JPEG Encoder Library Developer's Guide

DEVIMG-002-H
October 2005

Analog Devices Inc.

www.analog.com

Table of Contents

List of Tables

List of Figures

1. Introduction

This document describes how to use the JPEG Encoder Library.

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

The JPEG library was 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 the specifications of the implemented JPEG Encoder. In section 3, notes on how to use and integrate the library are given. Finally section 3.2.1 describes the high-level software structure of the library.

1.4. Version Information

Note that this document refers to the following software versions:
JPEG Encoder: ADI Version 3.0.3

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 Encoder 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 System IO Buffer Modules Supplementary Developer's Guide”, DEVIM1-001-B, Aug 2004.

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

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 encoder library implements still image compression according to the standard [1] for image data, using one of 3 possible encoding processes with restrictions as described below.

All images are input as 8 bit samples within each component (in a non interleaved fashion). When the number of components is 3, 2 Huffman AC tables, 2 Huffman DC tables, and 2 Quantisation tables are specified. These 2 sets of tables are used such that one is for the first component (nominally the luminance), and the second set is used for the next 2 components (nominally the chrominance components). The programmer can also control the quality of encoding by selecting a level of quantisation (a scaled copy of the quantisation matrices specified in Annex K of the JPEG standard [1]), or by using an adaptive quantisation algorithm.

The format of the output image file is not programmable, and is fixed to satisfy the JFIF specification when the input image data is in YUV format, or in JFIF/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 the APP14 marker to indicate that Adobe information is present at the start of the bit stream. Note that the Adobe file format is different from Adobe's Portable Document Format (PDF).

2.1. Sequential Process

Only image data with 1 or 3 components can be encoded. In the case of 3 components, the 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 or RGB);

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)

The image is encoded in a single scan.

Note that currently the sequential processing currently does not extend beyond baseline processing (which uses 8 bit inputs and Huffman entropy coding).

2.2. Progressive DCT Process (Huffman coding)

Only JPEG bit streams with 1 or 3 components can be encoded using spectral selection. Component sampling factors are restricted in the same way as for the Sequential process. The AC coefficients are encoded into 3 scans (coefficients 1 to 5 in first scan, 6 to 15 in the second and 16 to 63 in the third).

2.3. Input Image Data Format

The format of the input image data can be in one of 5 different formats: RGB, YUV4:4:4, YUV4:2:2, YUV4:2:0, or Y only (YUV4:0:0). In all formats, the components are not interleaved. In the case of 3 components, the first component is always at the same sampling as the input image frame, while the second and third components can be sub-sampled by a factor of 2:1 in both horizontal and vertical directions (YUV4:2:0), or only in the horizontal direction(YUV4:2:2). Both the second and third components of the input image are sampled in the same way.

The encoder library does not do any further sub-sampling or colour space transformation of the input image data. Also, the dimension of the input image is restricted, depending on the format of the input image data. For YUV4:2:0, the height and width must be a multiple of 16; for YUV4:2:2, the width must be a multiple of 16 and the height a multiple of 8; for the other 3 formats, the height and width must be a multiple of 8. This means that the input image would need to be padded appropriately by the application before calling the library encoder component.

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 and BF561 processors and four different test images in YUV420 format.

Both BF533 and BF561 systems show a near-linear correlation (R^2>0.98) between compression ratio and cycles/pixel. At 1.0 bit/pixel (a compression ratio of 12:1 for YUV420 images), approximately 47 cycles/pixel are expected for the BF533 processor and approximately 40 cycles/pixel are expected for the BF561 processor.

Please refer to the JPEG Product Highlight [3] for current benchmarks.

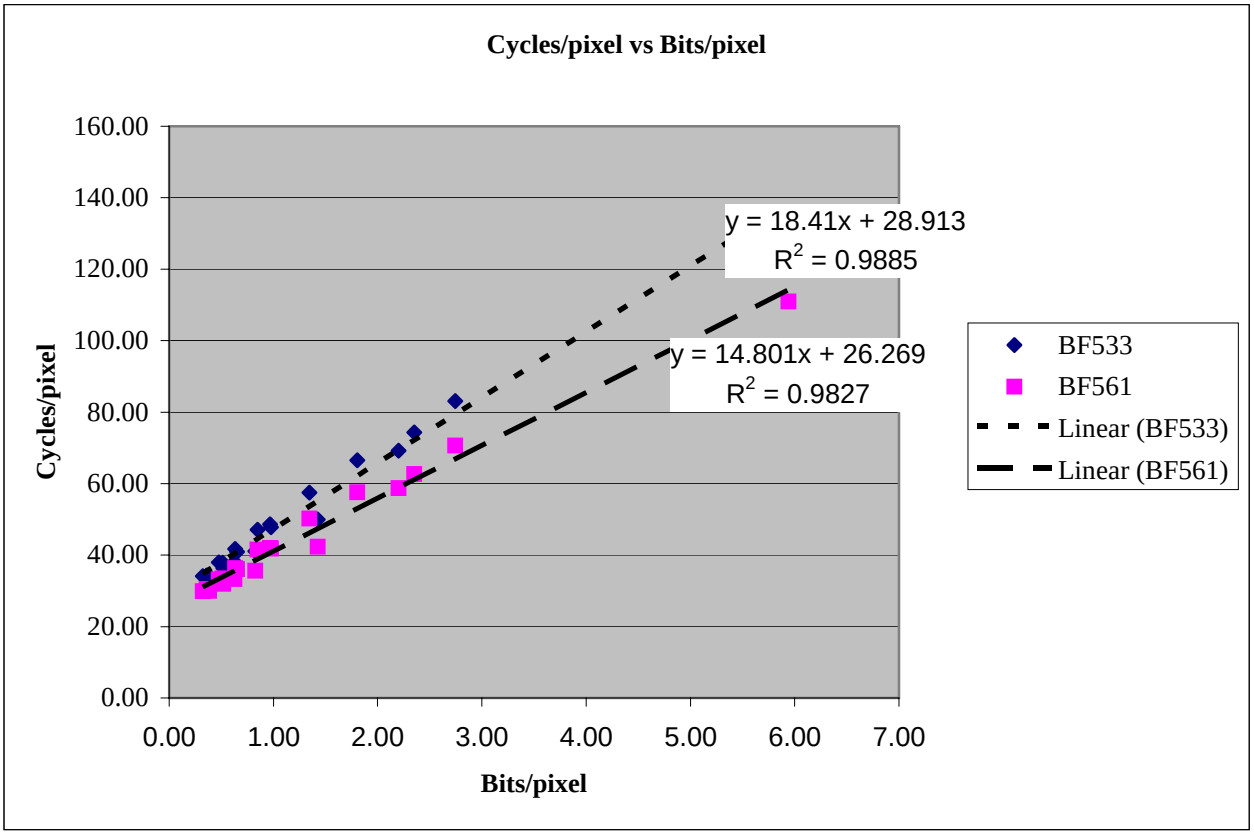

Figure 1 Comparison between BF533 and BF561 Cycles/pixel vs. Bits/pixel

2.5. Memory Requirements

The memory requirements for the JPEG Encoder depend on which mode is used. If only sequential or progressive encoded images are to be encoded, the appropriate format-specific encode function **`JPEG_EncodeSequentialImage`** or **`JPEG_EncodeProgressiveImage`** 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 requirements 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 JPEG Library Release Notes 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 JPEG_P0 (KB)	Program Memory JPEGENC_P0 (KB)	Program Memory JPEGENC_P1 (KB)	Data Memory JPEGENC_D0 (KB)	Heap Maximum (KB)
All encoding modes	0.7	10.3	3.6	4.9	2 + 6N*
Progressive mode only	0.7	6.7	3.6	4.6	2 + 6N*
Sequential mode only	0.7	9.1	-	3.3	2

Table 2: Blackfin JPEG Encoder 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:

- JPEGENC_P0: all JPEG code used in sequential mode, plus all JPEG code common to both modes
- JPEGENC_P1: JPEG code used only in progressive mode
- JPEG_P0: JPEG code that is common to both the JPEG encoder and Encoder libraries
- JPEGENC_D0: important 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 encoder library in an application, the library file **`jpeg_enc_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_encoder.h`** in source files where JPEG Library functions are called.

3.2.1. Replacing the Quantisation Tables

Default quantisation tables have been included in the encoder library. These tables are identical to those given as examples in Annex K.1 of the JPEG standard [1]. However, as Annex K.1 states, the tables are derived empirically using luminance and chrominance and 2:1 horizontal subsampling. They are provided as examples only and are not necessarily suitable for any particular application. These quantization values have been used with good results on 8-bit per sample luminance and chrominance images that are interleaved and subsampled, such as the YUV images input to the JPEG encoder.

To use different tables edit the example file JPEG_QuantizationTables.c in the demos\JPEG_Enc\code\src\ directory. Including the edited file in the application build will replace the default quantisation tables.

3.3. User-Defined Data Access Module

The user defined Data Access Module provides memory access functions for the JPEG Encoder Library, and allows the user to customise the input/output mechanism of the JPEG Encoder 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 Encoder 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 [2]. 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

The simplest method to encode a JPEG bit stream is to call the library functions in the following order:

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

ii. Call **`JPEG_Param_CONFIG()`** to set all the image parameters, input pointer (to the input buffer configuration object), and output pointer (to the output buffer configuration object).

iii. Call **`JPEG_Encoder_NEW()`** to create an instance of the JPEG encoder to handle this particular image using a set of parameters controlling the encoding process. This function can be called once for multiple images if they have the same JPEG parameters (i.e., provided **`JPEG_Param_CONFIG()`** is not called between each call of **`JPEG_Encoder_NEW()`**).

iv. Call **`JPEG_EncodeImage()`** (or **`JPEG_EncodeSequentialImage()`** etc) to encode the JPEG image (the user must first make sure the output bit stream buffer provided to this function call is large enough to store the compressed JPEG data),

v. Call **`JPEG_Encoder_DELETE()`** to delete this instance of the JPEG encoder.

The following code example uses these functions to JPEG-encode a sample 640x480 input stream. See the respective entries in the Programmer's Reference (3.2.1). For the format of the image source data, see the reference for the function JPEG_EncodeImage (4.2.3). Note that, as the dimensions of the image in this example are divisible by 16, no padding is required.

```
TJpegEncoder     lBaseJpegEnc = NULL;                  /* Handle to encoder object */
tJpegParam       lImageEncParam = { 0 };               /* JPEG parameters */
uint8            *FrameBuffer = NULL;                  /* Pointer to input data stream */
uint8            *StreamBuffer = NULL;                 /* Pointer to output data stream */
uint32           NumBytes = 0;                         /* Number of bytes encoded */

/*
    ...
    Allocate FrameBuffer and StreamBuffer buffers.
    Fill FrameBuffer with image source data, adding padding if required.
    ...
*/

JPEG_Param_CONFIG(&lImageEncParam, JPEG_FRAME_WIDTH, 640);  /* Horizontal dimension of image */
JPEG_Param_CONFIG(&lImageEncParam, JPEG_FRAME_HEIGHT, 480);  /* Vertical dimension of image */
JPEG_Param_CONFIG(&lImageEncParam, JPEG_IMAGEFORMAT, 1);
JPEG_Param_CONFIG(&lImageEncParam, JPEG_QUALITYFACTOR, 50);
JPEG_Param_CONFIG(&lImageEncParam, JPEG_ENCODINGMODE, SEQUENTIAL);
JPEG_Param_CONFIG(&lImageEncParam, JPEG_THRESHOLD, 0);
JPEG_Param_CONFIG(&lImageEncParam, JPEG_POINTER_INPUT, FrameBuffer);
JPEG_Param_CONFIG(&lImageEncParam, JPEG_POINTER_OUTPUT, StreamBuffer);

/* Create an instance of the JPEG Encoder: */
lBaseJpegEnc = JPEG_Encoder_NEW(&lImageEncParam);

if(lBaseJpegEnc == NULL)
{
    perror("Encoder creation failed\n");
    exit(1);
}

/* Encode the image, and write JPEG image to StreamBuffer: */
if(JPEG_EncodeImage(lBaseJpegEnc, &NumBytes) != 1)
{
    perror("Encoding process failed.\n");
    exit(1);
}

/* Delete the JPEG Encoder object: */
JPEG_Encoder_DELETE(lBaseJpegEnc);
```

4. Programmer’s Reference

This section describes each of the functions in the JPEG 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.

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 format of the input. 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 input format (interleave format 5) the JPEG encoder saves images in Adobe file format (but not PDF). For all other cases, it encodes in JFIF file format.
JPEG_QUALITYFACTOR	This parameter is used to control the quality of encoding. It can take values from 1 (worst quality) to 100 (best).
JPEG_ENCODINGMODE	This parameter is used to select between sequential and progressive encoding modes. Pass SEQUENTIAL for sequential (baseline) mode and PROGRESSIVE for progressive mode.
JPEG_THRESHOLD	This is adaptive thresholding, 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. It is not valid in Progressive mode and should be set to 0 (although its actual value is ignored).
JPEG_POINTER_INPUT	Pointer to the input data buffer, this buffer is owned by the application.
JPEG_POINTER_OUTPUT	Pointer to the output bitstream buffer, this buffer is owned by the application.

Table 3: JPEG Configuration Parameters

Return Value:

Returns E_SUCCESS if configuration was successful,
E_INVALIDPARAMETERITEM if an incorrect parameter was specified
E_FAILURE otherwise.

Description:

This function configures the JPEG parameters for a new encoder.

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 configuration was successful,
E_INVALIDPARAMETERITEM if an incorrect parameter was specified
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 [OUT]: Pointer to JPEG image parameters to initialise.
Return Value:
Returns E_SUCCESS if parameters initialised OK
E_FAILURE otherwise.
Description:
This is an auxiliary function. This function clears a JPEG parameter data structure.

4.2. JPEG Encoder library

4.2.1. JPEG_Encoder_NE W

Function Name:
`tJpegEncoder *JPEG_Encoder_NEW (tJpegParam *param)`
Parameters:
param [IN]: JPEG encoder parameters
Return Value:
Handle to the created JPEG encoder object. NULL upon failure.
Description:
This function creates an instance of JPEG encoder and returns the handle to the JPEG encoder 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_Encoder_DELETE

Function Name:
`void JPEG_Encoder_DELETE (tJpegEncoder* lBaseJpegEnc)`
Parameters:
lBaseJpegEnc [IN]: Handle to JPEG encoder object.
Return Value:
None
Description:
Deletes the JPEG encoder object.

4.2.3. JPEG_EncodeImage

Function Name:

```
int JPEG_EncodeImage (tJpegEncoder* handle, unsigned int* numBytes)
```

Parameters:

handle [IN]: Handle to JPEG encoder instance.

numBytes [OUT]: Number of bytes sent to bit stream after encoding.

Return Value:

Returns E_SUCCESS if image encoded OK

E_EVALUATIONLIMITREACHED if an evaluation package limit is reached

E_FAILURE otherwise

Description:

Encodes an image, with no restrictions on the available modes.

4.2.4. JPEG_EncodeProgressiveImage

Function Name:

```
int JPEG_EncodeProgressiveImage (tJpegEncoder* handle, int* numBytes)
```

Parameters:

handle [IN]: Handle to JPEG encoder instance.

numBytes [OUT]: Number of bytes output in bit stream after encoding.

Return Value:

Returns E_SUCCESS if successful

E_FORMATMISMATCH if image mode is incorrect

E_EVALUATIONLIMITREACHED if an evaluation package limit is reached

E_FAILURE otherwise

Description:

Encodes an image in the progressive DCT-based mode, where the image quality is progressively increased with successive scans. When the JPEG encoder has been set to encode in PROGRESSIVE mode (when it is created by JPEG_Encoder_NEW()) this function is the same as JPEG_EncodeImage(). If the encoding mode has not been set to PROGRESSIVE mode, E_FORMATMISMATCH is returned.

4.2.5. JPEG_EncodeSequentialImage

Function Name:

```
int JPEG_EncodeSequentialImage (tJpegEncoder* handle, int* numBytes)
```

Parameters:

handle [IN]: Handle to JPEG encoder instance.

numBytes [OUT]: Number of bytes output in bit stream after encoding.

Return Value:

Returns E_SUCCESS if successful

E_FORMATMISMATCH if image mode is incorrect

E_EVALUATIONLIMITREACHED if an evaluation package limit is reached

E_FAILURE otherwise

Description:

Encodes an image in the sequential (or baseline) DCT-based mode. When the JPEG encoder has been set to encode in SEQUENTIAL mode (when it is created by JPEG_Encoder_NEW()) this function is the same as JPEG_EncodeImage(). If the encoding mode has not been set to SEQUENTIAL mode, E_FORMATMISMATCH is returned.