



H.265 PC Decoding Library Software

## API Reference

Issue	02
Date	2015-02-10

**Copyright © HiSilicon Technologies Co., Ltd. 2014-2015. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of HiSilicon Technologies Co., Ltd.

## **Trademarks and Permissions**



**HISILICON**, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

## **HiSilicon Technologies Co., Ltd.**

Address: Huawei Industrial Base  
Bantian, Longgang  
Shenzhen 518129  
People's Republic of China

Website: <http://www.hisilicon.com>

Email: [support@hisilicon.com](mailto:support@hisilicon.com)



# Contents

<b>About This Document.....</b>	<b>vi</b>
<b>1 Overview.....</b>	<b>1</b>
1.1 Scope.....	1
1.2 API Format.....	2
1.3 Function List.....	2
1.4 Function Description Fields.....	3
1.5 Structure Description Fields.....	3
<b>2 API Description.....</b>	<b>4</b>
2.1 IHW265D_Create.....	4
2.2 IHW265D_Delete.....	6
2.3 IHW265D_GetVersion.....	6
2.4 IHW265D_DecodeFrame.....	7
2.5 IHW265D_DecodeAU.....	10
<b>3 Data Type and Structure.....</b>	<b>13</b>
3.1 Description of Common Data Types.....	13
3.2 Description of the Data Structure.....	13
3.2.1 HW265D_THREADTYPE.....	13
3.2.2 HW265D_OUTPUTORDER.....	14
3.2.3 HW265D_DECODEMODE.....	14
3.2.4 HW265D_FRAMETYPE.....	14
3.2.5 HW265D_DECODESTATUS.....	15
3.2.6 IHW265D_INIT_PARAM.....	15
3.2.7 IH265DEC_INARGS.....	15
3.2.8 IH265DEC_OUTARGS.....	16
3.2.9 HW265D_USERDATA.....	16
3.2.10 IHWVIDEO_ALG_VERSION_STRU.....	17
3.2.11 CU_OUTPUT_INFO.....	17
<b>4 API Application Instance.....</b>	<b>19</b>
4.1 Stream Decoding Process.....	19
4.2 Frame Decoding Process.....	20
4.3 Program Instance.....	20



## Figures

---

<b>Figure 4-1</b> Stream decoding process .....	19
<b>Figure 4-2</b> Frame decoding process.....	20



## Tables

---

<b>Table 1-1</b> Main components of the decoding library .....	1
<b>Table 1-2</b> Running environments of the decoding library .....	2



# About This Document

## Purpose

This document describes the document contents, related product versions, intended audience, conventions and update history.



### NOTE

This document uses the Hi3516A as an example. Unless otherwise stated, Hi3516D and Hi3516A contents are consistent.

## Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3516A	V100
Hi3516D	V100

## Intended Audience

This document is intended for the programmers who meet the following requirements:






- Be familiar with C/C++ programming language
- Be familiar with 32-bit Windows environment

## Conventions

### Symbol Conventions

The following symbols may be found in this document. They are defined as follows.



Symbol	Description
 <b>DANGER</b>	Indicates a hazard with a high level of risk which, if not avoided, will result in death or serious injury.
 <b>WARNING</b>	Indicates a hazard with a medium or low level of risk that, if not avoided, could result in minor or moderate injury.
 <b>CAUTION</b>	Indicates a potentially hazardous situation that, if not avoided, could cause equipment or component damage, data loss, and performance degradation, or unexpected results.
 <b>TIP</b>	Indicates a tip that may help you solve a problem or save time.
 <b>NOTE</b>	Provides additional information to emphasize or supplement important points of the main text.

## General Conventions

Convention	Description
Times New Roman	Normal paragraphs are in Times New Roman.
<b>Boldface</b>	Names of files, directories, folders, and users are in <b>boldface</b> . For example, log in as user <b>root</b> .
<i>Italic</i>	Book titles are in <i>italics</i> .
Courier New	Terminal display is in Courier New.

## Table Contents

Content	Description
-	Not applicable
*	A wild card

## Change History

Updates between document issues are cumulative. Therefore, the latest document issue contains all updates made in previous issues.



## **Issue 02 (2015-02-10)**

In section 2.1, the pstInitParam of the value range is modified.

## **Issue 01 (2014-12-20)**

The contents related to the Hi3516D are added.

## **Issue 00B01 (2014-08-11)**

This issue is the first draft release.





# 1 Overview

## 1.1 Scope

The H.265 PC decoding library provided by HiSilicon is the decoding software of high performance, high reliability, and superior compatibility. The decoding library completes the main processes for H.265 decoding and provides flexible and simple APIs for developers. In this case, developers can develop application programs quickly.

The decoding library provides the dynamic library and static library on Windows OS for developers to facilitate application program development. [Table 1-1](#) describes main components of the decoding library.

**Table 1-1** Main components of the decoding library

Component	Name	Description
API	IHWVideo_Typedef.h IHW265Dec_Api.h	In a project, IHWVideo_Typedef.h must be included before IHW265Dec_Api.h in the code for calling the APIs.
32-bit static library	hi_h265_dec_w32.lib	-
32-bit dynamic library	hi_h265_dec_w32.lib hi_h265_dec_w32.dll	-
64-bit static library	hi_h265_dec_w64.lib	-
64-bit dynamic library	hi_h265_dec_w64.lib hi_h265_dec_w64.dll	-
Sample code	hi_h265sample.c	The read file decoding is used as an example to illustrate how to call APIs of the decoding library.

Developers can develop application programs based on the decoding library in multiple environments. The decoding library is compatible with mainstream OSs provided by Microsoft, such as Windows XP, Windows 7, and Windows 8 and PC-oriented CPU chipset



provided by Intel and AMD. [Table 1-2](#) describes the running environments of the decoding library.

**Table 1-2** Running environments of the decoding library

Type	Compatible Configuration	Recommended Configuration	Description
Compiler	Visual Studio 2008 Visual Studio 2010 Visual Studio 2012	Visual Studio 2010	None
OS	Windows XP Windows 7 (32bit) Windows 7 (64bit) Windows 8 (32bit) Windows 8 (64bit)	Windows 7	None
Hardware	Intel Core series CPU Intel Pentium series CPU AMD Ax series CPU	Intel Core i5 series CPU	The decoding library is optimized based on the multimedia command sets, such as SSE, SSE2, SSE3, SSE4, and AVX. On PCs that these command sets are not supported, the decoding library performance is low.

## 1.2 API Format

None

## 1.3 Function List

Function	Function	Page
<a href="#">IHW265D_Create</a>	Create and initialize a decoder handle.	<a href="#">4</a>
<a href="#">IHW265D_Delete</a>	Destroy the decoder handle.	<a href="#">6</a>
<a href="#">IHW265D_GetVersion</a>	Query the decoding library version information.	<a href="#">6</a>
<a href="#">IHW265D_DecodeFrame</a>	Decode an input code stream and output the image.	<a href="#">7</a>
<a href="#">IHW265D_DecodeAU</a>	Decode an input code stream and output the frame image immediately.	<a href="#">10</a>



## 1.4 Function Description Fields

This section describes API reference information from six fields.

Field	Function
Purpose	Describes the major function of an API.
Syntax	Lists the syntax of an API.
Description	Describes the working process of an API.
Parameter	Describes the parameters and attributes of an API.
Return Value	Describes the return values of an API.
Note	Describes the notes for calling an API.

## 1.5 Structure Description Fields

Field	Function
Description	Describes the function of a data structure.
Definition	Describes the definition of a data structure.
Precautions	Lists the precautions of a data structure.



# 2 API Description

## 2.1 IHW265D\_Create

[Purpose]

Create and initialize a decoder handle.

[Syntax]

```
INT32 IHW265D_Create(IH265DEC_HANDLE *phDecoder, IHW265D_INIT_PARAM  
*pstInitParam);
```

[Description]

This API is used to create a decoder handle. During decoding, this API is called and performs the following operations:

Allocate the decoding space and initialize decoder-related variables and status.

Set decoder attributes, such as the maximum image width and height supported by the decoder, the maximum number of reference frames supported by the decoder, decoder thread type, and output image sequence.

Allocate the memory callback function and release the memory callback function and log callback function.

Upper-layer applications can create multiple decoders to ensure multi-channel decoding.

[Parameter]

Parameter	Member	Value Range	Input/Output	Description
phDecoder	-	-	Output	Handle of a decoder
pstInitParam	uiChannelID	-	Input	Channel ID
	iMaxWidth	[8,8192]	Input	Image width (in pixel)
	iMaxHeight	[8,8192]	Input	Image height (in pixel)
	iMaxRefNum	[0,15]	Input	Maximum number of reference frames



Parameter	Member	Value Range	Input/Output	Description
	eThreadType	[0,1]	Input	Thread type Single-thread and multi-thread are supported.
	eOutputOrder	[0,1]	Input	Output sequence Decoding sequence and display sequence are supported.
	pstUserData	-	Input	Input user data For details about the data type, see <a href="#">HW265D_USERDATA</a> .
	MallocFxn	-	Input	Allocation of the memory callback function
	FreeFxn	-	Input	Release of the memory callback function
	LogFxn	-	Input	Log output callback function

[Return Value]

Return Value	Macro Definition	Description
0	IHW265D_OK	The function is successfully called.
0xF0401000	IHW265D_INVALID_ARGUMENT	The input parameter is incorrect.
0xF0401002	IHW265D_INVALID_MAX_REF_PIC	The maximum number of reference frames exceeds the limit.
0xF0401003	IHW265D_INVALID_MALLOC_FCN	The Malloc callback function pointer is invalid.
0xF0401004	IHW265D_INVALID_FREE_FCN	The Free callback function pointer is invalid.
0xF0401006	IHW265D_INVALID_LOG_FCN	The Log callback function pointer is invalid.
0xE0404008	IHW265D_THREAD_ERROR	An error occurs during multi-thread decoding.

[Note]

- The decoding sequence output mode can be used only if the decoding sequence is consistent with the image output sequence. In normal cases, if an image does not contain the B frame, the decoding sequence output mode can be used to shorten the output delay.
- Multi-thread decoding applies to scenarios that have high requirements for single channel decoding performance. The scenarios include high-resolution and high frame



rate and insufficient performance for a single CPU core. Internal overheads are generated during multi-thread scheduling. Therefore, multi-thread decoding is not recommended in multi-channel scenarios.

- You must ensure sufficient system memory when creating a decoder for decoding oversized pictures. Otherwise, the decoder fails to be created.

## 2.2 IHW265D\_Delete

[Purpose]

Destroy the decoder handle.

[Syntax]

```
INT32 IHW265D_Delete( IHW265DEC_HANDLE hDecoder );
```

[Description]

After decoding is complete, this API is used to destroy the memory space allocated for the decoder to prevent memory leakage.

[Parameter]

Parameter	Member	Value Range	Input/Output	Description
hDecoder	-	-	Input	Decoder handle to be destroyed

[Return Value]

Return Value	Macro Definition	Description
0	IHW265D_OK	The function is successfully called.
0xF0401000	IHW265D_INVALID_ARGUMENT	The parameter is invalid.
0xF0401001	IHW265D_DECODER_NOT_CREATE	The decoder is not created.

[Note]

After the handle is destroyed, set **hDecoder** to **0** manually.

## 2.3 IHW265D\_GetVersion

[Purpose]

Query the decoding library version information.

[Syntax]

```
INT32 IHW265D_GetVersion(IHWVIDEO_ALG_VERSION_STRU *pstVersion);
```



[Description]

This API is used to query the decoding library version information.

[Parameter]

Parameter	Member	Value Range	Input/Output	Description
pstVersion	cVersionChar	-	Output	Decoding library version number
	cReleaseTime	-	Output	Compiling time
	uiCompileVersion	-	Output	Compiler version number

[Return Value]

Return Value	Macro Definition	Description
0	IHW265D_OK	The function is successfully called.
0xF0401000	IHW265D_INVALID_ARGUMENT	The parameter is invalid.

[Note]

None

## 2.4 IHW265D\_DecodeFrame

[Purpose]

Decode an input code stream and output the image.

[Syntax]

```
INT32 IHW265D_DecodeFrame(IH265DEC_HANDLE hDecoder, IH265DEC_INARGS  
*pstInArgs, IH265DEC_OUTARGS *pstOutArgs);
```

[Description]

This API supports stream decoding. For continuous and linear H.265 code streams using 000001 as the separator of nalu, users can configure any length for decoding.

[Parameter]

Parameter	Member	Value Range	Input/Output	Description
hDecoder	-	-	Input	Handle of a decoder
pstInArgs	pStream	-	Input	Start address of a code stream



Parameter	Member	Value Range	Input/Output	Description
	uiStreamLen	-	Input	Code stream length (in the unit of byte)
	uiTimeStamp	-	Input	Time stamp
	eDecodeMode	-	Input	Decoding mode In normal cases, IH265D_DECODE is used. When decoding the file end, IH265D_DECODE_END is used.
pstOutArgs	uiChannelID	-	Output	Channel ID
	uiBytsConsumed	-	Output	Number of consumed bytes
	uiTimeStamp	-	Output	Time stamp
	eFrameType	[0,2]	Output	Frame type
	eDecodeStatus	-	Output	Status of the decoder
	uiDecWidth	-	Output	Image width
	uiDecHeight	-	Output	Image height
	uiYStride	-	Output	Luminance component stride
	uiUVStride	-	Output	Chrominance component stride
	pucOutYUV	-	Output	Address of the Y, U, and V components
	pstUserData	-	Output	User data
	uiCodingBytesOfCurFrm	-	Output	Source code stream length (Number of bytes)
	uiAspectRatioIdc	-	Output	Aspect ratio
	uiSarWidth	-	Output	Aspect ratio width It is valid only when <b>uiAspectRatioIdc</b> is set to <b>255</b> .
	uiSarHeight	-	Output	Aspect ratio height It is valid only when <b>uiAspectRatioIdc</b> is set to <b>255</b> .
	uiVpsNumUnitsInTick	-	Output	Frame rate information
	uiVpsTimeScale	-	Output	





Parameter	Member	Value Range	Input/Output	Description
	stCuOutInfo	-	Output	Current output frame information, including number of various types of CUs in a frame
	bIsError	[0,1]	Output	Frame error flag

[Return Value]

Return Value	Macro Definition	Description
0	IHW265D_OK	The function is successfully called. A frame image is output.
1	IHW265D_NEED_MORE_BITS	The remained code stream is insufficient for decoding data of a frame. More code streams need to be configured.  This value is returned only if the decoding mode is IH265D_DECODE.
0xF0401000	IHW265D_INVALID_ARGUMENT	The parameter is invalid.
0xF0401001	IHW265D_DECODER_NOT_CREATE	The decoder is not created.
0xF0402008	IHW265D_STREAMBUF_NULL	The stream address is empty.

[Note]

Note the following when calling this function:

- This function can be used to decode a code stream of any length. The input parameters include the code stream buffer address **pStream** and the number of bytes in a code stream **uiStreamLen**. In normal cases, the decoding mode is set to IH265D\_DECODE. If IH265D\_GETDISPLAY is returned, a frame image is output. Users must process images stored in pstOutArgs in time.
- If a code stream is insufficient for decoding a frame, more code streams need to be input for decoding.
- If a code stream includes multiple frames, this function must be called repeatedly to decode remained code streams after a frame image is decoded.
- After the code streams are decoded, the decoding mode is set to IH265D\_DECODE\_END to enter the flush mode to clear remain images in the decoder. This function needs to be called repeatedly until IH265D\_NO\_PICTURE is returned, which indicates that no remaining image exists in the decoder.



- This function provides the function of transparent transmission of the time stamp. The input time stamp is stored in the image structure after decoding and output with the decoded image.

## 2.5 IHW265D\_DecodeAU

### [Purpose]

Decode an input code stream and output the frame image immediately.

### [Syntax]

```
INT32 IHW265D_DecodeAU(IH265DEC_HANDLE hDecoder, IH265DEC_INARGS *pstInArgs,  
IH265DEC_OUTARGS *pstOutArgs);
```

### [Description]

This API supports decoding by frame. The input stream must contain only one frame of H.265 code streams and output immediately based on the decoding sequence.

### [Parameter]

Parameter	Member	Value Range	Input/Output	Description
hDecoder	-	-	Input	Handle of a decoder
pstInArgs	pStream	-	Input	Start address of a code stream
	uiStreamLen	-	Input	Code stream length (in the unit of byte)
	uiTimeStamp	-	Input	Time stamp
	eDecodeMode	-	Input	Decoding mode (Invalid)
pstOutArgs	uiChannelID	-	Output	Channel ID
	uiBytsConsumed		Output	Number of consumed bytes
	uiTimeStamp		Output	Time stamp
	eFrameType		Output	Frame type
	eDecodeStatus		Output	Status of the decoder
	uiDecWidth		Output	Image width
	uiDecHeight		Output	Image height
	uiYStride		Output	Luminance component stride
	uiUVStride		Output	Chrominance component stride
	pucOutYUV		Output	Output of the Y, U, V component addresses
	pstUserData		Output	User data



Parameter	Member	Value Range	Input/Output	Description
	uiCodingBytesOfCurFrm	-	Output	Source code stream length (Number of bytes)
	uiAspectRatioIdc	-	Output	Aspect ratio
	uiSarWidth	-	Output	Aspect ratio width It is valid only when <b>uiAspectRatioIdc</b> is set to <b>255</b> .
	uiSarHeight	-	Output	Aspect ratio height It is valid only when <b>uiAspectRatioIdc</b> is set to <b>255</b> .
	uiVpsNumUnitsInTick	-	Output	Frame rate information
	uiVpsTimeScale	-	Output	
	stCuOutInfo	-	Output	Current output frame information, including number of various types of CUs in a frame
	bIsError	[0,1]	Output	Frame error flag

[Return Value]

Return Value	Macro Definition	Description
0	IHW265D_OK	The function is successfully called.
0xF0401000	IHW265D_INVALID_ARGUMENT	The parameter is invalid.
0xF0401001	IHW265D_DECODER_NOT_CREATE	The decoder is not created.
0xF0402008	IHW265D_STREAMBUF_NULL	The stream address is empty.

[Note]

Note the following when calling this function:

- The input code stream of this function must contain only one frame of video streams. The video streams exceeding or less than one frame are considered as incorrect code streams.
- This function supports only the decoding sequence output mode, and the image is output immediately after being decoded. This function cannot be used for video streams that have different decoding sequence and display sequence.



- This function provides the function of transparent transmission of the time stamp. The input time stamp is stored in the image structure after decoding and output with the decoded image.



# 3 Data Type and Structure

## 3.1 Description of Common Data Types

The following are the main data types used by an API:

```
typedef signed char      INT8;  
typedef signed short     INT16;  
typedef signed int       INT32;  
typedef unsigned char    UINT8;  
typedef unsigned short   UINT16;  
typedef unsigned int     UINT32;  
typedef      __int64     INT64;  
typedef unsigned __int64  UINT64;  
typedef void*           IH265DEC_HANDLE;
```

The following are the main function pointer types used by an API:

```
typedef void (* IHWVIDEO_ALG_MALLOC_FXN)( UINT32 uiChannelID, UINT32  
    uiSize);  
typedef void (* IHWVIDEO_ALG_FREE_FXN)( UINT32 uiChannelID, void *pMem);  
typedef void (* IHWVIDEO_ALG_LOG_FXN)( UINT32 uiChannelID,  
    IHWVIDEO_ALG_LOG_LEVEL eLevel, INT8 *pszMsg, ...);
```

## 3.2 Description of the Data Structure

### 3.2.1 HW265D\_THREADTYPE

[Description]

Thread type enumeration

[Syntax]

```
typedef enum tagHW265D_THREADTYPE  
{  
    IH265D_SINGLE_THREAD = 0, /* Single-thread */  
    IH265D_MULTI_THREAD   /* Multi-thread */  
} HW265D_THREADTYPE;
```

[Note]



None

### 3.2.2 HW265D\_OUTPUTORDER

[Description]

Output sequence enumeration

[Syntax]

```
typedef enum tagHW265D_OUTPUTORDER
{
    IH265D_DECODE_ORDER = 0,          /*output in the decoding sequence*/
    IH265D_DISPLAY_ORDER,             /*output in the display sequence*/
} HW265D_OUTPUTORDER;
```

[Note]

None

### 3.2.3 HW265D\_DECODEMODE

[Description]

Decoding mode

In a normal decoding process, some code streams and images are remained in the decoder. After the decoding is complete, these images must be output forcibly.

[Syntax]

```
typedef enum tagHW265D_DECODEMODE
{
    IH265D_DECODE = 0,                /* Normal decoding */
    IH265D_DECODE_END                 /* The decoding completes, and the decoder is required to
    output remaining images.*/
} HW265D_DECODEMODE;
```

[Note]

None

### 3.2.4 HW265D\_FRAME\_TYPE

[Description]

Frame type

[Syntax]

```
typedef enum tagHW265D_FRAME_TYPE
{
    IH265D_FRAME_I = 0,
    IH265D_FRAME_P,
    IH265D_FRAME_B,
    IH265D_FRAME_UNKNOWN
} HW265D_FRAME_TYPE;
```

[Note]



None

### 3.2.5 HW265D\_DECODESTATUS

[Description]

Status of the decoder

[Syntax]

```
typedef enum tagHW265D_DECODESTATUS
{
    IH265D_GETDISPLAY = 0,          /* A frame has been decoded and output, which can be
    displayed. */
    IH265D_NEED_MORE_BITS,         /* The decoder does not output an image, and more code
    streams are required. */
    IH265D_NO_PICTURE,             /* The decoding is complete, and all images are output.*/
    IH265D_ERR_HANDLE              /* Handle error */
} HW265D_DECODESTATUS;
```

[Note]

None

### 3.2.6 IHW265D\_INIT\_PARAM

[Description]

Decoder initialization parameters

[Syntax]

```
/* Decoder initialization parameters*/
typedef struct tagIHW265D_INIT_PARAM
{
    UINT32  uiChannelID;            /*Channel ID*/
    INT32    iMaxWidth;             /*Maximum width */
    INT32    iMaxHeight;           /*Maximum height */
    INT32    iMaxRefNum;           /* Maximum number of reference frames*/
    HW265D_THREADTYPE eThreadType; /*Thread type*/
    HW265D_OUTPUTORDER eOutputOrder; /* Output sequence. It is valid only for
    decoding frames. By default, decoding sequence is used for decoding AUs. */
    HW265D_USERDATA *pstUserData; /* User data */
    IHWVIDEO_ALG_MALLOC_FXN MallocFxn; /*Allocation of the memory callback function*/
    IHWVIDEO_ALG_FREE_FXN FreeFxn; /*Release of the memory callback function*/
    IHWVIDEO_ALG_LOG_FXN LogFxn; /* Log callback function */
} IHW265D_INIT_PARAM;
```

[Note]

None

### 3.2.7 IH265DEC\_INARGS

[Description]

Decoder input information

[Syntax]



```
/* Code stream, time stamp, and decoding mode inputted by the decoder*/
typedef struct tagIH265DEC_INARGS
{
    UINT8  *pStream;                /*Stream address*/
    UINT32 uiStreamLen;             /*Stream length*/
    UINT64 uiTimeStamp;             /*Time stamp*/
    HW265D_DECODEMODE eDecodeMode; /* Decoding mode 0: Normal mode; 1: Flush mode*/
} IH265DEC_INARGS;
```

[Note]

None

### 3.2.8 IH265DEC\_OUTARGS

[Description]

Decoder output information

[Syntax]

```
/*Decoder output information*/
typedef struct tagIH265DEC_OUTARGS
{
    UINT32      uiChannelID;        /*Channel ID*/
    UINT32      uiBytsConsumed;     /* Number of consumed bytes*/
    UINT64      uiTimeStamp;        /*Time stamp*/
    HW265D_FRAMETYPE eFrameType;    /*Output frame type*/
    HW265D_DECODESTATUS eDecodeStatus; /*Status of the decoder*/
    UINT32      uiDecWidth;         /* Image width*/
    UINT32      uiDecHeight;        /* Image height*/
    UINT32      uiYStride;          /* Image luminance stride*/
    UINT32      uiUVStride;         /* Image chrominance stride*/
    UINT8       *pucOutYUV[3];      /* Addresses of the Y, U, and V components*/
    HW265D_USERDATA *pstUserData;   /* User data */

    UINT32      uiCodingBytesOfCurFrm; /* Original code stream length (number of
    bytes) of the current frame*/
    // vui information
    UINT32      uiAspectRatioIdc;    /* Aspect ratio*/
    UINT32      uiSarWidth;
    UINT32      uiSarHeight;
    // vps information
    UINT32      uiVpsNumUnitsInTick; /* Frame rate information*/
    UINT32      uiVpsTimeScale;
    // cuinfo
    CU_OUTPUT_INFO stCuOutInfo;      /* */
    // errorinfo
    BOOL32      bIsError;            /* Frame error flag*/
} IH265DEC_OUTARGS;
```

### 3.2.9 HW265D\_USERDATA

[Description]

User data

[Syntax]





```
/* User data */
typedef struct tagHW265D_USERDATA
{
    UINT32 uiUserDataType;      /*User data type*/
    UINT32 uiUserDataSize;      /*User data length*/
    UINT8  *pucData;            /* User data buffer*/

    struct tagHW265D_USERDATA *pNext; /* Point to the next segment of user data */
}HW265D_USERDATA;
```

[Note]

None

### 3.2.10 IHWVIDEO\_ALG\_VERSION\_STRU

[Description]

Decoding library version information

[Syntax]

```
/*Decoding library version information*/
typedef struct tagIHWVIDEO_ALG_VERSION
{
    INT8  cVersionChar[IHWVIDEO_ALG_VERSION_LENGTH]; /*Version number*/
    INT8  cReleaseTime[IHWVIDEO_ALG_TIME_LENGTH];      /* Compiling time*/
    UINT32 uiCompileVersion;                             /*Compiler version number*/
}IHWVIDEO_ALG_VERSION_STRU;
```

[Note]

None

### 3.2.11 CU\_OUTPUT\_INFO

[Description]

Number of various types of CUs in each frame image

[Syntax]

```
/* Number of various types of CUs in each frame image*/
typedef struct tagCU_OUTPUT_INFO
{
    UINT32 uiCuNumIntra4;      /* intra 4x4 CU number */
    UINT32 uiCuNumIntra8;      /* intra 8x8 CU number */
    UINT32 uiCuNumIntra16;     /* intra 16x16 CU number */
    UINT32 uiCuNumIntra32;     /* intra 32x32 CU number */
    UINT32 uiCuNumIntra64;     /* intra 64x64 CU number */
    UINT32 uiCuNumPcm4;        /* IPCM 4x4 CU number */
    UINT32 uiCuNumPcm8;        /* IPCM 8x8 CU number */
    UINT32 uiCuNumPcm16;       /* IPCM 16x16 CU number */
    UINT32 uiCuNumPcm32;       /* IPCM 32x32 CU number */
    UINT32 uiCuNumPcm64;       /* IPCM 64x64 CU number */
    UINT32 uiCuNumInter8;      /* inter 8x8 CU number */
    UINT32 uiCuNumInter16;     /* inter 16x16 CU number */
    UINT32 uiCuNumInter32;     /* inter 32x32 CU number */
    UINT32 uiCuNumInter64;     /* inter 64x64 CU number */
}
```



```
    UINT32 uiCuNumSkip8;      /* skip 8x8 CU number */  
    UINT32 uiCuNumSkip16;     /* skip 16x16 CU number */  
    UINT32 uiCuNumSkip32;     /* skip 32x32 CU number */  
    UINT32 uiCuNumSkip64;     /* skip 64x64 CU number */  
}CU_OUTPUT_INFO;
```

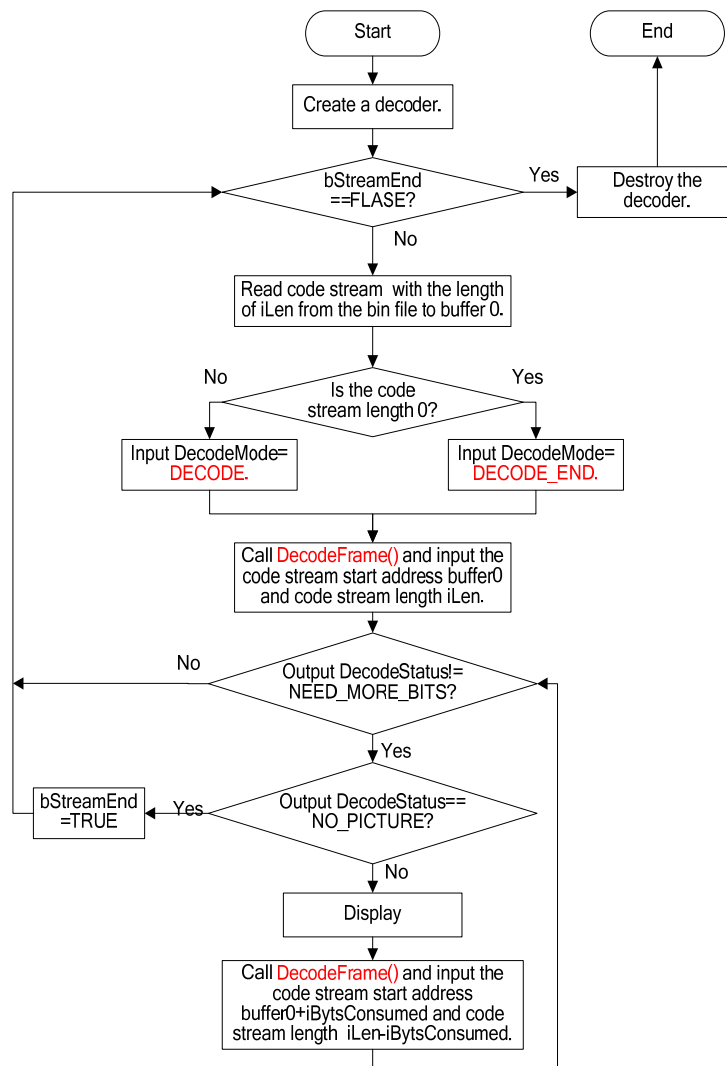
[Note]

None

# 4 API Application Instance

## 4.1 Stream Decoding Process

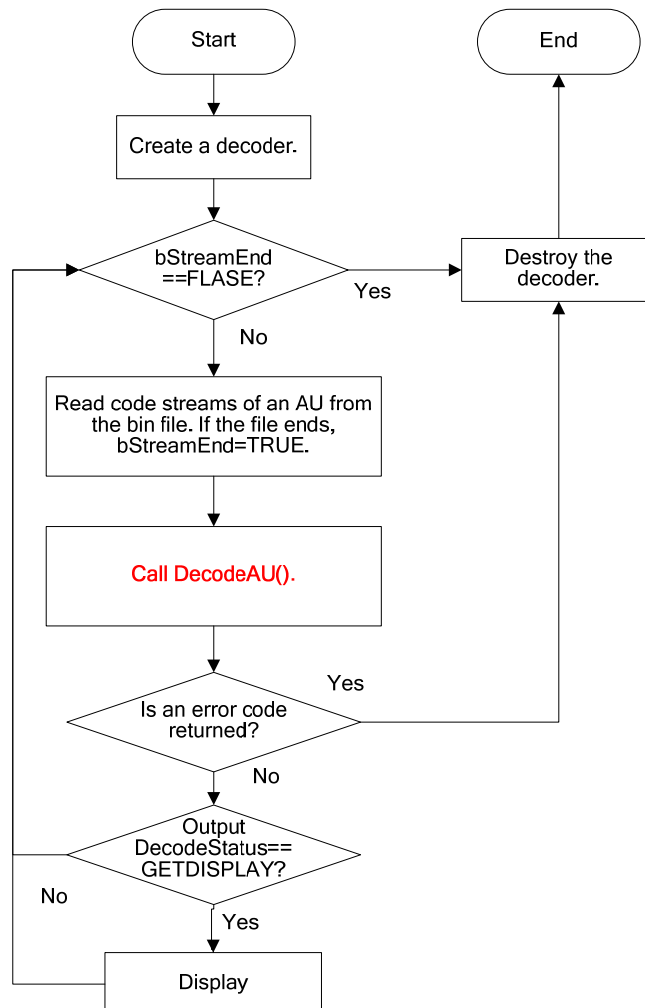
Figure 4-1 Stream decoding process





## 4.2 Frame Decoding Process

Figure 4-2 Frame decoding process



## 4.3 Program Instance

```
void *HW265D_Malloc(UINT32 channel_id, UINT32 size)
{
    return (void *)malloc(size);
}

void HW265D_Free(UINT32 channel_id, void * ptr)
{
    free(ptr);
}

void HW265D_Log( UINT32 channel_id, IHWVIDEO_ALG_LOG_LEVEL eLevel, INT8
```



```
*p_msg, ...)\n{\n    printf("%s.\\n", p_msg);\n}\n\nINT32 H265DecLoadAU(UINT8* pStream, UINT32 iStreamLen, UINT32* pFrameLen)\n{\n    UINT32 i;\n    UINT32 state = 0xffffffff;\n    BOOL32 bFrameStartFound=0;\n\n    *pFrameLen = 0;\n    if( NULL == pStream || iStreamLen <= 4)\n    {\n        return -1;\n    }\n\n    for( i = 0; i < iStreamLen; i++)\n    {\n        if( (state & 0xFFFFFFF7E) >= 0x100 &&\n            (state & 0xFFFFFFF7E) <= 0x13E )\n        {\n            if( 1 == bFrameStartFound )\n            {\n                if( (pStream[i+1]>>7) == 1)\n                {\n                    *pFrameLen = i - 4;\n                    return 0;\n                }\n            }\n            else\n            {\n                bFrameStartFound = 1;\n            }\n        }\n        /*find a vps, sps, pps*/\n        if( (state&0xFFFFFFF7E) == 0x140 ||\n            (state&0xFFFFFFF7E) == 0x142 ||\n            (state&0xFFFFFFF7E) == 0x144)\n        {\n            if(1 == bFrameStartFound)\n            {\n                *pFrameLen = i - 4;\n                return 0;\n            }\n        }\n    }\n}
```



```
        }
        else
        {
            bFrameStartFound = 1;
        }
    }

    state = (state << 8) | pStream[i];
}

*pFrameLen = i;
return -1;
}

int main(int argc, unsigned char** argv)
{
    FILE *fpInFile = NULL;
    FILE *fpOutFile = NULL;
    INT32 iRet = 0;
    UINT8 *pInputStream = NULL, *pStream;
    UINT32 uiChannelId = 0;
    UINT32 iFrameIdx = 0;
    BOOL32 bStreamEnd = 0;
    INT32 iFileLen;

    IH265DEC_HANDLE hDecoder = NULL;
    IHW265D_INIT_PARAM stInitParam = {0};
    IH265DEC_INARGS stInArgs;
    IH265DEC_OUTARGS stOutArgs = {0};

    /* open input stream file and output yuv file */
    fpInFile = fopen(argv[1], "rb");
    fpOutFile = fopen(argv[2], "wb");
    if (NULL == fpInFile || NULL == fpOutFile)
    {
        fprintf(stderr, "Unable to open h265 stream file %s or yuv file %s.\n",
            argv[1], argv[2]);
        goto exitmain;
    }

    printf("decoding file: %s...\n", argv[1]);
    printf("save yuv file: %s...\n", argv[2]);

    /* malloc stream buffer */
```



```
fseek( fpInFile, 0, SEEK_END);
iFileLen = ftell( fpInFile);
fseek( fpInFile, 0, SEEK_SET);
pInputStream = (unsigned char *) malloc(iFileLen);
if (NULL == pInputStream)
{
    fprintf(stderr, "Malloc failed! \n");
    goto exitmain;
}

/* create decode handle */
stInitParam.uiChannelID      = 0;
    stInitParam.iMaxWidth      = 1920;
    stInitParam.iMaxHeight     = 1088;
    stInitParam.iMaxRefNum     = 2;
    stInitParam.eThreadType    = IH265D_SINGLE_THREAD;    // or
IH265D_MULTI_THREAD;
    stInitParam.eOutputOrder   = IH265D_DECODE_ORDER;    // or
IH265D_DISPLAY_ORDER;
    stInitParam.MallocFxn      = HW265D_Malloc;
    stInitParam.FreeFxn        = HW265D_Free;
    stInitParam.LogFxn         = HW265D_Log;
iRet = IH265D_Create(&hDecoder, &stInitParam);
if (IHW265D_OK != iRet)
{
    fprintf(stderr, "Unable to create decoder.\n");
    goto exitmain;
}

/* read H.265 stream to stream buffer */
fread(pInputStream, 1, iFileLen, fpInFile);
pStream = pInputStream;

/* decode process */
while (!bStreamEnd)
{
    INT32 iNaluLen;
    H265DecLoadAU(pStream, iFileLen, &iNaluLen);

    stInArgs.eDecodeMode = (iNaluLen>0) ? IH265D_DECODE :
IH265D_DECODE_END;
    stInArgs.pStream      = pStream;
    stInArgs.uiStreamLen = iNaluLen;
```



```
pStream += iNaluLen;
iFileLen -= iNaluLen;

stOutArgs.eDecodeStatus = -1;
stOutArgs.uiBytsConsumed = 0;

while(stOutArgs.eDecodeStatus != IH265D_NEED_MORE_BITS)
{
    // when decoder is empty, exit loop;
    if(stOutArgs.eDecodeStatus == IH265D_NO_PICTURE)
    {
        bStreamEnd = 1;
        break;
    }

    // output one picture
    if (stOutArgs.eDecodeStatus == IH265D_GETDISPLAY)
    {
        // save yuv to output file
        if (fpOutFile != NULL)
        {
            UINT32 i;
            for (i=0; i<stOutArgs.uiDecHeight; i++)
            {
                fwrite(stOutArgs.pucOutYUV[0]+i*stOutArgs.uiYStride, 1,
                    stOutArgs.uiDecWidth, fpOutFile);
            }
            for (i=0; i<((stOutArgs.uiDecHeight)>>1); i++)
            {
                fwrite(stOutArgs.pucOutYUV[1]+i*stOutArgs.uiUVStride, 1,
                    stOutArgs.uiDecWidth>>1, fpOutFile);
            }
            for (i=0; i<((stOutArgs.uiDecHeight)>>1); i++)
            {
                fwrite(stOutArgs.pucOutYUV[2]+i*stOutArgs.uiUVStride, 1,
                    stOutArgs.uiDecWidth>>1, fpOutFile);
            }
        }
        iFrameIdx++;
    }

    // continue to decode the rest of stream
}
```





```
        stInArgs.pStream      += stOutArgs.uiBytsConsumed;
        stInArgs.uiStreamLen  -= stOutArgs.uiBytsConsumed;

        iRet = IHW265D_DecodeFrame(hDecoder, &stInArgs, &stOutArgs);
        if ((iRet != IHW265D_OK) && (iRet != IHW265D_NEED_MORE_BITS))
        {
            if (0 == iFileLen)
            {
                bStreamEnd = 1;
            }
            break;
        }
    }

exitmain:
    if (fpInFile != 0)
        fclose(fpInFile);

    if (fpOutFile != 0)
        fclose(fpOutFile);

    if (hDecoder != NULL)
    {
        IHW265D_Delete(hDecoder);
        hDecoder = NULL;
    }

    if (pInputStream != NULL)
    {
        free(pInputStream);
        pInputStream = NULL;
    }

    return 0;
}
```