB nal unit explicitly. This should be set when to encode the last source picture of sequence.

encodeVUI
A flag to encode VUI nal unit explicitly

encodeFiller
A flag to encode Filler nal unit explicitly (WAVE5 only)

Rect

```
typedef struct {
    Uint32 left;
    Uint32 top;
    Uint32 right;
    Uint32 bottom;
} Rect;
```

Description

This is a data structure for setting ROI.

left
A horizontal pixel offset of top-left corner of rectangle from (0, 0), top-left corner of a frame.

top
A vertical pixel offset of top-left corner of rectangle from (0, 0), top-left corner of a frame.

right
A horizontal pixel offset of bottom-right corner of rectangle from (0, 0), bottom-right corner of a frame.

bottom
A vertical pixel offset of bottom-right corner of rectangle from (0, 0), bottom-right corner of a frame.

EncParam

```
typedef struct {
   FrameBuffer*    sourceFrame;
   int             forceIPicture;
   int             skipPicture;
   int             quantParam;
   PhysicalAddress picStreamBufferAddr;
  int             picStreamBufferSize;
   int             fieldRun;
```

```
    int             coda9RoiEnable;
    Uint32 coda9RoiPicAvgQp;
    PhysicalAddress roiQpMapAddr;
    Uint32 nonRoiQp;
#ifdef ROI_MB_RC
    AvcRoiParam     setROI;
#endif

    HevcCtuOptParam ctuOptParam;
    int             forcePicQpEnable;
    int             forcePicQpSrcOrderEnable;
    int             forcePicQpI;
    int             forcePicQpP;
    int             forcePicQpB;
    int             forcePicTypeEnable;
    int             forcePicTypeSrcOrderEnable;
    int             forcePicType;
    int             srcIdx;
    int             srcEndFlag;
    EncCodeOpt      codeOption;
    Uint32  useCurSrcAsLongtermPic;
    Uint32  useLongtermRef;
    Uint32  pts;
    // belows are newly added for WAVE5 encoder
    HevcCustomMapOpt customMapOpt;
    Uint32 wpPixVarianceY;
    Uint32 wpPixVarianceCb;
    Uint32 wpPixVarianceCr;
    Uint32 wpPixMeanY;
    Uint32 wpPixMeanCb;
    Uint32 wpPixMeanCr;
    Uint32 forceAllCtuCoefDropEnable;

} EncParam;
```

Description

This is a data structure for configuring one frame encoding operation.

sourceFrame
This member must represent the frame buffer containing source image to be encoded.

forceIPicture
If this value is 0, the picture type is determined by VPU according to the various parameters such as encoded frame number and GOP size.

If this value is 1, the frame is encoded as an I-picture regardless of the frame number or GOP size, and I-picture period calculation is reset to initial state. In MPEG4 and H.263 case, I-picture is sufficient for decoder refresh. In H.264/AVC case, the picture is encoded as an IDR (Instantaneous Decoding Refresh) picture.

This value is ignored if skipPicture is 1.

skipPicture
If this value is 0, the encoder encodes a picture as normal.

If this value is 1, the encoder ignores sourceFrame and generates a skipped picture. In this case, the reconstructed image at decoder side is a duplication of the previous picture. The skipped picture is encoded as P-type regardless of the GOP size.

quantParam
This value is used for all quantization parameters in case of VBR (no rate control).

picStreamBufferAddr
The start address of a picture stream buffer under line-buffer mode and dynamic buffer allocation. (CODA encoder only)

This variable represents the start of a picture stream for encoded output. In buffer-reset mode, HOST might use multiple picture stream buffers for the best performance. By using this variable, applications could re-register the start position of the picture stream while issuing a picture encoding operation. This start address of this buffer must be 4-byte aligned, and its size is specified the following variable, picStreamBufferSize. In packet-based streaming with ring-buffer, this variable is ignored.

Note This variable is only meaningful when both line-buffer mode and dynamic buffer allocation are enabled.

picStreamBufferSize
The byte size of a picture stream chunk

This variable represents the byte size of a picture stream buffer. This variable is so crucial in line-buffer mode. That is because encoder output could be corrupted if this size is smaller than any picture encoded output. So this value should be big enough for storing multiple picture streams with average size. In packet-based streaming with ring-buffer, this variable is ignored.

fieldRun
- 0 : progressive (frame) encoding mode
- 1 : interlaced (field) encoding mode

coda9RoiEnable
It enables ROI for CODA980.

coda9RoiPicAvgQp
A average value of ROI QP for a picture (CODA9 only)

roiQpMapAddr
The start address of ROI QP map (CODA9 only)

nonRoiQp
A non-ROI QP for a picture

setROI
This value sets ROI. If setROI mode is 0, ROI does work and other member value of setROI is ignored.

ctuOptParam
the section called "HevcCtuOptParam" for H.265/HEVC encoding

forcePicQpEnable
A flag to use a force picture quantization parameter

forcePicQpSrcOrderEnable
A flag to use a force picture QP by source index order

forcePicQpI
A force picture quantization parameter for I picture

forcePicQpP
A force picture quantization parameter for P picture

forcePicQpB
A force picture quantization parameter for B picture

forcePicTypeEnable
A flag to use a force picture type

forcePicTypeSrcOrderEnable
A flag to use a force picture type by source index order

forcePicType
A force picture type (I, P, B, IDR, CRA)

srcIdx
A source frame buffer index

srcEndFlag
A flag indicating that there is no more source frame buffer to encode

codeOption
the section called "EncCodeOpt"

useCurSrcAsLongtermPic
A flag for the current picture to be used as a longterm reference picture later when other picture's encoding

useLongtermRef
A flag to use a longterm reference picture in DPB when encoding the current picture

pts
The presentation Timestamp (PTS) of input source

customMapOpt
the section called "HevcCustomMapOpt"

EncReportInfo

```
typedef struct {
    int enable;
    int type;
    int sz;
    PhysicalAddress addr;
} EncReportInfo;
```

Description

This structure is used for reporting encoder information.

enable
- 0 : reporting disable
- 1 : reporting enable

type
This value is used for picture type reporting in MVInfo.

sz
This value means size for each reporting data (MBinfo, MVinfo, Sliceinfo).

addr
The start address of each reporting buffer into which encoder puts data.

EncBwMonitor

```
typedef struct {
    osal_file_t fpBW;
    Uint32 prpBwRead;
    Uint32 prpBwWrite;
    Uint32 fbdYRead;
    Uint32 fbcYWrite;
    Uint32 fbdCRead;
    Uint32 fbcCWrite;
    Uint32 priBwRead;
    Uint32 priBwWrite;
    Uint32 secBwRead;
    Uint32 secBwWrite;
    Uint32 procBwRead;
    Uint32 procBwWrite;
}EncBwMonitor;
```

Description

This structure is used for reporting bandwidth (only for WAVE5).

EncOutputInfo

```
typedef struct {
    PhysicalAddress bitstreamBuffer;
    Uint32 bitstreamSize;
    int bitstreamWrapAround;
    int picType;
    int numOfSlices;
    int reconFrameIndex;
    FrameBuffer reconFrame;
    int rdPtr;
    int wrPtr;
#ifdef AUTO_FRM_SKIP_DROP
    int picDropped;
#endif

    // for WAVE420
    int picSkipped;
    int numOfIntra;
    int numOfMerge;
    int numOfSkipBlock;
    int avgCtuQp;
    int encPicByte;
    int encGopPicIdx;
    int encPicPoc;
    int encSrcIdx;
    int encVclNal;
    int encPicCnt;
    // Report Information
    EncReportInfo mbInfo;
    EncReportInfo mvInfo;
    EncReportInfo sliceInfo;
    int frameCycle;
    Uint64      pts;
    Uint32      encInstIdx;
    Uint32  encPrepareCycle;
    Uint32  encProcessingCycle;
    Uint32  encEncodingCycle;
```

```
} EncOutputInfo;
```

Description

This is a data structure for reporting the results of picture encoding operations.

bitstreamBuffer
The Physical address of the starting point of newly encoded picture stream

If dynamic buffer allocation is enabled in line-buffer mode, this value is identical with the specified picture stream buffer address by HOST.

bitstreamSize
The byte size of encoded bitstream

bitstreamWrapAround
This is a flag to indicate whether bitstream buffer operates in ring buffer mode in which read and write pointer are wrapped arounded. If this flag is 1, HOST application needs a larger buffer.

picType
Coded picture type
- H.263 and MPEG4
  - 0 : I picture
  - 1 : P picture
- H.264/AVC
  - 0 : IDR picture
  - 1 : Non-IDR picture

numOfSlices
The number of slices of the currently being encoded Picture

reconFrameIndex
A reconstructed frame index. The reconstructed frame is used for reference of future frame.

reconFrame
A reconstructed frame address and information. Please refer to *the section called "Frame-Buffer"*.

rdPtr
A read pointer in bitstream buffer, which is where HOST has read encoded bitstream from the buffer

wrPtr
A write pointer in bitstream buffer, which is where VPU has written encoded bitstream into the buffer

picDropped
A flag which represents whether the current encoding has been dropped or not.

picSkipped
A flag which represents whether the current encoding has been skipped or not.

numOfIntra
The number of intra coded block

numOfMerge
The number of merge block in 8x8

numOfSkipBlock
The number of skip block in 8x8

avgCtuQp
The average value of CTU QPs

encPicByte
The number of encoded picture bytes

encGopPicIdx
GOP index of the current picture

encPicPoc
POC(picture order count) value of the current picture

encSrcIdx
Source buffer index of the current encoded picture

encVclNal
Encoded NAL unit type of VCL

encPicCnt
Encoded picture number

mbInfo
The parameter for reporting MB data . Please refer to *the section called "EncReportInfo"* structure.

mvInfo
The parameter for reporting motion vector. Please refer to *the section called "EncReportInfo"* structure.

sliceInfo
The parameter for reporting slice information. Please refer to *the section called "EncReportInfo"* structure.

frameCycle
The parameter for reporting the cycle number of decoding/encoding one frame.

pts
Presentation Timestamp of encoded picture.

encInstIdx
An index of instance which have finished encoding with a picture at this command. This is only for Multi-Core encoder product.

EncParamSet

```
typedef struct {
   PhysicalAddress paraSet;
   int size;
} EncParamSet;
```

Description

This structure is used when HOST processor additionally wants to get SPS data or PPS data from encoder instance. The resulting SPS data or PPS data can be used in real applications as a kind of out-of-band information.

paraSet
An array of 32 bits which contains SPS RBSP

size
The size of stream in byte

EncHeaderParam

```
typedef struct {
   PhysicalAddress buf;
   BYTE *pBuf;
   size_t  size;
   Int32   headerType;
   BOOL    zeroPaddingEnable;
   Int32   failReasonCode;
} EncHeaderParam;
```

Description

This structure is used for adding a header syntax layer into the encoded bit stream. The parameter headerType is the input parameter to VPU, and the other two parameters are returned values from VPU after completing requested operation.

buf
A physical address pointing the generated stream location

pBuf
The virtual address according to buf. This address is needed when a HOST wants to access encoder header bitstream buffer.

size
The size of the generated stream in bytes

headerType
This is a type of header that HOST wants to generate and have values as VOL_HEADER, VOS_HEADER, VO_HEADER, SPS_RBSP or PPS_RBSP.

zeroPaddingEnable
It enables header to be padded at the end with zero for byte alignment.

failReasonCode
Not defined yet

Chapter 3
API DEFINITIONS

VPU_IsBusy()

Prototype

```
Int32 VPU_IsBusy (
                    Uint32 coreIdx
                  );
```

Description

This function returns whether processing a frame by VPU is completed or not.

Parameter

Parameter	Type	Description
coreIdx	input	An index of VPU core

Return Value

- 0 : VPU hardware is idle.
- Non-zero value : VPU hardware is busy with processing a frame.

VPU_WaitInterrupt()

Prototype

```
Int32 VPU_WaitInterrupt (
                          Uint32 coreIdx,
                          int timeout
                        );
```

Description

This function makes HOST application wait until VPU finishes processing a frame, or check a busy flag of VPU during the given timeout period. The behavior of this function depends on VDI layer's implementation.

Note Timeout may not work according to implementation of VDI layer.

Parameter

Parameter	Type	Description
coreIdx	input	An index of VPU core
timeout	output	See return value.

Return Value

- 1 : Wait time out
- Non -1 value : The value of InterruptBit

VPU_IsInit()

Prototype

```
Int32 VPU_IsInit (
                    Uint32 coreIdx
                  );
```

Description

This function returns whether VPU is currently running or not.

Parameter

Parameter	Type	Description
coreIdx	input	An index of VPU core

Return Value

- 0 : VPU is not running.
- 1 or more : VPU is running.

VPU_Init()

Prototype

```
RetCode VPU_Init (
                  Uint32 coreIdx
                );
```

Description

This function initializes VPU hardware and its data structures/resources. HOST application must call this function only once before calling VPU_DeInit().

Note Before use, HOST application needs to define the header file path of BIT firmware to BIT_CODE_FILE_PATH.

Parameter

Parameter	Type	Description
coreIdx	input	An index of VPU core. This value can be from 0 to (number of VPU core - 1).

VPU_InitWithBitcode()

Prototype

```
RetCode VPU_InitWithBitcode (
                                Uint32 coreIdx,
                                const Uint16 *bitcode,
                                Uint32 sizeInWord
                              );
```

Description

This function initializes VPU hardware and its data structures/resources. HOST application must call this function only once before calling VPU_DeInit().

VPU_InitWithBitcodec() is basically same as VPU_Init() except that it takes additional arguments, a buffer pointer where BIT firmware binary is located and the size. HOST application can use this function when they wish to load a binary format of BIT firmware, instead of it including the header file of BIT firmware. Particularly in multi core running environment with different VPU products, this function must be used because each core needs to load different firmware.

Parameter

Parameter	Type	Description
coreIdx	Input	An index of VPU core
bitcode	Input	Buffer where binary format of BIT firmware is located
sizeInWord	Input	Size of binary BIT firmware in short integer

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means VPU has been initialized successfully.

RETCODE_CALLED_BEFORE
This function call is invalid which means multiple calls of the current API function for a given instance are not allowed. In this case, VPU has been already initialized, so that this function call is meaningless and not allowed anymore.

RETCODE_NOT_FOUND_BITCODE_PATH
The header file path of BIT firmware has not been defined.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not received any response from VPU and has timed out.

RETCODE_FAILURE
Operation was failed.

VPU_DeInit()

Prototype

```
RetCode VPU_DeInit (
                    Uint32 coreIdx
                   );
```

Description

This function frees all the resources allocated by VPUAPI and releases the device driver. VPU_Init() and VPU_DeInit() always work in pairs.

Parameter

Parameter	Type	Description
coreIdx	Input	An index of VPU core

Return Value

none

VPU_GetOpenInstanceNum()

Prototype

```
int VPU_GetOpenInstanceNum (
                              Uint32 coreIdx
                            );
```

Description

This function returns the number of instances opened.

Parameter

Parameter	Type	Description
coreIdx	Input	An index of VPU core

Return Value

The number of instances opened

VPU_GetVersionInfo()

Prototype

```
RetCode VPU_GetVersionInfo (
                              Uint32 coreIdx,
                              Uint32 *versionInfo,
                              Uint32 *revision,
                              Uint32 *productId
                            );
```

Description

This function returns the product information of VPU which is currently running on the system.

Parameter

Parameter	Type	Description
coreIdx	Input	An index of VPU core
versionInfo	Output	• Version_number[15:12] - Major revision • Version_number[11:8] - Hardware minor revision • Version_number[7:0] - Software minor revision
revision	Output	Revision information
productId	Output	Product information. Refer to the *the section called "ProductId"* enumeration

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means version information is acquired successfully.

RETCODE_FAILURE
Operation was failed, which means the current firmware does not contain any version information.

RETCODE_NOT_INITIALIZED
VPU was not initialized at all before calling this function. Application should initialize VPU by calling VPU_Init() before calling this function.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding operation was not completed yet, so the given API function call cannot be performed this time. A frame-decoding operation should be completed by calling VPU_Dec Info(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not received any response from VPU and has timed out.

VPU_ClearInterrupt()

Prototype

```
void VPU_ClearInterrupt (
                           Uint32 coreIdx
                         );
```

Description

This function clears VPU interrupts that are pending.

Parameter

Parameter	Type	Description
coreIdx	Input	An index of VPU core

Return Value

none

VPU_SWReset()

Prototype

```
RetCode VPU_SWReset (
                      Uint32 coreIdx,
                      SWResetMode resetMode,
                      void *pendingInst
                    );
```

Description

This function stops operation of the current frame and initializes VPU hardware by sending reset signals. It can be used when VPU is having a longer delay or seems hang-up. After VPU has completed initialization, the context is rolled back to the state before calling the previous VPU_DecStartOneFrame() or VPU_EncStartOneFrame(). HOST can resume decoding from the next picture, instead of decoding from the sequence header. It works only for the current instance, so this function does not affect other instance's running in multi-instance operation.

This is some applicable scenario of using `VPU_SWReset()` when a series of hang-up happens. For example, when VPU is hung up with frame 1, HOST application calls `VPU_SWReset()` to initialize VPU and then calls `VPU_DecStartOneFrame()` for frame 2 with specifying the start address, read pointer. If there is still problem with frame 2, we recommend calling `VPU_SWReset()` and `seq_init()` or calling `VPU_SWReset()` and enabling iframe search.

Parameter

Parameter	Type	Description
coreIdx	Input	An index of VPU core
resetMode	Input	Way of reset • SW_RESET_SAFETY : It waits until AXI bus completes ongoing tasks. If remaining bus transactions are done, VPU enters the reset process. (recommended mode) • SW_RESET_FORCE : It forces to reset VPU no matter whether bus transactions are completed or not. It might affect what other blocks do with bus, so we do not recommend using this mode. • SW_RESET_ON_BOOT : This is the default reset mode that is executed once system boots up. This mode is actually executed in VPU_Init(), so does not have to be used independently.
pendingInst		

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means required information of the stream data to be decoded was received successfully.

VPU_HWReset()

Prototype

```
RetCode VPU_HWReset (
                      Uint32 coreIdx
                    );
```

Description

This function resets VPU as VPU_SWReset() does, but it is done by the system reset signal and all the internal contexts are initialized. Therefore after the `VPU_HWReset()`, HOST application needs to call `VPU_Init()`.

`VPU_HWReset()` requires vdi_hw_reset part of VDI module to be implemented before use.

Parameter

Parameter	Type	Description
coreIdx	Input	An index of VPU core

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means required information of the stream data to be decoded was received successfully.

VPU_SleepWake()

Prototype

```
RetCode VPU_SleepWake (
                        Uint32 coreIdx,
                        int iSleepWake
                      );
```

Description

This function saves or restores context when VPU powers on/off.

Note This is a tip for safe operation - call this function to make VPU enter into a sleep state before power down, and after the power off call this function again to return to a wake state.

Parameter

Parameter	Type	Description
coreIdx	Input	An index of VPU core
iSleepWake	Input	• 1 : saves all of the VPU contexts and converts into a sleep state. • 0 : restores all of the VPU contexts and converts back to a wake state.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means required information of the stream data to be decoded was received successfully.

VPU_GetMvColBufSize()

Prototype

```
int VPU_GetMvColBufSize (
                           CodStd codStd,
                           int width,
                           int height,
                           int num
                         );
```

Description

This function returns the size of motion vector co-located buffer that are needed to decode H.265/AVC stream. The mvcol buffer should be allocated along with frame buffers and given to VPU_DecRegisterFramebuffer() as an argument.

Parameter

Parameter	Type	Description
codStd	Input	Video coding standards
width	Input	Width of framebuffer
height	Input	Height of framebuffer
num	Input	Number of framebuffers.

Return Value

It returns the size of required mvcol buffer in byte unit.

VPU_GetFBCOffsetTableSize()

Prototype

```
RetCode VPU_GetFBCOffsetTableSize (
                                     CodStd codStd,
                                     int width,
                                     int height,
                                     int *ysize,
                                     int *csize
                                   );
```

Description

This function returns the size of FBC (Frame Buffer Compression) offset table for luma and chroma. The offset tables are to look up where compressed data is located. HOST should allocate the offset table buffers for luma and chroma as well as frame buffers and give their base addresses to VPU_DecRegisterFramebuffer() as an argument.

Parameter

Parameter	Type	Description
codStd	Input	Video coding standards
width	Input	Width of framebuffer
height	Input	Height of framebuffer
ysize	Output	Size of offset table for Luma in bytes
csize	Output	Size of offset table for Chroma in bytes

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means given value is valid and setting is done successfully.

RETCODE_INVALID_PARAM
The given argument parameter, ysize, csize, or handle was NULL.

VPU_GetFrameBufSize()

Prototype

```
int VPU_GetFrameBufSize (
                          int coreIdx,
                          int stride,
                          int height,
                          int mapType,
                          int format,
                          int interleave,
                          DRAMConfig *pDramCfg
                        );
```

Description

This function returns the size of frame buffer that is required for VPU to decode or encode one frame.

Parameter

Parameter	Type	Description
coreIdx	Input	VPU core index number
stride	Input	The stride of image
height	Input	The height of image
mapType	Input	The map type of framebuffer
format	Input	The color format of framebuffer
interleave	Input	Whether to use CBCR interleave mode or not
pDramCfg	Input	Attributes of DRAM. It is only valid for CODA960. Set NULL for this variable in case of other products.

Return Value

The size of frame buffer to be allocated

VPU_GetProductId()

Prototype

```
int VPU_GetProductId (
                        int coreIdx
                      );
```

Description

This function returns the product ID of VPU which is currently running.

Parameter

Parameter	Type	Description
coreIdx	Input	VPU core index number

Return Value

Product information. Please refer to the *the section called "ProductId"* enumeration.

VPU_DecOpen()

Prototype

```
RetCode VPU_DecOpen (
                      DecHandle *pHandle,
                      DecOpenParam *pop
                    );
```

Description

In order to decode, HOST application must open the decoder. By calling this function, HOST application can get a handle by which they can refer to a decoder instance. Because the VPU is multiple instance codec, HOST application needs this kind of handle. Once a HOST application gets a handle, the HOST application must pass this handle to all subsequent decoder-related functions.

Parameter

Parameter	Type	Description
pHandle	Output	A pointer to a DecHandle type variable which specifies each instance for HOST application.
pop	Input	A pointer to a DecOpenParam type structure which describes required parameters for creating a new decoder instance.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means a new decoder instance was created successfully.

RETCODE_FAILURE
Operation was failed, which means getting a new decoder instance was not done successfully. If there is no free instance anymore, this value is returned in this function call.

RETCODE_INVALID_PARAM
The given argument parameter, pOpenParam, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

RETCODE_NOT_INITIALIZED
This means VPU was not initialized yet before calling this function. Applications should initialize VPU by calling VPU_Init() before calling this function.

VPU_DecClose()

Prototype

```
RetCode VPU_DecClose (
                        DecHandle handle
                      );
```

Description

When HOST application finished decoding a sequence and wanted to release this instance for other processing, the applicaton should close this instance. After completion of this function call, the instance referred to by handle gets free. Once a HOST application closes an instance, the HOST application cannot call any further decoder-specific function with the current handle before re-opening a new decoder instance with the same handle.

Parameter

Parameter	Type	Description
handle	Output	A decoder handle obtained from VPU_DecOpen()

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means the current decoder instance was closed successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, handle, was invalid. This return code might be caused if

- `handle` is not the handle which has been obtained by VPU_DecOpen()
- `handle` is the handle of an instance which has been closed already, etc.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not received any response from VPU and has timed out.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding operation was not completed yet, so the given API function call cannot be performed this time. A frame decoding operation should be completed by calling VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_DecGetOutputInfo() to proceed this function call. function.

VPU_DecGetInitialInfo()

Prototype

```
RetCode VPU_DecGetInitialInfo (
                                  DecHandle handle,
                                  DecInitialInfo *info
                                );
```

Description

Applications must pass the address of *the section called "DecInitialInfo"* structure, where the decoder stores information such as picture size, number of necessary frame buffers, etc. For the details, see definition of *the section called "DecInitialInfo"* data structure. This function should be called once after creating a decoder instance and before starting frame decoding.

It is a HOST application's responsibility to provide sufficient amount of bitstream to the decoder by calling VPU_DecUpdateBitstreamBuffer() so that bitstream buffer does not get empty before this function returns. If HOST application cannot ensure to feed stream enough, they can use the Forced Escape option by using VPU_DecSetEscSeqInit().

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
info	Output	A pointer to *the section called "DecInitialInfo"* data structure

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means required information of the stream data to be decoded was received successfully.

RETCODE_FAILURE
Operation was failed, which means there was an error in getting information for configuring the decoder.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not a handle which has been obtained by VPU_DecOpen().
- `handle` is a handle of an instance which has been closed already, etc.

RETCODE_INVALID_PARAM
The given argument parameter, pInfo, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding operation was not completed yet, so the given API function call cannot be performed this time. A frame decoding operation should be completed by calling VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST should call VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. In this case, HOST might call this function before successfully

putting bitstream data by calling VPU_DecUpdateBitstreamBuffer(). In order to perform this functions call, bitstream data including sequence level header should be transferred into bitstream buffer before calling VPU_DecGetInitialInfo().

RETCODE_CALLED_BEFORE
This function call might be invalid, which means multiple calls of the current API function for a given instance are not allowed. In this case, decoder initial information has been already received, so that this function call is meaningless and not allowed anymore.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not received any response from VPU and has timed out.

Note When this function returns RETCODE_SUCCESS, HOST should call VPU_ClearInterrupt() function to clear out the interrupt pending status.

VPU_DecIssueSeqInit()

Prototype

```
RetCode VPU_DecIssueSeqInit (
                                DecHandle handle
                            );
```

Description

This function starts decoding sequence header. Returning from this function does not mean the completion of decoding sequence header, and it is just that decoding sequence header was initiated. Every call of this function should be matched with VPU_DecCompleteSeqInit() with the same handle. Without calling a pair of these funtions, HOST can not call any other API functions except for VPU_IsBusy(), VPU_DecGetBitstreamBuffer(), and VPU_DecUpdateBitstreamBuffer().

A pair of this function and VPU_DecCompleteSeqInit() or VPU_DecGetInitialInfo() should be called at least once after creating a decoder instance and before starting frame decoding.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means required information of the stream data to be decoded was received successfully

RETCODE_FAILURE
Operation was failed, which means there was an error in getting information for configuring the decoder.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not a handle which has been obtained by VPU_DecOpen().
- `handle` is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding operation was not completed yet, so the given API function call cannot be performed this time. A frame decoding operation should be completed by calling VPU_DecIssueSeqInit (). Even though the result of the current frame operation is not necessary, HOST should call VPU_DecIssueSeqInit () to proceed this function call.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. In this case, HOST application might call this function before successfully putting bitstream data by calling VPU_DecUpdateBitstreamBuffer(). In order to perform this functions call, bitstream data including sequence level header should be transferred into bitstream buffer before calling VPU_ DecIssueSeqInit ().

VPU_DecCompleteSeqInit()

Prototype

```
RetCode VPU_DecCompleteSeqInit (
                                  DecHandle handle,
                                  DecInitialInfo *info
                                );
```

Description

Application can get the information about sequence header. Applications must pass the address of DecInitialInfo structure, where the decoder stores information such as picture size, number of necessary frame buffers, etc. For more details, see definition of the section called the DecInitialInfo data structure.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
info	Output	A pointer to DecInitialInfo data structure

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means required information of the stream data to be decoded was received successfully.

RETCODE_FAILURE
Operation was failed, which means there was an error in getting information for configuring the decoder.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not a handle which has been obtained by VPU_DecOpen().
- `handle` is a handle of an instance which has been closed already, etc.

RETCODE_INVALID_PARAM
The given argument parameter, pInfo, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. It might happen because VPU_DecIssueSeqInit () with the same handle was not called before calling this function

RETCODE_CALLED_BEFORE
This function call might be invalid, which means multiple calls of the current API function for a given instance are not allowed. In this case, decoder initial information has been already received, so that this function call is meaningless and not allowed anymore.

VPU_DecSetEscSeqInit()

Prototype

```
RetCode VPU_DecSetEscSeqInit (
                                DecHandle handle,
                                int escape
                              );
```

Description

This is a special function to provide a way of escaping VPU hanging during DEC_SEQ_INIT. When this flag is set to 1 and stream buffer empty happens, VPU terminates automatically DEC_SEQ_INIT operation. If target applications ensure that high layer header syntax is periodically sent through the channel, they do not need to use this option. But if target applications cannot ensure that such as file-play, it might be very useful to avoid VPU hanging without HOST timeout caused by crucial errors in header syntax.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
escape	Input	A flag to enable or disable forced escape from SEQ_INIT

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means Force escape flag is successfully provided to BIT Processor.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not the handle which has been obtained by VPU_DecOpen().
- `handle` is the handle of an instance which has been closed already, etc.
- bitstreamMode of DecOpenParam structure is not BS_MODE_INTERRUPT.

VPU_DecRegisterFrameBuffer()

Prototype

```
RetCode VPU_DecRegisterFrameBuffer (
                                      DecHandle handle,
                                      FrameBuffer *bufArray,
                                      int num,
                                      int stride,
                                      int height,
                                      int mapType
                                    );
```

Description

This function is used for registering frame buffers with the acquired information from VPU_DecGetInitialInfo(). The frame buffers pointed to by bufArray are managed internally within VPU. These include reference frames, reconstructed frame, etc. Applications must not change the contents of the array of frame buffers during the life time of the instance, and num must not be less than minFrameBufferCount obtained by VPU_DecGetInitialInfo().

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
bufArray	Input	Allocated frame buffer address and information. If this parameter is set to 0, VPU allocates frame buffers.
num	Input	A number of frame buffers. VPU can allocate frame buffers as many as this given value.
stride	Input	A stride value of the given frame buffers
height	Input	Frame height
mapType	Input	Map type of frame buffer The distance between a pixel in a row and the corresponding pixel in the next row is called a stride. It comes from 64-bit access unit of AXI. The stride for luminance frame buffer should be at least equal or greater than the width of picture and a multiple of 8. It means the least significant 3-bit of the 13-bit stride should be always 0. The stride for chrominance frame buffers is the half of the luminance stride. So in case Cb/Cr non-interleave (separate Cb/Cr) map is used, make sure the stride for luminance frame buffer should be a multiple of 16 so that the stride for chrominance frame buffer can become a multiple of 8.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means registering frame buffer information was done successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not a handle which has been obtained by VPU_DecOpen().

- `handle` is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding operation was not completed yet, so the given API function call cannot be performed this time. A frame decoding operation should be completed by calling VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST should call VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. A HOST might call this function before calling VPU_DecGetInitialInfo() successfully. This function should be called after successful calling VPU_DecGetInitialInfo().

RETCODE_INVALID_FRAME_BUFFER
This happens when pBuffer was invalid, which means pBuffer was not initialized yet or not valid anymore.

RETCODE_INSUFFICIENT_FRAME_BUFFERS
This means the given number of frame buffers, num, was not enough for the decoder operations of the given handle. It should be greater than or equal to the value requested by VPU_DecGetInitialInfo().

RETCODE_INVALID_STRIDE
The given argument stride was invalid, which means it is smaller than the decoded picture width, or is not a multiple of 8 in this case.

RETCODE_CALLED_BEFORE
This function call is invalid which means multiple calls of the current API function for a given instance are not allowed. In this case, registering decoder frame buffers has been already done, so that this function call is meaningless and not allowed anymore.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not recieved any response from VPU and has timed out.

VPU_DecRegisterFrameBufferEx()

Prototype

```
RetCode VPU_DecRegisterFrameBufferEx (
                                        DecHandle handle,
                                        FrameBuffer *bufArray,
                                        int numOfDecFbs,
                                        int numOfDisplayFbs,
                                        int stride,
                                        int height,
                                        int mapType
                                      );
```

Description

This function is used for registering frame buffers with the acquired information from VPU_DecGetInitialInfo(). This function is functionally same as VPU_DecRegisterFrameBuffer(), but it can give linear (display) frame buffers and compressed buffers separately with different numbers unlike the way VPU_DecRegisterFrameBuffer() does. VPU_DecRegisterFrameBuffer() assigns only the same number of frame buffers for linear buffer and for compressed buffer, which can take up huge memory space.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
bufArray	Input	Allocated frame buffer address and information. If this parameter is set to 0, VPU allocates frame buffers.
numOfDecFbs	Input	Number of compressed frame buffer
numOfDisplayFbs	Input	Number of linear frame buffer when WTL is enabled. In WAVE4, this should be equal to or larger than framebufDelay of *the section called "DecInitialInfo"* + 2.
stride	Input	A stride value of the given frame buffers
height	Input	Frame height
mapType	Input	A Map type for GDI inferface or FBC (Frame Buffer Compression) For detailed map options, please refer to the *the section called "TiledMapType"*.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means registering frame buffer information was done successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not a handle which has been obtained by VPU_DecOpen().
- `handle` is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding operation was not completed yet, so the given API function call cannot be performed this time. A frame decoding operation should be completed by

calling VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST should call VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. A HOST might call this function before calling VPU_DecGetInitialInfo() successfully. This function should be called after successful calling VPU_DecGetInitialInfo().

RETCODE_INVALID_FRAME_BUFFER
This happens when pBuffer was invalid, which means pBuffer was not initialized yet or not valid anymore.

RETCODE_INSUFFICIENT_FRAME_BUFFERS
This means the given number of frame buffers, num, was not enough for the decoder operations of the given handle. It should be greater than or equal to the value requested by VPU_DecGetInitialInfo().

RETCODE_INVALID_STRIDE
The given argument stride was invalid, which means it is smaller than the decoded picture width, or is not a multiple of 8 in this case.

RETCODE_CALLED_BEFORE
This function call is invalid which means multiple calls of the current API function for a given instance are not allowed. In this case, registering decoder frame buffers has been already done, so that this function call is meaningless and not allowed anymore.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not recieved any response from VPU and has timed out.

VPU_DecAllocateFrameBuffer()

Prototype

```
RetCode VPU_DecAllocateFrameBuffer (
                                      DecHandle handle,
                                      FrameBufferAllocInfo info,
                                      FrameBuffer *frameBuffer
                                    );
```

Description

This is a special function that enables HOST to allocate directly the frame buffer for decoding (Recon) or for display or post-processor unit (PPU) such as Rotator or Tiled2Linear. In normal operation, VPU API allocates frame buffers when the argument bufArray in VPU_DecRegisterFrameBuffer() is set to 0. However, for any other reason HOST can use this function to allocate frame buffers by themselves.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
info	Input	Information required for frame bufer allocation
frameBuffer	Output	Data structure that holds information of allocated frame buffers

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means the framebuffer is allocated successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, handle, was invalid. This return code might be caused if

- `handle` is not the handle which has been obtained by VPU_DecOpen().
- `handle` is the handle of an instance which has been closed already, etc.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. It might happen because VPU_DecRegisterFrameBuffer() for (FramebufferAllocType.FB_TYPE_CODEC) has not been called, before this function call for allocating frame buffer for PPU (FramebufferAllocType.FB_TYPE_PPU).

RETCODE_INSUFFICIENT_RESOURCE
Fail to allocate a framebuffer due to lack of memory

RETCODE_INVALID_PARAM
The given argument parameter, index, was invalid, which means it has improper values.

VPU_DecGetFrameBuffer()

Prototype

```
RetCode VPU_DecGetFrameBuffer (
                                  DecHandle handle,
                                  int frameIdx,
                                  FrameBuffer *frameBuf
                                );
```

Description

This function returns the frame buffer information that was allocated by VPU_DecRegisterFrameBuffer() function.

It does not affect actual decoding and simply does obtain the information of frame buffer. This function is more helpful especially when frame buffers are automatically assigned by setting 0 to `bufArray` of VPU_DecRegisterFrameBuffer() and HOST wants to know about the allocated frame buffer.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
frameIdx	Input	An index of frame buffer
frameBuf	output	Allocated frame buffer address and information.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means registering frame buffer information was done successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not a handle which has been obtained by VPU_DecOpen().
- `handle` is a handle of an instance which has been closed already, etc.

RETCODE_INVALID_PARAM
The given argument parameter, frameIdx, was invalid, which means frameIdx is larger than allocated framebuffer.

VPU_DecGetBitstreamBuffer()

Prototype

```
RetCode VPU_DecGetBitstreamBuffer (
                                     DecHandle handle,
                                     PhysicalAddress *prdPrt,
                                     PhysicalAddress *pwrPtr,
                                     Uint32 *size
                                   );
```

Description

Before decoding bitstream, HOST application must feed the decoder with bitstream. To do that, HOST application must know where to put bitstream and the maximum size. Applications can get the information by calling this function. This way is more efficient than providing arbitrary bitstream buffer to the decoder as far as VPU is concerned.

The given size is the total sum of free space in ring buffer. So when a HOST application downloads this given size of bitstream, Wrptr could meet the end of stream buffer. In this case, the HOST application should wrap-around the Wrptr back to the beginning of stream buffer, and download the remaining bits. If not, decoder operation could be crashed.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
prdPrt	Output	A stream buffer read pointer for the current decoder instance
pwrPtr	Output	A stream buffer write pointer for the current decoder instance
size	Output	A variable specifying the available space in bitstream buffer for the current decoder instance

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means required information for decoder stream buffer was received successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not the handle which has been obtained by VPU_DecOpen().
- `handle` is the handle of an instance which has been closed already, etc.

RETCODE_INVALID_PARAM
The given argument parameter, pRdptr, pWrptr or size, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

VPU_DecUpdateBitstreamBuffer()

Prototype

```
RetCode VPU_DecUpdateBitstreamBuffer (
                                        DecHandle handle,
                                        int size
                                      );
```

Description

Applications must let decoder know how much bitstream has been transferred to the address obtained from VPU_DecGetBitstreamBuffer(). By just giving the size as an argument, API automatically handles pointer wrap-around and updates the write pointer.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
size	Input	A variable specifying the amount of bits transferred into bitstream buffer for the current decoder instance. • 0 : It means that no more bitstream exists to feed (end of stream). If 0 is set for `size`, VPU decodes just remaing bitstream and returns -1 to `indexFrameDisplay`. • -1: It allows VPU to continue to decode without VPU_DecClose(), even after remaining stream has completely been decoded by VPU_DecUpdateBitstreamBuffer(handle, 0). This is especially useful when a sequence changes in the middle of decoding. • -2 : It is for handling exception cases like error stream or failure of finding frame boundaries in interrupt mode. If such cases happen, VPU is unable to continue decoding. If -2 is set for `size`, VPU decodes until the current write pointer in the bitstream buffer and then force to end decoding. (WAVE only)

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means required information for decoder stream buffer was received successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not the handle which has been obtained by VPU_DecOpen().
- `handle` is the handle of an instance which has been closed already, etc.

RETCODE_INVALID_PARAM
The given argument parameter, pRdptr, pWrptr or size, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

RETCODE_SUCCESS
Operation was done successfully, which means putting new stream data was done successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not the handle which has been obtained by VPU_DecOpen().
- `handle` is the handle of an instance which has been closed already, etc.

RETCODE_INVALID_PARAM
The given argument parameter, size, was invalid, which means size is larger than the value obtained from VPU_DecGetBitstreamBuffer(), or than the available space in the bitstream buffer.

VPU_DecSetRdPtr()

Prototype

```
RetCode VPU_DecSetRdPtr (
                           DecHandle handle,
                           PhysicalAddress addr,
                           int updateWrPtr
                         );
```

Description

This function specifies the location of read pointer in bitstream buffer. It can also set a write pointer with same value of read pointer (addr) when updateWrPtr is not a zero value, which allows to flush up the bitstream buffer at once. This function is used to operate bitstream buffer in PicEnd mode.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
addr	Input	Updated read or write pointer
updateWrPtr	Input	A flag whether to move the write pointer to where the read pointer is located

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means required information of the stream data to be decoded was received successfully.

RETCODE_FAILURE
Operation was failed, which means there was an error in getting information for configuring the decoder.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not a handle which has been obtained by VPU_DecOpen().
- `handle` is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding operation was not completed yet, so the given API function call cannot be performed this time. A frame decoding operation should be completed by calling VPU_ DecSetRdPtr ().

VPU_DecGiveCommand()

Prototype

```
RetCode VPU_DecGiveCommand (
                              DecHandle handle,
                              CodecCommand cmd,
                              void *parameter
                            );
```

Description

This function is provided to let HOST have a certain level of freedom for re-configuring decoder operation after creating a decoder instance. Some options which can be changed dynamically during decoding as the video sequence has been included. Some command-specific return codes are also presented.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
cmd	Input	A variable specifying the given command of CodecComand type
parameter	Input/Output	A pointer to command-specific data structure which describes picture I/O parameters for the current decoder instance

Return Value

RETCODE_INVALID_COMMAND
The given argument, cmd, was invalid, which means the given cmd was undefined, or not allowed in the current instance.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not a handle which has been obtained by VPU_DecOpen().
- `handle` is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding operation was not completed yet, so the given API function call cannot be performed this time. A frame decoding operation should be completed by calling VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_DecGetOutputInfo() to proceed this function call. values.

Command

The list of commands can be summarized as follows:

- ENABLE_ROTATION
- DIABLE_ROTATION
- ENABLE_MIRRORING
- DISABLE_MIRRORING

- ENABLE_DERING
- DISABLE_DERING
- SET_MIRROR_DIRECTION
- SET_ROTATION_ANGLE
- SET_ROTATOR_OUTPUT
- SET_ROTATOR_STRIDE
- ENABLE_DEC_THUMBNAIL_MODE,
- DEC_SET_SPS_RBSP
- DEC_SET_PPS_RBSP
- ENABLE_REP_USERDATA
- DISABLE_REP_USERDATA
- SET_ADDR_REP_USERDATA
- SET_VIRT_ADDR_REP_USERDATA
- SET_SIZE_REP_USERDATA
- SET_USERDATA_REPORT_MODE
- SET_SEC_AXI
- SET_DRAM_CONFIG
- GET_DRAM_CONFIG
- ENABLE_REP_BUFSTAT
- DISABLE_REP_BUFSTAT
- ENABLE_REP_MBPARAM
- DISABLE_REP_MBPARAM
- ENABLE_REP_MBMV
- DISABLE_REP_MBMV
- SET_ADDR_REP_PICPARAM
- SET_ADDR_REP_BUF_STATE
- SET_ADDR_REP_MBMV_DATA
- SET_CACHE_CONFIG
- GET_TILEDMAP_CONFIG
- SET_LOW_DELAY_CONFIG
- DEC_GET_DISPLAY_OUTPUT_INFO
- SET_DECODE_FLUSH
- DEC_SET_FRAME_DELAY
- DEC_FREE_FRAME_BUFFER
- DEC_GET_FIELD_PIC_TYPE
- DEC_ENABLE_REORDER
- DEC_DISABLE_REORDER
- DEC_GET_FRAMEBUF_INFO
- DEC_RESET_FRAMEBUF_INFO
- DEC_SET_DISPLAY_FLAG
- DEC_GET_SEQ_INFO
- ENABLE_LOGGING
- DISABLE_LOGGING
- DEC_SET_2ND_FIELD_INFO
- DEC_ENABLE_AVC_MC_INTERPOL
- DEC_DISABLE_AVC_MC_INTERPOL

`ENABLE_ROTATION`

This command enables rotation of the post-rotator. In this case, `parameter` is ignored. This command returns RETCODE_SUCCESS.

`DISABLE_ROTATION`

This command disables rotation of the post-rotator. In this case, `parameter` is ignored. This command returns RETCODE_SUCCESS.

ENABLE_MIRRORING

This command enables mirroring of the post-rotator. In this case, `parameter` is ignored. This command returns RETCODE_SUCCESS.

DISABLE_MIRRORING

This command disables mirroring of the post-rotator. In this case, `parameter` is ignored. This command returns RETCODE_SUCCESS.

ENABLE_DERING

This command enables deringring filter of the post-rotator. In this case, `parameter` is ignored. This command returns RETCODE_SUCCESS.

DISABLE_DERING

This command disables deringing filter of the post-rotator. In this case, `parameter` is ignored. This command returns RETCODE_SUCCESS.

SET_MIRROR_DIRECTION

This command sets mirror direction of the post-rotator, and `parameter` is interpreted as a pointer to MirrorDirection. The `parameter` should be one of MIRDIR_NONE, MIRDIR_VER, MIRDIR_HOR, and MIRDIR_HOR_VER.

- MIRDIR_NONE: No mirroring
- MIRDIR_VER: Vertical mirroring
- MIRDIR_HOR: Horizontal mirroring
- MIRDIR_HOR_VER: Both directions

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means given mirroring direction is valid.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid, which means given mirroring direction is invalid.

SET_ROTATION_ANGLE

This command sets counter-clockwise angle for post-rotation, and `parameter` is interpreted as a pointer to the integer which represents rotation angle in degrees. Rotation angle should be one of 0, 90, 180, and 270.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means given rotation angle is valid.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid, which means given rotation angle is invalid.

SET_ROTATOR_OUTPUT

This command sets rotator output buffer address, and `parameter` is interpreted as the pointer of a structure representing physical addresses of YCbCr components of output frame. For storing the rotated output for display, at least one more frame buffer should be allocated. When multiple

display buffers are required, HOST application could change the buffer pointer of rotated output at every frame by issuing this command.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means given rotation angle is valid.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid, which means given frame buffer pointer is invalid.

SET_ROTATOR_STRIDE

This command sets the stride size of the frame buffer containing rotated output, and `parameter` is interpreted as the value of stride of the rotated output.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means given rotation angle is valid.
- RETCODE_INVALID_STRIDE: The given argument parameter, `parameter`, was invalid, which means given value of stride is invalid. The value of stride must be greater than 0 and a multiple of 8.

ENABLE_DEC_THUMBNAIL_MODE

This command decodes only an I-frame of picture from bitstream for using it as a thumbnail. It requires as little as size of frame buffer since I-picture does not need any reference picture. If HOST issues this command and sets one frame buffer address to FrameBuffer array in VPU_DecRegisterFrameBuffer(), only the frame buffer is used. And please make sure that the number of frame buffer `num` should be registered as minFrameBufferCount.

DEC_SET_SPS_RBSP

This command applies SPS stream received from a certain out-of-band reception scheme to the decoder. The stream should be in RBSP format and in big Endian. The argument `parameter` is interpreted as a pointer to DecParamSet structure. In this case, `paraSet` is an array of 32 bits which contains SPS RBSP, and `size` is the size of the stream in bytes.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means transferring an SPS RBSP to decoder was done successfully.
- RETCODE_INVALID_COMMAND: The given argument `cmd` was invalid, which means the given `cmd` was undefined, or not allowed in the current instance. In this case, current instance might not be an H.264/AVC decoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

DEC_SET_PPS_RBSP

This command applies PPS stream received from a certain out-of-band reception scheme to the decoder. The stream should be in RBSP format and in big Endian. The argument `parameter` is interpreted as a pointer to a DecParamSet structure. In this case, paraSet is an array of 32 bits which contains PPS RBSP, and `size` is the size of the stream in bytes.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means transferring a PPS RBSP to decoder was done successfully.
- RETCODE_INVALID_COMMAND: The given argument `cmd` was invalid, which means the given cmd was undefined, or not allowed in the current instance. In this case, current instance might not be an H.264/AVC decoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

ENABLE_REP_USERDATA

This command enables user data report. This command ignores `parameter`.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means enabling user data report is done successfully.
- RETCODE_USERDATA_BUF_NOT_SET: This means user data buffer address and size have not set yet.

DISABLE_REP_USERDATA

This command disables user data report. This command ignores `parameter` and returns RETCODE_SUCCESS.

SET_ADDR_REP_USERDATA

This command sets user data buffer address. `parameter` is interpreted as a pointer to address. This command returns as follows.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means given value of address is valid and setting is done successfully.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` was invalid, which means given value of address is invalid. The value of address must be greater than 0 and a multiple of 8.

SET_VIRT_ADDR_REP_USERDATA

This command sets user data buffer address (virtual address) as well as physical address by using SET_ADDR_REP_USERDATA `parameter` is interpreted as a pointer to address. This command returns as follows.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means given value of address is valid and setting is done successfully.
- RETCODE_USERDATA_BUF_NOT_SET: SET_ADDR_REP_USERDATA command was not been executed
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` was invalid, which means given value of address is invalid. The value of address must be greater than 0 and a multiple of 8.

SET_SIZE_REP_USERDATA

This command sets the size of user data buffer which is set with SET_ADDR_REP_USERDATA command. `parameter` is interpreted as a pointer to the value of size. This command returns RETCODE_SUCCESS.

According to codec standards, user data type means as below.

- H.264/AVC
  - 4 : user_data_registered_itu_t_t35
  - 5 : user_data_unregistered

More details are in Annex D of H.264 specifications.

- VC1
  - 31 : Sequence Level user data
  - 30 : Entry-point Level user data
  - 29 : Frame Level user data
  - 28 : Field Level user data
  - 27 : Slice Level user data
- MPEG2
  - 0 : Sequence user data
  - 1 : GOP user data
  - 2 : Picture user data
- MPEG4
  - 0 : VOS user data
  - 1 : VIS user data
  - 2 : VOL user data
  - 3 : GOV user data

Note This command is available soon.

The user data size 0 - 15 is used to make offset from userDataBuf Base + 8x17. It specifies byte size of user data 0 to 15 excluding 0 padding byte, which exists between user data. So HOST reads 1 user data from userDataBuf Base + 8x17 + 0 User Data Size + 0 Padding. Size of 0 padding is (8 - (User Data Size % 8))%8.

SET_USERDATA_REPORT_MODE

TBD

SET_SEC_AXI

This command sets the secondary channel of AXI for saving memory bandwidth to dedicated memory. The argument `parameter` is interpreted as a pointer to SecAxiUse which represents an enable flag and physical address which is related with the secondary channel for BIT processor, IP/AC-DC predictor, de-blocking filter, overlap filter respectively.

This command has one of the following return codes

- RETCODE_SUCCESS: Operation was done successfully, which means given value for setting secondary AXI is valid.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid, which means given value is invalid.

SET_DRAM_CONFIG

TBD

GET_DRAM_CONFIG

TBD

ENABLE_REP_BUFSTAT

This command enables frame buffer status report. This command ignores `parameter` and returns RETCODE_SUCCESS.

If this option is enabled, frame buffer status is reported. Each frame buffers can be used for display, reference or not used. Decoder reports the status of each frame buffer by using 4 bits [3:0]

- [3] Not used
- [2] USE_DIS
- [1] USE_REF
- [0] Not used

For example, if the value of frame buffer status is 6, then the frame buffer is used for reference and display. If 4, the frame buffer is used for display and is not used for reference.

In H.264/AVC, bit field definition is as bellows:

- [3] Not used
- [2] USE_DIS
- [1] USE_OUT
- [0] USE_REF

If USE_OUT is 1, it means that the Frame is in DPB buffer.

DISABLE_REP_BUFSTAT

This command disables frame buffer status report. This command ignores `parameter` and returns RETCODE_SUCCESS.

ENABLE_REP_MBPARAM

This command enables MB Parameter report. This command ignores `parameter` and returns RETCODE_SUCCESS.

If this option is enabled, error flag, Slice Boundary and QP are reported using 8 bits.

- [7] : Error Map. If error is detected in macroblock decoding, this bit field is set to 1.
- [6] : Slice Boundary. Whenever new slice header is decoded, this bit field is toggled.
- [5:0] : An macroblock QP value

DISABLE_REP_MBPARAM

This command disables MB Parameter report. This command ignores `parameter` and returns RETCODE_SUCCESS.

ENABLE_REP_MV

This command enables MV report. This command ignores `parameter` and returns RETCODE_SUCCESS.

If this option is enabled, decoder reports 1 motion vector for P picture and 2 motion vectors for B picture.

DISABLE_REP_MV

This command disables MV report. This command ignores `parameter` and returns RETCODE_SUCCESS.

SET_ADDR_REP_PICPARAM

This command sets the address of picture parameter base. `parameter` is interpreted as a pointer to a address.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means given value of address is valid and setting is done successfully.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid, which means given value of address is invalid. The value of address must be greater than 0 and a multiple of 8.

To report frame buffer status, MB parameter and Motion vector, VPU reads the base addresses of external memory, which are specified by PicParaBaseAddr.

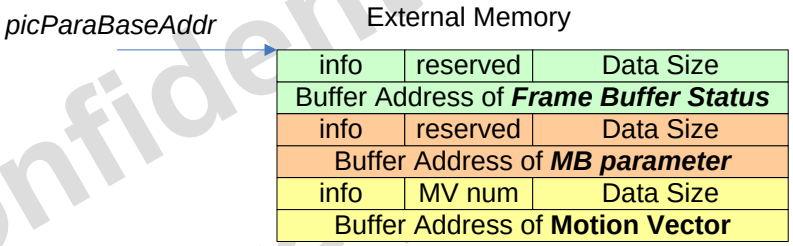


Figure 3.1. Decoder Picture parameter base address structure

When `picUserDataEnable`, `mvReportEnable`, `mbParamEnable` or `frmBufStaEnable` in CMD_DEC_PIC_OPTION register are enabled, HOST application should specify buffer addresses in *Figure 3.1, "Decoder Picture parameter base address structure"*. VPU reports each data and fills info, Data Size and MV num fields to these buffer addresses of external memory. For VPU to report data properly, HOST application needs to specify these 3 buffer addresses preserving 8 byte alignment and buffer sizes need to be multiples of 256.

SET_ADDR_REP_BUF_STATE

This command sets the buffer address of frame buffer status. The `parameter` is interpreted as a pointer to the address.

This command has one of the following return codes

- RETCODE_SUCCESS: Operation was done successfully, which means given value of address is valid and setting is done successfully.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid, which means given value of address is invalid. The value of address must be greater than 0 and a multiple of 8.

SET_ADDR_REP_MBMV_DATA

This command sets the buffer address of motion vector information reporting array. The `parameter` is interpreted as a pointer to the address.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means given value of address is valid and setting is done successfully.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid, which means given value of address is invalid. The value of address must be greater than 0 and a multiple of 8.

The motion vector information reporting array consists of below fields for each macroblock. The array element has 32 bit data per a macroblock.

Table 3.1. Description of SET_ADDR_REP_MBMV_DATA

Bit range	Field name	Description
31	Inter MB	A flag to indicate whether the macroblock is inter macroblock or not
29:16	Mv X	Signed motion vector value for X axis
13:00	Mv Y	Signed motion vector value for Y axis

`SET_CACHE_CONFIG`

This command sets the configuration of cache. The `parameter` is interpreted as a pointer to MaverickCacheConfig.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means given value is valid and setting is done successfully.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid. The value of address must be not zero.

`GET_TILEDMAP_CONFIG`

This command gets tiled map configuration according to `TiledMapConfig` structure.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means given value is valid and setting is done successfully.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

`SET_LOW_DELAY_CONFIG`

This command sets the low delay decoding options which enable low delay decoding and indicate the number of MB row. The argument `parameter` is interpreted as a pointer to LowDelayInfo which represents an enable flag and the number of MB row. If low delay decoding is enabled, VPU sends an interrupt and indexFrameDisplay to HOST when the number of MB row decoding is done. If the interrupt is issued, HOST should clear the interrupt and read indexFrameDisplay from the RET_DEC_PIC_FRAME_IDX register in order to display.

`DEC_GET_DISPLAY_OUTPUT_INFO`

HOST can get decoder output information according to display index in DecOutputInfo structure. HOST can set display index using member variable `indexFrameDisplay`. This command returns RETCODE_SUCCESS.

- Example code

```
DecOutputInfo decOutputInfo;
decOutputInfo. indexFrameDisplay = disp_index;
VPU_DecGiveCommand(handle, DEC_GET_DISPLAY_OUTPUT_INFO, & decOutputInfo);
```

SET_DECODE_FLUSH

This command is used to change bitstream buffer mode from Interrupt mode to Picture End mode. When HOST receives an interrupt about bistream buffer empty in Interrupt Mode, but there is no more bitstream for the current frame in bitstream buffer, this command completes decoding only with remaining bitstream.

This command returns RETCODE_SUCCESS.

DEC_SET_FRAME_DELAY

HOST can set the number of frame to be delayed before display (H.264/AVC only) by using this command. This command is useful when display frame buffer delay is supposed to happen for buffering decoded picture reorder and HOST is sure of that. Unless this command is executed, VPU has display frame buffer delay as frameBufDelay value of DecInitialInfo structure.

DEC_FREE_FRAME_BUFFER

HOST can free all the frame buffers allocated by VPUAPI. This command is useful when VPU detects sequence change. For example, if HOST knows resolution change while decoding through `sequenceChanged` variable of DecOutputInfo structure, HOST should change the size of frame buffer accordingly. This command is used to release the frame buffers allocated for the previous sequence. Then VPU_DecGetInitialInfo() and VPU_DecIsseuSeqInit() are called before frame buffer allocation for a new sequence.

DEC_SET_DISPLAY_FLAG

Applications can set a display flag for each frame buffer by calling this function after creating decoder instance. If a certain display flag of frame buffer is set, the frame buffer cannot be used in the decoding process. Applications can control displaying a buffer with this command to prevent VPU from using buffer in every decoding process.

This command is the opposite of what VPU_DecClrDispFlag() does. All of buffer flags are initialized with 0x00000000 which means being able to decode. Unless it is called, VPU starts decoding with available frame buffer that has been cleared in round robin order.

DEC_GET_SEQ_INFO

This command returns the information of the current sequence in the form of *the section called “DecInitialInfo”* structure. HOST can get this information with more accuracy after VPU_DecIssueSeqInit() or VPU_DecGetOutputInfo() is called.

ENABLE_LOGGING

HOST can activate message logging once VPU_DecOpen() or VPU_EncOpen() is called.

DISABLE_LOGGING

HOST can deactivate message logging which is off as default.

DEC_ENABLE_AVC_MC_INTERPOL

This is option for enable fast MC interpolation. this command is only used for H.264/AVC decoder.

DEC_DISABLE_AVC_MC_INTERPOL

This is option for disable fast MC interpolation. this command is only used for H.264/AVC decoder.

DEC_SET_2ND_FIELD_INFO

This command sets bistream buffer information for a second field after a first field has been decoded. In case of H.264/AVC, MPEG2, or VC1 decoder, they issue an INT_BIT_DEC_FIELD interrupt after first field decoding. This command then can give VPU the address and size of bitstream buffer for a second field. This commands is used when first field and second field are in separate bistreams - not together in a bitstream. If HOST gives this command and clears the INT_BIT_DEC_FIELD interrupt, VPU starts decoding from the given base address until the size of bitstream buffer.

DEC_GET_FIELD_PIC_TYPE

This command gets a field picture type of decoded picture after INT_BIT_DEC_FIELD interrupt is issued.

DEC_ENABLE_REORDER

HOST can enable display buffer reordering when decoding H.264 streams. In H.264 case output decoded picture may be re-ordered if pic_order_cnt_type is 0 or 1. In that case, decoder must delay output display for re-ordering but some applications (ex. video telephony) do not want such display delay.

DEC_DISABLE_REORDER

HOST can disable output display buffer reorder-ing. Then BIT processor does not re-order output buffer when pic_order_cnt_type is 0 or 1. If In H.264/AVC case. pic_order_cnt_type is 2 or the other standard case, this flag is ignored because output display buffer reordering is not allowed.

DEC_GET_FRAMEBUF_INFO

This command gives HOST the information of framebuffer in the form of DecGetFramebufInfo structure.

DEC_RESET_FRAMEBUF_INFO

This command resets the information of framebuffer. Unlike DEC_FREE_FRAME_BUFFER, it does not release the assigned memory itself. This command is used for sequence change along with DEC_GET_FRAMEBUF_INFO.

ENABLE_REP_CUDATA

This command enables to report CU data.

DISABLE_REP_CUDATA

This command disables to report CU data.

SET_ADDR_REP_CUDATA

This command sets the address of buffer where CU data is written.

SET_SIZE_REP_CUDATA

This command sets the size of buffer where CU data is written.

DEC_SET_WTL_FRAME_FORMAT

This command sets FrameBufferFormat for WTL.

DEC_SET_AVC_ERROR_CONCEAL_MODE

This command sets error conceal mode for H.264 decoder. This command must be issued through VPU_DecGiveCommand() before calling VPU_DecGetInitialInfo() or VPU_DecIssueSeqInit(). In other words, error conceal mode cannot be applied once a sequence initialized.

- AVC_ERROR_CONCEAL_MODE_DEFAULT - VPU performs error concealment in default mode.
- AVC_ERROR_CONCEAL_MODE_ENABLE_SELECTIVE_CONCEAL_MISSING_REFERENCE - VPU performs error concealment using another framebuffer if the error comes from missing reference frame.
- AVC_ERROR_CONCEAL_MODE_DISABLE_CONCEAL_MISSING_REFERENCE - VPU does not perform error concealment if the error comes from missing reference frame.
- AVC_ERROR_CONCEAL_MODE_DISABLE_CONCEAL_WRONG_FRAME_NUM - VPU does not perform error concealment if the error comes from wrong frame_num syntax.

DEC_GET_SCALER_INFO

This command returns setting information to downscale an image such as enable, width, and height.

DEC_SET_SCALER_INFO

This command sets information to downscale an image such as enable, width, and height.

DEC_SET_TARGET_TEMPORAL_ID

This command decodes only a frame whose temporal id is equal to or less than the given target temporal id.

Note | It is only for H.265/HEVC decoder.

DEC_SET_BWB_CUR_FRAME_IDX

This command specifies the index of linear frame buffer which needs to be changed to due to change of inter-frame resolution while decoding.

Note | It is only for VP9 decoder.

DEC_SET_FBC_CUR_FRAME_IDX

This command specifies the index of FBC frame buffer which needs to be changed to due to change of inter-frame resolution while decoding.

Note | It is only for VP9 decoder.

DEC_SET_INTER_RES_INFO_ON

This command informs inter-frame resolution has been changed while decoding. After this command issued, VPU reallocates one frame buffer for the change.

Note | It is only for VP9 decoder.

DEC_SET_INTER_RES_INFO_OFF

This command releases the flag informing inter-frame resolution change. It should be issued after reallocation of one frame buffer is completed.

Note | It is only for VP9 decoder.

DEC_FREE_FBC_TABLE_BUFFER

This command frees one FBC table to deal with inter-frame resolution change.

Note | It is only for VP9 decoder.

DEC_FREE_MV_BUFFER

This command frees one MV buffer to deal with inter-frame resolution change.

Note | It is only for VP9 decoder.

DEC_ALLOC_FBC_Y_TABLE_BUFFER

This command allocates one FBC luma table to deal with inter-frame resolution change.

Note | It is only for VP9 decoder.

DEC_ALLOC_FBC_C_TABLE_BUFFER

This command allocates one FBC chroma table to deal with inter-frame resolution change.

Note | It is only for VP9 decoder.

DEC_ALLOC_MV_BUFFER

This command allocates one MV buffer to deal with inter-frame resolution change.

Note | It is only for VP9 decoder.

VPU_DecConfigSecondAxi()

Prototype

```
RetCode VPU_DecConfigSecondAxi (
                                  DecHandle handle,
                                  int stride,
                                  int height
                                );
```

Description

This function sets the secondary channel of AXI for saving memory bandwidth to dedicated memory. It works same as SET_SEC_AXI command which is only able to be called before VPU_DecRegisterFramebuffer(). This function can be called before VPU_DecStartOneFrame() to change information about the secondary channel of AXI.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
stride	Input	Width of framebuffer
height	Input	Height of framebuffer

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means given value for setting secondary AXI is valid.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding operation was not completed yet, so the given API function call cannot be allowed.

RETCODE_INSUFFICIENT_RESOURCE
This means that VPU cannot allocate memory due to lack of memory.

RETCODE_VPU_RESPONSE_TIMEOUT
This means that VPU response time is too long, time out.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not a handle which has been obtained by VPU_DecOpen().
- `handle` is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding operation was not completed yet, so the given API function call cannot be performed this time. A frame decoding operation should be completed by calling VPU_ DecSetRdPtr ().

VPU_DecStartOneFrame()

Prototype

```
RetCode VPU_DecStartOneFrame (
                                  DecHandle handle,
                                  DecParam *param
                                );
```

Description

This function starts decoding one frame. Returning from this function does not mean the completion of decoding one frame, and it is just that decoding one frame was initiated. Every call of this function should be matched with VPU_DecGetOutputInfo() with the same handle. Without calling a pair of these funtions, HOST cannot call any other API functions except for VPU_IsBusy(), VPU_DecGetBitstreamBuffer(), and VPU_DecUpdateBitstreamBuffer().

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
param	Input	A pointer to a DecParam type structure which describes picture decoding parameters for the given decoder instance

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means decoding a new frame was started successfully.

Note This return value does not mean that decoding a frame was completed successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not a handle which has been obtained by VPU_DecOpen().
- `handle` is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding operation was not completed yet, so the given API function call cannot be performed this time. A frame decoding operation should be completed by calling VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST should call VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. A HOST might call this function before successfully calling VPU_DecRegisterFrameBuffer(). This function should be called after calling VPU_ DecRegisterFrameBuffer() successfully.

VPU_DecGetOutputInfo()

Prototype

```
RetCode VPU_DecGetOutputInfo (
                                  DecHandle handle,
                                  DecOutputInfo *info
                                );
```

Description

VPU returns the result of decoding which includes information on decoded picture, syntax value, frame buffer, other report values, and etc. HOST should call this function after frame decoding is finished and before starting the further processing.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
info	Output	A pointer to a DecOutputInfo type structure which describes picture decoding results for the current decoder instance.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means receiving the output information of the current frame was done successfully.

RETCODE_FAILURE
Operation was failed, which means there was an error in getting information for configuring the decoder.

RETCODE_INVALID_HANDLE
This means argument handle is invalid. This includes cases where `handle` is not a handle which has been obtained by VPU_DecOpen(), `handle` is a handle to an instance already closed, or `handle` is a handle to a decoder instance. Also, this value is returned when VPU_DecStartOneFrame() is matched with VPU_DecGetOutputInfo() with different handles.

RETCODE_INVALID_PARAM
The given argument parameter, pInfo, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

VPU_DecFrameBufferFlush()

Prototype

```
RetCode VPU_DecFrameBufferFlush (
                                    DecHandle handle,
                                    DecOutputInfo *pRemainingInfo,
                                    Uint32 *pRetNum
                                );
```

Description

This function flushes all of the decoded framebuffer contexts that remain in decoder firmware. It is used to start seamless decoding operation without randome access to the buffer or calling VPU_DecClose().

Note In WAVE4, this function returns all of the decoded framebuffer contexts that remain. pRetNum always has 0 in CODA9.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
pRemainingInfo	Output	All of the decoded framebuffer contexts are stored in display order as array of DecOutputInfo. If this is NULL, the remaining information is not returned.
pRetNum	Output	The number of the decoded frame buffer contexts. It this is null, the information is not returned.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means receiving the output information of the current frame was done successfully.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding operation was not completed yet, so the given API function call cannot be performed this time. A frame decoding operation should be completed by calling VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST should call VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_INVALID_HANDLE
This means argument handle is invalid. This includes cases where `handle` is not a handle which has been obtained by VPU_DecOpen(), `handle` is a handle to an instance already closed, or `handle` is a handle to an decoder instance. Also,this value is returned when VPU_DecStartOneFrame() is matched with VPU_DecGetOutputInfo() with different handles.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not recieved any response from VPU and has timed out.

VPU_DecClrDispFlag()

Prototype

```
RetCode VPU_DecClrDispFlag (
                              DecHandle handle,
                              int index
                            );
```

Description

This function clears a display flag of frame buffer. If display flag of frame buffer is cleared, the frame buffer can be reused in the decoding process. With VPU_DecClrDispFlag(), HOST application can control display of frame buffer marked with the display index which is given by VPU.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
index	Input	A frame buffer index to be cleared

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means receiving the output information of the current frame was done successfully.

RETCODE_INVALID_HANDLE
This means argument handle is invalid. This includes cases where `handle` is not a handle which has been obtained by VPU_DecOpen(), `handle` is a handle to an instance already closed, or `handle` is a handle to an decoder instance. Also,this value is returned when VPU_DecStartOneFrame() is matched with VPU_DecGetOutputInfo() with different handles.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. It might happen because VPU_DecRegisterFrameBuffer() with the same handle was not called before calling this function.

RETCODE_INVALID_PARAM
The given argument parameter, index, was invalid, which means it has improper values.

VPU_EncOpen()

Prototype

```
RetCode VPU_EncOpen (
                      EncHandle *handle,
                      EncOpenParam *encOpParam
                    );
```

Description

In order to start a new encoder operation, HOST application must open a new instance for this encoder operation. By calling this function, HOST application can get a handle specifying a new encoder instance. Because VPU supports multiple instances of codec operations, HOST application needs this kind of handles for the all codec instances now on running. Once a HOST application gets a handle, the HOST application must use this handle to represent the target instances for all subsequent encoder-related functions.

Parameter

Parameter	Type	Description
handle	Output	A pointer to a EncHandle type variable which specifies each instance for HOST application. If no instance is available, null handle is returned.
encOpParam	Input	A pointer to a EncOpenParam type structure which describes required parameters for creating a new encoder instance.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means a new encoder instance was opened successfully.

RETCODE_FAILURE
Operation was failed, which means getting a new encoder instance was not done successfully. If there is no free instance anymore, this value is returned in this function call.

RETCODE_INVALID_PARAM
The given argument parameter, pOpenParam, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

RETCODE_NOT_INITIALIZED
VPU was not initialized at all before calling this function. Application should initialize VPU by calling VPU_Init() before calling this function.

VPU_EncClose()

Prototype

```
RetCode VPU_EncClose (
                        EncHandle handle
                      );
```

Description

When HOST application finished encoding operations and wanted to release this instance for other processing, the HOST application should close this instance by calling this function. After completion of this function call, the instance referred to by pHandle gets free. Once HOST application closes an instance, the HOST application cannot call any further encoder-specific function with the current handle before re-opening a new instance with the same handle.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()

Return Value

RETCODE_SUCCESS
Operation was done successfully. That means the current encoder instance was closed successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- pHandle is not a handle which has been obtained by VPU_EncOpen().
- pHandle is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding or encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding or decoding operation should be completed by calling VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not recieved any response from VPU and has timed out.

VPU_EncGetInitialInfo()

Prototype

```
RetCode VPU_EncGetInitialInfo (
                                  EncHandle handle,
                                  EncInitialInfo *encInitInfo
                                );
```

Description

Before starting encoder operation, HOST application must allocate frame buffers according to the information obtained from this function. This function returns the required parameters for VPU_EncRegisterFrameBuffer(), which follows right after this function call.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
encInitInfo	Output	Minimum number of frame buffer for this encoder instance

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means receiving the initial parameters was done successfully.

RETCODE_FAILURE
Operation was failed, which means there was an error in getting information for configuring the encoder.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- pHandle is not a handle which has been obtained by VPU_EncOpen().
- pHandle is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding or encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding or decoding operation should be completed by calling VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_INVALID_PARAM
The given argument parameter, pInitialInfo, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

RETCODE_CALLED_BEFORE
This function call is invalid which means multiple calls of the current API function for a given instance are not allowed. In this case, encoder initial information has been received already, so that this function call is meaningless and not allowed anymore.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not recieved any response from VPU and has timed out.

VPU_EncRegisterFrameBuffer()

Prototype

```
RetCode VPU_EncRegisterFrameBuffer (
                                      EncHandle handle,
                                      FrameBuffer *bufArray,
                                      int num,
                                      int stride,
                                      int height,
                                      int mapType
                                    );
```

Description

This function registers frame buffers requested by VPU_EncGetInitialInfo(). The frame buffers pointed to by pBuffer are managed internally within VPU. These include reference frames, reconstructed frames, etc. Applications must not change the contents of the array of frame buffers during the life time of the instance, and num must not be less than minFrameBufferCount obtained by VPU_EncGetInitialInfo().

The distance between a pixel in a row and the corresponding pixel in the next row is called a stride. The value of stride must be a multiple of 8. The address of the first pixel in the second row does not necessarily coincide with the value next to the last pixel in the first row. In other words, a stride can be greater than the picture width in pixels.

Applications should not set a stride value smaller than the picture width. So, for Y component, HOST application must allocate at least a space of size (frame height * stride), and Cb or Cr component, (frame height/2 * stride/2), respectively. But make sure that in Cb/Cr non-interleave (separate Cb/Cr) map, a stride for the luminance frame buffer should be multiple of 16 so that a stride for the luminance frame buffer becomes a multiple of 8.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
bufArray	Input	Allocated frame buffer address and information. If this parameter is set to -1, VPU allocates frame buffers.
num	Input	A number of frame buffers. VPU can allocate frame buffers as many as this given value.
stride	Input	A stride value of the given frame buffers
height	Input	Frame height
mapType	Input	Map type of frame buffer

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means registering frame buffers were done successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- handle is not a handle which has been obtained by VPU_EncOpen().

- handle is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding or encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding or decoding operation should be completed by calling VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. HOST application might call this function before calling VPU_EncGetInitialInfo() successfully. This function should be called after successful calling of VPU_EncGetInitialInfo().

RETCODE_INVALID_FRAME_BUFFER
This means argument pBuffer were invalid, which means it was not initialized yet or not valid anymore.

RETCODE_INSUFFICIENT_FRAME_BUFFERS
This means the given number of frame buffers, num, was not enough for the encoder operations of the given handle. It should be greater than or equal to the value of minFrameBufferCount obtained from VPU_EncGetInitialInfo().

RETCODE_INVALID_STRIDE
This means the given argument stride was invalid, which means it is 0, or is not a multiple of 8 in this case.

RETCODE_CALLED_BEFORE
This function call is invalid which means multiple calls of the current API function for a given instance are not allowed. It might happen when registering frame buffer for this instance has been done already so that this function call is meaningless and not allowed anymore.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not recieved any response from VPU and has timed out.

VPU_EncAllocateFrameBuffer()

Prototype

```
RetCode VPU_EncAllocateFrameBuffer (
                                      EncHandle handle,
                                      FrameBufferAllocInfo info,
                                      FrameBuffer *frameBuffer
                                    );
```

Description

This is a special function that enables HOST application to allocate directly the frame buffer for encoding or for Pre-processor (PRP) such as Rotator. In normal operation, VPU API allocates frame buffers when the argument bufArray in VPU_EncRegisterFrameBuffer() is set to 0. However, for any other reason HOST application can use this function to allocate frame buffers by themselves.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
info	Input	Information required for frame bufer allocation
frameBuffer	Output	Data structure that holds information of allocated frame buffers

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means the framebuffer is allocated successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- handle is not a handle which has been obtained by VPU_EncOpen().
- handle is a handle of an instance which has been closed already, etc.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. It might happen because VPU_EncRegisterFrameBuffer() for (FramebufferAllocType.FB_TYPE_CODEC) has not been called, before this function call for allocating frame buffer for PRP (FramebufferAllocType.FB_TYPE_PPU).

RETCODE_INSUFFICIENT_RESOURCE
Fail to allocate a framebuffer due to lack of memory

RETCODE_INVALID_PARAM
The given argument parameter, index, was invalid, which means it has improper values

VPU_EncGetFrameBuffer()

Prototype

```
RetCode VPU_EncGetFrameBuffer (
                                  EncHandle handle,
                                  int frameIdx,
                                  FrameBuffer *frameBuf
                                );
```

Description

This function gets the frame buffer information that was allocated by VPU_EncRegisterFrameBuffer() function.

It does not affect actual encoding and simply does obtain the information of frame buffer. This function is more helpful especially when frame buffers are automatically assigned by setting 0 to `bufArray` of VPU_EncRegisterFrameBuffer() and HOST wants to know about the allocated frame buffer.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
frameIdx	Input	An index of frame buffer
frameBuf	output	Allocated frame buffer address and information

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means registering frame buffer information was done successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not a handle which has been obtained by VPU_DecOpen().
- `handle` is a handle of an instance which has been closed already, etc.

RETCODE_INVALID_PARAM
The given argument parameter, frameIdx, was invalid, which means frameIdx is larger than allocated framebuffer.

VPU_EncGiveCommand()

Prototype

```
RetCode VPU_EncGiveCommand (
                              EncHandle handle,
                              CodecCommand cmd,
                              void *parameter
                            );
```

Description

This function is provided to let HOST application have a certain level of freedom for re-configuring encoder operation after creating an encoder instance. Some options which can be changed dynamically during encoding as the video sequence has been included. Some command-specific return codes are also presented.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
cmd	Input	A variable specifying the given command of CodecComand type
parameter	Input/Output	A pointer to command-specific data structure which describes picture I/O parameters for the current encoder instance

Return Value

RETCODE_INVALID_COMMAND
This means the given argument, cmd, was invalid which means the given cmd was undefined, or not allowed in the current instance.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- pHandle is not a handle which has been obtained by VPU_EncOpen().
- pHandle is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding or encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding or decoding operation should be completed by calling VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo() to proceed this function call.

Command

The list of commands can be summarized as follows:

- ENABLE_ROTATION
- DIABLE_ROTATION

- ENABLE_MIRRORING
- DISABLE_MIRRORING
- SET_MIRROR_DIRECTION
- SET_ROTATION_ANGLE
- ENC_ADD_PPS
- ENC_SET_ACTIVE_PPS
- ENC_GET_ACTIVE_PPS
- ENC_PUT_VIDEO_HEADER
- ENC_PUT_MP4_HEADER
- ENC_PUT_AVC_HEADER
- ENC_GET_VIDEO_HEADER
- ENC_SET_INTRA_MB_REFRESH_NUMBER
- ENC_ENABLE_HEC
- ENC_DISABLE_HEC
- ENC_SET_SLICE_INFO
- ENC_SET_GOP_NUMBER
- ENC_SET_INTRA_QP
- ENC_SET_BITRATE
- ENC_SET_FRAMERATE
- ENC_SET_SEARCHRAM_PARAM
- ENC_SET_REPORT_MBINFO
- ENC_SET_REPORT_MVINFO
- ENC_SET_REPORT_SLICEINFO
- ENC_SET_PIC_PARA_ADDR
- SET_SEC_AXI
- ENABLE_LOGGING
- DISABLE_LOGGING
- ENC_CONFIG_SUB_FRAME_SYNC
- ENC_SET_SUB_FRAME_SYNC

ENABLE_ROTATION

This command enables rotation of the pre-rotator. In this case, `parameter` is ignored. This command returns RETCODE_SUCCESS.

DIABLE_ROTATION

This command disables rotation of the pre-rotator. In this case, `parameter` is ignored. This command returns RETCODE_SUCCESS.

ENABLE_MIRRORING

This command enables mirroring of the pre-rotator. In this case, `parameter` is ignored. This command returns RETCODE_SUCCESS.

DISABLE_MIRRORING

This command disables mirroring of the pre-rotator. In this case, `parameter` is ignored. This command returns RETCODE_SUCCESS.

SET_MIRROR_DIRECTION

This command sets mirror direction of the pre-rotator, and `parameter` is interpreted as a pointer to MirrorDirection. The `parameter` should be one of MIRDIR_NONE, MIRDIR_VER, MIRDIR_HOR, and MIRDIR_HOR_VER.

- MIRDIR_NONE: No mirroring

- MIRDIR_VER: Vertical mirroring
- MIRDIR_HOR: Horizontal mirroring
- MIRDIR_HOR_VER: Both directions

This command has one of the following return codes

- RETCODE_SUCCESS: Operation was done successfully, which means given mirroring direction is valid.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid, which means given mirroring direction is invalid.

SET_ROTATION_ANGLE

This command sets counter-clockwise angle for post-rotation, and `parameter` is interpreted as a pointer to the integer which represents rotation angle in degrees. Rotation angle should be one of 0, 90, 180, and 270.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means given rotation angle is valid.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid, which means given rotation angle is invalid.

Note Rotation angle could not be changed after sequence initialization, because it might cause problems in handling frame buffers.

ENC_ADD_PPS

This command adds PPS header syntax of AVC/H.264 bitstream

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument cmd was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an H.264/AVC encoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` was invalid, which means it has a null pointer, or given values for some member variables are improper values.

ENC_SET_ACTIVE_PPS

This command sets active PPS header syntax of AVC/H.264 bitstream, and `parameter` is interpreted] as an active PPS index.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument cmd was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an MPEG4 encoder instance.

- RETCODE_INVALID_PARAM: The given argument parameter `parameter` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_GET_ACTIVE_PPS

This command gets active PPS header syntax of AVC/H.264 bitstream, and `parameter` is interpreted as an active PPS index.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument cmd was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an MPEG4 encoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` was invalid, which means it has a null pointer, or given values for some member variables are improper values.

ENC_PUT_VIDEO_HEADER

This command inserts an MPEG4 header syntax or SPS or PPS to the AVC/H.264 bitstream to the bitstream during encoding. The argument `parameter` is interpreted as a pointer to EncHeaderParam structure holding

- buf is a physical address pointing the generated stream location
- size is the size of generated stream in bytes
- headerType is a type of header that HOST application wants to generate and have values as VOL_HEADER, VOS_HEADER,, VO_HEADER, SPS_RBSP or PPS_RBSP.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument cmd was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an MPEG4 encoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or headerType was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_PUT_MP4_HEADER

This command inserts an MPEG4 header syntax to the bitstream during encoding. The argument `parameter` is interpreted as a pointer to EncHeaderParam structure holding

- `buf` is a physical address pointing the generated stream location

- `size` is the size of generated stream in bytes
- `headerType` is a type of header that HOST application wants to generate and have values as VOL_HEADER, VOS_HEADER, or VO_HEADER.

  This command has one of the following return codes.

  - RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
  - RETCODE_INVALID_COMMAND: This means the given argument `cmd` was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an MPEG4 encoder instance.
  - RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
  - RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_PUT_AVC_HEADER

This command inserts an SPS or PPS to the AVC/H.264 bitstream during encoding.

The argument `parameter` is interpreted as a pointer to EncHeaderParam structure holding

- buf is a physical address pointing the generated stream location
- size is the size of generated stream in bytes
- headerType is type of header that HOST application wants to generate and have values as SPS_RBSP or PPS_RBSP.

  This command has one of the following return codes.

  - RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
  - RETCODE_INVALID_COMMAND: This means the given argument `cmd` was invalid which means the given `cmd` was undefined, or not allowed in the current instance. In this case, the current instance might not be an AVC/H.264 encoder instance.
  - RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
  - RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_GET_VIDEO_HEADER

This command gets a SPS to the AVC/H.264 bitstream to the bitstream during encoding. Because VPU does not generate bitstream but HOST application generate bitstream in this command, HOST application has to set pBuf value to access bitstream buffer. The argument `parameter` is interpreted as a pointer to EncHeaderParam structure holding

- buf is a physical address pointing the generated stream location
- pBuf is a virtual address pointing the generated stream location

- size is the size of generated stream in bytes
- headerType is a type of header that HOST application wants to generate and have values as SPS_RBSP.

  This command has one of the following return codes.

  - RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
  - RETCODE_INVALID_COMMAND: This means the given argument cmd was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an AVC/H.264 encoder instance.
  - RETCODE_INVALID_PARAM: The given argument parameter `parameter` or headerType was invalid, which means it has a null pointer, or given values for some member variables are improper values.
  - RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_SET_INTRA_MB_REFRESH_NUMBER

This command sets intra MB refresh number of header syntax. The argument `parameter` is interpreted as a pointer to integer which represents an intra refresh number. It should be between 0 and macroblock number of encoded picture.

This command returns the following code.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_ENABLE_HEC

This command enables HEC(Header Extension Code) syntax of MPEG4.

This command ignores the argument `parameter` and returns one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument, cmd, was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an MPEG4 encoder instance.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_DISABLE_HEC

This command disables HEC(Header Extension Code) syntax of MPEG4.

This command ignores the argument `parameter` and returns one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.

- RETCODE_INVALID_COMMAND: This means the given argument, cmd, was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an MPEG4 encoder instance.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_SET_SLICE_INFO

This command sets slice inforamtion of header syntax. The argument `parameter` is interpreted as a pointer to EncSliceMode structure holding

- sliceModeuf is a mode which means enabling multi slice structure
- sliceSizeMode is the mode representing mode of calculating one slicesize
- sliceSize is the size of one slice.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_SET_GOP_NUMBER

This command sets GOP number of header syntax. The argument `parameter` is interpreted as a pointer to the integer which represents a GOP number.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_SET_INTRA_QP

This command sets intra QP value of header syntax. The argument `parameter` is interpreted as a pointer to the integer which represents a Constant I frame QP. The Constant I frame QP should be between 1 and 31 for MPEG4 and between 0 and 51 for AVC/H.264.

This command has one of the following return codes

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument `cmd` was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an encoder instance.

- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_SET_BITRATE

This command sets bitrate inforamtion of header syntax. The argument `parameter` is interpreted as a pointer to the integer which represents a bitrate. It should be between 0 and 32767.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument `cmd` was invalid which means the given `cmd` was undefined, or not allowed in the current instance. In this case, the current instance might not be an encoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_SET_FRAMERATE

This command sets frame rate of header syntax. The argument `parameter` is interpreted as a pointer to the integer which represents a frame rate value. The fraem rate should be greater than 0.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument `cmd` was invalid which means the given `cmd` was undefined, or not allowed in the current instance. In this case, the current instance might not be an encoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_SET_SEARCHRAM_PARAM

This command sets the parameter of search sram used in encoder. The `parameter` is interpreted as a pointer to SearchRamParam.

This command has one of the following return codes.

- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `SearchRamSize` was invalid, which means it has a null pointer, or given values for some member variables are improper values.

ENC_SET_REPORT_MBINFO

This command sets MB data report structure. The argument `parameter` is interpreted as a pointer to address and returns RETCODE_SUCCESS.

If this option is set, the slice boundary and QP are reported using 1 bytes.

- [7] : Reserved
- [6] : Slice Boundary. Whenever a new slice header is decoded, this bit field is toggled.
- [5:0] : An macroblock QP value

ENC_SET_REPORT_MVINFO

This command sets motion vector report structure. The argument `parameter` is interpreted as a pointer to address and returns RETCODE_SUCCESS.

If this option is set, the motion vector information are reported using 4 bytes.

- [31] : Intra Flag (1: intra, 0: inter)
- [30:16] : X value of motion vector
- [16:0] : Y value of motion vector

ENC_SET_REPORT_SLICEINFO

This command sets slice information report structure. The argument `parameter` is interpreted as a pointer to address and returns RETCODE_SUCCESS.

If this option is set, the slice information are reported using 8 bytes.

- [63:48] : Reserved.
- [47:32] :The last macroblock indeed of a slice (zero based-index)
- [31:0] : Total of bits used for encoding a slice.

ENC_SET_PIC_PARA_ADDR

This command sets the address of picture parameter base. The argument `parameter` is interpreted as a pointer to `address` and returns RETCODE_SUCCESS.

To report MB data, Motion Vector, and Slice information, VPU should reads the base address of external memory, which all are specified by picParaBaseAddr.

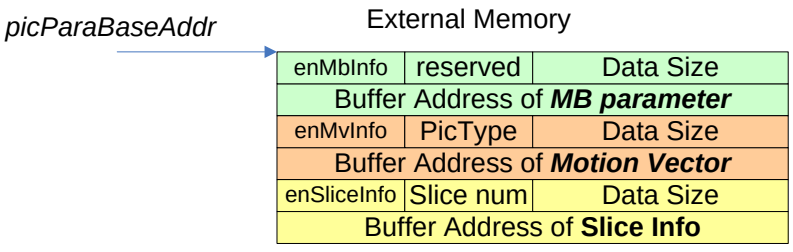


Figure 3.2. Encoder Picture parameter base address structure

When `enReportMBInfo`, `enReportMVInfo`, and `enReportSliceinfo` in CMD_ENC_PIC_OPTION register are enabled, HOST application should specify the buffer

addresses in *Figure 3.2, "Encoder Picture parameter base address structure"*. VPU reports each data and fills info, Data Size and other fields to these buffer addresses of external memory. For VPU to report data properly, HOST application needs to specify these 3 buffer addresses preserving 8 byte alignment and buffer sizes need to be multiples of 256.

SET_SEC_AXI

This command sets the secondary channel of AXI for saving memory bandwidth to dedicated memory. The argument `parameter` is interpreted as a pointer to SecAxiUse which represents an enable flag and physical address which is related with the secondary channel for BIT processor, IP/AC-DC predictor, de-blocking filter, overlap filter respectively.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means given value for setting secondary AXI is valid.
- RETCODE_INVALID_PARAM: The given argument parameter, `parameter`, was invalid, which means given value is invalid.

ENABLE_LOGGING

HOST can activate message logging once VPU_DecOpen() or VPU_EncOpen() is called.

DISABLE_LOGGING

HOST can deactivate message logging which is off as default.

ENC_CONFIG_SUB_FRAME_SYNC

This command sets the configuration of sub frame sync mode. The `parameter` is interpreted as a pointer to EncSubFrameSyncConfig. This command returns RETCODE_SUCCESS.

ENC_SET_SUB_FRAME_SYNC

This command sets the status of sub frame sync signal. The `parameter` is interpreted as a pointer to EncSubFrameSyncState. This command returns RETCODE_SUCCESS.

ENC_SET_PARA_CHANGE

This command changes encoding parameter(s) during the encoding operation in H.265/AVC encoder The argument `parameter` is interpreted as a pointer to *the section called "EncChangeParam"* structure holding

- changeParaMode : OPT_COMMON (only valid currently)
- enable_option : Set an enum value that is associated with parameters to change, *the section called "ChangeCommonParam"* (multiple option allowed).

For instance, if bitrate and framerate need to be changed in the middle of encoding, that requires some setting like below.

```
EncChangeParam changeParam;
changeParam.changeParaMode    = OPT_COMMON;
changeParam.enable_option     = ENC_RC_TARGET_RATE_CHANGE | ENC_FRAME_RATE_CHANGE;
```

```
changeParam.bitrate           = 14213000;
changeParam.frameRate         = 15;
VPU_EncGiveCommand(handle, ENC_SET_PARA_CHANGE, &changeParam);
```

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument `cmd` was invalid which means the given `cmd` was undefined, or not allowed in the current instance. In this case, the current instance might not be an AVC/H.264 encoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_SET_PROFILE

This command encodes videos into H.264/AVC stream with HOST application-given profile in header syntax. (CODA9 only)

ENC_SET_SEI_NAL_DATA

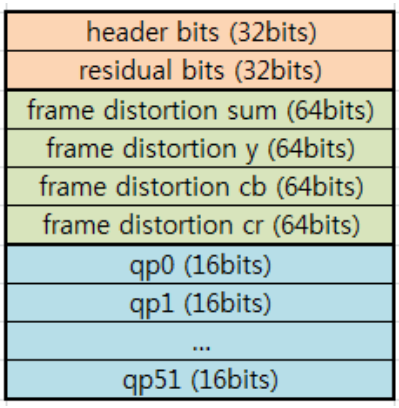
This command sets variables in HevcSEIDataEnc structure for SEI encoding.

SET_ENC_WTL_INFO

This command sets information of frame buffer such as Y/Cb/Cr addr, cbcrinterleave, nv21, and number of linear framebuffer for WTL-enabled encoder.

SET_ADDR_ENC_RC_REPORT_BASE

This command sets the base address of report buffer. It should be specified when any of these bitInfoReportEnable, frameDistortionReportEnable, or qpHistogramReportEnable in EncParam is on. In total 144bytes of data are reported to the report buffer from w4EncReportBaseAddr in the following order. If a certain ReportEnable flag is diabled i.e frameDistortionReportEnable, dummy data are written into the frameDistortion region of the buffer.



- bit information
  - Header bits : total slice header bits of the current frame except start code
  - Residual bits : total residual bits of the current frame

header bits are in the beginning 32bits of w4EncReportBaseAddr, and header bits are in the following 32bits of the report buffer.

- frame distortion

With this command, frame distortion data - frame distortion sum, y, cb, and cr for each 8bytes, totally 32bytes) are reported 8bytes away from w4EncReportBaseAddr (w4EncReportBaseAddr + 8) when picture encoding is completed.

- QP histogram

With this command, QP histogram for every subCTU(CU32) taking 16bits are reported 40bytes away from w4EncReportBaseAddr (w4EncReportBaseAddr + 40) when picture encoding is completed.

SET_SIZE_ENC_RC_REPORT

This command sets the size of report buffer. It should be specified when any of these bitInfoReportEnable, frameDistortionReportEnable, or qpHistogramReportEnable in EncParam is on.

VPU_EncStartOneFrame()

Prototype

```
RetCode VPU_EncStartOneFrame (
                                  EncHandle handle,
                                  EncParam *param
                                );
```

Description

This function starts encoding one frame. Returning from this function does not mean the completion of encoding one frame, and it is just that encoding one frame was initiated.

Every call of this function should be matched with VPU_EncGetOutputInfo() with the same handle. Without calling a pair of these funtions, HOST application cannot call any other API functions except for VPU_IsBusy(), VPU_EncGetBitstreamBuffer(), and VPU_EncUpdateBitstreamBuffer().

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
param	Input	A pointer to EncParam type structure which describes picture encoding parameters for the current encoder instance.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means encoding a new frame was started successfully.

Note This return value does not mean that encoding a frame was completed successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- handle is not a handle which has been obtained by VPU_EncOpen().
- handle is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding or encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding or decoding operation should be completed by calling VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. In this case, HOST application might call this function before successfully calling VPU_EncRegisterFrameBuffer(). This function should be called after successfully calling VPU_EncRegisterFrameBuffer().

RETCODE_INVALID_PARAM
The given argument parameter, `parameter`, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

RETCODE_INVALID_FRAME_BUFFER
This means sourceFrame in input structure EncParam was invalid, which means sourceFrame was not valid even though picture-skip is disabled.

VPU_EncGetOutputInfo()

Prototype

```
RetCode VPU_EncGetOutputInfo (
                                EncHandle handle,
                                EncOutputInfo *info
                              );
```

Description

This function gets information of the output of encoding. Application can know about picture type, the address and size of the generated bitstream, the number of generated slices, the end addresses of the slices, and macroblock bit position information. HOST application should call this function after frame encoding is finished, and before starting the further processing.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen().
info	Output	A pointer to a EncOutputInfo type structure which describes picture encoding results for the current encoder instance.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means the output information of the current frame encoding was received successfully.

RETCODE_INVALID_HANDLE
The given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- handle is not a handle which has been obtained by VPU_EncOpen(), for example a decoder handle,
- handle is a handle of an instance which has been closed already,
- handle is not the same handle as the last VPU_EncStartOneFrame() has, etc.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. HOST application might call this function before calling VPU_EncStartOneFrame() successfully. This function should be called after successful calling of VPU_EncStartOneFrame().

RETCODE_INVALID_PARAM
The given argument parameter, pInfo, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

VPU_EncGetBitstreamBuffer()

Prototype

```
RetCode VPU_EncGetBitstreamBuffer (
                                     EncHandle handle,
                                     PhysicalAddress *prdPrt,
                                     PhysicalAddress *pwrPtr,
                                     int *size
                                   );
```

Description

After encoding frame, HOST application must get bitstream from the encoder. To do that, they must know where to get bitstream and the maximum size. Applications can get the information by calling this function.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
prdPrt	Output	A stream buffer read pointer for the current encoder instance
pwrPtr	Output	A stream buffer write pointer for the current encoder instance
size	Output	A variable specifying the available space in bitstream buffer for the current encoder instance

Return Value

RETCODE_SUCCESS
Operation was done successfully. That means the current encoder instance was closed successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- pHandle is not a handle which has been obtained by VPU_EncOpen().
- pHandle is a handle of an instance which has been closed already, etc.

RETCODE_INVALID_PARAM
The given argument parameter, pRdptr, pWrptr or size, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

VPU_EncUpdateBitstreamBuffer()

Prototype

```
RetCode VPU_EncUpdateBitstreamBuffer (
                                        EncHandle handle,
                                        int size
                                      );
```

Description

Applications must let encoder know how much bitstream has been transferred from the address obtained from VPU_EncGetBitstreamBuffer(). By just giving the size as an argument, API automatically handles pointer wrap-around and updates the read pointer.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
size	Input	A variable specifying the amount of bits being filled from bitstream buffer for the current encoder instance

Return Value

RETCODE_SUCCESS
Operation was done successfully. That means the current encoder instance was closed successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- pHandle is not a handle which has been obtained by VPU_EncOpen().
- pHandle is a handle of an instance which has been closed already, etc.

RETCODE_INVALID_PARAM
The given argument parameter, size, was invalid, which means size is larger than the value obtained from VPU_EncGetBitstreamBuffer().

VPU_EncSetWrPtr()

Prototype

```
RetCode VPU_EncSetWrPtr (
                          EncHandle handle,
                          PhysicalAddress addr,
                          int updateRdPtr
                        );
```

Description

This function specifies the location of write pointer in bitstream buffer. It can also set a read pointer with the same value of write pointer (addr) when updateRdPtr is not a zero value. This function can be used regardless of bitstream buffer mode.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
addr	Input	Updated write or read pointer
updateRdPtr	Input	A flag whether to move the read pointer to where the write pointer is located

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means required information of the stream data to be encoded was received successfully.

RETCODE_FAILURE
Operation was failed, which means there was an error in getting information for configuring the encoder.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not a handle which has been obtained by VPU_EncOpen().
- `handle` is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding operation should be completed by calling VPU_ EncSetRdPtr ().

About Chips&Media

Chips&Media, Inc. is a leading video IP provider, headquartered in Seoul, South Korea. The company was established in 2003 by leading members with years of profound experiences in video standard technologies and the semiconductor industry. Our extensive catalogue of IP solutions includes video postprocessing, video frame buffer compression as well as video codecs covering vast range of video standards from MPEG-2, MPEG-4, H.263, Sorenson, H.264, VC-1, AVS/AVS+, VP8/9, HEVC(H.265), and AV1 for HD to UHD (4K/8K) resolution.

Chips&Media has been developing a line of reliable, high-quality IP solutions that allow our customers and partners to satisfy the growing consumer demand for high-performance multi-media digital devices. Especially as a leading multi-standard video codec solution provider, we have been providing our advanced ultra-low power multi-codec video IPs to top-tier semiconductor companies.

With a mission to become a global top SoC IP provider, Chips&Media has introduced Image Signal Processing (ISP) IP and Deep learning-based super resolution IP to the market and continues to widen our solution portfolio to cope with growing demand on image processing and related solutions. To find further information, please visit the company's web site at *http://www.chipsnmedia.com*