WAVE511 HEVC and AVC Multi-Decoder IP

API Reference Manual

Version 1.9.0

WAVE511 HEVC and AVC Multi-Decoder IP: API Reference Manual

Version 1.9.0

Revision History

Date	Revision	Change
2018/7/30	1.0.0	Release version was generated.
2018/8/27	1.1.0	API control flow with parameter change was added.
2018/10/2	1.2.0	The latest API manual was generated for release.
2018/10/31	1.3.0	The API manual was generated with custom features on.
2019/5/29	1.4.0	P10_16BIT_MSB and P10_16BIT_LSB format description of FrameBufferFormat structure was fixed.
2019/6/25	1.5.0	forcedIdrHeaderEnable was included.
2019/3/27	1.4.0	Multi-core, new scheduler, custom feature related structures and commands were added.
2019/5/23	1.5.0	ENC_SET_PARAM in CodecCommand was described.
2019/5/27	1.6.0	Some variables were further described: numOfTotMBs, numOfTotMBsInDisplay, framecycle/seekCycle/parseCycle/DecodedCycle, outputFlag, frameSkipDisable, userQpMax, encNuts/encNuts1, and PicDistortionLow/PicDistortionHigh.
2019/6/24	1.7.0	EncWaveParam.profile was described for AVC encoder. maxIntraSize, userMaxDeltaQp, userMinDeltaQp and userQpMin of EncOpenParam structure were described for intended use for CODA9. Missing definitions or descriptions in EncOpenParam were included.
2019/6/28	1.7.1	Description on EncOpenParam.bitrate was fixed.
2019/8/29	1.8.0	rcWeightParam, rcWeightBuf, MaxContinuousFrameSkipNum, MaxContinuousFrameDropNum and a few others were described.
2019/9/16	1.9.0	encodeHrdRbspInVUI was 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

Preface ix

1. About This Document ix
2. Intended audience ix
3. Scope ix
4. Typographical conventions ix
5. Further reading ix

Chapter 1. VPU API Overview

1.1. Basic Architecture 1
1.2. Decoder Operation Flow 2
1.3. Decoder Reset Scenario 5
1.4. DPB Flush Flow 7

Chapter 2. DATA TYPE DEFINITIONS

2.1. Data Types 9
- Uint8 9
- Uint32 9
- Uint16 9
- Int8 9
- Int32 9
- Int16 9
- PhysicalAddress 10
- BYTE 10
- VpuHandle 10
- DecHandle 10
- EncHandle 10

2.2. Eumerations 11
- CodStd 11
- SET_PARAM_OPTION 11
- DEC_PIC_HDR_OPTION 12
- DEC_PIC_OPTION 12
- ENC_QUERY_WRPTR_SEL 13
- RetCode 13
- CodecCommand 16
- AVCErrorConcealMode 29
- CbCrOrder 29
- MirrorDirection 30
- FrameBufferFormat 30
- ScalerImageFormat 32
- PackedFormatNum 32
- InterruptBit 33
- Wave5InterruptBit 33
- PicType 33
- AvcNpfFieldInfo 35
- FrameFlag 35
- BitStreamMode 35
- SWResetMode 36
- ProductId 36
- TiledMapType 36
- FramebufferAllocType 38

Wave5ChangeParam 38
Mp4HeaderType 39
AvcHeaderType 39
WaveEncHeaderType 40
ENC_PIC_CODE_OPTION 40
GOP_PRESET_IDX 40
2.3. Data Structures 42
VpuAttr 42
TiledMapConfig 44
DRAMConfig 45
FrameBuffer 45
FrameBufferAllocInfo 47
VpuRect 48
ThoScaleInfo 49
Vp8ScaleInfo 49
LowDelayInfo 50
SecAxiUse 50
CacheSizeCfg 51
MaverickCacheConfig 52
DecParamSet 53
AvcVuiInfo 53
MP2BarDataInfo 55
MP2PicDispExtInfo 55
DecOpenParam 56
DecInitialInfo 59
DecParam 64
DecOutputExtData 66
Vp8PicInfo 67
MvcPicInfo 67
AvcFpaSei 68
AvcHrdInfo 69
AvcRpSei 70
H265RpSei 70
SvacInfo 71
Avs2Info 71
Av1Info 71
DecOutputInfo 72
DecGetFramebufInfo 80
QueueStatusInfo 81
EncMp4Param 81
EncH263Param 82
CustomGopPicParam 82
CustomGopParam 83
WaveCustomMapOpt 83
HevcVuiParam 84
EncWaveParam 85
EncChangeParam 95
AvcPpsParam 101
EncAvcParam 102
EncSliceMode 103
EncOpenParam 104
EncInitialInfo 111
EncCodeOpt 112
EncParam 113
EncReportInfo 117

EncOutputInfo 117
EncHeaderParam 120

Chapter 3. API DEFINITIONS

List of Figures

1.1. Decoder Operation Flow 3
1.2. VPU API Scenario with Decoder Reset 5
1.3. Flushing DPB 7

List of Tables

2.1. Upsampling Ratio by Scale Factor 50

Preface

1. About This Document

This document is the API reference manual for WAVE511 Video Decoder IP .

2. Intended audience

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

3. Scope

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

4. Typographical conventions

The following typographical conventions are used in this document:

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

5. Further reading

This section lists documents which are related to this product.

- *WAVE511 Datasheet*
- *WAVE511 Programmer's Guide*
- *WAVE511 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.

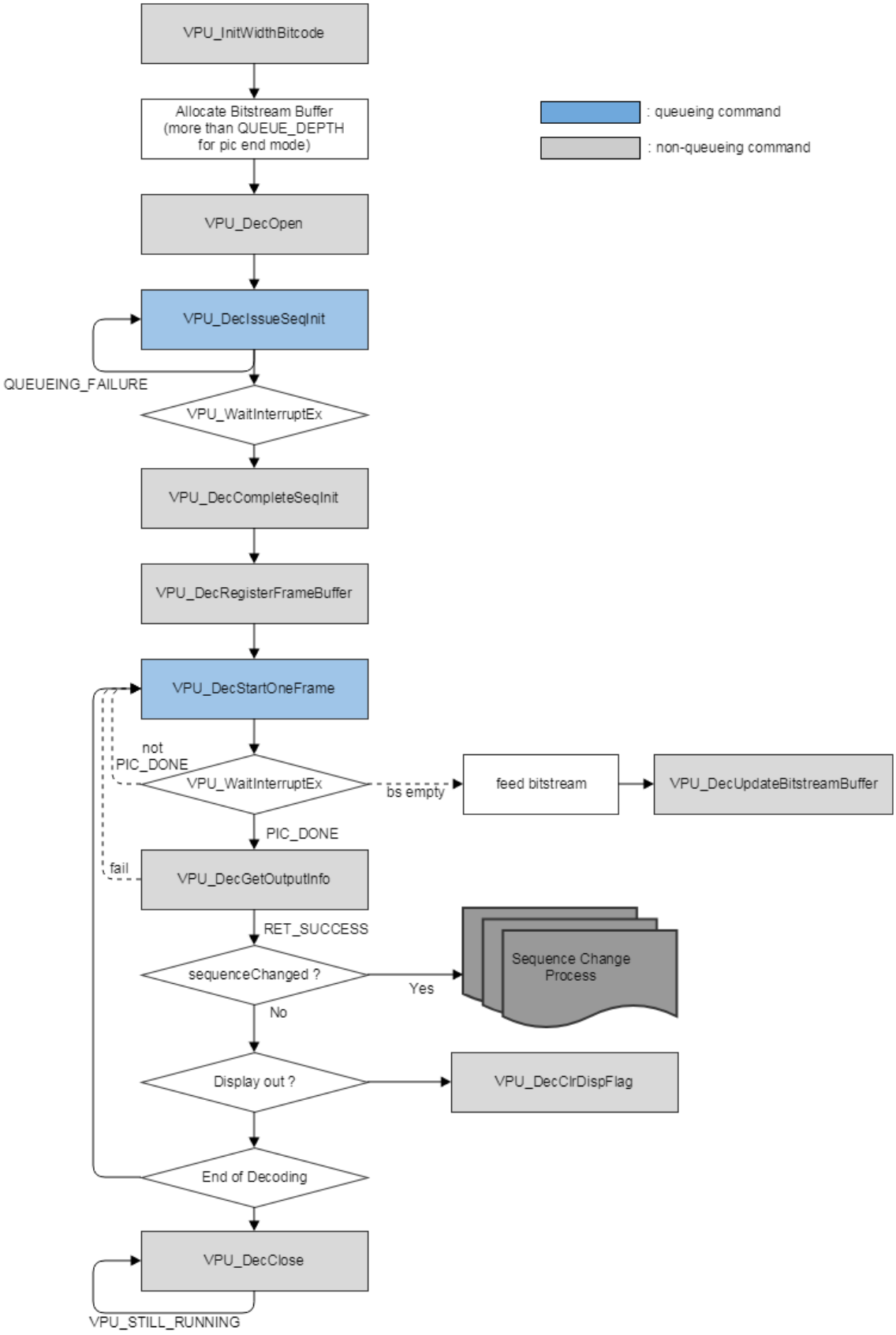


Figure 1.1. Decoder Operation Flow

1. VPU_InitWithBitcode() loads VPU firmware whose path is defined by BIT_CODE_FILE_PATH in config.h file and begins to boot up.
2. Allocate bitstream buffer. In case of line buffer mode, bitstream buffer should be assigned as equal as QUEUE_DEPTH or more than QUEUE_DEPTH.
3. Open a decoder instance by using VPU_DecOpen().
4. Decodes a header of video sequence by using VPU_DecIssueSeqInit(). In this call, bitstream buffer mode and RdPtr/WrPtr are specified.
5. VPU_DecCompleteSeqInit() returns crucial sequence information for decode operation such as picture size, frame rate, required number of frame buffers, etc.
6. Application should allocate minFrameBufferCount of frame buffers and register the frame buffer for VPU by calling VPU_DecRegisterFrameBuffer().
7. Starts picture decode operation with VPU_DecStartOneFrame().
   a. Fill more bitstream in the buffer if stream empty occurrs during decoding operation.
8. VPU_DecGetOutputInfo() returns the result of picture decode operation and output information. The address and size of report buffer should be given in this call.
9. Go back to 7. to go on decode operation until the entire sequence is completely decoded.
10. Display the decoded frame and call VPU_DecClrDispFlag() to clear the buffer displayuse flag.
11. Terminate the sequence operation by calling VPU_DecClose().

1.3. Decoder Reset Scenario

Figure 1.2, "VPU API Scenario with Decoder Reset" shows the reset scenario of decoder. Host application should follow the steps below in order to reset safely during decoding sequence.

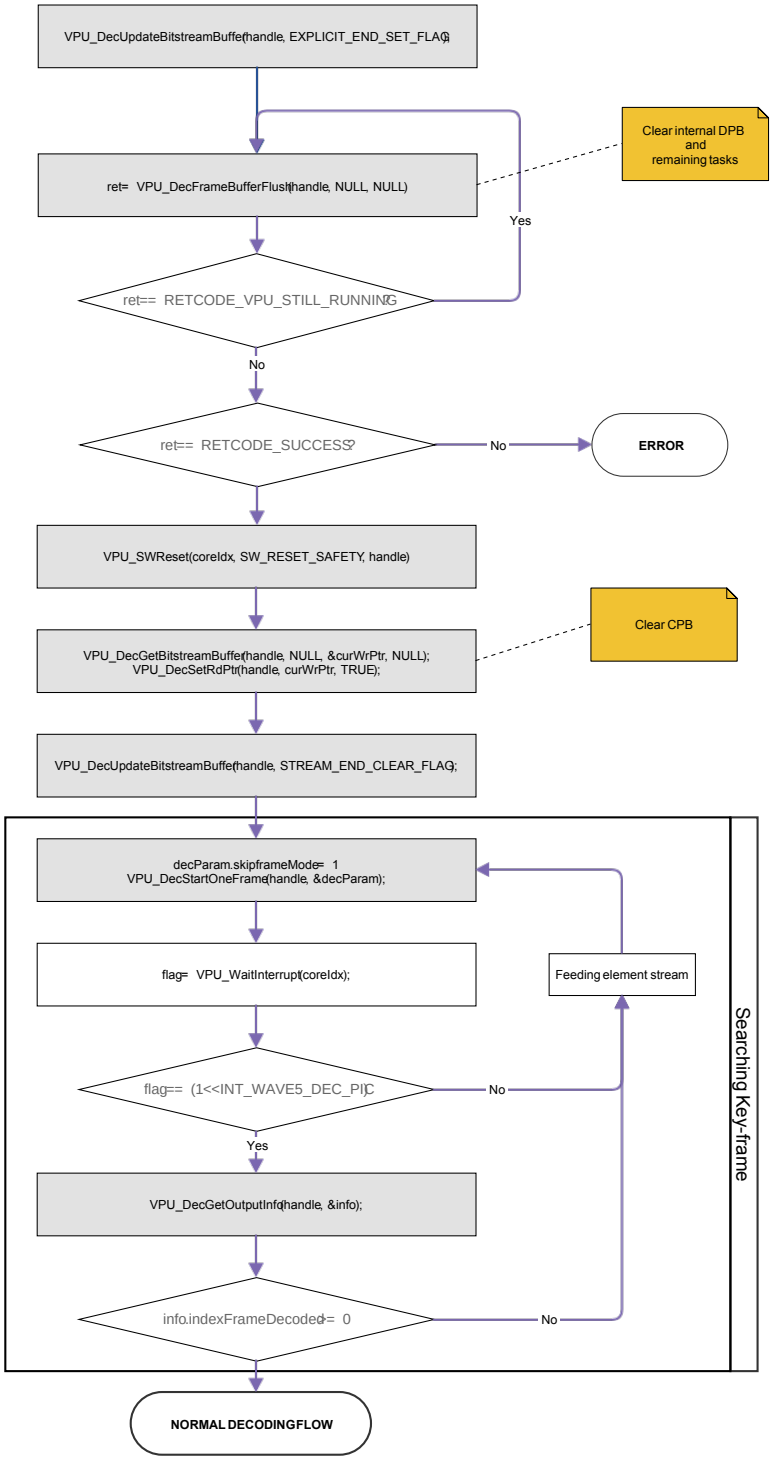


Figure 1.2. VPU API Scenario with Decoder Reset

1. Call VPU_DecUpdateBistreamBuffer() with the input argument of `size` of -2 (EXPLICIT_END_SET_FLAG), which ends up the current pending job anyway.

2. Keep calling VPU_DecFrameBufferFlush() to clear internal DPB and other remaining tasks until it returns RETCODE_SUCCESS.

3. Call VPU_SWReset() with SWResetMode of SW_RESET_SAFETY.

4. Then call VPU_DecGetBitstreamBuffer to get the current write pointer. Subsequently call VPU_DecSetRdPtr() to make the write pointer and read pointer in the same place. This is to initialize the pointers in the bitstream buffer.

5. Call VPU_DecUpdateBistreamBuffer() with the input argument of `size` of -1 (STREAM_END_CLEAR_FLAG) to clear the EXPLICIT_END_SET_FLAG.

6. Host application is now ready to call VPU_DecStartOneFrame() with decParam.skipframeMode of 1. VPU_DecStartOneFrame() is kept calling until a key frame is decoded (DecOutputInfo.indexFrameDecoded is equal to or greater than 0).

   While seeking a key frame, host application might have INT_WAVE5_BSBUF_EMPTY interrupt. In that case, feed more stream in the bitstream buffer.

1.4. DPB Flush Flow

There might be some situation in which host application needs to flush all the current DPB with display flag and restart decoding. Application can follow some steps as guided in *Figure 1.3, "Flushing DPB"* which allows decoder to flush the current DPB and to safely return to decode operation.

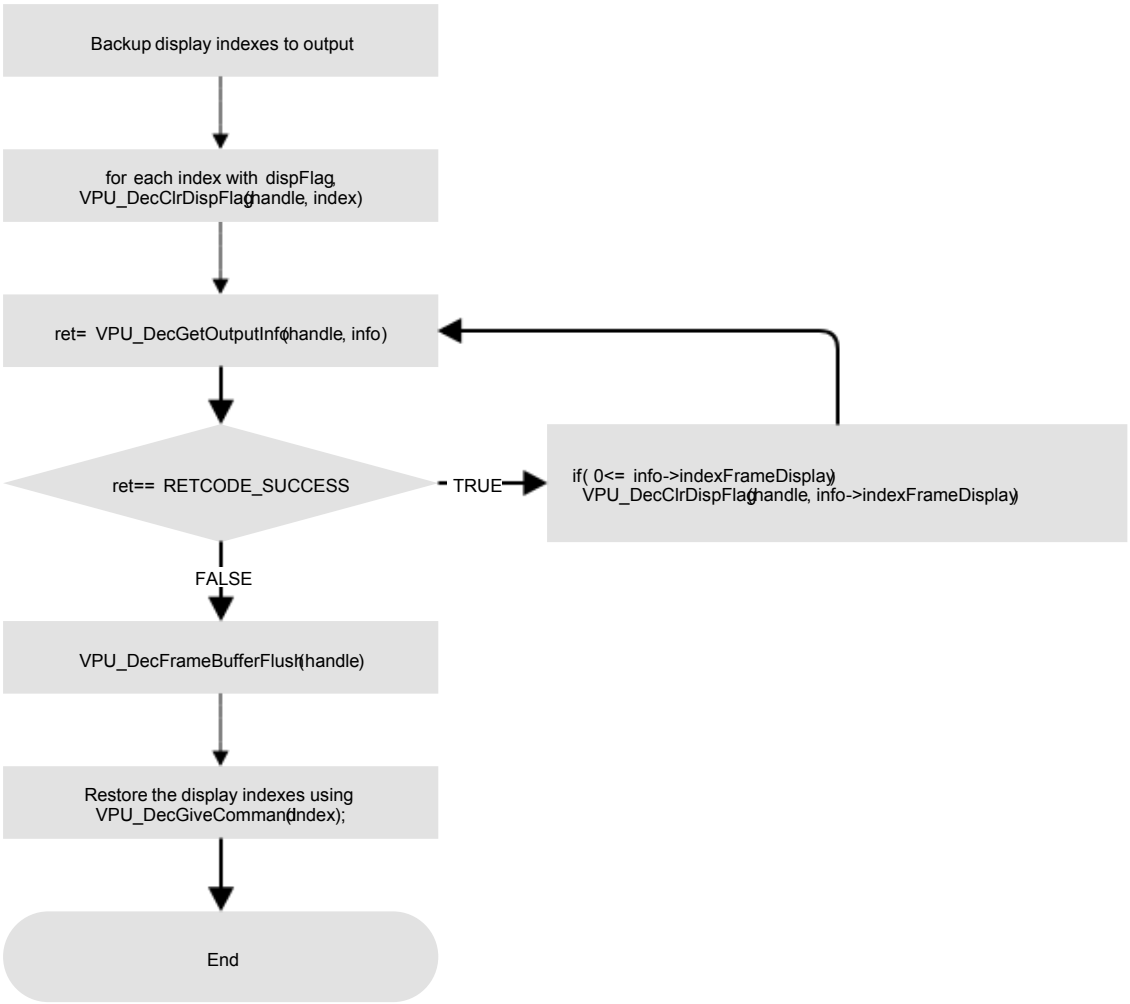


Figure 1.3. Flushing DPB

1. Backup the DPBs marked with display flag that have not been cleared yet before calling VPU_DecFrameBufferFlush().
2. Clear all the DPBs with display flag.
3. Retrieve the information of the DPBs, *the section called "DecOutputInfo"* by calling VPU_DecGetOutputInfo() and clear the index of the DPBs.
4. Call VPU_DecFrameBufferFlush() to initialize DPBs.
5. Call VPU_DecGiveCommand(DEC_SET_DISPLAY_FLAG) which sets a display flag to the backed up DPB, which is an optional step.

6. Call VPU_DecStartOneFrame() to continue decoding a frame.

The code below is written in the VPU reference software, which demonstrates the safe process of clearing DPB and starting over frame decode operation.

```
static void ClearDpb(DecHandle handle, Queue* displayQ, BOOL backupDpb)
{
  Uint32          timeoutCount;
  Int32           intReason;
  DecOutputInfo   outputInfo, *p;
  Uint32          index;
  Uint32          flushedFbs      = 0;

  if (TRUE == backup) {
          while ((p=Queue_Dequeue(displayQ))) {
          flushedFbs |= (1<<p->indexFrameDisplay);
          VPU_DecClrDispFlag(handle, p->indexFrameDisplay);
      }
  }

  while (RETCODE_SUCCESS == VPU_DecGetOutputInfo(ctx->handle, &outputInfo)) {
      if (0 <= outputInfo.indexFrameDisplay) {
      flushedFbs |= outputInfo.indexFrameDisplay;
      VPU_DecClrDispFlag(ctx->handle, outputInfo.indexFrameDisplay);
      VLOG(INFO, "<%s> FLUSH DPB INDEX: %d\n", __FUNCTION__, outputInfo.indexFrameDisplay);
      }
      osal_msleep(1);
  }

  VLOG(INFO, "========== FLUSH FRAMEBUFFER & CMDs ========== \n");
  timeoutCount = 0;
  while (VPU_DecFrameBufferFlush(ctx->handle, NULL, NULL) == RETCODE_VPU_STILL_RUNNING) {
      /* Clear an interrupt */
      if (0 < (intReason=VPU_WaitInterruptEx(ctx->handle, VPU_WAIT_TIME_OUT_CQ))) {
      VPU_ClearInterruptEx(ctx->handle, intReason);
      VPU_DecGetOutputInfo(ctx->handle, &outputInfo); // ignore return value, outputinfo
      }

      if (timeoutCount >= VPU_BUSY_CHECK_TIMEOUT) {
          VLOG(ERR, "NO RESPONSE FROM VPU_DecFrameBufferFlush()\n");
      }
      timeoutCount++;
  }

  if (TRUE == backupDpb) {
      for (index=0; index<32; index++) {
          if (flushedFbs & (1<<index)) {
              VLOG(INFO, "SET DISPLAY FLAG : %d\n", index);
              VPU_DecGiveCommand(handle, DEC_SET_DISPLAY_FLAG , &index);
          }
      }
   }
}
```

Chapter 2
DATA TYPE DEFINITIONS

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

2.1. Data Types

Uint8

```
typedef uint8_t         Uint8;
```

Description

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

Uint32

```
typedef uint32_t        Uint32;
```

Description

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

Uint16

```
typedef uint16_t        Uint16;
```

Description

This type is a 16-bit unsigned integral type.

Int8

```
typedef int8_t          Int8;
```

Description

This type is an 8-bit signed integral type.

Int32

```
typedef int32_t         Int32;
```

Description

This type is a 32-bit signed integral type.

Int16

```
typedef int16_t         Int16;
```

Description

This type is a 16-bit signed integral type.

PhysicalAddress

```
typedef Uint32 PhysicalAddress;
```

Description

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

BYTE

```
typedef unsigned char   BYTE;
```

Description

This type is an 8-bit unsigned integral type.

VpuHandle

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

Description

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

DecHandle

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

Description

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

Note This type is vaild for decoder only.

EncHandle

```
typedef EncInst * EncHandle;
```

Description

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

Note This type is vaild for encoder only.

2.2. Eumerations

CodStd

```
typedef enum {
    STD_AVC,
    STD_VC1,
    STD_MPEG2,
    STD_MPEG4,
    STD_H263,
    STD_DIV3,
    STD_RV,
    STD_AVS,
    STD_THO = 9,
    STD_VP3,
    STD_VP8,
    STD_HEVC,
    STD_VP9,
    STD_AVS2,
    STD_SVAC,
    STD_AV1,
    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, RealVideo 8/9/10, AVS Jizhun/Guangdian profile, AVS2, Theora, VP3, VP8/VP9 and SVAC.

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    = 0x10,
#ifdef SUPPORT_LOOK_AHEAD_RC
    OPT_LOOKAHEAD_PARAM_1 = 0x100,
    OPT_LOOKAHEAD_PARAM_2 = 0x101,
    OPT_LOOKAHEAD_PARAM_3 = 0x102,
    OPT_LOOKAHEAD_PARAM_4 = 0x103,
    OPT_LOOKAHEAD_PARAM_5 = 0x104,
#endif
} 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 (WAVE Encoder only)

OPT_LOOKAHEAD_PARAM_1
SET_PARAM command option for Look Ahead RC parameters setting

OPT_LOOKAHEAD_PARAM_2
SET_PARAM command option for Look Ahead RC parameters setting

OPT_LOOKAHEAD_PARAM_3
SET_PARAM command option for Look Ahead RC parameters setting

OPT_LOOKAHEAD_PARAM_4
SET_PARAM command option for Look Ahead RC parameters setting

OPT_LOOKAHEAD_PARAM_5
SET_PARAM command option for Look Ahead RC parameters setting

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,
    SKIP_SVAC_BL        = 0x20,
    SKIP_SVAC_EL        = 0x40
} 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.

SKIP_SVAC_BL
It skips base layer pictures.

SKIP_SVAC_EL
It skips enhance layer pictures.

ENC_QUERY_WRPTR_SEL

```
typedef enum {
    GET_ENC_PIC_DONE_WRPTR      = 0x00,
    GET_ENC_BSBUF_FULL_WRPTR    = 0x01,
    GET_ENC_LOW_LATENCY_WRPTR   = 0x02,
} ENC_QUERY_WRPTR_SEL;
```

Description

This is an enumeration for declaring options of getting a write pointer of bitstream buffer. (WAVE encoder only)

GET_ENC_PIC_DONE_WRPTR
It reads the write pointer of bitstream buffer after picture encoding is done.

GET_ENC_BSBUF_FULL_WRPTR
It reads the write pointer of bitstream buffer when buffer full is occurred.

GET_ENC_LOW_LATENCY_WRPTR
It reads the write pointer of bitstream buffer when low latency encoding is done.

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_VLC_BUF_FULL,
    RETCODE_INVALID_SFS_INSTANCE,
    RETCODE_ERROR_FW_FATAL,
} RetCode;
```

Description

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

RETCODE_SUCCESS
This means that operation was done successfully.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

RETCODE_STREAM_BUF_FULL
This means that stream buffer is full in encoder.

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

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

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

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

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

RETCODE_VLC_BUF_FULL
This means that VLC buffer is full in encoder. (WAVE5 only)

RETCODE_INVALID_SFS_INSTANCE
This means that current instance can't run sub-framesync. (already an instance was running with sub-frame sync (WAVE5 only)

RETCODE_ERROR_FW_FATAL
This means that firmware detects a fatal error. (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,

    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_GET_SCALER_INFO,
    DEC_SET_SCALER_INFO,
    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,
    DEC_SET_TEMPORAL_ID_MODE,
    ENC_SET_PARAM,
    //vpu put header stream to bitstream buffer
    ENC_PUT_VIDEO_HEADER,
    ENC_SET_INTRA_MB_REFRESH_NUMBER,
    ENC_ENABLE_HEC,
    ENC_DISABLE_HEC,
    ENC_SET_SLICE_INFO,
    ENC_SET_GOP_NUMBER,
    ENC_SET_INTRA_QP,
    ENC_SET_BITRATE,
    ENC_SET_FRAME_RATE,

    ENC_SET_PARA_CHANGE,
    ENABLE_LOGGING,
    DISABLE_LOGGING,
    DEC_GET_QUEUE_STATUS,
    ENC_GET_QUEUE_STATUS,
    GET_BANDWIDTH_REPORT,      /* WAVE52x products. */
    ENC_WRPTR_SEL,
    SET_CYCLE_PER_TICK,
    ENC_GET_SRC_BUF_FLAG,
    GET_DEBUG_INFORM,
#ifdef SUPPORT_LOOK_AHEAD_RC
    ENC_SET_LARC_DATA, /*  < This command sets Look Ahead RC data */
#endif
    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.

ENABLE_ROTATION
This command enables rotation. In this case, parameter is ignored. This command returns RETCODE_SUCCESS.

DISABLE_ROTATION
This command disables rotation. In this case, parameter is ignored. This command returns RETCODE_SUCCESS.

ENABLE_MIRRORING
This command enables mirroring. In this case, parameter is ignored. This command returns RETCODE_SUCCESS.

DISABLE_MIRRORING
This command disables mirroring. 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. (CODA decoder only) The `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. (CODA decoder only) The `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.

DEC_GET_SEQ_INFO

This command returns the information of the current sequence with *the section called "DecInitialInfo"*. This command is mainly used for getting new sequence information after change of sequence.

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.

DEC_SET_SEQ_CHANGE_MASK

This command sets DEC_SET_SEQ_CHANGE_MASK which allows VPU to notify change of sequence information such as picture size, DPB count, profile, and bit-depth.

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_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_DERING

This command enables deringing filter of the post-rotator. (CODA decoder only) In this case, parameter is ignored. This command returns RETCODE_SUCCESS.

DISABLE_DERING

This command disables deringing filter of the post-rotator. (CODA decoder only) In this case, parameter is ignored. This command returns RETCODE_SUCCESS.

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 *the section called "SecAxiUse"* which represents an enable flag and physical address which is related with the secondary channel.

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
This command sets the DRAM attributes to use tiled map. The argument parameter is interpreted as a pointer to *the section called "DRAMConfig"*. It returns RETCODE_INVALID_PARAM when any value is not given to the argument parameter, parameter. (CODA960 only)

GET_DRAM_CONFIG
This command gets the DRAM attributes to use tiled map. The argument parameter is interpreted as a pointer to *the section called "DRAMConfig"*. It returns RETCODE_INVALID_PARAM when any value is not given to the argument parameter, parameter. (CODA960 only)

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

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

This command sets the interrupt flag of user data buffer overflow. (CODA9 only)

- 0 : interrupt mode
- 1 : interrupt disable mode

SET_CACHE_CONFIG

This command sets the configuration of cache. The `parameter` is interpreted as a pointer to MaverickCacheConfig. (CODA9 only)

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. (CODA9 only)

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. (CODA9 decoder only) 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.

GET_LOW_DELAY_OUTPUT
This command gets the low delay decoding options which enable low delay decoding and indicate the number of MB row. (CODA decoder only) 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_SET_FRAME_DELAY
HOST can set the frameBufDelay value of *the section called "DecInitialInfo"*. (CODA9 H.264/AVC decoder only) This command is useful when HOST is sure of display reorder delay of stream and wants to display soonner than frameBufDelay value of *the section called "DecInitialInfo"* which is calculated based on video specification by VPU. However, if HOST set an invalid frameBufDelay value, it might lead to failure of display.

DEC_SET_WTL_FRAME_FORMAT
This command sets FrameBufferFormat for WTL.

DEC_GET_FIELD_PIC_TYPE
This command gets a field picture type of decoded picture after INT_BIT_DEC_FIELD interrupt is issued.

DEC_GET_DISPLAY_OUTPUT_INFO
HOST can get decoder output information according to display index in *the section called "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);
```

DEC_ENABLE_REORDER
HOST can enable display buffer reordering when decoding H.264 streams. (CODA9 H.264 decoder and WAVE decoder only) 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_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_FREE_FRAME_BUFFER

HOST can free all the frame buffers allocated by VPUAPI. (CODA9 only) This command is useful when VPU detects sequence change. For example, if HOST knows resolution change while decoding through `sequenceChanged` variable of *the section called "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_GET_FRAMEBUF_INFO

This command gives HOST the information of framebuffer in *the section called "DecGetFramebufInfo"*. (WAVE only)

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_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_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.

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. (H.265/HEVC decoder only)

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. (VP9 decoder only)

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. (VP9 decoder only)

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. (VP9 decoder only)

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. (VP9 decoder only)

DEC_FREE_FBC_TABLE_BUFFER
This command frees one FBC table to deal with inter-frame resolution change. (VP9 decoder only)

DEC_FREE_MV_BUFFER
This command frees one MV buffer to deal with inter-frame resolution change. (VP9 decoder only)

DEC_ALLOC_FBC_Y_TABLE_BUFFER
This command allocates one FBC luma table to deal with inter-frame resolution change. (VP9 decoder only)

DEC_ALLOC_FBC_C_TABLE_BUFFER
This command allocates one FBC chroma table to deal with inter-frame resolution change. (VP9 decoder only)

DEC_ALLOC_MV_BUFFER
This command allocates one MV buffer to deal with inter-frame resolution change. (VP9 decoder only)

DEC_SET_TEMPORAL_ID_MODE
This command specifies the target temporal layer id selection mode. (WAVE decoder only)

ENC_SET_PARAM
This command writes an RBSP format of the SPS/PPS to parameter buffer.(CODA9 encoder only) This command has one of the following return codes.::

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument cmd was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an MPEG4 encoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` was invalid, which means it has a null pointer, or given values for some member variables are improper values.

ENC_PUT_VIDEO_HEADER

This command inserts an MPEG4 header syntax or SPS or PPS to the HEVC/AVC bitstream to the bitstream during encoding. It is valid for all types of encoders. The argument `parameter` is interpreted as a pointer to *the section called "EncHeaderParam"* holding

- buf is a physical address pointing the generated stream location
- size is the size of generated stream in bytes
- headerType is a type of header that HOST application wants to generate and have values as *the section called "Mp4HeaderType"*, *the section called "AvcHeaderType"*, *the section called "WaveEncHeaderType"*.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument cmd was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an MPEG4 encoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or headerType was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not recieved any response from VPU and has timed out.

ENC_SET_INTRA_MB_REFRESH_NUMBER

This command changes intra MB refresh number of header syntax during encoding. (ChangeParam command for CODA9 encoder only) The argument `parameter` is interpreted as a pointer to integer which represents an intra refresh number. It should be between 0 and macroblock number of encoded picture.

This command returns the following code.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not received any response from VPU and has timed out.

ENC_ENABLE_HEC

This command enables HEC(Header Extension Code) syntax of MPEG4.

This command ignores the argument `parameter` and returns one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument, cmd, was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an MPEG4 encoder instance.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not received any response from VPU and has timed out.

ENC_DISABLE_HEC
This command disables HEC(Header Extension Code) syntax of MPEG4.

This command ignores the argument `parameter` and returns one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument, cmd, was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an MPEG4 encoder instance.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not received any response from VPU and has timed out.

ENC_SET_SLICE_INFO
This command changes slice inforamtion of header syntax during encoding. (ChangeParam command for CODA9 encoder only) The argument `parameter` is interpreted as a pointer to *the section called "EncSliceMode"* structure holding

- sliceMode is a mode which means enabling multi slice structure
- sliceSizeMode is the mode representing mode of calculating one slicesize
- sliceSize is the size of one slice.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not received any response from VPU and has timed out.

ENC_SET_GOP_NUMBER
This command changes GOP number of header syntax during encoding. (ChangeParam command for CODA9 encoder only) The argument `parameter` is interpreted as a pointer to the integer which represents a GOP number.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not received any response from VPU and has timed out.

ENC_SET_INTRA_QP
This command changes intra QP value of header syntax during encoding. (ChangeParam command for CODA9 encoder only) The argument `parameter` is interpreted as a pointer to the integer which represents a Constant I frame QP. The Constant I frame QP should be between 1 and 31 for MPEG4 and between 0 and 51 for H.264/AVC.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument `cmd` was invalid which means the given cmd was undefined, or not allowed in the current instance. In this case, the current instance might not be an encoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not received any response from VPU and has timed out.

ENC_SET_BITRATE

This command changes bitrate inforamtion of header syntax during encoding. (ChangeParam command for CODA9 encoder only) The argument `parameter` is interpreted as a pointer to the integer which represents a bitrate. It should be between 0 and 32767.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument `cmd` was invalid which means the given `cmd` was undefined, or not allowed in the current instance. In this case, the current instance might not be an encoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not received any response from VPU and has timed out.

ENC_SET_FRAME_RATE

This command changes frame rate of header syntax during encoding. (ChangeParam command for CODA9 encoder only) The argument `parameter` is interpreted as a pointer to the integer which represents a frame rate value. The fraem rate should be greater than 0.

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument `cmd` was invalid which means the given `cmd` was undefined, or not allowed in the current instance. In this case, the current instance might not be an encoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not received any response from VPU and has timed out.

ENC_SET_PARA_CHANGE

This command changes encoding parameter(s) during the encoding operation. (WAVE encoder only) The argument `parameter` is interpreted as a pointer to *the section called "EncChangeParam"* structure holding

- enable_option : Set an enum value that is associated with parameters to change (multiple option allowed).

For instance, if bitrate and framerate need to be changed in the middle of encoding, that requires some setting like below.

```
EncChangeParam changeParam;
changeParam.enable_option  = ENC_RC_TARGET_RATE_CHANGE | ENC_FRAME_RATE_CHANGE;
```

```
changeParam.bitrate        = 14213000;
changeParam.frameRate      = 15;
VPU_EncGiveCommand(handle, ENC_SET_PARA_CHANGE, &changeParam);
```

This command has one of the following return codes.

- RETCODE_SUCCESS: Operation was done successfully, which means the requested header syntax was successfully inserted.
- RETCODE_INVALID_COMMAND: This means the given argument `cmd` was invalid which means the given `cmd` was undefined, or not allowed in the current instance. In this case, the current instance might not be an H.264/AVC encoder instance.
- RETCODE_INVALID_PARAM: The given argument parameter `parameter` or `headerType` was invalid, which means it has a null pointer, or given values for some member variables are improper values.
- RETCODE_VPU_RESPONSE_TIMEOUT: Operation has not received any response from VPU and has timed out.

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_GET_QUEUE_STATUS

This command returns the number of queued commands for the current decode instance and the number of queued commands for the total decode instances.

ENC_GET_QUEUE_STATUS

This command returns the number of queued commands for the current encode instance and the number of queued commands for the total encode instances.

GET_BANDWIDTH_REPORT

This command reports the amount of bytes which are transferred on AXI bus.

ENC_WRPTR_SEL

This command sets *the section called "ENC_QUERY_WRPTR_SEL"*.

SET_CYCLE_PER_TICK

This commmand sets the count of cycles per tick which is mostly a constant value to internal timer. Cycle per tick is used to calculate the number of cycle from the number of ticks that firmware returns. (WAVE5 only) This command returns RETCODE_SUCCESS.

ENC_GET_SRC_BUF_FLAG
This command obtains the value of releaseSrcFlag when srcReleaseIntEnable is 1 and source release interrupt is occurred. (WAVE5 only) This command has one of the following return codes.::

- RETCODE_SUCCESS: Requested operation was done successfully.
- RETCODE_QUERY_FAILURE: This means this query command was not successful. (WAVE5 only)
- RETCODE_INVALID_COMMAND: This means the given argument `cmd` was invalid which means the given `cmd` was undefined, or not allowed in the current instance.

GET_DEBUG_INFORM
This command returns the debug information on error situations such as hang-up. (WAVE5 only) This command has one of the following return codes.::

- RETCODE_SUCCESS: Requested operation was done successfully.
- RETCODE_REPORT_NOT_READY: This means that report is not ready for this command. (WAVE5 only)
- RETCODE_QUERY_FAILURE: This means this query command was not successful. (WAVE5 only)
- RETCODE_INVALID_COMMAND: This means the given argument `cmd` was invalid which means the given `cmd` was undefined, or not allowed in the current instance.

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.

MIRDIR_NONE
No mirroring

MIRDIR_VER
Vertical mirroring

MIRDIR_HOR
Horizontal mirroring

MIRDIR_HOR_VER
Horizontal and vertical mirroring

FrameBufferFormat

```
typedef enum {
   FORMAT_ERR           = -1,
   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 |000000xx|xxxxxxxx | msb */
   FORMAT_420_P10_16BIT_LSB ,    /* lsb |xxxxxxx |xx000000 | 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 |000000xx |xxxxxxxx | msb */
   FORMAT_422_P10_16BIT_LSB ,    /* lsb |xxxxxxxx |xx000000 | msb */
   FORMAT_422_P10_32BIT_MSB ,    /* lsb |00xxxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
   FORMAT_422_P10_32BIT_LSB ,    /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxxx00| msb */

   FORMAT_YUYV              ,
   FORMAT_YUYV_P10_16BIT_MSB,    /* lsb |000000xxxxxxxxxx | msb */
   FORMAT_YUYV_P10_16BIT_LSB,    /* lsb |xxxxxxxxxx000000 | msb */
   FORMAT_YUYV_P10_32BIT_MSB,    /* lsb |00xxxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
   FORMAT_YUYV_P10_32BIT_LSB,    /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxxx00| msb */
   FORMAT_YVYU              ,
```

```
    FORMAT_YVYU_P10_16BIT_MSB,    /* lsb |000000xxxxxxxxxx | msb */
    FORMAT_YVYU_P10_16BIT_LSB,    /* lsb |xxxxxxxxxx000000 | msb */
    FORMAT_YVYU_P10_32BIT_MSB,    /* lsb |00xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
    FORMAT_YVYU_P10_32BIT_LSB,    /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxxx00| msb */
    FORMAT_UYVY               ,
    FORMAT_UYVY_P10_16BIT_MSB,    /* lsb |000000xxxxxxxxxx | msb */
    FORMAT_UYVY_P10_16BIT_LSB,    /* lsb |xxxxxxxxxx000000 | msb */
    FORMAT_UYVY_P10_32BIT_MSB,    /* lsb |00xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
    FORMAT_UYVY_P10_32BIT_LSB,    /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxxx00| msb */
    FORMAT_VYUY               ,
    FORMAT_VYUY_P10_16BIT_MSB,    /* lsb |000000xxxxxxxxxx | msb */
    FORMAT_VYUY_P10_16BIT_LSB,    /* lsb |xxxxxxxxxx000000 | msb */
    FORMAT_VYUY_P10_32BIT_MSB,    /* lsb |00xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx| msb */
    FORMAT_VYUY_P10_32BIT_LSB,    /* lsb |xxxxxxxxxxxxxxxxxxxxxxxxxxxx00| msb */
    FORMAT_MAX,
} FrameBufferFormat;
```

Description

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

FORMAT_YUYV
8bit packed format : Y0U0Y1V0 Y2U1Y3V1 …

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ScalerImageFormat

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

Description

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

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

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

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

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

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

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

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

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

PackedFormatNum

```
typedef enum {
    NOT_PACKED = 0,
```

```
    PACKED_YUYV,
    PACKED_YVYU,
    PACKED_UYVY,
    PACKED_VYUY,
} PackedFormatNum;
```

Description

This is an enumeration type for representing YUV packed format.

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,
#ifdef SUPPORT_SOURCE_RELEASE_INTERRUPT
    INT_WAVE5_ENC_SRC_RELEASE   = 10,
#endif
    INT_WAVE5_ENC_LOW_LATENCY   = 13,
    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_KEY          = 0,
    PIC_TYPE_P            = 1,
    PIC_TYPE_INTER        = 1,
    PIC_TYPE_B            = 2,
    PIC_TYPE_REPEAT       = 2,
    PIC_TYPE_AV1_INTRA    = 2,
    PIC_TYPE_VC1_BI       = 2,
```

```
    PIC_TYPE_VC1_B        = 3,
    PIC_TYPE_D            = 3,
    PIC_TYPE_S            = 3,
    PIC_TYPE_AVS2_F       = 3,
    PIC_TYPE_AV1_SWITCH   = 3,
    PIC_TYPE_VC1_P_SKIP   = 4,
    PIC_TYPE_MP4_P_SKIP_NOT_CODED = 4,
    PIC_TYPE_AVS2_S       = 4,
    PIC_TYPE_IDR          = 5,
    PIC_TYPE_AVS2_G       = 5,
    PIC_TYPE_AVS2_GB      = 6,
    PIC_TYPE_MAX
}PicType;
```

Description

This is an enumeration type for representing picture types.

PIC_TYPE_I
I picture

PIC_TYPE_KEY
KEY frame for SVAC and AV1

PIC_TYPE_P
P picture

PIC_TYPE_INTER
Inter frame for SVAC and AV1

PIC_TYPE_B
B picture (except VC1)

PIC_TYPE_REPEAT
Repeat frame (VP9 only)

PIC_TYPE_AV1_INTRA
Intra only frame (AV1 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_AVS2_F
F picture in AVS2

PIC_TYPE_AV1_SWITCH
Switch frame (AV1 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_AVS2_S
S picture in AVS2

PIC_TYPE_IDR
H.264/H.265 IDR picture

PIC_TYPE_AVS2_G
G picture in AVS2

PIC_TYPE_AVS2_GB
GB picture in AVS2

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_512,
    PRODUCT_ID_515,
    PRODUCT_ID_521,
    PRODUCT_ID_511,
    PRODUCT_ID_517,
    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,
    ARM_COMPRESSED_FRAME_MAP                    = 10,  // AFBC enabled WAVE decoder
    COMPRESSED_FRAME_MAP_V50_LOSSLESS_8BIT      = 11,
    COMPRESSED_FRAME_MAP_V50_LOSSLESS_10BIT     = 12,
    COMPRESSED_FRAME_MAP_V50_LOSSY              = 13,
    COMPRESSED_FRAME_MAP_V50_LOSSLESS_422_8BIT  = 14,
    COMPRESSED_FRAME_MAP_V50_LOSSLESS_422_10BIT = 15,
    COMPRESSED_FRAME_MAP_V50_LOSSY_422          = 16,
    COMPRESSED_FRAME_MAP                        = 17,  // WAVE4 only
    COMPRESSED_FRAME_MAP_SVAC_SVC_BL            = 18,
    COMPRESSED_FRAME_MAP_DUAL_CORE_8BIT         = 19,
    COMPRESSED_FRAME_MAP_DUAL_CORE_10BIT        = 20,
    PVRIC_COMPRESSED_FRAME_LOSSLESS_MAP         = 21,
    PVRIC_COMPRESSED_FRAME_LOSSY_MAP            = 22,
    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)

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

COMPRESSED_FRAME_MAP_V50_LOSSLESS_8BIT
CFRAME50(Chips&Media Frame Buffer Compression) compressed framebuffer type

COMPRESSED_FRAME_MAP_V50_LOSSLESS_10BIT
CFRAME50(Chips&Media Frame Buffer Compression) compressed framebuffer type

COMPRESSED_FRAME_MAP_V50_LOSSY
CFRAME50(Chips&Media Frame Buffer Compression) compressed framebuffer type

COMPRESSED_FRAME_MAP_V50_LOSSLESS_422_8BIT
CFRAME50(Chips&Media Frame Buffer Compression) compressed 4:2:2 framebuffer type

COMPRESSED_FRAME_MAP_V50_LOSSLESS_422_10BIT
CFRAME50(Chips&Media Frame Buffer Compression) compressed 4:2:2 framebuffer type

COMPRESSED_FRAME_MAP_V50_LOSSY_422
CFRAME50(Chips&Media Frame Buffer Compression) compressed 4:2:2 framebuffer type

COMPRESSED_FRAME_MAP
Compressed frame map type (WAVE only)

COMPRESSED_FRAME_MAP_SVAC_SVC_BL
Compressed frame map type for base layer in SVAC encoder

COMPRESSED_FRAME_MAP_DUAL_CORE_8BIT
8bit compressed frame map type for dual core

COMPRESSED_FRAME_MAP_DUAL_CORE_10BIT
10bit compressed frame map type for dual core

PVRIC_COMPRESSED_FRAME_LOSSLESS_MAP
PVRIC compressed frame map type for lossless mode

PVRIC_COMPRESSED_FRAME_LOSSY_MAP
PVRIC compressed frame map type for lossy mode

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

Wave5ChangeParam

```
typedef enum {
    // COMMON parameters which can be changed frame by frame.
    ENC_SET_CHANGE_PARAM_PPS                = (1<<0),
    ENC_SET_CHANGE_PARAM_INTRA_PARAM        = (1<<1),
    ENC_SET_CHANGE_PARAM_RC_TARGET_RATE     = (1<<8),
```

```
    ENC_SET_CHANGE_PARAM_RC                 = (1<<9),
    ENC_SET_CHANGE_PARAM_RC_MIN_MAX_QP      = (1<<10),
    ENC_SET_CHANGE_PARAM_RC_BIT_RATIO_LAYER  = (1<<11),
    ENC_SET_CHANGE_PARAM_RC_INTER_MIN_MAX_QP = (1<<12),
    ENC_SET_CHANGE_PARAM_RC_WEIGHT          = (1<<13),
    ENC_SET_CHANGE_PARAM_INDEPEND_SLICE     = (1<<16),
    ENC_SET_CHANGE_PARAM_DEPEND_SLICE       = (1<<17),
    ENC_SET_CHANGE_PARAM_RDO                = (1<<18),
    ENC_SET_CHANGE_PARAM_NR                 = (1<<19),
    ENC_SET_CHANGE_PARAM_BG                 = (1<<20),
    ENC_SET_CHANGE_PARAM_CUSTOM_MD          = (1<<21),
    ENC_SET_CHANGE_PARAM_CUSTOM_LAMBDA      = (1<<22),
    ENC_SET_CHANGE_PARAM_VUI_HRD_PARAM      = (1<<23),
} Wave5ChangeParam;
```

Description

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

Mp4HeaderType

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

Description

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

VOL_HEADER
Video object layer header

VOS_HEADER
Visual object sequence header

VIS_HEADER
Video object header

AvcHeaderType

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

Description

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

SPS_RBSP
Sequence parameter set

PPS_RBSP
Picture parameter set

SPS_RBSP_MVC
Subset sequence parameter set

PPS_RBSP_MVC
Picture parameter set for dependent view

WaveEncHeaderType

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

Description

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

CODEOPT_ENC_VPS
A flag to encode VPS nal unit explicitly

CODEOPT_ENC_SPS
A flag to encode SPS nal unit explicitly

CODEOPT_ENC_PPS
A flag to encode PPS nal unit explicitly

ENC_PIC_CODE_OPTION

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

Description

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

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

CODEOPT_ENC_VCL
A flag to encode VCL nal unit explicitly

GOP_PRESET_IDX

```
typedef enum {
    PRESET_IDX_CUSTOM_GOP       = 0,
    PRESET_IDX_ALL_I            = 1,
    PRESET_IDX_IPP              = 2,
    PRESET_IDX_IBBB             = 3,
    PRESET_IDX_IBPBP            = 4,
    PRESET_IDX_IBBBP            = 5,
    PRESET_IDX_IPPPP            = 6,
    PRESET_IDX_IBBBB            = 7,
    PRESET_IDX_RA_IB            = 8,
} GOP_PRESET_IDX;
```

Description

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

PRESET_IDX_CUSTOM_GOP
User defined GOP structure

PRESET_IDX_ALL_I
All Intra, gopsize = 1

PRESET_IDX_IPP
Consecutive P, cyclic gopsize = 1

PRESET_IDX_IBBB
Consecutive B, cyclic gopsize = 1

PRESET_IDX_IBPBP
gopsize = 2

PRESET_IDX_IBBBP
gopsize = 4

PRESET_IDX_IPPPP
Consecutive P, cyclic gopsize = 4

PRESET_IDX_IBBBB
Consecutive B, cyclic gopsize = 4

PRESET_IDX_RA_IB
Random Access, cyclic gopsize = 8

2.3. Data Structures

VpuAttr

```
typedef struct {
    Uint32  productId;
    Uint32  productNumber;
    char    productName[8];
    Uint32  productVersion;
    Uint32  fwVersion;
    Uint32  customerId;
    Uint32  supportDecoders;
    Uint32  supportEncoders;
    Uint32  numberOfMemProtectRgns;
    Uint32  maxNumVcores;
    BOOL    supportWTL;
    BOOL    supportTiled2Linear;
    BOOL    supportTiledMap;
    BOOL    supportMapTypes;
    BOOL    supportLinear2Tiled;
    BOOL    support128bitBus;
    BOOL    supportThumbnailMode;
    BOOL    supportBitstreamMode;
    BOOL    supportFBCBWOptimization;
    BOOL    supportGDIHW;
    BOOL    supportCommandQueue;
    BOOL    supportBackbone;
    BOOL    supportNewTimer;
    BOOL    support2AlignScaler;
    BOOL    supportAVC10bitEnc;
    BOOL    supportHEVC10bitEnc;
    Uint32  supportEndianMask;
    Uint32  framebufferCacheType;
    Uint32  bitstreamBufferMargin;
    Uint32  hwConfigDef0;
    Uint32  hwConfigDef1;
    Uint32  hwConfigFeature;
    Uint32  hwConfigDate;
    Uint32  hwConfigRev;
    Uint32  hwConfigType;
    BOOL    supportDualCore;
    BOOL    supportVcoreBackbone;
} VpuAttr;
```

Description

This is a data structure of hardware attributes for VPU.

productId
The product ID

productNumber
The product number. ex) 512

productName
The product name in ascii code

productVersion
The product version number

fwVersion
The F/W version

customerId
Customer ID number

supportDecoders
bitmask: See *the section called “CodStd”*

supportEncoders
bitmask: See *the section called "CodStd"*

numberOfMemProtectRgns
It is always 10.

maxNumVcores
The number of core in the VPU

supportWTL
This indicates whether a product supports the WTL or not

supportTiled2Linear
This indicates whether a product supports the Tile2Linear or not

supportTiledMap
This indicates whether a product supports the TiledMap or not

supportMapTypes
This indicates whether a product supports map types.

supportLinear2Tiled
This indicates whether a product supports the Linear2Tiled. - Encoder feature

support128bitBus
This indicates whether a product supports 128bit bus.

supportThumbnailMode
This indicates whether a product supports the thumbnail mode.

supportBitstreamMode
This indicates whether a product supports bitstream modes such as pic-end mode and ring-buffer mode.

supportFBCBWOptimization
This indicates whether a product supports the FBC optimization.

supportGDIHW
This indicates whether a product supports the GDI.

supportCommandQueue
This indicates whether a product supports the Command Queue.

supportBackbone
This indicates whether a product supports the Backbone. Higher version of the GDI.

supportNewTimer
0: timeeScale=32768, 1: timerScale=256 (tick = cycle/timerScale)

support2AlignScaler
This indicates whether a product supports the 2-align scaler.

supportAVC10bitEnc
This indicates whether a product supports AVC 10bit encoder

supportHEVC10bitEnc
This indicates whether a product supports HEVC 10bit encoder

supportEndianMask
A variable of supported endian mode in product

framebufferCacheType
0: None, 1: Maverick-I, 2: Maverick-II

bitstreamBufferMargin
A variable to leave a margin in bistream buffer for GBU operation. This is valid when ringBufferEnable is 1. (CODA9 only)

hwConfigDef0
System configuration information (internal use only) such as external memory interface, external model information, and output support

hwConfigDef1
General hardware configuration information (internal use only)

hwConfigFeature
supported codec standards

hwConfigDate
Configuation Date Decimal, ex)20151130

hwConfigRev
SVN revision, ex) 83521

hwConfigType
Not defined yet

supportDualCore
This indicates whether a product has two vcores.

supportVcoreBackbone
This indicates whether a product supports the vcore backbone version.

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 WAVE 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;
    int tx16y;
    int tx16c;
} DRAMConfig;
```

Description

This is a data structure of DRAM information(CODA960 and BODA950 only) and CFRAME50 configuration(WAVE5 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)

tx16y
This value is used for CFRAME50(Chips&Media Frame Buffer Compression) (WAVE5 only)

tx16c
This value is used for CFRAME50(Chips&Media Frame Buffer Compression) (WAVE5 only)

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;
    TiledMapType 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 : Cb data are written in Cb frame memory and Cr data are written in Cr frame memory. (chroma separate mode)
- 1 : Cb and Cr data are written in the same 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
- 0 : Cb data are written in Cb frame memory and Cr data are written in Cr frame memory. (chroma separate mode)
- 1 : Cb and Cr data are written in the same chroma memory. (chroma interleave mode)

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 picture region to be displayed.

picHeight
This value is used for height of the picture region to be displayed.

picOffsetX
This value is located at the lower-left corner of the picture region to be displayed.

picOffsetY
This value is located at the lower-left corner of the picture region to be displayed.

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 unit of sample

picHeight
Picture height in unit 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;
        } coda9;
        struct {
            // for Decoder
            int useSclEnable;
            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).

useSclEnable
This enables AXI secondary channel for Scaler line buffer.

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;
    BitStreamMode   bitstreamMode;
    Uint32          coreIdx;
    vpu_buffer_t    vbWork;
    Uint32          virtAxiID;
    BOOL            bwOptimization;
    BOOL            tempIdSelectMode;
    int             av1Format;
} 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 non-linear and a linear format of the decoded frame to the frame buffer at the same time. - first in linear map and second in tiled or compressed map. Basically, VPU only writes a non-linear format of frame. If you want to get a linear format of frame for display, WTL should be enabled.

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 : Cb data are written in Cb frame memory and Cr data are written in Cr frame memory. (chroma separate mode)
- 1 : Cb and Cr data are written in the same 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
The endianess of the frame buffer for reference and reconstruction
- 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
The endianess of the bitstream buffer
- 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 a read pointer reaches a 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
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.

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.

tempIdSelectMode

It sets the selection mode of temporal ID.

- 0: use the target temporal_id as absolute value.
  - TARGET_DEC_TEMP_ID = TARGET_DEC_TEMP_ID_PLUS1 -1
  - When using an absolute value, decoder can keep the decoding layer ID.
  - If the SPS_MAX_SUB_LAYER is changed in the bitstream, the temporal skip ratio can be changed.
- 1: use the target temporal_id as relative value
  - TARGET_DEC_TEMP_ID = SPS_MAX_SUB_LAYER - REL_TARGET_TEMP_ID
  - SPS_MAX_SUB_LAYER is signalled from bitstream.
  - When using a relative value, decoder can keep the temporal skip ratio, regardless of the change of SPS_MAX_SUB_LAYER in the bitstream.

av1Format

- 0: AV1_STREAM_IVF AV1 bistream contained by IVF format
- 1: AV1_STREAM_OBU OBU syntax bitstream
- 2: AV1_STREAM_ANNEXB Length delimited bitstream format

Note This variable is only valid when decoding AV1 stream.

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;
    Uint32           seqInitErrReason;
    Int32           warnInfo;
    PhysicalAddress rdPtr;
    PhysicalAddress wrPtr;
```

```
    AvcVuiInfo      avcVuiInfo;
    MP2BarDataInfo  mp2BardataInfo;
    Uint32          sequenceNo;
    Int32           spatialSvcEnable;
    Int32           spatialSvcMode;
    Uint32          outputBitDepth;
    Uint32          vlcBufSize;
    Uint32          paramBufSize;
} DecInitialInfo;
```

Description

This is a 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 temporal sub-layers.

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 : Spatially Scalable
  - 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
- AVS2
  - 8'b0001 0010 : Main picture profile
  - 8'b0010 0000 : Main profile
  - 8'b0010 0010 : Main10 profile
- VC1 : level in struct B
- VP8 : Profile 0 - 3
- VP9 : Profile 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

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

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.

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.

maxNumRefFrm

num_ref_frames syntax value of SPS (CODA9 H.264 decoder only)

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*.

warnInfo

This is warning information 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.

spatialSvcEnable
This is an spatial SVC flag.
- 0 : Non-SVC stream
- 1 : SVC stream

spatialSvcMode
This is an spatial SVC mode.
- 0 : disable inter-layer prediction (simulcast).
- 1 : enable inter-layer prediction in which EL picture is predicted with motion vector information from the base layer picture.

outputBitDepth
This is an output bit-depth. This is only for AVS2
- 8 : output bit-depth is 8.
- 10 : output bit-depth is 10.

vlcBufSize
The size of task buffer

paramBufSize
The size of task buffer

DecParam

```
typedef struct {
    Int32 iframeSearchEnable;
    Int32 skipframeMode;
    Int32 selSvacLayer;
    union {
        Int32 mp2PicFlush;
        Int32 rvDbkMode;
    } DecStdParam;

    BOOL    craAsBlaFlag;
    Uint32  qosEnable;
    Uint32  qosConfig;
    Uint32  qosValVcore0;
    Uint32  qosValVcore1;
} DecParam;
```

Description

The data structure for options of picture decoding.

iframeSearchEnable

- 0 : disable
- 1 : enable
- 2 : I frame search enable (H.264/AVC only)

If this option is enabled, then decoder performs skipping frame decoding until decoder meets an I (IDR) frame. If there is no I frame in given stream, decoder waits for I (IDR) frame. Especially in H.264/AVC stream decoding, they might have I-frame and IDR frame. Therefore HOST application should set iframeSearchEnable value according to frame type. If HOST application wants to search IDR frame, this flag should be set to 1 like other standards. Otherwise if HOST application wants to search I frame, this flag should be set to 2.

Note that when decoder meets EOS (End Of Sequence) code during I-Search, decoder returns -1 (0xFFFF). And if this option is enabled, skipframeMode options are ignored.

Note CODA9 only supports it.

skipframeMode

Skip frame function enable and operation mode

In case of CODA9,

- 0 : skip frame disable
- 1 : skip frames except I (IDR) frames
- 2 : skip non-reference frames

After the skip, decoder returns -2 (0xFFFE) of the decoded index when decoder does not have any frames displayed.

In case of WAVE5,

- 0x0 : skip frame off
- 0x1 : skip non-RAP pictures (skip any picture which is neither IDR, CRA, nor BLA).
- 0x2 : skip non-reference pictures
- 0x3 : reserved
- 0x4~0xf : reserved

Note When decoder meets EOS (End Of Sequence) code during frame skip, decoder returns -1(0xFFFF).

selSvacLayer

Selects which layer of SVAC spatial SVC stream is decoded.

- 0: decodes both BL(Base Layer) and EL(Enhanced Layer) picture.
- 1: decodes BL picture.
- 2: decodes EL picture.

mp2PicFlush

Forces to flush a display index of the frame buffer that delayed without decoding of the current picture.

- 0 : disable
- 1 : enable flushing

rvDbkMode

Sets a de-blocking filter mode for RV streams.

- 0 : enable de-blocking filter for all pictures.
- 1 : disable de-blocking filter for all pictures.
- 2 : disable de-blocking filter for P and B pictures.
- 3 : disable de-blocking filter only for B pictures.

craAsBlaFlag
It handles CRA picture as BLA picture not to use reference from the previous decoded pictures (H.265/HEVC only)

qosEnable
It updates QoS control information. To update QoS values, qosEnable must be 1.

qosConfig
It enables register based QoS control. If register based QoS control feature is enabled, QoS for AXI is set by given qosValVcore0 and qosValVcore1 respectively.

qosValVcore0
Qos value for VCORE0

qosValVcore1
Qos value for VCORE1

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_RECOVERY_POINT       (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 *the section called "DecOutputInfo"* structure.

viewIdxDecoded
This is view index order of decoded frame buffer corresponding to indexFrameDecoded of *the section called "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
It indicates cpb_cnt_minus1 syntax.

vclHrdParamFlag
It indicates vcl_hrd_parameters_present_flag syntax.

nalHrdParamFlag
It indicates nal_hrd_parameters_present_flag syntax.

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
It indicates recovery_frame_cnt syntax.

exactMatchFlag
It indicates exact_match_flag syntax.

brokenLinkFlag
It indicates broken_link_flag syntax.

changingSliceGroupIdc
It indicates changing_slice_group_idc syntax.

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

SvacInfo

```
typedef struct {
    int spatialSvcMode;
    int spatialSvcLayer;
    int spatialSvcFlag;
} SvacInfo;
```

Description

This is a data structure of spatial scalable video coding information such as SVC layer and SVC mode (SVAC only)

spatialSvcMode
This is the spatial SVC mode.
- 0 : disable inter-layer prediction. (simulcast)
- 1 : enable inter-layer prediction in which EL picture is predicted with motion vector information from the base layer picture.

spatialSvcLayer
This is the layer information of SVC stream.
- 0 : base layer picture
- 1 : enhance layer picture

spatialSvcFlag
An spatial SVC flag

Avs2Info

```
typedef struct {
    int decodedPOI;
    int displayPOI;
} Avs2Info;
```

Description

This is a data structure for AVS2 specific picture information.

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

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

Av1Info

```
typedef struct {
    int enableScreenContents;
    int enableIntraBlockCopy;
} Av1Info;
```

Description

This is a data structure for AV1 specific picture information.

enableScreenContents
It indicates whether screen content tool is enabled.

enableIntraBlockCopy
It indicates whether intra block copy is enabled.

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;
    Uint32 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;
    SvacInfo svacInfo;
    Avs2Info avs2Info;
    Av1Info  av1Info;
    int decodedPOC;
    int displayPOC;
    int temporalId;
    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;
    Uint32 errorReasonExt;
    int warnInfo;
    Uint32 sequenceNo;
    int rvTr;
    int rvTrB;
    int indexFramePrescan;
    Uint32  decHostCmdTick;
    Uint32  decSeekStartTick;
    Uint32  decSeekEndTick;
    Uint32  decParseStartTick;
    Uint32  decParseEndTick;
    Uint32  decDecodeStartTick;
    Uint32  decDecodeEndTick;
    Uint32  seekCycle;
    Uint32  parseCycle;
    Uint32  DecodedCycle;
    Int32 ctuSize;
    Int32 outputFlag;
} DecOutputInfo;
```

Description

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

indexFrameDisplay

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

- -3(0xFFFD) or -2(0xFFFE) : it is when a display output cannot be given due to picture reordering or skip option.
- -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 total number of error coded unit(MB/CTU) in the decoded picture.

numOfTotMBs
This is the total number of coded unit(MB/CTU) in the decoded picture.

numOfErrMBsInDisplay
This is the total number of error coded unit(MB/CTU) in the display picture of indexFrameDisplay.

numOfTotMBsInDisplay
This is the total number of coded unit(MB/CTU) in the display picture of indexFrameDisplay. In normal stream, numOfTotMBs and numOfTotMBsInDisplay always have the same value. However, in case of sequence change stream(resolution change), they might be different.

refMissingFrameFlag
This indicates that the current frame's references are missing in decoding.

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 : progressive frame which consists of one picture
- 1 : 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: 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"*.

svacInfo
This field reports the information of SVC stream. Refer to *the section called "SvacInfo"*.

avs2Info
This is AVS2 picture information. Refer to *the section called "Avs2Info"*.

av1Info
This is AVS1 picture information. Refer to *the section called "Av1Info"*.

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

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 : top field
  - 2 : bottom field
  - 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 number of cycles for processing a 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)

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

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. (WAVE 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.

decHostCmdTick
Tick of DEC_PIC command for the picture

decSeekStartTick
Start tick of seeking slices of the picture

decSeekEndTick
End tick of seeking slices of the picture

decParseStartTick
Start tick of parsing slices of the picture

decParseEndTick
End tick of parsing slices of the picture

decDecodeStartTick
Start tick of decoding slices of the picture

decDecodeEndTick
End tick of decoding slices of the picture

seekCycle
The number of cycles for seeking slices of a picture in the internal pipeline. (Theoretically, framecycle = seekCycle + parseCycle + decodedCycle)

parseCycle
The number of cycles for parsing slices of a picture in the internal pipeline. (Theoretically, framecycle = seekCycle + parseCycle + decodedCycle)

DecodedCycle
The number of cycles for decoding slices of a picture in the internal pipeline. (Theoretically, framecycle = seekCycle + parseCycle + decodedCycle)

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

outputFlag
This variable reports the value of pic_output_flag syntax in HEVC slice_segment_header. (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];
   vpu_buffer_t  vbTask;
```

```
    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 co-located motion vector buffer

vbTask
The information of task buffer used for command queue

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].

QueueStatusInfo

```
typedef struct {
    Uint32  instanceQueueCount;
    Uint32  reportQueueCount;
} QueueStatusInfo;
```

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.

reportQueueCount
This variable indicates the number of report queued commands of all instances.

EncMp4Param

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

Description

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

mp4DataPartitionEnable
It encodes with MPEG4 data_partitioned coding tool.

mp4ReversibleVlcEnable
It encodes with MPEG4 reversible_vlc coding tool.

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

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

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

EncH263Param

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

Description

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

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

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

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

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

CustomGopPicParam

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

Description

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

picType
A picture type of Nth picture in the custom GOP

pocOffset
A POC of Nth picture in the custom GOP

picQp
A quantization parameter of Nth picture in the custom GOP

numRefPicL0
The number of reference L0 of Nth picture in the custom GOP

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

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

temporalId
A temporal ID of Nth picture in the custom GOP

CustomGopParam

```
typedef struct {
    int customGopSize;
    CustomGopPicParam picParam[MAX_GOP_NUM];
} CustomGopParam;
```

Description

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

customGopSize
The size of custom GOP (0~8)

picParam
Picture parameters of Nth picture in custom GOP

WaveCustomMapOpt

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

Description

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

roiAvgQp
It sets an average QP of ROI map.

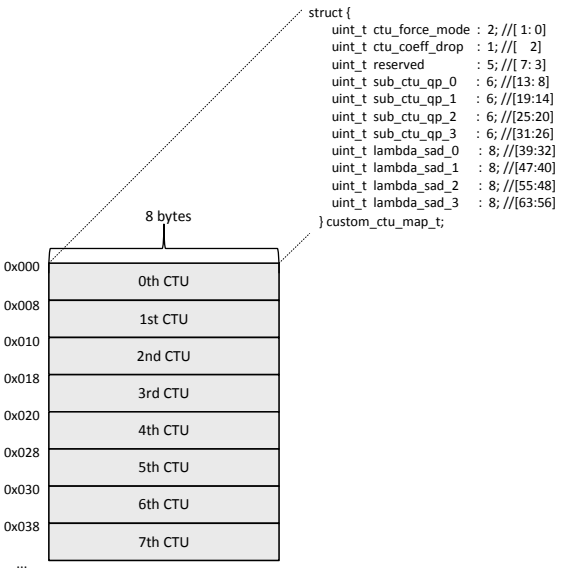
customRoiMapEnable
It enables ROI map.

customLambdaMapEnable
It enables custom lambda map.

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

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

addrCustomMap
It indicates the start buffer address of custom map. Each custom CTU map takes 8 bytes and holds mode, coefficient drop flag, QPs, and lambdas like the below illustration.



HevcVuiParam

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

Description

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

vuiParamFlags
It specifies vui_parameters_present_flag.

vuiAspectRatioIdc
It specifies aspect_ratio_idc.

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

vuiOverScanAppropriate
It specifies overscan_appropriate_flag.

videoSignal
It specifies video_signal_type_present_flag.

vuiChromaSampleLoc
It specifies chroma_sample_loc_type_top_field and chroma_sample_loc_type_bottom_field.

vuiDispWinLeftRight
It specifies def_disp_win_left_offset and def_disp_win_right_offset.

vuiDispWinTopBottom
It specifies def_disp_win_top_offset and def_disp_win_bottom_offset.

EncWaveParam

```
typedef struct {
    int profile;
    int enStillPicture;
    int level;
    int tier;
    int internalBitDepth;
    int losslessEnable;
    int constIntraPredFlag;
    int gopPresetIdx;
    int decodingRefreshType;
    int intraQP;
    int intraPeriod;
    int forcedIdrHeaderEnable;
    int confWinTop;
    int confWinBot;
    int confWinLeft;
    int confWinRight;
    int indepandSliceMode;
    int indepandSliceModeArg;
    int dependSliceMode;
    int dependSliceModeArg;
    int intraRefreshMode;
    int intraRefreshArg;
    int useRecommendEncParam;
    int scalingListEnable;
    int cuSizeMode;
    int tmvpEnable;
    int wppEnable;
    int maxNumMerge;
    int disableDeblk;
    int lfCrossSliceBoundaryEnable;
    int betaOffsetDiv2;
    int tcOffsetDiv2;
    int skipIntraTrans;
    int saoEnable;
    int intraNxNEnable;
    int bitAllocMode;
    int fixedBitRatio[MAX_GOP_NUM];
    int cuLevelRCEnable;
    int hvsQPEnable;
    int hvsQpScale;
    int hvsMaxDeltaQp;
    // CUSTOM_GOP
    CustomGopParam gopParam;
    int roiEnable;
    Uint32 numUnitsInTick;
    Uint32 timeScale;
    Uint32 numTicksPocDiffOne;
    int chromaCbQpOffset;
    int chromaCrQpOffset;
    int initialRcQp;
    Uint32  nrYEnable;
    Uint32  nrCbEnable;
    Uint32  nrCrEnable;
    // ENC_NR_WEIGHT
    Uint32  nrIntraWeightY;
    Uint32  nrIntraWeightCb;
    Uint32  nrIntraWeightCr;
    Uint32  nrInterWeightY;
    Uint32  nrInterWeightCb;
    Uint32  nrInterWeightCr;
    Uint32  nrNoiseEstEnable;
```

```
    Uint32  nrNoiseSigmaY;
    Uint32  nrNoiseSigmaCb;
    Uint32  nrNoiseSigmaCr;
    Uint32 useLongTerm;
    // newly added for WAVE Encoder
    Uint32 monochromeEnable;
    Uint32 strongIntraSmoothEnable;
    Uint32 weightPredEnable;
    Uint32 bgDetectEnable;
    Uint32 bgThrDiff;
    Uint32 bgThrMeanDiff;
    Uint32 bgLambdaQp;
    int    bgDeltaQp;
    Uint32 customLambdaEnable;
    Uint32 customMDEnable;
    int    pu04DeltaRate;
    int    pu08DeltaRate;
    int    pu16DeltaRate;
    int    pu32DeltaRate;
    int    pu04IntraPlanarDeltaRate;
    int    pu04IntraDcDeltaRate;
    int    pu04IntraAngleDeltaRate;
    int    pu08IntraPlanarDeltaRate;
    int    pu08IntraDcDeltaRate;
    int    pu08IntraAngleDeltaRate;
    int    pu16IntraPlanarDeltaRate;
    int    pu16IntraDcDeltaRate;
    int    pu16IntraAngleDeltaRate;
    int    pu32IntraPlanarDeltaRate;
    int    pu32IntraDcDeltaRate;
    int    pu32IntraAngleDeltaRate;
    int    cu08IntraDeltaRate;
    int    cu08InterDeltaRate;
    int    cu08MergeDeltaRate;
    int    cu16IntraDeltaRate;
    int    cu16InterDeltaRate;
    int    cu16MergeDeltaRate;
    int    cu32IntraDeltaRate;
    int    cu32InterDeltaRate;
    int    cu32MergeDeltaRate;
    int    coefClearDisable;
    int    minQpI;
    int    maxQpI;
    int    minQpP;
    int    maxQpP;
    int    minQpB;
    int    maxQpB;
    PhysicalAddress customLambdaAddr;
    PhysicalAddress userScalingListAddr;
    // SVAC encoder only
    int svcEnable;
    int svcMode;
    int lumaDcQpOffset;
    int chromaDcQpOffset;
    int chromaAcQpOffset;
    // for H.264 on WAVE
    int avcIdrPeriod;
    int rdoSkip;
    int lambdaScalingEnable;
    int transform8x8Enable;
    int avcSliceMode;
    int avcSliceArg;
    int intraMbRefreshMode;
    int intraMbRefreshArg;
    int mbLevelRcEnable;
    int entropyCodingMode;

    int s2fmeDisable;
    Uint32 rcWeightParam;
    Uint32 rcWeightBuf;
    HevcVuiParam vuiParam;
#ifdef SUPPORT_LOOK_AHEAD_RC
    int larcEnable;
    int larcPass;
    int larcSize;
    int larcWeight;
#endif
}EncWaveParam;
```

Description

This is a data structure of HEVC/AVC encoder parameters for WAVE5.

profile
A profile indicator (HEVC only)
- 0 : The firmware determines a profile according to internalbitdepth.
- 1 : Main profile
- 2 : Main10 profile
- 3 : Main still picture profile

Note In AVC encoder, a profile cannot be set by host application. The firmware decides it based on internalbitdepth. It is HIGH profile for bitdepth of 8 and HIGH10 profile for bitdepth of 10.

enStillPicture
Still picture profile

level
A level indicator (level * 10)

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

internalBitDepth
An internal bit-depth which is used for actual encoding

For example, if you set internalBitDepth as 8 for 10bit source picture, VPU encodes the 10bit source picture into 8bit picture stream. If nothing is given to internalBitDepth, VPU encodes source YUV based on srcBitDepth.

losslessEnable
It enables lossless coding.

constIntraPredFlag
It enables constrained intra prediction.

gopPresetIdx
A GOP structure preset option
- 0 : custom GOP
- Other values : *the section called “GOP_PRESET_IDX”*

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

intraQP
A quantization parameter of intra picture

intraPeriod
A period of intra picture in GOP size

forcedIdrHeaderEnable
t enables every IDR frame to include VPS/SPS/PPS

- 0 : No froced Header(VPS/SPS/PPS)
- 1 : Forced Header before IDR frame
- 2 : Forced Header before key frame

confWinTop
A top offset of conformance window

confWinBot
A bottom offset of conformance window

confWinLeft
A left offset of conformance window

confWinRight
A right offset of conformance window

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

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

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

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

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

intraRefreshArg
It Specifies an intra CTU refresh interval. Depending on intraRefreshMode, it can mean one of the followings.
- The number of consecutive CTU rows for IntraCtuRefreshMode of 1
- The number of consecutive CTU columns for IntraCtuRefreshMode of 2
- A step size in CTU for IntraCtuRefreshMode of 3
- The number of Intra CTUs to be encoded in a picture for IntraCtuRefreshMode of 4

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

scalingListEnable
It enables a scaling list.

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

tmvpEnable
It enables temporal motion vector prediction.

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

maxNumMerge
It specifies the number of merge candidates in RDO (1 or 2). 2 of maxNumMerge (default) offers better quality of encoded picture, while 1 of maxNumMerge improves encoding performance.

disableDeblk
It disables in-loop deblocking filtering.

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

betaOffsetDiv2
It sets BetaOffsetDiv2 for deblocking filter.

tcOffsetDiv2
It sets TcOffsetDiv3 for deblocking filter.

skipIntraTrans
It enables the transform stage to be skipped for an intra CU. This value is set to transform_skip_enabled_flag in PPS.

saoEnable
It enables SAO (Sample Adaptive Offset).

intraNxNEnable
It enables intra NxN PUs.

bitAllocMode
It specifies picture bits allocation mode. It is only valid when RateControl is enabled and GOP size is larger than 1.
- 0 : More referenced pictures have more bits than less referenced pictures do.
- 1 : All pictures in GOP have similar amount of bits.
- 2 : Each picture in GOP is allocated a portion (fixedBitRatio) of total bit budget.

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

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

cuLevelRCEnable
It enable CU level rate control.

hvsQPEnable
It enable CU QP adjustment for subjective quality enhancement.

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

hvsMaxDeltaQp
A maximum delta QP for HVS

gopParam
the section called "CustomGopParam"

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

numUnitsInTick
It specifies the number of time units of a clock operating at the frequency time_scale Hz. This is used to to calculate frameRate syntax.

timeScale
It specifies the number of time units that pass in one second. This is used to to calculate frameRate syntax.

numTicksPocDiffOne
It specifies the number of clock ticks corresponding to a difference of picture order count values equal to 1. This is used to calculate frameRate syntax.

chromaCbQpOffset
The value of chroma(Cb) QP offset

chromaCrQpOffset
The value of chroma(Cr) QP offset

initialRcQp
The value of initial QP by HOST application. This value is meaningless if INITIAL_RC_QP is 63.

nrYEnable
It enables noise reduction algorithm to Y component.

nrCbEnable
It enables noise reduction algorithm to Cb component.

nrCrEnable
It enables noise reduction algorithm to Cr component.

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

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

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

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

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

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

nrNoiseEstEnable
It enables noise estimation for noise reduction. When this is disabled, host carries out noise estimation with nrNoiseSigmaY/Cb/Cr.

nrNoiseSigmaY
It specifies Y noise standard deviation when nrNoiseEstEnable is 0.

nrNoiseSigmaCb
It specifies Cb noise standard deviation when nrNoiseEstEnable is 0.

nrNoiseSigmaCr
It specifies Cr noise standard deviation when nrNoiseEstEnable is 0.

useLongTerm
It enables long-term reference function.

monochromeEnable
It enables monochrom encoding mode.

strongIntraSmoothEnable
It enables strong intra smoothing.

weightPredEnable
It enables to use weighted prediction.

bgDetectEnable
It enables background detection.

bgThrDiff
It specifies the threshold of max difference that is used in s2me block. It is valid when background detection is on.

bgThrMeanDiff
It specifies the threshold of mean difference that is used in s2me block. It is valid when background detection is on.

bgLambdaQp
It specifies the minimum lambda QP value to be used in the background area.

bgDeltaQp
It specifies the difference between the lambda QP value of background and the lambda QP value of foreground.

customLambdaEnable
It enables custom lambda table.

customMDEnable
It enables custom mode decision.

pu04DeltaRate
A value which is added to the total cost of 4x4 blocks

pu08DeltaRate
A value which is added to the total cost of 8x8 blocks

pu16DeltaRate
A value which is added to the total cost of 16x16 blocks

pu32DeltaRate
A value which is added to the total cost of 32x32 blocks

pu04IntraPlanarDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 4x4 Planar intra prediction mode.

pu04IntraDcDeltaRate
A value which is added to rate when calculating cost (=distortion + rate) in 4x4 DC intra prediction mode.

pu04IntraAngleDeltaRate
A value which is added to rate when calculating cost (=distortion + rate) in 4x4 Angular intra prediction mode.

pu08IntraPlanarDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 8x8 Planar intra prediction mode.

pu08IntraDcDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 8x8 DC intra prediction mode.

pu08IntraAngleDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 8x8 Angular intra prediction mode.

pu16IntraPlanarDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 16x16 Planar intra prediction mode.

pu16IntraDcDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 16x16 DC intra prediction mode

pu16IntraAngleDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 16x16 Angular intra prediction mode

pu32IntraPlanarDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 32x32 Planar intra prediction mode

pu32IntraDcDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 32x32 DC intra prediction mode

pu32IntraAngleDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 32x32 Angular intra prediction mode

cu08IntraDeltaRate
A value which is added to rate when calculating cost for intra CU8x8

cu08InterDeltaRate
A value which is added to rate when calculating cost for inter CU8x8

cu08MergeDeltaRate
A value which is added to rate when calculating cost for merge CU8x8

cu16IntraDeltaRate
A value which is added to rate when calculating cost for intra CU16x16

cu16InterDeltaRate
A value which is added to rate when calculating cost for inter CU16x16

cu16MergeDeltaRate
A value which is added to rate when calculating cost for merge CU16x16

cu32IntraDeltaRate
A value which is added to rate when calculating cost for intra CU32x32

cu32InterDeltaRate
A value which is added to rate when calculating cost for inter CU32x32

cu32MergeDeltaRate
A value which is added to rate when calculating cost for merge CU32x32

coefClearDisable
It disables the transform coefficient clearing algorithm for P or B picture. If this is 1, all-zero coefficient block is not evaluated in RDO.

minQpI
A minimum QP of I picture for rate control

maxQpI
A maximum QP of I picture for rate control

minQpP
A minimum QP of P picture for rate control

maxQpP
A maximum QP of P picture for rate control

minQpB
A minimum QP of B picture for rate control

maxQpB
A maximum QP of B picture for rate control

customLambdaAddr
It specifies the address of custom lambda map.

userScalingListAddr
It specifies the address of user scaling list file.

svcEnable
It enables to encode with SVC spatial (picture size) scalability. (SVAC encoder only)

svcMode
It specifies an SVC mode. (SVAC encoder only)

lumaDcQpOffset
A delta quantization parameter for luma DC coefficients (SVAC encoder only)

chromaDcQpOffset
A delta quantization parameter for chroma DC coefficients (SVAC encoder only)

chromaAcQpOffset
A delta quantization parameter for chroma AC coefficients (SVAC encoder only)

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

rdoSkip
It skips RDO(rate distortion optimization).

lambdaScalingEnable
It enables lambda scaling using custom GOP.

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

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

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

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

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

mbLevelRcEnable
It enables MB-level rate control.

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

0 : CAVLC 1 : CABAC

s2fmeDisable
It disables s2me_fme (only for AVC encoder).

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

rcWeightBuf

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

vuiParam

the section called "HevcVuiParam"

EncChangeParam

```
typedef struct {
    int enable_option;
    // ENC_SET_CHANGE_PARAM_PPS (lossless, WPP can't be changed while encoding)
    int constIntraPredFlag;
    int lfCrossSliceBoundaryEnable;
    int weightPredEnable;
    int disableDeblk;
    int betaOffsetDiv2;
    int tcOffsetDiv2;
    int chromaCbQpOffset;
    int chromaCrQpOffset;
    int lumaDcQpOffset;
    int chromaDcQpOffset;
    int chromaAcQpOffset;
    int transform8x8Enable;
    int entropyCodingMode;
    // ENC_SET_CHANGE_PARAM_INDEPEND_SLICE
    int independSliceMode;
    int independSliceModeArg;
    // ENC_SET_CHANGE_PARAM_DEPEND_SLICE
    int dependSliceMode;
    int dependSliceModeArg;
    int avcSliceArg;
    int avcSliceMode;
    // ENC_SET_CHANGE_PARAM_RDO (cuSizeMode, MonoChrom, and RecommendEncParam
    //  can't be changed while encoding)
    int coefClearDisable;
    int intraNxNEnable;
    int maxNumMerge;
    int customLambdaEnable;
    int customMDEnable;
    int rdoSkip;
    int lambdaScalingEnable;
    // ENC_SET_CHANGE_PARAM_RC_TARGET_RATE
    int bitRate;
    // ENC_SET_CHANGE_PARAM_RC
    // (rcEnable, cuLevelRc, bitAllocMode, RoiEnable, RcInitQp can't be changed while encoding)
    int hvsQPEnable;
    int hvsQpScale;
    int vbvBufferSize;
    int mbLevelRcEnable;
    // ENC_SET_CHANGE_PARAM_RC_MIN_MAX_QP
    int minQpI;
    int maxQpI;
    int hvsMaxDeltaQp;
    // ENC_SET_CHANGE_PARAM_RC_INTER_MIN_MAX_QP
    int minQpP;
    int minQpB;
    int maxQpP;
    int maxQpB;
    // ENC_SET_CHANGE_PARAM_RC_BIT_RATIO_LAYER
    int fixedBitRatio[MAX_GOP_NUM];

    // ENC_SET_CHANGE_PARAM_BG (bgDetectEnable can't be changed while encoding)
    int    s2fmeDisable;
    Uint32 bgThrDiff;
    Uint32 bgThrMeanDiff;
    Uint32 bgLambdaQp;
    int    bgDeltaQp;
    // ENC_SET_CHANGE_PARAM_NR
    Uint32  nrYEnable;
    Uint32  nrCbEnable;
    Uint32  nrCrEnable;
    Uint32  nrNoiseEstEnable;
    Uint32  nrNoiseSigmaY;
```

```
    Uint32  nrNoiseSigmaCb;
    Uint32  nrNoiseSigmaCr;
    Uint32  nrIntraWeightY;
    Uint32  nrIntraWeightCb;
    Uint32  nrIntraWeightCr;
    Uint32  nrInterWeightY;
    Uint32  nrInterWeightCb;
    Uint32  nrInterWeightCr;
    // ENC_SET_CHANGE_PARAM_CUSTOM_MD
    int    pu04DeltaRate;
    int    pu08DeltaRate;
    int    pu16DeltaRate;
    int    pu32DeltaRate;
    int    pu04IntraPlanarDeltaRate;
    int    pu04IntraDcDeltaRate;
    int    pu04IntraAngleDeltaRate;
    int    pu08IntraPlanarDeltaRate;
    int    pu08IntraDcDeltaRate;
    int    pu08IntraAngleDeltaRate;
    int    pu16IntraPlanarDeltaRate;
    int    pu16IntraDcDeltaRate;
    int    pu16IntraAngleDeltaRate;
    int    pu32IntraPlanarDeltaRate;
    int    pu32IntraDcDeltaRate;
    int    pu32IntraAngleDeltaRate;
    int    cu08IntraDeltaRate;
    int    cu08InterDeltaRate;
    int    cu08MergeDeltaRate;
    int    cu16IntraDeltaRate;
    int    cu16InterDeltaRate;
    int    cu16MergeDeltaRate;
    int    cu32IntraDeltaRate;
    int    cu32InterDeltaRate;
    int    cu32MergeDeltaRate;
    // ENC_SET_CHANGE_PARAM_CUSTOM_LAMBDA
    PhysicalAddress customLambdaAddr;
    // ENC_SET_CHANGE_PARAM_INTRA_PARAM
    int intraQP;
    int intraPeriod;
    int avcIdrPeriod;
    int forcedIdrHeaderEnable;

    // ENC_SET_CHANGE_PARAM_RC_WEIGHT
    Uint32 rcWeightBuf;
    Uint32 rcWeightParam;
}EncChangeParam;
```

Description

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

enable_option
A flag to decide which parameter will change, *the section called “Wave5ChangeParam”*

constIntraPredFlag
It enables constrained intra prediction.

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

weightPredEnable
It enables to use weighted prediction.

disableDeblk
It disables in-loop deblocking filtering.

betaOffsetDiv2
It sets BetaOffsetDiv2 for deblocking filter.

tcOffsetDiv2
It sets TcOffsetDiv3 for deblocking filter.

chromaCbQpOffset
The value of chroma(Cb) QP offset

chromaCrQpOffset
The value of chroma(Cr) QP offset

lumaDcQpOffset
The value of DC QP offset for Y component (for SVAC encoder)

chromaDcQpOffset
The value of DC QP offset for Cb/Cr component (for SVAC encoder)

chromaAcQpOffset
The value of AC QP offset for Cb/Cr component (for SVAC encoder)

transform8x8Enable
(for H.264 encoder)

entropyCodingMode
(for H.264 encoder)

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

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

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

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

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

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

coefClearDisable
It disables the transform coefficient clearing algorithm for P or B picture. If this is 1, all-zero coefficient block is not evaluated in RDO.

intraNxNEnable
It enables intra NxN PUs.

maxNumMerge
It specifies the number of merge candidates in RDO (1 or 2). 2 of maxNumMerge (default) offers better quality of encoded picture, while 1 of maxNumMerge improves encoding performance.

customLambdaEnable
It enables custom lambda table.

customMDEnable
It enables custom mode decision.

rdoSkip
It skips RDO(rate distortion optimization) in H.264 encoder.

lambdaScalingEnable
It enables to use custom lambda scaling list.

bitRate
A target bitrate when separateBitrateEnable is 0

hvsQPEnable
It enables CU QP adjustment for subjective quality enhancement.

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

vbvBufferSize
It specifies the size of the VBV buffer in msec (10 ~ 3000). For example, 3000 should be set for 3 seconds. This value is valid when rcEnable is 1. VBV buffer size in bits is EncBitrate * VbvBufferSize / 1000.

mbLevelRcEnable
It enables MB level rate control. (for H.264 encoder)

minQpI
A minimum QP of I picture for rate control

maxQpI
A maximum QP of I picture for rate control

hvsMaxDeltaQp
A maximum delta QP for HVS

minQpP
A minimum QP of P picture for rate control

minQpB
A minimum QP of B picture for rate control

maxQpP
A maximum QP of P picture for rate control

maxQpB
A maximum QP of B picture for rate control

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

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

s2fmeDisable
It disables s2me_fme (only for AVC encoder).

bgThrDiff
It specifies the threshold of max difference that is used in s2me block. It is valid when background detection is on.

bgThrMeanDiff
It specifies the threshold of mean difference that is used in s2me block. It is valid when background detection is on.

bgLambdaQp
It specifies the minimum lambda QP value to be used in the background area.

bgDeltaQp
It specifies the difference between the lambda QP value of background and the lambda QP value of foreground.

nrYEnable
It enables noise reduction algorithm to Y component.

nrCbEnable
It enables noise reduction algorithm to Cb component.

nrCrEnable
It enables noise reduction algorithm to Cr component.

nrNoiseEstEnable
It enables noise estimation for noise reduction. When this is disabled, host carries out noise estimation with nrNoiseSigmaY/Cb/Cr.

nrNoiseSigmaY
It specifies Y noise standard deviation when nrNoiseEstEnable is 0.

nrNoiseSigmaCb
It specifies Cb noise standard deviation when nrNoiseEstEnable is 0.

nrNoiseSigmaCr
It specifies Cr noise standard deviation when nrNoiseEstEnable is 0.

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

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

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

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

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

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

pu04DeltaRate
A value which is added to the total cost of 4x4 blocks

pu08DeltaRate
A value which is added to the total cost of 8x8 blocks

pu16DeltaRate
A value which is added to the total cost of 16x16 blocks

pu32DeltaRate
A value which is added to the total cost of 32x32 blocks

pu04IntraPlanarDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 4x4 Planar intra prediction mode.

pu04IntraDcDeltaRate
A value which is added to rate when calculating cost (=distortion + rate) in 4x4 DC intra prediction mode.

pu04IntraAngleDeltaRate
A value which is added to rate when calculating cost (=distortion + rate) in 4x4 Angular intra prediction mode.

pu08IntraPlanarDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 8x8 Planar intra prediction mode.

pu08IntraDcDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 8x8 DC intra prediction mode.

pu08IntraAngleDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 8x8 Angular intra prediction mode.

pu16IntraPlanarDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 16x16 Planar intra prediction mode.

pu16IntraDcDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 16x16 DC intra prediction mode

pu16IntraAngleDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 16x16 Angular intra prediction mode

pu32IntraPlanarDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 32x32 Planar intra prediction mode

pu32IntraDcDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 32x32 DC intra prediction mode

pu32IntraAngleDeltaRate
A value which is added to rate when calculating cost(=distortion + rate) in 32x32 Angular intra prediction mode

cu08IntraDeltaRate
A value which is added to rate when calculating cost for intra CU8x8

cu08InterDeltaRate
A value which is added to rate when calculating cost for inter CU8x8

cu08MergeDeltaRate
A value which is added to rate when calculating cost for merge CU8x8

cu16IntraDeltaRate
A value which is added to rate when calculating cost for intra CU16x16

cu16InterDeltaRate
A value which is added to rate when calculating cost for intra CU16x16

cu16MergeDeltaRate
A value which is added to rate when calculating cost for intra CU16x16

cu32IntraDeltaRate
A value which is added to rate when calculating cost for intra CU32x32

cu32InterDeltaRate
A value which is added to rate when calculating cost for intra CU32x32

cu32MergeDeltaRate
A value which is added to rate when calculating cost for intra CU32x32

customLambdaAddr
It specifies the address of custom lambda map.

intraQP
A quantization parameter of intra picture

intraPeriod
A period of intra picture in GOP size

avcIdrPeriod
A period of IDR picture (0 ~ 1024). 0 - implies an infinite period(for H.264 encoder)

forcedIdrHeaderEnable
t enables every IDR frame to include VPS/SPS/PPS
- 0 : No froced Header(VPS/SPS/PPS)
- 1 : Forced Header before IDR frame
- 2 : Forced Header before key frame

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

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

AvcPpsParam

```
typedef struct {
    int ppsId;
    int entropyCodingMode;
```

```
    int cabacInitIdc;
    int transform8x8Mode;
} AvcPpsParam;
```

Description

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

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

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

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

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

EncAvcParam

```
typedef struct {
    int constrainedIntraPredFlag;
    int disableDeblk;
    int deblkFilterOffsetAlpha;
    int deblkFilterOffsetBeta;
    int chromaQpOffset;
    int audEnable;
    int frameCroppingFlag;
    int frameCropLeft;
    int frameCropRight;
    int frameCropTop;
    int frameCropBottom;
    int level;
    int profile;
} EncAvcParam;
```

Description

This is a data structure for configuring H.264/AVC-specific parameters in encoder applications. (CODA9 only)

constrainedIntraPredFlag
- 0 : disable
- 1 : enable

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

deblkFilterOffsetAlpha
It sets deblk_filter_offset_alpha (-6 to 6).

deblkFilterOffsetBeta
It sets deblk_filter_offset_beta (-6 to 6).

chromaQpOffset
It sets chroma_qp_offset (-12 to 12).

audEnable
- 0 : disable
- 1 : enable

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

frameCroppingFlag
- 0 : disable
- 1 : enable

If this is 1, VPU generates frame_cropping_flag syntax in the SPS header.

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

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

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

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

level
H.264/AVC level_idc in SPS

profile
- 0 : Baseline profile
- 1 : Main profile
- 2 : High profile

EncSliceMode

```
typedef struct{
    int sliceMode;
    int sliceSizeMode;
    int sliceSize;
} EncSliceMode;
```

Description

This structure is used for declaring an encoder slice mode and its options. It is newly added for more flexible usage of slice mode control in encoder. (CODA9 only)

sliceMode
- 0 : one slice per picture

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

Slice separation

In MPEG4, resync-marker and packet header are inserted between slice boundaries. In short video header (H.263) with Annex K of 0, GOB headers are inserted at every GOB layer start. In short video header (H.263) with Annex K of 1, multiple slices are generated. In AVC, multiple slice layer RBSP is generated.

sliceSizeMode

This parameter means the size of generated slice.

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

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

sliceSize

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

EncOpenParam

```
typedef struct {
    PhysicalAddress bitstreamBuffer;
    Uint32          bitstreamBufferSize;
    CodStd          bitstreamFormat;
    int             ringBufferEnable;
    int             picWidth;
    int             picHeight;
    int             linear2TiledEnable;
    int             linear2TiledMode;
    int             frameRateInfo;
    int             MESearchRangeX;
    int             MESearchRangeY;
    int             rcGopIQpOffsetEn;
    int             rcGopIQpOffset;
    int             MESearchRange;
    int             vbvBufferSize;
    int             frameSkipDisable;

    int             gopSize;
    int idrInterval;
    int             meBlkMode;
    EncSliceMode    sliceMode;
    int             intraRefreshNum;
    int             ConscIntraRefreshEnable;
    int             userQpMax;

    //h.264 only
    int             maxIntraSize;
    int             userMaxDeltaQp;
    int             userMinDeltaQp;
    int             userQpMin;
    int             MEUseZeroPmv;
    int             intraCostWeight;
    //mp4 only
    int             rcIntraQp;
    int             userGamma;
    int             rcIntervalMode;
    int             mbInterval;
    int             bitRate;
    int             bitRateBL;
    int             rcInitDelay;
    int             rcEnable;
```

```
    union {
        EncMp4Param     mp4Param;
        EncH263Param    h263Param;
        EncAvcParam     avcParam;
        EncWaveParam    waveParam;
    } EncStdParam;

    // Maverick-II Cache Configuration
    int             cacheBypass;
    int             cbcrInterleave;
    int             cbcrOrder;
    int             frameEndian;
    int             streamEndian;
    int             sourceEndian;
    int             bwbEnable;
    int               lineBufIntEn;
    int               packedFormat;
    FrameBufferFormat srcFormat;
    FrameBufferFormat outputFormat;
    int               srcBitDepth;
    Uint32          coreIdx;
    int             nv21;

    Uint32          virtAxiID;
    BOOL            enablePTS;
    int             lowLatencyMode;
    BOOL            enableNonRefFbcWrite;
    int             picWidthBL;
    int             picHeightBL;
    int             sourceBufCount;
    int             streamBufCount;
    int             streamBufSize;
    int             rcWeightFactor;
#ifdef SUPPORT_SOURCE_RELEASE_INTERRUPT
    int             srcReleaseIntEnable;
#endif
    int             ringBufferWrapEnable;
} EncOpenParam;
```

Description

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

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

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

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

ringBufferEnable
- 0 : line-buffer mode
- 1 : ring-buffer mode

This flag sets the streaming mode for the current encoder instance. There are two streaming modes: packet-based streaming with ring-buffer (buffer-reset mode) and frame-based streaming with line buffer (buffer-flush mode).

picWidth
The width of a picture to be encoded in unit of sample.

picHeight
The height of a picture to be encoded in unit of sample.

linear2TiledEnable

It is a linear-to-tiled enable mode. (CODA9 only) The source frame can be converted from linear format to tiled format in PrP (Pre-Processing) block.

- 0 : disable linear-to-tiled-map conversion
- 1 : enable linear-to-tiled-map conversion

linear2TiledMode

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

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

frameRateInfo

(CODA9) The 16 LSB bits, [15:0], is a numerator and 16 MSB bits, [31:16], is a denominator for calculating frame rate. The numerator means clock ticks per second, and the denominator is clock ticks between frames minus 1.

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

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

(WAVE) A frame rate indicator for rate control For example, the value 30 of frameRateInfo represents 30 frames/sec.

MESearchRangeX

The horizontal search range for Motion Estimation (CODA980 only)

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

MESearchRangeY

The vertical search range for Motion Estimation (CODA980 only)

- 0 : vertical search range (-48 ~ 47)
- 1 : vertical search range(-32 ~ 31)
- 2 : vertical search range(-16 ~ 15)

rcGopIQpOffsetEn

An enable flag for initial QP offset for I picture in GOP. (CODA980 only)

- 0 : disable (default)
- 1 : enable

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

rcGopIQpOffset

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

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

MESearchRange

The search range for Motion Estimation (CODA960 only)

- 0 : horizontal(-128 ~ 127) and vertical(-64 ~ 63)
- 1 : horizontal(-64 ~ 63) and vertical(-32 ~ 31)

- 2 : horizontal(-32 ~ 31) and vertical(-16 ~ 15)
- 3 : horizontal(-16 ~ 15) and vertical(-16 ~ 15)

vbvBufferSize
vbv_buffer_size in bits

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

frameSkipDisable
Frame skip indicates that encoder can skip frame encoding automatically when bitstream has been generated much so far considering the given target bitrate. (CODA9 only) This parameter is ignored if rate control is disabled.
- 0 : enable frame skip function.
- 1 : disable frame skip function.

Note This variable is for CODA9. For WAVE, host can use EncParam.skipPicture.

gopSize
This variable defines the interval of I picture. (CODA9 only)
- 0 : only first I picture
- 1 : all I pictures
- 2 : IPIP …
- 3 : IPPIPP …

The maximum value is 32767, but in practice, a smaller value should be chosen by HOST application for error resilience.

idrInterval
An interval of adding an IDR picture (CODA9 only)

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

sliceMode
the section called “EncSliceMode”

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

ConscIntraRefreshEnable
Consecutive intra MB refresh mode (CODA9 only)

This option is valid only when IntraMbRefresh-Num[15:0] is not 0.
- 0 : Consecutive intra MB refresh mode is disabled. IntraMbRefreshNum of MBs are encoded as intra MB at the predefined interval size.
- 1 : IntraMbRefreshNum of consecutive MBs are encoded as intra MB.

userQpMax
The maximum quantized step parameter for encoding process

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

Note This variable is only for CODA9. For WAVE, host can use waveParam.maxQpI, maxQpP and maxQpB in EncOpenParam.

maxIntraSize
The maximum bit size for intra frame. (CODA9 H.264/AVC only)

userMaxDeltaQp
The maximum delta QP for encoding process. (CODA9 H.264/AVC only)

userMinDeltaQp
The minimum delta QP for encoding process. (CODA9 H.264/AVC only)

userQpMin
The minimum quantized step parameter for encoding process. (CODA9 H.264/AVC only)

MEUseZeroPmv
The PMV option for Motion Estimation. (CODA9 only) If this field is 1, encoding quality can be worse than when it is 0.
- 0 : Motion Estimation engine uses PMV that was derived from neighbor MV.
- 1 : Motion Estimation engine uses Zero PMV.

intraCostWeight
Additional weight of intra cost for mode decision to reduce intra MB density (CODA9 only)

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

FIC = ICOST + W

So, if this field is not zero, the Final Intra Cost have additional weight. Then the mode decision logic is likely to decide inter mode rather than intra mode for MBs.

rcIntraQp
The quantization parameter for I frame (CODA9 only)

When this value is -1, the quantization parameter for I frame is automatically determined by VPU.

userGamma
A gamma is a smoothing factor in motion estimation. A value for gamma is factor * 32768, the factor value is selected from the range $0 \leq$ factor ≤ 1. (CODA9 only)
- If the factor value is close to 0, QP changes slowly.
- If the factor value is close to 1, QP changes quickly.

The default gamma value is 0.75 * 32768

rcIntervalMode
Encoder rate control mode setting (CODA9 only)
- 0 : normal mode rate control - QP changes for every MB
- 1 : FRAME_LEVEL rate control - QP changes for every frame
- 2 : SLICE_LEVEL rate control - QP changes for every slice

- 3 : USER DEFINED MB LEVEL rate control - QP changes for every number of mbInterval

mbInterval
The user defined MB interval value (CODA9 only)

This value is used only when rcIntervalMode is 3.

bitRate
- CODA9
  - Target bit rate in kbps. If it is 0, it means that rate control is disabled.
- WAVE series
  - Target bit rate in bps

bitRateBL
The bitrate value of base layer (SVAC encoder only)

rcInitDelay
Time delay in mili-seconds (CODA9 only)

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

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

rcEnable
- WAVE series
  - 0 : rate control is off.
  - 1 : rate control is on.
- CODA9
  - 0 : constant QP (VBR, rate control off)
  - 1 : constant bitrate (CBR)
  - 2 : average bitrate (ABR)
  - 4 : picture level rate control

mp4Param
the section called “EncMp4Param”

h263Param
the section called “EncH263Param”

avcParam
the section called “EncAvcParam”

waveParam
the section called “EncWaveParam”

cacheBypass
Cache MC bypass (CODA9 only)
- 0 : MC uses a cache.
- 1 : MC does not use a cache.

cbcrInterleave
- 0 : Cb data are written in Cb frame memory and Cr data are written in Cr frame memory. (chroma separate mode)
- 1 : Cb and Cr data are written in the same chroma memory. (chroma interleave mode)

cbcrOrder

CbCr order in planar mode (YV12 format)

- 0 : Cb data are written first and then Cr written in their separate plane.
- 1 : Cr data are written first and then Cb written in their separate plane.

frameEndian

The endianess of the frame buffer for reference and reconstruction

- 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

The endianess of the bitstream 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*.

sourceEndian

The endianess of the frame buffer for source YUV

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

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

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.

lineBufIntEn

- 0 : Disable
- 1 : Enable

This flag is used to encode frame-based streaming video with line buffer. If this field is set, VPU sends a buffer full interrupt when line buffer is full and waits until the interrupt is cleared. HOST should read the bitstream in line buffer and clear the interrupt. If this field is not set, VPU does not send a buffer full interrupt even if line buffer is full.

packedFormat

the section called "PackedFormatNum"

srcFormat

A color format of source image defined in *the section called "FrameBufferFormat"*.

outputFormat
A color format of output image defined in *the section called "FrameBufferFormat"*.

srcBitDepth
A bit-depth of source image

coreIdx
VPU core index number

- 0 to (number of VPU core - 1)

nv21
- 0 : CbCr data is interleaved in chroma source frame memory. (NV12)
- 1 : CrCb data is interleaved in chroma source frame memory. (NV21)

virtAxiID
AXI-ID for the V-CPU part (for virtualization)

enablePTS
An enable flag to report PTS(Presentation Timestamp)

lowLatencyMode
2bits low latency mode setting. bit[1]: low latency interrupt enable, bit[0]: fast bit-stream-packing enable (only for WAVE5)

enableNonRefFbcWrite
If it is TRUE, FBC data of non-reference picture are written into framebuffer.

picWidthBL
The width of the BL(base layer) picture in SVC (SVAC encoder only)

picHeightBL
The height of the BL(base layer) picture in SVC (SVAC encoder only)

sourceBufCount
The number of source frame buffer in encoder

streamBufCount
The number of stream buffer in encoder

streamBufSize
The size of stream buffer in encoder

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

srcReleaseIntEnable
It enables released source buffer interrupt.

ringBufferWrapEnable
It enables read and write pointers to be wrapped around under buffer fullness in ring buffer mode.

EncInitialInfo

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

```
    int             maxLatencyPictures;
    int             seqInitErrReason;
    int             warnInfo;
    Uint32          vlcBufSize;
    Uint32          paramBufSize;
} EncInitialInfo;
```

Description

This is a data structure which contains the number of source frame buffer and reconstructed frame buffer required for running an encoder instance. This is returned after calling GetInitialInfo().

minFrameBufferCount
Minimum number of frame buffer

minSrcFrameCount
Minimum number of source buffer

maxLatencyPictures
Maximum number of picture latency

seqInitErrReason
Error information

warnInfo
Warn information

vlcBufSize
The size of task buffer

paramBufSize
The size of task buffer

EncCodeOpt

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

Description

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

implicitHeaderEncode
Whether HOST application encodes a header implicitly or not. If this value is 1, three encode options encodeVPS, encodeSPS, and encodePPS are ignored.

encodeVCL
A flag to encode VCL nal unit explicitly

encodeVPS
A flag to encode VPS nal unit explicitly

encodeSPS
A flag to encode SPS nal unit explicitly

encodePPS
A flag to encode PPS nal unit explicitly

encodeAUD
A flag to encode AUD nal unit explicitly

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

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

encodeVUI
A flag to encode VUI nal unit explicitly

encodeFiller
A flag to encode Filler nal unit explicitly (WAVE5 only)

EncParam

```
typedef struct {
   FrameBuffer*    sourceFrame;
   int             forceIPicture;
   int             skipPicture;
   int             quantParam;
   PhysicalAddress picStreamBufferAddr;
  int             picStreamBufferSize;
   int             fieldRun;


   int             forcePicQpEnable;
   int             forcePicQpI;
   int             forcePicQpP;
   int             forcePicQpB;
   int             forcePicTypeEnable;
   int             forcePicType;
   int             srcIdx;
   int             srcEndFlag;
   EncCodeOpt      codeOption;
   Uint32  useCurSrcAsLongtermPic;
   Uint32  useLongtermRef;
   Uint64  pts;
   Uint32 coda9RoiEnable;
   Uint32 coda9RoiPicAvgQp;
   PhysicalAddress roiQpMapAddr;
   Uint32 nonRoiQp;
   // belows are newly added for WAVE5 encoder
   WaveCustomMapOpt customMapOpt;
   Uint32 wpPixSigmaY;
   Uint32 wpPixSigmaCb;
   Uint32 wpPixSigmaCr;
   Uint32 wpPixMeanY;
   Uint32 wpPixMeanCb;
   Uint32 wpPixMeanCr;
   Uint32 forceAllCtuCoefDropEnable;
   // only for SVAC encoder
   Int32 userFilterLevelEnable;
   Int32 lfFilterLevel;
   Int32 sharpLevel;
   Int32 lfRefDeltaIntra;
   Int32 lfRefDeltaRef0;
   Int32 lfRefDeltaRef1;
   Int32 lfModeDelta;
   Int32 svcLayerFlag;
```

```
    FrameBuffer*    OffsetTblBuffer;
    Uint32  qosEnable;
    Uint32  qosConfig;
    Uint32  qosValVcore0;
    Uint32  qosValVcore1;
#ifdef SUPPORT_LOOK_AHEAD_RC
    int  larcData[3];
#endif
} EncParam;
```

Description

This is a data structure for configuring picture encode operation. The variables can change every time one picture is encoded.

sourceFrame
This member must represent the frame buffer containing source image to be encoded.

forceIPicture
If this value is 0, the picture type is determined by the encoder according to the various parameters such as encoded frame number and GOP size.

If this value is 1, the frame is encoded as an I-picture regardless of the frame number or GOP size, and I-picture period calculation is reset to initial state. In MPEG4 and H.263 case, I-picture is sufficient for decoder refresh. In H.264/AVC case, the picture is encoded as an IDR (Instantaneous Decoding Refresh) picture.

This value is ignored if skipPicture is 1. (CODA9 only)

skipPicture
If this value is 0, the encoder encodes a picture as normal.

If this value is 1, the encoder ignores sourceFrame and generates a skipped picture. (WAVE only) In this case, the reconstructed image at decoder side is a duplication of the previous picture. The skipped picture is encoded as P-type regardless of the GOP size.

quantParam
This value is used for all quantization parameters in case of VBR - no rate control. (CODA9 only)

picStreamBufferAddr
The start address of picture stream buffer under line-buffer mode.

This variable represents the start of picture stream for encoded output. In buffer-reset mode, HOST might use multiple picture stream buffers for the best performance. By using this variable, HOST application could re-register the start position of the picture stream while issuing a picture encoding operation. The buffer size is specified by the following variable, picStreamBufferSize. In packet-based streaming with ring-buffer, this variable is ignored.

Note This variable is only meaningful when line-buffer mode is enabled.

picStreamBufferSize
This variable represents the byte size of picture stream buffer. This variable is so crucial in line-buffer mode. That is because encoder output could be corrupted if this size is smaller than any picture encoded output. So this value should be big enough for storing multiple picture streams with average size. In packet-based streaming with ring-buffer, this variable is ignored.

fieldRun
- 0 : progressive (frame) encoding
- 1 : interlaced (field) encoding

This is only for CODA9.

forcePicQpEnable
A flag to use a force picture quantization parameter (WAVE only)

forcePicQpI
A force picture quantization parameter for I picture. It is valid when forcePicQpEnable is 1. (WAVE only)

forcePicQpP
A force picture quantization parameter for P picture. It is valid when forcePicQpEnable is 1. (WAVE only)

forcePicQpB
A force picture quantization parameter for B picture. It is valid when forcePicQpEnable is 1. (WAVE only)

forcePicTypeEnable
A flag to use a force picture type (WAVE only)

forcePicType
A force picture type (I, P, B, IDR, CRA). It is valid when forcePicTypeEnable is 1. (WAVE only)

srcIdx
A source frame buffer index (WAVE only)

srcEndFlag
A flag indicating that there is no more source frame buffer to encode (WAVE only)

codeOption
the section called "EncCodeOpt" (WAVE only)

useCurSrcAsLongtermPic
A flag for the current picture to be used as a longterm reference picture later when other picture's encoding (WAVE only)

useLongtermRef
A flag to use a longterm reference picture in DPB when encoding the current picture (WAVE only)

pts
The presentation Timestamp (PTS) of input source

coda9RoiEnable
A flag to use ROI (CODA9 only)

coda9RoiPicAvgQp
A average value of ROI QP for a picture (CODA9 only)

roiQpMapAddr
The start address of ROI QP map (CODA9 only)

nonRoiQp
A non-ROI QP for a picture

customMapOpt
the section called "WaveCustomMapOpt"

wpPixSigmaY
Pixel variance of Y component for weighted prediction

wpPixSigmaCb
Pixel variance of Cb component for weighted prediction

wpPixSigmaCr
Pixel variance of Cr component for weighted prediction

wpPixMeanY
Pixel mean value of Y component for weighted prediction

wpPixMeanCb
Pixel mean value of Cb component for weighted prediction

wpPixMeanCr
Pixel mean value of Cr component for weighted prediction

forceAllCtuCoefDropEnable
It forces all coefficients to be zero after TQ .

userFilterLevelEnable
It enables to set the user filter level. (SVAC encoder only)

lfFilterLevel
It specifies the loop filter level. (0 ~ 63) The userFilterLevelEnable must be 1. (SVAC encoder only)

sharpLevel
It specifies the sharpness level of filter. (0 ~ 7) (SVAC encoder only)

lfRefDeltaIntra
It specifies a delta value of filter level for key frame. (-63 ~ 63) (SVAC encoder only)

lfRefDeltaRef0
It specifies a delta value of filter level for Ref0 (Dynamic Ref) frame. (-63 ~ 63) (SVAC encoder only)

lfRefDeltaRef1
It specifies a delta value of filter level for Ref1 (Optional Ref) frame. (-63 ~ 63) (SVAC encoder only)

lfModeDelta
It specifies a delta value of filter level according to inter/intra mode. (-63 ~ 63) (SVAC encoder only)

svcLayerFlag
It specifies an spatial SVC layer.
- 0 : base layer picture
- 1 : enhanced layer picture

OffsetTblBuffer
A offset table buffer address for Cframe50

qosEnable
It updates QoS control information. To update QoS values, qosEnable must be 1.

qosConfig
It enables register based QoS control. If register based QoS control feature is enabled, QoS for AXI is set by given qosValVcore0 and qosValVcore1 respectively.

qosValVcore0
Qos value for VCORE0

qosValVcore1
Qos value for VCORE1

EncReportInfo

```
typedef struct {
    int enable;
    int type;
    int sz;
    PhysicalAddress addr;
} EncReportInfo;
```

Description

This structure is used for reporting encoder information. (CODA9 only)

enable
- 0 : reporting disable
- 1 : reporting enable

type
This value is used for picture type reporting in MVInfo and Sliceinfo.

sz
This value means the size for each reporting data (MBinfo, MVinfo, Sliceinfo).

addr
The start address of each reporting buffer into which encoder puts data.

EncOutputInfo

```
typedef struct {
    PhysicalAddress bitstreamBuffer;
    Uint32 bitstreamSize;
    int bitstreamWrapAround;
    int picType;
    int numOfSlices;
    int reconFrameIndex;
    FrameBuffer reconFrame;
    int rdPtr;
    int wrPtr;
    int picSkipped;
    int numOfIntra;
    int numOfMerge;
    int numOfSkipBlock;
    int avgCtuQp;
    int encPicByte;
    int encGopPicIdx;
    int encPicPoc;
    int releaseSrcFlag;
    int encSrcIdx;
    int encNumNut;
    int encVclNut;
    int encPicCnt;
    int errorReason;
    int warnInfo;
    // Report Information
    EncReportInfo mbInfo;
    EncReportInfo mvInfo;
    EncReportInfo sliceInfo;
    int frameCycle;
    Uint64 pts;
    Uint32   cyclePerTick;
    Uint32   encHostCmdTick;
    Uint32   encPrepareStartTick;
```

```
    Uint32   encPrepareEndTick;
    Uint32   encProcessingStartTick;
    Uint32   encProcessingEndTick;
    Uint32   encEncodeStartTick;
    Uint32   encEncodeEndTick;
    Uint32   prepareCycle;
    Uint32   processing;
    Uint32   EncodedCycle;
    Uint32  picDistortionLow;
    Uint32  picDistortionHigh;
    Uint32  isSvcLayerEL;
#ifdef SUPPORT_LOOK_AHEAD_RC
    int  larcData[3];
#endif
    RetCode result;
} EncOutputInfo;
```

Description

This is a data structure for reporting the results of picture encoding operations.

bitstreamBuffer
The physical address of the starting point of newly encoded picture stream

If dynamic buffer allocation is enabled in line-buffer mode, this value is identical with the specified picture stream buffer address by HOST.

bitstreamSize
The byte size of encoded bitstream

bitstreamWrapAround
This is a flag to indicate that the write point is wrapped around in bitsteam buffer in case of ring buffer mode. If this flag is 1 in line buffer mode, it indicates fullness of bitstream buffer. Then HOST application needs a larger bitstream buffer.

picType
the section called “PicType”

numOfSlices
The number of slices of the currently being encoded Picture

reconFrameIndex
A reconstructed frame index. The reconstructed frame can be used for reference of future frame.

reconFrame
A reconstructed frame address and information. Please refer to *the section called “FrameBuffer”*.

rdPtr
A read pointer in bitstream buffer, which is where HOST has read encoded bitstream from the buffer

wrPtr
A write pointer in bitstream buffer, which is where VPU has written encoded bitstream into the buffer

picSkipped
A flag which represents whether the current encoding has been skipped or not. (WAVE5 only)

numOfIntra
The number of intra coded block (WAVE5 HEVC only)

numOfMerge
The number of merge block in 8x8 (WAVE5 HEVC only)

numOfSkipBlock
The number of skip block in 8x8 (WAVE5 HEVC only)

avgCtuQp
The average value of CTU QPs (WAVE5 only)

encPicByte
The number of encoded picture bytes (WAVE5 only)

encGopPicIdx
The GOP index of the currently encoded picture (WAVE5 only)

encPicPoc
The POC(picture order count) value of the currently encoded picture (WAVE5 only)

releaseSrcFlag
The source buffer indicator of the encoded pictures(WAVE5 only)

encSrcIdx
The source buffer index of the currently encoded picture (WAVE5 only)

encNumNut
The number of nal_unit_type of the currently encoded picture (WAVE5 only)

encVclNut
The VCL NAL unit type of the currently encoded picture(WAVE5 only). For more information, refer to the RET_QUERY_ENC_VCL_NUT register in programmer's guide.

encPicCnt
The encoded picture number (WAVE5 only)

errorReason
The error reason of the currently encoded picture (WAVE5 only)

warnInfo
The warning information of the currently encoded picture (WAVE5 only)

mbInfo
The parameter for reporting MB data(CODA9 only) . Please refer to *the section called "EncReportInfo"* structure.

mvInfo
The parameter for reporting motion vector(CODA9 only). Please refer to *the section called "EncReportInfo"* structure.

sliceInfo
The parameter for reporting slice information(CODA9 only). Please refer to *the section called "EncReportInfo"* structure.

frameCycle
The parameter for reporting the cycle number of encoding one frame.

pts
The PTS(Presentation Timestamp) of the encoded picture which is retrieved and managed from VPU API

cyclePerTick
The number of cycles per tick

encHostCmdTick
Tick of ENC_PIC command for the picture

encPrepareStartTick
Start tick of preparing slices of the picture

encPrepareEndTick
End tick of preparing slices of the picture

encProcessingStartTick
Start tick of processing slices of the picture

encProcessingEndTick
End tick of processing slices of the picture

encEncodeStartTick
Start tick of encoding slices of the picture

encEncodeEndTick
End tick of encoding slices of the picture

prepareCycle
The number of cycles for preparing slices of a picture

processing
The number of cycles for processing slices of a picture

EncodedCycle
The number of cycles for encoding slices of a picture

picDistortionLow
The 32bit lowest difference value(SSD) between an original block and a reconstructed block in the encoded picture. It can be an indicator for image quality. Host cannot tune quality by using this value, because this value is not an input parameter, but a reporting value.

picDistortionHigh
The 32bit highest difference value(SSD) between an original block and a reconstructed block in the encoded picture. It can be an indicator for image quality. Host cannot tune quality by using this value, because this value is not an input parameter, but a reporting value.

isSvcLayerEL
SVC layer type of the encoded picture
- 0 : BL picture
- 1 : EL picture

result
The return value of VPU_EncGetOutputInfo()

EncHeaderParam

```
typedef struct {
   PhysicalAddress buf;
   size_t  size;
   Int32   headerType;
   BOOL    zeroPaddingEnable;
```

```
} EncHeaderParam;
```

Description

This structure is used for adding a header syntax layer into the encoded bitstream. The headerType, buf, and zeropaddingEnable are input parameters to VPU. The size is a returned value from VPU after completing requested operation.

buf
A physical address pointing the generated stream location

size
The size of the generated stream in bytes

headerType
This is a type of header that HOST wants to generate such as *the section called "Mp4HeaderType"*, *the section called "AvcHeaderType"* or *the section called "WaveEncHeaderType"*.

zeroPaddingEnable
It enables header to be padded at the end with zero for byte alignment. (CODA9 only)

Chapter 3
API DEFINITIONS

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_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_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_WaitInterrupt()

Prototype

```
Int32 VPU_WaitInterrupt (
                          Uint32 coreIdx,
                          int timeout
                        );
```

Description

This function waits for interrupts to be issued from VPU during the given timeout period. VPU sends an interrupt when it completes a command or meets an exceptional case. (CODA9 only)

The behavior of this function depends on VDI layer's implementation. Timeout may not work according to implementation of VDI layer.

Parameter

Parameter	Type	Description
coreIdx	Input	An index of VPU core
timeout	Input	Time to wait

Return Value

- -1 : timeout
- Non -1 value : The value of InterruptBit

VPU_WaitInterruptEx()

Prototype

```
Int32 VPU_WaitInterruptEx (
                            VpuHandle handle,
                            int timeout
                          );
```

Description

This function waits for interrupts to be issued from VPU during the given timeout period. VPU sends an interrupt when it completes a command or meets an exceptional case. (WAVE only)

The behavior of this function depends on VDI layer's implementation. Timeout may not work according to implementation of VDI layer.

Parameter

Parameter	Type	Description
handle	Input	A decoder/encoder handle obtained from VPU_DecOpen()/ VPU_EncOpen()
timeout	Input	Time to wait

Return Value

- -1 : timeout
- Non -1 value : The value of InterruptBit

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_ClearInterruptEx()

Prototype

```
void VPU_ClearInterruptEx (
                             VpuHandle handle,
                             Int32 intrFlag
                           );
```

Description

This function clears VPU interrupts for a specific instance.

Parameter

Parameter	Type	Description
handle	Input	A decoder/encoder handle obtained from VPU_DecOpen()/ VPU_EncOpen()
intrFlag	Input	An interrupt flag to be cleared

Return Value

None

VPU_SWReset()

Prototype

```
RetCode VPU_SWReset (
                      Uint32 coreIdx,
                      SWResetMode resetMode,
                      void *pendingInst
                    );
```

Description

This function stops the current decode or encode operation and resets the internal blocks - BPU_CORE, BPU_BUS, VCE_CORE, and VCE_BUS. It can be used when VPU is having a longer delay(timeout) or seems hang-up.

SW_RESET_SAFETY

SW_RESET_SAFETY moves the context back to the state before calling the current VPU_DecStartOneFrame()/VPU_EncStartOneFrame(). After calling `VPU_SWReset()` with SW_RESET_SAFETY, HOST can resume decoding/encoding from the next picture by calling VPU_DecStartOneFrame()/VPU_EncStartOneFrame(). 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 SW_RESET_SAFETY especially for occurrance of hang-up. For example, when VPU is hung up with frame 1, HOST application calls `VPU_SWReset()` and calls `VPU_DecStartOneFrame()` for frame 2 with specifying the start address and read pointer. If there is still problem with frame 2, we recommend as below.

- calling `VPU_SWReset()` with SW_RESET_SAFETY and `seq_init()`
- calling `VPU_SWReset()` with SW_RESET_SAFETY and enabling iframe search.

SW_RESET_FORCE

Unlike the SW_RESET_SAFETY, SW_RESET_FORCE requires to restart the whole process from initialization of VPU , setting parameters, and registering frame buffers.

Parameter

Parameter	Type	Description
coreIdx	Input	An index of VPU core
resetMode	Input	Way of reset • SW_RESET_SAFETY : It waits for completion of ongoing bus transactions. If remaining bus transactions are done, VPU goes into the reset process. (recommended mode) • SW_RESET_FORCE : It forces to do SW_RESET immediately no matter whether bus transactions are completed or not. It might affect what other blocks do with bus. We do not recommend using this mode. • SW_RESET_ON_BOOT : This is the default reset mode which is only executed while system boots up. In fact, SW_RESET_ON_BOOT is executed in VPU_Init() and there is no separate use case.
pendingInst	Input	An instance handle

Return Value

RETCODE_SUCCESS
Operation was done 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, HOST application needs to call `VPU_Init()` after `VPU_HWReset()`.

`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.

VPU_SleepWake()

Prototype

```
RetCode VPU_SleepWake (
                        Uint32 coreIdx,
                        int iSleepWake
                      );
```

Description

This function saves or restores context when VPU is powered on or 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.

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_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_DecGetOutputInfo(). 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_GetProductInfo()

Prototype

```
RetCode VPU_GetProductInfo (
                              Uint32 coreIdx,
                              VpuAttr *productInfo
                            );
```

Description

This function returns the hardware attributes for VPU. *the section called "VpuAttr"*

Parameter

Parameter	Type	Description
coreIdx	Input	An index of VPU core
productInfo	Output	*the section called "VpuAttr"*

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means version information is acquired successfully.

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_GetFrameBufSize()

Prototype

```
int VPU_GetFrameBufSize (
                          VpuHandle handle,
                          int coreIdx,
                          int stride,
                          int height,
                          int mapType,
                          int format,
                          int interleave,
                          DRAMConfig *pDramCfg
                        );
```

Description

This function returns the size of one frame buffer that is required for VPU to decode or encode a frame.

Parameter

Parameter	Type	Description
handle	Input	A decoder/encoder handle obtained from VPU_DecOpen()/ VPU_EncOpen()
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_DecOpen()

Prototype

```
RetCode VPU_DecOpen (
                      DecHandle *pHandle,
                      DecOpenParam *pop
                    );
```

Description

This function opens a decoder instance in order to start a new decoder operation. By calling this function, HOST application can get a handle by which they can refer to a decoder instance. HOST application needs this kind of handle under multiple instances running codec. Once 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 DecHandle type variable which specifies each instance for HOST application.
pop	Input	A pointer to *the section called "DecOpenParam"* 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, `pop`, 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

This function closes the given decoder instance. By calling this function, HOST application can end decoding of the sequence and release the instance. After completion of this function call, relevant resources of the instance get free. Once 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	Input	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.

VPU_DecGetInitialInfo()

Prototype

```
RetCode VPU_DecGetInitialInfo (
                                DecHandle handle,
                                DecInitialInfo *info
                              );
```

Description

This function decodes the sequence header in the bitstream buffer and returns crucial information for running decode operation such as the required number of frame buffers. 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 HOST application's responsibility to provide sufficient amount of bitstream to the decoder 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().

This function call plays the same role of calling DecIssueSeqInit() and DecCompleteSeqInit().

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, `info`, 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 that frame decoding operation was not completed yet, so the given API function call cannot be allowed.

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.

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_DecGetBitstreamBuffer(), and VPU_DecUpdateBitstreamBuffer().

A pair of VPU_DecIssueSeqInit() and VPU_DecCompleteSeqInit() or just 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 the request for information of the stream data to be decoded was sent 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().

RETCODE_QUEUEING_FAILURE
This means that the current API function call cannot be queued because queue buffers are full at the moment.

VPU_DecCompleteSeqInit()

Prototype

```
RetCode VPU_DecCompleteSeqInit (
                                  DecHandle handle,
                                  DecInitialInfo *info
                                );
```

Description

This function returns the *the section called "DecInitialInfo"* structure which holds crucial sequence information for decoder such as picture size, number of necessary frame buffers, etc.

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 a failure in getting sequence information for some reason - syntax errror, unableness to find sequence header, or missing complete sequence header.

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 allow HOST to escape from waiting state while VPU_DecIssueSeqInit()/VPU_DecCompleteSeqInit() are executed. When escape flag is set to 1 and stream buffer empty happens, VPU terminates VPU_DecIssueSeqInit()/ VPU_DecCompleteSeqInit() operation without issuing empty interrupt.

This function only works in ring buffer mode of bitstream buffer.

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 set.

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
BitstreamMode of DecOpenParam structure is not BS_MODE_INTERRUPT (ring buffer mode).

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() or VPU_DecCompleteSeqInit(). The frame buffers pointed to by bufArray are managed internally within VPU. These include reference frames, reconstructed frame, etc. num must not be less than minFrameBufferCount obtained by VPU_DecGetInitialInfo() or VPU_DecCompleteSeqInit().

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
bufArray	Input	The allocated frame buffer address and information in *the section called "FrameBuffer"*. If this parameter is NULL, this function (not HOST application) 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	A frame height
mapType	Input	A Map type for GDI inferface or FBC (Frame Buffer Compression) in *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 VPU operation was not completed yet, so the given API function call cannot be performed this time.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. 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 AXI bus width 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() or VPU_DecCompleteSeqInit(). 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	The allocated frame buffer address and information in *the section called "FrameBuffer"*. If this parameter is NULL, this function (not HOST application) allocates frame buffers.
numOfDecFbs	Input	The number of compressed frame buffer
numOfDisplayFbs	Input	The number of linear frame buffer when WTL is enabled. In WAVE, 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	A frame height
mapType	Input	A Map type for GDI inferface or FBC (Frame Buffer Compression) in *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 VPU operation was not completed yet, so the given API function call cannot be performed this time.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. 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 AXI bus width 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_DecUpdateFrameBuffer()

Prototype

```
RetCode VPU_DecUpdateFrameBuffer (
                                    DecHandle handle,
                                    FrameBuffer *fbcFb,
                                    FrameBuffer *linearFb,
                                    Int32 mvColIndex,
                                    Int32 picWidth,
                                    Int32 picHeight
                                  );
```

Description

This is a special function for VP9 decoder that allows HOST application to replace one of the registered array of frame buffers with a single new set of frame buffers - linear frame buffer, FBC frame buffer, and ColMv buffer. This is the dedicated function only for the case that a new inter-frame is coded using a different resolution than the previous frame.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
fbcFb	Input	The new FBC frame buffer index
linearFb	Input	The new linear frame buffer index
mvColIndex	Input	The new co-located motion vector buffer index
picWidth	Input	The new frame width
picHeight	Input	The new frame height

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_NOT_SUPPORTED_FEATURE
This means that HOST application uses this API call in other than VP9 decoder.

RETCODE_INSUFFICIENT_RESOURCE
This means failure to allocate a framebuffer due to lack of memory.

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 allows 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	*the section called "FrameBufferAllocInfo"*
frameBuffer	Output	*the section called "FrameBuffer"* 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
This means failure 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 frame buffer information of the given frame buffer index.

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 in 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 in *the section called "FrameBuffer"*.

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_DecStartOneFrame()

Prototype

```
RetCode VPU_DecStartOneFrame (
                                DecHandle handle,
                                DecParam *param
                              );
```

Description

This function starts decoding one frame. For the completion of decoding one frame, VPU_DecGetOutputInfo() should be called with the same handle.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
param	Input	*the section called "DecParam"* 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 VPU operation was not completed yet, so the given API function call cannot be performed this time.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. HOST might call this function before successfully calling VPU_DecRegisterFrameBuffer(). This function should be called after calling VPU_ DecRegisterFrameBuffer() successfully.

RETCODE_QUEUEING_FAILURE
This means that the current API function call cannot be queued because queue buffers are full at the moment.

VPU_DecGetOutputInfo()

Prototype

```
RetCode VPU_DecGetOutputInfo (
                                DecHandle handle,
                                DecOutputInfo *info
                              );
```

Description

VPU returns the result of frame 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.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
info	Output	A pointer to *the section called "DecOutputInfo"* 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.

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_QUERY_FAILURE
This means this query command was not successful. (WAVE5 only)

RETCODE_REPORT_NOT_READY
This means that report is not ready for this query(GET_RESULT) command. (WAVE5 only)

VPU_DecGiveCommand()

Prototype

```
RetCode VPU_DecGiveCommand (
                              DecHandle handle,
                              CodecCommand cmd,
                              void *parameter
                            );
```

Description

This function executes an additional command such to set Secondary AXI or to report user data which is given by HOST application. It allows HOST application to set directly the variables that can be set only through the API layer. 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 *the section called "CodecCommand"*
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 VPU operation was not completed yet, so the given API function call cannot be performed this time.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not received any response from VPU and has timed out.

RETCODE_QUEUEING_FAILURE
This means that the current API function call cannot be queued because queue buffers are full at the moment.

VPU_DecGetBitstreamBuffer()

Prototype

```
RetCode VPU_DecGetBitstreamBuffer (
                                     DecHandle handle,
                                     PhysicalAddress *prdPrt,
                                     PhysicalAddress *pwrPtr,
                                     Uint32 *size
                                   );
```

Description

This function returns the read pointer, write pointer, available space of the bitstream in ringbuffer mode. 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 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

This function notifies VPU of how much bitstream has been transferred to the bitstream buffer. 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 bit-stream 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 bit-stream and returns -1 to `indexFrameDisplay`. • -1: It enables to resume decoding without call-ing VPU_DecClose() after remaining stream has completely been decoded to the end of stream by VPU_DecUpdateBitstreamBuffer(handle, 0). • -2 : It enables to decode until the current write pointer and force to end decoding. It is for an exceptional case such as failure of finding sequence header in interrupt mode. If that happens, VPU is in a state seeking sequence header, while HOST keeps feeding to the end of bistream, but never gets the command done signal for a long time.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means putting new stream data was done success-fully.

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 a same value of read pointer (addr) when updateWrPtr is not a zero value. 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_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 can be used to do random access (like skip picture) or to continue seamless decode operation without calling VPU_DecClose() after change of sequence.

Note In WAVE, 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 *the section called "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 the given index of frame buffer. If the display flag of frame buffer is cleared, the frame buffer can be reused in the decoding process. VPU API keeps the display index of frame buffer remained until VPU_DecClrDispFlag() is called.

Parameter

Parameter	Type	Description
handle	Input	A decoder handle obtained from VPU_DecOpen()
index	Input	A frame buffer index to be cleared

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means receiving the output information of the current frame was done successfully.

RETCODE_INVALID_HANDLE
This means argument handle is invalid. This includes cases where `handle` is not a handle which has been obtained by VPU_DecOpen(), `handle` is a handle to an instance already closed, or `handle` is a handle to an decoder instance. Also,this value is returned when VPU_DecStartOneFrame() is matched with VPU_DecGetOutputInfo() with different handles.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. It might happen because VPU_DecRegisterFrameBuffer() with the same handle was not called before calling this function.

RETCODE_INVALID_PARAM
The given argument parameter, index, was invalid, which means it has improper values.

VPU_EncOpen()

Prototype

```
RetCode VPU_EncOpen (
                      EncHandle *handle,
                      EncOpenParam *encOpParam
                    );
```

Description

This function opens an encoder instance in order to start a new encoder operation. By calling this function, HOST application can get a handle specifying a new encoder instance. Because VPU supports multiple instances of codec operations, HOST application needs this kind of handles for the all codec instances now on running. Once HOST application gets a handle, the HOST application must use this handle to represent the target instances for all subsequent encoder-related functions.

Parameter

Parameter	Type	Description
handle	Output	A pointer to EncHandle type variable which specifies each instance for HOST application. If no instance is available, null handle is returned.
encOpParam	Input	A pointer to *the section called "EncOpenParam"* structure which describes required parameters for creating a new encoder instance.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means a new encoder instance was opened successfully.

RETCODE_FAILURE
Operation was failed, which means getting a new encoder instance was not done successfully. If there is no free instance anymore, this value is returned in this function call.

RETCODE_INVALID_PARAM
The given argument parameter, pOpenParam, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

RETCODE_NOT_INITIALIZED
VPU was not initialized at all before calling this function. Application should initialize VPU by calling VPU_Init() before calling this function.

VPU_EncClose()

Prototype

```
RetCode VPU_EncClose (
                        EncHandle handle
                      );
```

Description

This function closes the given encoder instance. By calling this function, HOST application can end encoding of the sequence and release the instance. After completion of this function call, relevant resources of the instance get free. Once HOST application closes an instance, the HOST application cannot call any further encoder-specific function with the current handle before re-opening a new encoder instance with the same handle.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()

Return Value

RETCODE_SUCCESS
Operation was done successfully. That means the current encoder instance was closed successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- pHandle is not a handle which has been obtained by VPU_EncOpen().
- pHandle is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding or encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding or decoding operation should be completed by calling VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not recieved any response from VPU and has timed out.

VPU_EncGetInitialInfo()

Prototype

```
RetCode VPU_EncGetInitialInfo (
                                EncHandle handle,
                                EncInitialInfo *encInitInfo
                              );
```

Description

This function sets sequence information including source width and height and many other parameters such as coding tools, GOP preset, rate control, etc. It also returns the required parameters such as minFrameBufferCount. (CODA9 only)

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
encInitInfo	Output	A pointer to *the section called "EncInitialInfo"* structure which describes required sequence information for the current encoder instance.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means receiving the initial parameters was done successfully.

RETCODE_FAILURE
Operation was failed, which means there was an error in getting information for configuring the encoder.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- pHandle is not a handle which has been obtained by VPU_EncOpen().
- pHandle is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding or encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding or decoding operation should be completed by calling VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_INVALID_PARAM
The given argument parameter, pInitialInfo, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

RETCODE_CALLED_BEFORE
This function call is invalid which means multiple calls of the current API function for a given instance are not allowed. In this case, encoder initial information has been received already, so that this function call is meaningless and not allowed anymore.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not recieved any response from VPU and has timed out.

VPU_EncIssueSeqInit()

Prototype

```
RetCode VPU_EncIssueSeqInit (
                               EncHandle handle
                             );
```

Description

Before starting encoder operation, HOST application should set sequence information including source width and height and many other parameters such as coding tools, GOP preset, rate control, etc. (WAVE only)

Parameter

Parameter	Type	Description
handle	Input	A encoder handle obtained from VPU_EncOpen()

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means receiving the initial parameters was done successfully.

RETCODE_FAILURE
Operation was failed, which means there was an error in getting information for configuring the encoder.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- pHandle is not a handle which has been obtained by VPU_EncOpen().
- pHandle is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding or encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding or decoding operation should be completed by calling VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_INVALID_PARAM
The given argument parameter, pInitialInfo, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

RETCODE_CALLED_BEFORE
This function call is invalid which means multiple calls of the current API function for a given instance are not allowed. In this case, encoder initial information has been received already, so that this function call is meaningless and not allowed anymore.

RETCODE_QUEUEING_FAILURE
This means that the current API function call cannot be queued because queue buffers are full at the moment.

VPU_EncCompleteSeqInit()

Prototype

```
RetCode VPU_EncCompleteSeqInit (
                                  EncHandle handle,
                                  EncInitialInfo *info
                                );
```

Description

Before starting encoder operation, HOST application must allocate frame buffers according to the information obtained from this function. This function returns the required parameters such as minFrameBufferCount. (WAVE only)

Parameter

Parameter	Type	Description
handle	Input	A encoder handle obtained from VPU_EncOpen()
info	Output	A pointer to *the section called "EncInitialInfo"* structure which describes required sequence information for the current encoder instance.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means receiving the initial parameters was done successfully.

RETCODE_FAILURE
Operation was failed, which means there was an error in getting information for configuring the encoder.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- pHandle is not a handle which has been obtained by VPU_EncOpen().
- pHandle is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding or encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding or decoding operation should be completed by calling VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_INVALID_PARAM
The given argument parameter, pInitialInfo, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

RETCODE_CALLED_BEFORE
This function call is invalid which means multiple calls of the current API function for a given instance are not allowed. In this case, encoder initial information has been received already, so that this function call is meaningless and not allowed anymore.

VPU_EncRegisterFrameBuffer()

Prototype

```
RetCode VPU_EncRegisterFrameBuffer (
                                      EncHandle handle,
                                      FrameBuffer *bufArray,
                                      int num,
                                      int stride,
                                      int height,
                                      TiledMapType mapType
                                    );
```

Description

This function registers frame buffers requested by VPU_EncGetInitialInfo(). The frame buffers pointed to by pBuffer are managed internally within VPU. These include reference frames, reconstructed frames, etc. Applications must not change the contents of the array of frame buffers during the life time of the instance, and `num` must not be less than minFrameBufferCount obtained by VPU_EncGetInitialInfo().

The distance between a pixel in a row and the corresponding pixel in the next row is called a stride. The value of stride must be a multiple of 8. The address of the first pixel in the second row does not necessarily coincide with the value next to the last pixel in the first row. In other words, a stride can be greater than the picture width in pixels.

Applications should not set a stride value smaller than the picture width. So, for Y component, HOST application must allocate at least a space of size (frame height * stride), and Cb or Cr component, (frame height/2 * stride/2), respectively. But make sure that in Cb/Cr non-interleave (separate Cb/Cr) map, a stride for the luminance frame buffer should be multiple of 16 so that a stride for the chrominance frame buffer becomes a multiple of 8.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
bufArray	Input	Allocated frame buffer address and information in *the section called "FrameBuffer"*. If this parameter is set to -1, VPU allocates frame buffers.
num	Input	A number of frame buffers. VPU can allocate frame buffers as many as this given value.
stride	Input	A stride value of the given frame buffers
height	Input	Frame height
mapType	Input	Map type of frame buffer

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means registering frame buffers were done successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- handle is not a handle which has been obtained by VPU_EncOpen().

- handle is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding or encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding or decoding operation should be completed by calling VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. HOST application might call this function before calling VPU_EncGetInitialInfo() successfully. This function should be called after successful calling of VPU_EncGetInitialInfo().

RETCODE_INVALID_FRAME_BUFFER
This means argument pBuffer were invalid, which means it was not initialized yet or not valid anymore.

RETCODE_INSUFFICIENT_FRAME_BUFFERS
This means the given number of frame buffers, num, was not enough for the encoder operations of the given handle. It should be greater than or equal to the value of minFrameBufferCount obtained from VPU_EncGetInitialInfo().

RETCODE_INVALID_STRIDE
This means the given argument stride was invalid, which means it is 0, or is not a multiple of 8 in this case.

RETCODE_CALLED_BEFORE
This function call is invalid which means multiple calls of the current API function for a given instance are not allowed. It might happen when registering frame buffer for this instance has been done already so that this function call is meaningless and not allowed anymore.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not recieved any response from VPU and has timed out.

VPU_EncAllocateFrameBuffer()

Prototype

```
RetCode VPU_EncAllocateFrameBuffer (
                                      EncHandle handle,
                                      FrameBufferAllocInfo info,
                                      FrameBuffer *frameBuffer
                                    );
```

Description

This is a special function that enables HOST application to allocate directly the frame buffer for encoding or for Pre-processor (PRP) such as Rotator. In normal operation, VPU API allocates frame buffers when the argument bufArray in VPU_EncRegisterFrameBuffer() is set to 0. However, for any other reason HOST application can use this function to allocate frame buffers by themselves.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
info	Input	*the section called "FrameBufferAllocInfo"*
frameBuffer	Output	*the section called "FrameBuffer"* that holds information of allocated frame buffers

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means the framebuffer is allocated successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- handle is not a handle which has been obtained by VPU_EncOpen().
- handle is a handle of an instance which has been closed already, etc.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. It might happen because VPU_EncRegisterFrameBuffer() for (FramebufferAllocType.FB_TYPE_CODEC) has not been called, before this function call for allocating frame buffer for PRP (FramebufferAllocType.FB_TYPE_PPU).

RETCODE_INSUFFICIENT_RESOURCE
This means failure 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_EncStartOneFrame()

Prototype

```
RetCode VPU_EncStartOneFrame (
                               EncHandle handle,
                               EncParam *param
                             );
```

Description

This function starts encoding one frame. Returning from this function does not mean the completion of encoding one frame, and it is just that encoding one frame was initiated.

Every call of this function should be matched with VPU_EncGetOutputInfo() with the same handle. In other words, HOST application should call VPU_EncGetOutputInfo() once to get the result of VPU_EncStartOneFrame() call.

For CODA9, without "sequential" calling a pair of VPU_EncStartOneFrame() and VPU_EncGetOutputInfo(), HOST application cannot call any other API functions except VPU_EncGetBitstreamBuffer(), and VPU_EncUpdateBitstreamBuffer().

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
param	Input	A pointer to *the section called "EncParam"* structure which describes picture encoding parameters for the current encoder instance.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means encoding a new frame was started successfully.

Note This return value does not mean that encoding a frame was completed successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- handle is not a handle which has been obtained by VPU_EncOpen().
- handle is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding or encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding or decoding operation should be completed by calling VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. In this case, HOST application might call this function before suc-

cessfully calling VPU_EncRegisterFrameBuffer(). This function should be called after successfully calling VPU_EncRegisterFrameBuffer().

RETCODE_INVALID_PARAM
The given argument parameter, `parameter`, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

RETCODE_INVALID_FRAME_BUFFER
This means sourceFrame in input structure EncParam was invalid, which means sourceFrame was not valid even though picture-skip is disabled.

RETCODE_QUEUEING_FAILURE
This means that the current API function call cannot be queued because queue buffers are full at the moment.

VPU_EncGetOutputInfo()

Prototype

```
RetCode VPU_EncGetOutputInfo (
                                EncHandle handle,
                                EncOutputInfo *info
                              );
```

Description

This function gets information of the output of encoding. Application can obtain the picture type, the address and size of the generated bitstream, and the number of generated slices. HOST application should call this function after frame encoding is finished, and before starting the further processing.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen().
info	Output	A pointer to *the section called "EncOutputInfo"* structure which describes picture encoding results for the current encoder instance.

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means the output information of the current frame encoding was received successfully.

RETCODE_INVALID_HANDLE
The given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- handle is not a handle which has been obtained by VPU_EncOpen(), for example a decoder handle,
- handle is a handle of an instance which has been closed already,
- handle is not the same handle as the last VPU_EncStartOneFrame() has, etc.

RETCODE_WRONG_CALL_SEQUENCE
This means the current API function call was invalid considering the allowed sequences between API functions. HOST application might call this function before calling VPU_EncStartOneFrame() successfully. This function should be called after successful calling of VPU_EncStartOneFrame().

RETCODE_INVALID_PARAM
The given argument parameter, pInfo, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

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

RETCODE_REPORT_NOT_READY
This means that report is not ready for this query(GET_RESULT) command. (WAVE5 only)

VPU_EncGiveCommand()

Prototype

```
RetCode VPU_EncGiveCommand (
                              EncHandle handle,
                              CodecCommand cmd,
                              void *parameter
                            );
```

Description

This function executes an additional function such rotator or changeParam which is given by HOST application. It allows HOST application to set directly the variables that can be set only through the API layer. Some command-specific return codes are also presented.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
cmd	Input	A variable specifying the given command of *the section called "CodecCommand"*
parameter	Input/Output	A pointer to command-specific data structure which describes picture I/O parameters for the current encoder instance

Return Value

RETCODE_INVALID_COMMAND
This means the given argument, cmd, was invalid which means the given cmd was undefined, or not allowed in the current instance.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- pHandle is not a handle which has been obtained by VPU_EncOpen().
- pHandle is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame decoding or encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding or decoding operation should be completed by calling VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo(). Even though the result of the current frame operation is not necessary, HOST application should call VPU_EncGetOutputInfo() or VPU_DecGetOutputInfo() to proceed this function call.

RETCODE_VPU_RESPONSE_TIMEOUT
Operation has not received any response from VPU and has timed out.

RETCODE_QUEUEING_FAILURE
This means that the current API function call cannot be queued because queue buffers are full at the moment.

VPU_EncGetBitstreamBuffer()

Prototype

```
RetCode VPU_EncGetBitstreamBuffer (
                                     EncHandle handle,
                                     PhysicalAddress *prdPrt,
                                     PhysicalAddress *pwrPtr,
                                     int *size
                                   );
```

Description

After encoding a frame, HOST application must get the bitstream from encoder. This function returns the location of encoded stream and the maximum size in bitstream buffer.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
prdPrt	Output	A stream buffer read pointer for the current encoder instance
pwrPtr	Output	A stream buffer write pointer for the current encoder instance
size	Output	A variable indicating the written stream size in bitstream buffer for the current encoder instance

Return Value

RETCODE_SUCCESS
Operation was done successfully. That means the current encoder instance was closed successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- pHandle is not a handle which has been obtained by VPU_EncOpen().
- pHandle is a handle of an instance which has been closed already, etc.

RETCODE_INVALID_PARAM
The given argument parameter, pRdptr, pWrptr or size, was invalid, which means it has a null pointer, or given values for some member variables are improper values.

VPU_EncUpdateBitstreamBuffer()

Prototype

```
RetCode VPU_EncUpdateBitstreamBuffer (
                                        EncHandle handle,
                                        int size
                                      );
```

Description

This function informs VPU API of how much bitstream has been transferred by HOST application from the address obtained from VPU_EncGetBitstreamBuffer(). By just giving the size as an argument, API automatically handles pointer wrap-around and updates the read pointer.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
size	Input	A variable indicating the stream size read by HOST application from the bitstream buffer

Return Value

RETCODE_SUCCESS
Operation was done successfully. That means the current encoder instance was closed successfully.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, pHandle, was invalid. This return code might be caused if

- pHandle is not a handle which has been obtained by VPU_EncOpen().
- pHandle is a handle of an instance which has been closed already, etc.

RETCODE_INVALID_PARAM
The given argument parameter, size, was invalid, which means size is larger than the value obtained from VPU_EncGetBitstreamBuffer().

VPU_EncSetWrPtr()

Prototype

```
RetCode VPU_EncSetWrPtr (
                          EncHandle handle,
                          PhysicalAddress addr,
                          int updateRdPtr
                        );
```

Description

This function specifies the location of write pointer in bitstream buffer. It can also set a read pointer with the same value of write pointer (addr) when updateRdPtr is not a zero value. This function can be used regardless of bitstream buffer mode.

Parameter

Parameter	Type	Description
handle	Input	An encoder handle obtained from VPU_EncOpen()
addr	Input	Updated write pointer
updateRdPtr	Input	A flag whether to move the read pointer to where the write pointer is located

Return Value

RETCODE_SUCCESS
Operation was done successfully, which means required information of the stream data to be encoded was received successfully.

RETCODE_FAILURE
Operation was failed, which means there was an error in getting information for configuring the encoder.

RETCODE_INVALID_HANDLE
This means the given handle for the current API function call, `handle`, was invalid. This return code might be caused if

- `handle` is not a handle which has been obtained by VPU_EncOpen().
- `handle` is a handle of an instance which has been closed already, etc.

RETCODE_FRAME_NOT_COMPLETE
This means frame encoding operation was not completed yet, so the given API function call cannot be performed this time. A frame encoding operation should be completed by calling VPU_ EncSetRdPtr ().

About Chips&Media

Chips&Media, Inc. is a leading video IP provider, headquartered in Seoul, South Korea. The company was established in 2003 by leading members with years of profound experiences in video standard technologies and semiconductor industry. Our extensive catalogue of IP solutions includes video postprocessing, video frame buffer compression as well as video codecs covering vast range of video standards from MPEG-2, MPEG-4, DivX, H.263, Sorenson, H.264, RV, VC-1, VP8, AVS, AVS+, HEVC(H.265) and VP9 for HD to UHD (4K/8K) resolution.

Chips&Media has been developing a line of reliable, high-quality IP solutions that allow our customers and partners to satisfy the growing consumer demand for high-performance multi-media digital devices. Especially as a leading multi-standard video codec solution provider, we have been providing our advanced ultra-low power multi-codec video IPs to top-tier semiconductor companies.

With a mission to become global top SoC IP provider, Chips&Media has introduced Image Signal Processing (ISP) IP and Deep learning-based object detection IP to the market and continues to widen our solution portfolio to cope with growing demand on image processing and related solutions. To find further information, please visit the company's web site at *http://www.chipsnmedia.com*