MMC and SD Media Driver for MPC5xxx User Guide

Version 1.20

For use with MMC and SD Media Driver for MPC5xxx versions 1.01 and above

Date: 18-Aug-2017 15:07

Table of Contents

1 System Overview

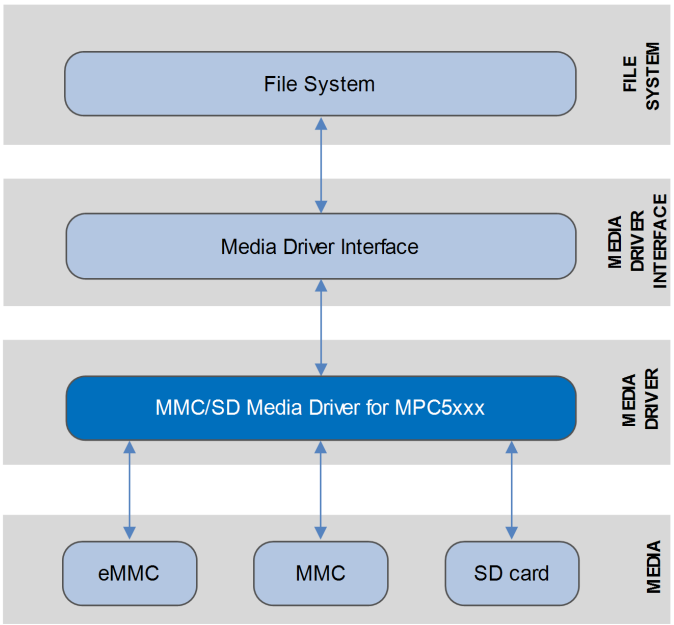
1.1 Introduction

This guide is for those who want to use HCC Embedded's MMC/SD Media Driver for MPC5xxx devices in their system. This module provides a media driver for MPC5xxx microcontrollers. This guide covers all aspects of configuration and use.

This media driver conforms to the *HCC Media Driver Interface Specification*. It provides an interface for a file system to read from and write to Secure Digital (SD), MultiMediaCard (MMC), or eMMC (embedded MMC) storage devices. A single media driver can support one or more physical media, each of these being represented as a different drive at the media driver interface. The file system handles all drives identically, regardless of their internal design features.

If eMMC is used, this media driver can be used with HCC's eMMC Management Driver. This is an extension to HCC's MMC and SD media drivers and is independent of any particular micro-controller and its MMC/SD controller. For details, see the *HCC eMMC Management Extension for MMC and SD Drivers User Guide*.

The diagram below shows a typical system architecture including a file system, media driver and media.



1.2 Feature Check

The main features of the media driver are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Designed for integration with both RTOS and non-RTOS based systems.
- Conforms to the *HCC Media Driver Interface Specification*.
- Supports multiple card types: MMC and SD and also SDHC (Secure Digital High Capacity) and SDXC (Secure Digital eXtended Capacity).
- Supports eMMC (embedded MMC) and can be used with HCC's eMMC Management Driver.
- Supports MPC5xxx devices.
- Supports Direct Memory Access (DMA) transfers.
- Supports 1 bit and 4 bit transfer width.

1.3 Packages and Documents

Packages

The table below lists the packages that you need in order to use this module:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
media_drv_base	The base media driver package that includes the framework all media drivers use.
media_drv_mmcsd_mpc5x	The media driver package described in this document.
oal_base	The base OS Abstraction Layer (OAL) package.

Documents

For an overview of HCC file systems and data storage, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC Media Driver Interface Guide

This document describes the HCC Media Driver Interface Specification.

HCC eMMC Management Extension for MMC and SD Drivers User Guide

This document describes HCC's embedded MMC extension.

HCC MMC and SD Media Driver for MPC5xxx User Guide

This is this document.

1.4 Change History

This section describes past changes to this manual.

- To view or download earlier manuals, see Archive: MMC and SD Media Driver for MPC5xxx User Guide.
- For the history of changes made to the package code itself, see History: media_drv_mmcsd_mpc5x.

The current version of this manual is 1.20. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.20	2017-08-18	1.01	Updated *Packages* list.
1.10	2017-06-23	1.01	New *Change History* format.
1.00	2017-04-25	1.01	First online version.

2 Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

> **Note:** Do not modify any files except the configuration file.

2.1 API Header File

The file **src/api/api_mdriver_mpc5x.h** is the only file that should be included by an application using this module. For details of the single API function, see Application Programming Interface.

2.2 Configuration File

The file **src/config/config_mdriver_mmcsd.h** contains all the configurable parameters of the system. Configure these as required. This is the only file in the module that you should modify. For details of these options, see Configuration Options.

2.3 System Files

The following files in the directory **src/media-drv/mmcsd/ppc5121** are the source code for the media driver. **These files should only be modified by HCC**.

File	Description
mmcsd.c	MMC/SD source file.
mmcsd.h	MMC/SD initialization header file.
mmcsd_dsc.h	*t_mmcsd_dsc* typedef header file.

2.4 Version Files

The file **src/version/ver_mdriver_mmcsd.h** contains the version number of the module. The version number is checked by all modules that use a module to ensure system consistency over upgrades.

3 Configuration Options

Set the system configuration options in the file **src/config/config_mdriver_mmcsd.h**. This section lists the available options and their default values.

REF_CLK

The system oscillator. The default is 33333333.

SDHC_SPEED_LIMIT_READ

The read speed limit. The default is 0, giving a value of 25000.

SDHC_SPEED_LIMIT_WRITE

The write speed limit. The default is 0, giving a value of 10000.

FLUSH_DATACACHE

Keep the default of 1 to use data cache flushing. Set it to 0 to disable this.

SECTOR_BUFFER_SIZE

The number of 512 byte sectors the temporary buffer can hold. See the full description below. The default is 8.

SD_ALLOW_4BIT

To allow 4 bit mode, keep the default of 1. On a Silicontkx board DAT2 is not connected so 4 bit is not supported. For this board set the value to 0.

RETRY_COUNT

The retry count. The default is 3.

3.1 SECTOR_BUFFER_SIZE

SECTOR_BUFFER_SIZE defines the number of 512 byte sectors the temporary buffer can hold. Keep this number as high as possible because then multiple sector read/write can be efficiently used.

The buffer is required for two reasons: byte order and caching.

Byte order

The SDHC module in PPC5121 reads/writes data in 32 bit chunks, always in a reversed order. This means that reading data byte by byte gets the data in the wrong order. Therefore an intermediate buffer is used where data is DMA-ed from/to the DBA register in reversed order (memory access size is 8 bits, DMA is started from the end of the buffer and steps backward through the memory).

In this way 32 bit chunks are obtained in the right order and data just needs to be copied there/back in 32 bit chunks from/to the end of the intermediate buffer to/from the user buffer. This means swapping of byte order is avoided and this improves performance.

Caching

DMA accesses the memory and not the data cache, so cache flushing is required before/after writing /reading. The data cache is organized in 8 word blocks but the user buffer might not be aligned to 8 words, so while performing the transfer the system may re-cache the beginning or the end of the user buffer. This can lead to accessing of incorrect data.

In order to get the system running with data cache enabled, there are two options:

- Put the *sector_buffer* array in an uncached area (using MMU).
- Set FLUSH_DATACACHE to 1. This way *sector_buffer* is flushed when needed (setting this option makes use of the *_flush_data_cache* function defined in **mmcsd.c**).

4 Application Programming Interface

This section describes the single function, the structure it uses, and the error codes.

When the media driver is used:

1. The file system calls the media driver's **mmcsd_initfunc()** function.
2. **mmcsd_initfunc()** returns a pointer to an F_DRIVER structure containing a set of functions for accessing the media driver.

4.1 mmcsd_initfunc

Use this function to initialize the interface with the driver.

The caller passes a parameter to the initialization function of a conforming driver. The driver returns a pointer to an F_DRIVER structure defining the interface to that driver.

> **Note:** The call must allocate or use a static structure for the F_DRIVER structure. It must return a pointer to this structure, which must contain all the driver entry points, and also other data as required.

Format

```
F_DRIVER * ( mmcsd_initfunc )( unsigned long driver_param )
```

Arguments

Argument	Description	Type
driver_param	This identifies the drive to use. The first drive is 0. This cannot be greater than MDRIVER_MAX_VOLUME - 1.	unsigned long

Return values

Return value	Description
F_DRIVER *	A pointer to the driver structure, or NULL if the request failed.

4.2 F_DRIVER Structure

This is the format of the *F_DRIVER* structure. This structure is defined in the *HCC Media Driver Interface Specification*.

Element	Type	Description
separated	int	Non-zero if the driver is separated.
user_data	unsigned long	User-defined data.
user_ptr	void *	User-defined pointer.
writesector	F_WRITESECTOR	Write a sector to the drive. This is mandatory if format or any write access is required.
writemultiplesector	F_WRITEMULTIPLESECTOR	Write a series of sectors to the drive. If this is unavailable F_WRITESECTOR may be used.
readsector	F_READSECTOR	Read a sector from the drive.
readmultiplesector	F_READMULTIPLESECTOR	Read a series of sectors from the drive. If this is unavailable F_READSECTOR may be used.
getphy	F_GETPHY	Used to get the physical properties of the drive, such as the number of sectors.
getstatus	F_GETSTATUS	(Only for removable drives) Used to test whether a drive has been removed or changed.
release	F_RELEASE	Release any resources associated with a drive when it is freed by the host (file) system.
ioctl	F_IOCTL	Used to send user-defined messages to the driver and get a response.

4.3 Error Codes

If a function executes successfully, it returns with MMCSD_NO_ERROR, a value of zero. The following table shows the meaning of the error codes.

Return Value	Value	Description
MMCSD_ERR_NOTPLUGGED	-1	For high level.
MMCSD_NO_ERROR	0	Successful execution.
MMCSD_ERR_NOTINITIALIZED	101	Driver not initialized.
MMCSD_ERR_INIT	102	Initialization error.
MMCSD_ERR_CMD	103	Command error.
MMCSD_ERR_STARTBIT	104	Start bit error.
MMCSD_ERR_BUSY	105	Driver already active.
MMCSD_ERR_CRC	106	CRC error.
MMCSD_ERR_WRITE	107	Write error.
MMCSD_ERR_WRITEPROTECT	108	Media is write-protected.
MMCSD_ERR_DATAOK	109	
MMCSD_ERR_RX	110	Receive error.
MMCSD_ERR_NOTAVAILABLE	111	Media is not available.

5 Integration

This section describes all aspects of the module that require integration with your target project. This includes porting and configuration of external resources.

5.1 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer.

The media driver makes use of the following standard PSP function:

Function	Package	Component	Description
psp_memset()	psp_base	psp_string	Sets the specified area of memory to the defined value.