# JPEG Decoder Library Developer's Guide

**DEVIMG-001-F**
**October 2005**

**Analog Devices Inc.**

**www.analog.com**

# Table of Contents

**1. Introduction ..... 6**
1.1. Scope ..... 6
1.2. Target platform ..... 6
1.3. Organisation of this Guide ..... 6
1.4. Version Information ..... 6
1.5. System Requirements ..... 6
1.6. Acronyms ..... 7
1.7. References ..... 7
1.8. Additional Information ..... 7

**2. Specifications ..... 8**
2.1. Sequential Process ..... 8
2.2. Extended Sequential DCT Process (Huffman coding) ..... 8
2.3. Progressive DCT Process (Huffman coding) ..... 8
2.4. MIPS Performance ..... 8
2.5. Memory Requirements ..... 9

**3. Usage ..... 10**
3.1. Tools ..... 10
3.2. Software Integration ..... 10
3.3. User-Defined Data Access Module ..... 10
3.4. Programming Example ..... 11

**4. Programmer's Reference ..... 12**
4.1. JPEG Common library ..... 12
4.1.1. JPEG_Param_CONFIG ..... 12
4.1.2. JPEG_Param_STATUS ..... 13
4.1.3. JPEG_Param_INIT ..... 13
4.2. JPEG Decoder library ..... 14
4.2.1. JPEG_Decoder_NEW ..... 14
4.2.2. JPEG_Decoder_DELETE ..... 14
4.2.3. JPEG_ProcessHeader ..... 14
4.2.4. JPEG_DecodeImage ..... 15
4.2.5. JPEG_DecodeSequentialImage ..... 15
4.2.6. JPEG_DecodeProgressiveImage ..... 16

# List of Tables

Table 1: Tool revisions....6
Table 2: Blackfin JPEG Decoder Memory Requirements....9
Table 3: JPEG Configuration Parameters....13

# List of Figures

Figure 1 Cycles/pixel vs. Bit Rate....9
Figure 2: Data Access Module....10

# 1. Introduction

This document describes how to use the JPEG Decoder Library.

## 1.1. Scope

This guide is intended for C-language developers proficient in the application of the Analog Devices Blackfin family of DSPs and related software tools. It is assumed that the reader is familiar with all aspects of these DSPs.

## 1.2. Target platform

The JPEG Decoder Library was designed for the Analog Devices Blackfin DSPs.

## 1.3. Organisation of this Guide

Information supplied in this guide is organised in the following way. Section 2 provides the specifications of the implemented JPEG Decoder. In section 3, notes on how to use and integrate the library are given. Finally section 4 describes the high level software structure of the library.

## 1.4. Version Information

Note that this document refers to the following software versions:
***JPEG Decoder:*** ADI Version 3.0.4

Any information that accompanies the software libraries (in the form of 'readme' files, etc.) should be assumed to supersede the information contained in this document.

## 1.5. System Requirements

The JPEG Decoder library module was created and tested with versions of tools listed in Table 1. To ensure the maximum level of compatibility, please use these versions of the tools (listed tools correspond to the September 2005 update for VDSP++ 4.0).

| Tool Name | Tool Description | Tool Version |
|---|---|---|
| ccblkfn | C/C++ compiler | 7.1.3.6 |
| easmblkfn | BlackFin assembler | 2.6.9.8 |
| elfar | ELF Librarian/Archive Utility | 4.5.1.3 |
| linker | Linker | 3.5.8.1 |

**Table 1: Tool revisions**

## 1.6. Acronyms

The following acronyms are used in this document.

| | |
|---|---|
| AC | Non-zero frequency (components, tables, etc) |
| API | Application Programming Interface |
| APPn | One of the JPEG markers. The APPn markers are reserved for application segments. |
| DC | Zero frequency (components, tables, etc) |
| DCT | Discrete Cosine Transform |
| DMA | Direct Memory Access |
| DSP | Digital Signal Processor, Digital Signal Processing |
| JPEG | Joint Photographic Experts Group |
| JFIF | JPEG File Interchange Format |
| PDF | Adobe Portable Document Format |
| RGB | The Red, Green, Blue colour model |
| MCU | Minimum Coded Unit |
| MIPS | Million Instructions per Second |
| RAM | Random-Access Memory |
| SRAM | Static RAM |
| YUV | The YUV colour model (also known as YCbCr). |

## 1.7. References

[1] ITU-T, “Information Technology- Digital Compression and Coding of Continuous-Tone Still Images- Requirements and Guidelines”, Recommendation T.81 (09/92), September 1992.

[2] Analog Devices Inc., “JPEG/M-JPEG for the ADSP-BF5xx Blackfin Processor Product Highlight”, SSHMKT-046, October 2005.

[3] Analog Devices Inc., “JPEG System IO Buffer Modules Supplementary Developer's Guide”, DEVIM1-001, Nov 2003.

## 1.8. Additional Information

For more information on the latest ADI processors, silicon errata, code examples, development tools, system services and devices drivers, technical support and any other additional information, please visit our website at www.analog.com/processors.

# 2. Specifications

The JPEG decoder library implements still image decompression according to the standard [1] for images encoded using one of 4 possible encoding processes with restrictions as described below.

The input image file must satisfy the JFIF specification when the input image data is in YUV format, or in Adobe file format if the input image data is in RGB format. In each case the bit stream is a JPEG bit stream with the first using the APP0 marker to indicate that JFIF information is present, and the other using both the APP0 marker and the APP14 marker to indicate that the bitstream contains RGB image data. Note that the Adobe file format is different from Adobe's Portable Document Format (PDF). All other APP marker segments and comment segments are ignored by the JPEG decoder library.

The image must begin with a Start of Image (SOI) marker, and the bitstream between the SOI marker and the mandatory End of Image (EOI) marker must be in accordance with the JPEG standard [1]. Any information following the EOI marker is ignored.

All images are output as 8 bit samples within each component (in a non interleaved, planar fashion).

## 2.1. Sequential Process

Only JPEG bit streams with 1 or 3 components are decoded. In the case of 3 components, component sampling factors must correspond to one of 3 cases (with the notation below following the conventions in [1]):

a) $H_1=H_2=H_3=1$ & $V_1=V_2=V_3=1$ (corresponding to the video frame format YUV444);

b) $H_1=2$, $H_2=H_3=1$ & $V_1=V_2=V_3=1$ (corresponding to the video frame format YUV422);

c) $H_1=2$, $H_2=H_3=1$ & $V_1=2$, $V_2=V_3=1$ (corresponding to the video frame format YUV420)

## 2.2. Extended Sequential DCT Process (Huffman coding)

Only JPEG bit streams with 1 or 3 components are decoded. Component sampling factors are restricted in the same way as for the Baseline process.

## 2.3. Progressive DCT Process (Huffman coding)

Only JPEG bit streams with 1 or 3 components are decoded. Component sampling factors are restricted in the same way as for the Baseline process.

## 2.4. MIPS Performance

There are many factors that affect performance, some of which are data dependent. For the example applications, the system configuration consists of running the code from L1 memory, allocating the heap in external SRAM (L3 memory), and using both L1 data banks as cache. In this configuration, a plot of the cycles required per pixel vs. the compression ratio (bits/pixel) is shown in Figure 1 for BF533 processors and four different test images in YUV420 format.

The cycle counts are shown in Figure 1 graphed against the bit rate (bits per pixel) of the JPEG files.

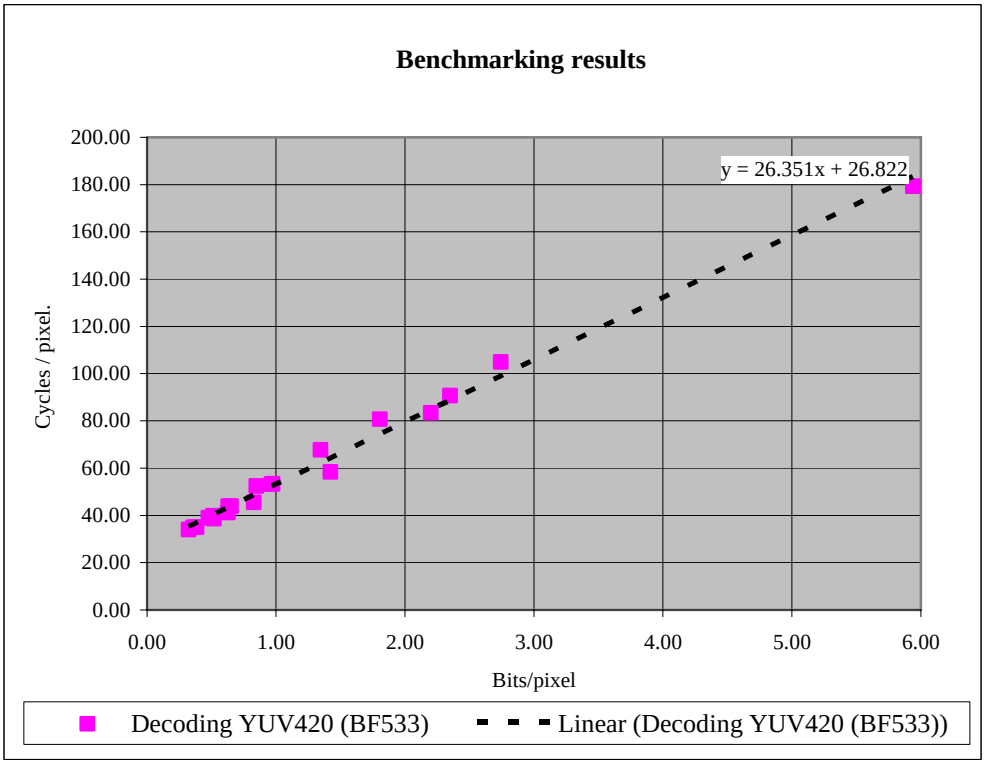


**Figure 1 Cycles/pixel vs. Bit Rate**

## 2.5. Memory Requirements

The memory requirements for the JPEG Decoder depend on which mode is used, as shown in Table 2. If only sequential or progressive encoded images are to be decoded, the appropriate format-specific decode function **JPEG_DecodeSequentialImage** or **JPEG_DecodeProgressiveImage** can be used exclusively. This requires the project to be linked with the –e option to strip any unused functions from the executable.

The memory requirement for sequential mode is independent of the image size. The progressive mode, however, requires memory for the entire JPEG bitstream output to be allocated on the heap and is therefore image size dependent.

All of the numbers in Table 2 exclude both the memory used by the user-defined modules and any standard C library functions that may be required at link time (since these may be shared by the application). See the latest datasheet [3] for up to date specifications. The numbers also exclude any external memory that needs to be allocated to store the actual image data and the JPEG bit stream.

| **Encoding Mode** | **Program Memory JPEGDEC_P0 (KB)** | **Program Memory JPEGDEC_P1 (KB)** | **Program Memory JPEG_P0 (KB)** | **Data Memory JPEGDEC_D0 (KB)** | **Heap Maximum (KB)** |
|---|---|---|---|---|---|
| All encoding modes | 11.1 | 6.3 | 0.2 | 0.2 | 11.18 + 6N* |
| Progressive mode only | 8.6 | 6.3 | 0.2 | 0.2 | 11.18 + 6N* |
| Sequential mode only | 10.6 | - | 0.2 | 0.2 | 7.8 |

**Table 2: Blackfin JPEG Decoder Memory Requirements**

* progressive mode requires approximately 6 bytes/pixel from the heap (for RGB format images, the worst case).

The library is partitioned into the following sections:

- JPEGDEC_P0: all JPEG code used in sequential mode, plus all JPEG code common to both modes
- JPEGDEC_P1: JPEG code used only in progressive mode
- JPEG_P0: JPEG code that is common to both the JPEG encoder and decoder libraries
- JPEGDEC_D0: all static data allocated by the library

Ensure that memory location 0x0000 0000 is not mapped to a code or data section. The User-Defined Data Access Module controls the placement of data memory on the heap.

# 3. Usage

## 3.1. Tools

The VisualDSP++ integrated development environment for Blackfin is used to build the library and example applications on a Windows PC. The required version of this tool is stated in the **readme.txt** file in the top-level directory of the package.

## 3.2. Software Integration

To use the JPEG decoder library in an application, the library file **jpeg_dec_lib_533_XXXX.dlb** needs to be included in the application makefile (where **XXXX** identifies the library version). If the VisualDSP graphical interface is used, it is simply a matter of adding this file to the project under Project Options→Link→Additional Options (by typing in the full filename of the library), specifying the path that points to this library under Project Options→Link→Search Directories, and specifying the path that points to the library header files under Project Options→Compile→Preprocessor→Additional Include Directories. Alternatively, one can also drag and drop the library by selecting the library file from an open Windows Explorer window to the Project Window in VisualDSP++.

Include the header file **jpeg_api_decoder.h** in source files where JPEG Library functions are called.

## 3.3. User-Defined Data Access Module

The user defined Data Access Module provides memory access functions for the JPEG Decoder Library, and allows the user to customise the input/output mechanism of the JPEG Decoder Library for specific systems, allowing optimal memory transfer methods. The usage of the Data Access Module is illustrated in Figure 2. How the data is transferred between the JPEG Decoder Library and the Input/Output Peripherals is entirely dependant on the system. The only requirement the user has to satisfy are the C interfaces of a small number of user defined functions which the library uses to read input data and write output data.

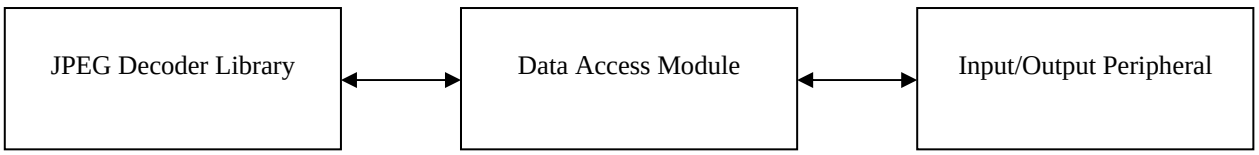


**Figure 2: Data Access Module**

The Data Access Module consists of a few groups of functions: the Raw image data buffer (McuBuffer) functions, the Bit stream data buffer (BitsBuffer) functions, and the Memory allocation (MemAlloc) functions.

For a complete description of this module and how the user can define their own data access module, see the JPEG System IO Buffer Modules Supplementary Developer's Guide [3]. Examples are provided where the input/output peripheral is simply the external SDRAM (sufficient memory is assumed to be available to store the entire frame of input image data and the resulting bit-stream).

## 3.4. Programming Example

To decode a JPEG bit stream, the library functions must be called in the following order:

i. Start by initialising a JPEG parameter structure with **JPEG_Param_INIT()**

ii. Call **JPEG_Param_CONFIG()** to set the input pointer. Image parameters may also be set if desired, with the same function. If omitted, these may be read directly from the image header.

iii. Call **JPEG_Decoder_NEW()** to create an instance of the JPEG decoder to handle this particular JPEG image, This function can be called once for multiple images if the images have the same JPEG parameters (for example, height and width, encoding mode and format).

iv. *[Optional]*
Call **JPEG_ProcessHeader()** to decode the JPEG header. Image parameters shall be read from the JPEG image header, and returned to the user. Note that if the user does not call this function, then the library will automatically call this function for each image.

v. Call **JPEG_Param_CONFIG()** to set the output pointer. The user must only set the JPEG_POINTER_OUTPUT configuration parameter at this point. It is the user's responsibility to ensure that the output buffer is able to handle the total amount of image data decoded from the bitstream.

vi. Call **JPEG_DecodeImage()** (or **JPEG_DecodeSequentialImage()** etc) to decode the JPEG image,

vii. Call **JPEG_Decoder_DELETE()** to delete this instance of the JPEG decoder.

The following code example uses these functions to decode a sample JPEG file. Note that the sample code provided in the user-defined module JPEG_BitsBuffer.c requires the application to load the whole bitstream into a buffer in external memory and to pass the buffer location and length via a data structure of type **tIMG_BufferInfo** to the library using the **JPEG_Param_CONFIG()** function.

```
JPEG_Param_INIT(&lImageParam);

/* get length of input bitstream (from length of input file) & allocate a buffer for it */
fseek(inPtr, 0, SEEK_END);
lStreamBufferInfo.Length  = ftell (inPtr);
fseek(inPtr, 0, SEEK_SET);
lStreamBufferInfo.Pointer = (uint8 *)malloc( lStreamBufferInfo.Length );

/* ... Create input and output buffers here ... */
JPEG_Param_CONFIG(&lImageParam, JPEG_POINTER_INPUT, (int)(&lStreamBufferInfo));
JPEG_Param_CONFIG(&lImageParam, JPEG_POINTER_OUTPUT, (int)lFrameBuffer);

/* Creating the JPEG decoder instance */
lJpegDec = JPEG_Decoder_NEW(&lImageParam);
if (lJpegDec == 0)
{
        fprintf(TestReportFile, "Cannot create the JPEG decoder\n");
        exit (EXITERROR);
}

/*****************************/
/* DECODE EACH FRAME         */
/*****************************/
lNumFrames = 0;
isInputDataAvailable = E_TRUE;
while (E_TRUE == isInputDataAvailable) {

        /* Decode image */
        lResult = JPEG_DecodeImage(lJpegDec, &lImageParam);

        /* ... Write the decoded image frame here ... */
```

```
}; // For each frame

/* Free and destroy used resources */
JPEG_Encoder_DELETE(lJpegEnc);
```

Subsequent images with common parameters (identical width, height, encoding mode and image data format) may be decoded by repeated calls to **JPEG_DecodeImage.** An error occurs if images with dissimilar parameters are decoded. The decoder instance must be deleted (**JPEG_Decoder_DELETE()**), and then recreated (**JPEG_Decoder_NEW()**) in the correct format for an image.

# 4. Programmer's Reference

This section describes each of the functions in the JPEG Decoder library.

The function arguments are described in the Parameters sections as [IN] or [OUT]. This defines whether the function will read (IN pointers) or write to the memory (OUT pointers).

All error codes defined in the following sections have numerical values assigned to them in the header file IMG_common.h. The user should always use the macro defined names, rather than actual numerical values within their application code.

Image parameters may be supplied to the decoder, or they may be read from the JPEG header.

## 4.1. JPEG Common library

### 4.1.1. JPEG_Param_CONFIG

***Function Name:***
int JPEG_Param_CONFIG (tJpegParam *param, JPEG_config item, unsigned int value)
***Parameters:***
param [IN]: Pointer to JPEG parameters data structure to be configured
item [IN]: Item to be configured (see Table 3).
value [IN]: 32 bit value to be stored (see Table 3).

| Item parameter: one of | Value |
|---|---|
| **JPEG_FRAME_WIDTH** | Maximum dimension value in horizontal direction among all the components of the image, including any padding. |
| **JPEG_FRAME_HEIGHT** | Maximum dimension value in vertical direction among all the components of the image, including any padding. |
| **JPEG_IMAGE_WIDTH** | Actual width dimension of the image, excludes padding. |
| **JPEG_IMAGE_HEIGHT** | Actual height dimension of the image, excludes padding. |
| **JPEG_IMAGEFORMAT** | Denotes the image format. Set to 1 for YUV4:2:0, 2 for YUV 4:2:2, 3 for YUV 4:4:4, 4 for YUV 4:0:0 and 5 for RGB 24. For an RGB image format (interleave format 5) the JPEG library decodes from Adobe file format (but not PDF). For all other cases, it decodes in JFIF file format. |
| **JPEG_QUALITYFACTOR** | This parameter is used to control the quality of JPEG Encoding. It can take values from 1 (worst quality) to 100 (best). This parameter is not used by the JPEG Decoder. |
| **JPEG_ENCODINGMODE** | This parameter is used to select between sequential and progressive modes. Pass SEQUENTIAL for sequential (baseline) mode, PROGRESSIVE for progressive mode. |

| Item parameter: one of | Value |
|---|---|
| **JPEG_THRESHOLD** | This is adaptive thresholding for the JPEG Encoder, and is selectable only in the Sequential mode. Enabling this results in better compression without much loss in perceptual quality. Enabling this mode requires a slightly greater number of cycles. Set to 1 to enable; 0 to disable. This setting is ignored in Progressive mode. This parameter is not used by the JPEG Decoder. |
| **JPEG_POINTER_INPUT** | Pointer to the input (MCU buffer) configuration object, this object is owned by the application. |
| **JPEG_POINTER_OUTPUT** | Pointer to the output (bitstream buffer) configuration object, this object is owned by the application. |

**Table 3: JPEG Configuration Parameters**

***Return Value:***
Returns E_SUCCESS if configuration was successful, or:

E_INVALIDPARAMETERITEM if an incorrect parameter is specified, or

E_FAILURE otherwise.

***Description:***
This function configures the JPEG parameters for a new encoder or decoder.

### 4.1.2. JPEG_Param_STATUS

***Function Name:***
int JPEG_Param_STATUS (tJpegParam *param, JPEG_config item, unsigned int *value)
***Parameters:***
param [IN]: Pointer to JPEG parameters data structure to be read
item [IN]: Item requested (see Table 3 for the possible values).
value [OUT]: 32 bit number representing the value of the requested parameter (see Table 3).
***Return Value:***
Returns E_SUCCESS if no errors encountered, or:

E_INVALIDPARAMETERITEM if an incorrect parameter is specified, or

E_FAILURE otherwise.

***Description:***
This function returns the value of the requested JPEG parameter from the parameter's data structure.

### 4.1.3. JPEG_Param_INIT

***Function Name:***
int JPEG_Param_INIT (tJpegParam *param)
***Parameters:***
param [IN]: Pointer to JPEG parameters data structure to be initialized
***Return Value:***
Returns E_SUCCESS if initialization was successful, or:

E_CANNOTINITIALIZE if the parameters could not be initialized

E_FAILURE otherwise

***Description:***
This function initializes the JPEG parameters structure to a known set of values.

## 4.2. JPEG Decoder library

### 4.2.1. JPEG_Decoder_NEW

***Function Name:***
tJpegDecoder *JPEG_Decoder_NEW (tJpegParam* param)
***Parameters:***
param [IN]: JPEG decoder parameters
***Return Value:***
Handle to the created JPEG decoder object. NULL upon failure.
***Description:***
This function creates an instance of JPEG decoder and returns the handle to the JPEG decoder instance. The create function performs the memory allocation based on the parameter values passed through the parameter structure. If memory allocation fails it returns NULL.

### 4.2.2. JPEG_Decoder_DELETE

***Function Name:***
void JPEG_Decoder_DELETE (tJpegDecoder* handle)
***Parameters:***
handle [IN]: Handle to JPEG decoder object.
***Return Value:***
None
***Description:***
Deletes the JPEG decoder object.

### 4.2.3. JPEG_ProcessHeader

***Function Name:***
int JPEG_ProcessHeader(tJpegDecoderStruct *handle, tJpegParam *param)
***Parameters:***
handle [IN]: Handle to JPEG decoder instance.
param [IN/OUT]: pointer to a parameter structure
***Return Value:***
E_SUCCESS if successful, or:

| | |
|---|---|
| E_CANNOTPARSE | if the JPEG structure could not be parsed |
| E_OUT_OF_MEMORY | if memory could not be allocated (heap is not large enough) |
| E_EVALUATIONLIMITREACHED | if an evaluation package limit is reached |
| E_FORMATMISMATCH | if the incorrect JPEG parameters were specified using the JPEG_Param_Config function, or if the previous JPEG bitstream decoded using the same JPEG Decoder Handle is encoded with different parameters to the current JPEG bitstream |
| E_FAILURE | otherwise. |

***Description:***
Reads and interprets the JPEG header, and returns the image parameters.

### 4.2.4. JPEG_DecodeImage

***Function Name:***

int JPEG_DecodeImage (tJpegDecoder* handle, tJpegParam *param)

***Parameters:***

handle [IN]: Handle to JPEG decoder instance.
param [IN/OUT]: pointer to a parameter structure

***Value:***

E_SUCCESS if successful, or:

| | |
|---|---|
| E_CANNOTPARSE | if the JPEG structure could not be parsed |
| E_CANNOTDECODESEQUENTIAL | if a sequentially encoded image could not be decoded |
| E_CANNOTDECODEPROGRESSIVE | if a progressively encoded image could not be decoded |
| E_MODENOTSUPPORTED | if the image is in an unsupported format |
| E_OUT_OF_MEMORY | if memory could not be allocated (heap is not large enough) |
| E_EVALUATIONLIMITREACHED | if an evaluation package limit is reached |
| E_FORMATMISMATCH | if the incorrect JPEG parameters were specified using the JPEG_Param_Config function, or if the previous JPEG bitstream decoded using the same JPEG Decoder Handle is encoded with different parameters to the current JPEG bitstream |
| E_FAILURE | otherwise. |

***Description:***

Decodes a JPEG frame, with no restrictions on the available modes.

This function may be called repeatedly for consecutive images. There is no need to call JPEG_ProcessHeader() after the first image. Consecutive images must have the same image parameters (for example, height and width, encoding mode and format) as those used in the initial call to JPEG_ProcessHeader().

### 4.2.5. JPEG_DecodeSequentialImage

***Function Name:***

int JPEG_DecodeSequentialImage (tJpegDecoder* handle, tJpegParam *param)

***Parameters:***

handle [IN]: Handle to JPEG decoder instance.

param [IN/OUT]: pointer to a parameter structure

***Return Value:***

E_SUCCESS on success, or:

| | |
|---|---|
| E_CANNOTPARSE | if the JPEG structure could not be parsed |
| E_CANNOTDECODESEQUENTIAL | if a sequentially encoded image could not be decoded |
| E_MODENOTSUPPORTED | if the image is in an unsupported format |
| E_OUT_OF_MEMORY | if memory could not be allocated (heap is not large enough) |
| E_EVALUATIONLIMITREACHED | if an evaluation package limit is reached |
| E_FORMATMISMATCH | if the incorrect JPEG parameters were specified using the JPEG_Param_Config function, or if the previous JPEG bitstream decoded using the same JPEG Decoder Handle is encoded with different parameters to the current JPEG bitstream |
| E_FAILURE | otherwise. |

***Description:***

Decodes a JPEG frame in sequential (or baseline) mode. When the JPEG decoder has been set to decode in SEQUENTIAL mode (when it is created by JPEG_Decoder_NEW()) this function is the same as JPEG_DecodeImage(). If the decoding mode has not been set to SEQUENTIAL mode, an error code is returned.

### 4.2.6. JPEG_DecodeProgressiveImage

***Function Name:***

int JPEG_DecodeProgressiveImage (tJpegDecoder* handle, tJpegParam *param)

***Parameters:***

handle [IN]: Handle to JPEG decoder instance.

param [IN/OUT]: pointer to a parameter structure

***Return Value:***

E_SUCCESS if successful, or:

| | |
|---|---|
| E_CANNOTPARSE | if the JPEG structure could not be parsed |
| E_CANNOTDECODEPROGRESSIVE | if a progressively encoded image could not be decoded |
| E_MODENOTSUPPORTED | if the image is in an unsupported format |
| E_OUT_OF_MEMORY | if memory could not be allocated (heap is not large enough) |
| E_EVALUATIONLIMITREACHED | if an evaluation package limit is reached |
| E_FORMATMISMATCH | if the incorrect JPEG parameters were specified using the JPEG_Param_Config function, or if the previous JPEG bitstream decoded using the same JPEG Decoder Handle is encoded with different parameters to the current JPEG bitstream |
| E_FAILURE | otherwise. |

***Description:***

Decodes a JPEG frame in progressive mode. When the JPEG decoder has been set to decode in PROGRESSIVE mode (when it is created by JPEG_Decoder_NEW()) this function is the same as JPEG_DecodeImage(). If the decoding mode has not been set to PROGRESSIVE mode, an error code is returned.