JPEG System IO Buffer Modules
Supplementary Developer's Guide

DEVIM1-001-C
October 2005

Analog Devices Inc

www.analog.com

Table of Contents

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

2. Implementation Details 8
2.1. User-Defined Data Access Modules 8
2.2. Functionality 8
2.2.1. Raw image and Bit stream data buffer function calls 8
2.2.1.1. Encoder 8
2.2.1.2. Decoder 10
2.2.2. Memory allocation function calls 12

3. Programmer’s Reference 12
3.1. User-Defined Data Structures 12
3.1.1. MCU Buffer Structure 12
3.1.2. Bits Buffer Structure 13
3.1.2.1. Decoder (input) 13
3.1.2.2. Encoder (output) 13
3.1.3. Temporary Bits Buffer Structure 13
3.1.4. Memory Allocation Structure 13
3.2. Raw image data buffer functions 14
3.2.1. JPEG_McuBuffer_NEW 14
3.2.2. JPEG_McuBuffer_DELETE 14
3.2.3. JPEG_McuBuffer_CONFIG 14
3.2.4. JPEG_McuBuffer_INIT 15
3.2.5. JPEG_McuBuffer_RESET 15
3.2.6. JPEG_McuBuffer_READ 16
3.2.7. JPEG_McuBuffer_PROCESS 16
3.2.8. JPEG_McuBuffer_RELEASE 16
3.2.9. JPEG_McuBuffer_REQUEST 16
3.2.10. JPEG_McuBuffer_WRITE 17
3.2.11. JPEG_McuBuffer_FLUSH 17

3.3. Bit stream data buffer functions 17
3.3.1. JPEG_BitsBuffer_NEW 17
3.3.2. JPEG_BitsBuffer_DELETE 17
3.3.3. JPEG_BitsBuffer_CONFIG 18
3.3.4. JPEG_BitsBuffer_INIT 18
3.3.5. JPEG_BitsBuffer_RESET 18
3.3.6. JPEG_BitsBuffer_READ 19
3.3.7. JPEG_BitsBuffer_PROCESS 19
3.3.8. JPEG_BitsBuffer_RELEASE 19
3.3.9. JPEG_BitsBuffer_REQUEST 19
3.3.10. JPEG_BitsBuffer_WRITE 20
3.3.11. JPEG_BitsBuffer_FLUSH 20
3.4. Temporary bit stream buffer functions 20
3.4.1. JPEG_TempBuffer_NEW 20
3.4.2. JPEG_TempBuffer_DELETE 20
3.4.3. JPEG_TempBuffer_CONFIG 20
3.4.4. JPEG_TempBuffer_INIT 21
3.4.5. JPEG_TempBuffer_RESET 21
3.4.6. JPEG_TempBuffer_REQUEST 21
3.4.7. JPEG_TempBuffer_WRITE 21
3.4.8. JPEG_TempBuffer_FLUSH 21
3.4.9. JPEG_TempBuffer_CAT 21
3.5. Memory allocation module functions 21
3.5.1. JPEG_MemAlloc_NEW 21
3.5.2. JPEG_MemAlloc_DELETE 22
3.5.3. JPEG_MemAlloc_ADDRESS 22

List of Tables

Table 1: Raw image and Bit stream data buffer calls for JPEG_Encoder_NEW 9
Table 2: Raw image and Bit stream data buffer calls for JPEG_EncodeImage 9
Table 3: Raw image and Bit stream data buffer calls for JPEG_Encoder_DELETE 10
Table 4: Raw image and Bit stream data buffer calls for JPEG_Decoder_NEW 10
Table 5: Raw image and Bit stream data buffer calls for JPEG_ProcessHeader 10
Table 6: Raw image and Bit stream data buffer calls for JPEG_DecodeImage 11
Table 7: Raw image and Bit stream data buffer calls for JPEG_Decoder_DELETE 11
Table 8 MCU Data Format 13
Table 9 Configuration Parameters for MCU Buffers (Pre-defined) 15
Table 10 Configuration Parameters for BitsBuffer (Pre-defined) 18

List of Figures

Figure 1 Data Access Module 8

1. Introduction

This document describes the user defined Data Access modules that accompany the JPEG libraries.

1.1. Scope

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

1.2. Target platform

These modules were designed for the Analog Devices Blackfin processors.

1.3. Organisation of this Guide

Information supplied in this guide is organised in the following way. Section 2 provides a programmers reference of the buffer module requirements.

1.4. Version Information

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 supplied data access modules were created and tested in conjunction with the JPEG libraries supplied by ADI. To ensure the maximum level of compatibility, please use the versions of the tools specified in the JPEG Library developer’s guides.

1.6. References

[1] ADI, “JPEG Encoder Library Developer's Guide”, DEVIMG-002.

[2] ADI, “JPEG Decoder Library Developer's Guide”, DEVIMG-001.

1.7. 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. Implementation Details

2.1. User-Defined Data Access Modules

This module provides data access functions for the JPEG Libraries. The functions are defined in their entirety by the user, to allow memory access and implementation to be determined by the system in which the JPEG library is to be integrated. Thus the user can customise the JPEG Libraries for specific processing environments, allowing optimal memory transfer methods, such as DMA, to be implemented. The function of the Data Access Module is illustrated in Figure 1.

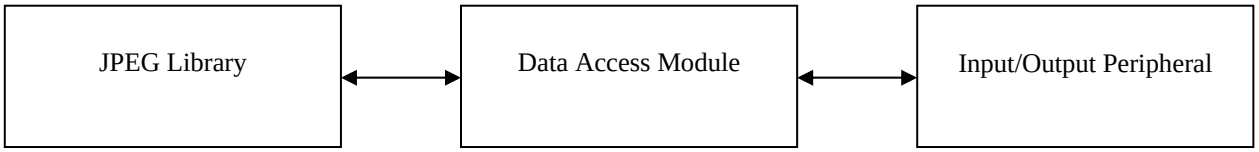

Figure 1 Data Access Module

The code which calls these functions are embedded within the JPEG libraries and are not accessible to the user (see [1] for details). The function implementation, the prototypes of which are specified in this section, are *supplied by the user and is system dependant*. This allows flexibility in the choice of data format of the source image data, and allows access to memory in any way. The user must, of course, adhere to the function's parameter list and return type, and must fulfil the purpose of the function, which is defined in this specification. These modules use the concept of data pipes in managing the flow of data in and out of the JPEG Library.

The Data Access Module consists of four groups of functions: the raw image data buffer functions, bit stream data buffer functions, temporary bit stream buffer functions, and memory allocation functions.

ADI will provide example function implementations for common access methods. A basic example contained in the files JPEG_MCUBuffer.c and JPEG_BitsBuffer.c uses external memory as the source/sink for input/output data. The user may rewrite these modules to bypass the external memory and connect directly peripherals (such as the PPI and SPI), depending on their particular system configuration.

2.2. Functionality

2.2.1. Raw image and Bit stream data buffer function calls

2.2.1.1. Encoder

The library functions **`JPEG_Encoder_NEW`**, **`JPEG_EncodeImage`**, and **`JPEG_Encoder_DELETE`** call the following raw image and bitstream buffer functions, as shown by Table 1, Table 2, and Table 3, respectively. The table shows that the Encoder Library calls the functions in a predefined order, and the Data Access Module function must be written by the user to fulfil the function's predefined action. The exact implementation of the function's action (for example, utilising double buffering, multiple processors, etc) is entirely up to the user.

Please note that for PROGRESSIVE mode the encoding loop in Table 2 is repeated for each MCU over each YUV component. Therefore for all modes except greyscale the processing loops over each MCU 3 times.

When encoding using PROGRESSIVE mode the TempBuffer module will also be utilised. The TempBuffer functions are called at the same time as their corresponding bits buffer functions. The additional TempBuffer function **`JPEG_TempBuffer_CAT`** is called as the very last operation to be performed by **`JPEG_EncodeImage`**, after the last function call in Table 2 is complete.

Step	Encoder Library Action	Encoder Library Function Call	Function Action
1	Create the input (MCU) and output (Bits) buffers.	JPEG_McuBuffer_NEW	Create a new instance of the input MCU buffer module.
		JPEG_BitsBuffer_NEW	Create a new instance of the output bit buffer stream.
2	Set the encoding parameters of the MCU and Bits buffers. Call once for each parameter.	JPEG_McuBuffer_CONFIG	The following parameters are provided by the JPEG library MCU_ENCODER (true), MCU_FRAME_WIDTH (JPEG_FRAME_WIDTH), MCU_FRAME_HEIGHT (JPEG_FRAME_HEIGHT), MCU_IMAGE_FORMAT (JPEG_IMAGEFORMAT), MCU_ENCODING_MODE (JPEG_ENCODINGMODE), MCU_POINTER_C1 (JPEG_POINTER_INPUT)
		JPEG_BitsBuffer_CONFIG	The following parameters are provided by the JPEG library BITSBUF_ENCODER (true), BITSBUF_POINTER (JPEG_POINTER_OUTPUT)
3	Initialise the buffers	JPEG_McuBuffer_INIT JPEG_BitsBuffer_INIT	Initialise any memory required by the buffer objects.

Table 1: Raw image and Bit stream data buffer calls for JPEG_Encoder_NEW

Step	Encoder Library Action	Encoder Library Function Call	Function Action
1	Reset the input (MCU) and output (bit) buffers in preparation for a new image to encode.	JPEG_McuBuffer_RESET	Initialise all state variables of the MCU buffer.
		JPEG_BitsBuffer_RESET	Initialise all state variables of the output bit buffer stream.
2	Configure the JPEG header.		
3	Write the JPEG header.	JPEG_BitsBuffer_REQUEST JPEG_BitsBuffer_WRITE	(See below)
4	Pre-Load an MCU buffer.	JPEG_McuBuffer_READ	Read the next input MCU buffer
START of processing loop, for each MCU			
5a	Obtain a pointer to the next available Bit Stream buffer to be filled by the processed output data. Preload an MCU ahead of time. *Note: these steps may occur concurrently or in the reverse order.*	JPEG_BitsBuffer_REQUEST	Select the next available output bit stream buffer to be filled & return a pointer to it.
		JPEG_McuBuffer_READ	Read the next input MCU buffer
5b	Obtain a pointer to the current MCU buffer to be processed.	JPEG_McuBuffer_PROCESS	Select the next MCU buffer to be processed by the library, make it the current buffer & return a pointer to it.
5c	Encode the current MCU buffer.		
5d	Release the MCU buffer.	JPEG_McuBuffer_RELEASE	Free the current MCU buffer
5e	Send the encoded data to the output bit stream.	JPEG_BitsBuffer_WRITE	Write the encoded bitstream data to the available output bit stream buffer selected by JPEG_BitsBuffer_REQUEST.
END of processing loop, for each MCU			
6	Configure the JPEG end-of-file		
7	Write the JPEG end-of-file	JPEG_BitsBuffer_REQUEST JPEG_BitsBuffer_WRITE	
8	Flush the output buffer at the end of all processing	JPEG_BitsBuffer_FLUSH	Write the remaining contents of the output buffer.

Table 2: Raw image and Bit stream data buffer calls for JPEG_EncodeImage

Step	Encoder Library Action	Encoder Library Function Call	Function Action
1	Delete the MCU and Bits buffers.	JPEG_McuBuffer_DELETE	Delete the MCU buffer instance.
		JPEG_BitsBuffer_DELETE	Delete the bits buffer instance.

Table 3: Raw image and Bit stream data buffer calls for JPEG_Encoder_DELETE

2.2.1.2. Decoder

The library functions **`JPEG_Decoder_NEW`**, **`JPEG_ProcessHeader`**, **`JPEG_DecodeImage`**, and **`JPEG_Decoder_DELETE`** call the following raw image and bitstream buffer functions, as shown by Table 4, Table 5, Table 6, and Table 7, respectively. The table shows that the Decoder Library calls the functions in a predefined order, and the Data Access Module function must be written by the user to fulfil the function's predefined action. The exact implementation of the function's action (for example, utilising double buffering, multiple processors, etc) is entirely up to the user.

Note that the input and output sections in Table 6 are repeated as many times as necessary during the decoding process.

Step	Decoder Library Action	Decoder Library Function Call	User-Defined Function Action
1	Create the input (Bits) and output (MCU) buffers.	JPEG_BitsBuffer_NEW	Create a new instance of the output bit buffer stream.
		JPEG_McuBuffer_NEW	Create a new instance of the MCU buffer.
2	Set the encoding parameters of the MCU and Bits buffers. Call once for each parameter.	JPEG_BitsBuffer_CONFIG	The following parameters are provided by the JPEG library BITSBUF_ENCODER (false), BITSBUF_POINTER (JPEG_POINTER_INPUT)
3	Initialise the Input Bit stream buffer Module	JPEG_BitsBuffer_INIT	Initialise any memory required by the buffer objects.

Table 4: Raw image and Bit stream data buffer calls for JPEG_Decoder_NEW

Step	Decoder Library Action	Decoder Library Function Call	User-Defined Function Action
1	**Reset Section:**		
1.1	Reset the input (bit) buffer module in preparation for a new image to decode.	JPEG_BitsBuffer_RESET	Initialise all state variables of the bits buffer.
2	**Input Section (Parsing the JPEG Headers):**		
2.1	Pre-Load a bits buffer.	JPEG_BitsBuffer_READ	Fill the next free bits buffer from input stream
2.2	Obtain a pointer to the current bits buffer to be processed.	JPEG_BitsBuffer_PROCESS	Select the next bits buffer to be processed, make it the current buffer & return a pointer to it.
2.3	Release the bits buffer when the input bitstream data is exhausted.	JPEG_BitsBuffer_RELEASE	Free the current bits buffer, making it available to be refilled.

Table 5: Raw image and Bit stream data buffer calls for JPEG_ProcessHeader

Step	Decoder Library Action	Decoder Library Function Call	Function Action
1	**Reset Section:**		
1.1	Set the encoding parameters of the MCU buffer module. Call once for each parameter	JPEG_McuBuffer_CONFIG	The following parameters are provided by the JPEG library MCU_ENCODER (false), MCU_FRAME_WIDTH (JPEG_FRAME_WIDTH), MCU_FRAME_HEIGHT (JPEG_FRAME_HEIGHT), MCU_ORIGINAL_FRAME_WIDTH (JPEG_IMAGE_WIDTH), MCU_ORIGINAL_FRAME_HEIGHT (JPEG_IMAGE_HEIGHT), MCU_IMAGE_FORMAT (JPEG_IMAGEFORMAT), MCU_ENCODING_MODE (JPEG_ENCODINGMODE), MCU_POINTER_C1 (JPEG_POINTER_OUTPUT)
	Initialise the output (MCU) buffer module	JPEG_McuBuffer_INIT	Initialise any memory required by the buffer objects.
1.2	Reset the output (MCU) buffer module in preparation for a new image to decode.	JPEG_McuBuffer_RESET	Initialise all state variables of the MCU buffer.
2	**Input Section (when current bits buffer has run out of data):**		
2.1	Release the current bits buffer.	JPEG_BitsBuffer_RELEASE	Free the current bits buffer, making it available to be refilled.
2.2	Pre-Load a bits buffer with input bitstream data.	JPEG_BitsBuffer_READ	Fill the next free bits buffer from input stream
2.3	Obtain a pointer to the current bits buffer to be processed.	JPEG_BitsBuffer_PROCESS	Select the next bits buffer to be processed, make it the current buffer & return a pointer to it.
3	**Output Section (when current MCU has been reconstructed):**		
3.1	Obtain a pointer to the next available MCU buffer to be filled by the reconstructed output data.	JPEG_McuBuffer_REQUEST	Select the next available output MCU buffer to be filled & return a pointer to it.
3.2	Send the image data to the "display".	JPEG_McuBuffer_WRITE	Send the reconstructed MCU data to the display (output by the JPEG library in the buffer selected by JPEG_McuBuffer_REQUEST).

Table 6: Raw image and Bit stream data buffer calls for JPEG_DecodeImage

Step	Decoder Library Action	Decoder Library Function Call	Function Action
1	Delete the Bits and MCU buffers.	JPEG_BitsBuffer_DELETE	Delete the bits buffer instance.
		JPEG_McuBuffer_DELETE	Delete the MCU buffer instance.

Table 7: Raw image and Bit stream data buffer calls for JPEG_Decoder_DELETE

2.2.2. Memory allocation function calls

The MemAlloc functions are called from both within the JPEG Decoder library and by some of the other user-defined functions.

JPEG_MemAlloc_ADDRESS is always called immediately following the success of **JPEG_MemAlloc_NEW**. These two functions are called by the following functions: **JPEG_Decoder_NEW**, **JPEG_McuBuffer_NEW**, **JPEG_BitsBuffer_NEW**, **JPEG_TempBuffer_NEW**, and **JPEG_McuBuffer_INIT**.

JPEG_MemAlloc_DELETE is called by **JPEG_Decoder_DELETE**, **JPEG_McuBuffer_DELETE**, **JPEG_BitsBuffer_DELETE**, and **JPEG_TempBuffer_DELETE**, and during any corresponding NEW function call that fails.

3. Programmer's Reference

This section describes each of the functions in the user defined data access modules used by the JPEG libraries.

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).

3.1. User-Defined Data Structures

There are four user defined data structures that correspond to the four groups of functions. The raw image data buffer functions, bit stream data buffer functions, temporary bit stream buffer functions and memory allocation functions operate on the user defined **tMcuBuffer**, **tBitsBuffer**, **tTempBitsBuffer**, and **McuObjHandle** data structures, respectively.

These structures are entirely user-defined (apart from their names), as shown by the example below.

```
/* User-defined structure definitions: */
typedef struct{

    /*
    Place user code here to define the tMcuBuffer components as
    appropriate for the user's buffering requirements.
    ...
    */

} tMcuBuffer;
```

For correct interaction with the library, the user-defined data structures must conform to the following requirements.

3.1.1. MCU Buffer Structure

The user should include key **tMcuBuffer** structure members that define the MCU. These parameters are passed by the JPEG Library to the Data Access Module via the user-defined function **JPEG_McuBuffer_CONFIG**. The parameters are enumerated in Table 9 for that function, and allow the user code the data access module functions to define and manipulate MCU data blocks and buffers. The required arrangement of MCU data is defined in Table 8.

Image Format	MCU Data Format
YUV420	16x16 luminance block arranged in 4 8x8 blocks in raster scan order where each 8x8 block is in contiguous memory, followed by an 8x8 block (raster scanned) of the U component, followed by the 8x8 block of the V component
YUV422	16x8 luminance block arranged in 2 8x8 blocks in left-to-right order where each 8x8 block is in contiguous memory, followed by an 8x8 block (raster scanned) of the U component, followed by the 8x8 block of the V component
YUV444	8x8 luminance block (raster scanned), followed by an 8x8 block (raster scanned) of the U component, followed by the 8x8 block of the V component
RGB	8x8 block of the red component (raster scanned), followed by an 8x8 block (raster scanned) of the green component, followed by the 8x8 block of the blue component
Monochrome	8x8 luminance block (raster scanned)

Table 8 MCU Data Format

3.1.2. Bits Buffer Structure

3.1.2.1. Decoder (input)

The user should ensure that bits buffers are aligned on 4 byte boundaries and are no less than 64 bits in size. When inputting from a JPEG file into a bits buffer, a 64-bit cache allows words of up to 32 bits, of any alignment, to be input. A typical buffer would not be so small, as the JPEG file would be read excessively frequently. The sample bits buffer module uses a buffer size of 2Kbytes.

In addition the JPEG decoder library requires the data contained in the input bits buffers to be byte swapped (within groups of 4 bytes). Note that the Blackfin processor has a little-endian architecture (the address of a 32-bit integer is the address of the least-significant byte of that integer).

For example, the sequence of bytes read from the input bitstream as **`01 23 45 67 89 AB CD EF`** must be rearranged (byte swapped) in the following order in the bits buffer, where the first byte falls on a 4 byte aligned memory location: **`67 45 23 01 EF CD AB 89`**.

3.1.2.2. Encoder (output)

The JPEG Encoder Library outputs the bitstream in the correct byte order and so no byte swapping is required to be performed on the encoded bitstream data.

3.1.3. Temporary Bits Buffer Structure

This data structure can be the same as the main Bits Buffer structure. See section 3.4 for details on how the Temporary Bits Buffer Module is used by the library.

3.1.4. Memory Allocation Structure

The user must ensure that enough information is stored in this data structure to enable effective usage of the API functions defined in Section 3.5.

3.2. Raw image data buffer functions

These functions provide a way for the JPEG systems to manage the MCU data buffer.

3.2.1. JPEG_McuBuffer_NEW

Function Name:
`tMcuBuffer *JPEG_McuBuffer_NEW (void)`
Parameters:
None.
Return Value:
Handle to new MCU buffer instance. NULL upon failure.
Description:
This function must create a new instance of MCU buffer object, and must set all fields to default values. After calling this function, `JPEG_McuBuffer_CONFIG` is called by the library to set all fields to required values.

3.2.2. JPEG_McuBuffer_DELETE

Function Name:
`void JPEG_McuBuffer_DELETE (tMcuBuffer *handle)`
Parameters:
handle [IN]: Handle to MCU buffer instance
Return Value:
None
Description:
This function must delete the MCU buffer object.

3.2.3. JPEG_McuBuffer_CONFIG

Function Name:
`int JPEG_McuBuffer_CONFIG (tMcuBuffer *handle, eMcu_config item, unsigned int value)`
Parameters:

handle [IN]:	Handle to MCU Buffer data access instance.
item [IN]:	Item to configure (see Table 9).
value [IN]:	Configuration value (see Table 9). Note that these parameters are pre-defined. The user code must include them.

Item Parameter: one of	Value
MCU_POINTER_C1	Pointer to a system specific data structure containing information for the low level buffer. This parameter is the pointer passed in to the JPEG Encoder Library as **JPEG_POINTER_INPUT** or passed in to the JPEG Decoder Library as **JPEG_POINTER_OUTPUT**. The definition of the data structure and its management are user defined. *In the example code, memory address for the output image frame data for the given image format and size*.
MCU_ORIGINAL_FRAME_HEIGHT	Original height of frame (before padding)
MCU_ORIGINAL_FRAME_WIDTH	Original width of frame (before padding)
MCU_FRAME_HEIGHT	Height of padded frame
MCU_FRAME_WIDTH	Width of padded frame
MCU_IMAGE_FORMAT	Denotes the format of the input. Uses the values of **JPEG_IMAGEFORMAT** as defined in [1].
MCU_ENCODING_MODE	Denotes the encoding mode. Uses values of **JPEG_ENCODINGMODE** as defined in [1].

Table 9 Configuration Parameters for MCU Buffers (Pre-defined)

Return Value:

TRUE(1) or FALSE(0).

Description:

This function must configure the instance of an image data access object with the given configuration value. Items that must be configured include the chosen data format (YUV420, YUV422 etc), the pointer to the whole image data (in external memory), image width and height, etc.

3.2.4. JPEG_McuBuffer_INIT

Function Name:

```
int JPEG_McuBuffer_INIT (tMcuBuffer *handle)
```

Parameters:

handle [IN]: Handle to MCU Buffer data access instance.

Return Value:

TRUE(1) or FALSE(0).

Description:

This function must initialise the instance of an image data access object, allocating any memory. It is called by the library after the final call to `JPEG_McuBuffer_CONFIG`.

3.2.5. JPEG_McuBuffer_RESET

Function Name:

```
int JPEG_McuBuffer_RESET (tMcuBuffer *handle)
```

Parameters:

handle [IN]: Handle to MCU Buffer data access instance.

Return Value:

TRUE(1) or FALSE(0).

Description:

This function must reset the instance of an image data access object, setting all states to initial values. Fields which are set through calls to `JPEG_McuBuffer_CONFIG` must not be affected.

3.2.6. JPEG_McuBuffer_READ

Function Name:

```
int JPEG_McuBuffer_READ (tMcuBuffer *handle)
```

Parameters:

handle [IN]: Handle to MCU Buffer data access instance.

Return Value:

TRUE(1) or FALSE(0).

Description:

This function must preload a data buffer for processing. FALSE must be returned if there are no empty data buffers available for filling. This function is called by the library at least once before calling the function `JPEG_McuBuffer_PROCESS`.

3.2.7. JPEG_McuBuffer_PROCESS

Function Name:

```
unsigned char *JPEG_McuBuffer_PROCESS (tMcuBuffer *handle)
```

Parameters:

handle [IN]: Handle to MCU Buffer data access instance.

Return Value:

Pointer to the data buffer containing data ready for processing.

Description:

This function must indicate to the system that the next available MCU buffer is ready to be processed. The MCU data buffer must previously have been filled by a call to the function `JPEG_McuBuffer_READ`. When processing is finished, the library calls `JPEG_McuBuffer_RELEASE` to release the data buffer. NULL is to be returned if there is no valid data buffer available for processing.

3.2.8. JPEG_McuBuffer_RELEASE

Function Name:

```
void JPEG_McuBuffer_RELEASE (tMcuBuffer *handle)
```

Parameters:

handle [IN]: Handle to MCU Buffer data access instance.

Return Value:

None.

Description:

This function must release a data buffer that is currently being processed. This is called by the library after calling the function `JPEG_McuBuffer_PROCESS`.

3.2.9. JPEG_McuBuffer_REQUEST

Function Name:

```
unsigned char *JPEG_McuBuffer_REQUEST (tMcuBuffer *handle)
```

Parameters:

handle [IN]: Handle to MCU Buffer data access instance.

Return Value:

Pointer to an empty data buffer ready for storing the processing output. NULL upon failure.

Description:

This function must obtain the next available MCU buffer about to be filled by the processed output data. When processing is finished, the library calls `JPEG_McuBuffer_WRITE` to empty and release the data buffer. NULL is to be returned if there is no valid data buffer available for filling.

3.2.10. JPEG_McuBuffer_WRITE

Function Name:
`int JPEG_McuBuffer_WRITE (tMcuBuffer *handle)`
Parameters:
handle [IN]: Handle to MCU Buffer data access instance.
Return Value:
TRUE(1) or FALSE(0).
Description:
This function must empty a data buffer to the system's output mechanism. FALSE is returned if there was no data buffer being processed. This is called by the library after calling the function `JPEG_McuBuffer_REQUEST` to indicate that the library has fished processing data in this buffer.

3.2.11. JPEG_McuBuffer_FLUSH

Function Name:
`int JPEG_McuBuffer_FLUSH (tMcuBuffer *handle)`
Parameters:
handle [IN]: Handle to MCU Buffer data access instance.
Return Value:
Zero on failure, non-zero otherwise.
Description:
This function must flush all data buffers at the end of all processing.

3.3. Bit stream data buffer functions

These functions provide a way for the JPEG systems to manage the Bit stream data buffer.

3.3.1. JPEG_BitsBuffer_NEW

Function Name:
`tBitsBuffer *JPEG_BitsBuffer_NEW (void)`
Parameters:
None.
Return Value:
Handle to new Bit Stream buffer instance. NULL upon failure.
Description:
This function creates a new instance of Bit Stream buffer object, and must set all fields to default values. After calling this function, `JPEG_BitsBuffer_CONFIG` is called by the library to set some of the fields to required values.

3.3.2. JPEG_BitsBuffer_DELETE

Function Name:
`void JPEG_BitsBuffer_DELETE (tBitsBuffer *handle)`
Parameters:
handle [IN]: Handle to Bit Stream buffer instance
Return Value:
None
Description:
Deletes the Bit Stream buffer object.

3.3.3. JPEG_BitsBuffer_CONFIG

Function Name:

```
int JPEG_BitsBuffer_CONFIG (tBitsBuffer *handle, eBitsBuf_config item, unsigned int value)
```

Parameters:

handle [IN]: Handle to Bit Stream buffer instance

item [IN]: Item to configure (see Table 10).

value [IN]: Configuration value (see Table 10). Note that these parameters are pre-defined. The user code must include them.

Item Parameter: one of	Value
BITS_POINTER	Pointer to a system specific data structure containing information for the low level buffer. This parameter is the pointer passed in to the JPEG Encoder Library as **JPEG_POINTER_OUTPUT** or passed in to the JPEG Decoder Library as **JPEG_POINTER_INPUT**. The definition of the data structure and its management are user defined. *In the example code, when used in the decoder, this structure contains a pointer to a buffer containing the entire JPEG bitstream and the length of the bitstream in bytes.*

Table 10 Configuration Parameters for BitsBuffer (Pre-defined)

Return Value:

TRUE(1) or FALSE(0).

Description:

This function must configure the instance of an image data access object with the given configuration value. Items that are configured are listed in Table 10.

3.3.4. JPEG_BitsBuffer_INIT

Function Name:

```
int JPEG_BitsBuffer_INIT (tBitsBuffer *handle)
```

Parameters:

handle [IN]: Handle to Bit Stream buffer instance

Return Value:

TRUE(1) or FALSE(0).

Description:

This function initialises the instance of a bitstream buffer object, allocating any memory. It is called by the library after the final call to `JPEG_BitsBuffer_CONFIG`.

3.3.5. JPEG_BitsBuffer_RESET

Function Name:

```
int JPEG_BitsBuffer_RESET (tBitsBuffer *handle)
```

Parameters:

handle [IN]: Handle to Bit Stream Buffer data access instance.

Return Value:

TRUE(1) or FALSE(0).

Description:

This function must reset the instance of a bit stream data access object, setting all states to default starting values. Fields which are set through calls to `JPEG_BitsBuffer_CONFIG` must not be affected.

3.3.6. JPEG_BitsBuffer_READ

Function Name:

```
int JPEG_BitsBuffer_READ (tBitsBuffer *handle)
```

Parameters:

handle [IN]: Handle to Bit Stream Buffer data access instance.

Return Value:

TRUE(1) or FALSE(0).

Description:

This function must preload a data buffer for processing by the JPEG library. FALSE must be returned if there are no empty data buffers available for filling. This is called by the JPEG library at least once before calling the function `JPEG_BitsBuffer_PROCESS`. If the system cannot obtain any more data from the bitstream source, then the end of image marker should be sent to the JPEG library (the 2 byte sequence 0xFF, 0xD9) by appending these 2 bytes at the end of the stream of bytes piped to the JPEG library.

3.3.7. JPEG_BitsBuffer_PROCESS

Function Name:

```
unsigned char *JPEG_BitsBuffer_PROCESS (tBitsBuffer *handle, int* pNumBytes)
```

Parameters:

handle [IN]: Handle to Bit Stream Buffer data access instance.

pNumBytes [OUT]: Pointer to address the length of the buffer will be stored.

Return Value:

Pointer to the data buffer containing data ready for processing. NULL upon failure.

Description:

This function must indicate to the system that the next available Bit Stream buffer is about to be processed. The Bit Stream data buffer can be assumed to have been filled by the library after calling the function `JPEG_BitsBuffer_READ`. When processing is finished, `JPEG_BitsBuffer_RELEASE` is called by the library to release the data buffer. NULL is to be returned if there is no valid data buffer available for processing. On success, the length of the buffer (bytes) is to be written to the location pointed to by pNumBytes.

3.3.8. JPEG_BitsBuffer_RELEASE

Function Name:

```
void JPEG_BitsBuffer_RELEASE (tBitsBuffer *handle)
```

Parameters:

handle [IN]: Handle to Bit Stream Buffer data access instance.

Return Value:

None.

Description:

This function must release a data buffer that is currently being processed. This is called by the library after calling the function `JPEG_BitsBuffer_PROCESS`.

3.3.9. JPEG_BitsBuffer_REQUEST

Function Name:

```
unsigned char *JPEG_BitsBuffer_REQUEST (tBitsBuffer *handle)
```

Parameters:

handle [IN]: Handle to Bit Stream Buffer data access instance.

Return Value:

Pointer to an empty data buffer ready for storing the processing output. NULL upon failure.

Description:
This function must obtain the next available Bit Stream buffer to be filled by the processed output data. When processing is finished, the library will make a call to `JPEG_BitsBuffer_WRITE` to empty and release the data buffer. NULL is to be returned if there is no valid data buffer available for filling.

3.3.10. JPEG_BitsBuffer_WRITE

Function Name:
`int JPEG_BitsBuffer_WRITE (tBitsBuffer *handle, int NumBytes)`
Parameters:
handle [IN]: Handle to Bit Stream Buffer data access instance.
NumBytes [IN]: The number of bytes that have been written to the buffer since calling `JPEG_BitsBuffer_REQUEST`.
Return Value:
TRUE(1) or FALSE(0).
Description:
This function is used to empty a data buffer after processing and write the data to the output bit stream. FALSE is to be returned if there is no data buffer to be processed. This is called by the library after calling the function `JPEG_BitsBuffer_REQUEST`.

3.3.11. JPEG_BitsBuffer_FLUSH

Function Name:
`int JPEG_BitsBuffer_FLUSH (tBitsBuffer *handle)`
Parameters:
handle [IN]: Handle to Bit Stream Buffer data access instance.
Return Value:
Total number of bytes processed since calling JPEG_BitsBuffer_RESET().
Description:
This function must flush all data buffers at the end of all processing.

3.4. Temporary bit stream buffer functions

These functions provide a way for the JPEG systems to manage the temporary bit stream data buffer. The temporary bit stream data buffers are slightly different to the Bit stream data buffers in that the data can be retrievable. This behaviour is necessary when encoding images in progressive mode.

As an example, when data has been written to the temp buffer by the JPEG encoder (using **`JPEG_TempBuffer_WRITE`**) it will need to be read from that buffer later (by **`JPEG_TempBuffer_CAT`**). In contrast, when data has been written to the bit stream buffer (by **`JPEG_BitsBuffer_WRITE`**), that data is not used by the JPEG Encoder again.

3.4.1. JPEG_TempBuffer_NEW

Please refer to the function description for **`JPEG_BitsBuffer_NEW`** in section 3.3.1.

3.4.2. JPEG_TempBuffer_DELETE

Please refer to the function description for **`JPEG_BitsBuffer_DELETE`** in section 3.3.2.

3.4.3. JPEG_TempBuffer_CONFIG

Please refer to the function description for **`JPEG_BitsBuffer_CONFIG`** in section 3.3.3.

3.4.4. JPEG_TempBuffer_INIT

Please refer to the function description for **JPEG_BitsBuffer_INIT** in section 3.3.4.

3.4.5. JPEG_TempBuffer_RESET

Please refer to the function description for **JPEG_BitsBuffer_RESET** in section 3.3.5.

3.4.6. JPEG_TempBuffer_REQUEST

Please refer to the function description for **JPEG_BitsBuffer_REQUEST** in section 3.3.9.

3.4.7. JPEG_TempBuffer_WRITE

Please refer to the function description for **JPEG_BitsBuffer_WRITE** in section 3.3.10.

3.4.8. JPEG_TempBuffer_FLUSH

Please refer to the function description for **JPEG_BitsBuffer_FLUSH** in section 3.3.11.

3.4.9. JPEG_TempBuffer_CAT

Function Name:

```
void JPEG_TempBuffer_CAT (tBitsBuffer *dest, const tTempBitsBuffer *src)
```

Parameters:

dest [OUT]:	Pointer to Bit Stream data access instance. NOTE: The destination Bit Stream memory must be large enough to hold the concatenated data.
src [IN]:	Pointer to Temporary Bit Stream data access.

Return Value:

None.

Description:

This function must concatenate the contents of a Temporary buffer to a Bits Buffer.

3.5. Memory allocation module functions

These user-defined functions provide a way for the user to control how memory allocation is performed in the system.

3.5.1. JPEG_MemAlloc_NEW

Function Name:

```
MemObjHandle *JPEG_MemAlloc_NEW (int N_element, int size, Mem_type attr)
```

Parameters:

N_element [IN]:	the number of elements that the requested area of memory will store
size [IN]:	the number of bytes occupied by each element.
attr [IN]:	one of **MEM_TYPE_DATA, MEM_TYPE_TABLE** and **MEM_TYPE_OBJECT** that identifies the type of memory being allocated. Note that these parameters are pre-defined. The user cannot modify them.

Return Value:
Pointer to a memory object handle that contains a pointer to the (nobj x size) bytes uninitialised memory space. NULL upon failure.

Description:
This function is used by the library to allocate memory. When it is not required any more, the library calls `JPEG_MemAlloc_DELETE` to release the data buffer memory. NULL is to be returned if there is not enough memory available. Please note that all memory allocated by this function must be aligned to 4 byte boundaries.

3.5.2. JPEG_MemAlloc_DELETE

Function Name:
```
void JPEG_MemAlloc_DELETE (MemObjHandle *mem_obj)
```
Parameters:
mem_obj [IN]: Handle to memory object instance.

Return Value:
None

Description:
This function must free the data buffer memory object that was allocated after calling the function `JPEG_MemAlloc_NEW`.

3.5.3. JPEG_MemAlloc_ADDRESS

Function Name:
```
void *JPEG_MemAlloc_ADDRESS (MemObjHandle *mem_obj)
```
Parameters:
mem_obj [IN]: Pointer to memory object

Return Value:
Pointer to an uninitialised memory space containing (N_element x size) bytes that was allocated by `JPEG_MemAlloc_NEW`.

Description:
This function must return a pointer to the data buffer memory that was allocated within the memory object `mem_obj`.