

MMC and SD Media Driver for Kinetis User Guide

Version 1.60

For use with MMC and SD Media Driver for Kinetis
module versions 1.14 and above

Date: 23-Jun-2017 10:48

All rights reserved. This document and the associated software are the sole property of HCC Embedded. Reproduction or duplication by any means of any portion of this document without the prior written consent of HCC Embedded is expressly forbidden.

HCC Embedded reserves the right to make changes to this document and to the related software at any time and without notice. The information in this document has been carefully checked for its accuracy; however, HCC Embedded makes no warranty relating to the correctness of this document.

Table of Contents

System Overview	3
Introduction	3
Feature Check	4
Packages and Documents	5
Packages	5
Documents	5
Change History	6
Source File List	7
API Header File	7
Configuration File	7
Source Code Files	7
Version File	7
Platform Support Package (PSP) Files	8
Configuration Options	9
Application Programming Interface	11
mmcsd_initfunc	12
mmcsd_get_cid	13
F_DRIVER Structure	14
t_mmcsd_cid	15
Error Codes	16
Integration	17
OS Abstraction Layer	17
PSP Porting	17
psp_mmcsd_init	19
psp_mmcsd_delete	20
psp_mmcsd_power	21
psp_mmcsd_card_plugged	22
psp_mmcsd_card_writeprotect	23
psp_mmcsd_get_speed	24
psp_mmcsd_set_speed	25
psp_mmcsd_drive_cmd	26
psp_mmcsd_drive_d0	27

1 System Overview

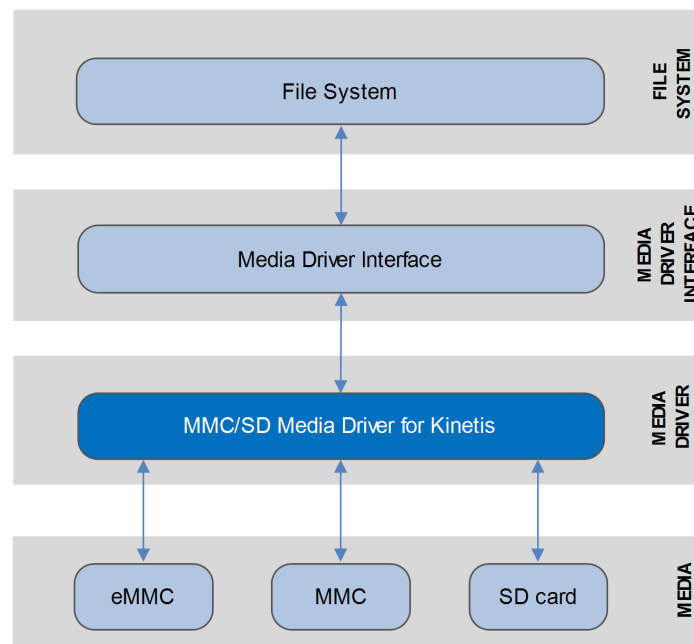
1.1 Introduction

This guide is for those who want to use HCC Embedded's MMC and SD Media Driver for Kinetis MCUs from Freescale™ in their system. This guide covers all aspects of configuration and use. This module can also be used for Vybrid MCUs and Freescale™ i.MX25 MCUs.

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](#).
- Compatible with all Kinetis and Vybrid MCUs and also some Freescale™ i.MX application processors.
- 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 1 bit, 4 bit, and 8 bit transfer width.
- Supports Direct Memory Access (DMA) transfers.
- An API function lets you read the card's manufacturer configuration.
- The voltage range is configurable.

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_init	This contains the generic mmcsd_initcad() routine used by MMC/SD media drivers to initialize a card.
media_drv_mmcsd_kinetis	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 Kinetis 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 Kinetis User Guide](#).
- For the history of changes made to the package code itself, see [History: media_drv_mmcsd_kinetis](#).

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	2017-06-23	1.14	New <i>Change History</i> format.
1.50	2017-04-24	1.14	Added eMMC to <i>Introduction</i> .
1.40	2016-02-02	1.13	Changed <i>Feature Check</i> .
1.30	2016-01-20	1.12	Various small changes.
1.20	2016-01-05	1.12	Added functions to <i>PSP Porting</i> .
1.10	2015-05-08	1.10	Various small changes.
1.00	2015-03-20	1.10	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 API functions, 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-driv/mmcsd` hold the source code for the media driver. **These files should only be modified by HCC.**

File	Description
<code>mmcsd.c</code>	Source code.
<code>mmcsd.h</code>	Header file.

2.4 Version File

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

2.5 Platform Support Package (PSP) Files

These files provide functions the core code needs to call, depending on the hardware. They are in the directory **src/psp/target**. There are folders for the K60 series and for the K70 series.

Note: You must modify these PSP implementations for your specific microcontroller and development board; see [PSP Porting](#) for details.

File	Description
include/hcc_mk60f_reg.h or hcc_mk70f_reg.h	Register setup.
mmcsd/psp_mmcsd.c	Source code.
mmcsd/psp_mmcsd.h	Function definitions.

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. Do not change this value; it is always 1 for this driver.

MMCSD1_VOLTAGE_RANGE_170_195

Set this to 1 to set the voltage range in which the signals of unit 1 operate to 1.7-1.95V. The default is 0.

MMCSD1_VOLTAGE_RANGE_270_360

Keep this at the default of 1 to set the voltage range in which the signals of unit 1 operate to 2.7-3.6V.

MMCSD1_ALLOW_4BIT

Keep this at the default of 1 to set the mode to 4-bit.

MMCSD1_ALLOW_8BIT

Set this to 1 to set the mode to 8-bit. The default is 0.

MMCSD1_SPEED_LIMIT

Set `MMCSDx_SPEED_LIMIT` to the maximum desired frequency in kHz when testing prototype boards, the wiring of which supports only lower speeds.

Keep this at the default of zero to disable the speed limit.

MMCSD_DMA_SECTORS

The number of sectors the DMA buffer holds. The default is 8. The DMA buffer is required for non-aligned transfers only.

MMCSD_ALLOW_EMMC_MANAGEMENT

Set this to 1 if the eMMC management module is used. The default is 0.

The eMMC module provides features including the following:

- Get/set card partitioning settings.
- Get/set Reliable Write settings for User Data Area and partitions.
- Get card Health Status.

MMCSD_IRQ_DRIVEN_IF

Set this to 1 for interrupt-driven Command and Data transfer. The default is 0.

MMCSA_ISR_ID

The Interrupt Vector ID. The default of 81 equates to SDHC_IRQn.

MMCSA_ISR_PRIORITY

The Interrupt priority. The default is (configLIBRARY_LOWEST_INTERRUPT_PRIORITY - 1).

4 Application Programming Interface

This section describes the API functions, the structures used, 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 mmc_sd_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 * ( mmc_sd_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 mmc_sd_get_cid

Use this function to get the content of the Card ID (CID) register.

Format

```
int mmc_sd_get_cid (  
    unsigned long  driver_param,  
    t_mmc_sd_cid * p_cid )
```

Arguments

Argument	Description	Type
driver_param	This is not used. It identifies the device to use.	unsigned long
p_cid	A pointer to the CID buffer.	t_mmc_sd_cid *

Return values

Return value	Description
Zero	Successful execution.
Non-zero	Card is not initialized.

4.3 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.

4.4 t_mmcsd_cid

The `t_mmcsd_cid` typedef sets the Card ID (CID) register fields:

Element	Type	Description
manuf_id	uint8_t	MID, Manufacturer ID (3 = SanDisk, 2 = Kingston, and so on).
oem_id[3]	char_t	OID, OEM/Application ID (ASCII characters on SD, ID on MMC).
product_name[7]	char_t	PNM, Product name (5 or 6 ASCII characters).
version_major	uint8_t	PRV, major product revision.
version_minor	uint8_t	PRV, minor product revision.
serial_number	uint32_t	PSN, product serial number.
manuf_year	uint16_t	MDT, Manufacturing year (decoded from MDT).
manuf_month	uint8_t	MDT, Manufacturing month (decoded from MDT).

4.5 Error Codes

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

Return Value	Value	Description
<code>MMCSO_ERR_NOTPLUGGED</code>	-1	For high level.
<code>MMCSO_NO_ERROR</code>	0x00	Successful execution.
<code>MMCSO_ERR_NOTINITIALIZED</code>	0x65	Driver not initialized.
<code>MMCSO_ERR_INIT</code>	0x66	Initialization error.
<code>MMCSO_ERR_CMD</code>	0x67	Command error.
<code>MMCSO_ERR_STARTBIT</code>	0x68	Start bit error.
<code>MMCSO_ERR_BUSY</code>	0x69	Driver already active.
<code>MMCSO_ERR_CRC</code>	0x70	CRC error.
<code>MMCSO_ERR_WRITE</code>	0x71	Write error.
<code>MMCSO_ERR_WRITEPROTECT</code>	0x72	Media is write-protected.
<code>MMCSO_ERR_NOTAVAILABLE</code>	0x73	Media 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	1

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_memmove()	psp_base	psp_string	Copies n bytes from one memory area to another memory area.
psp_memset()	psp_base	psp_string	Sets the specified area of memory to the defined value.

The module makes use of the following standard PSP macros:

Macro	Package	Element	Description
PSP_RD_LE32	psp_base	psp_endianness	Reads a 32 bit value stored as little-endian from a memory location.
PSP_WR_LE32	psp_base	psp_endianness	Writes a 32 bit value to be stored as little-endian to a memory location.

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/mmcsd_hw.c**.

Function	Description
psp_mmcsd_init()	Initializes the hardware (clocks, GPIO, and so on).
psp_mmcsd_delete()	Deletes the device, releasing the associated resources.
psp_mmcsd_power()	Powers on the device.
psp_mmcsd_card_plugged()	Gets the card status, non-zero if the card is powered on.
psp_mmcsd_card_writeprotect()	Checks whether a card's write protect switch is on. This returns zero if it is off.
psp_mmcsd_get_speed()	Gets the SDHC clock frequency in kHz.
psp_mmcsd_set_speed()	Configures SDHC to generate a specific clock frequency.
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_mmc_s_d_init

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

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

Format

```
int psp_mmc_s_d_init ( void )
```

Arguments

None.

Return Values

Return value	Description
MMCS_D_NO_ERROR	Successful execution.
MMCS_D_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
MMCSO_NO_ERROR	Successful execution.
MMCSO_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_mmc_ssd_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_mmc_ssd_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_get_speed

This function is provided by the PSP to get the SDHC unit's clock frequency in kHz.

Format

```
uint32_t psp_mmcsd_get_speed ( uint32_t uid )
```

Arguments

Parameter	Description	Type
uid	The MMC/SD unit ID. This is ignored in this implementation.	uint32_t

Return Values

Return value	Description
The SDIO clock frequency	Successful execution.
MMCSD_ERROR	Operation failed.

psp_mmcsd_set_speed

This function is provided by the PSP to configure the SDHC unit to generate a specific clock frequency.

Format

```
void psp_mmcsd_set_speed (  
    uint32_t  uid,  
    uint32_t  khz )
```

Arguments

Parameter	Description	Type
uid	The unit ID. This is ignored in this implementation.	uint32_t
khz	The desired frequency in KHz.	uint32_t

Return Values

Return value	Description
MMCSD_NO_ERROR	Successful execution.
MMCSD_ERROR	Operation failed.

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
MMCSHD_NO_ERROR	Successful execution.
MMCSHD_ERROR	Operation failed.