08-6289-SIS-ZCH66

8/11/2009

2.0

Application Programmers Interface for MPEG2 Decoder

ABSTRACT:

Application Programmers Interface for MPEG2 Decoder

KEYWORDS:

Multimedia codecs, Video, MPEG2

APPROVED:

Wang Ze Ning

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	26-Jun-2006	Manoj Arvind	Initial Draft
1.0	30-Jun-2006	Manoj Arvind	Incorporated Review comments
1.1	22-Aug-2006	Durgaprasad S.Bilagi	Modifications of API to support decoding at frame boundary
1.2	22-Aug-2006	Manoj Arvind	Updated the API
1.3	01-Sep-2006	Durgaprasad S.Bilagi	Updated the API
1.4	09-Oct-2006	Durgaprasad S.Bilagi	Added the bitrate element into the structure
1.5	22-Oct-2007	Eagle Zhou	Add comments for display frame size unaligned by 16 pixels
1.6	05-Nov-2007	Eagle Zhou	Add error handling, update comments for API return code
1.7	08-Nv_2007	Wang Zening	Add Direct rendering change
1.8	17-Mar-2008	Eagle Zhou	Modify callback schema, and the old interface still be compatible.
1.9	27-June-2008	Eagle Zhou	Add API version information, release decoder description and demo protection return type
2.0	11-Aug-2009	Eagle Zhou	Add feature for skipping B frames

Table of Contents

Introduction 4
1.1 Purpose 4
1.2 Scope 4
1.3 Audience Description 4
1.4 References 4
1.4.1 Standards 4
1.4.2 Freescale Multimedia References 4
1.5 Definitions, Acronyms, and Abbreviations 5
1.6 Document Location 5
2 API Description 6
2.1 Data Structures 6
2.2 Enumerations and Typedefs 10
2.2.1 Library API Return codes 10
2.2.2 Functional State of Decoder 10
2.2.3 Memory Alignment 10
2.2.4 Callback type 11
2.2.5 Return for Callback setting 11
2.2.6 Memory type 11
2.2.7 Decoding Scheme 11
2.2.8 Buffer getter 13
2.2.9 Buffer Rejecter 13
2.2.10 Buffer Releaser 13
2.2.11 Buffer Manager 14
2.3 Application Programmer Interface Functions 14
2.3.1 Call back register function. 14
2.3.2 Query Memory 14
2.3.3 Initialization 15
2.3.4 Re-Query Memory 15
2.3.5 Re-Initialization 16
2.3.6 Decode 16
2.3.7 Release 17
2.3.8 Input buffer interface 17
2.3.9 Debug Logging 18
2.3.10 Set a Buffer Manager 18
2.3.11 Set Callback Functions 18
2.3.12 API Version 19
2.3.13 Skip B Frames 19
Example Lib Usage 20

Introduction

1.1 Purpose

This document gives the application programmer's interface to MPEG2-MP@ML Decoder library. The document also illustrates an example of an application written using these API's.

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 Audience Description

The reader is expected to have basic understanding of video processing and video coding standards.

1.4 References

1.4.1 Standards

- ISO/IEC 13818-2:2000 Information technology -- Generic coding of moving pictures and associated audio information: Video

1.4.2 Freescale Multimedia References

Table 1. MPEG2 Decoder Requirements Book – mpeg2_dec_reqb.doc
Table 2. MPEG2 Decoder Test Plan – mpeg2_dec_test_plan.doc
Table 3. MPEG2 Decoder Release notes – mpeg2_dec_release_notes.doc
Table 4. MPEG2 Decoder Test Results – mpeg2_dec_test_results.doc
Table 5. MPEG2 Decoder Performance Results – mpeg2_dec_perf_results.doc
Table 6. MPEG2 Interface Header – mpeg2_dec_api.h
Table 7. MPEG2 Application code – MPEG2DecTestApp.c

1.5 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
API	Application Programming Interface
ARM	Advanced RISC Machine
DCT	Discrete Cosine Transform
DSP	Digital Signal Processing
FSL	Freescale
IDCT	Inverse Discrete Cosine Transform
IPU	Image Processing Unit
ISO	International Standards Organization
ITU	International Telecommunication Union
MC	Motion Compensation
ME	Motion Estimation
MPEG	Moving Pictures Expert Group
ML	Main Level
MP	Main Profile (in MPEG-2)
RVDS	RealView Development Suite for ARM
UNIX	Linux PC x/86 C-reference binaries
VLD	Variable Length Decoding
TBD	To be decided

1.6 Document Location

docs/mpeg2_dec

2 API Description

2.1 Data Structures

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

sMpeg2DecObject

This is the main data structure which should be passed to all the decoder functions. The definition of the structure is given below.

```
typedef struct
{
    sMpeg2DecMemAllocInfo   sMemInfo;    /*!< memory requirements info */
    sMpeg2DecoderParams     sDecParam;   /*!< decoder parameters       */
    void                    *pvMpeg2Obj; /*!< decoder library object   */
    void                    *pvAppContext; /*!< Anything app specific  */
    eDecodeState          eState; /*!< Indicates current Decoder State */
    int (* ptr_cbkMPEG2DBufRead) (int *, unsigned char **, int , void *);

} sMpeg2DecObject;
```

Description of structure *sMpeg2DecObject*

sMemInfo

This is memory information structure. This is further described later.

sDecParam

The output of the decoder is encapsulated in this structure. This is described later.

pvMpeg2Obj

This is an internal video object context for the decoder and application should not change this.

pvAppContext

This space is provided for the application to keep its context and the decoder does not use it.

eState

Indicates current decoder state

ptr_cbkMPEG2DBufRead

Call back function pointer that needs to be set by the application

sMpeg2DecMemAllocInfo

This structure holds the pre allocated memory chunks for decoding. The decoder memory requirements are given to the application when *eMPEG2DQuerymem* is called. The decoder specifies number of memory chunks needed by filling *s32NumReqs* in this structure. For each memory chunk, size, type, align parameters are set in asMemBlks array based on decoder requirement. Application shall allocate the memory required by the decoder by looking at this structure before initializing the decoder.

```
typedef struct
 {
        int                     s32NumReqs;
        int                     s32BlkNum ;_
sMpeg2DecMemBlock  asMemBlks[MAX_NUM_MEM_REQS];
        int                     s32MinFrameBufferNum;
 } sMpeg2DecMemAllocInfo;
```

Description of structure *sMpeg2DecMemAllocInfo*

s32NumReqs

Number of memory blocks (chunks) required. Decoder will set this to required value when *eMPEG2DQuerymem and eMPEG2D_Re_Querymem* functions are called. The application has to allocate the specified number of memory chunks.

s32BlkNum

This field indicates the index of the starting memory block. The test application has to allocate memory for s32NumReqs memory blocks starting from the memory block indexed by this field.

asMemBlks

Array of memory block structure, for each request defined in *s32NumReqs* application should pre-allocate the memory. The function *s32AllocateMem2Decoder* illustrates the usage.

s32MinFrameBufferNum

A integer number that indicates the minimum frame buffers that need by the decoder during decoding, this value will be used for correctly creating the frame buffer manger. This value will be got after invoking eMPEG2D_Re_Querymem()

MAX_NUM_MEM_REQS is the maximum number of memory chunk requests the decoder can make. Currently this value is set to 30.

sMpeg2DecMemBlock

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 library shall use the memory given by the application.

```
typedef struct
 {
    int         s32Size;
    int         s32Type;
    int          s32Priority;
    int         s32Align;
    void        *pvBuffer;
 } sMpeg2DecMemBlock;
```

Description of the structure *sMpeg2DecMemBlock*

s32Size

The size (in bytes) of the memory required to be allocated by the test application (filled by the decoder).

s32Type

This field is an informative field (filled by the decoder). This can be used by the test application to decide the memory pool (fast memory / slow memory / memory to be shared

with driver etc) from which memory needs to be allocated for the decoder. This field is not used in the present release.

s32Align

The alignment required by the memory block. The values are defined in Memory Alignment(filled by the decoder). This field is not used in the present release.

pvBuffer

The pointer of the memory allocated by the application should be populated in this field. This will be used by the decoder to populate pointers used internally.

s32Priority

This field is not used in the present release.

sMpeg2DecoderParams

Data structure for the decoder parameters. All the values of this structure are filled by decoder after decoding a frame.

```
typedef struct
{
   sMpeg2DecYCbCrBuf    sOutputBuffer;
   signed char          *p8MbQuants;
   unsigned short int   u16FrameWidth;
   unsigned short int   u16FrameHeight;
   unsigned int         bitrate;
} sMpeg2DecoderParams;
```

Description of the structure *sMpeg4DecoderParams*

sOutputBuffer

Decoded output is stored in this structure.

p8MbQuants

Pointer to the memory, which stores the quantization value for the decoded frame as required by the IPU for post processing. The quant values are specified in raster scan order of MBs, one byte for each MB. This is not used in the current release.

u16FrameWidth

Display width of the frame. The width of the frame may be not a multiple of 16.

u16FrameHeigth

Display height of the frame. The height of the frame may be not a multiple of 16.

Bitrate

Bitrate of the video bitstream.

[Note]

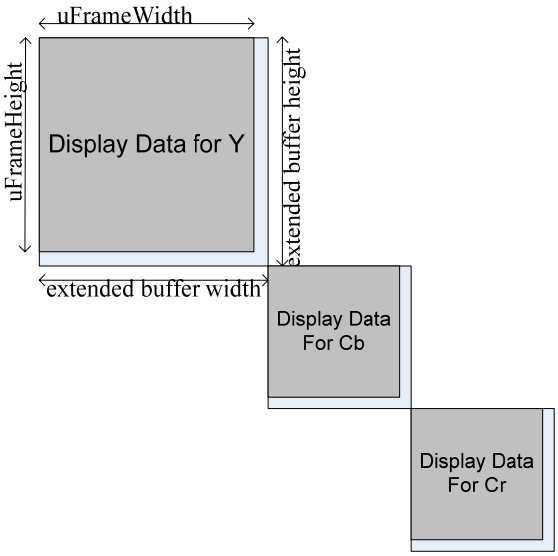
The output buffers represented by sOutputBuffer are aligned by 16 pixels for width and height. If u16FrameWidth or u16FrameHeight is not multiple of 16, *application need to crop the valid data from the output buffers*.

For example:

If uFrameWidth=160, uFrameHeight=120, then application can compute the real output buffer size according below formula:

[(160+15)&0xFFFFFFF0 , (128+15)&0xFFFFFFF0)= [160,128]

The format for buffer is listed below:



sMpeg2DecYCbCrBuf

This Data structure encapsulates the decoded YCbCr buffer.

```
typedef struct
{
    unsigned char       *pu8YBuf;
    int                 s32YBufLen;
} sMpeg2DecYCbCrBuffer;
```

Description of the structure *sMpeg2DecYCbCrBuffer*

pu8YBuf

The decoder generates Y, Cb, Cr output in 4:2:0 format. The output is stored in a continuous buffer with Y component, followed with Cb and Cr components. The start address of the Y component is populated by the decoder in this field. The Cb and Cr addresses must be computed by the application. If the display width and height is not multiple of 16 pixels, the buffer will be extended to multiple of 16 for Y,Cb,Cr respectively by decoder.

s32YBufLen

Length of the Y Buffer. It is set by the decoder. Application developer need not set it, if set it will be overridden by Decoder. This field is used by application to calculate the offset of the Cb and Cr components (see *eMPEG2KevWrite*).

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_MPEG2D_SUCCESS = 0,    /*!< Success */
    E_MPEG2D_PARTIAL_DECODE, /*!Only one field is decoded*/
    E_MPEG2D_ENDOF_BITSTREAM,/*!< End of Bit Stream */
    E_MPEG2D_FRAME_READY,/* Frame is available to application */
    E_MPEG2D_FRAME_SKIPPED, /*Frame is skipped by decoder*/
    E_MPEG2D_NOT_ENOUGH_BITS = 31,/*!<Not enough bits are provided*/
    E_MPEG2D_OUT_OF_MEMORY,        /*!< Out of Memory*/
    E_MPEG2D_WRONG_ALIGNMENT,      /*!< Incorrect Memory alignment*/
    E_MPEG2D_SIZE_CHANGED,         /*!< Image size changed*/
    E_MPEG2D_INVALID_ARGUMENTS,    /*!< API arguments are invalid*/
    E_MPEG2D_DEMO_PROTECT          /* the output is corrupted by demo
                                   protection    */
    E_MPEG2D_ERROR_STREAM = 51,    /*!< Errored Bitstream. In such case,
                                   the decoder can work normally, but will
                                   output bad data. The application decide
                                   whether continue decode the current
                                   stream.*/
    E_MPEG2D_FAILURE,              /*!< Failure. In such case, application
                                   should stop decoding current stream.*/
    E_MPEG2D_UNSUPPORTED,          /*!< Unsupported Format */
    E_MPEG2D_NO_IFRAME,       /*!< MPEG2D_first frame is not an I frame */
    E_MPEG2D_SIZE_NOT_FOUND, /*!< Frame size not found in bitstream */
    E_MPEG2D_NOT_INITIALIZED /*!< Decoder Not Initialised*/
} eMpeg2DecRetType;
```

2.2.2 Functional State of Decoder

This enum holds the current functional state of the Decoder.

```
typedef enum
{
   E_MPEG2D_INVALID = 0,      /*!< Invalid Decoder State */
   E_MPEG2D_PLAY,           /*!< Decoder is decoding frames */
   E_MPEG2D_FF,            /*!< Decoder is skipping frames */
   E_MPEG2D_REW             /*!< Decoder is skipping frames in a direction
                       opposite to PLAY and FF. Current Decoder
                       doesn't support REW feature */
} eDecodeState
```

2.2.3 Memory Alignment

This enum holds the memory alignment type.

```
typedef enum
{
    E_MPEG2D_ALIGN_NONE = 0,/*!< buffer can start at any place */
    E_MPEG2D_ALIGN_HALF_WORD, /*!< start address's last bit has to be 0*/
    E_MPEG2D_ALIGN_WORD   /*!< start address's last 2 bits has to be 0 */
```

```
} eMpeg2DecMemAlignType;
```

2.2.4Callback type

This enumeration holds the callback function type

```
typedef enum
{
    E_GET_FRAME =0,
    E_REJECT_FRAME,
    E_RELEASE_FRAME,
} eCallbackType;
```

`E_GET_FRAME` – register buffer getter callback function

`E_REJECT_FRAME` – register buffer rejecter callback function

`E_RELEASE_FRAME` – register buffer releaser callback function

2.2.5Return for Callback setting

This enumeration holds the return value of callback setting

```
typedef enum
{
   E_CB_SET_OK =0,
   E_CB_SET_FAIL,
} eCallbackSetRet;
```

`E_CB_SET_OK` – callback function is set successfully

`E_CB_SET_FAIL` – callback function is not set successfully

2.2.6Memory type

The following defines specifies the memory types of the memory blocks requested by the decoder. In the current release, the decoder doesnot populate this field. This will be used in future releases.

2.2.7Decoding Scheme

This release of MPEG2 decoder supports MP@ML with 4:2:0 output and video frames with height and width which are multiples of 16.

INPUT

The decoder works on a frame boundary. For every call of the decode, the decoder raises a callback function, *cbkMPEG2DBufRead*. The application needs to identify the frame boundary of the bitstream for decoding and must populate the address of the input bitstream and the length of the bitstream corresponding to the context.

1) When the call back function is raised from the *eMPEG2D_Re_Querymem*, the test application has to provide an input buffer with bitstream data including the

sequence start code (beginning of the bitstream) till just before the first picture start code [00 00 01 00](exclusive of start code).

2) When the call back function is raised from the first call to the decode, the test application must provide input buffer with bitstream data including the sequence start code till just before the second picture start code (exclusive of the second start code)
3) For the next call back from the decoder, the test application must provide input buffer with bitstream data starting from the second picture start code till just before the third picture start code (exclusive of the third picture start code).
4) For subsequent call backs, the test application must provide bitstream data as described in (3)
5) For the very last video frame, the test application must provide input buffer with bitstream data after the last picture start code till the end of bitstream sequence.

OUTPUT

The decoder returns control to the application at the end of decoding one frame / partial frame of video. The enum `E_MPEG2D_FRAME_READY` indicates the availability of one completely decoded frame. Enum `E_MPEG2D_PARTIAL_DECODE` is used to signal partial decoding of a frame. The enum E_MPEG2D_FRAME_SKIPPED indicates the frame is skipped by decoder. The decoder can be called till the decoder returns the enum `E_MPEG2D_ENDOF_BITSTREAM,` which indicates the end of sequence for that particular bitstream. The address of the frame to be displayed and the length will be populated in the structure `sMpeg2DecYCbCrBuffer`. The decoded output format will be 4:2:0, with Y(Luma), Cb and Cr components in a continuous memory. The starting address of the frame is populated on the structure element `pu8YBuf`. The total size of the Luma (Y) of video frame is *s32YBufLen.* The offset for start of Cb component is *s32YBufLen*. The length of the Cb buffer is *s32YBufLen/4*. The offset for start of Cr component is *s32YBufLen*1.25*. The length of the Cr buffer is *s32YBufLen/4*. A sample implementation for output write is illustrated in *eMPEG2KevWrite (MPEG2DecTestApp.c)*.

MEMORY QUERYING MECHANISM

The application queries the decoder for memory requirement. The APIs, viz *eMPEG2DQuerymem* and *eMPEG2D_Re_Querymem* are used to query the memory requirements. The application allocates memory and invokes the initialization APIs viz., *eMPEG2D_Init* and *eMPEG2D_ReInit*.

The application needs to ensure that before invoking the *eMPEG2D_Re_Querymem* the bitstream read call back function must be registered using *Mpeg2_register_func*

This function raises the call back function to extract the parameters that are required by the decoder for requesting memory, which has a dependency on the bitstream. For this callback, the application must provide bitstream data from the beginning (sequence start code) till just before the first picture start code (exclusive).

The application must ensure that memory is allocated for all the requested chunks by the decoder. If the application is unable to allocate all the chunks of memory requested, the application must not invoke the *eMPEG2D_Init* and *eMPEG2D_ReInit* APIs. (see *main* in MPEG2DecTestApp.c for sequencing of the APIs.)

DECODING PROCESS

The decoding of a frame of video is started when application calls *eMPEG2Decode*. The return enum is described in the OUTPUT section above.

2.2.8Buffer getter

Since the Direct Rendering is adopted, the decoder will ask to get a new buffer for decoding. A function which is implemented by the application (frame work) will be used to perform getting a frame buffer for decoding.

Prototype:

```
typedef void* (*bufferGetter)( void* /*pvAppContext*/);
```

Arguments:

- Application context

Return value:

- A frame buffer

2.2.9Buffer Rejecter

Since the Direct Rendering is adopted, the decoder will ask to get a new buffer for decoding.
It's possible that the gotten frame buffer may be refused by the decoder. Decoder need to inform the application (framework) that this frame is rejected.
A function which is implemented by the application (frame work) will be used to perform reject ion of a frame buffer.

Prototype:

```
typedef void (*bufferRejecter)( void* /*mem_ptr*/, void*
/*pvAppContext*/);
```

Arguments:

- A rejected frame buffer
- Application context

Return value:

- None

2.2.10 Buffer Releaser

Since the Direct Rendering is adopted, the decoder need one way to notify that the buffer acquired from bufferGetter() will not be referenced by other frames. As result, application can reuse and modify the buffer released.
A function which is implemented by the application (frame work) will be used to perform release action of a frame buffer.

Prototype:

```
typedef void (*bufferReleaser)( void* /*mem_ptr*/, void*
/*pvAppContext*/);
```

Arguments:

- A released frame buffer
- Application context

Return value:

- None

2.2.11 Buffer Manager

For clarifying the concept and simplifying the API, we group the 2 function pointer we described above into a structure named DR_BufferManager:

```
typedef struct __MPEG2D_FrameManager
{
    bufferGetter BfGetter;
    bufferRejecter BfRejector;
}MPEG2D_FrameManager;
```

2.3 Application Programmer Interface Functions

2.3.1 Call back register function.

Application needs to register a function that is being used for reading the buffer by the library using this register function. This register function would initialize the callback function pointer which is part of the library. So the library would make a call back to the application using this function pointer whenever it needs more data.

Prototype:

void *mpeg2_register_func(**sMpeg2DecObject *** mp2DecObjPtr,*
int (*cbkMPEG2DBufRead_ptr**) (int * , unsigned char **, int , void *)***
);

Arguments:

- Mp2DecObjPtr — Decoder parameter structure pointer
- cbkMPEG2DBufRead_ptr — Pointer to function of return type int, and accepting four arguments of type int *, unsigned char **, int and void respectively.

Return value:

None

2.3.2 Query Memory

This function returns the memory requirement for the decoder, which is independent of bitstream. The decoder populates the *sMpeg2DecObject.sMemInfo* structure. The application will use this structure to pre-allocate the requested memory block (chunks) by setting the pointers of *asMemBlks* in *sMpeg2DecObject.sMemInfo* structure to the required size, type & aligned memory.

Prototype:

eMpeg2DecRetType *eMPEG2DQuerymem (**sMpeg2DecObject** *psMp2Obj)*

Arguments:

- *psMp2Obj* — Pointer to *sMpeg2DecObject*

Return value:

eMpeg2DecRetType Tells whether assignment of parameters needed for memory allocation was successful or not. Enumeration is described in the above section.

Return values are -

E_MPEG2D_SUCCESS	- Function successful.
Other values	- Error

2.3.3 Initialization

Initializations required for the decoder are done in e*MPEG2D_Init*. This function must be called before the main decoder functions are invoked. The application must invoke this function only if all the chunks of memory requested by the decoder have been allocated. This API must be called after `eMPEG2DQuerymem.`

Prototype:

```
eMpeg2DecRetType eMPEG2D_Init (sMpeg2DecObject *psMp2Obj)
```

Arguments:

- *psMp2Obj* Pointer to `sMpeg2DecObject`

Return value:

eMpeg2DecRetType Tells whether decoder has been successfully initialized or not. Enumeration is described in the above section

Return values are -

E_MPEG2D_SUCCESS	-	Function successful.
Other values	-	Error

2.3.4Re-Query Memory

This function returns the memory requirement for the decoder, which is dependent of bitstream. The decoder populates the *sMpeg2DecObject.sMemInfo* structure after parsing the bitstream. The application will use this structure to pre-allocate the requested memory block (chunks) by setting the pointers of *asMemBlks* in *sMpeg2DecObject.sMemInfo* structure to the required size, type & aligned memory. The application must ensure that bitstream is available prior to calling this API. This API must be invoked after `eMPEG2D_Init.`

Prototype:

```
eMpeg2DecRetType eMPEG2D_Re_Querymem (sMpeg2DecObject
                                      *psMp2Obj)
```

Arguments:

- *psMp2Obj* Pointer to `sMpeg2DecObject`

Return value:

eMpeg2DecRetType Tells whether assignment of parameters needed for memory allocation was successful or not. Enumeration is described in the above section.

Return values are -

E_MPEG2D_SUCCESS	- Function successful.
Other values	- Error

2.3.5 Re-Initialization

Initializations of bitstream dependent elements required for the decoder are done in ***eMPEG2D_ReInit***. This API must be invoked after `eMPEG2D_Re_Querymem`. This API will reset all decoder parameters.

Prototype:

```
eMpeg2DecRetType eMPEG2D_ReInit(sMpeg2DecObject *psMp2Obj)
```

Arguments:

- *psMp2Obj* Pointer to `sMpeg2DecObject`

Return value:

eMpeg2DecRetType Tells whether decoder has been successfully initialized or not. Enumeration is described in the above section

Return values are -

E_MPEG2D_SUCCESS	-	Function successful.
Other values	-	Error

2.3.6 Decode

The main decoder function is *eMPEG2Decode*. This API decodes the MPEG2 bit stream in the input buffers to generate one frame of decoder output in every call (4:2:0 format). The address of the output frame is populated in the structure `sMpeg2DecObject`.*sDecParam.sOutputBuffer*

Prototype:

```
eMpeg2DecRetType eMPEG2Decode(sMpeg2DecObject *psMp2Obj, unsigned int
*s32decodedBytes, void *pvAppContext);
```

Arguments:

- *psMp2Obj* Pointer to `sMpeg2DecObject`
- `s32decodedBytes` Pointer to the number of bytes decoded by the library. This will be populated by the decoder for each call.
- `pvAppContext` Caller context is used to distinguish between calls from different decoding threads in a muti-threaded environments. This parameter can be ignored in a single threaded application. The test application needs to provide this information for every call to the decode function.

Return value:

eMpeg2DecRetType Tells whether decoder has been successfully initialized or not. Enumeration is described in the above section

Return values are -

E_MPEG2D_SUCCESS	-	Function successful.
E_MPEG2D_PARTIAL_DECODE	-	Partial decode of the frame.
E_MPEG2D_FRAME_READY	-	One decoded frame available to the Application
E_MPEG2D_FRAME_SKIPPED,	-	Frame is skipped
E_MPEG2D_ENDOF_BITSTREAM	-	Completion of decoding the complete

bitstream
E_MPEG2D_UNSUPPORTED - Bitstream not supported
Other values - Error

2.3.7Release

The release function is *eMPEG2DFree*. This API release related resources used by decoder object. It will be called after decoding all frames.

Prototype:

```
eMpeg2DecRetType eMPEG2DFree(sMpeg2DecObject *psMp2Obj);
```

Arguments:

- *psMp2Obj* Pointer to *sMpeg2DecObject*

Return value:

eMpeg2DecRetType Tells whether decoder has been successfully released. Enumeration is described in the above section

Return values are -
E_MPEG2D_SUCCESS - Function successful.
Other values - Error

2.3.8 Input buffer interface

cbkMPEG2DBufRead is a synchronous call used by the decoder to read the encoded data from the application. This function is called by the decoder in *eMPEG2D_Re_Querymem*, and *eMPEG2Decode* functions, when it runs out of current bit stream buffer.

This function is not part of the library and has to be supplied by the user of the decoder library. Application developer has complete control in implementing this API based on his/her system requirement. The decoder cannot continue till this function returns with required bitstream pointer and the length of the bitstream to be decoded. Refer section 2.2.5 for details on how the test application must provide the input to the decoder.

Prototype:

```
int  cbkMPEG2DBufRead ( int *s32BufLen,
                         unsigned char **pu8Buf,
                        int s32Offset,
                        void * pvAppContext );
```

Arguments:

- s32BufLen Length of the bitstream corresponding to the frame to be decoded. The test application populates this value.
- pu8Buf Pointer to the input bitstream buffer has the start address of the bitstream buffer corresponding to the frame to be decoded . The test application populates this address.
- s32Offset The current release doesnot use this parameter.

- pvAppContext Caller context, is used to distinguish between calls from different decoding threads in a muti-threaded environments. This parameter can be ignored in a single threaded application. The decoder will populate this field.

Return value:
Returns the size of the buffer copied else return -1.

2.3.9Debug Logging

The current release doesnot support Debug logging.

2.3.10 Set a Buffer Manager

As section 2.2.8, 2.2.9 and 2.2.11 mentioned, a buffer manager is needed for DR. This object should be set by the application (frame work) before decoding and after initialization.
The function specified below is used to set the buffer manager.

Prototype:
```
void MPEG2DSetBufferManager (sMpeg2DecObject *psMp2Obj,
MPEG2D_FrameManager* manager);
```

Arguments:
- *psMp2Obj* Pointer to *sMpeg2DecObject*
- manager a pointer to a frame buffer manager which is used to get and reject buffer.

Return value:
None

2.3.11 Set Callback Functions

More callback functions can be set through this interface besides getter and rejecter described in buffer manager.
This function should be called by application before decoding and after initialization.

Prototype:
```
eCallbackSetRet MPEG2DSetAdditionalCallbackFunction
 (sMpeg2DecObject *psMp2Obj, eCallbackType funcType, void* cbFunc);
```

Arguments:
- psMp2Obj a pointer which is used to indicate the decoder object.
- funcType callback function type
- cbFunc callback function pointer

Return value:
eCallbackSetRet Tells whether callback function has been set successfully
Return values are -

E_CB_SET_OK	-	set successful.
E_CB_SET_FAIL	-	set fail.

2.3.12 API Version

This is the decoder function to get the API version information

Prototype:

```
const char * MPEG2DCodecVersionInfo();
```

Arguments:

- None

Return value:

const char * The pointer to the constant char string of the version information string

2.3.13 Skip B Frames

This is the function used to enable or disable skip B frames mode. It should be called before decode frame function *eMPEG2Decode()* .

Prototype:

```
eMpeg2DecRetType MPEG2DEnableSkipBMode(sMpeg2DecObject
*psMp2Obj,int enableflag);
```

Arguments:

- sMpeg2DecObject a pointer which is used to indicate the decoder object.
- enableflag enable(value 1) or disable(value 0) skip B mode

Return value:

eMpeg2DecRetType Tells whether mode has been successfully set.

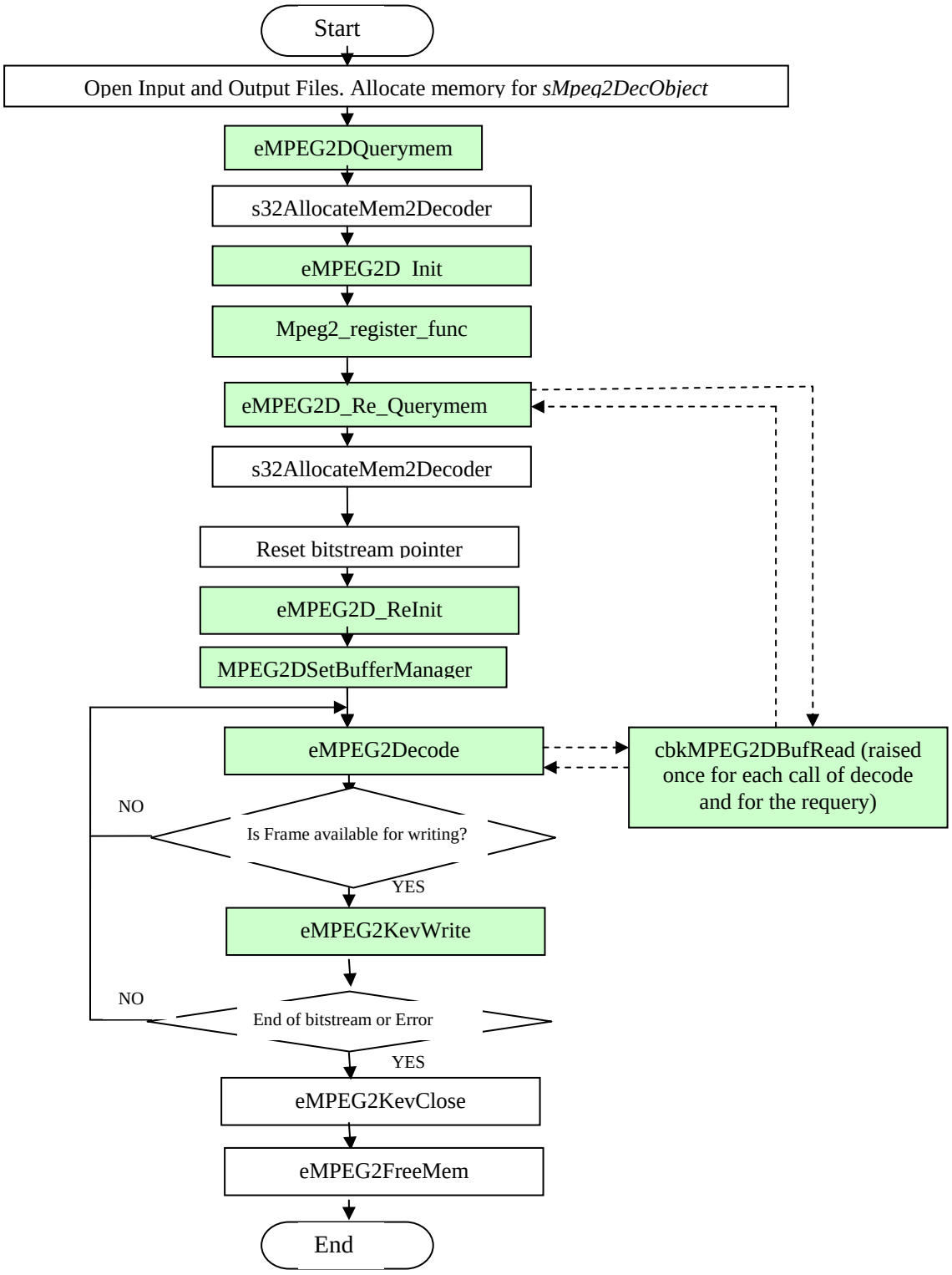
Return values are -

E_MPEG2D_SUCCESS - Function successful.

Other values - Error

Example Lib Usage

Section 2.2.5 explains the flow of the MPEG2 decoder. For details please refer the *MPEG2DecTestApp.c*. The flow graph of MPEG2 Decoder test application is illustrated below. The boxes in green color represent the APIs.


Start
Open Input and Output Files. Allocate memory for sMpeg2DecObject
eMPEG2DQuerymem
s32AllocateMem2Decoder
eMPEG2D Init
Mpeg2_register_func
eMPEG2D_Re_Querymem
s32AllocateMem2Decoder
Reset bitstream pointer
eMPEG2D_ReInit
MPEG2DSetBufferManager
eMPEG2Decode
cbkMPEG2DBufRead (raised once for each call of decode and for the requery)
NO
Is Frame available for writing?
YES
eMPEG2KevWrite
NO
End of bitstream or Error
YES
eMPEG2KevClose
eMPEG2FreeMem
End