

JPEG Decoder User's Manual

C64x platform – version 1.2
Non-XDAIS version

Mecoso Technology Inc.
Nov 2005

Contents

1 Introduction.....	3
1.1 Overview.....	3
1.2 JPEG 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.....	5
4 Example.....	11

1 Introduction

1.1 Overview

This document describes Mecoso's JPEG Decoder 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 Decoder library please send to info@mecoso.com. For bug reports/integration support please use support@mecoso.com

1.2 JPEG algorithm

JPEG coding 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.ijg.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 decoder delivers high performance yet requires minimal resource. Code size is about 10KBytes. Scratch memory (L2SRAM) requirement is 16KByte (doesn't matter how large the decoded 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 Decoder:

- `ijpegdec_mecoso.h`: This header file defines the parameter structure and status structure used by the decoder.

2.1 Params structure

`IJPEGDEC_MECOSO_Params` and `IJPEGDEC_MECOSO_Status` are defined in `ijpegdec_mecoso.h`.

```
typedef struct IJPEGDEC_MECOSO_Params {
    int size;           // must be first field of all params structures
    int pitch;         // this is used in case the image to decode lies
                    // within a larger image (pitch is the width of the
                    // larger image)
                    // setting to 0 to make decoded pitch the same
                    // as decoded width
} IJPEGDEC_MECOSO_Params;

typedef struct IJPEGDEC_MECOSO_Status {
    int size;           // must be first field of all params structures
    int width,height;  // size of the decoded image
    int pitch;         // this is used in case the image to decode lies
                    // within a larger image (pitch is the width of the
                    // larger image)

    int format;        // input and output format. Currently support:
                    // 0 --> grey-scale (YUV400); 1 --> YUV420; 2--> YUV422
                    // other values are reserved.
    int jfif;          // signal if JFIF header is present
    int restart;       // signal if restart is present and its value
} IJPEGDEC_MECOSO_Status;
```

2.2 Status structure

see above

3 API

JPEG Decoder functions:

- jpegd_create()
- jpegd_setpitch()
- jpegd_decode()

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 (%)
- memcpy() : memory block copy

Name: `jpegd_create()`

Synopsis: `void jpegd_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 decoder.

- `instance_mem` is a pointer to the (pre-allocated) memory used to keep all parameters related to this instance of the decoder. Each instance of the decoder needs a separate memory space.

- `onchip_mem` is a pointer to the scratch memory used by the decoder as DMA and processing buffer. All the decoding will happen in this buffer, then image data will be DMAed from this buffer out. For this reason we need it to be in a fast memory space (on-chip L2 memory). All decoder 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 = 384

- `onchip_mem`: - scratch memory
 - 8-byte aligned (or 128-byte aligned for better cache efficiency)
 - length = 10240

When we don't need an decoder 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 decoder instances sharing a single scratch memory:

```
#define JPEGDEC_INSTANCE_SIZE      384
#define JPEGDEC_SCRATCH_SIZE      10240

// space for decoder instance
#pragma DATA_ALIGN(my_instance_mem,4)
unsigned char my_instance_mem[JPEGDEC_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[JPEGDEC_SCRATCH_SIZE];

int i;
void *my_jpegdec[3];
```

```
for (i=0;i<3;i++)
{
    // point my_jpegdec[i] to a memory block with enough
    // space to hold a new instance
    my_jpegdec[i] = (void *) (my_instance_mem + i*JPEGDEC_INSTANCE_SIZE);

    // create a new instance, all new instance share the
    // same scratch memory on my_scratch
    jpegd_create(my_jpegdec[i] , (void *)my_scratch);
}
```

Name: `jpegd_setpitch()`

Synopsis: `void jpegd_setpitch(void *decoder_instance, int pitch);`

Argument: `void *decoder_instance`
`int pitch`

Return value: `void`

Description: This function is called after `jpegd_create()` to set the pitch for the decoder instance. Pitch is the only parameter we can set. All other parameters are extracted from the decoded image.

When a decoder instance is created, its pitch is set at zero. This will make the decoded pitch the same value as decoded width. Use this function if we want to set another value for pitch.

Example:

```
#define JPEGDEC_INSTANCE_SIZE      384
#define JPEGDEC_SCRATCH_SIZE      10240

// space for decoder instance
#pragma DATA_ALIGN(my_jpegdec,4)
unsigned char my_jpegdec[JPEGDEC_INSTANCE_SIZE];

// 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[JPEGDEC_SCRATCH_SIZE];

int new_pitch;

// create a new instance,
jpegd_create(my_jpegdec, (void *)my_scratch);

// after creation, the pitch value of my_jpegdec is zero.
// set to another value by jpegd_setpitch
new_pitch = 1024;
jpegd_setpitch( my_jpegdec, new_pitch);
```

Name: `jpege_decode()`

Synopsis:

```
int jpege_decode(void *decoder_instance, unsigned char
                *bitstream, unsigned char *image_ptr[3], int
                bitstream_length)
```

Argument:

```
void *decoder_instance
unsigned char *bitstream
unsigned char *image_ptr[3]
int bitstream_length
```

Return value: `int`

Description: Decode function. Call to decode JPEG data pointed to by `bitstream`, the length of the data is `bitstream_length`. The result is written out to image buffer that has component addresses in `image_ptr[]`. The actual length of the decoded data is returned from the function.

- `decoder_instance`: pointer to a JPEG decoder instance
- `bitstream`: pointer to the input bitstream buffer
- `image`: pointer to the (Y,U,V) components of out frame. In case of greyscaled frame, only the Y component is used
- `bitstream_size`: the size of the input JPEG bitstream. This parameter is used by the decoder to removed the 0xFF byte from the bitstream. The removing occurs in-place in the input buffer. This means we will change the buffer content. *If the input buffer is used for several tasks, user should copy JPEG bitstream into a separate buffer before calling decode() function.*
- The returning value is the number of bytes actually used by the decoder to decode the image bitstream.
-
- **Special case:**
- `bitstream_size = 0`: reserved value
- `bitstream_size = -1`: get image parameters from JPEG header.
 - If we want to extract the image parameters before calling `decode()`, we can do so by calling `decode()` with `bitstream_size=-1`. A pointer to a `IJPEGDEC_MECOSO_Status` structure will replace `image[]`. Function returns with value 0 if there is no error in header, in this case the status structure will be filled with decoded image parameters. If there is error in JPEG header, the function return non-zero value. The following example extract the image parameters and put them into `my_jpegd_status` before decode the whole image:

```
IJPEGDEC_MECOSO_Status my_jpegd_status;

// ...

// check the header if we want to (otherwise just go ahead
// with decoding)
if (jpegd_decode(my_jpegdec,input_bitstream,&my_jpegd_status,-1))
    exit(-1);          // something wrong with header

// now image params are in my_jpegd_status

// decode image, result = number of bytes in the bitstream
// used to decode the image. Negative value signals error
bitstream_length = jpegd_decode(my_jpegdec,input_bitstream,out_ptr,insize);
```

4 Example

A simple application using JPEG Decoder library:

```

/*
   This is a very simple example that shows how to use the
   JPEG decoder (non-XDAIS version).
   The example creates an decoder instance, set its
   parameters and then decode a jpeg image on the host computer.
   The output is written into file test.yuv.

   Note:

   1/ File is written out by C standard I/O function --> slow.

   2/ We can't extract the statistics in main() because statistics
       only work in a task. If we want to extract the statistics,
       e.g. how many cycles it takes to decode a frame, we have to
       run the decoder from a task and bracket it with STS_set and
       STS_delta functions. The code to do that is in tskTest.
       In this example tskTest is defined in cdb file and will run
       after main() exit. If Statistics window is opened, you'll see
       the statistics.

   Mecoso Technology Inc.
   All rights reserved.
*/

#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 "jpegdec_mecoso.h"

#define MAX_WIDTH          2048
#define MAX_HEIGHT        1024

// Simulate input and output memory
#define MAX_IMAGESIZE      (MAX_WIDTH*MAX_HEIGHT*2)
#pragma DATA_ALIGN(output_image_buffer,128)

```

```

unsigned char output_image_buffer[MAX_IMSIZE+8];
unsigned char input_bitstream[MAX_IMSIZE/2];

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

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

void error(char *err_msg)
{ exit(-1);
}
int read_file(char *iname,unsigned char *inbuf)
{ FILE *infile;
  int size=0,k;

  if (!(infile=fopen(inname,"rb"))) error("Cann't read input file");
  do
  { k=fgetc(infile);
    *inbuf++=k;
    size++;
  }
  while (k!=EOF);
  fclose(infile);

  return size-1;
}
void write_file2(unsigned char *out_ptr[3],int width,int height,int yuv420)
{ FILE *outfile;
  int colorsize;

  if (yuv420) colorsize=width*height/4;
  else colorsize=width*height/2;

  if (!(outfile=fopen("test.yuv","wb"))) error("Cann't write input file");
  fwrite(out_ptr[0],1,width*height, outfile);
  fwrite(out_ptr[1],1,colorsize,outfile);
  fwrite(out_ptr[2],1,colorsize,outfile);
  fclose(outfile);
}

// these are made global so they can be viewed from
// both main() and tskTest()
unsigned char *out_ptr[3];
int insize;

/*

```

```

main() function
- init the CSL library and DAT module
- create a new decoder instance
- set the parameters for the new decoder instance
- read a test image and decode it
- write the result out in a file (test.yuv)

*/
main()
{
    int bitstream_length;
    IJPEGDEC_MECOSO_Status my_jpegd_status;

    int result;

    // 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 decoder
    // to work. If DAT module is not initialized the decoder
    // won't work
    DAT_open(DAT_CHAANY, DAT_PRI_LOW, DAT_OPEN_2D);

    // Create a jpeg_decode instance and assign on-chip memory for it
    // my_jpegdec is the memory used for decoder instance
    //                                     (external, persistent)
    // my_scratch is the memory used for scratch mem
    //                                     (internal, onchip L2 RAM)
    jpegd_create(my_jpegdec,my_scratch);

    // set a new pitch value if we want to. Default value
    // of pitch after creating an instance is 0. This
    jpegd_setpitch(my_jpegdec,0);

    // read in a jpg test file into bitstream memory
    insize=read_file("lena_cif.jpg",input_bitstream);

    // preset the output pointers for the maximum size that the
    // output image buffer could handle
    out_ptr[0]=output_image_buffer;
    out_ptr[1]=out_ptr[0]+MAX_WIDTH*MAX_HEIGHT;
    out_ptr[2]=out_ptr[1]+MAX_WIDTH*MAX_HEIGHT/2;

    // check the header if we want to, otherwise just go ahead
    // with decoding
    if (jpegd_decode(my_jpegdec,input_bitstream,&my_jpegd_status,-1))
        exit(-1); // something wrong with header

```

```

// decode image, bitstream_length = number of bytes in the bitstream
// used to decode the image. Negative value signals error
bitstream_length = jpegd_decode(my_jpegdec,input_bitstream,out_ptr,insize);

// write the result YUV data out
if (result>0)
{
// write the output bitstream to file
if (my_jpegd_status.format==0 || my_jpegd_status.format==1)
write_file2(out_ptr,my_jpegd_status.width,my_jpegd_status.height,1);
else if (my_jpegd_status.format==2)
write_file2(out_ptr,my_jpegd_status.width,my_jpegd_status.height,0);
}
}

extern STS_Obj decodeSts;          // decode statistics defined in cdb
void tskTest()
{ int bitstream_length;

// decode and extract the time (cycles) to decode
STS_set(&decodeSts,CLK_gettime());
bitstream_length = jpegd_decode(my_jpegdec,input_bitstream,out_ptr,insize);
STS_delta(&decodeSts,CLK_gettime());
}

```