WAVE510 HEVC Decoder IP

API Reference Manual

Version 2.0.0

WAVE510 HEVC Decoder IP: API Reference Manual

Version 2.0.0

Revision History

Date	Revision	Change
2014/12/31	1.0.0	Initial version
2015/2/3	1.1.0	Updated version
2015/2/3	1.3.0	Integrated version for Ref-sw5.0
2015/3/12	1.4.0	SUPPORT_ROI_50 supported
2015/3/27	1.5.0	addrRoiCtuMap and roiDeltaQp of HevcRoiParam, RETCODE_ACCESS_VIOLATION_HW, and refMissingFrameFlag were added.
2015/4/28	1.6.0	internalBitDepth of EncHevcParam, GOP_PRESET_IDX were defined.
2015/5/11	1.7.0	VPU_EncSetWrPtr() and ENC_SET_PARA_CHANGE of VPU_EncGiveCommand() were defined.
2015/7/31	1.9.0	updated for support of SEI and VUI encoding
2015/11/13	1.10.0	region and roiLevel were removed from HevcRoiParam. ROI structure for H.264 has renamed from EncSetROI to AvcRoiParam.
2015/12/3	1.11.0	HevcRoiParam.roiNum was removed. Description on rcGopIQpOffsetEn and rcGopIQpOffset were fixed.
2015/12/21	1.12.0	DecParam.craAsBlaFlag was defined.
2016/3/9	1.13.0	Noise reduction parameters were included in EncHevcParam and EncChangeParam.
2016/3/10	1.14.0	DecOutputInfo.indexInterFrameDecoded was added.
2016/3/31	1.15.0	New encoder features like rc report, long-term reference, etc. added
2016/4/14	1.16.0	Parameters for sei/hrd/vui encoding were added
2016/5/12	1.17.0	Missing descriptions on struct members, VPU_EncGiveCommand()/VPU_DecGiveCommand() were added
2016/5/18	1.18.0	More descriptions on EncOpenParam.mvReportEnable, addrCtuModeMap, etc. were added.
2016/5/30	1.19.0	Wrong references to the appendix in Programmer's guide were fixed.
2016/7/6	1.20.0	CustomHeaderParam structure was included.
2016/8/26	1.21.0	Updates for dual-core encoder with customization features
2016/10/18	1.22.0	DEC_ENABLE_AVC_MC_INTERPOL and DEC_DISABLE_AVC_MC_INTERPOL are added in VPU_DecGiveCommand().
2017/2/9	1.24.0	MPEG-4 profile and level of DecInitialInfo were updated in accordance with API.

Date	Revision	Change
2017/2/21	1.25.0	Support for CFrame50 compressed frame as one of encoder's source format was added. Also variable size -2 of VPU_DecUpdateBitstreamBuffer() was also desribed.
2017/6/16	1.26.0	Description on all enumerations and structures was reviewed
2017/7/25	2.0.0	Unused macros and variables, WAVE4/CODA7 product related code and description were removed.

Proprietary Notice

Address and Phone Number

Chips&Media
V&S Tower, 11/12/13th FL, 891-46, Daechi-dong, Gangnam-gu, 135-280
Seoul, Korea
Tel: +82-2-568-3767
Fax: +82-2-568-3767

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

Table of Contents

List of Figures

List of Tables

Preface

1. About This Document

This document is the API reference manual for .

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.

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

Chapter 1 **VPU API Overview**

This section describes the basic architecture of VPU (Video Processing Unit) 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 like VPU_EncOpen() or VPU_EncStartOneFrame()
- Decoder API : API functions for decoder operation including VPU_DecOpen(), VPU_DecGetInitialInfo(), and so forth

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.

In order 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. Decoder Operation Flow

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

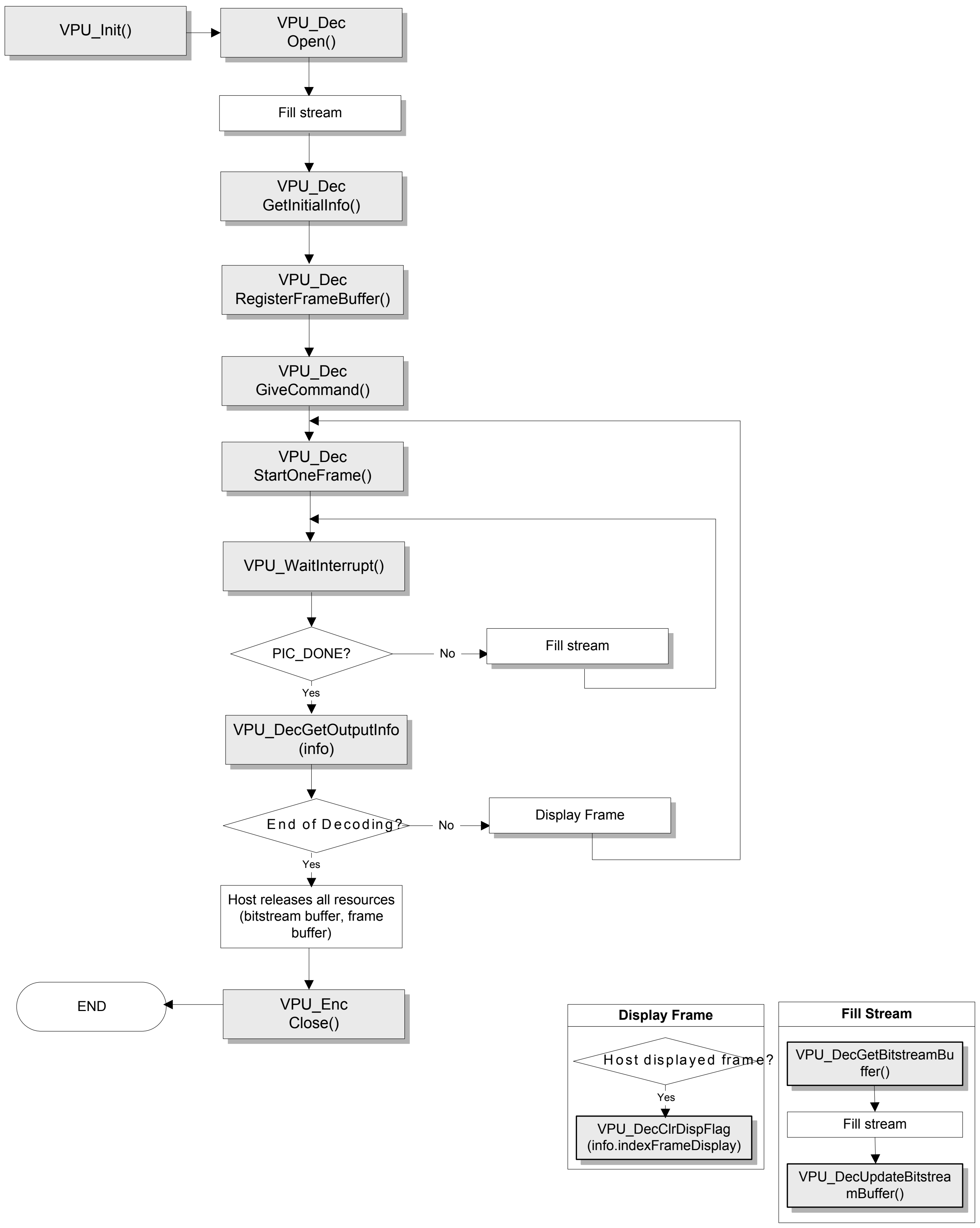

Figure 1.1. Decoder Operation Flow

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( ) checks the completion of picture decoder operation.

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

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

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

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

12. 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.

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)

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,
} 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)

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,
    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,
    ENABLE_LOGGING,
    DISABLE_LOGGING,
    DEC_GET_QUEUE_STATUS,
    ENC_GET_BW_REPORT,  // wave520 only
    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()"*.

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 |00xxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
    FORMAT_YUYV_P10_32BIT_LSB,         /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxx00| 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 |00xxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
    FORMAT_YVYU_P10_32BIT_LSB,         /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxx00| 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 |00xxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
    FORMAT_UYVY_P10_32BIT_LSB,         /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxx00| 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 |00xxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
    FORMAT_VYUY_P10_32BIT_LSB,         /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxx00| 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.

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_510,
    PRODUCT_ID_512,
    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     = 11,  // 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

2.3. Data Structures

ProductInfo

```
typedef struct {
    Uint32 productId;
    Uint32 fwVersion;
    Uint32 productName;
    Uint32 productVersion;
    Uint32 customerId;
    Uint32 stdDef0;
    Uint32 stdDef1;
    Uint32 confFeature;
    Uint32 configDate;
    Uint32 configRevision;
    Uint32 configType;
    Uint32 configVcore[4];
}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 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.

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;
        } wave;
    } 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;
} 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.

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;
} 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.

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 Recovery Point 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;
    // 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 dis-

played 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.

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.

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.

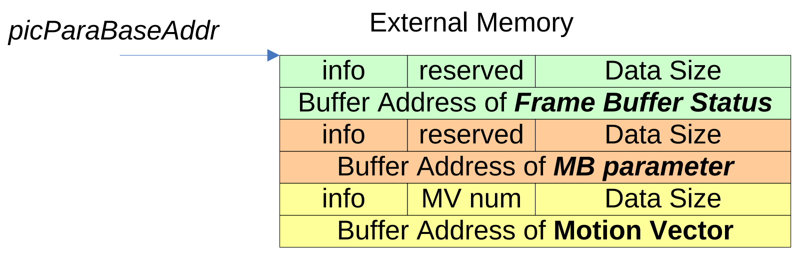
picParaBaseAddr
External Memory
info
reserved
Data Size
Buffer Address of Frame Buffer Status
info
reserved
Data Size
Buffer Address of MB parameter
info
MV num
Data Size
Buffer Address of Motion Vector

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.

About Chips&Media

Chips&Media, Inc. is a leading video IP provider. Over the past decade, they have been doing high quality product design focusing on hardware implementation of high speed, low power, cost effective video solution for H.264, MPEG-4, H.263, MPEG-1/2, VC-1, AVS, MVC, VP8, Theora, Sorenson, and up to recent H.265/HEVC along with its remarkable bandwidth reduction technology.

They have a broad range of products from low-end 1080p HD mobile solutions to high-end 4K 60fps dual-core solution for e.g. Ultra-HD TV or even brandnew HEVC/H.265 and VP9 decoder IP to fulfill target demands from various multimedia device markets.

Chips&Media's IPs are proven in over a hundred million of devices and by more than 60 top-tier semiconductor companies. The headquarter is located in Seoul, Korea (Republic of) with sales offices in Japan, China, Taiwan, and the USA. To find further information, please visit the company's web site at *http://www.chipsnmedia.com*