MMC and SD Media Driver for Atmel SAM User Guide

Version 1.60

For use with MMC and SD Media Driver for Atmel® SAM versions 2.03 and above

Date: 18-Aug-2017 15:15

Table of Contents

1 System Overview

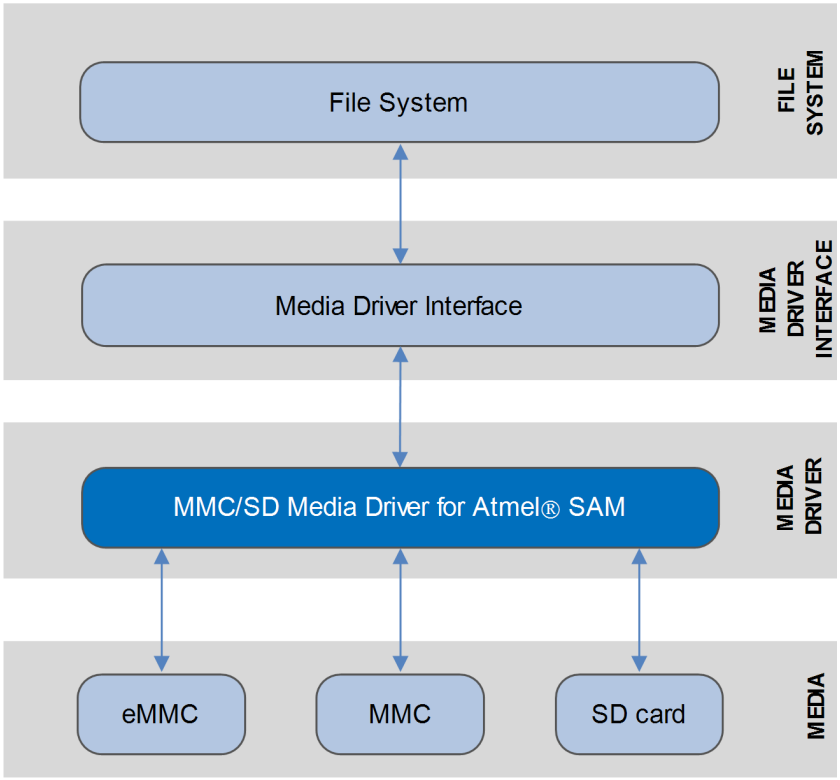
1.1 Introduction

This guide is for those who want to use HCC Embedded's MMC and SD Media Driver for the Atmel® SAM range of micro-controllers. This supports the Atmel SAM9G, SAM3S, SAM4E, and SAM4S families. These use the Atmel® SDIO Secure Digital (SD) controller. 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. As an example, this shows one MMC, one eMMC, and one SD card.



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 MMC/SD devices.
- Supports the Atmel® SDIO controller used in the SAM range of microcontrollers (the Atmel SAM9G, SAM3S, SAM4E, and SAM4S families).
- 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 Direct Memory Access (DMA) transfers.
- Supports 4 and 8 bit modes.

1.3 Device Description

This table summarizes the properties of the relevant device types:

	SAM3S	SAM4E	SAM4S	SAM9G
Flash (kB)	64 to 512			
SRAM (kB)	16 to 64		1	64
L1 cache memory (kB)				2 x 32
Error-Correcting Code (ECC)			64	1 bit SLC
Planes			2	2
Bus width			x8	x16

Requirement

1 bit per 528 bytes. The sample driver includes a software ECC algorithm. This can be modified to use hardware ECC if this is provided by the host micro-controller.

1.4 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_init	The generic **mmcsd_initcad()** routine used by MMC/SD media drivers to initialize a card.
util_hcc_mem	The HCC memory management utility.
media_drv_mmcsd_atmel_mcipdc	The media driver package described in this document.

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 Atmel SAM User Guide

This is this document.

1.5 Change History

This section describes past changes to this manual.

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

The current version of this manual is 1.60. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.60	2017-08-18	2.03	Updated *Packages* list.
1.50	2017-06-23	2.03	New *Change History* format.
1.40	2017-04-24	2.03	Reverted module name to **media_drv_atmel_mcipdc**.
1.30	2016-01-20	2.02	Added *OS Abstraction Layer* section. Removed some PSP functions.
1.20	2015-05-08	2.01	Renamed module.
1.10	2015-04-22	2.01	Added functions to *PSP Porting*.
1.00	2015-03-20	1.04	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 and PSP files.

2.1 API Header File

The file **src/api/api_mdriver_mmcsd.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. For details of these options, see Configuration Options.

2.3 Source Code Files

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

File	Description
mmcsd.c	Main source code.
mmcsd.h	Header file.
mmcsd_mci_regs.h	Register definitions.

2.4 Version File

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

2.5 Platform Support Package (PSP) Files

These files in the directory **src/psp/target/mmcsd** provide functions and other elements the core code needs to use, depending on the hardware. Modify these files as required for your hardware.

> **Note:** You must modify these PSP implementations for your specific microcontroller and development board; see PSP Porting for details.

File	Description
psp_mmcsd.c	Source code.
psp_mmcsd.h	Function definitions and other elements.

3 Configuration Options

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

MMCSD_NUM_UNITS

The number of MMC/SD channels (units). The default is 1.

MMCSD1_VOLTAGE_RANGE_170_195, MMCSD1_VOLTAGE_RANGE_270_360

These two options set the voltage range in which unit 1's signals operate:

- Set the first to 1 to set the valid voltage range to 1.7 - 1.95V. Its default is 0.
- Set the second to 1 to set the valid voltage range to 2.7 - 3.6V. This is the default.

MMCSD2_VOLTAGE_RANGE_170_195, MMCSD2_VOLTAGE_RANGE_270_360

These two options set the voltage range in which unit 2's signals operate:

- Set the first to 1 to set the valid voltage range to 1.7 - 1.95V. Its default is 0.
- Set the second to 1 to set the valid voltage range to 2.7 - 3.6V. This is the default.

MMCSD1_ALLOW_4BIT, MMCSD1_ALLOW_8BIT

To set the mode of unit 1 to 4 bit, keep the first option at the default of 1, and the second at 0. To set 8 bit mode, reverse the settings.

MMCSD2_ALLOW_4BIT, MMCSD2_ALLOW_8BIT

To set the mode of unit 2 to 4 bit, keep the first option at the default of 1, and the second at 0. To set 8 bit mode, reverse the settings.

MMCSD1_SPEED_LIMIT, MMCSD2_SPEED_LIMIT

For each unit, set this to the maximum desired frequency in kHz when testing prototype boards with wiring that supports only lower speeds. The defaults are both 20000.

Setting an option to 0 disables the speed limit.

MCIPDC_DMA_SECTORS

The number of sectors the DMA buffer holds. The default is 32. The DMA buffer is required for non-aligned transfers only.

MMCSD_ALLOW_EMMC_MANAGEMENT

This eMMC management option is for future use so not supported in this driver version. Do not change this from the default of 0.

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 **atmel_mcipdc_initfunc()** function.
2. **atmel_mcipdc_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

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

Arguments

Argument	Description	Type
driver_param	The drive to use. 0 is the first drive.	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	0x00	Successful execution.
MMCSD_ERR_NOTINITIALIZED	0x65	Driver not initialized.
MMCSD_ERR_INIT	0x66	Initialization error.
MMCSD_ERR_CMD	0x67	CMD error.
MMCSD_ERR_TRANS	0x68	Transfer error.
MMCSD_ERR_STARTBIT	0x69	Start bit error.
MMCSD_ERR_BUSY	0x6A	CMD error.
MMCSD_ERR_CRC	0x6B	CRC error.
MMCSD_ERR_WRITE	0x6C	Write error.
MMCSD_ERR_WRITEPROTECT	0x6D	Media is write-protected.
MMCSD_ERR_NOTAVAILABLE	0x6E	Resource 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 OS Abstraction Layer

All HCC modules use the OS Abstraction Layer (OAL). This allows modules to run seamlessly with a wide variety of RTOSes, or without an RTOS.

This module uses the following OAL components:

OAL Resource	Number Required
Tasks	0
Mutexes	1
Events	0

5.2 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 module makes use of the following standard PSP functions:

Function	Package	Component	Description
psp_memcpy()	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.
psp_memset()	psp_base	psp_string	Sets the specified area of memory to the defined value.

The module makes use of the following PSP functions. These functions are provided by the PSP to perform various tasks. Their design makes it easy for you to port them to work with your hardware solution. The package includes samples in the PSP file **src/psp/target/mmcsd/psp_mmcsd.c**:

Function	Description
psp_mmcsd_init()	Initializes the device (clocks, GPIO pin, and so on).
psp_mmcsd_delete()	Deletes the device, releasing the associated resources.
psp_mmcsd_power()	Turns on the card power. (On the default development board this has no effect).
psp_mmcsd_card_plugged()	Checks whether an MMC/SD card is present.
psp_mmcsd_card_writeprotect()	Checks whether the card's write-protect switch is on. (On the default development board this has no effect).
psp_mmcsd_drive_cmd()	Selects push-pull or open-drain mode on the CMD pin.
psp_mmcsd_drive_d0()	Enables or disables pull-up on the D0 pin.

These functions are described in the following sections.

psp_mmcsd_init

This function is provided by the PSP to initialize the device.

This enables the clocks, GPIO pin, and so on.

Format

```
int psp_mmcsd_init ( void )
```

Arguments

None.

Return Values

Return value	Description
MMCSD_NO_ERROR	Successful execution.
MMCSD_ERROR_INIT	Operation failed.

psp_mmcsd_delete

This function is provided by the PSP to delete the device, releasing the associated resources.

Format

```
int psp_mmcsd_delete ( void )
```

Arguments

None.

Return Values

Return value	Description
MMCSD_NO_ERROR	Successful execution.
MMCSD_ERROR	Operation failed.

psp_mmcsd_power

This function is provided by the PSP to turn on the card's power.

This call blocks: it only returns when the power level is correct.

> **Note:** On the default development board the MMC/SD power cannot be turned off, so this call has no effect.

Format

```
void psp_mmcsd_power ( int on )
```

Arguments

Parameter	Description	Type
on	The power setting.	int

Return Values

Return value	Description
MMCSD_NO_ERROR	Successful execution.
MMCSD_ERROR	Operation failed.

psp_mmcsd_card_plugged

This function is provided by the PSP to check whether an MMC/SD card is present.

This returns the state of the CD pin.

Format

```
int psp_mmcsd_card_plugged ( void )
```

Arguments

None.

Return Values

Return value	Description
0	No card is present.
1	A card is present.

psp_mmcsd_card_writeprotect

This function is provided by the PSP to check a card's write-protect state.

> **Note:** On the default development board the write-protect switch is not connected to the MCU, so this call always returns non-protected status.

Format

```
int psp_mmcsd_card_writeprotect ( void )
```

Arguments

None.

Return Values

Return value	Description
0	The card is not write-protected.
1	The card is write-protected.

psp_mmcsd_drive_cmd

This function is provided by the PSP to select push-pull or open-drain mode on the CMD pin.

Format

```
void psp_mmcsd_drive_cmd (
    uint32_t   uid,
    uint32_t   push_pull )
```

Arguments

Parameter	Description	Type
uid	The unit ID. This is ignored in this implementation.	uint32_t
push_pull	Set this to 1 to activate push-pull mode. Otherwise, the pin is in open-drain mode.	uint32_t

Return Values

Return value	Description
MMCSD_NO_ERROR	Successful execution.
MMCSD_ERROR	Operation failed.

psp_mmcsd_drive_d0

This function is provided by the PSP to enable or disable pull-up on the D0 pin.

Format

```
void psp_mmcsd_drive_d0 (
    uint32_t   uid,
    uint32_t   pull_up )
```

Arguments

Parameter	Description	Type
uid	The MMC/SD unit ID. This is ignored in this implementation.	uint32_t
pull-up	Set this to 1 to enable pull-up on D0. Set it to 0 to disable it.	uint32_t

Return Values

Return value	Description
MMCSD_NO_ERROR	Successful execution.
MMCSD_ERROR	Operation failed.