Codec Memory Management(CMM) API Specification (Linux)

S3C6400/6410

August 26, 2008

REV 1.11

Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. Samsung assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained herein.

Samsung reserves the right to make changes in its products or product specifications with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

This publication does not convey to a purchaser of semiconductor devices described herein any license under the patent rights of Samsung or others.

Samsung makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Samsung assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

"Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by the customer's technical experts.

Samsung products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, for other applications intended to support or sustain life, or for any other application in which the failure of the Samsung product could create a situation where personal injury or death may occur.

Should the Buyer purchase or use a Samsung product for any such unintended or unauthorized application, the Buyer shall indemnify and hold Samsung and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that Samsung was negligent regarding the design or manufacture of said product

S3C6400/6410 RISC Microprocessor
CMM API specification

Samsung Electronics Co., Ltd.
San #24 Nongseo-Dong, Giheung-Gu
Yongin-City Gyeonggi-Do, Korea
446-711

Home Page: http://www.samsungsemi.com/

E-Mail: mobilesol.cs@samsung.com

Printed in the Republic of Korea

Revision History

Revision No	Description of Change	Refer to	Author(s)	Date
1.00	Initial Version	-	Jiun Yu	2008-07-05
1.10	Free API is added		Jiun Yu	2008-07-19
1.11	Cache clean, invalidatation are added		Jiun Yu	2008-08-26

Contents

Figures

1 Introduction

1.1 Purpose

The purpose of the document is to describe the CMM API for easy portability into different platforms by developers.

1.2 Scope

The scope of this document is to describe

- Software architecture of CMM
- Usage of CMM API
- How to use CMM driver

1.3 Intended Audience

Intended Audience	Tick whenever Applicable
Project Manager	Yes
Project Leader	Yes
Project Team Member	Yes
Test Engineer	Yes

1.4 Definitions, Acronyms, and Abbreviations

Abbreviations	Description
CMM	Codec Memory Management
API	Application Program Interface

1.5 References

Number	Reference	Description
1	SMDK6400_WinCE6.0_FMD_PortingGuide.doc	OS porting guide
2	SMDK6400_WinCE6.0_VideoDriver_UserManual.doc	Video Driver specification
3	S3C6400_6410_Linux2.6.21_CMM_API_REV1.00_20080705.doc	Cmm v1.00 api document

2 Software Architecture

2.1 Overview

When multimedia player use s/w decoder, Performance problem is often issued. CMM(Codec Memory Management) driver helps to improve rendering performance.

In common multimedia player, Decoded YUV data is transferred to video memory using memcpy(). It decreases much performance when the resolution of movie is large.

CMM Driver provides the interface to transfer decoded YUV data to video memory directly. At first, It allocates virtual address to the player. The virtual address is surely cacheable area. So, s/w decoder can utilize cache. After decoding, the player request for CMM to flush cached area. And then, the player request physical address of YUV buffer to CMM. With the physical address, the player calls video driver API for rendering. YUV data is transferred to h/w post processor by DMA.

It does not only reduce memcpy() time, but also make player to decode and render at the same time. Because rendering is done by only h/w, decoding performance is not decreased. You should make decoding and rendering as multi-threaded.

There are 2 methods to render YUV data. The one is using local path between h/w post processor and LCD. It doesn't posses data BUS. But local path only supports RGB888. The other is using DMA between post processor and LCD.

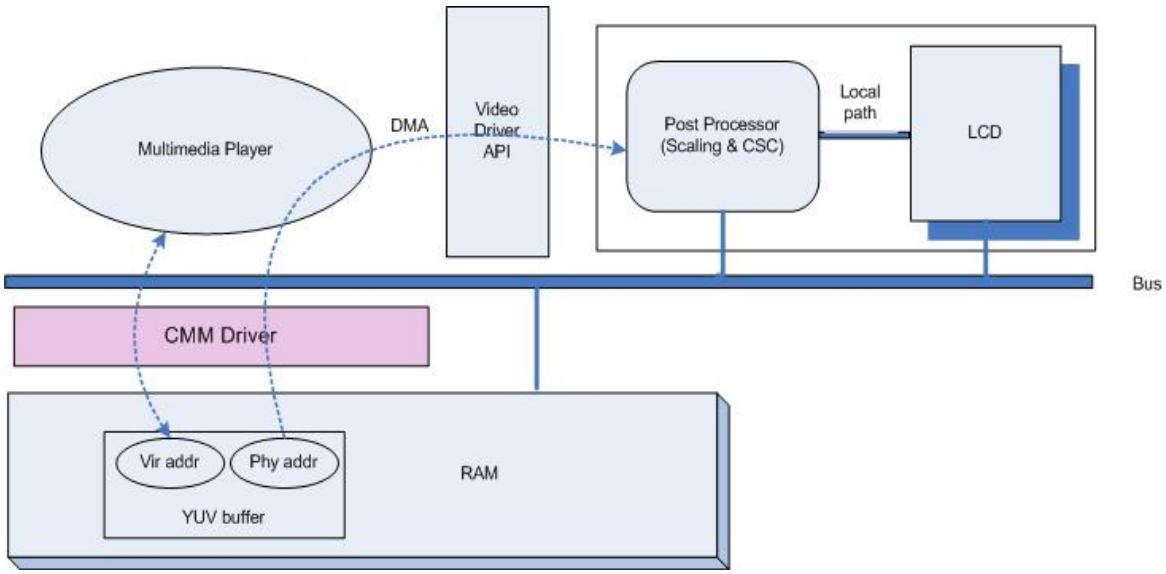


Figure . Architecture of CMM API

3 API

3.1 Linux Device Driver's File I/O Operation

API Function	Description
open	Open CMM driver
ioctl	IOCTL_CODEC_MEM_ALLOC IOCTL_CODEC_MEM_FREE IOCTL_CODEC_CACHE_FLUSH IOCTL_CODEC_GET_PHY_ADDR IOCTL_CODEC_MERGE_FRAGMENTATION
mmap	Mapping reserved memory for application
close	Close CMM driver

3.1.1 open

open	
Syntax	Int open(const char *path, int oflag);
Description	This function opens cmm driver
Parameters	path[IN] : path of the CMM device driver's node oflag[IN] : flags of CMM driver
Returns	File descriptor of CMM driver

3.1.2 ioctl

ioctl	
Syntax	Int ioctl(int fd, int cmd, int arg)
Description	Most of functions are developed in ioctl. This system call has many functions which is seperated by cmd
Parameters	fd[IN] : file descriptor returned by open() function cmd[IN] : Control codes for the operation. Detailed information of cmd will explain below. Arg[IN] : Structure of the CMM arguments
Returns	If the operation completes successfully, the return value is nonzero. If the operation fails or is pending, the return value is zero.

3.1.3 mmap

mmap	
Syntax	void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t off);
Description	This function maps physically continuous memory. This memory can share user application and device driver.
Parameters	addr[IN] : none len[IN] : mapped memory size prot[IN] : memory access permission(PROT_READ, PROT_WRITE, etc) flag[IN] : attribute of memory (MAP_SHARED, etc) fd [IN] : descriptor of CMM driver off[IN] : - 0 : mapping cacheable memory - CODEC_CACHED_MEM_SIZE : mapping non-cacheable memory
Returns	Base address of codec memory. This address can use in user application

3.1.4 close

close	
Syntax	Int close(int fd)
Description	Close CMM's file descriptor
Parameters	fd[IN] : file descriptor of CMM driver
Returns	If the function succeeds, the return value is nonzero If the function fails, the return value is zero.

3.2 Command of CMM's ioctl

IOCTL_CODEC_MEM_ALLOC	
Syntax	See 3.1.2
Description	It allocates memory
Parameters	arg[IN/OUT] : Pointer to CODEC_MEM_ALLOC_ARG structure
Returns	If the operation completes successfully, the return value is nonzero. If the operation fails or is pending, the return value is zero.

IOCTL_CODEC_MEM_FREE	
Syntax	See 3.1.2
Description	It frees memory
Parameters	arg[IN/OUT] : Pointer to CODEC_MEM_FREE_ARG structure
Returns	If the operation completes successfully, the return value is nonzero. If the operation fails or is pending, the return value is zero.

IOCTL_CODEC_CACHE_FLUSH	
Syntax	See 3.1.2
Description	It flush cached area using virtual address
Parameters	arg[IN/OUT] : Pointer to CODEC_CACHE_FLUSH_ARG structure
Returns	If the operation completes successfully, the return value is nonzero. If the operation fails or is pending, the return value is zero.

IOCTL_CODEC_GET_PHY_ADDR	
Syntax	See 3.1.2
Description	It returns physical address mapped to virtual address
Parameters	arg[IN/OUT] : Pointer to CODEC_ GET_PHY_ADDR_ARG structure
Returns	If the operation completes successfully, the return value is nonzero. If the operation fails or is pending, the return value is zero.

IOCTL_CODEC_MERGE_FRAGMENTATION	
Syntax	See 3.1.2
Description	It merges fragment memory.
Parameters	None
Returns	If the operation completes successfully, the return value is nonzero. If the operation fails or is pending, the return value is zero.

IOCTL_CODEC_CACHE_INVALIDATE	
Syntax	See 3.1.2
Description	It invalidates cache
Parameters	arg[IN/OUT] : Pointer to CODEC_CACHE_FLUSH_ARG structure
Returns	If the operation completes successfully, the return value is nonzero. If the operation fails or is pending, the return value is zero.

IOCTL_CODEC_CACHE_CLEAN

Syntax	See 3.1.2
Description	It writes cache data into memory
Parameters	arg[IN/OUT] : Pointer to CODEC_CACHE_FLUSH_ARG structure
Returns	If the operation completes successfully, the return value is nonzero. If the operation fails or is pending, the return value is zero.

IOCTL_CODEC_CACHE_CLEAN_INVALIDATE	
Syntax	See 3.1.2
Description	It invalidates cache and writes cache data into memory
Parameters	arg[IN/OUT] : Pointer to CODEC_CACHE_FLUSH_ARG structure
Returns	If the operation completes successfully, the return value is nonzero. If the operation fails or is pending, the return value is zero.

3.3 Data Structure for Passing the IOCTL Arguments

3.3.1 CODEC_MEM_ALLOC_ARG

CODEC_MEM_GET_ARG	
char cacheFlag	[IN] 1 : allocating cacheable memory 0 : allocating non-cacheable memory
int buffsize	[IN] buffer size that user wants to allocate
unsigned int cached_mapped_addr	[IN] mapped cacheable address returned by mmap() function
unsigned int non_cached_mapped_addr	[IN] mapped non-cacheable address returned by mmap() function
unsigned int out_addr	[OUT] Virtual address of allocated memory

3.3.2 CODEC_MEM_FREE_ ARG

CODEC_ MEM_FREE_ARG	
unsigned int u_addr	[IN] virtual address returned by CODEC_MEM_ALLOC ioctl

3.3.3 CODEC_CACHE_FLUSH_ARG

CODEC_CACHE_FLUSH_ARG	
unsigned int u_addr	[IN] virtual address returned by CODEC_MEM_ALLOC ioctl
int size	[IN] flush size

3.3.4 CODEC_GET_PHY_ADDR_ARG

CODEC_ GET_PHY_ADDR_ARG	
`unsigned int u_addr`	[IN] virtual address returned by CODEC_MEM_ALLOC ioctl
`unsigned int p_addr`	[OUT] It returns physical address of user_addr

4 Scenario using CMM driver

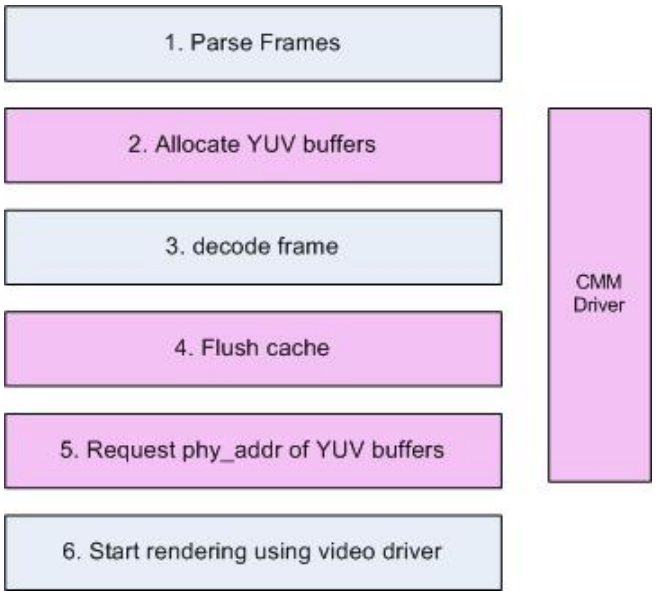


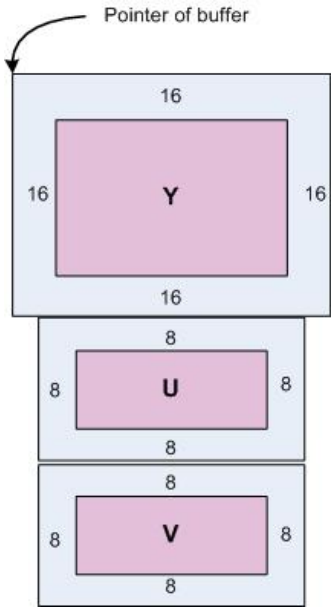
Figure . Simple Decoding Scenario using CMM

1. s/w decoder request YUV buffers using IOCTL_CODEC_MEM_GET. Normally, you may need 3 buffers for YUV. Maximum number of buffer is defined in MAX_BUFFER_NUM at CMMDriver.h For codec memory, It shares RAM area with MFC H/W by default. So, you can't use H/W MFC and CMM at the same time. If you want to change address, you may change CODEC_MEM_START
2. s/w decoder do decoding.
3. s/w decoder request to flush YUV buffer using IOCTL_CODEC_CACHE_FLUSH.
4. To render YUV data, multimedia player request physical address of YUV buffer.
5. using video driver API, rendering can be started.

5 How to use CMM driver

5.1 Padded Buffer

CMM also supports padded buffer like below picture. Both virtual addr and physical addr points the start of buffer including padding. After decoding, you should set padding size in video driver API. In the case of below picture, offset is 16.




```
pp_param.SrcFullWidth = IMG_WIDTH + 2*16;
pp_param.SrcFullHeight = IMG_HEIGHT + 2*16;
pp_param.SrcStartX = 16;
pp_param.SrcStartY = 16;
pp_param.SrcWidth  = pp_param.SrcFullWidth - 2*pp_param.SrcStartX;
pp_param.SrcHeight = pp_param.SrcFullHeight -
          2*pp_param.SrcStartY;
pp_param.SrcCSpace                  = YC420;
pp_param.DstFullWidth = 800;        // destination width
pp_param.DstFullHeight = 480;       // destination height
pp_param.DstWidth = 800;
pp_param.DstHeight = 480;
pp_param.DstCSpace = RGB16;
pp_param.OutPath  = POST_DMA;
pp_param.Mode = ONE_SHOT;
```

5.2 Test code

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <ctype.h>
#include <unistd.h>
#include <sys/mman.h>
#include <string.h>
#include <errno.h>
#include <sys/time.h>
#include <signal.h>
#include <pthread.h>

#include "../cmm_drv/s3c-cmm.h"
#define CMM_DRIVER_NAME             "/dev/misc/s3c-cmm"


int main()
{
        int                     cmm_fd;
        int                     ret;
        unsigned char   *cached_addr, *non_cached_addr;

        CODEC_MEM_ALLOC_ARG             codec_mem_alloc_arg;
        CODEC_MEM_FREE_ARG          codec_mem_free_arg;
```

```
        CODEC_CACHE_FLUSH_ARG       codec_cache_flush_arg;
        CODEC_GET_PHY_ADDR_ARG      codec_get_phy_addr_arg;


        /* Open CMM(Codec Memory Management) driver which supports multi-instances*/
        cmm_fd = open(CMM_DRIVER_NAME, O_RDWR | O_NDELAY);
        if(cmm_fd < 0) {
                printf("CMM driver open error\n");
                return -1;
        }


        /* Mapping cacheable memory area */
        cached_addr = (unsigned char *)mmap(0, CODEC_CACHED_MEM_SIZE,                   \
                PROT_READ | PROT_WRITE, MAP_SHARED, cmm_fd, 0);
        if(cached_addr == NULL) {
                printf("CMM driver buffer mapping failed\n");
                return -1;
        }


        /* Mapping non-cacheable memory area */
        non_cached_addr = (unsigned char *)mmap(0, CODEC_NON_CACHED_MEM_SIZE,           \
                PROT_READ | PROT_WRITE, MAP_SHARED, cmm_fd, CODEC_CACHED_MEM_SIZE);
        if(non_cached_addr == NULL) {
                printf("CMM driver buffer mapping failed\n");
                return -1;
        }
        printf("[USER] Cached address : 0x%X, Non-cached address : 0x%X\n",     \
                (unsigned int)cached_addr, (unsigned int)non_cached_addr);


        /* Request memory allocation */
        codec_mem_alloc_arg.buffSize = 1*1024;
        codec_mem_alloc_arg.cached_mapped_addr = (unsigned int)cached_addr;
        codec_mem_alloc_arg.non_cached_mapped_addr = (unsigned int)non_cached_addr;
        codec_mem_alloc_arg.cacheFlag = 1;    /* 1: cacheable, 0: non-cacheable */
        ret = ioctl(cmm_fd, IOCTL_CODEC_MEM_ALLOC, &codec_mem_alloc_arg);
        if (ret == 0) {
                printf("Memory allocation failed, ret = %d\n", ret);
                return -1;
        }
        printf("Allocated memory address : 0x%X, size : %d, cached flag : %d\n",        \
                codec_mem_alloc_arg.out_addr, codec_mem_alloc_arg.buffSize,
codec_mem_alloc_arg.cacheFlag);


        /* Clean the cacheable memory area */
        codec_cache_flush_arg.u_addr = codec_mem_alloc_arg.out_addr;
        codec_cache_flush_arg.size = codec_mem_alloc_arg.buffSize;
        ioctl(cmm_fd, IOCTL_CODEC_CACHE_FLUSH, &codec_cache_flush_arg);


        /* Get physical address */
        codec_get_phy_addr_arg.u_addr = codec_mem_alloc_arg.out_addr;
        ioctl(cmm_fd, IOCTL_CODEC_GET_PHY_ADDR, &codec_get_phy_addr_arg);
        printf("User address : 0x%X, --> Physical address : 0x%X\n",       \
                codec_get_phy_addr_arg.u_addr, codec_get_phy_addr_arg.p_addr);
```

```
        /* Free memory */
        codec_mem_free_arg.u_addr = codec_mem_alloc_arg.out_addr;
        ret = ioctl(cmm_fd, IOCTL_CODEC_MEM_FREE, &codec_mem_free_arg);


        close(cmm_fd);

        return 0;
}
```