

***MPEG4 Video Simple Profile
Decoder User's Manual***

*C64 & C64+ platform – version 1.1
Non-XDAIS version*

***Mecoso Technogy Inc.
2008***

Contents

1 Introduction.....	3
2 Header files.....	5
3 API.....	7
4 Example.....	13
5 Differences between ver 1.0 and 1.1	15

1 Introduction

1.1 Overview

This document describes Mecoso's MPEG4 Simple Profile Video 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 MP4V Decoder library please send to info@mecoso.com. For bug reports/integration support please use support@mecoso.com

1.2 Performance and memory requirement

Mecoso's MP4V decoder delivers high performance yet requires minimal resource. Code size is about 40KBytes. Scratch memory (on chip SRAM) requirement is moderate (depends on the video size, for examples with D1 resolution is 12K for nodma version and 32K for DMA version). On DM642 (600MHz), decoding a D1 resolution image (720*480) at moderate quality takes around 6-7ms. This means three decoding channels could be run in parallel on a single chip. On DM6437 the performance is better and up to four channels could be run on a single chip. These performance figures are for DMA versions of the decoder.

The MP4V decoder library consists of two libraries:

- mp4vd_lib.lib: contains functions that are used only by decoder
- mp4v_common.lib: contains functions that are used by both decoder and decoder

Code size: ~ 40 KB

Data size: ~ 10 KB

1.3 Decoder libraries

The MP4V decoder library consists of 4 libraries:

C64 libraries:

- *mp4vd_a64_nonxdais_nodma.lib* nodma version
- *mp4vd_a64_nonxdais_dat.lib* dma version

C64+ libraries:

- *mp4vd_a64p_nonxdais_nodma.lib* nodma version
- *mp4vd_a64p_nonxdais_acpy3.lib* dma version

Evaluation libraries have the suffix “eval” to their names.

The reasons for four libraries is:

- Binaries compiled for C64+ core are in general more efficient than for C64 core. Therefore we should use C64+ libraries for C64+ core and C64 libraries for C64 core.
- The DMA mechanism on C64 core (EDMA2) and C64+ core (EDMA3) are different. Thus the way a C64 library make calls to DMA functions (based on CSL) are also different from the way a C64+ library make calls to its DMA functions (based on DMAN3)
- When developers have difficulty with DMA versions (e.g. codec hang when using DMAN3) they can always fall back to the nodma versions of the library, which is slower but more stable.

2 Header files

There are two header files:

- *mp4vd_nonxdais_nodma.h*
- *mp4vd_nonxdais_dma.h*

Platform	Library	Use header file
C64	<i>mp4vd_a64_nonxdais_nodma.lib</i>	<i>mp4vd_nonxdais_nodma.h</i>
C64	<i>mp4vd_a64_nonxdais_dat.lib</i>	<i>mp4vd_nonxdais_dma.h</i>
C64+	<i>mp4vd_a64p_nonxdais_nodma.lib</i>	<i>mp4vd_nonxdais_nodma.h</i>
C64+	<i>mp4vd_a64p_nonxdais_acpy3.lib</i>	<i>mp4vd_nonxdais_dma.h</i>

These two header files defined just 3 memory size parameters:

MP4VD_INSTANCE_SIZE

MP4VD_SCRATCH_SIZE

MP4VD_FRAMEBUF_SIZE

,but the value of the last two parameters are different depend on the decoded video resolution and the library we're using. The header files will calculate the required memory sizes based on the previously defined MAXWIDTH and MAXHEIGHT. A decoder instance with memory define by MAXWIDTH and MAXHEIGHT will work will all video having the same or lower resolutions. Thus these two parameters needed to be defined before reading the header file. For example:

```
#define MAXWIDTH 720
#define MAXHEIGHT 480
#include <mp4vd_nonxdais_dma.h>
unsigned char my_mp4vd[MP4VD_INSTANCE_SIZE];
```

```
unsigned char my_scratch[MP4VD_SCRATCH_SIZE];  
unsigned char decoder_framebuf[MP4VD_FRAMEBUF_SIZE];  
  
...  
  
// create a decoder instance  
mp4vd_create(&my_mp4vd,my_scratch,720,decoder_framebuf,720,sizeof(decoder_framebuf));
```

This decoder instance defined here will be able to decode all D1 (720x480) and smaller resolution video.

3 API

MP4V Decoder functions:

Initialization functions:

mp4vd_create()

mp4vd_get_dman3()

mp4vd_release_man3

for C64+ DMA version only

for C64+ DMA version only

Decoding function:

mp4vd_decode()

Name: `mp4vd_create()`

Synopsis: `int mp4vd_create(void *instance_mem,
unsigned char *scratch_buf, int max_width
unsigned char *frame_buf, int frame_buf_size)`

Argument: `void *instance_mem,
unsigned char *scratch_buf,
int max_width,
unsigned char *frame_buf,
int frame_buf_size`

Return value: `int (should be zero)`

Description: This is the first function to call to create an instance of the mp4vd 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 separated memory space.
- `scratch_buf` is a pointer to the on chip (L2 SRAM) memory used by the decoder as DMA and processing buffer. Image data will be DMAed into this buffer, then all the decoding will happen here. 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.
- `max_width`: this parameter defines the size of `scratch_buf`. For the predefined values of scratch memory in `mp4vd_params.h`, set `max_width=720`. That means this decoder will be able to decode all video streams that have resolution at D1 or lower.
- `frame_buf` is a pointer to the memory space used by the decoder to store its reconstructed frame buffers.
- `frame_buf_size` is the size of `frame_buf`. Decoder uses this value to check each time it start to decode a video stream to see if it has enough internal memory for internal reconstructed buffers. For the predefined values of `frame_buf` size in `mp4vd_params.h` (`MP4VD_FRAMEBUF_SIZE`), set `frame_buf_size = MP4VD_FRAMEBUF_SIZE`
- As we don't assume the direct control of memory allocation functions, these three memory space should be allocated by application before calling `mp4vd_create`. Their requirement sizes are defined in `mp4vd_params.h`

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 run anymore.

Example:

This example creates 3 separated decoder instances. Each one has its own instance memory and frame buffer. All of them share a single scratch memory:

```
#include mp4vd_params.h

// space for decoder instance
unsigned char mp4vd_instance[3][MP4VD_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[MP4VD_SCRATCH_SIZE];

// space for frame buffers
unsigned char mp4vd_framebuf[3][MP4VD_FRAMEBUF_SIZE];

...

// create 3 decoder instances, all share the
// same scratch memory on my_scratch

// first instance can decode all video streams D1 or lower
mp4vd_create(&mp4vd_instance[0][0], &my_scratch, 720,
            &mp4vd_framebuf[0][0], MP4VD_FRAMEBUF_SIZE);

// second instance can decode all video streams D1 or lower
mp4vd_create(&mp4vd_instance[1][0], &my_scratch, 720,
            &mp4vd_framebuf[1][0], MP4VD_FRAMEBUF_SIZE);

// third instance can decode all video streams D1 or lower
mp4vd_create(&mp4vd_instance[2][0], &my_scratch, 720,
            &mp4vd_framebuf[2][0], MP4VD_FRAMEBUF_SIZE);

...
```

Name: `mp4vd_get_dman3()`

Synopsis: `int mp4vd_get_dman3(void *instance_mem)`

Argument: `void *instance_mem`

Return value: `int (should be zero)`

Description:

This function is only used for *mp4vd_a64p_nonxdais_acpy3.lib*.

The C64+ DMA version (*mp4vd_a64p_nonxdais_acpy3.lib*) needs to access and use the DMA resource. We will use DMAN3 and ACPY3 for this purpose.

After the decoder instance is initialized, this function is called to allocate DMAN3 resources to the decoder. Later on when `mp4vd_decode()` is called, the decoder instance will use these DMAN3 resources through calls to ACPY3 library.

If the return value is non-zero there is an error in the allocation of DMAN3 resources (not enough memory or not enough free EDMA3 channels)

When we don't use a decoder instance any more, the DMAN3 resources allocated to it should be returned to the DMA manager, this is done by calling `mp4vd_release_dman3()`

Name: `mp4vd_release_dman3()`

Synopsis: `int mp4vd_release_dman3(void *instance_mem)`

Argument: `void * instance_mem`

Return value: `int (should be zero)`

Note that the C64 DMA version (*mp4vd_a64_nonxdais_dat.lib*) also use DMA resources but this is done though CSL DAT module. We just need to have DAT module initialized at the beginning of the program and there is no need for a function equivalent to `mp4vd_get_dman3()` for *mp4vd_a64_nonxdais_dat.lib*

Name: `mp4vd_decode()`

Synopsis:

```
int mp4vd_decode(void *decoder_instance,
unsigned char *bitstream_buf,
unsigned char *outframe_ptr[3],
mp4vd_params *decoded_params );
```

Argument:

```
void *decoder_instance,
unsigned char *bitstream_buf
unsigned char *outframe_ptr[3]
mp4vd_params *decoded_params
```

Return value: `int` (zero if OK, non-zero --> error)

Description: Decode function. Call to decode a frame that have addresses stored in `frame_ptr[]`, using parameters in `decode_params`. The result is written out at `bitstream_buf`. The length of the writing bitstream is returned from the function.

- decoder instance: one decoder instance that has been created using `mp4vd_create()`
- `bitstream_buf`: input bitstream buffer
- `outframe_ptr[]` is a set of 3 pointers to output, decoded frame buffer
 - `outframe_ptr[0]` is Y pointer
 - `outframe_ptr[1]` is U pointer
 - `outframe_ptr[2]` is V pointer
- `decode_params`: this structure will be filled by the decoder
 - width, height: videodecoded size
 - `pic_type`: decoded picture type, 0 is I-frame, 1 is P-frame
 - `bitstream_length`: the number of bytes used to decode this frame
 - `error_flag`: same as return value of `mp4vd_decode()`, should be zero for error-free.

Notes:

- This function uses DMA to transfer video (frame) data from onchip scratch memory to external frame buffers. Thus before calling decoding function, we have to make sure that the output frame buffers are cache-cleaned. One way to meet this requirement is to call CSL function `CACHE_clean()` (for C64 version) or `BCACHE_wbInV()` (for C64+ version) for this block of memory if other processing tasks have directly accessed it before decoding.
- There is no need to clean the cache for input bitstream buffer. It is read directly by the decoder without using DMA.

- In C64 dma version (*mp4vd_a64_nonxdais_dat.lib*) , data is dma-out by calling DAT functions from CSL library. Therefore applications using this decoding library has to be linked with either CSL library or one of its subset containing DAT module and the DAT module needs to be initialized by calling DAT_open().
- In C64+ dma version (*mp4vd_a64p_nonxdais_acpy3.lib*), data is dma-out by calling ACPY3 functions that work in the framework of DMAN3 manager. Therefore we need to call mp4vd_get_dman3() after each mp4vd_create() to allocate the necessary DMAN3 resources for the newly created instance

4 Example

Examples in the evaluation suite are in two groups:

- Examples for C64 libraries:

These examples run on DM642EVM. The code is good for ver1 and ver2 of the EVM. For ver3 of the EVM (TI video encoder chip) you can replace the video header, video parameter files and the vport library and recompile. All examples are built under CCS 2.21 for maximum compatibility.

- Examples for C64+ libraries:

These examples run on DM6437EVM. All examples are built under CCS 3.3.

Both set of examples for C64 and C64+ have the same structure and (equivalent) files like following:

- <i>doc</i>	documentation
- <i>examples</i>	
<i>common</i>	common files used for examples
<i>mp4vd_nonxdais_test</i>	simple project to test the functionality of the decoding library
<i>mp4v_nonxdais_loopback</i>	simple project to test the encoder working together with decoder
- <i>inc</i>	header files
- <i>lib</i>	library files

Evaluation suite content:

We will describe the files in the evaluation suite example for C64 core (running on DM642EVM) here but the same is true for the files in the C64+ suite examples.

4.1/ mp4vd_a64_nonxdais_test:

In this folder, there are two projects:

- *mp4vd_a64_nonxdais_dat.prj* test using dma library
- *mp4vd_a64_nonxdais_nodma.prj* test using nodma library

They are nearly the same with the exception of linking to different headers and libraries. Each project reads in a bitstream file (test.m4v), decode and write the results out as test1000.yuv, test1001.yuv, Each is a raw YUV420 file with the size as (width*height*3/2). An image viewer that can view YUV420 raw file can be used to view these file. For example the free viewer Xnview at <http://www.xnview.com>

4.2/ mp4v_a64_nonxdais_loopback:

In this folder there are four projects:

- *mp4v_a64_loopback_vbr_nodma.prj* loopback test in constant Q, variable bit rate mode, using nodma library
- *mp4v_a64_loopback_vbr_dat.prj* loopback test in constant Q, variable bit rate mode, using dma library
- *mp4v_a64_loopback_cbr_nodma.prj* loopback test in constant bit rate mode, using nodma library
- *mp4v_a64_loopback_cbr_dat.prj* loopback test in constant bit rate mode, using dma library

The constant bit rate example is built based on a simple bit allocation algorithm (*rate_control.c*). You can design and test your own rate control algorithm here.

In each example, after compiling and loading, we should open the following window:

- Dsp/Bios Statistics View shows the related statistics, eg. encoder and decoder cycles
- Dsp/Bios Cpu Load Graph shows Cpu load
- Dsp/Bios Message Log shows realtime bitrate in "trace"

For the vbr examples also open View Watch window. Watch and change these variables and see the effects:

- *cur_gop* change gop size (any value ≥ 1), *gop=1* means only I-frame
- *cur_quant* change Q (1-31)
- *cur_speed* change speed (0-5)

5. Differences between version 1.0 and 1.1

5.1/ Number of libraries

In version 1.0, the decoder is supplied as two libraries:

- *mp4vd_lib.lib* contains functions that are used only by decoder
- *mp4v_common.lib* contains functions that are used by both encoder and decode

In version 1.1, all the common parts are included in the single decoder library, but instead of one library for C64, there are 4 libraries for C64 and C64+. Each platform (C64 and C64+) has two libraries: nodma-based version and dma-based version. Dma-based versions are usually faster than non-dma based versions. C64+ version are often faster (more efficient code) than C64 version.

C64 libraries:

- *mp4vd_a64_nonxdais_nodma.lib* nodma version
- *mp4vd_a64_nonxdais_dat.lib* dma version

C64+ libraries:

- *mp4vd_a64p_nonxdais_nodma.lib* nodma version
- *mp4vd_a64p_nonxdais_acpy3.lib* dma version

5.2/ The scratch memory size has been reduced significantly. This allows the decoder to work with larger resolutions that weren't supported in version 1.0. For example, D1 resolution requires about 90KB of scratch memory in version 1.0. In version 1.1 it's 12KB (nodma version) and 32K (dma version).