08-6466-SIS-ZCH66
2010-01-22
3.8

Application Programmers Interface for JPEG Decoder

ABSTRACT:

Application Programmers Interface for JPEG Decoder

KEYWORDS:

Multimedia codecs, JPEG, image

APPROVED:

Wang Zening

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	03- Dec-2003	B.Venkatarao	Initial Draft
1.0	26-Dec-2003	B.Venkatarao	Review changes
1.1	02-Jan-2003	Harsha D G	Reformat content and added more details
1.2	21-Jan-2004	B.Venkatarao	Added appendix and changes in the interface of jpegd_get_new_data() function.
2.0	27-Jan-2004	Harsha D G	Added the scale down factor in the API
2.1	28-Jan-2004	B.Venkatarao	Error codes added in the Appendix.
2.2	28-Jan-2004	Harsha D G	Added descriptions to certain TBD items
2.3	23-Mar-2004	B.Venkatarao	Changes to use same functions for all output formats. Changed some of the Decoder structure names and definitions. Added enum for output format.
2.4	05-Jul-2004	B.Venkatarao	Updated with latest changes
2.5	16-Nov-2004	B.Venkatarao	Updated with new PCS requirements
2.6	29-Nov-2004	B.Venkatarao	Review comments incorporated
2.7	11-Jan-2005	B.Venkatarao	Prefixing data types. One section added to describe debug logs. Updated with latest changes.
2.8	08-Sep-2005	Shankar Narayan P.S.	Section added to describe implementation of call back
3.0	06-Feb-2006	Lauren Post	Using new format
3.1	28-March-2006	Sriram Shankar	Document review
3.2	1-Aug-2008	Wang Zening	Add API version information
3.3	12-Nov-2008	Eagle Zhou	Add BGR output format
3.4	02-Mar-2009	Eagle Zhou	Add API for decoding the whole frame
3.5	25-May-2009	Eagle Zhou	Add VPU decoding
3.6	19-Oct-2009	Eagle Zhou	Refine
3.7	12-Jan-2009	Eagle Zhou	Modify downscale, add orientation
3.8	22-Jan-2010	Eagle Zhou	Add comment for no wait mode

Table of Contents

1 Introduction

1.1 Purpose

This document gives the application programmer's interface for JPEG Decoder.

1.2 Scope

This document does not give the details of implementation of the JPEG Decoder. It only explains the functions and variables exposed in the API.

1.3 Audience Description

The reader is expected to have basic understanding of Image processing and JPEG Sequential and Progressive decoding.

1.4 References

1.4.1 Standards

- DIS 10918-1 and draft DIS 10918-2
- "JPEG Still Image Data Compression Standard" by William B. Penne baker and Joan L. Mitchell published by Van No strand Reinhold, 1993, ISBN 0-442-01272-1. 638 pages, price US$59.95. This book includes the complete text of the ISO JPEG standards (DIS 10918-1 and draft DIS 10918-2).

1.4.2 General References

- Wallace, Gregory K. "The JPEG Still Picture Compression Standard", Communications of the ACM, April 1991 (vol. 34 no. 4), pp. 30-44.

1.4.3 Freescale Multimedia References

- JPEG Decoder Application Programming Interface – jpeg_dec_api.doc
- JPEG Decoder Interface header – jpeg_dec_interface.h
- JPEG Decoder Test Application – jpeg_dec_app.c

1.5 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
API	Application Programming Interface

ARM	Advanced RISC Machine
Data Unit	JPEG proposal defines a data unit as a sample in predictive codecs and a [8x8] block in case of DCT based codecs
DCT	Discrete Cosine Transform
FSL	Freescale
IDCT	Inverse Discrete Cosine Transform
IJG	Independent JPEG Group
JPEG	Joint Photographic Experts Group
MCU	Minimum Coded unit. JPEG proposal defines an MCU as the smallest group of interleaved data units
RVDS	ARM RealView Development Suite
TBD	To Be Determined
UNIX	Linux PC x/86 C-reference binaries
VPU	Video Process Unit

1.6 Document Location

docs/jpeg_dec

2 Type and Structure Definitions

This section describes the list of the types and structures used in the JPEG decoder API.

2.1 Type Definitions

This subsection describes the common data types used in the JPEG decoder API functions.

2.1.1 JPEGD_UINT8

```
typedef unsigned char JPEGD_UINT8;
```

2.1.2 JPEGD_INT8

```
typedef char JPEGD_INT8;
```

2.1.3 JPEGD_UINT16

```
typedef unsigned short JPEGD_UINT16;
```

2.1.4 JPEGD_INT16

```
typedef short JPEGD_INT16;
```

2.1.5 JPEGD_UINT32

```
typedef unsigned long JPEGD_UINT32;
```

2.1.6 JPEGD_INT32

```
typedef long  JPEGD_INT32;
```

2.1.7 JPEGD_UINT64

```
typedef unsigned long long  JPEGD_UINT64;
```

2.1.8 JPEGD_INT64

```
typedef long long JPEGD_INT64;
```

2.1.9 JPEGD_DCT_METHOD

/* DCT/IDCT algorithm options. */

```
typedef enum
{
```

```
    JPEGD_IDCT_SLOW,     /* Slow but accurate integer algorithm */
    JPEGD_IDCT_FAST,     /* Faster, less accurate integer method */
    JPEGD_IDCT_FLOAT,     /* Floating-point: Reserved, not supported*/
    JPEGD_IDCT_FIRST = JPEGD_IDCT_SLOW,
    JPEGD_IDCT_LAST = JPEGD_IDCT_FLOAT
} JPEGD_DCT_METHOD;
```

2.1.10 JPEGD_OUTPUT_FORMAT

/* Output formats */

```
typedef enum
{
    JPEGD_OFMT_ENC_IMAGE_FMT,        /* Same as Encoded image format */
    JPEGD_OFMT_RGB_565,
    JPEGD_OFMT_RGB_888,              /* Default */
    JPEGD_OFMT_BGR_565,
    JPEGD_OFMT_BGR_888,
    JPEGD_OFMT_FIRST = JPEGD_OFMT_ENC_IMAGE_FMT,
    JPEGD_OFMT_LAST = JPEGD_OFMT_BGR_888
} JPEGD_OUTPUT_FORMAT;
```

2.1.11 JPEGD_THUMBNAIL_TYPE

```
typedef enum
{
    JPEGD_NO_THUMBNAIL = 0,
    JPEGD_THUMBNAIL_JPEG,
    JPEGD_THUMBNAIL_UNCOMPRESSED,
    JPEGD_THUMBNAIL_CORRUPT,
    JPEGD_THUMBNAIL_UNKNOWN,
} JPEGD_THUMBNAIL_TYPE;
```

2.1.12 JPEGD_RET_TYPE

/* Error types */

```
typedef enum
{
    /* Successfull return values */
    JPEGD_ERR_NO_ERROR = 0,
    /* Warnings:
     *      The application can check the warnings and can continue decoding.
     */
    JPEGD_ERR_WARNINGS_START = 11,
    JPEGD_ERR_SUSPENDED = JPEGD_ERR_WARNINGS_START,
    JPEGD_ERR_WARNINGS_END,
    /* Recoverable errors
     *      These are the application errors. The application can
     *      correct the error and call the decoder again from the beginning
     */
    JPEGD_ERR_REC_ERRORS_START = 61,
    JPEGD_ERR_INVALID_DCT_METHOD =    JPEGD_ERR_REC_ERRORS_START,
    JPEGD_ERR_INVALID_OUTPUT_FORMAT,
    JPEGD_ERR_INVALID_OUT_BUFFER_PTR,
    JPEGD_ERR_INVALID_OUT_STRIDE_WIDTH,
    JPEGD_ERR_MEM_NOT_INITIALIZED,
    JPEGD_ERR_MEM_NOT_ALIGNED,
    JPEGD_ERR_OUT_BUFFER_NOT_ALIGNED,
    JPEGD_ERR_REC_ERRORS_END,
    /* Fatal errors: These are the codec errors which can not be recovered  */
    JPEGD_ERR_FATAL_ERRORS_START = 111,
```

```
    JPEGD_ERR_ARITH_NOTIMPL = JPEGD_ERR_FATAL_ERRORS_START,
    JPEGD_ERR_BAD_COMPONENT_ID,
    JPEGD_ERR_BAD_DCTSIZE,
    JPEGD_ERR_BAD_HUFF_TABLE,
    JPEGD_ERR_BAD_J_COLORSPACE,
    JPEGD_ERR_BAD_LENGTH,
    JPEGD_ERR_BAD_LIB_VERSION,
    JPEGD_ERR_BAD_MCU_SIZE,
    JPEGD_ERR_BAD_PRECISION,
    JPEGD_ERR_BAD_PROGRESSION,
    JPEGD_ERR_BAD_SAMPLING,
    JPEGD_ERR_BAD_STATE,
    JPEGD_ERR_BAD_STRUCT_SIZE,
    JPEGD_ERR_CCIR601_NOTIMPL,
    JPEGD_ERR_COMPONENT_COUNT,
    JPEGD_ERR_CONVERSION_NOTIMPL,
    JPEGD_ERR_DAC_INDEX,
    JPEGD_ERR_DAC_VALUE,
    JPEGD_ERR_DHT_INDEX,
    JPEGD_ERR_DQT_INDEX,
    JPEGD_ERR_EMPTY_IMAGE,
    JPEGD_ERR_EOI_EXPECTED,
    JPEGD_ERR_FRACT_SAMPLE_NOTIMPL,
    JPEGD_ERR_IMAGE_TOO_BIG,
    JPEGD_ERR_NOTIMPL,
    JPEGD_ERR_NOT_COMPILED,
    JPEGD_ERR_NO_HUFF_TABLE,
    JPEGD_ERR_NO_IMAGE,
    JPEGD_ERR_NO_QUANT_TABLE,
    JPEGD_ERR_NO_SOI,
    JPEGD_ERR_OUT_OF_MEMORY,
    JPEGD_ERR_SOF_DUPLICATE,
    JPEGD_ERR_SOF_NO_SOS,
    JPEGD_ERR_SOF_UNSUPPORTED,
    JPEGD_ERR_SOI_DUPLICATE,
    JPEGD_ERR_SOS_NO_SOF,
    JPEGD_ERR_TOO_LITTLE_DATA,
    JPEGD_ERR_UNKNOWN_MARKER,
    JPEGD_ERR_WIDTH_OVERFLOW,
    JPEGD_ERR_BAD_THUMBNAIL_DATA,       /* Thumbnail related errors */
    JPEGD_ERR_BAD_INPUT_PARAM_EXIF,
    JPEGD_ERR_BAD_INPUT_PARAM_MODE,
    JPEGD_ERR_UNCOMPRESSED_THUMBNAIL,
    JPEGD_ERR_THUMBNAIL_OFFSET_NOT_FOUND,
    JPEGD_ERR_BAD_THUMBNAIL_TYPE,
    JPEGD_ERR_FATAL_ERRORS_END,
    /* Vpu errors: These are the vpu errors   */
    JPEGD_ERR_VPU_START=311,
    JPEGD_ERR_VPU_SETTING_ERROR=JPEGD_ERR_VPU_START,
    JPEGD_ERR_VPU_UNSUPPORTED_FMT,
    JPEGD_ERR_VPU_INVALID_MEMORY,
    JPEGD_ERR_VPU_INIT_FAILURE,
    JPEGD_ERR_VPU_OPEN_FAILURE,
    JPEGD_ERR_VPU_FILL_BUFFER_FAILURE,
    JPEGD_ERR_VPU_GET_INFO_FAILURE,
    JPEGD_ERR_VPU_REGISTER_FRAME_FAILURE,
    JPEGD_ERR_VPU_DECODE_FAILURE,
    JPEGD_ERR_VPU_GET_OUTPUT_FAILURE,
    JPEGD_ERR_VPU_END,
} JPEGD_RET_TYPE;
```

Description:

Any of the decoder API functions can return an error code if the decoder encountered an error during execution. And the application should take appropriate according to the returned error code. The error codes have been classified into successful returns, application errors and codec errors.

Some error types (including all application errors) are described here.

ERROR CODE	DESCRIPTION
JPEGD_ERR_NO_ERROR	No error
JPEGD_ERR_SUSPENDED	Obsolete
JPEGD_ERR_INVALID_DCT_METHOD	Invalid DCT method configured by the application
JPEGD_ERR_INVALID_OUTPUT_FORMAT	Invalid output format configured by the application
JPEGD_ERR_INVALID_OUT_BUFFER_PTR	Output buffer pointers are NULL
JPEGD_ERR_INVALID_OUT_STRIDE_WIDTH	Invalid output stride width configured by the application
JPEGD_ERR_MEM_NOT_INITIALIZED	Memory chunk pointers are NULL
JPEGD_ERR_MEM_NOT_ALIGNED	Memory chunk pointers are not aligned to the requested alignment size.
JPEGD_ERR_OUT_BUFFER_NOT_ALIGNED	Output buffer pointer not aligned to 2 byte boundary for RGB-565 or BGR-565 output format

2.1.13 Memory Type

```
enum
{
    JPEGD_STATIC_MEMORY = 0,     // normal virtual memory for software
    JPEGD_SCRATCH_MEMORY,        // reserved
    JPEGD_PHY_SUCCESSIVE_MEMORY, // physical successive memory for hardware
};
```

2.2 Structure Definitions

This subsection describes the structure definitions used in JPEG decoder API.

2.2.1 JPEGD_Decoder_Object

This structure contains all the information about the decoder instance.

```
typedef struct
{
    JPEGD_Mem_Alloc_Info        mem_info;
    JPEGD_Decoder_Params        dec_param;
    JPEGD_Decoder_Info          dec_info;
    void                        * cinfo;
    JPEGD_exif_info             exif_info;
    JPEGD_jfif_info             jfif_info;
    JPEGD_UINT8                (*jpegd_get_new_data_fun) (JPEGD_UINT8 **, JPEGD_UINT32 *
,JPEGD_UINT32 , JPEGD_UINT8 ,void *);
} JPEGD_Decoder_Object;
```

Description:

mem_info

The member structure contains the codec memory requirements. The structure may be filled memory requirement information by the library after call API function jpegd_query_mem().

dec_param

The member structure contains the decoding parameters. The application may configure JPEG decoder library with variables in this structure.

dec_info

The member structure contains the decoding information decoder library reported to application decoder library decoding information and is read-only for the application decoder library decoding information and is read-only for the application. The application should only read the variables in this structure, never to write it.

cinfo

The reserved pointer is used by decoder library internally. The application should ignore it and never to write it.

exif_info

The member structure contains the EXIF information structure. The decoder library presents the EXIF information if EXIF contains in the input jpeg file/stream.

jfif_info

The member structure contains the JFIF information structure. The decoder library presents the JFIF information if JFIF contains in the input jpeg file/stream.

*(*jpegd_get_new_data_fun)*

The function pointer variable contains the callback function, which decoder library uses it to ask for more data.

2.2.3 JPEGD_Mem_Alloc_Info

```
typedef struct
{
    JPEGD_INT32                 num_reqs;
    JPEGD_Mem_Alloc_Info_Sub    mem_info_sub[JPEGD_MAX_NUM_MEM_REQS];
} JPEGD_Mem_Alloc_Info;
```

Description:

num_reqs

The number of the available items store in the *JPEGD_Mem_Alloc_Info_Sub* array. This information is returned from the API *jpegd_query_dec_mem* call to clarify the member allocation requirements.

mem_info_sub

The array of the *JPEGD_Mem_Alloc_Info_Sub* to stores the decoder's memory allocation requirements. Every item in the array describes the details of the memory requirements.

2.2.4 JPEGD_Mem_Alloc_Info_Sub

```
typedef struct
{
    JPEGD_INT32   align;
    JPEGD_INT32   size;
    JPEGD_INT32   mem_type_speed;
    JPEGD_INT32   mem_type_usage;
    JPEGD_INT32   priority;
    void *        ptr;
    void *        phy_ptr;
} JPEGD_Mem_Alloc_Info_Sub;
```

Description:

align

The byte alignment requirements of the memory block start address. The application should allocate the memory with this start address alignment.

size

The size of the required memory block

mem_type_speed

Reserved

mem_type_usage

The usage type of the decoding using this memory block.
JPEGD_STATIC_MEMORY: normal virtual memory
JPEGD_PHY_SUCCESSIVE_MEMORY: physical successive memory for hardware

priority

Reserved

ptr

The pointer of the required memory block's virtual start address.

phy_ptr

The pointer of the required memory block's physical start address. The application should get the physical address if the *mem_type_usage* equals to physical and successive memory.

2.2.4 JPEGD_Decoder_Params

These parameters should be set by the application, before calling any decoder functions.

```
typedef struct
{
    JPEGD_DCT_METHOD       dct_method;
    JPEGD_OUTPUT_FORMAT    output_format;
    JPEGD_UINT8     decoding_mode;
    JPEGD_UINT8     exif_info_needed;
    JPEGD_UINT8     scale;
    JPEGD_UINT8     vpu_enable;
    JPEGD_UINT32    vpu_bitstream_buf_size;
    JPEGD_UINT32    vpu_fill_size;
    JPEGD_UINT32    vpu_wait_time; /*decoder's waiting time before vpu need more data,
unit: millisecond*/
} JPEGD_Decoder_Params;
```

Description:

dct_format

The DCT/IDCT algorithm option is configured by the application to software decoder. The default value is JPEGD_IDCT_FAST, fast IDCT.

output_format

The output format can be one of the values in the JPEGD_OUTPUT_FORMAT for software decoder, but only JPEGD_OFMT_ENC_IMAGE_FMT for vpu enabled mode.

decoding_mode

The decoding mode can be configured as primary image or thumbnail image decoding.

exif_info_needed

The flag of whether output EXIF information is only available for EXIF format file decoding.

scale

The downscale value.
1: no scale
2: 1/2 x 1/2 downscale
4: 1/4 x 1/4 downscale

8: 1/8 x 1/8 downscale
Other: forbidden

vpu_enable

The VPU decoding mode is implemented or not. 1: enable; 0: disable.

vpu_bitstream_buf_size

The size of VPU input bit stream buffer is only valid for VPU decoding mode. Just set it to 0 to configure the decoder to use the default value.

vpu_fill_size

The unit size of data which feed to VPU every time. Just set it to 0 to configure the decoder to use the default value.

vpu_wait_time

Interval times every checking of vpu status.
Decoder will enter wait status (switch to other thread) before checking whether vpu has finished consuming current data. Unit is millisecond.

0:	Use the default wait time set by decoder internal when user doesn't care it.
0xFFFFFFFF:	Notify decoder not to wait (polling mode).
Other value:	Other wait time set by user

2.2.5 JPEGD_Decoder_Info

This structure defines the decoder library decoding information and is read-only for the application.

```
typedef struct
{
    JPEGD_UINT8       mode;
    JPEGD_UINT8       h_samp_max;
    JPEGD_UINT8       v_samp_max;
    JPEGD_UINT8       num_components;
    JPEGD_UINT16      original_image_width;
    JPEGD_UINT16      original_image_height;
    JPEGD_UINT16      actual_output_width;
    JPEGD_UINT16      actual_output_height;
    JPEGD_UINT16      output_scanline;
    JPEGD_INT32       max_lines;
    JPEGD_INT32       num_lines;
    JPEGD_THUMBNAIL_TYPE   thumbnail_type;
    JPEGD_UINT8       file_format;
    JPEGD_UINT32      min_size_exif;
    /* Information of the components present in the JPEG file */
    JPEGD_Component_Info   comp_info[JPEGD_MAX_NUM_COMPS];
} JPEGD_Decoder_Info;
```

Description:

mode

The encoded mode: Sequential or Progressive mode. VPU enabled mode is only support Sequential mode.

h_samp_max and v_samp_max

The Maximum horizontal/vertical sampling factor for software decoder to resize decoding.

num_components

The number of the components in the JPEG file. It indicates the valid number of component information array.

original_image_width and original_image_height

The original image width/height presents in the JPEG bit stream.

actual_image_width and actual_image_height

The actual image width/height presents for output image. The value of width/height is based on the configured parameter dec_param->desired_output_height. And they can be different from (always <=) the value of the dec_param->desired_output_height.

output_scanline

The number of lines decoded so far. It is valid for software decoder mode.

max_lines

The maximum number of lines decoder can emit when *jpegd_decode_mcu_row()* is called. It is valid for software decoder mode.

num_lines

The number of lines returned by calling *jpegd_decode_mcu_row()*. It is valid for software decoder mode.

thumbnail_type

The thumbnail image type would be report by decoder.

file_format

The file format of the input JPEG bit stream: JFIF file or EXIF file.

min_size_exif

The minimum size of data callback function feed to decoder in its first call.

comp_info

The array contains all of the components' information.

2.2.6 JPEGD_Component_Info

```
typedef struct
{
    JPEGD_UINT8       h_samp;
    JPEGD_UINT8       v_samp;
    JPEGD_UINT16      actual_output_width;
    JPEGD_UINT16      actual_output_height;
    /* Number of lines decoded so far of a comp */
    JPEGD_UINT16      output_scanline;
    /* Maximum number of lines decoder can emit for a component */
    JPEGD_INT32       max_lines;
    /* Number of lines returned for a comp by jpegd_decode_mcu_row() */
    JPEGD_INT32       num_lines;
} JPEGD_Component_Info;
```

Description:

h_samp and v_sam

The horizontal / vertical sampling factor of the component.

actual_image_width and actual_image_height

The actual image width/height presents for the component.

output_scanline

The number of lines decoded so far. It is valid for software decoder mode.

max_lines

The maximum number of lines decoder can emit when *jpegd_decode_mcu_row()* is called. It is valid for software decoder mode.

num_lines

The number of lines returned by calling *jpegd_decode_mcu_row()*. It is valid for software decoder mode.

2.2.7 JPEGD_tag

```
typedef struct
{
    JPEGD_UINT32    count; /* count is size of the tag in bytes */
    void*           ptr;
} JPEGD_tag;
```

2.2.8 JPEGD_jfif_info

```
typedef struct
{
    JPEGD_UINT8     jfif_major_version;
    JPEGD_UINT8     jfif_minor_version;
    JPEGD_UINT8     density_unit; // 0 = no unit, 1 = dots per inch, 2 = dots per cm
    JPEGD_UINT16    Xdensity;
    JPEGD_UINT16    Ydensity;
    //JPEGD_UINT8   compressed_thumbnail;
    JPEGD_THUMBNAIL_TYPE   thumbnail_type;
} JPEGD_jfif_info;
```

Description:

jfif_version

JFIF version, 4-byte info

resolution_unit

Unit for measuring X-resolution and Y-resolution;
0 – aspect ratio, 1 – inches, 2 - centimeters

x_density

Number of horizontal pixels per resolution unit

y_density

Number of vertical pixels per resolution unit

thumbnail_type

1 - Compressed thumbnail is present
0 - Compressed thumbnail is not present

2.2.9 JPEGD_exif_info

```
typedef struct
{
    JPEGD_UINT8     endianness;
    /* flags to indicate the presence of IFDs */
    JPEGD_UINT8     IFD0_flag;
    JPEGD_UINT8     ExifIFD_flag;
    JPEGD_UINT8     IFD1_flag;
    JPEGD_UINT8     InteropIFD_flag;
    JPEGD_UINT8     GpsIFD_flag;
    JPEGD_THUMBNAIL_TYPE   thumbnail_type;
    /* IFD structs to hold the tag info */
    JPEGD_IFD0_appinfo     ifd0_info;
    JPEGD_exifIFD_appinfo  exififd_info;
    JPEGD_IFD1_appinfo     ifd1_info;
    /* currently doesn't support GPS IFD and Interop IFD */
} JPEGD_exif_info;
```

2.2.10 JPEGD_IFD0_appinfo

```
typedef struct
{
    JPEGD_UINT32    x_resolution[2];
    JPEGD_UINT32    y_resolution[2];
    JPEGD_UINT16    resolution_unit;
    JPEGD_UINT16    ycbcr_positioning;
    JPEGD_UINT16    orientation;
} JPEGD_IFD0_appinfo;
```

Description:

x_resolution

Number of pixels per resolution unit in the horizontal direction

y_resolution

Number of pixels per resolution unit in the vertical direction

resolution_unit

Unit for measuring X-resolution and Y-resolution

2 – inches; 3- centimeters; Other- reserved

ycbcr_positioning

The position of chrominance components in relation to the luminance component

1 = centered 2 = co-sited

orientation

Rotation information for the original image

0 - No related information in stream

1 -The 0th row is at the visual top of the image; the 0th column is the visual left-hand side

2 -The 0th row is at the visual top of the image; the 0th column is the visual right-hand side

3 -The 0th row is at the visual bottom of the image; the 0th column is the visual right-hand side

4 -The 0th row is at the visual bottom of the image; the 0th column is the visual left-hand side

5 -The 0th row is the visual left-hand side of the image; the 0th column is the visual top

6 -The 0th row is the visual right-hand side of the image; the 0th column is the visual top

7 -The 0th row is the visual right-hand side of the image; the 0th column is the visual bottom

8 -The 0th row is the visual left-hand side of the image; the 0th column is the visual bottom

Other –Reserved

2.2.11 JPEGD_exifIFD_appinfo

```
typedef struct
{
    JPEGD_UINT8     exif_version[4];
    JPEGD_UINT8     componentsconfiguration[4];
    JPEGD_UINT8     flashpix_version[4];
    JPEGD_UINT16    colorspace;
    JPEGD_UINT16    pixel_x_dimension;
    JPEGD_UINT16    pixel_y_dimension;
} JPEGD_exifIFD_appinfo;
```

Description:

exif_version

EXIF version, 4-byte ASCII.

components_configuration

Component configuration specific to compressed data

4,5,6,0 - RGB uncompressed data
1,2,3,0 - Other cases (Y-Cb-Cr)
Other - Reserved

flashpix_version

Flashpix version, 4-byte ASCII (Example 00.01).

colorspace

Color space
1 - sRGB
FFFF - Uncalibrated (can be treated as sRGB when it is converted to Flashpix)

pixel_x_dimension

The valid image width, when the compressed file is recorded

pixel_y_dimension

The valid image height, when the compressed file is recorded

2.2.12 JPEGD_IFD1_appinfo

```
typedef struct
{
    JPEGD_UINT32    x_resolution[2];
    JPEGD_UINT32    y_resolution[2];
    JPEGD_UINT16    resolution_unit;
    JPEGD_UINT16    compression; /* = 6 for compressed thumbnail, others invalid */
    JPEGD_UINT32    jpeg_interchange_format; /* offset to thumbnail image */
    JPEGD_UINT32    jpeg_interchange_format_length; /* size of thumbnail image */
    JPEGD_UINT16    orientation
} JPEGD_IFD1_appinfo;
```

Description:

x_resolution

Number of pixels per resolution unit in the horizontal direction, for thumbnail image

y_resolution

Number of pixels per resolution unit in the vertical direction, for thumbnail image

resolution_unit

Unit for measuring X-resolution and Y-resolution of the thumbnail image
2 – inches; 3- centimeters; Other- reserved

compression

Compression scheme for thumbnail image
1 – Uncompressed, 6 – Compressed, Other – reserved

jpeg_interchange_format

Offset to thumbnail image

jpeg_interchange_format_length

Size of thumbnail image

orientation

Rotation information for the original image
The same with definition in structure JPEGD_IFD0_appinfo

3 API Definitions

This section provides the general descriptions of the JPEG decoder API functions for application to control the decoder library.

3.1.1 jpegd_register_jpegd_get_new_data

Prototype:

```
JPEGD_RET_TYPE jpegd_register_jpegd_get_new_data (
            JPEGD_UINT8 (* func)(JPEGD_UINT8 **
                                 , JPEGD_UINT32 *
                                 , JPEGD_UINT32
                                 , JPEGD_UINT8
                                 , void *),
            JPEGD_Decoder_Object * obj_ptr);
```

Arguments:

- * func: a function pointer to register the callback function to decoder library
- obj_dec: decoder object structure pointer

Return value:

- JPEGD_ERR_NO_ERROR - Successful.
- Other codes - Error

Description:
The decoder library works in “pull” mode, it may call the callback function to ask for more input data. The application calls this API function to register the callback function to decoder.

3.1.2 Callback function -- jpegd_get_new_data

Prototype:

```
JPEGD_UINT8 jpegd_get_new_data (
                    JPEGD_UINT8 ** ppBuf,
                    JPEGD_UINT32  * pLen,
                    JPEGD_UINT32 mcu_offset,
                    JPEGD_UINT8 begin_flag,
                    void *obj_ptr);
```

Arguments:

- ppBuf: a pointer to the buffer pointer of input data.
- pLen: a pointer to buffer size variable.
- mcu_offset: offset to the current MCU from the end of the current buffer. It is no used in current version.
- begin_flag: flag to indicate whether decoder needs bit stream from the beginning of the file.
- obj_ptr: decoder object pointer.

Return value:

- JPEGD_SUCCESS - Buffer allocation successful
- JPEGD_END_OF_FILE - End of file

Description:
This API function defines the interface of callback function. The application defines the callback function instance and registers it to decoder through the API function *jpegd_register_jpegd_get_new_data*.

Remark:
In the callback function implements, it needs implement the different data feed method by the different value of the input argument *ppBuf*.
If this buffer pointer is NULL, application need fill up a local data buffer and assign this buffer's start address to this buffer pointer. Also the pointer of variable *pLen* would be assigned with the available length of the buffer.
If this buffer pointer is not NULL, application need fill up data to the memory pointer by this buffer pointer with the desired length from the value of the variable *pLen*.

3.1.3 jpegd_get_file_info

Prototype:

```
JPEGD_RET_TYPE jpegd_get_file_info (
            JPEGD_Decoder_Object * dec_obj,
            JPEGD_UINT8 * file_format,
            JPEGD_THUMBNAIL_TYPE * thumbnail_type,
            JPEGD_UINT32 * min_size_exif);
```

Arguments:

- dec_obj: Decoder object structure pointer
- file_format: a pointer to variable for file format information output
- thumbnail_type: a pointer to variable for thumbnail image type information output
- min_size_exif: a pointer to variable for minimum size of the first buffer to feed by the application

Return value:

- JPEGD_ERR_NO_ERROR - Successful.
- Other codes - Error

Description:
This API function is used for application to get the JPEG file information, which is needed for decoding, such as JPEG file format, thumbnail image format and so on,
The output value of the file format and thumbnail format variables would like below:
file_format:

- JPEGD_FILE_IS_JFIF - it is a JFIF file
- JPEGD_FILE_IS_EXIF - it is an EXIF file

thumbnail_type:

- JPEGD_NO_THUMBNAIL - There is no thumbnail present
- JPEGD_THUMBNAIL_JPEG - JPEG compressed thumbnail is present
- JPEGD_THUMBNAIL_UNCOMPRESSED - Uncompressed thumbnail is present
- JPEGD_THUMBNAIL_CORRUPT, - Thumbnail is corrupted
- JPEGD_THUMBNAIL_UNKNOWN - Unknown thumbnail is present

3.1.4 jpegd_query_dec_mem

Prototype:

```
JPEGD_RET_TYPE jpegd_query_dec_mem (JPEGD_Decoder_Object * dec_obj);
```

Arguments:

- dec_obj: decoder object structure pointer.

Return value:

- JPEGD_ERR_NO_ERROR - Memory query successful
- Other codes - Error

Description:
This API function is used for application to query the decoder's memory requirements for JPEG file decoding. The memory requirements result in the structure *JPEGD_Mem_Alloc_Info* of the decoder object structure.

3.1.5 jpegd_decoder_init

JPEGD_RET_TYPE jpegd_decoder_init (JPEGD_Decoder_Object *dec_obj);

Prototype:

```
JPEGD_RET_TYPE jpegd_decoder_init (JPEGD_Decoder_Object * dec_obj);
```

Arguments:

dec_obj: decoder object structure pointer.

Return value:

JPEGD_ERR_NO_ERROR - Initialization successful.
Other codes - Initialization Error

Description:
This API function is used for application to initialize the decoder library to make it ready for decoding operation.

3.1.6 jpegd_decode_frame

Prototype:

```
JPEGD_RET_TYPE jpegd_decode_frame (
                    JPEGD_Decoder_Object *dec_obj,
                    JPEGD_UINT8 ** buffer_ptrs,
                    JPEGD_UINT16 *out_stride_width);
```

Arguments:

- dec_obj: decoder object structure pointer
- buffer_ptrs: a pointer to the array of pointers to the output buffer of all components
- out_stride_width: a pointer to variable of the stride length of each component buffer, (row width)

Return value:

- JPEGD_ERR_NO_ERROR - indicates decoding was successful.
- Others - indicates error

Description:
This API function is used for application to start the decoding operation. This function decodes the JPEG bit stream to generate the whole output image as the requested format. It implements the same action as API function *jpegd_decode_mcu_row* except the number of output MCU rows.

For RGB format outputs, the decoder library will updates the value of the variable *dec_info->output_scanline* for the scan line started decoding, and also *dec_info->num_lines* for the number of lines.
For YUV format outputs, the decoder library would also update the two parameters *dec_info->comp_info[i]->output_scanline and dec_info->comp_info[i]->num_lines* of every component.

Before calling this function, application should make sure that the stride length of every component should be equal to or greater than the *actual_output_width* of each component.

3.1.7 jpegd_decode_mcu_row

Prototype:
```
JPEGD_RET_TYPE      jpegd_decode_mcu_row  (
                            JPEGD_Decoder_Object *dec_obj,
                            JPEGD_UINT8 ** buffer_ptrs,
                            JPEGD_UINT16 *out_stride_width);
```
Arguments:
- dec_obj : decoder object structure pointer
- buffer_ptrs : a pointer to the array of pointers to the output buffer of all components
- out_stride_width : a pointer to variable of the stride length of each component buffer, (row width)

Return value:
- JPEGD_ERR_NO_ERROR - indicates decoding was successful.

Others - indicates error

Description:
This API function is used for application to start the decoding operation. It decodes the JPEG bit stream to generate 1 MCU row of the output image as the requested format.
For RGB format outputs, the decoder library will updates the value of the variable *dec_info->output_scanline* for the scan line started decoding, and also *dec_info->num_lines* for the number of lines.
For YUV format outputs, the decoder library would also update the two parameters *dec_info->comp_info[i]->output_scanline and dec_info->comp_info[i]->num_lines* of every component.

Before calling this function, application should make sure that the stride length of every component should be equal to or greater than the actual_output_width of each component.

This function is not supported in VPU decoding mode.

3.1.8 jpegd_CodecVersionInfo

Prototype:

```
const char * jpegd_CodecVersionInfo ();
```

Arguments:

None

Return value:

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

Description:

This API function is used for application to get the decoder version information.

4 API Usage Description

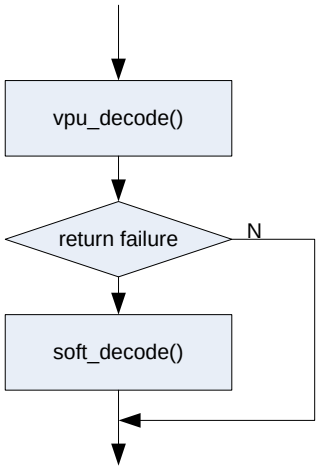
This is the most important section of this document. This section describes the steps followed by the application to call the API functions of the JPEG decoder library.

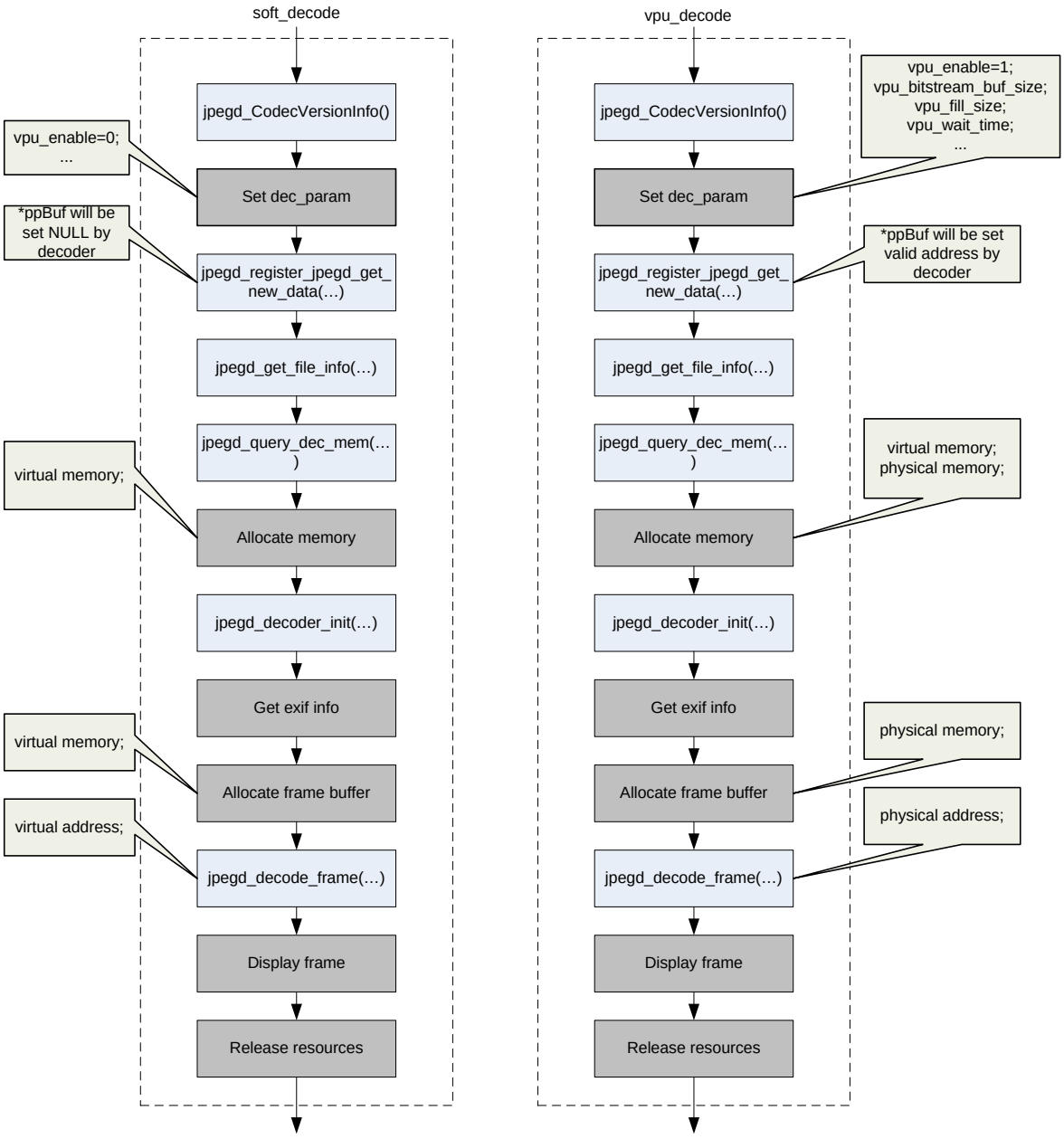
4.1 API Usage Summary

This part summaries the work flow for using JPEG decoding.

The current JPEG library includes both VPU and software decoder. The VPU decoder implements the decoding with hardware, its decoding speed is very fast, but as the VPU HW can only support JPEG Baseline mode; software decoder can support both baseline and progressive mode, so the application can switch to software decoder if the VPU HW fails to decode.

The figure below shows the decoding strategy for application.





4.2 API Operation Flow

Step 1: Create Decoder object instance

In the first step, the application allocates memory for the JPEG decoder object instance firstly. The object structure *JPEGD_Decoder_Object* contains the decoding parameters, decoding information, picture information and memory information structures, which would be filled in later steps.

```
typedef struct
{
    JPEGD_Mem_Alloc_Info        mem_info;
    JPEGD_Decoder_Params        dec_param;
    JPEGD_Decoder_Info          dec_info;
    void                          * cinfo;
    JPEGD_exif_info             exif_info;
    JPEGD_jfif_info             jfif_info;
    JPEGD_UINT8                 (*jpegd_get_new_data_fun) (JPEGD_UINT8 **, JPEGD_UINT32
* ,JPEGD_UINT32 , JPEGD_UINT8 ,void *);
} JPEGD_Decoder_Object;
```

Step 2: Register callback function to JPEG library for new data getting

In this step, the application registers the callback function that provides new data for the decoder. For example, the function *jpegd_get_new_data* is registered to decoder through the decoder API -- *jpegd_register_jpegd_get_new_data*.
This API function may store the callback function pointer into the decoder object structure and in future the decoder library calls this function to ask for new data for decoding.

The implement of the callback function instance should follow the definitions: if decoder library call back with null value buffer address, the callback function may provide a local buffer to decoder; if decoder library call back with valid buffer address, the callback function needs to fill up the buffer with the desired length.

Step 3: Get the JPEG file information

In this step, application may get the JPEG file information, such as whether the file contains EXIF information or nor, whether the file contains thumbnail image or not. The API function *jpegd_get_file_info* should be called to parse the JPEG bit stream and find the required decoding information.

In this API function call, it may require application to get more input bit stream data through callback function *jpegd_get_new_data*.
The API function *jpegd_get_file_info* returns three parameters: file format type, thumbnail format type and minimum size of the EXIF information. And that information would also be filled in the decoder object's dec_info structure by this function.

If input JPEG file contains the EXIF information, the return parameter "min_size_exif" means the minimum size of the input data the callback function should provide in the first time when call the API function *jpegd_query_dec_mem* and *jpegd_decoder_init*. Refer to step7 for more details.

Important Note: In the first time the callback function is called, the application should provide JPEG bit stream from the beginning of the JPEG stream when API function *jpegd_get_file_info*, *jpegd_query_dec_mem* and *jpegd_decoder_init* implemented.

Step 4: Fill up Decoder Parameters Structure

In this step, the application configures the JPEG decoder library by filling up the member structure *JPEGD_Decoder_Params* in the decoder object structure.
The decoding information need to configure as: whether use VPU or software decoding mode by member variable *vpu_enable*; whether need output EXIF information by the member variable *exif_info_needed*; what the desired pixel output format by *output_format*; primary/thumbnail image decoding by *decoding_mode*; downscale value by variables scale.

```
/*
 * Decoder parameters. These parameters should be set by the
 *  application, before calling any decoder functions.
 */
typedef struct
{
    JPEGD_DCT_METHOD       dct_method;    /* default is JPEGD_IDCT_FAST, fast IDCT */
    JPEGD_OUTPUT_FORMAT    output_format;     /* default is JPEGD_OFMT_ENC_IMAGE_FMT */

    JPEGD_UINT8     decoding_mode;         /* JPEGD_PRIMARY, JPEGD_THUMBNAIL */
    JPEGD_UINT8     exif_info_needed;     /* Used only if file_format is exif */
    JPEGD_UINT8     scale;                 /* downscale set */
    JPEGD_UINT8     vpu_enable;              /*1: enable; 0: disable*/
    JPEGD_UINT32    vpu_bitstream_buf_size; /*buffer size used for raw data*/
    JPEGD_UINT32    vpu_fill_size; /*unit size of data which decoder feed to vpu every
time*/
    JPEGD_UINT32    vpu_wait_time; /*decoder's waiting time before vpu need more data,
unit: millisecond*/
} JPEGD_Decoder_Params;
```

Step 5: Query Decoder Memory Requirements

The JPEG decoder library doesn't allocate any memory internally. So in this step the application would allocate memory for decoder and transmit the memory block start address to decoder library. Firstly the application calls decoder API function *jpegd_query_dec_mem* to get memory needs of decoder library. And then allocate the memory in the next step.
In this API function, it parses JPEG bit stream from the file or stream beginning to get JPEG information and calculate the memory requirements based on JPEG information, and return memory needs through the memory information structure array.
Important Note:
Before call this API, the application should reset the read pointer to the beginning of the JPEG file or stream, so the callback function may get the JPEG data from the beginning of the file or stream.

Step 6: Data Memory allocations

In this step the application allocates the memory followed the decoder requirements. As all of the memory requirements stores in the structure *JPEGD_Mem_Alloc_Info* returned by the memory query API call, Firstly application need get the number of memory buffer needed, nd then allocate the memory one-by-one with right size, alignment, whether it need physical continuous, and so on. The allocated buffers' start addresses would be filled back to the memory pointer member field *ptr* in structures *JPEGD_Mem_Alloc_Info_Sub*.

If VPU is enabled for decoding, the memory buffer's physical address should also be filled to member field *phy_ptr*.

```
/* JPEGD_Mem_Alloc_Info and JPEGD_Mem_Alloc_Info_Sub are the memory allocation
 *  structures filled by the decoder in jpegd_query_dec_mem() function.
 *  Then memory pointers (ptr) will be initialized by the application
 */
typedef struct
{
    JPEGD_INT32   align;          /* Alignment of memory in bytes */
    JPEGD_INT32   size;           /* Size in bytes */
    JPEGD_INT32   mem_type_speed; /* Memory type Fast or Slow: reserved */
    JPEGD_INT32   mem_type_usage; /* Memory type: normal or physical*/
    JPEGD_INT32   priority;       /* Memory priority: reserved */
    void          * ptr;          /* Pointer to the memory */
    void          * phy_ptr;      /* Pointer to the physical memory */
} JPEGD_Mem_Alloc_Info_Sub;

typedef struct
{
    JPEGD_INT32                 num_reqs;
    JPEGD_Mem_Alloc_Info_Sub    mem_info_sub[JPEGD_MAX_NUM_MEM_REQS];
} JPEGD_Mem_Alloc_Info;
```

Step 8: Streaming Input

The application has to allocate the memory needed for the input buffer. The decoder, whenever it needs the JPEG bit-stream, shall call the function *jpegd_get_new_data*.

jpegd_get_new_data should be implemented by the application. The application might have different techniques to implement this function. Sample code is given in released package.

This function serves a dual purpose. Firstly, the decoder can decide the buffer address through filling **ppBuf* appropriately. The application can then fill data into this address. Secondly, the buffer address can also be decided by application. The application can identify which instance of the decoder has called this function through judge whether *ppBuf is equal to NULL. If the application wants to decoder quick quit, it may set valid data length to 0 byte.

If *begin_flag* is set to 1, application has to give the bitstream from the beginning of the JPEG file. Also if it is an EXIF file and *begin_flag* is set to 1, it has to provide the bitstream of the size greater than or equal to "min_size_exif" that is given by the function *jpegd_get_file_info*. The decoder sets *begin_flag* to 1, when this function is called first time from the functions *jpegd_query_dec_mem and jpeg_decoder_init.*

Important Note:

- The application has to pass the buffers from the beginning of the JPEG file, when "begin_flag" is set to 1.
- In case of EXIF files, the decoder expects the first bit stream buffer should be of size greater than the size *min_size_exif* returned by *jpegd_get_file_info.*

Step 9: Decoder Initialization routine

The API function *jpegd_decoder_init* is used for application to initialize the decoder library after all of the decoding preparation work is done.
Firstly, the application should reset the read pointer to the beginning of the JPEG file or stream, so the callback function may get the JPEG data from the beginning of the file or stream.
Then, application calls the API function *jpegd_decoder_init* to do the decoder library initialization. The decoder library return the decoding information in the member structure *JPEGD_Decoder_Info* in the decoder object structure.
After decoder is initialized successfully, the application can use the actual output image size actual_output_width and actual_output_height to allocate the memory for the output buffer.

If decoding parameter *dec_params->exif_info_needed* is set to 1, the decoder initialization procedure would also fill the EXIF information structure *dec_obj->exif_info*.

```
typedef struct
{
    /* Relative horizontal sampling factor of a component */
    JPEGD_UINT8       h_samp;
    /* Relative vertical sampling factor of a component */
    JPEGD_UINT8       v_samp;

    /* Output width and height of a component */
    JPEGD_UINT16      actual_output_width;
    JPEGD_UINT16      actual_output_height;
    /* Number of lines decoded so far of a comp */
    JPEGD_UINT16      output_scanline;

    /* Maximum number of lines decoder can emit for a component */
    JPEGD_INT32       max_lines;

    /* Number of lines returned for a comp by jpegd_decode_mcu_row() */
    JPEGD_INT32       num_lines;
} JPEGD_Component_Info;

/* Following structure is the decoder info and is read-only for the application */
typedef struct
{
    JPEGD_UINT8       mode;       /* Sequential or Progressive */
    JPEGD_UINT8       h_samp_max; /* Maximum Horizontal sampling factor */
    JPEGD_UINT8       v_samp_max; /* Maximum vertical sampling factor */
    JPEGD_UINT8       num_components; /* Number of components in JPEG file */

    JPEGD_UINT16      original_image_width; /* Input Image width, as present in
                                               the JPEG bitstream */
    JPEGD_UINT16      original_image_height; /* Input Image height, as present in
                                               the JPEG bitstream */
    JPEGD_UINT16      actual_output_width;  /*final output width*/
    JPEGD_UINT16      actual_output_height; /*final output height*/

    /* For YUV outputs, following three parameters output_scanline, max_lines,
     * and num_lines are the parameters of the maximum component present
     * in the JPEG file. */
    /* Number of lines decoded so far */
    JPEGD_UINT16      output_scanline;
    /* Maximum number of lines decoder can emit when jpegd_decode_mcu_row
     * is called
     */
    JPEGD_INT32       max_lines;

    /* Number of lines returned by jpegd_decode_mcu_row() */
```

```
    JPEGD_INT32       num_lines;

    JPEGD_THUMBNAIL_TYPE thumbnail_type;
    JPEGD_UINT8       file_format;
    JPEGD_UINT32      min_size_exif;

    /* Information of the components present in the JPEG file */
    JPEGD_Component_Info     comp_info[JPEGD_MAX_NUM_COMPS];
} JPEGD_Decoder_Info;
```

Important Note:
The above decoder information is initialized in *jpegd_decoder_init()* and will remain constant throughout the decoding except for the following parameters. The following parameters are initialized in *jpegd_decoder_init()* and will be updated in the decoder API *jpegd_decode_mcu_row().*

```
dec_info->output_scanline
dec_info->num_lines
dec_info->comp_info[i]->output_scanline
dec_info->comp_info[i]->num_lines
```

Also note that the component information structure *dec_info->comp_info* will be filled by the decoder only for YUV outputs.

Step 10: Memory allocation for output buffer

In this step, the application allocates memory for the output buffers to hold YUV or RGB Data. The decoder does not request this memory when *jpegd_query_dec_mem* is called.

If the application wants to allocate buffers which need to hold only one MCU row of data, the size of the buffers can be calculated from the following parameters,

For RGB outputs

dec_info->actual_output_width and dec_info->max_lines

RGB565 -> pixel size = 2 bytes
RGB888 -> pixel size = 3 bytes

Size = dec_info->actual_output_width * dec_info->max_lines * pixel_size;

For YUV outputs

dec_info->comp_info[i]-> actual_output_width and
dec_info->comp_info[i]-> max_lines
pixel_size -> pixel size = 1 byte (for each component)

Size of component #i = dec_info->comp_info[i]-> actual_output_width *
dec_info->comp_info[i]-> max_lines * pixel_size;

If the application wants to allocate buffers to hold the whole image, the size of the buffers can be calculated from the following parameters.

For RGB outputs

dec_info->actual_output_width and dec_info->actual_output_height

RGB565 -> pixel size = 2 bytes
RGB888 -> pixel size = 3 bytes
Size = dec_info->actual_output_width * dec_info->actual_output_height * pixel_size;

For YUV outputs
dec_info->comp_info[i]-> actual_output_width and
dec_info->comp_info[i]-> actual_output_height
pixel_size -> pixel_size = 1 byte (for each component)
Size of component #i = dec_info->comp_info[i]-> actual_output_width * dec_info->comp_info[i]-> actual_output_height * pixel_size;

Important Note:

To get maximum performance in speed, output buffer pointers should be aligned to 8-byte boundary. For RGB-565 outputs, make sure that output buffer pointer is aligned at least to 2-byte boundary.

Step 11: Start the decoding routine (whole image mode)

In this step, the application calls the main decoding API function *jpegd_decode_frame* to generate the whole image as the requested format.

Before calling this function, application should make sure that the stride length of every component should be equal to or greater than the *actual_output_width* of each component.

For RGB format outputs, the decoder library will updates the value of the variable *dec_info->output_scanline* for the scan line started decoding, and also *dec_info->num_lines* for the number of lines.
For YUV format outputs, the decoder library would also update the two parameters *dec_info->comp_info[i]->output_scanline and dec_info->comp_info[i]->num_lines* of every component.

If decode configuration vpu_enable is 1 and when decoding function returns error code as JPEGD_ERR_VPU_xxx, it means VPU HW fails to decoder the JPEG. So the application would switch to software method by set vpu_enable to 0, and restart from step 1.

If decode configuration vpu_enable is 0 and when decoding function returns error code as JPEGD_ERR_xxx, it means software decoder also can not support this file or stream decoding, it need jump step 13 to exit.

Step 12: Start the decoding routine (MCU row mode)

The main decoder function is *jpegd_decode_mcu_row*. This function decodes the JPEG bit stream to generate the output image in the requested format. During the process of decoding, the function *jpegd_get_new_data* gets called whenever the decoder runs out of input. And the calling application needs to feed fresh data to decoder.

If the bit stream has errors, the decoder handles these errors internally.
The application is responsible for passing the appropriate values in the *buffer_ptrs* array. This array shall contain the pointers to the output buffers.

This function is not supported in VPU decoding mode.

When *jpegd_decode_mcu_row* is called the decoder is expected to decode 1 MCU row. The decoder shall also fill *dec_info->output_scanline*, the scanline it has started decoding. It shall also fill the number of lines it has decoded in *dec_info->num_lines*. For YUV outputs, it shall also fill the above two parameters of each component *dec_info->comp_info[i]->output_scanline and dec_info->comp_info[i]->num_lines.*

Important: In the case of RGB outputs, one buffer pointer and one output stride length is passed. In the case of YUV outputs, buffer pointer and output stride length for each component is passed. Before calling this function, user should make sure that the stride length of each component should always be greater than or equal to the actual_output_width of each component.

Step 13: Free memory

The application releases the memory that it allocated to JPEG Decoder if it no longer needs the decoder instance.

5 Example calling Routine

Please get details from file jpeg_dec_app.c in release package.