

JPEG Encoder User's Manual

C64x platform – version 1.2
Non-XDAIS version

Mecoso Technogy Inc.
Nov 2005

Contents

1	Introduction.....	2
1.1	Overview.....	2
1.2	JPEG Encoding algorithm.....	3
1.3	Performance and memory requirement.....	3
2	Header files.....	4
2.1	Params structure.....	4
2.2	Status structure.....	4
3	API.....	4
4	Example.....	14

1 Introduction

1.1 Overview

This document describes Mecoso's JPEG Encoder library for TI's C64x platforms. The library is *not* XDAIS-compatible and is provided for customers who need to write applications that don't follow XDAIS guidelines.

Questions regarding general features of JPEG Encoder library please send to info@mecoso.com. For bug reports/integration support please use support@mecoso.com

1.2 JPEG Encoding algorithm

JPEG encoding is a method to compress images. The format has gained wide popularity with its usage in website content. JPEG works by dividing images into small blocks of size 8*8. Each block is then quantized and entropy (Huffman) coded. The compression ratio is controlled at the quantization step.

Full information about JPEG algorithm and bitstream syntax could be found in ITU-T Recommendation (standard text) T.81:
INFORMATION TECHNOLOGY – DIGITAL COMPRESSION AND CODING OF
CONTINUOUS-TONE STILL IMAGES – REQUIREMENTS AND GUIDELINES

The Independent JPEG group (<http://www.iijg.org/>) implements an excellent JPEG codec that is platform-independent and free. This codec could serve as reference software for developers working with JPEG technology.

1.3 Performance and memory requirement

Mecoso's JPEG encoder delivers high performance yet requires minimal resource. Code size is about 12KBytes. Scratch memory (L2SRAM) requirement is 10KByte (doesn't matter how large the input image is). On DM642 (600MHz), encoding a D1 resolution image (720*480) at quality setting Q=75 and YUV420 format takes around 8ms. This means 4 channels could be run in parallel on a single chip.

2 Header files

Only one header file needed to use JPEG Encoder:

- ijpegenc_mecoso.h: This header file defines the parameter structure and status structure used by the encoder.

2.1 Params structure

IJPEGENC_MECOSO_Params is defined in ijpegenc_mecoso. Parameters are described in function `jpege_setparams()`

```
typedef struct IJPEGENC_MECOSO_Params {
    int size;                // must be first field of all params structures

    // image params
    int width,height;
    int pitch;
    int format;

    // bitstream params
    int jfif;
    int restart;

    //quality params
    int quality;
    unsigned char *quant_tab;
} IJPEGENC_MECOSO_Params;
```

2.2 Status structure

IJPEGENC_MECOSO_Status is defined similar to IJPEGENC_MECOSO_Params.

3 API

JPEG Encoder functions:

- jpege_create()
- jpege_setparams()
- jpege_encode()

External functions that are called by the library:

- DMA functions provided by CSL DAT module

- DAT_wait()
- DAT_copy2d()

- Support functions provided by standard library

- divi() : integer division (/)
- remi() : integer remainder (%)

Name: `jpege_create()`

Synopsis: `void jpege_create(void *instance_mem, void *onchip_mem);`

Argument: `void *instance_mem`

`void *onchip_mem`

Return value: `void`

Description: This is the first function to call to create an instance of the jpeg encoder.
- `instance_mem` is a pointer to the (pre-allocated) memory used to keep all parameters related to this instance of the encoder. Each instance of the encoder needs a separate memory space.

- `onchip_mem` is a pointer to the scratch memory used by the encoder as DMA and processing buffer. Image data will be DMAed into this buffer, then all the encoding will happen here. For this reason we need it to be in a fast memory space (on-chip L2 memory). All encoder instances and other algorithm instances running at the same priority can and should share the same scratch memory to save precious on-chip memory resources.

- As we don't assume the direct control of memory allocation function, the memory requirement are listed here.

`instance_mem`: - persistent memory

- 4-byte aligned

- length = 600

`onchip_mem`: - scratch memory

- 8-byte aligned (or 128-byte aligned for better cache efficiency)

- length = 10240

When we don't need an encoder instance anymore, just re-use it's instance memory space for other tasks. A scratch memory space can only be released if all algorithm instances that share it don't exist anymore.

Example:

This example creates 3 separate encoder instances sharing a single scratch memory:

```
#define JPEGENC_INSTANCE_SIZE    600
#define JPEGENC_SCRATCH_SIZE    10240

// space for encoder instance
#pragma DATA_ALIGN(my_instance_mem,4)
unsigned char my_instance_mem[JPEGENC_INSTANCE_SIZE*3];

// space for srcatch memory
#pragma DATA_SECTION(my_scratch, ".ISRAM") // assume ISRAM is onchip memory
#pragma DATA_ALIGN(my_scratch,128)
unsigned char my_scratch[JPEGENC_SCRATCH_SIZE];

int i;
void *my_jpegenc[3];

for (i=0;i<3;i++)
```

```
{  
    // point my_jpegenc[i] to a memory block with enough  
    // space to hold a new instance  
    my_jpegenc[i] = (void *) (my_instance_mem + i*JPEGENC_INSTANCE_SIZE);  
  
    // create a new instance, all new instance share the  
    // same scratch memory on my_scratch  
    jpege_create(my_jpegenc[i] , (void *)my_scratch);  
}
```

Name: `jpege_setparams()`

Synopsis: `int jpege_setparams(void *encoder_instance,
IJPEGENC_MECOSO_Params *instance_params);`

Argument: `void *encoder_instance
IJPEGENC_MECOSO_Params *instance_params`

Return value: `int`

Description: This function is called after `jpege_create()` to set the parameters of an encoder instance. Parameters can be set many times in the life of an instance. For example a single encoder instance could work with several images of different sizes and formats if `setparams()` is called each time the image parameters are changed.

- `encoder_instance` points to a encoder instance that has been created before
 - `instance_params` points to an `IJPEGENC_MECOSO_Params` structure that defined the all the parameters of input image, bitstream structure and encoding quality.
- `IJPEGENC_MECOSO_Params` is defined as:

```
typedef struct IJPEGENC_MECOSO_Params {  
    int size;           // must be first field of all params structures  
  
    // image params  
    int width,height;  
    int pitch;  
    int format;  
  
    // bitstream params  
    int jfif;  
    int restart;  
  
    //quality params  
    int quality;  
    unsigned char *quant_tab;  
  
} IJPEGENC_MECOSO_Params;
```

The same param structure is used for both non-XDAIS version and XDAIS version of the library.

- `size`: must be the first field. Its value is the size of the structure.

Image parameters consist of:

- `width,height`: the size of input image.
 - Only sizes are defined here, the address of frame data is provided later on in the call to encode.
 - Values must be positive. Zero or negative values will return error code -1.
 - Values don't need to be multiple of 8 or 16. However if width value is multiple of 8 and is 4-byte aligned the DMA (to move data into onchip scratch memory) will be faster.

- `pitch`: For a typical example where we want to encode the whole image, pitch is the same as image width. In case we want to encode a sub-image inside the original one, pitch will be the width of the outer image while (`width,height`) will be the size of the sub-image. The Y pointer in this case will point to the top-left corner of this sub-image. For example:
 - encoding a D1 image: `width=720, height=480, pitch=720`
 - encoding a CIF image at the center of the above D1 image: `width=320, height=240, pitch=720` (the Y pointer now point to the upper-left corner of the sub-image)

Note: `pitch<width` is treated by encoder as `pitch=width` (this is compatible with the `pitch=0` case on the decoder side)

- `format`: There are currently 3 supported formats.
 - `format=0` → YUV400 (grey-scale). In this mode, the input image has Y component only
 - `format=1` → YUV420: input image is three non-interleaved memory blocks having Y,U (or Cr) and V (or Cr) component. Color components U and V are down-sampled twice in horizontal and vertical compared to Y.
 - `format=2` → YUV422: input image is three non-interleaved memory blocks having Y,U (or Cr) and V (or Cr) component. Color components U and V are down-sampled twice in horizontal compared to Y.
 - Permitted values are 0 (YUV400), 1 (YUV420) and 2 (YUV422).
 - Other values will return error code -2

When the format is 1 or 2 (color images) the color components will always be interleaved in the output bitstream. This makes the bitstream compatible with a wide range of applications, including digital camera standard EXIF.

Bitstream parameters consist of:

- `jfif`: signal if we should generate JFIF info in header

- `restart`: This parameter controls the inserting of restart code into the JPEG bitstream. Restart code could be inserted at arbitrary number of Minimum Code Unit (MCU).

- restart=0: no restart code is generated
- restart=N: restart code will be inserted after each group of N MCU.
- restart=M where $M > 65535$: this is a shortcut for the above case where N equals the number of MCU in each slice.

Restart code is a good feature if the JPEG bitstream is transferred through error-proned environment.

Encoding parameters consist of:

- quality: factors used to control the encoding quality. The values are from [0,100]. Other values will return an error code (negative value).
 - Value = [1, 100] set the quality from lowest (1) to highest (100). The internal quantization matrix is set in a way similar to Independent JPEG group.
 - Value = 0 is a special value that signal that we will use an external quantization matrix, the address of this external matrix is provided in the next param quant_tab. In this case, quantization values (64 bytes for luminance, followed by 64 bytes for chrominance) will be copied from this external matrix to inside of the encoder instance.
- quant_tab: pointer to user-defined quantization table. This table will be used if quality value is zero. All quantizer value must be positive (eg. from 1 to 255). Zero values will be treated as 1.

Example: This example creates an encoder instance, then set its parameter:

```
unsigned char my_jpegenc[JPEGENC_INSTANCE_SIZE];
unsigned char my_scratch[JPEGENC_SCRATCH_SIZE];
IJPEGENC_MECOSO_Params my_params;

// create a jpeg_encode instance and assign on-chip memory for it
jpeg_create(my_jpegenc,my_scratch);

// prepare the parameters for our jpeg encoder
my_params.width=720;
my_params.height=480;
my_params.pitch=0;
my_params.format=1;
my_params.jfif=1;
my_params.restart=0;
my_params.quality=75;
my_params.quant_tab=0;

// set the parameters
jpeg_setparams(my_jpegenc,&my_params);
```

Name: `jpege_encode()`

Synopsis: `int jpege_encode(void *encoder_instance, unsigned char
*image_ptr[], unsigned char *bitstream_ptr)`

Argument: `void *encoder_instance
unsigned char *image_ptr[]
unsigned char *bitstream_ptr`

Return value: `void`

Description: Encode function. Call to encode an image that have addresses stored in `image_ptr[]`, using parameters in the encoder instance. The result is written out at `bitstream_ptr`. The length of the bitstream is returned from the function.

Depending on the image format set in the parameters used to control the encoder instance, the format of the input image and `image_ptr[]` is at following:

- YUV400: greyscale encoding only. Input image has only Y component.
 - `image_ptr[0]`: point to Y component
- YUV420: color encoding with input as YUV420 format.
 - `image_ptr[0]`: point to Y component. The length is `width*height`.
 - `image_ptr[1]`: point to U component. The length is `width*height/4`
 - `image_ptr[2]`: point to V component. The length is `width*height/4`
- YUV422: color encoding with input as YUV422 format.
 - `image_ptr[0]`: point to Y component. The length is `width*height`.
 - `image_ptr[1]`: point to U component. The length is `width*height/2`
 - `image_ptr[2]`: point to V component. The length is `width*height/2`

Note that all the above formats assume luminance and chrominance as separated blocks of data. This is a non-interleaved configuration for input data. In case we need to work with interleaved input data such as YUV2 format, an external converting function could be used to convert the interleaved data into non-interleaved one. After that `jpege_encode()` is called for the newly converted data.

Example: if the width and height of input image are 720 and 480, all color components are adjacent in memory and frame data started at `0x80000000`. We will need to set the `image_ptr[]` as :

```
- YUV400: image_ptr[0] = 0x80000000;

- YUV420: image_ptr[0] = 0x80000000;
          image_ptr[1] = image_ptr[0] + 720*480 = 0x80054600;
          image_ptr[2] = image_ptr[1] + 720*480/4 = 0x80069780;

- YUV422: image_ptr[0] = 0x80000000;
          image_ptr[1] = image_ptr[0] + 720*480 = 0x80054600;
          image_ptr[2] = image_ptr[1] + 720*480/2 = 0x8007E980;
```

- DMA in, direct-write out:

This function uses DMA to transfer image data from external memory into onchip scratch memory. Encoding functions are done in scratch memory. The result will be written out directly into output bitstream buffer (external or internal). Thus before calling encoding function, we have to make sure that the input image data memory is cache-cleaned. One way to meet this requirement is to call CSL function `CACHE_clean()` for this block of memory if other processing tasks have directly accessed it before encoding.

There is no need to clean the cache for output buffer. But if after the encoding there are other processing tasks that access the output buffer by DMA these tasks will have to clean cache for themselves.

Data is DMA-in by calling `DAT` function from CSL library. Therefore applications using this jpeg encoding library has to be linked with either CSL library or one of its subset containing `DAT` module.

- Output buffer size:

The encoding function assumes that the output buffer has enough memory to store all the bitstream content of the image. No check is done about buffer overflow. Therefore the designer has to allocate enough memory to the output buffer for the maximum possible length of the bitstream.

4 Example

A simple application using JPEG Encoder library:

```

#include <std.h>
#include <tsk.h>
#include <sem.h>
#include <gio.h>
#include <sts.h>
#include <csl_dat.h>
#include <csl_cache.h>
#include <stdio.h>
#include <stdlib.h>

#include <evmdm642.h>
#include "ijpegenc_mecoso.h"

// test data (bus.yuv is a YUV420 of size 720*480) is
// defined in an assembly file
extern unsigned char bus[];
#define WIDTH 720
#define HEIGHT 480

// at very high quality, e.g. Q=100 --> output is more than 1/2 input -->
// we define the output buffer as same size as input buffer
#pragma DATA_ALIGN(output_bitstream_buffer,128)
unsigned char output_bitstream_buffer[720*480*2];

// memory for encoder instance and scratch
// my_jpegenc is allocated in SDRAM
// my_scratch is allocated in onchip L2 SRAM
#pragma DATA_ALIGN(my_jpegenc,8)
unsigned char my_jpegenc[JPEGENC_INSTANCE_SIZE];

#pragma DATA_ALIGN(my_scratch,128)
#pragma DATA_SECTION(my_scratch, ".ISRAM")
unsigned char my_scratch[JPEGENC_SCRATCH_SIZE];

void write_file(char *outname, unsigned char *outbuf, int wid, int hei)
{ FILE *outfile;
  if (!(outfile=fopen(outname,"wb")))
    exit(-1);
  fwrite(outbuf,1,wid*hei,outfile);
  fclose(outfile);
}

```

```
/*
  main() function
  - init the CSL library and DAT module
  - create a new encoder instance
  - set the parameters for the new encoder instance
  - encode a test image
  - write the result out in a file (test1.jpg)
  - set some new parameters
  - encode the same test image using the new params
  - write the result out in another file (test2.jpg)
*/

main()
{ int bitstream_length;
  unsigned char *in_ptr[3];
  IJPEGENC_MECOSO_Params my_params;

  // Open CSL and set cache mode
  CSL_init();
  CACHE_setL2Mode(CACHE_128KCACHE);
  CACHE_enableCaching(CACHE_EMIFA_CE00);
  CACHE_enableCaching(CACHE_EMIFA_CE01);
  // Clean all the cache to make sure cache-coherency in input image
  CACHE_clean(CACHE_L2ALL,0,0);

  // Init DAT module. We need DAT module for the encoder
  // to work. If DAT module is not initialized the encoder
  // won't work
  DAT_open(DAT_CHAANY, DAT_PRI_LOW, DAT_OPEN_2D);

  // Create a jpeg_encode instance and assign on-chip memory for it
  // my_jpegenc is the memory used for encoder instance
  //                                     (external, persistent)
  // my_scratch is the memory used for scratch mem
  //                                     (internal, onchip L2 RAM)
  jpeg_create(my_jpegenc,my_scratch);

  // Prepare the parameters for our jpeg encoder
  my_params.width=WIDTH;
  my_params.height=HEIGHT;
  my_params.pitch=0;
  my_params.format=1;
  my_params.jfif=1;
  my_params.restart=0;
  my_params.quality=75;           // quality = 75
  my_params.quant_tab=0;         // no user-defined quantization table

  // Set the encoding parameters to the defined params
  jpege_setparams(my_jpegenc,&my_params);

  // Clean all the cache to make sure cache-coherency in input image
  CACHE_clean(CACHE_L2ALL,0,0);
```

```
// Set the input pointers
in_ptr[0] = bus;
if (my_params.format==1)
{ in_ptr[1] = in_ptr[0] + WIDTH*HEIGHT;
  in_ptr[2] = in_ptr[1] + WIDTH*HEIGHT/4;
}
else if (my_params.format==2)
{ in_ptr[1] = in_ptr[0] + WIDTH*HEIGHT;
  in_ptr[2] = in_ptr[1] + WIDTH*HEIGHT/2;
}

// Call the encode function
bitstream_length = jpege_encode(my_jpegenc,in_ptr,output_bitstream_buffer);

// Write the result out
write_file("test1.jpg",output_bitstream_buffer,bitstream_length,1);

// Change some parameters and set them to our encoder instance
// the coding image will be the first quarter of the original image
// with a lower quality. All other parameters still have the same
// values as above
my_params.width=WIDTH/2;
my_params.height=HEIGHT/2;
my_params.pitch=WIDTH;
my_params.quality=20;
jpege_setparams(my_jpegenc,&my_params);

// Encode the image, this time with the new params. There is no need to clean
// cache for input image because we haven't touched it since the last clean time
bitstream_length = jpege_encode(my_jpegenc,in_ptr,output_bitstream_buffer);

// Write the new result out, check to see if it's the first quarter
write_file("test2.jpg",output_bitstream_buffer,bitstream_length,1);

}
```