09-7428-API-ZCH70

9-SEPT-2009

1.2

Application Programmers Interface for MPEG4 ASP Decoder

ABSTRACT:

Application Programmers Interface for MPEG4 Decoder

KEYWORDS:

MPEG4 ASP, Video codec

APPROVED:

Wang Zening

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	01-Dec-2008	Wang Zening	Initial Draft
1.0	10-Mar-2009	Ding Qiang	Refine for version 1.0 release
1.1	31-Mar-2009	Chen Qianzong	Update for ARM9/ARM11 release
1.2	9-Sep-2009	Chen Qianzong	Update DSV configuration

Table of Contents

1 Introduction

1.1 Purpose

This document gives the application programmer's interfaces to MPEG4-ASP / H263-BL Decoder library.

1.2 Scope

This document does not detail the implementation of the decoder. It only explains the APIs and data structures exposed to the application developer for using the decoder library

1.3 References

1.3.1 Standards

- ISO/IEC 14496-2:2003
  Information technology -- Coding of Audio-Visual Objects – Part2: Visual
- ITU-T H.263 video coding specification.
- ITU-T H.263 Annex X, Profiles and levels definition (SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS, Infrastructure of audiovisual services – Coding of moving video, 4/2001)

1.3.2 Freescale Multimedia References

- MPEG4 ASP Decoder Application Programming Interface – mpeg4_asp_dec_api.doc
- MPEG4 ASP Decoder Release notes - mpeg4_asp_dec_release_notes_arm9.doc
- MPEG4 ASP Decoder Release notes - mpeg4_asp_dec_release_notes_arm11.doc
- MPEG4 ASP Decoder Datasheet - mpeg4_asp_dec_datasheet_arm9.doc
- MPEG4 ASP Decoder Datasheet - mpeg4_asp_dec_datasheet_arm11.doc
- MPEG4 ASP Decoder Interface Header – mpeg4_asp_api.h

1.4 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
API	Application Programming Interface
ARM	Advanced RISC Machine
FSL	Freescale
ISO	International Standards Organization
ITU	International Telecommunication Union

MPEG	Moving Pictures Expert Group
ASP	MPEG4 Advanced Simple Profile
SP	MPEG4 Simple Profile
VOP	Video Object Plane
CDB	Configurable Decoding Buffer which means the decoding buffer are configured before decode every frame
DSV	Down Scaled Video

2 API Description

This section describes the APIs supported by the MPEG4 Decoder library. The salient features of the Decoder are mentioned below.

- Decoder shall be available as a static library and also as a shared library
- Decoder provides APIs along with a set of Data structures to facilitate the process of decoding
- Library exposes C-style API interfaces
- CPU Scalability

Decoder provides APIs to enable or disable some features of codec to get a better tradeoff between performance and visual quality.
Currently, there are 3 features which can be switched. Skip B gtames, P/B frames, enable DSV.
DSV is a scalability decoding option. When DSV is enabled, the video is down-sampled during decoding, the out put video size is half of the original width/height.

2.1 Data Structures

This section describes the data structures used in the decoder interface.

2.1.1 Decoder Handle

The handle of decoder is the only identifier of a certain decoder instance.

```
typedef void* MPEG4DHandle;
```

2.1.2 sMpeg4DecInitInfo

This structure holds initial information that the decoder needs to work.

```
typedef struct
{
    sMpeg4DecMemAllocInfo     sMemInfo;
    sMpeg4DecStreamInfo       sStreamInfo;
    S32                       s32MinFrameBufferCount;
} sMpeg4DecInitInfo;
```

Description of the structure *sMpeg4DecInitInfo*

sMemInfo

The decoder wouldn't allocate any block of memory internally and the application should be responsible for memory allocation according to decoder's request. This field indicates all the memory requirements expect frame buffers. See section 2.1.3 .

sStreamInfo

High level stream information after initial header parsing. See section 2.1.6 .

s32MinFrameBufferCount

The minimum number of frame buffers that decoder will need. This field is helpful for the application to allocate frame buffers, filled by decoder.

2.1.3 sMpeg4DecMemAllocInfo

The application should call *eMPEG4DQueryInitInfo*, see section 2.3.1, to get the initial memory requirement. Generally, memory allocated by the application can be classified into two types, fast and slow. Fast type memory would be SRAM blocks embedded in the chip and slow type would be external memory such as SDRAM, DDR, etc.
The application should try its best to allocate faster memory for a sFastMemBlks. If the platform only provides memory with the same access speed, the application can ignore the different speed requirement and allocate required memory to these 2 chunks.

```
typedef struct
{
    sMpeg4DecMemBlock sFastMemBlk;  /*!Fast memory Block */
    sMpeg4DecMemBlock sSlowMemBlk;  /*!Slow memory Block */
} sMpeg4DecMemAllocInfo;
```

Description of the structure *sMpeg4DecMemAllocInfo*

sFastMemBlk

Memory block structure to record fast memory requirement

sSlowMemBlk

Memory block structure to record slow memory requirement

2.1.4 sMpeg4DecMemBlock

This describes the memory chunk requirement details such as size, type, etc. The application shall allocate memory depending on the requirement and set the pointer in the space provided. The decoder shall use the memory given by the application.

```
typedef struct
{
    S32                    s32Size;       /*!< size of the memory block           */
    eMpeg4DecMemAlignType  eAlign;        /*!< alignment of the memory block      */
    void                   *pvBuffer;     /*!< pointer to allocated memory buffer */
} sMpeg4DecMemBlock;
```

Description of the structure ***sMpeg4DecMemBlock***

s32Size

The size of the memory required (filled by the decoder).

eAlign

The required alignment type of the memory block
The values are defined in the section 2.1.5 .

pvBuffer

This will be updated by the application based on the memory requirement.

2.1.5 eMpeg4DecMemAlignType

This enum holds the memory alignment type.

```
typedef enum
{
    E_MPEG4D_ALIGN_1BYTE = 0, /*!< buffer can start at any place    */
    E_MPEG4D_ALIGN_2BYTE,     /*!< start address's last 1 bit has to be 0  */
    E_MPEG4D_ALIGN_4BYTE,     /*!< start address's last 2 bits has to be 0 */
    E_MPEG4D_ALIGN_8BYTE,     /*!< start address's last 3 bits has to be 0 */
    E_MPEG4D_ALIGN_16BYTE,    /*!< start address's last 4 bits has to be 0 */
    E_MPEG4D_ALIGN_32BYTE,    /*!< start address's last 5 bits has to be 0 */
} eMpeg4DecMemAlignType;
```

2.1.6 sMpeg4DecStreamInfo

The application should call *eMPEG4DQueryInitInfo* to get the high level stream information.

```
typedef struct
{
/*VOS level*/
     S32 s32Profile; /*stream profile, 0 for SP, 1 for ASP*/
     S32 s32Level;   /*stream level*/
/*VOL level*/
    U16      u16PaddedFrameWidth;      /*!< Padded FrameWidth*/
    U16      u16PaddedFrameHeight;     /*!< Padded FrameHeight*/
    U16      u16ActFrameWidth;         /*!< Actual FrameWidth */
    U16      u16ActFrameHeight;        /*!< Actual FrameHeight*/
    U16      u16LeftOffset;            /*cropping origin x*/
    U16      u16TopOffset;             /*cropping origin y*/
} sMpeg4DecStreamInfo;
```

Description of structure *sMpeg4StreamInfo*

s32Profile

Stream Profile, 0 for SP, 1 for ASP, -1 for others.
If there is no this information in the stream, the default value is 1(ASP).

s32Level

Stream Level
If there is no this information in the stream, the default value is 7(level 3b).

u16PaddedFrameWidth

This item is the extended and padded frame width.
In the decoder, if the actual picture size if not 16 multiple, the decoding frame size will be extended to 16 multiple integers. And then it will be padded by a 16 pixel band for accelerating the decoding, see **Figure 1: Storage format and pointers for one padded output frame**.

Deleted: Figure 1

u16PaddedFrameHeight

This item is the extended and padded frame height

u16ActFrameWidth

This item is the actual frame width that is indicated in the stream header

u16ActFrameHeight

This item is the actual frame height that is indicated in the stream header

u16LeftOffset

Since the exported frame buffer is extended and padded, it is need to be cropped when display. This item indicate the cropping origin coordinate x

u16TopOffset

This item indicate the cropping origin coordinate y

2.1.7 sMpeg4DecYCbCrBuf

This Data structure encapsulates the decoded YCbCr buffer.

```
typedef struct
{
    unsigned char   *pu8YBuf;     /*!< Y Buf       must be 4 bytes aligned*/
    unsigned char   *pu8UBuf;     /*!< U Buf       must be 4 bytes aligned*/
    unsigned char   *pu8VBuf;     /*!< V Buf       must be 4 bytes aligned*/

    S32 s32YBufLength; /*size must be padded_width x padded_height, maybe need not this item*/
    S32 s32UBufLength; /*size must be padded_width x padded_height/4, maybe need not this item*/
    S32 s32VBufLength; /*size must be padded_width x padded_height/4, maybe need not this item*/
    void *pUsrTag           /*a Tag that may be used by App. App can use this tag to easily manage the buffers.
                              It's App     implementation dependent, decoder will not use or change this tag,
```

```
                                App can ignore this tag also.*/
} sMpeg4DecYCbCrBuffer;
```

Since the MPEG4 ASP decoder uses CDB scheme (see section 2.1.9), the output format will only be the padded YUV picture. Three pointers are used to present YUV plans respectively to allow the flexibility. The Y,U,V buffer should be continuous for the current version.
Please note that the meaningful picture is put at the left-top blue area if DSV is enabled

Description of structure *`sMpeg4DecYCbCrBuffer`*

pu8YBuf
: pointer to the Y plan of the padded decode buffer

pu8UBuf
: pointer to the U plan of the padded decode buffer

pu8VBuf
: pointer to the U plan of the padded decode buffer

s32YBufLength
: the size of Y plan of the padded decode buffer. It must be padded_width x padded_height

s32UBufLength
: the size of U plan of the padded decode buffer. It must be padded_width x padded_height/4

S32VBufLength
: the size of V plan of the padded decode buffer. It must be padded_width x padded_height/4

pUsrTag
: This field is reserved for the application use. How to use is up to the application. For example, the application can use this tag to mark the frame buffer. If the application doesn't need to use it, it can simply ignore this tag.

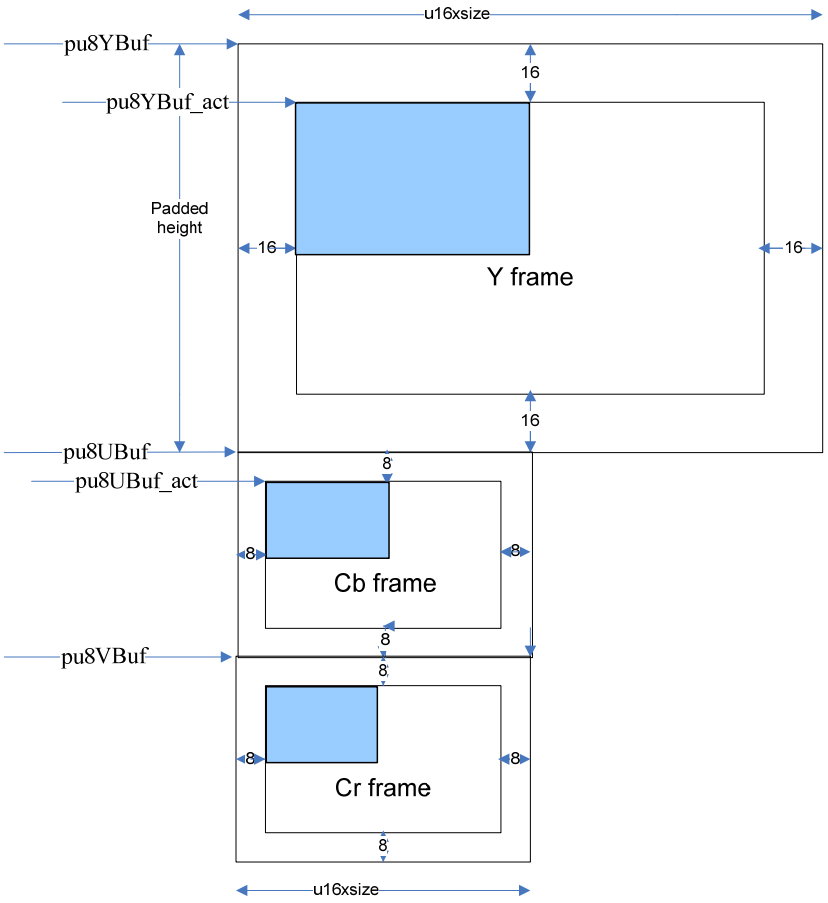


Figure 1: Storage format and pointers for one padded output frame

In the above diagram, the value 16 vertically and horizontally specifies the pad.

In case of YUV padded output format, to calculate the pointer to the actual data the following relations can be used:

pu8YBuf _act = pu8YBuf + ((16*u16xsize) + 16)
pu8UBuf _act = pu8UBuf + ((8*u16cxsize) + 8)

Note: The padded height calculation is based on the assumption that Y buffer is followed by Cb buffer.

2.1.8 sMpeg4DecAppCap

This structure can be used to facilitate the decoder to refine configuration according to the application/platform's setting. Currently the memory capacity is included. When the application query the initialization info from the decoder, it will tells the decoder how much fast memory and slow memory that the application can allocate for decoder. By this the decoder can post feasible memory requirement.

```
typedef struct
{
        S32 s32MaxFastMem;
        S32 s32MaxSlowMem;
} sMpeg4DecAppCap;
```

Description of structure *sMpeg4DecAppCap*

s32MaxFastMem

The maximum size of fast memory (in bytes) that the application can allocate for the decoder, -1 for unrestricted

s32MaxSlowMem

The maximum size of slow memory (in bytes) that the application can allocate for the decoder, -1 for unrestricted

2.1.9 sMpeg4DecFrameManager

This decoder uses a Configurable Decoding Buffer (CDB) mechanism to deal with the frame buffers. This CDB mechanism means that the frame buffer which would be decoded into is appointed by the application before decoding every frame.

By adopting this scheme, the application would manage all frame buffers and thus enable much more flexibility for system level design. It is up to the application's decision if copying the decoded frame buffer into display domain or directly rendering it. The CBD mechanism is encapsulated by a pair of callback functions which should be implemented by the application.

In brief the decoder will use one callback functions to ask a decoding buffer before decoding one frame and might use another callback function to notify the application that the buffer provided cannot be used because of some reasons, for example, this frame contains a reference frame. Additional callback function is used to notify the application that the frame buffer can be reused.

In order to clarify the concept and simplify the API, these 3 callback functions are grouped into a structure named FrameManager. The prototype of these 3 callback functions are described in section (section 2.1.10 and 2.1.11)

Deleted: 2.1.10

Deleted: 2.1.11

```
typedef struct _sMpeg4DecFrameManager
{
    cbGetOneFrameBuffer       GetterBuffer;
    cbRejectOneFrameBuffer    RejectorBuffer;
    cbReleaseOneFrameBuffer   ReleaseBuffer;
    void*                     pvAppContext
}sMpeg4DecFrameManager;
```

Description of structure *FrameManager*

GetterBuffer

The callback functions for getting frame buffers

RejectorBuffer

The callback functions for rejecting frame buffers

ReleaseBuffer

The callback function is used to notify the application the decoder will not use this buffer.

pvAppContext

The application context that set by the application and must be passed back to the application when getting or rejecting buffers.

2.1.10 Callback functions for getting frame buffers

The callback functions used for getting frame buffers is defined below; it will get one frame buffer for decoder. This callback function is implemented by the the application.

Prototype:

```
typedef sMpeg4DecYCbCrBuffer * (*cbGetOneFrameBuffer)( void* pvAppContext);
```

Arguments:

`pvAppContext` the App context that is registered with the callback.

Return value:

It must be a frame buffer with **padded picture size** (section 2.1.6 and 2.1.7), the pointers of YUV plans must be 4 bytes aligned. When the application can't provide frame buffer anymore, it can return NULL, and decoder will stop decoding and return E_MPEG4D_NO_FRAME_BUFFER.

2.1.11 Callback functions for rejecting frame buffers

It is possible that the gotten frame buffer for decoding the current frame still stored the reference data so the decoder has to reject the buffer. After the rejection by calling this callback function, the decoder will invoke *cbGetOneFrameBuffer* to ask for frame buffer again.
A rejected buffer should not be provided to decode (via *cbGetOneFrameBuffer*) immediately.
If the stream do not has B-Frame (such as MPEG4 SP), the rejected buffer must not be provided to decoder again in the next invoking of *cbGetOneFrameBuffer*, if the Stream has B-Frame, it must wait 2 times.

Prototype:

```
typedef void (*cbRejectOneFrameBuffer)( sMpeg4DecYCbCrBuffer * mem_ptr, void*
pvAppContext);
```

Arguments:

`mem_ptr` A rejected frame buffer

`pvAppContext` The App context the registered.

Return value:

None

2.1.12 Callback functions for release frame buffers

It is used to release one frame buffer. It means the decoder notify the application this frame buffer don't need occupied as a reference frame or for post process.

Prototype:

```
typedef void (*cbReleaseOneFrameBuffer) ( sMpeg4DecYCbCrBuffer * mem_ptr, void*
pvAppContext);
```

Arguments:

`mem_ptr` A released frame buffer

`pvAppContext` The App context the registered.

Return value:

None

2.2 Enumerations and Typedefs

2.2.1 Library API Return codes

This enum holds the return types of the APIs.

```
typedef enum
{
        /* Successfull return values */
        E_MPEG4D_SUCCESS = 0,      /* Success */
        E_MPEG4D_NO_OUTPUT,       /* decoded a frame but didn't finish, or it's NULL frame */
        E_MPEG4D_FRAME_SKIPPED    /* skipped this frame*/
        /* Successful return with a warning, decoding can continue */

        /* Recoverable error return, correct the situation and continue */
        E_MPEG4D_NOT_ENOUGH_BITS=31,  /* Not enough bits are provided   */
        E_MPEG4D_OUT_OF_MEMORY,       /* Out of Memory                      */
        E_MPEG4D_WRONG_ALIGNMENT,     /* Incorrect Memory Alignment         */
        E_MPEG4D_SIZE_CHANGED,        /* Image size changed           */
        E_MPEG4D_INVALID_ARGUMENTS,   /* API arguments are invalid          */
        E_MPEG4D_NO_HEADER_INFO,      /*no header in the stream when start to decode*/

        /* irrecoverable error type */
        E_MPEG4D_ERROR_STREAM=51,     /* Errored Bitstream          */
        E_MPEG4D_FAILURE,             /* Failure            */
        E_MPEG4D_UNSUPPORTED,         /* Unsupported Format          */
        E_MPEG4D_NO_FRAME_BUFFER      /* decoder can't get frame buffer */

} eMpeg4DecRetType;
```

2.3 Application Programmer Interface Functions

2.3.1 Query Initialization information

The application uses this function to query the initialization info such as memory requirement and minimal decoder buffer number of the decoder for a specific stream. This function would return the initialization info for the decoder with being fed with the stream header. The decoder will parse the input bit stream to determine the type of video content, and count the memory requirement depends on the parsed info.
The application will use these information to allocate the requested memory block (chunks) by setting the pointers of *sFastMemBlks and sSlowMemBlks* in *sMpeg4DecInitInfo.sMemInfo* structure. And the application need to use the s32MinFrameBufferCount to prepare the frame manager.

Prototype:

```
eMpeg4DecRetType  eMPEG4DQueryInitInfo(sMpeg4DecInitInfo *psInitInfo,
                                unsigned char  *pu8BitBuffer,
                                signed long  int s32NumBytes,
                                sMpeg4DecAppCap *AppCap);
```

Arguments:

- psInitInfo [out] pointer to the initialization info
- pu8BitBuffer [in] pointer to the bitstream buffer, the input data must be 4 byte aligned. And the buffer size should be multiple of 4.

- s32NumBytes [in] length of the input buffer
- pAPPCap [in] max fast and slow memory that the application can provide

Return value:

eMpeg4DecRetType Tells whether this function executed successfully or not.
Enumeration is described in the above section. Return values are –

E_MPEG4D_SUCCESS	Function successful
E_MPEG4D_INVALID_ARGUMENTS	API arguments are invalid
E_MPEG4D_NO_HEADER_INFO	no header in the input stream
E_MEPG4D_ERROR_STREAM	detected error in the input stream
E_MPEG4D_NOT_ENOUGH_BITS	Input data is not enough to decoder headers
E_MPEG4D_WRONG_ALIGNMENT	input data alignment error, must be 4 bytes aligned
E_MPEG4D_UNSUPPORTED	unsupported Profile/level/parameter

2.3.2 Create a Decoder instance

After getting the initial information and allocating memory for the decoder, the application can create the decoder instance. *sMpeg4DecInitInfo.sStreamInfo* should be the same as what the application get by invoking *eMPEG4DQueryInitInfo*.
A void pointer will be output as the decoder handle.

Prototype:

```
eMpeg4DecRetType  eMPEG4DCreate (sMpeg4DecInitInfo* psInitInfo, sMpeg4DecFrameManager*
                                 pFrameManager,      MPEG4DHandle* phMp4DecHandle);
```

Arguments:

- psInitInfo [in] Initialization info such as allocated memory.
- pFrameManager [in] a pointer to a frame manager which is used to get and reject buffer.
- phMp4DecHandle [out] pointer of the created decoder handle

Return value:

eMpeg4DecRetType Tells whether decoder has been successfully created or not.
Enumeration is described in the above section. Return values are –

E_MPEG4D_SUCCESS	Function successful.
E_MPEG4D_INVALID_ARGUMENTS	API arguments are invalid
E_MPEG4D_OUT_OF_MEMORY	the application did not provide enough memory

2.3.3 Decode Frame

eMPEG4DdecodeFrame is the main decoder function which should be called for decoding each frame.
The input to this function is bitstream for an encoded frame and size of the encoded stream. The input data address should be 4 Byte aligned.
The input data must at least include one frame or a header such as VOS, VO, VOL.
At the first invoking of this function, the application should feed data which include at least the stream header such as VOS, VO, VOL.

After decoding, how many input data is consumed will be return by the 3rd argument. If the input buffer contains more than one frame, this function should be invoked again with remained data. Buffer requirement is the same as the previous call.

Prototype:

```
EXTERN eMpeg4DecRetType eMPEG4DDecodeFrame (MPEG4DHandle hMp4DecHandle,
                                            void *pvBSBuf,
                                            long *s32NumBytes);
```

Arguments:

- hMp4DecHandle [in] Decoder handle.
- pvBSBuf [in]Stream data buffer for an encoded MPEG4-SP video frame, the address of pvBSBuf point to should be 4bytes alignment. And the buffer size should be multiple of 4.
- s32NumBytes [in/out] Length of the stream data buffer in number of bytes. Decoder will also use this argument as an output to inform the application how much data are consumed.

Return value:

eMpeg4DecRetType Tells whether frames were decoded successfully or not.
Enumeration is described in the above section. Return values are –

E_MPEG4D_SUCCESS	One frame is successfully decoded.
E_MPEG4D_NO_OUTPUT	at the first invoking, only decoded a header
E_MPEG4D_SIZE_CHANGED	decoded a new header and detected that in the new header frame size is changed, App need to restart the decoding process
E_MPEG4D_FRAME_SKIPPED	a frame is skipped, need not to get frame
E_MPEG4D_INVALID_ARGUMENTS	API arguments are invalid
E_MPEG4D_NO_HEADER_INFO	no header in the stream when start to decode
E_MPEG4D_NOT_ENOUGH_BITS	Input data is not enough for decoding a frame
E_MPEG4D_WRONG_ALIGNMENT	input data alignment error, should be 4 bytes aligned
E_MEPG4D_ERROR_STREAM	detected unconcealable error in the input stream
E_MPEG4D_NO_FRAME_BUFFER	decoder can't get frame buffer
E_MPEG4D_UNSUPPORTED	unsupported Profile/level/parameter

2.3.4 Get output frame

After successfully decoding one frame, the application should call eMPEG4DGetOutputFrame to get the frame to be displayed.
If *ppsOutBuffer is set to NULL, it means the decoder does not have proper frame be exported.
The cropping work will be carried out in the application's scope.
This API would be called when the following APIs notify that a frame has been successfully decoded.

- *eMPEG4DDecodeFrame*

Since the decode order and display order might be different (for the ASP), the decoder may hold more than one decoded frame inside the decoder. Under some circumstance such as the end of decoding or seeking the stream, the application might need to get the frame several times. At this

time, the application needs to call function eMPEG4DFlushFrame to flush the last frame. Detailed flush procedure is described in section 3.6 .

Prototype:

```
        eMpeg4DecRetType eMPEG4DGetOutputFrame (MPEG4DHandle hMp4DecHandle,
sMpeg4DecYCbCrBuffer** ppsOutBuffer);
```

Arguments:

- pMp4DecHandle [in] Decoder object handle
- ppsOutBuffer [out] output argument to record the decoded picture

Return value:

eMpeg4DecRetType Tells whether this function executed successfully or not.
Enumeration is described in the above section. Return values are –

E_MPEG4D_SUCCESS	Function successful.
E_MPEG4D_INVALID_ARGUMENTS	API arguments are invalid
E_MPEG4D_FAILURE	Error

2.3.5 Flush one frame

Flush one frame to avoid the frame was reserved by decoder.

Prototype:

```
        eMpeg4DecRetType eMPEG4DFlushFrame (MPEG4DHandle hMp4DecHandle);
```

Arguments:

- pMp4DecHandle [in] Handle of the Decoder that need to be deleted.

Return value:

eMpeg4DecRetType Tells whether this function executed successfully or not.
Enumeration is described in the above section. Return values are –

E_MPEG4D_SUCCESS	Function successful.
E_MPEG4D_FAILURE	Error

2.3.6 Set Parameter

This function sets some indications to decoder to configure the additional features of the decoder. The additional features *skip B frame*, *skip B and P frame*,*enable DSV*.
These configurations can be set and reset at anytime.

- For *skip B and P frame* setting, the decoder will record this setting and decode I frame only. The application may reset this setting, it is, resume from skpping. Because the P frame might be used as the reference frame and if it has been skipped, the following B frame would not be decoded correctly. So, the decoder will still skip the B and P frame until meeting an I frame.
- For *enable DSV* setting, the decoder will output video with half size.The application may reset this setting but the video quality will not recover until meeting an I frame.
- These features could be configured at the same time by a 32-bit parameter *`eParaName`*
    - bit0 skip B frame

One means enable the shortcut, “skip B frames of streams”, zero means disable the shortcut.
.

- bit1 skip P and B frame
  One means enable the shortcut, “skip P and B frames of streams”, zero means disable the shortcut.
- bit2 Enable DSV
  One means enable the shortcut, “enable down-scaled video”, zero means disable the shortcut.
- bit3-31 reserved

Prototype:

```
eMpeg4DecRetType eMPEG4DSetParameter (MPEG4DHandle hMp4DecHandle, int eParaName)
```

Arguments:

- pMp4DecHandle [in] Decoder object handle
- eParaName [in] CPU scalability parameter

Return value:

eMpeg4DecRetType Tells whether this function executed successfully or not.
Enumeration is described in the above section. Return values are –

E_MPEG4D_SUCCESS Function successful.

2.3.7 Get Parameter

This function is used for querying the current configuration of the decoder.
Just like the eMPEG4DSetParameter (section 2.3.6), this function takes the parameter names from the enumeration eMPEG4DParameter.
The configuration will be modified by *eMPEG4DdecodeFrame* if the DSV feature is not supported.

Prototype:

```
eMpeg4DecRetType eMPEG4DGetParameter (MPEG4DHandle hMp4DecHandle, U32* pu32ParaValue )
```

Arguments:

- pMp4DecHandle [in] Decoder object handle
- pu32ParaValue [out] the value of the parameters

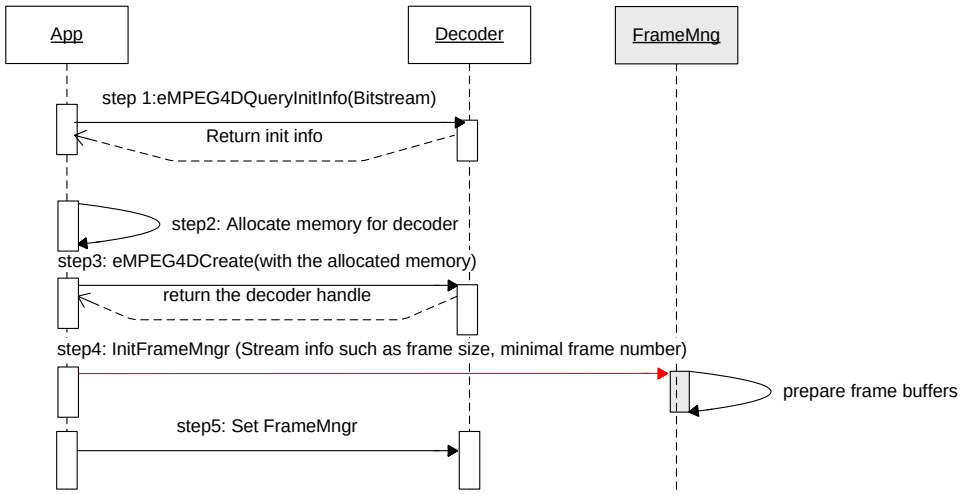
Return value:

eMpeg4DecRetType Tells whether this function executed successfully or not.
Enumeration is described in the above section. Return values are –

E_MPEG4D_SUCCESS Function successful.

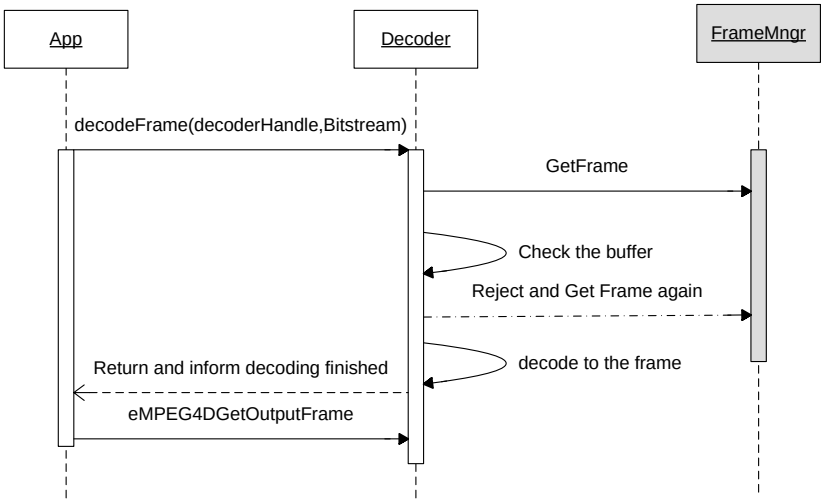
3 Decoder Usage

3.1 Initialization



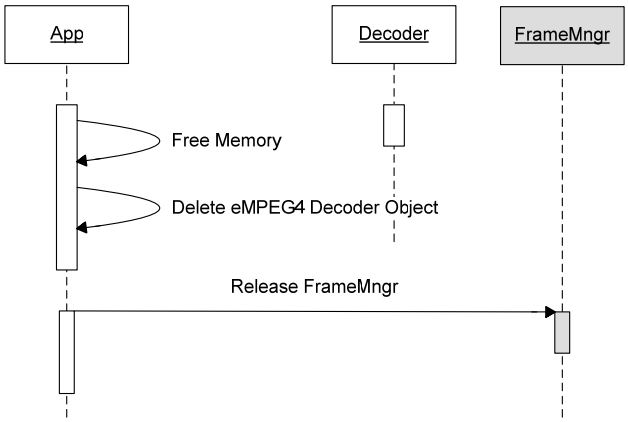
As the above figure illustrated, there are 5 steps for initialization of decoder, of cause, the step 4 which is red colored is out of decoder's scope, and the application may have its own decision on this step.

3.2 Frame Decode



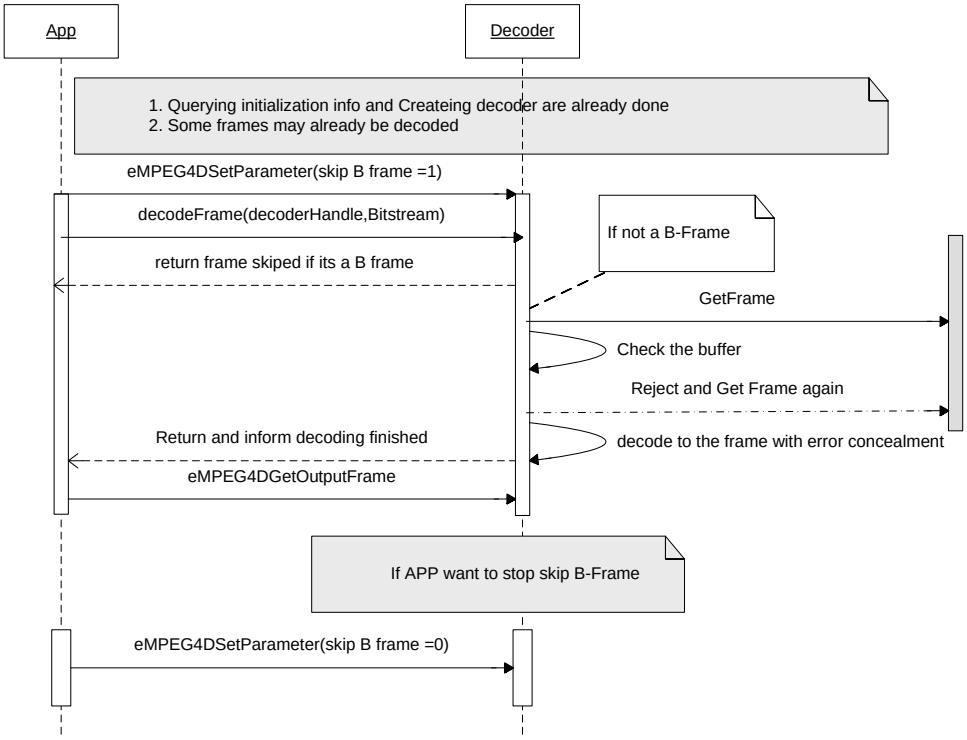
As the above figure illustrated, when decoding, decoder may invoke Frame Manager many times to get acceptable buffers. **The Frame Manager should hold more than 3 buffers at least for MPEG4 ASP decoder.**

3.3 Finish



As shown in figure above, the finish is simple, application free the memory block which is allocated in the initialization step3, and maybe ask Frame manager to release frame buffers

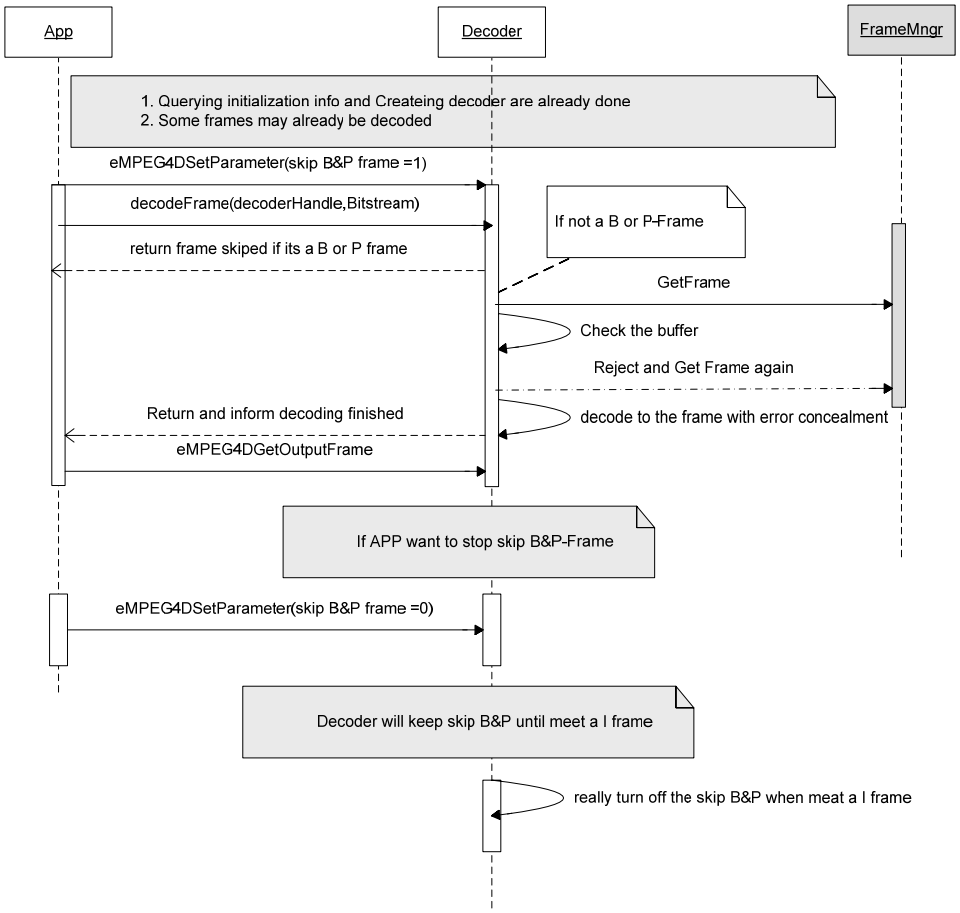
3.4 Use skip B Frame feature



As shown in the figure above, using skip B-Frame feature can be performed by switching the related parameter via API function *eMPEG4DSetParameter*.
This figure illustrated frame decoding case, the packet decoding case is just the same.

3.5 Use skip B and P Frame feature

As shown in figure below, using skip B&P-Frame feature can be performed by switching the related parameter via API function *eMPEG4DSetParameter*.
This figure illustrated frame decoding case, the packet decoding case is just the same.



3.6 Use flush out feature

As shown in figure below, using flush out feature can be performed by via API function *eMPEG4DFlushFrame*.