# API Document
# S3C6400/6410 Multi-Format Codec

**S3C6400/6410**

**August 29, 2008**

**(Preliminary) REV 3.20**

# Important Notice

**S3C6400/6410 RISC Microprocessor**
**API Document**

Samsung Electronics Co., Ltd.
San #24 Nongseo-Dong, Giheung-Gu
Yongin-City Gyeonggi-Do, Korea
446-711

Home Page: http://www.samsungsemi.com/

E-Mail: mobilesol.cs@samsung.com

Printed in the Republic of Korea

# Revision History

| Revision No | Description of Change | Refer to | Author(s) | Date |
|---|---|---|---|---|
| 1.00 | Initial Release for review | - | Simon Chun | Jul. 23, 2007 |
| 2.10 | Encoding case and linux sample code are added | | Jiun Yu | Aug. 31, 2007 |
| 2.11 | Configuration macros are added | | Jiun Yu | Sep. 1, 2007 |
| 2.20 | Description about config stream in LINE_BUF and API are added | | Jiun Yu | Oct. 15, 2007 |
| 2.21 | | | Jiun Yu | Dec. 21, 2007 |
| 2.30 | Rotation function is added | | Jiun Yu | Jan. 24, 2008 |
| 2.40 | Encoding option are added | | Jiun Yu | Mar. 31, 2008 |
| 3.00 | Supporting S3C6400 and 6410 | | Jiun Yu | April 14, 2008 |
| 3.10 | Supporting Hybrid divx decoder | | Jiun Yu | July 5, 2008 |
| 3.20 | MPEG4_DEC_SETCONF_CACHE_XXX, MPEG4_DEC_SETCONF_PADDING_SIZE was added for Hybrid divx decoder | | Jiun Yu | August 29, 2008 |

## Contents

## Figures

## Tables

# 1 Introduction

## 1.1 Purpose

This document is prepared for the purpose of describing the S3C6400/6410 HW codec (MFCv1.0) API so that users can implement their multimedia application easily.

## 1.2 Scope

The scope of this document is to describe

- Software architecture of Encoder/Decoder
- Data structures and API used for Encoder/Decoder
- Usage example of Encoder/Decoder

## 1.3 Intended Audience

| Intended Audience | Tick whenever Applicable |
|---|---|
| Project Manager | Yes |
| Project Leader | Yes |
| Project Team Member | Yes |
| Test Engineer | Yes |

## 1.4 Definitions, Acronyms, and Abbreviations

| Abbreviations | Description |
|---|---|
| MFC | Multi-Format Codec (HW codec in S3C6400/6410 Samsung AP) |
| API | Application Program Interface |

## 1.5 References

| Number | Reference | Description |
|---|---|---|
| 1 | S3C6400 Datasheet | MFC H/W data sheet |
| 2 | S3C6400WM60MfcLib_API_REV1.32_20071228.doc | API specification of Windows Mobile 6.0 MFC device driver |

# 2 Software Architecture

## 2.1 Overview

The S3C6400/6410 Multi-Format Codec's Encoder/Decoder SW package consists of two parts:

- S3C6400/6410 Multi-Format Codec Encoder/Decoder Library (located in user region)
- S3C6400/6410 Multi-Format Codec Device Driver (located in OS region)

The software architecture is shown in **Fig - S3C6400/6410 MFC Encoder/Decoder SW Architecture**

.

User's multimedia application can call the API functions provided by S3C6400/6410 Multi-Format Codec Encoder/Decoder Library to encode and decode the multimedia data. Moreover, it can call the OS(WIN32, VFS of linux) file I/O functions directly because those functions are also exposed in the user region.

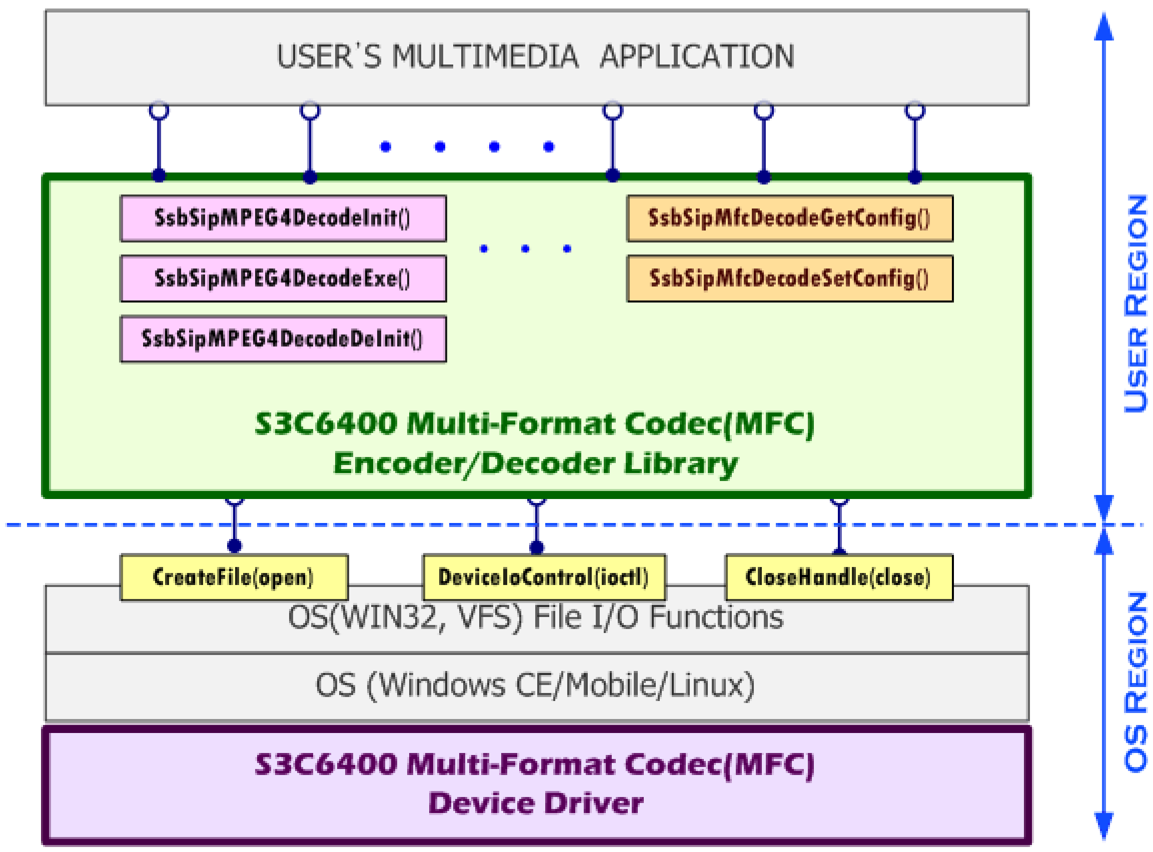
USER'S MULTIMEDIA APPLICATION
SsbSipMPEG4DecodeInit()
SsbSipMPEG4DecodeExe()
SsbSipMPEG4DecodeDeInit()
SsbSipMfcDecodeGetConfig()
SsbSipMfcDecodeSetConfig()
S3C6400 Multi-Format Codec(MFC) Encoder/Decoder Library
CreateFile(open)
DeviceIoControl(ioctl)
CloseHandle(close)
OS(WIN32, VFS) File I/O Functions
OS (Windows CE/Mobile/Linux)
S3C6400 Multi-Format Codec(MFC) Device Driver
USER REGION
OS REGION

**Fig - S3C6400/6410 MFC Encoder/Decoder SW Architecture**

## 2.2 Decoding

In decoding process, the S3C6400/6410 MFC HW codec supports two modes (LINE_BUF and RING_BUF modes) for the input stream.

## 2.2.1 Decoding in LINE_BUF mode

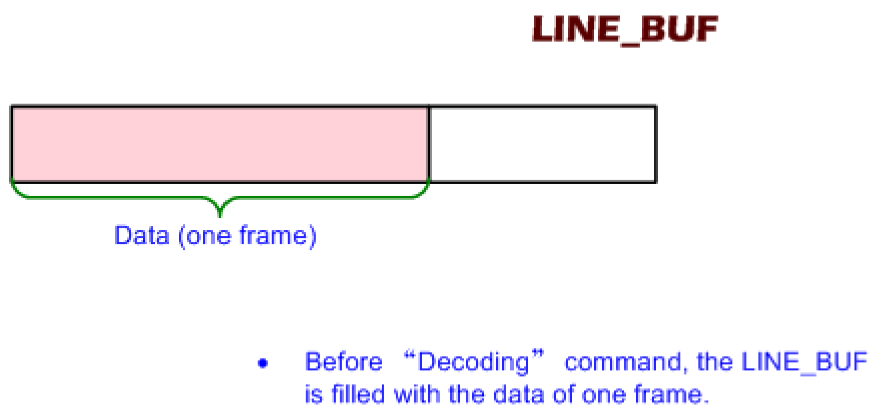
LINE_BUF
Data (one frame)
Before "Decoding" command, the LINE_BUF is filled with the data of one frame.

**Fig - LINE_BUF in decoding**

In this mode, the application needs to fill the input buffer with the video stream of the exact size of one frame.
LINE_BUF mode supports MPEG4/H.263, H.264 and VC-1 decoding.

| PIC # | Values | Stream bytes | Type |
|---|---|---|---|
| 1 | **Visual Object Sequence** | 00 00 01 **B0** 03 | CONFIG Stream |
| | **Visual Object** | 00 00 01 **B5** 09 | |
| | **Video Object** | 00 00 01 **00** | |
| | **Visual Object Layer** | 00 00 01 **20** 00 86 … 8F | |
| | **User Data** | 00 00 01 **B2** CC CC … 63 | |
| | **Video Object Plane (I-picture)** | 00 00 01 **B6** 10 00 ... | |
| 2 | **Video Object Plane (P-picture)** | 00 00 01 **B6** 50 7D ... | VOP Stream |
| 3 | **Video Object Plane (P-picture)** | 00 00 01 **B6** 50 FA ... | VOP Stream |
| 4 | **Video Object Plane (P-picture)** | 00 00 01 **B6** 50 E) ... | VOP Stream |
| ... | ... | ... ... | |
| 467 | **Video Object Plane (I-picture)** | 00 00 01 **B6** 10 00 ... | VOP Stream |
| 468 | **Video Object Plane (P-picture)** | 00 00 01 **B6** 57 54 ... | VOP Stream |

**Table - MPEG4 video stream (Example)**

In , the CONFIG stream (yello color) is introduced to the MPEG4 decoder for the configuration. It consists of several stream data which are Visual Object Sequence, Visual Object, Video Object, Visual Object Layer and User Data.
The VOP streams (pink color) are introduced individually to the decoder for obtaining decoded YUV420 frame.
When it happens to have the CONFIG stream in the middle of the VOP streams, it should be merged with the next VOP stream and then introduced to decoder for decoding.

| PIC # | Values | Stream bytes | Type |
|---|---|---|---|
| 1 | Video Header | 00 00 80 02 08 0C ... | CONFIG Stream |
| | Video Data (I-picture) | | |
| 2 | Video Header | 00 00 80 0A 0A 10 ... | VOP Stream |
| | Video Data (P-picture) | | |
| 3 | Video Header | 00 00 80 12 0A 10 ... | VOP Stream |
| | Video Data (P-picture) | | |
| ... | ... | ... | VOP Stream |
| | ... | ... ... | |
| 148 | Video Header | 00 00 82 0A 0A 10 ... | VOP Stream |
| | Video Data (P-picture) | | |

**Table - H.263 video stream (Example)**

In case of H.263, each compressed video frame has its header. The video frame of I-picture is used as initializing CONFIG stream (yellow color) for the H.263 decoding.

| PIC # | NAL Unit Type | Stream bytes (Example) | Type |
|---|---|---|---|
| 1 | Sequence Parameter Set (SPS) | 00 00 00 01 **27** 42 ……… 7C 04 | CONFIG Stream |
| | Picture Parameter Set (PPS) | 00 00 00 01 **28** CE 09 C8 | |
| | Suppl. Enhancement Info. (SEI) | 00 00 00 01 **26** 05 … 80 | |
| | Coded slice (I) | 00 00 00 01 **25** B8 … AF 78 | |
| 2 | Coded slice (P) | 00 00 00 01 **21** E1 … 98 | VIDEO Stream |
| 3 | Coded slice (P) | 00 00 00 01 **21** E2 … E0 | VIDEO Stream |
| 4 | Coded slice (P) (multi-slice 1) | 00 00 00 01 **21** E3 … E0 | VIDEO Stream |
| | Coded slice (P) (multi-slice 2) | 00 00 00 01 **21** E3 … E8 | |
| ... | ... ... | ... ... | |
| 305 | Coded slice (I) | 00 00 00 01 **25** B8 … EA 62 | VIDEO Stream |
| 306 | Coded slice (P) | 00 00 00 01 **21** E2 … 24 | VIDEO Stream |

**Table - H.264 video stream (Example)**

In , the CONFIG stream (yellow color) is introduced to the H.264 decoder for the configuration. Note that it is including the first I-slice. It consists of SPS, PPS, SEI and first I-slice.
The VIDEO streams (pink color) are introduced individually to the decoder for obtaining decoded YUV420 frame.
When it happens to have the SPS, PPS and/or SEI NALs in the middle of the VIDEO streams, it should be merged with the next VIDEO stream(commonly it is I-slice) and then introduced to decoder for decoding.
H.264 standard supports the multi-sliced NALs. The PIC # 7 in  shows the multi-sliced NALs. The MFC requires that the input video stream should be complete one picture in RING_BUF mode. Multiple slices are put together in the input buffer if they are part of one picture.

**[ NOTE ]**

**For the MPEG4/H.263/H.264 decoding, CONFIGURATION call is followed by the procedure of 'input buffer fill' with the next VOP stream. Then DECODE call comes for decoding it.**

**Multi-sliced NALs need to be put together in H.264 decoding if they are part of one picture.**

### 2.2.2 Decoding in RING_BUF mode

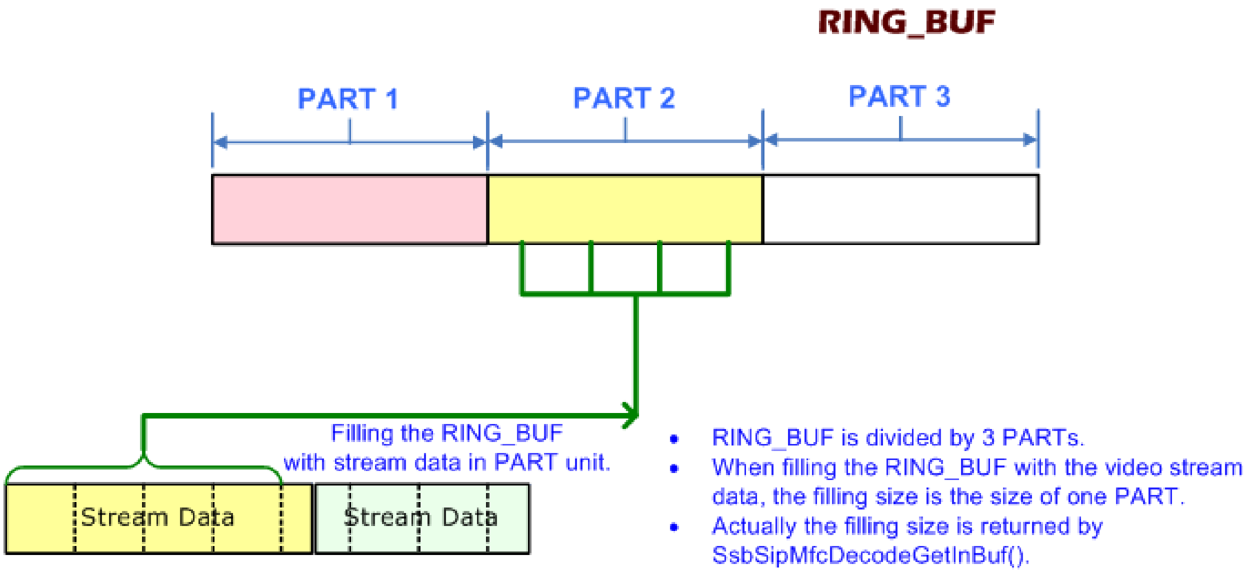
RING_BUF
PART 1
PART 2
PART 3
Filling the RING_BUF with stream data in PART unit.
Stream Data
Stream Data
RING_BUF is divided by 3 PARTs.
When filling the RING_BUF with the video stream data, the filling size is the size of one PART.
Actually the filling size is returned by SsbSipMfcDecodeGetInBuf().

**Fig - RING_BUF in decoding**

In this mode, the application needs to fill the input buffer with the video stream of the size of PART. The size of PART is determined by the device driver.

### 2.2.3 Comparison of LINE_BUF and RING_BUF mode

| | LINE_BUF | RING_BUF |
|---|---|---|
| **File Format** | mp4, 3g2, mov, avi | m4v, 264, wmv, rcv |
| **Decoder Algorithm** | MPEG4, H.263, H.264, VC-1 | MPEG4, H.263, H.264, VC-1 |
| **Stream Parser** | External parser for mp4/3g2/mov/avi is required | Internal MFC parser is used (The external parser is required for wmv.) |
| **Buffer Fill** | One buffer fill per one DECODE call. Filling size is varying & determined by the current frame. | One buffer fill per several DECODE call. Filling size is fixed & pre-determined. |

| | | |
|---|---|---|
| **Random Position** | **Random seeking is possible if I-frame is found.** | **Decoder instance should be closed and created again whenever user moves its position.** |

**Table - Comparison of LINE_BUF and RING_BUF mode**

shows the comparison between LINE_BUF and RING_BUF modes.

## 2.3 Encoding

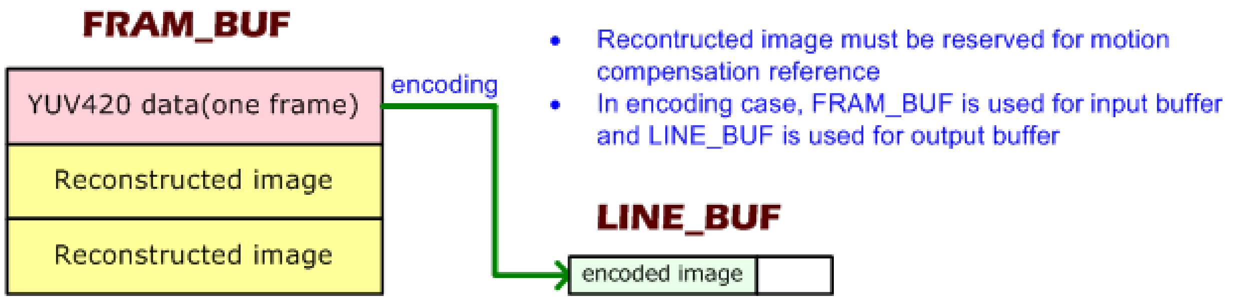
FRAM_BUF
YUV420 data(one frame)
Reconstructed image
Reconstructed image
encoding
Recontructed image must be reserved for motion compensation reference
In encoding case, FRAM_BUF is used for input buffer and LINE_BUF is used for output buffer
LINE_BUF
encoded image

**Fig - Using buffer in Encoding**

Fig 2.4 describes input and output buffer in Encoding case. Reconstructed images are used for motion compensation.

# 3 Data Structure

## 3.1 SSBSIP_MPEG4_STREAM_INFO

| SSBSIP_MPEG4_STREAM_INFO | |
|---|---|
| int width | width of output frame |
| int height | height of output frame |

## 3.2 SSBSIP_H264_STREAM_INFO

| SSBSIP_H264_STREAM_INFO | |
|---|---|
| int width | width of output frame |
| int height | height of output frame |

## 3.3 SSBSIP_VC1_STREAM_INFO

| SSBSIP_VC1_STREAM_INFO | |
|---|---|
| int width | width of output frame |
| int height | height of output frame |

## 3.4 SSBSIP_MFC_STREAM_INFO

| SSBSIP_MFC_STREAM_INFO | |
|---|---|
| int width | width of output frame |
| int height | height of output frame |

# 4 Decoder API

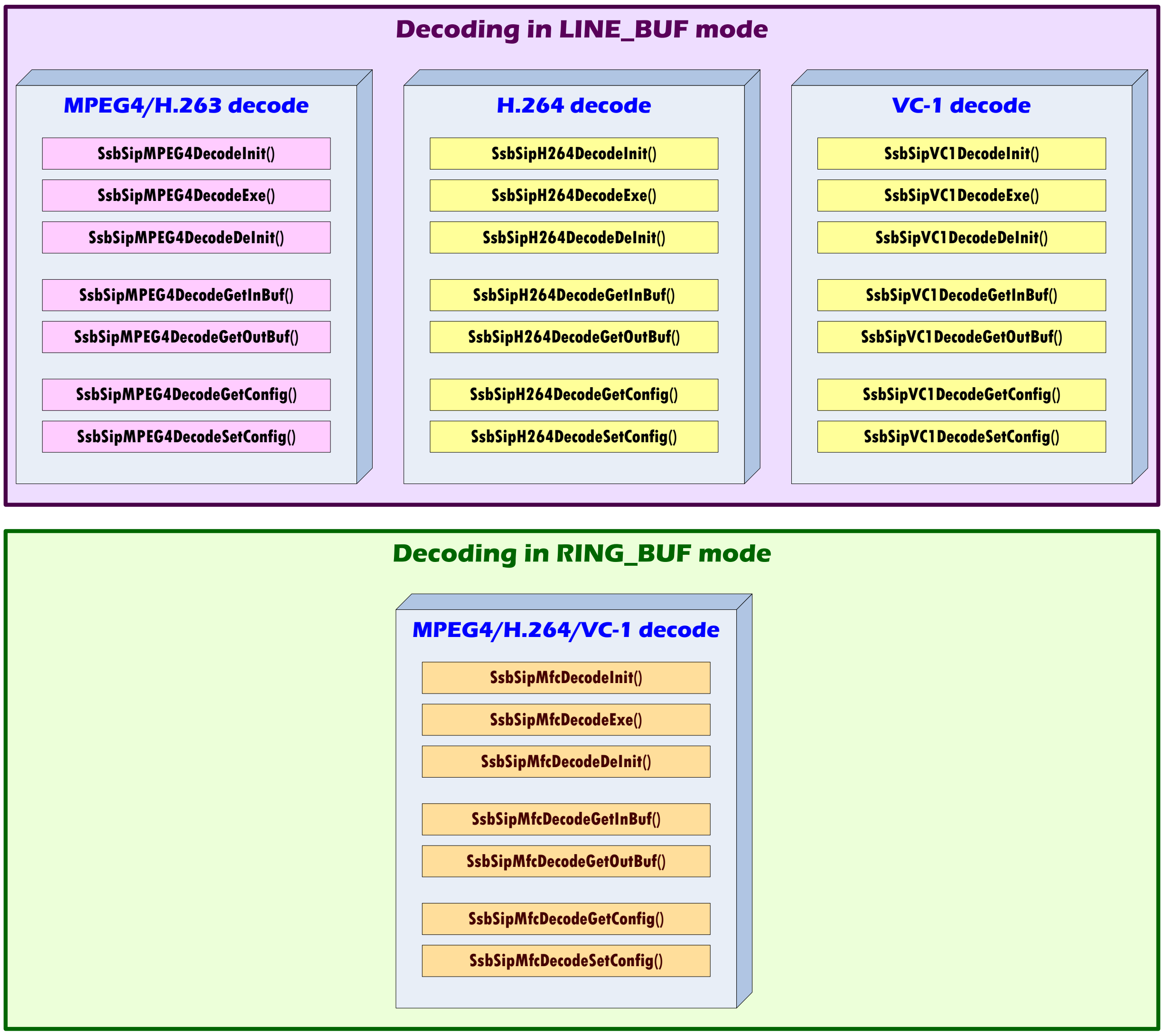
Decoding in LINE_BUF mode
MPEG4/H.263 decode
SsbSipMPEG4DecodeInit()
SsbSipMPEG4DecodeExe()
SsbSipMPEG4DecodeDeInit()
SsbSipMPEG4DecodeGetInBuf()
SsbSipMPEG4DecodeGetOutBuf()
SsbSipMPEG4DecodeGetConfig()
SsbSipMPEG4DecodeSetConfig()
H.264 decode
SsbSipH264DecodeInit()
SsbSipH264DecodeExe()
SsbSipH264DecodeDeInit()
SsbSipH264DecodeGetInBuf()
SsbSipH264DecodeGetOutBuf()
SsbSipH264DecodeGetConfig()
SsbSipH264DecodeSetConfig()
VC-1 decode
SsbSipVC1DecodeInit()
SsbSipVC1DecodeExe()
SsbSipVC1DecodeDeInit()
SsbSipVC1DecodeGetInBuf()
SsbSipVC1DecodeGetOutBuf()
SsbSipVC1DecodeGetConfig()
SsbSipVC1DecodeSetConfig()
Decoding in RING_BUF mode
MPEG4/H.264/VC-1 decode
SsbSipMfcDecodeInit()
SsbSipMfcDecodeExe()
SsbSipMfcDecodeDeInit()
SsbSipMfcDecodeGetInBuf()
SsbSipMfcDecodeGetOutBuf()
SsbSipMfcDecodeGetConfig()
SsbSipMfcDecodeSetConfig()

**Fig - MFC APIs for Decoding**

In , the decoding functions are shown. Since the MFC supports LINE_BUF mode and RING_BUF mode, the functions of two modes look same but slightly different. We will see in this chapter that the argument of GetInBuf is different.

## 4.1 MPEG4 Decoder (LINE_BUF Mode)

### 4.1.1 SsbSipMPEG4DecodeInit

| SsbSipMPEG4DecodeInit () | |
|---|---|
| Description | This function is<br>• to create the MPEG4/H.263 decoder instance |

| | |
|---|---|
| Syntax | void *<br>SsbSipMPEG4DecodeInit (void); |
| Parameters | |
| Returns | Return handle of the MFC MPEG4/H.263 Decoder instance. |

### 4.1.2 SsbSipMPEG4DecodeExe

| SsbSipMPEG4DecodeExe () | |
|---|---|
| Description | This function is<br>• to decode MPEG4/H.263 video stream |
| Syntax | int<br>SsbSipMPEG4DecodeExe (void *openHandle, long lengthBufFill); |
| Parameters | [IN] openHandle - Return handle from SsbSipMPEG4DecodeInit ()<br>[IN] lengthBufFill – Length of data filled in the input buffer |
| Returns | int returns error code. |

### 4.1.3 SsbSipMPEG4DecodeDeInit

| SsbSipMPEG4DecodeDeInit () | |
|---|---|
| Description | This function is<br>• to release codec resources |
| Syntax | int<br>SsbSipMPEG4DecodeDeInit (void *openHandle); |
| Parameters | [IN] openHandle - Return handle after MPEG4/H.263 initialization. |
| Returns | int returns error code. |

### 4.1.4 SsbSipMPEG4DecodeGetInBuf

| SsbSipMPEG4DecodeGetInBuf () | |
|---|---|
| Description | This function is<br>• to get memory address for decoding input buffer |
| Syntax | void *<br>SsbSipMPEG4DecodeGetInBuf (void *openHandle, long size); |
| Parameters | [IN] openHandle - Return handle from SsbSipMPEG4DecodeInit ().<br>[IN] size - Allocation size(byte) |
| Returns | It returns memory address of decoding input buffer.<br>In H/W codec, physical address of decoding input buffer is |

| | |
|---|---|
| | statically set during initialization. Size is limited by 4MB.<br>In S/W codec, stream buffer is allocated dynamically. |

## 4.1.5 SsbSipMPEG4DecodeGetOutBuf

| **SsbSipMPEG4DecodeGetOutBuf ()** | |
|---|---|
| Description | This function is<br>• to get memory address for decoding output buffer |
| Syntax | void *<br>SsbSipMPEG4DecodeGetOutBuf (void *openHandle, long *size); |
| Parameters | [IN] openHandle - Return handle from SsbSipMPEG4DecodeInit ().<br>[IN] size – Output buffer size in byte |
| Returns | It returns memory address of YUV420 Frame buffer.<br>The size of frame buffer will be returned thru ‘size’ parameter. |

## 4.1.6 SsbSipMPEG4DecodeSetConfig

| **SsbSipMPEG4DecodeSetConfig ()** | |
|---|---|
| Description | This function is<br>• to set codec variables |
| Syntax | int<br>SsbSipMPEG4DecodeSetConfig (void *openHandle,<br>MPEG4_DEC_CONF conf_type,<br>void *value); |
| Parameters | [IN] openHandle<br>Return handle from SsbSipMPEG4DecodeInit ().<br>[IN] type<br>Configuration type defined 5.Defintion and Error codes<br>[IN] value<br>Configuration value. |
| Returns | int returns error code. |

## 4.1.7 SsbSipMPEG4DecodeGetConfig

| **SsbSipMPEG4DecodeGetConfig ()** | |
|---|---|
| Description | This function is<br>• to get codec variables |
| Syntax | int<br>SsbSipMPEG4DecodeGetConfig (void *openHandle,<br>MPEG4_DEC_CONF conf_type,<br>void *value); |
| Parameters | [IN] openHandle<br>Return handle from SsbSipMPEG4DecodeInit ().<br>[IN] type |

| | |
|---|---|
| | Configuration type defined 5.Defintion and Error codes<br>[OUT] value<br>Configuration value |
| Returns | int returns error code. |

## 4.2 MFC Decoder (RINGE_BUF Mode)

### 4.2.1 SsbSipMfcDecodeInit

| SsbSipMfcDecodeInit () | |
|---|---|
| Description | This function is<br>• to create the MFC decoder instance of Ring Buffer mode. |
| Syntax | void *<br>SsbSipMfcDecodeInit (int dec_type); |
| Parameters | [IN] dec_type – decoder type<br>SSBSIPMFCDEC_MPEG4 : MPEG4 decoder<br>SSBSIPMFCDEC_H263 : H.263 decoder<br>SSBSIPMFCDEC_H264 : H.264 decoder<br>SSBSIPMFCDEC_VC1 : VC-1 decoder |
| Returns | Return handle of the MFC Decoder instance (Ring Buffer Mode). |

### 4.2.2 SsbSipMfcDecodeExe

| SsbSipMfcDecodeExe () | |
|---|---|
| Description | This function is<br>• to decode compressed video stream |
| Syntax | int<br>SsbSipMfcDecodeExe (void *openHandle, long lengthBufFill); |
| Parameters | [IN] openHandle - Return handle from SsbSipMfcDecodeInit ().<br>[IN] lengthBufFill – Length of data filled in the input buffer. |
| Returns | int returns error code. |

### 4.2.3 SsbSipMfcDecodeDeInit

| SsbSipMfcDecodeDeInit () | |
|---|---|
| Description | This function is<br>• to release codec resources |

| | |
|---|---|
| Syntax | int<br>SsbSipMfcDecodeDeInit (void *openHandle); |
| Parameters | [IN] openHandle - Return handle from SsbSipMfcDecodeInit (). |
| Returns | int returns error code. |

### 4.2.4 SsbSipMfcDecodeGetInBuf

| **SsbSipMfcDecodeGetInBuf ()** | |
|---|---|
| Description | This function is<br>• to get memory address for decoding input buffer |
| Syntax | void *<br>SsbSipMfcDecodeGetInBuf (void *openHandle, long *size); |
| Parameters | [IN] openHandle - Return handle from SsbSipMfcDecodeInit ().<br>[OUT] size – size of data to be filled in the input buffer |
| Returns | It returns memory address of decoding input buffer.<br>Application must fill the input buffer with the video stream of size which is returned thru 'size' parameter. |

### 4.2.5 SsbSipMfcDecodeGetOutBuf

| **SsbSipMfcDecodeGetOutBuf ()** | |
|---|---|
| Description | This function is<br>• to get memory address for decoding output buffer |
| Syntax | void *<br>SsbSipMfcDecodeGetOutBuf (void *openHandle, long *size); |
| Parameters | [IN] openHandle - Return handle from SsbSipMfcDecodeInit ().<br>[OUT] size – Output buffer size in byte |
| Returns | It returns memory address of YUV420 Frame buffer.<br>The size of frame buffer will be returned thru 'size' parameter. |

### 4.2.6 SsbSipMfcDecodeSetConfig

| **SsbSipMfcDecodeSetConfig ()** | |
|---|---|
| Description | This function is<br>• to set codec variables |
| Syntax | int<br>SsbSipMfcDecodeSetConfig (void *openHandle,<br>MFC_DEC_CONF conf_type,<br>void *value); |

| | |
|---|---|
| Parameters | [IN] openHandle<br>Return handle from SsbSipMfcDecodeInit ().<br>[IN] type<br>Configuration type defined 5.Defintion and Error codes<br>[IN] value<br>Configuration value. |
| Returns | int returns error code. |

### 4.2.7 SsbSipMfcDecodeGetConfig

| SsbSipMfcDecodeGetConfig () | |
|---|---|
| Description | This function is<br>• to get codec variables |
| Syntax | int<br>SsbSipMfcDecodeGetConfig (void *openHandle,<br>MFC_DEC_CONF conf_type,<br>void *value); |
| Parameters | [IN] openHandle<br>Return handle from SsbSipMfcDecodeInit ().<br>[IN] type<br>Configuration type defined 5.Defintion and Error codes<br>[OUT] value<br>Configuration value |
| Returns | int returns error code. |

# 5 Encoder API

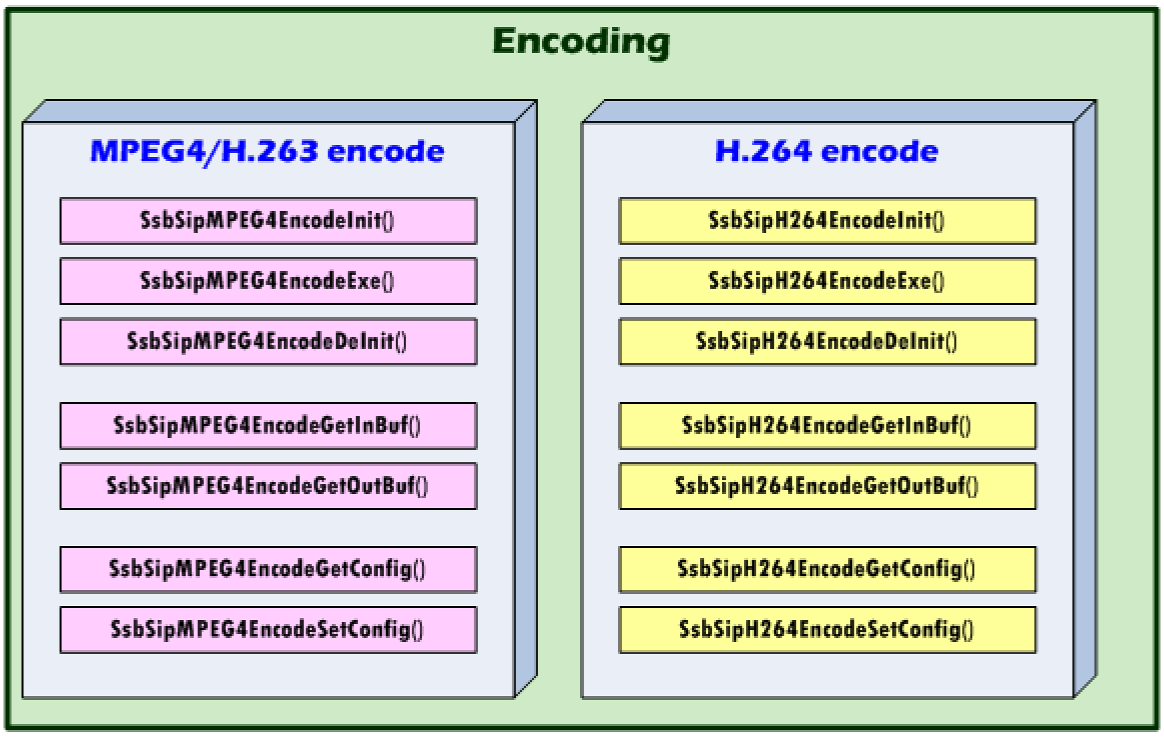
Encoding
MPEG4/H.263 encode
SsbSipMPEG4EncodeInit()
SsbSipMPEG4EncodeExe()
SsbSipMPEG4EncodeDeInit()
SsbSipMPEG4EncodeGetInBuf()
SsbSipMPEG4EncodeGetOutBuf()
SsbSipMPEG4EncodeGetConfig()
SsbSipMPEG4EncodeSetConfig()
H.264 encode
SsbSipH264EncodeInit()
SsbSipH264EncodeExe()
SsbSipH264EncodeDeInit()
SsbSipH264EncodeGetInBuf()
SsbSipH264EncodeGetOutBuf()
SsbSipH264EncodeGetConfig()
SsbSipH264EncodeSetConfig()

**Fig - MFC APIs for Encoding**

## 5.1 MPEG4 Encoder

### 5.1.1 SsbSipMPEG4EncodeInit

| **SsbSipMPEG4DecodeInit ()** | |
|---|---|
| Description | This function is<br>• to create the MPEG4 encoder instance |
| Syntax | void *<br>SsbSipMPEG4EncodeInit (int strmType<br>unsigned int uiWidth,<br>unsigned int uiHeight,<br>unsigned int uiFramerate,<br>unsigned int uiBitrate_kbps,<br>unsigned int uiGOPNum) |
| Parameters | [IN] strmType :<br>SSBSIPMFCENC_MPEG4 – MPEG4 encoding<br>SSBSIPMFCENC_H263 - H.263 encoding<br>[IN] uiWidth – Width of YUV420 frame to be MPEG4-encoded<br>[IN] uiHeight – Height of YUV420 frame to be MPEG4-encoded<br>[IN] uiFramerate – encoding frame rate in fps(frame/second)<br>[IN] uiBitrate_kbps – bitrate in kbps |

| | |
|---|---|
| | [IN] uiGOPNum – I-frame inser |
| Returns | Return handle of the MFC MPEG4 Encoder instance. |

### 5.1.2 SsbSipMPEG4EncodeExe

| **SsbSipMPEG4EncodeExe ()** | |
|---|---|
| Description | This function is<br>• to encode YUV420 frames into the MPEG4 video stream |
| Syntax | int<br>SsbSipMPEG4EncodeExe (void *openHandle); |
| Parameters | [IN] openHandle - Return handle from SsbSipMPEG4EncodeInit () |
| Returns | int returns error code. |

### 5.1.3 SsbSipMPEG4EncodeDeInit

| **SsbSipMPEG4EncodeDeInit ()** | |
|---|---|
| Description | This function is<br>• to release codec resources |
| Syntax | int<br>SsbSipMPEG4EncodeDeInit (void *openHandle); |
| Parameters | [IN] openHandle - Return handle after MPEG4 initialization. |
| Returns | int returns error code. |

### 5.1.4 SsbSipMPEG4EncodeGetInBuf

| **SsbSipMPEG4EncodeGetInBuf ()** | |
|---|---|
| Description | This function is<br>• to get memory address for decoding input buffer |
| Syntax | void *<br>SsbSipMPEG4EncodeGetInBuf (void *openHandle, long size); |
| Parameters | [IN] openHandle - Return handle from SsbSipMPEG4EncodeInit ().<br>[IN] size - Allocation size(byte) |
| Returns | It returns memory address of decoding input buffer.<br>In H/W codec, physical address of decoding input buffer is statically set during initialization. Size is limited by 4MB.<br>In S/W codec, stream buffer is allocated dynamically. |

### 5.1.5 SsbSipMPEG4EncodeGetOutBuf

| **SsbSipMPEG4EncodeGetOutBuf ()** | |
|---|---|
| Description | This function is<br>• to get memory address for decoding output buffer |
| Syntax | void *<br>SsbSipMPEG4EncodeGetOutBuf (void *openHandle, long *size); |
| Parameters | [IN] openHandle - Return value from SsbSipMPEG4DecodeInit ().<br>[IN] size – Output buffer size in byte |
| Returns | It returns memory address of YUV420 Frame buffer.<br>The size of frame buffer will be returned thru ‘size’ parameter. |

### 5.1.6 SsbSipMPEG4EncodeSetConfig

| **SsbSipMPEG4EncodeSetConfig ()** | |
|---|---|
| Description | This function is<br>• to set codec variables |
| Syntax | int<br>SsbSipMPEG4EncodeSetConfig (void *openHandle,<br>MPEG4_DEC_CONF conf_type,<br>void *value); |
| Parameters | [IN] openHandle - Return handle from SsbSipMPEG4EncodeInit ().<br>[IN] type - Configuration type defined 5.Defintion and Error codes<br>[IN] value - Configuration value. |
| Returns | int returns error code. |

### 5.1.7 SsbSipMPEG4EncodeGetConfig

| **SsbSipMPEG4EncodeGetConfig ()** | |
|---|---|
| Description | This function is<br>• to get codec variables |
| Syntax | int<br>SsbSipMPEG4EncodeGetConfig (void *openHandle,<br>MPEG4_DEC_CONF conf_type,<br>void *value); |
| Parameters | [IN] openHandle - Return handle from SsbSipMPEG4EncodeInit ().<br>[IN] type - Configuration type defined 5.Defintion and Error codes<br>[OUT] value - Configuration value |
| Returns | int returns error code. |

# 6 Definition and Error Codes

## 6.1 Configuration

### 6.1.1 XXX_DEC_GETCONF_STREAMINFO

| Definition | Description |
|---|---|
| MPEG4_DEC_GETCONF_STREAMINFO | To get width and height of the corresponding MPEG4 stream. |
| H264_DEC_GETCONF_STREAMINFO | To get width and height of the corresponding h.264 stream. |
| VC1_DEC_GETCONF_STREAMINFO | To get width and height of the corresponding VC-1 stream |
| MFC_DEC_GETCONF_STREAMINFO | To get width and height of the corresponding video stream. |

Value parameter :

```
typedef struct
{
       int width;
       int height;
} SSBSIP_MPEG4_STREAM_INFO;

typedef struct
{
       int width;
       int height;
} SSBSIP_H264_STREAM_INFO;

typedef struct
{
       int width;
       int height;
} SSBSIP_VC1_STREAM_INFO;

typedef struct
{
       int width;
       int height;
} SSBSIP_MFC_STREAM_INFO;
```

### 6.1.2 XXX_DEC_GETCONF_PHYADDR_FRAM_BUF

| Definition | Description |
|---|---|
| MPEG4_DEC_GETCONF_PHYADDR_FRAM_BUF | To get the physical address of YUV420 output buffer (FRAM_BUF) in MPEG4/H.263 decoding |
| H264_DEC_GETCONF_PHYADDR_FRAM_BUF | To get the physical address of YUV420 output buffer (FRAM_BUF) in H.264 decoding |

| | |
|---|---|
| VC1_DEC_GETCONF_PHYADDR_FRAM_BUF | To get the physical address of YUV420 output buffer (FRAM_BUF) in VC-1 decoding |
| MFC_DEC_GETCONF_PHYADDR_FRAM_BUF | To get the physical address of YUV420 output buffer (FRAM_BUF) video decoding. |

Value parameter :

int [2]

- int [0] : physical address of YUV420 output buffer
- int [1] : length of YUV420 output buffer

### 6.1.3 MPEG4_DEC_GETCONF_MPEG4_XXX

Below information is used for Hybrid divx decoder.

| Definition | Description | Where should it be called? |
|---|---|---|
| MPEG4_DEC_GETCONF_MPEG4_FCODE | It returns FCODE. The size is 4 bytes. | After decoding |
| MPEG4_DEC_GETCONF_MPEG4_TRD | It returns TRD. The size is 4 bytes. | After decoding |
| MPEG4_DEC_GETCONF_MPEG4_TIME_BASE_LAST | It returns TIME_BASE_LAST. The size is 4 bytes | After decoding |
| MPEG4_DEC_GETCONF_MPEG4_NONB_TIME_LAST | It returns NONB_TIME_LAST. The size is 4 bytes | After decoding |
| MPEG4_DEC_GETCONF_MPEG4_VOP_TIME_RES | It returns VOP_TIME_RES. The size is 4 bytes. | After initialization |
| MPEG4_DEC_GETCONF_MPEG4_MV_ADDR | It returns MV address and MB Type(=MV addr + 25920) address. It use big endian. MB type is not a MB type. It just represents coded or not coded. So the range of MB type is 0~1. | After decoding |
| MPEG4_DEC_GETCONF_MPEG4_CONSUMED | Number of bytes of P-frame which is decoded by MFC in P and B frames | After decoding |

**Byte order of MV value and MB type**

MV value and MB type use big endian. Please becarefull

<MV value>

0 32 64

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| MV4 | MV3 | MV2 | MV1 | MV8 | MV7 | MV6 | MV5 | |
| 8bits | | | | | | | | |

<MB type>

0 32 64

| | | | |
|---|---|---|---|
| Y1 | X1 | Y2 | X2 |
| 16bit | | | |

### 6.1.4 XXX_DEC_GETCONF_FRAM_NEED_COUNT

| Definition | Description |
|---|---|

| | |
|---|---|
| MPEG4_DEC_GETCONF_FRAM_NEED_COUNT | To get the count of frame buffer in MPEG4/H.263 decoding |
| H264_DEC_GETCONF_FRAM_NEED_COUNT | To get the count of frame buffer in H.264 decoding |
| VC1_DEC_GETCONF_ FRAM_NEED_COUNT | To get the count of frame buffer in VC-1 decoding |
| MFC_DEC_GETCONF_ FRAM_NEED_COUNT | To get the count of frame buffer video decoding. |

Value parameter :
int [2]
- int [0] : Count of frame buffer which is MFC's output buffer
- int [1] : Not Used

## 6.1.5 XXX_DEC_SETCONF_POST_ROTATE

Post rotation mode configures the MFC to rotate and/or mirror the output YUV image during the decoding.

| Definition | Description |
|---|---|
| MPEG4_DEC_SETCONF_POST_ROTATE | To set the Post rotation mode for the YUV420 output. |
| H264_DEC_SETCONF_POST_ROTATE | To set the Post rotation mode for the YUV420 output. |
| VC1_DEC_SETCONF_POST_ROTATE | To set the Post rotation mode for the YUV420 output. |

Value parameter :
int [1]
- int[0]: PostRotateMode

| PostRotateMode | HorMir | VerMir | RotAng |
|---|---|---|---|
| 0x0000 | X | X | X |
| 0x0010 | X | X | X |
| 0x0011 | X | X | $90^\circ$ rotate |
| 0x0012 | X | X | $180^\circ$ rotate |
| 0x0013 | X | X | $270^\circ$ rotate |
| 0x0014 | X | O | X |
| 0x0015 | X | O | $90^\circ$ rotate |
| 0x0016 | X | O | $180^\circ$ rotate |
| 0x0017 | X | O | $270^\circ$ rotate |
| 0x0018 | O | X | X |
| 0x0019 | O | X | $90^\circ$ rotate |
| 0x001A | O | X | $180^\circ$ rotate |
| 0x001B | O | X | $270^\circ$ rotate |
| 0x001C | O | O | X |
| 0x001D | O | O | $90^\circ$ rotate |

| | | | |
|---|---|---|---|
| 0x001E | O | O | 180° rotate |
| 0x001F | O | O | 270° rotate |

**Table - Post rotate mode value in decoding**

## 6.1.6 XXX_ENC_SETCONF_NUM_SLICES

Number of multiple slices mode configures the MFC to encode the YUV images into multiple slices only for H.263 and H.264.

| Definition | Description |
|---|---|
| `MPEG4_ENC_SETCONF_H263_NUM_SLICES` | To produce H.263 stream with multiple slices. |
| `H264_ENC_SETCONF_NUM_SLICES` | To produce H.264 stream with multiple slices. |

Value parameter :

int [2]

- o int [0] : 0=single slice, 1=multiple slices
- o int [1] : Number of multiple slices (range: 1 ~ 256)

## 6.1.7 XXX_ENC_SETCONF_PARAM_CHANGE

The encoding parameters such as bitrate, frame rate and intra qp can be changed dynamically while the encoding process is going on. The encoding parameters are initially fixed at MFC encoder instance initialization step. Once it is initialized, their changes are possible through this method.

| Definition | Description |
|---|---|
| `MPEG4_ENC_SETCONF_PARAM_CHANGE` | To change MPEG4/H.263 encoding parameters after the MFC encoder instance initialization. |
| `H264_ENC_SETCONF_PARAM_CHANGE` | To change H.264 encoding parameters after the MFC encoder instance initialization. |

Value parameter :

int [2]

- o int [0] : ID for parameter change
- o int [1] : New value of the parameter

| Value parameter Int[0] | Value parameter Int[1] | Description |
|---|---|---|
| `MPEG4_ENC_PARAM_GOP_NUM` | 0 ~ 60 | 0 - I, P, P, P, …<br>1 - I, I, I, I, …<br>2 - I, P, I, P, …<br>3 - I, P, P, I, P, P, I, … |
| `MPEG4_ENC_PARAM_INTRA_QP` | 1 ~ 31 | Intra frame picture quantized step parameter |
| `MPEG4_ENC_PARAM_BITRATE` | 1 ~ 32767 | Target bitrate in kbps |
| `MPEG4_ENC_PARAM_F_RATE` | [FrameRateDiv-1]<br>[FrameRateRes] | Bits 31~16 : FrameRateDiv-1<br>Bits 15~0 : FrameRateRes<br><br>F_RATE =<br>[FrameRateRes] / |

| | | |
|---|---|---|
| | | [FrameRateDiv-1] |
| MPEG4_ENC_PARAM_INTRA_REF | 0 ~ N | Intra MB refresh number.<br>0 - Intra MB refresh is not used<br>N - At least N number of MBs are encoded as intra mode at every picture |
| MPEG4_ENC_PARAM_SLICE_MODE | 0 ~ 256 | Number of multiple slices per picture<br>0 - Single slice<br>1 ~ 256 - Multiple slices |
| H264_ENC_PARAM_GOP_NUM | 0 ~ 60 | 0 - I, P, P, P, …<br>1 - I, I, I, I, …<br>2 - I, P, I, P, …<br>3 - I, P, P, I, P, P, I, … |
| H264_ENC_PARAM_INTRA_QP | 0 ~ 51 | Intra frame picture quantized step parameter |
| H264_ENC_PARAM_BITRATE | 1 ~ 32767 | Target bitrate in kbps |
| H264_ENC_PARAM_F_RATE | [FrameRateDiv-1]<br>[FrameRateRes] | Bits 31~16 : FrameRateDiv-1<br>Bits 15~0 : FrameRateRes<br><br>F_RATE =<br>[FrameRateRes] /<br>[FrameRateDiv-1] |
| H264_ENC_PARAM_INTRA_REF | 0 ~ N | Intra MB refresh number.<br>0 - Intra MB refresh is not used<br>N - At least N number of MBs are encoded as intra mode at every picture |
| H264_ENC_PARAM_SLICE_MODE | 0 ~ 256 | Number of multiple slices per picture<br>0 - Single slice<br>1 ~ 256 - Multiple slices |

**Table - Parameter change value in SET_CONF while encoding**

## 6.1.8 XXX_ENC_SETCONF_CUR_PIC_OPT

Set the encoding option such as VOP type, encode skip for the current picture. These encoding options affect the current picture only. Therefore the options need to be set at every time whenever you want.

| Definition | Description |
|---|---|
| MPEG4_ENC_SETCONF_CUR_PIC_OPT | To set the MPEG4/H.263 encoding option for the current picture. |
| H264_ENC_SETCONF_CUR_PIC_OPT | To set the H.264 encoding option for the current picture. |

Value parameter :

int [2]

- int [0] : 0=single slice, 1=multiple slices
- int [1] : Number of multiple slices (range: 1 ~ 256)

| Value parameter Int[0] | Value parameter Int[1] | Description |
|---|---|---|
| MPEG4_ENC_PIC_OPT_IDR | 1 | The current source image is encoded as 'I' picture. |
| MPEG4_ENC_PIC_OPT_SKIP | 1 | The current source image is ignored. (Encoding is skipped.) |
| H264_ENC_PIC_OPT_IDR | 1 | The current source image is encoded as 'IDR' picture. |
| H264_ENC_PIC_OPT_SKIP | 1 | The current source image is ignored. (Encoding is skipped.) |
| H264_ENC_PIC_OPT_RECOVERY | 1 ~ 7 | The current and several following images will be encoded as multiple slices for the gradual recovery. The SEI message which is containing the recovery point is generated. |

**Table - Parameter change value in SET_CONF while encoding**

### 6.1.9 MPEG4_DEC_SETCONF_CACHE_XXX

Cache operation for data buff in MFC

| Definition | Description |
|---|---|
| MPEG4_DEC_SETCONF_CACHE_INVALIDATE | To clear Cached data without matching with memory. |
| MPEG4_DEC_SETCONF_CACHE_CLEAN | To match data in cache with memory. |
| MPEG4_DEC_SETCONF_CACHE_CLEAN_INVALIDATE | Clear and invalidate cache. |

Value parameter :
int [2]
- int [0] : the virtual address of starting
- int [1] : memory size

### 6.1.10 MPEG4_DEC_SETCONF_PADDING_SIZE

To set padding size in MFC's decoding output buffer

| Definition | Description |
|---|---|
| MPEG4_DEC_SETCONF_PADDING_SIZE | To set decoder's output buffer padding size |

Value parameter :
int [2]
- int [0] : padding size. It has to be multiple of 8.
- int [1] : memory size

## 6.2 Error Codes

### 6.2.1 MPEG4 Decode Error Codes

| Error Code | Description |
|---|---|
| SSBSIP_MPEG4_DEC_RET_OK | Success |
| SSBSIP_MPEG4_DEC_RET_ERR_INVALID_PARAM | Invalid parameter for function argument |
| SSBSIP_MPEG4_DEC_RET_ERR_INVALID_HANDLE | Input handle is NULL or invalid. |
| SSBSIP_MPEG4_DEC_RET_ERR_CONFIG_FAIL | SsbSipMPEG4DecodeExe() returns this error when configuration fails. |
| SSBSIP_MPEG4_DEC_RET_ERR_DECODE_FAIL | SsbSipMPEG4DecodeExe() returns this error when MPEG4 decoding fails. |

### 6.2.2 H.264 Decode Error Codes

| Error Code | Description |
|---|---|
| SSBSIP_H264_DEC_RET_OK | Success |
| SSBSIP_H264_DEC_RET_ERR_INVALID_PARAM | Invalid parameter for function argument |
| SSBSIP_H264_DEC_RET_ERR_INVALID_HANDLE | Input handle is NULL or invalid. |
| SSBSIP_H264_DEC_RET_ERR_CONFIG_FAIL | SsbSipH264DecodeExe() returns this error when configuration fails. |
| SSBSIP_H264_DEC_RET_ERR_DECODE_FAIL | SsbSipH264DecodeExe () returns this error when H.264 decoding fails. |

### 6.2.3 MPEG4 Encode Error Codes

| Error Code | Description |
|---|---|
| SSBSIP_MPEG4_ENC_RET_OK | Success |
| SSBSIP_MPEG4_ENC_RET_ERR_INVALID_PARAM | Invalid parameter for function argument |
| SSBSIP_MPEG4_ENC_RET_ERR_INVALID_HANDLE | Input handle is NULL or invalid. |
| SSBSIP_MPEG4_ENC_RET_ERR_DECODE_FAIL | SsbSipMPEG4EncodeExe() returns this error when MPEG4 encoding fails. |

### 6.2.4 H.264 Encode Error Codes

| Error Code | Description |
|---|---|
| `SSBSIP_H264_ENC_RET_OK` | Success |
| `SSBSIP_H264_ENC_RET_ERR_INVALID_PARAM` | Invalid parameter for function argument |
| `SSBSIP_H264_ENC_RET_ERR_INVALID_HANDLE` | Input handle is NULL or invalid. |
| `SSBSIP_H264_ENC_RET_ERR_DECODE_FAIL` | `SsbSipH264EncodeExe()` returns this error when H.264 encoding fails. |

# 7 Sample Codes

## 7.1 Windows CE/Mobile Case

### 7.1.1 MPEG4 Decoder Sample

**MPEG4 Decode (LINE_BUF mode)**

```
int mpeg4dec_test(char *filename)
{
      void    *handle;

      SSBSIP_MPEG4_STREAM_INFO stream_info;

      FILE    *fp_in,*fp_out;

      int      nLoop, nFrames;

      void              *pStrmBuf;
      int                nFrameLeng;
      unsigned char     *pYUVBuf;
      int                nYUVLeng;

      FRAMEX_CTX  *pFrameExCtx;


      ////////////////////////
      // Opening Input File //
      ////////////////////////
      fp_in = fopen(filename, "rb");
      if (fp_in == NULL) {
            RETAILMSG(1,(L"File not found\n"));
            return 0;

      }
      /////////////////////////
      // Opening Output File //
      /////////////////////////
      fp_out = fopen("\\Temp\\output.yuv","wb");
      if (fp_out == NULL) {
            RETAILMSG(1,(L"Cannot open the output file.\n"));
            return 0;

      }

      ///////////////////////////////////
      // FrameExtractor Initialization //
      ///////////////////////////////////
      pFrameExCtx = FrameExtractorInit(FRAMEX_IN_TYPE_FILE,
                                       delimiter_mpeg4,
                                       sizeof(delimiter_mpeg4),
                                       1);
      FrameExtractorFirst(pFrameExCtx, fp_in);


      //////////////////////////////////////
      ///    1. Create new instance      ///
      ///      (SsbSipMPEG4DecodeInit)   ///
      //////////////////////////////////////
      handle = SsbSipMPEG4DecodeInit();
      if (handle == NULL) {
            RETAILMSG(1,(L"SsbSipMPEG4DecodeInit Failed.\n"));
            return 0;

      }

      ///////////////////////////////////////////////
      ///    2. Obtaining the Input Buffer      ///
      ///      (SsbSipMPEG4DecodeGetInBuf)      ///
      ///////////////////////////////////////////////
      pStrmBuf = SsbSipMPEG4DecodeGetInBuf(handle, nFrameLeng);
      if (pStrmBuf == NULL) {
            RETAILMSG(1,(L"SsbSipMPEG4DecodeGetInBuf Failed.\n"));
            SsbSipMPEG4DecodeDeInit(handle);
            return 0;
      }

      ////////////////////////////////////
      // MPEG4 CONFIG stream extraction //
      ////////////////////////////////////
      nFrameLeng = ExtractConfigStreamMpeg4(pFrameExCtx, fp_in,
                                      pStrmBuf, INPUT_BUFFER_SIZE);


      ////////////////////////////////////////////////////////////////
      ///    3. Configuring the instance with the config stream    ///
      ///       (SsbSipMPEG4DecodeExe)                             ///
      ////////////////////////////////////////////////////////////////
      if (SsbSipMPEG4DecodeExe(handle, nFrameLeng) != SSBSIP_MPEG4_DEC_RET_OK)
{
            RETAILMSG(1,(L"MPEG4 Decoder Configuration Failed.\n"));
            return 0;
      }

      /////////////////////////////////////
      ///   4. Get stream information   ///
      /////////////////////////////////////
      if (SsbSipMPEG4DecodeGetConfig(handle, MPEG4_DEC_GETCONF_STREAMINFO,
&stream_info) != SSBSIP_MPEG4_DEC_RET_OK)
            return 0;


      RETAILMSG(1,(L"\t<STREAMINFO> width=%d   height=%d.\n",
                       stream_info.width, stream_info.height));


      nFrames = 0;
      for (nLoop=0; nLoop < 4000; nLoop++) {

            ///////////////////////////////////
            ///       5. DECODE             ///
            ///    (SsbSipMPEG4DecodeExe)   ///
            ///////////////////////////////////
            if (SsbSipMPEG4DecodeExe(handle, nFrameLeng) !=
                     SSBSIP_MPEG4_DEC_RET_OK)
                  break;

            ///////////////////////////////////////////////
            ///    6. Obtaining the Output Buffer      ///
            ///      (SsbSipMPEG4DecodeGetInBuf)       ///
            ///////////////////////////////////////////////
            pYUVBuf = SsbSipMPEG4DecodeGetOutBuf(handle, &nYUVLeng);

            if (nLoop > 10 && nLoop < 30)
                  fwrite(pYUVBuf, 1, nYUVLeng, fp_out);

            RETAILMSG(1,(L"\t [%d]  decoded.\n", nLoop));

            ///////////////////////////
            // Next MPEG4 VOP stream //
            ///////////////////////////
            nFrameLeng = NextFrameMpeg4(pFrameExCtx, fp_in, pStrmBuf,
                                        INPUT_BUFFER_SIZE);
            if (nFrameLeng < 4)
                  break;
      }


      ////////////////////////////////////////
      ///    7. SsbSipMPEG4DecodeDeInit    ///
      ////////////////////////////////////////
      SsbSipMPEG4DecodeDeInit(handle);

      fclose(fp_in);
      fclose(fp_out);

      return 0;
}
```

## 7.1.2 H.264 Decoder Sample

**H.264 Decode (LINE_BUF mode)**

```
int h264dec_test(char *filename)
{
      void    *handle;

      SSBSIP_H264_STREAM_INFO stream_info;

      FILE    *fp_in,*fp_out;

      int      nLoop, nFrames;

      void              *pStrmBuf;
      int                nFrameLeng;
      unsigned char     *pYUVBuf;
      int                nYUVLeng;


      FRAMEX_CTX  *pFrameExCtx;


      ////////////////////////
      // Opening Input File //
      ////////////////////////
      fp_in = fopen(filename, "rb");
```

```
	if (fp_in == NULL) {
		RETAILMSG(1,(L"File not found\n"));
		return 0;
	}
	/////////////////////////
	// Opening Output File //
	/////////////////////////
	fp_out = fopen("\\Temp\\output.yuv","wb");
	if (fp_out == NULL) {
		RETAILMSG(1,(L"Cannot open the output file.\n"));
		return 0;

	}

	///////////////////////////////////
	// FrameExtractor Initialization //
	///////////////////////////////////
	pFrameExCtx = FrameExtractorInit(FRAMEX_IN_TYPE_FILE,
	                                 delimiter_h264,
	                                 sizeof(delimiter_h264),
	                                 1);
	FrameExtractorFirst(pFrameExCtx, fp_in);


	//////////////////////////////////////
	///    1. Create new instance      ///
	///      (SsbSipH264DecodeInit)   ///
	//////////////////////////////////////
	handle = SsbSipH264DecodeInit();
	if (handle == NULL) {
		RETAILMSG(1,(L"H264_Dec_Init Failed.\n"));
		return 0;
	}

	/////////////////////////////////////////////
	///    2. Obtaining the Input Buffer      ///
	///      (SsbSipH264DecodeGetInBuf)       ///
	/////////////////////////////////////////////
	pStrmBuf = SsbSipH264DecodeGetInBuf(handle, nFrameLeng);
	if (pStrmBuf == NULL) {
		RETAILMSG(1,(L"SsbSipH264DecodeGetInBuf Failed.\n"));
		SsbSipH264DecodeDeInit(handle);
		return 0;
	}

	////////////////////////////////////
	// H264 CONFIG stream extraction //
	////////////////////////////////////
	nFrameLeng = ExtractConfigStreamH264(pFrameExCtx, fp_in, pStrmBuf,
	                                     INPUT_BUFFER_SIZE, 1);


	////////////////////////////////////////////////////////////////
	///    3. Configuring the instance with the config stream    ///
	///       (SsbSipH264DecodeExe)                              ///
	////////////////////////////////////////////////////////////////
	if (SsbSipH264DecodeExe(handle, nFrameLeng) != SSBSIP_H264_DEC_RET_OK) {
		RETAILMSG(1,(L"H.264 Decoder Configuration Failed.\n"));
		return 0;
	}
```

```
       //////////////////////////////////////
       ///   4. Get stream information   ///
       //////////////////////////////////////
       SsbSipH264DecodeGetConfig(handle,
                                 H264_DEC_GETCONF_STREAMINFO,
                                 &stream_info);

       RETAILMSG(1,(L"\t<STREAMINFO> width=%d   height=%d.\n",
                 stream_info.width, stream_info.height));

       nFrames = 0;
       for (nLoop=0; nLoop < 4; nLoop++) {

             //////////////////////////////////
             ///       5. DECODE            ///
             ///    (SsbSipH264DecodeExe)   ///
             //////////////////////////////////
             if (SsbSipH264DecodeExe(handle, nFrameLeng) !=
                         SSBSIP_H264_DEC_RET_OK)
                   break;

             ////////////////////////////////////////////////
             ///    6. Obtaining the Output Buffer      ///
             ///      (SsbSipH264DecodeGetOutBuf)       ///
             ////////////////////////////////////////////////
             pYUVBuf = SsbSipH264DecodeGetOutBuf(handle, &nYUVLeng);

             if (nLoop > 10 && nLoop < 12)
                   fwrite(pYUVBuf, 1, nYUVLeng, fp_out);

             RETAILMSG(1,(L"\t [%d]  decoded.\n", nLoop));

             /////////////////////////////
             // Next H.264 VIDEO stream //
             /////////////////////////////
             nFrameLeng = NextFrameH264(pFrameExCtx, fp_in, pStrmBuf,
                                        INPUT_BUFFER_SIZE);
             if (nFrameLeng < 4)
                   break;

       }

       ////////////////////////////////////////
       ///    7. SsbSipH264DecodeDeInit    ///
       ////////////////////////////////////////
       SsbSipH264DecodeDeInit(handle);

       fclose(fp_in);
       fclose(fp_out);

       return 0;
}
```

## 7.2 Linux Case

### 7.2.1 MPEG4 Decoder Sample

**MPEG4 Decode (LINE_BUF mode)**

```
int Test_MPEG4_Decoder_Line_Buffer(int argc, char **argv)
{
	void			*handle;
	void			*pStrmBuf;
	int			nFrameLeng = 0;
	unsigned char	*pYUVBuf;
	long			nYUVLeng;
	int			in_fd, out_fd;
	int			file_size;
	char			*in_addr;

	struct stat		s;
	FRAMEX_CTX		*pFrameExCtx;	// frame extractor context
	FRAMEX_STRM_PTR		file_strm;
	SSBSIP_MPEG4_STREAM_INFO stream_info;

#ifdef FPS
	struct timeval	start, stop;
	unsigned int		time = 0;
	int			frame_cnt = 0;
#endif


	if (argc != 3) {
		printf("Usage : mfc <MPEG4 input filename> <output filename>\n");
		return -1;
	}

	////////////////////////////////////
	// Input/Output Stream File Open //
	////////////////////////////////////
	in_fd	= open(argv[1], O_RDONLY);
	out_fd	= open(argv[2], O_RDWR | O_CREAT | O_TRUNC, 0644);
	if( (in_fd < 0) || (out_fd < 0) ) {
		LOG_MSG(LOG_ERROR, "Test_MPEG4_Decoder_Line_Buffer", "Input/Output file open
failed\n");
		return -1;
	}

	// get input file size
	fstat(in_fd, &s);
	file_size = s.st_size;

	// Input file should be mapped with memory.
	// because file operations have a lot of performance down.
	// So, I Strongly recommend you to use mmap() of input file.
	//////////////////////////////////////
	// Input/Output Buffer Memory Mapping //
	//////////////////////////////////////
	in_addr = (char *)mmap(0, file_size, PROT_READ, MAP_SHARED, in_fd, 0);
	if(in_addr == NULL) {
		LOG_MSG(LOG_ERROR, "Test_MPEG4_Decoder_Line_Buffer", "Mmap of Input file was
failed\n");
		return -1;
	}

	////////////////////////////////////
	// FrameExtractor Initialization	//
	////////////////////////////////////
```

```
        pFrameExCtx = FrameExtractorInit(FRAMEX_IN_TYPE_MEM, delimiter_mpeg4,
sizeof(delimiter_mpeg4), 1);
        file_strm.p_start = file_strm.p_cur = (unsigned char *)in_addr;
        file_strm.p_end = (unsigned char *)(in_addr + file_size);
        FrameExtractorFirst(pFrameExCtx, &file_strm);


        //////////////////////////////////////
        ///    1. Create new instance      ///
        ///      (SsbSipMPEG4DecodeInit)   ///
        //////////////////////////////////////
        handle = SsbSipMPEG4DecodeInit();
        if (handle == NULL) {
                LOG_MSG(LOG_ERROR, "Test_MPEG4_Decoder_Line_Buffer", "MPEG4_Dec_Init
Failed.\n");
                return -1;
        }

        /////////////////////////////////////////////
        ///    2. Obtaining the Input Buffer      ///
        ///      (SsbSipMPEG4DecodeGetInBuf)      ///
        /////////////////////////////////////////////
        pStrmBuf = SsbSipMPEG4DecodeGetInBuf(handle, nFrameLeng);
        if (pStrmBuf == NULL) {
                LOG_MSG(LOG_ERROR, "Test_MPEG4_Decoder_Line_Buffer",
"SsbSipMPEG4DecodeGetInBuf Failed.\n");
                SsbSipMPEG4DecodeDeInit(handle);
                return -1;
        }

        ////////////////////////////////////
        // MPEG4 CONFIG stream extraction //
        ////////////////////////////////////
        nFrameLeng = ExtractConfigStreamMpeg4(pFrameExCtx, &file_strm, pStrmBuf,
INPUT_BUFFER_SIZE, NULL);


        //////////////////////////////////////////////////////////////
        ///    3. Configuring the instance with the config stream  ///
        ///       (SsbSipMPEG4DecodeExe)                           ///
        //////////////////////////////////////////////////////////////
        if (SsbSipMPEG4DecodeExe(handle, nFrameLeng) != SSBSIP_MPEG4_DEC_RET_OK) {
                LOG_MSG(LOG_ERROR, "Test_MPEG4_Decoder_Line_Buffer", "MPEG4 Decoder
Configuration Failed.\n");
                return -1;
        }


        /////////////////////////////////////
        ///   4. Get stream information   ///
        /////////////////////////////////////
        SsbSipMPEG4DecodeGetConfig(handle, MPEG4_DEC_GETCONF_STREAMINFO, &stream_info);

        LOG_MSG(LOG_TRACE, "Test_MPEG4_Decoder_Line_Buffer", "\t<STREAMINFO> width=%d
height=%d.\n", stream_info.width, stream_info.height);


        while(1) {
        #ifdef FPS
```

```
                gettimeofday(&start, NULL);
        #endif

                //////////////////////////////////
                ///      5. DECODE               ///
                ///   (SsbSipMPEG4DecodeExe)     ///
                //////////////////////////////////
                if (SsbSipMPEG4DecodeExe(handle, nFrameLeng) != SSBSIP_MPEG4_DEC_RET_OK)
                        break;

        #ifdef FPS
                gettimeofday(&stop, NULL);
                time += measureTime(&start, &stop);
                frame_cnt++;
        #endif

                /////////////////////////////
                // Next MPEG4 VIDEO stream  //
                /////////////////////////////
                nFrameLeng = NextFrameMpeg4(pFrameExCtx, &file_strm, pStrmBuf,
INPUT_BUFFER_SIZE, NULL);
                if (nFrameLeng < 4)
                        break;

                /////////////////////////////////////////////
                ///   6. Obtaining the Output Buffer        ///
                ///     (SsbSipMPEG4DecodeGetOutBuf)        ///
                /////////////////////////////////////////////
                pYUVBuf = SsbSipMPEG4DecodeGetOutBuf(handle, &nYUVLeng);

        #ifndef FPS
                write(out_fd, pYUVBuf, (stream_info.width * stream_info.height * 3) >> 1);
        #endif

        }

#ifdef FPS
        LOG_MSG(LOG_TRACE, "Test_MPEG4_Decoder_Line_Buffer",     \
                "Decoding Time : %u, Frame Count : %d, FPS : %f\n", time, frame_cnt,
(float)frame_cnt*1000/time);
#endif

        ////////////////////////////////////
        ///   7. SsbSipMPEG4DecodeDeInit   ///
        ////////////////////////////////////
        SsbSipMPEG4DecodeDeInit(handle);

        LOG_MSG(LOG_TRACE, "Test_MPEG4_Decoder_Line_Buffer", "\n\n@@@ Program ends.\n");

        close(in_fd);
        close(out_fd);

        return 0;
}
```

### 7.2.2 H.264 Decoder Sample

**H.264 Decode (LINE_BUF mode)**

```
int Test_H264_Decoder_Line_Buffer(int argc, char **argv)
{
        void                *handle;
        void                *pStrmBuf;
        int                 nFrameLeng = 0;
        unsigned char       *pYUVBuf;
        long                nYUVLeng;
        int                 in_fd, out_fd;
        int                 file_size;
        char                *in_addr;

        struct stat         s;
        FRAMEX_CTX          *pFrameExCtx;   // frame extractor context
        FRAMEX_STRM_PTR             file_strm;
        SSBSIP_H264_STREAM_INFO stream_info;

#ifdef FPS
        struct timeval      start, stop;
        unsigned int        time = 0;
        int                 frame_cnt = 0;
#endif


        if (argc != 3) {
                printf("Usage : mfc <H.264 input filename> <output filename>\n");
                return -1;
        }

        ///////////////////////////////////
        // Input/Output Stream File Open     //
        ///////////////////////////////////
        in_fd   = open(argv[1], O_RDONLY);
        out_fd  = open(argv[2], O_RDWR | O_CREAT | O_TRUNC, 0644);
        if( (in_fd < 0) || (out_fd < 0) ) {
                LOG_MSG(LOG_ERROR, "Test_H264_Decoder_Line_Buffer", "Input/Output file open
failed\n");
                return -1;
        }

        // get input file size
        fstat(in_fd, &s);
        file_size = s.st_size;

        // Input file should be mapped with memory.
        // because file operations have a lot of performance down.
        // So, I Strongly recommend you to use mmap() of input file.
        /////////////////////////////////////
        // Input/Output Buffer Memory Mapping //
        /////////////////////////////////////
        in_addr = (char *)mmap(0, file_size, PROT_READ, MAP_SHARED, in_fd, 0);
        if(in_addr == NULL) {
                LOG_MSG(LOG_ERROR, "Test_H264_Decoder_Line_Buffer", "Mmap of Input file was
failed\n");
                return -1;
        }

        ///////////////////////////////////
        // FrameExtractor Initialization      //
        ///////////////////////////////////
        pFrameExCtx = FrameExtractorInit(FRAMEX_IN_TYPE_MEM, delimiter_h264,
sizeof(delimiter_h264), 1);
        file_strm.p_start = file_strm.p_cur = (unsigned char *)in_addr;
        file_strm.p_end = (unsigned char *)(in_addr + file_size);
        FrameExtractorFirst(pFrameExCtx, &file_strm);


        //////////////////////////////////////
        ///    1. Create new instance      ///
        ///      (SsbSipH264DecodeInit)    ///
        //////////////////////////////////////
        handle = SsbSipH264DecodeInit();
        if (handle == NULL) {
                LOG_MSG(LOG_ERROR, "Test_H264_Decoder_Line_Buffer", "H264_Dec_Init Failed.\n");
                return -1;
        }

        /////////////////////////////////////////////
        ///    2. Obtaining the Input Buffer      ///
        ///      (SsbSipH264DecodeGetInBuf)       ///
        /////////////////////////////////////////////
        pStrmBuf = SsbSipH264DecodeGetInBuf(handle, nFrameLeng);
        if (pStrmBuf == NULL) {
                LOG_MSG(LOG_ERROR, "Test_H264_Decoder_Line_Buffer", "SsbSipH264DecodeGetInBuf
Failed.\n");
                SsbSipH264DecodeDeInit(handle);
                return -1;
        }

        ////////////////////////////////////
        //  H264 CONFIG stream extraction //
        ////////////////////////////////////
        nFrameLeng = ExtractConfigStreamH264(pFrameExCtx, &file_strm, pStrmBuf,
INPUT_BUFFER_SIZE, 1);


        /////////////////////////////////////////////////////////////////
        ///    3. Configuring the instance with the config stream     ///
        ///       (SsbSipH264DecodeExe)                               ///
        /////////////////////////////////////////////////////////////////
        if (SsbSipH264DecodeExe(handle, nFrameLeng) != SSBSIP_H264_DEC_RET_OK) {
                LOG_MSG(LOG_ERROR, "Test_H264_Decoder_Line_Buffer", "H.264 Decoder Configuration
Failed.\n");
                return -1;
        }


        /////////////////////////////////////
        ///   4. Get stream information   ///
        /////////////////////////////////////
        SsbSipH264DecodeGetConfig(handle, H264_DEC_GETCONF_STREAMINFO, &stream_info);

        LOG_MSG(LOG_TRACE, "Test_H264_Decoder_Line_Buffer", "\t<STREAMINFO> width=%d
height=%d.\n", stream_info.width, stream_info.height);


        while(1) {
        #ifdef FPS
                gettimeofday(&start, NULL);
        #endif

                ///////////////////////////////////
                ///       5. DECODE               ///
                ///    (SsbSipH264DecodeExe)      ///
                ///////////////////////////////////
                if (SsbSipH264DecodeExe(handle, nFrameLeng) != SSBSIP_H264_DEC_RET_OK)
                        break;

        #ifdef FPS
                gettimeofday(&stop, NULL);
                time += measureTime(&start, &stop);
                frame_cnt++;
        #endif

                ///////////////////////////////////////////////
                ///    6. Obtaining the Output Buffer       ///
                ///      (SsbSipH264DecodeGetOutBuf)        ///
                ///////////////////////////////////////////////
                pYUVBuf = SsbSipH264DecodeGetOutBuf(handle, &nYUVLeng);

        #ifndef FPS
                write(out_fd, pYUVBuf, (stream_info.width * stream_info.height * 3) >> 1);
        #endif

                /////////////////////////////
                // Next H.264 VIDEO stream //
                /////////////////////////////
                nFrameLeng = NextFrameH264(pFrameExCtx, &file_strm, pStrmBuf,
INPUT_BUFFER_SIZE, NULL);
                if (nFrameLeng < 4)
                        break;
        }

#ifdef FPS
        LOG_MSG(LOG_TRACE, "Test_H264_Decoder_Line_Buffer",       \
                "Decoding Time : %u, Frame Count : %d, FPS : %f\n", time, frame_cnt,
(float)frame_cnt*1000/time);
#endif

        ////////////////////////////////////////
        ///    7. SsbSipH264DecodeDeInit     ///
        ////////////////////////////////////////
        SsbSipH264DecodeDeInit(handle);

        LOG_MSG(LOG_TRACE, "Test_H264_Decoder_Line_Buffer", "\n\n@@@ Program ends.\n");

        close(in_fd);
        close(out_fd);

        return 0;
}
```

## 7.2.3 MPEG4 Encoder Sample

| MPEG4 Encode |
|---|
| int Test_MPEG4_Encoder(int argc, char **argv)<br>{ |

```
        int             in_fd, out_fd;
        char            *in_addr;
        int             file_size;
        int             frame_count;
        int             frame_size;
        void            *handle;
        int             width, height, frame_rate, bitrate, gop_num;
        unsigned char   *p_inbuf;
        unsigned char   *p_outbuf;
        long            size;
        int             ret;
        struct stat     s;

#ifdef FPS
        struct timeval  start, stop;
        unsigned int    time = 0;
        int             frame_cnt = 0;
#endif


        if (argc != 8) {
                printf("Usage : mfc <YUV file name> <output filename> <width> <height> ");
                printf("<frame rate> <bitrate> <GOP number>\n");
                return -1;
        }

        //////////////////////////
        // Input/Output File Open //
        //////////////////////////
        in_fd   = open(argv[1], O_RDONLY);
        out_fd  = open(argv[2], O_RDWR | O_CREAT | O_TRUNC, 0644);
        if( (in_fd < 0) || (out_fd < 0) ) {
                printf("input/output file open error\n");
                return -1;
        }

        // get input file size
        fstat(in_fd, &s);
        file_size = s.st_size;

        // mapping input file to memory
        ////////////////////////////////////
        // Input/Output Buffer Memory Mapping //
        ////////////////////////////////////
        in_addr = (char *)mmap(0, file_size, PROT_READ, MAP_SHARED, in_fd, 0);
        if(in_addr == NULL) {
                printf("input file memory mapping failed\n");
                return -1;
        }

        width           = atoi(argv[3]);
        height          = atoi(argv[4]);
        frame_rate      = atoi(argv[5]);
        bitrate         = atoi(argv[6]);
        gop_num                 = atoi(argv[7]);

        frame_size      = (width * height * 3) >> 1;
        frame_count     = file_size / frame_size;
        printf("file_size : %d, frame_size : %d, frame count : %d\n", file_size, frame_size, frame_count);


        ////////////////////////////////////////////////////////////
        // 1. Create new instance and set the encoder parameters //
        //                (SsbSipMPEG4EncodeInit)                //
        ////////////////////////////////////////////////////////////
        handle = SsbSipMPEG4EncodeInit(SSBSIPMFCENC_MPEG4, width, height, frame_rate, bitrate,
gop_num);
        if (handle == NULL) {
                LOG_MSG(LOG_ERROR, "Test_Encoder", "SsbSipMPEG4EncodeInit Failed\n");
                return -1;
        }

        /////////////////////////////////////////////
        ///     2. Obtaining the Input Buffer      ///
        ///         (SsbSipMPEG4EncodeGetInBuf)     ///
        /////////////////////////////////////////////
        p_inbuf = SsbSipMPEG4EncodeGetInBuf(handle, 0);

        while(frame_count > 0)
        {
                printf("frame count : %d\n", frame_count);

                 ////////////////////////////////////
                 // Copy YUV data into input buffer //
                 ////////////////////////////////////
                memcpy(p_inbuf, in_addr, frame_size);
                in_addr += frame_size;

        #ifdef FPS
                gettimeofday(&start, NULL);
        #endif

                //////////////////////////////
                //      3. ENCODE          //
                // (SsbSipMPEG4EncodeExe)  //
                //////////////////////////////
                ret = SsbSipMPEG4EncodeExe(handle);

        #ifdef FPS
                gettimeofday(&stop, NULL);
                time += measureTime(&start, &stop);
                frame_cnt++;
        #endif

                ///////////////////////////////////////////////
                ///     4. Obtaining the Output Buffer      ///
                ///         (SsbSipMPEG4EncodeGetOutBuf)     ///
                ///////////////////////////////////////////////
                p_outbuf = SsbSipMPEG4EncodeGetOutBuf(handle, &size);

        #ifndef FPS
                write(out_fd, p_outbuf, size);
        #endif

                frame_count--;
        }
```

```
#ifdef FPS
	printf("Decoding Time : %u, Frame Count : %d, FPS : %f\n", time, frame_cnt,
(float)frame_cnt*1000/time);
#endif


	close(in_fd);
	close(out_fd);

	return 0;
}
```