Media Driver for Compact Flash Cards with an IO Interface User Guide

Version 1.10

For use with module versions 1.01 and above

Date: 23-Jun-2017 17:39

Table of Contents

1 System Overview

1.1 Introduction

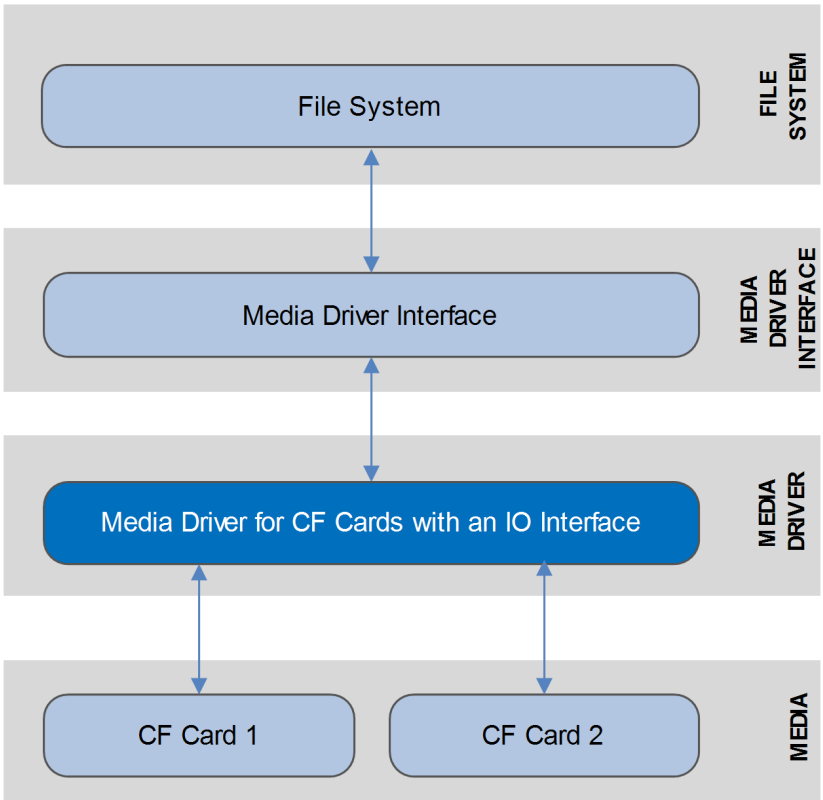
This guide is for those who want to use HCC Embedded's media driver for Compact Flash (CF) cards with serial Input/Output (IO) interfaces as part of their system. This guide covers all aspects of configuration and use.

Compact Flash is a standard storage media. There are three types of interface between host systems and CF cards:

- Integrated Drive Electronics (IDE) mode.
- Serial Input/Output (IO) mode – covered in this manual.
- Memory (MEM) mode.

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 a CF card storage device. 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.



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 Compact Flash cards.

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_cfc_io	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 Media Driver for Compact Flash Cards with an IO Interface 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: Media Driver for Compact Flash Cards with an IO Interface User Guide.
- For the history of changes made to the package code itself, see History: media_drv_cfc_io.

The current version of this manual is 1.10. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.10	2017-06-23	1.01	New *Change History* format.
1.00	2015-12-11	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_cfc_io.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_cfc_io.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 file **src/media-drv/cfc_io/cfc_io_drv.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_cfc_io.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.

3 Configuration Options

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

MDRIVER_CFC_IO_DRV_CNT

The maximum number of CFC cards. The default is 1.

CFC_DEF_SECTOR_SIZE

The default sector size. The default is 512.

CFC_IO_BASE0

The CFC base memory address. The default is 0x01000000.

CFC_IO_HCC_HW

Set this to 1 to use HCC hardware. The default is 0.

CFC_IO_TOVALUE

The iteration count for timeout. The default is 1300000.

CFC_IO_WAITCYCLE

The required delay for get status. The default is 50000.

4 Application Programming Interface

This section describes the single Application Programming Interface (API) function, the structure it uses, and the error codes.

When the media driver is used:

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

4.1 cfc_io_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 * cfc_io_initfunc ( unsigned long driver_param )
```

Arguments

Argument	Description	Type
driver_param	This identifies the drive to use. The first drive is 0. This value 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 **cfc_io_initfunc()** executes successfully, it returns with CFC_IO_NO_ERROR, a value of zero. The following table shows the meaning of the error codes.

Return Value	Value	Description
CFC_IO_ERR_NOTPLUGGED	-1	Drive not plugged in (high level error).
CFC_IO_NO_ERROR	0	Successful execution.
CFC_IO_ERR_BUSY_ATCYL	101	Waiting for write operation to finish.
CFC_IO_ERR_BUSY_ATDRQ	102	Waiting for data request.
CFC_IO_ERR_BUSY_ATCMD	103	Cannot start command.
CFC_IO_ERR_TIMEOUT	104	Timed out.
CFC_IO_ERR_STATE	105	Driver is already active.
CFC_IO_ERR_SECCOU	106	Sector count error.
CFC_IO_ERR_NOTAVAILABLE	107	Not available.

5 Integration

This section specifies the elements of this package that need porting, depending on the target environment.

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 function:

Function	Package	Component	Description
psp_memcpy()	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.

The module makes use of the following standard PSP macros:

Macro	Package	Element	Description
PSP_RD_BE16	psp_base	psp_endianness	Reads a 16 bit value stored as big-endian from a memory location.
PSP_RD_LE16	psp_base	psp_endianness	Reads a 16 bit value stored as little-endian from a memory location.
PSP_RD_LE32	psp_base	psp_endianness	Reads a 32 bit value stored as little-endian from a memory location.