MMC and SD Media Driver for SPI User Guide

Version 1.60

For use with MMC and SD Media Driver for SPI versions 2.02 and above

Date: 22-Mar-2018 10:21

Table of Contents

1 System Overview

This chapter contains the fundamental information for this module.

The component sections are as follows:

- Introduction – describes the main elements of the module.
- Feature Check – summarizes the main features of the module as bullet points.
- Packages and Documents – the *Packages* section lists the packages that you need in order to use this module. The *Documents* section lists the relevant user guides.
- Change History – lists the earlier versions of this manual, giving the software version that each manual describes.

1.1 Introduction

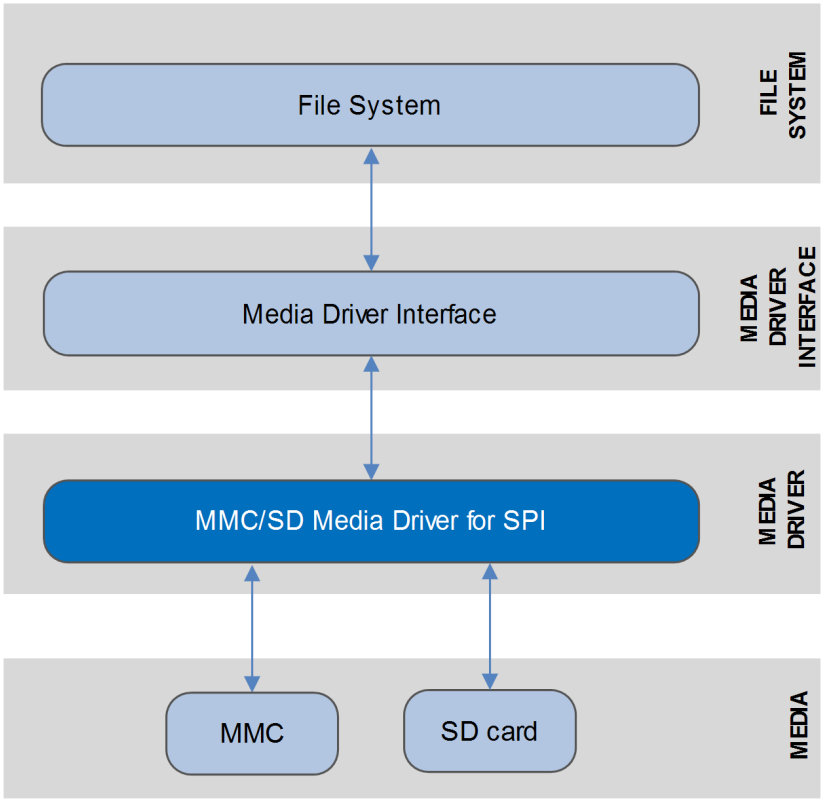
This guide is for those who want to use HCC Embedded's MMC and SD Media Driver for SPI in their system. This module provides a media driver for HCC's psp_spi interface, which can be provided for any standard microcontroller. You simply need to port psp_spi to your target and the driver calls your functions. This guide covers all aspects of configuration and use.

> **Note:** HCC provides tested SPI drivers for most standard microcontrollers that conform to HCC's PSP SPI specification.

This media driver conforms to the *HCC Media Driver Interface Specification*.

The media driver provides an interface for a file system to read from and write to Secure Digital (SD) or MultiMediaCard (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.

The diagram below shows a typical system architecture including a file system, media driver and media. As an example, this shows one MMC 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 multiple card types: MMC/SD and also SDHC (Secure Digital High Capacity) and SDXC (Secure Digital eXtended Capacity).
- Supports eMMC (embedded MMC).

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 that includes the framework all media drivers use.
media_drv_mmcsd_spi	The media driver package described in this document.
psp_template_spi	The SPI Platform Support Package (PSP).
oal_base	The base OS Abstraction Layer (OAL) package, needed if an RTOS is used.

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 MMC and SD Media Driver for SPI 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 File System Media Driver PDFs.
- For the history of changes made to the package code itself, see History: media_drv_mmcsd_spi.

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	2018-03-22	2.02	Added SPI_MAX_BAUDRATE configuration option.
1.50	2018-03-09	2.01	Added **oal_base** to *Packages* list. Functions "mmcsd_xxxx" renamed to "mmcsd_spi_xxxx". Error messages renamed "MMCSD_SPI_xxxxx". Added configuration option ENABLE_HIGH_SPEED.
1.40	2017-06-23	1.19	New *Change History* format.
1.30	2015-11-23	1.18	Added note to *Introduction*.
1.20	2015-08-24	1.18	Various small changes.
1.10	2015-05-08	1.15	Added *Change History*.
1.00	2015-03-20	1.15	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 files and PSP files.

2.1 API Header File

The file **src/api/api_mdriver_mmcsd_spi.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 Files

The following files in the directory **src/config** contain all the configurable parameters of the system. Configure these as required. For details of these options, see Configuration Options.

File	Description
config_mdriver_mmcsd_spi.h	Configuration options.
config_mdriver_mmcsd_spi.c	SPI unit ID configuration.

2.3 System File

The file **src/media-drv/mmc-spi/mmcsd_spi.c** is the source code for the media driver. **This file should only be modified by HCC**.

2.4 Version File

The file **src/version/ver_mdriver_mmcsd_spi.h** contains the version number of this module. This version number is checked by all modules that use this 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/mmcsd-spi**.

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

File	Description
mmcsd-spi_hw.c	Source code.
mmcsd-spi_hw.h	Function definitions.

3 Configuration Options

System configuration is controlled by two files in **src/config**.

3.1 config_mdriver_mmcsd_spi.h

Set the system configuration options in this file. This section lists the available configuration options and their default values.

USE_CRC

Keep the default of 1 to use CRC for communication. Otherwise, set this to 0.

CRC_ROM_TABLE

Keep the default of 1 to put the CRC table in ROM. Otherwise, set this to 0.

RTOS_SUPPORT

Keep the default of 1 if the driver is used with an RTOS. This is only required to perform wait cycles with *oal_task_sleep* instead of SPI transactions.

MAX_UNITS

The maximum number of MMC/SD units supported. The default is 1.

OVERRIDE_SDHC_TIMEOUTS

Set this to 1 for any card that does not meet SDHC standards, for example a Swissbit® card. The default is 0.

VALID_VOLTAGE_RANGE

The valid voltage range. The default of 0x00FF8000 means 2.7 - 3.6 V.

ENABLE_HIGH_SPEED

Keep the default of 1 to enable switching to High Speed mode for cards that support this mode. Otherwise, set this to 0.

SPI_MAX_BAUDRATE

The maximum SPI clock speed in [Hz]; set this as required. The default of 0 means there is no limit.

3.2 config_mdriver_mmcsd_spi.c

This file sets the SPI unit ID configuration. This is defined by default as follows:

```
const uint8_t  g_mmcsd_spi_uid[MAX_UNITS] =
{
  0
};
```

4 Application Programming Interface

This section describes the functions, the structure and typedef they use, and the error codes.

When the media driver is used:

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

4.1 mmcsd_spi_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_spi_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 mmcsd_spi_get_cid

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

> **Note:** You must call **mmcsd_spi_initfunc()** before this function.

Format

```
int mmcsd_spi_get_cid (
    unsigned long    driver_param,
    t_mmcsd_cid *    p_cid )
```

Arguments

Argument	Description	Type
driver_param	This identifies the drive to use. The first drive is 0.	unsigned long
p_cid	A pointer to the CID buffer.	t_mmcsd_cid *

Return values

Return value	Description
MMCSD_SPI_NO_ERROR	Successful execution.
MMCSD_SPI_ERR_NOTINITIALIZED	The drive 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 CID Register

This is the format of the *t_mmcsd_cid* typedef:

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	char_t	PNM – Product name (ASCII, 5 or 6 chararcters).
version_major	uint8_t	PRV – Product revision, major.
version_minor	uint8_t	PRV – Product revision, minor.
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 MMCSD_SPI_NO_ERROR, a value of zero. The following table shows the meaning of the error codes.

Return Value	Value	Description
MMCSD_SPI_ERR_NOTPLUGGED	-1	For high level.
MMCSD_SPI_NO_ERROR	0	Successful execution.
MMCSD_SPI_ERR_NOTINITIALIZED	0x65	Driver not initialized.
MMCSD_SPI_ERR_INIT	0x66	Initialization error.
MMCSD_SPI_ERR_CMD	0x67	Command error.
MMCSD_SPI_ERR_STARTBIT	0x68	Start bit error.
MMCSD_SPI_ERR_BUSY	0x69	Driver already active.
MMCSD_SPI_ERR_CRC	0x70	CRC error.
MMCSD_SPI_ERR_WRITE	0x71	Write error.
MMCSD_SPI_ERR_WRITEPROTECT	0x72	Media is write-protected.
MMCSD_SPI_ERR_NOTAVAILABLE	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 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 standard PSP SPI functions. For details of these functions, refer to the *HCC SPI Driver PSP User Guide*.

Function	Package	Element	Description
psp_spi_init()	psp_base	psp_spi	Initializes the SPI port.
psp_spi_start()	psp_base	psp_spi	Starts the SPI port.
psp_spi_cs_hi()	psp_base	psp_spi	Sets chip select high.
psp_spi_cs_lo()	psp_base	psp_spi	Sets chip select low.
psp_spi_get_baudrate()	psp_base	psp_spi	Gets the baudrate.
psp_spi_set_baudrate()	psp_base	psp_spi	Sets the baudrate.
psp_spi_rx()	psp_base	psp_spi	Receives a number of bytes.
psp_spi_tx()	psp_base	psp_spi	Transmits a number of bytes.
psp_spi_tx1()	psp_base	psp_spi	Transmits one byte.
psp_spi_lock()	psp_base	psp_spi	Locks the SPI for the specific unit. This can be useful if multiple units are attached to the same SPI bus.
psp_spi_unlock()	psp_base	psp_spi	Unlocks the SPI for the specific unit. This can be useful if multiple units are attached to the same SPI bus.

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-spi/mmcsd-spi_hw.c**.

Function	Description
mmcsd_hw_init()	Initializes the hardware.
mmcsd_hw_delete()	Deletes the device, releasing associated resources.
mmcsd_hw_card_is_present()	Checks whether a card is present.
mmcsd_hw_card_is_wp()	Checks whether the card's write-protect switch is on.

These functions are described in the following sections.

mmcsd_hw_init

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

Format

```
int mmcsd_hw_init ( uint8_t uid )
```

Arguments

Argument	Description	Type
uid	The unit ID.	uint8_t

Return Values

Return value	Description
MMCSD_HW_SUCCESS	Successful execution.
MMCSD_HW_ERROR	Operation failed.

mmcsd_hw_delete

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

Format

```
int mmcsd_hw_delete( uint8_t uid )
```

Arguments

Argument	Description	Type
uid	The unit ID.	uint8_t

Return Values

Return value	Description
MMCSD_HW_SUCCESS	Successful execution.
MMCSD_HW_ERROR	Operation failed.

mmcsd_hw_card_is_present

This function is provided by the PSP to report whether a card is inserted.

Format

```
int mmcsd_hw_card_is_present ( uint8_t uid )
```

Arguments

Argument	Description	Type
uid	The unit ID.	uint8_t

Return Values

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

mmcsd_hw_card_is_wp

This function is provided by the PSP to get the Write Protect state of the card.

Format

```
int mmcsd_hw_card_is_wp ( uint8_t uid )
```

Arguments

Argument	Description	Type
uid	The unit ID.	uint8_t

Return Values

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