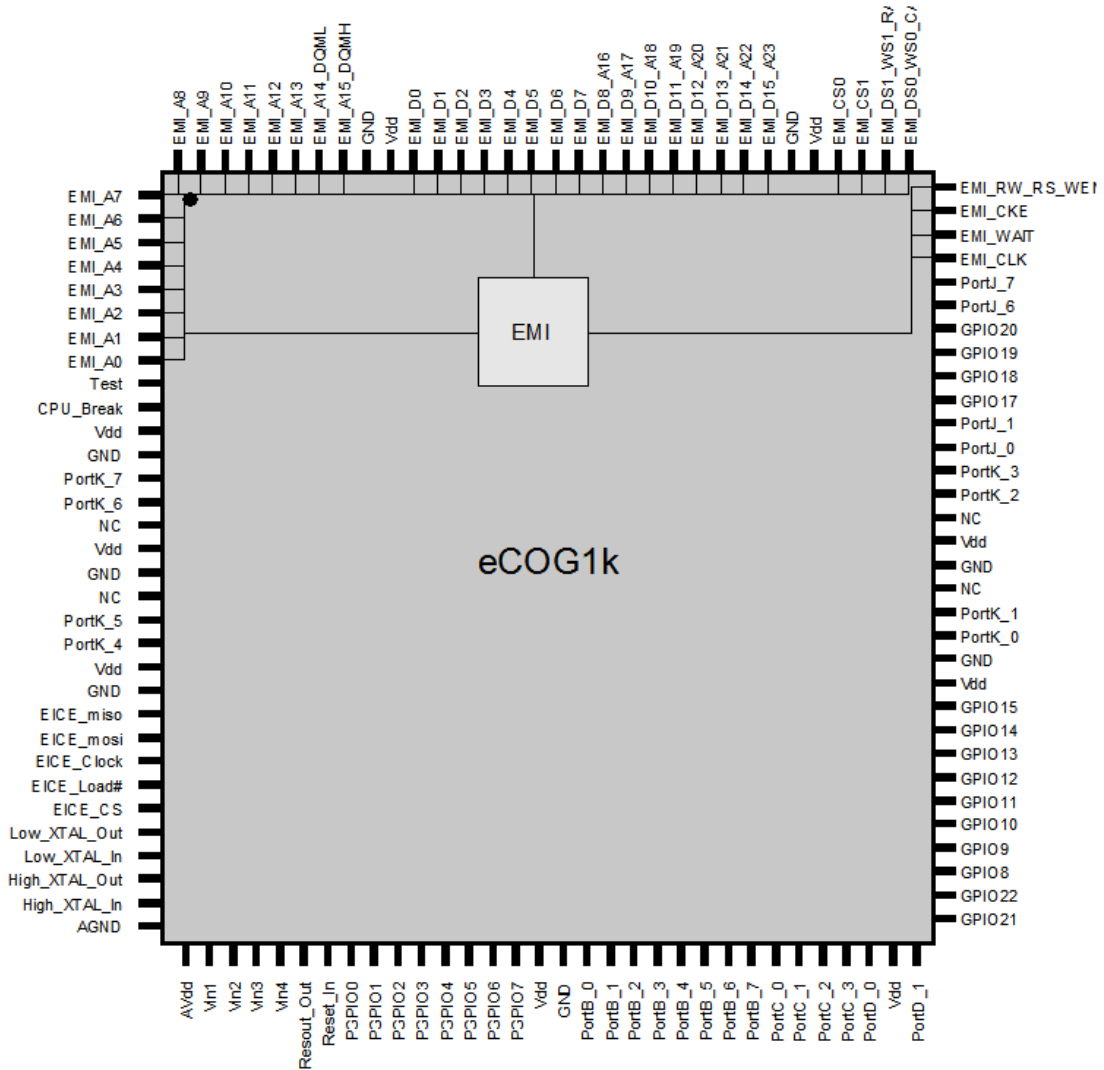


# AN043 – Interfacing to a CompactFlash Memory Card

## Version 1.0

This application note describes interfacing a CompactFlash (CF) memory card to the eCOG1k microcontroller via the External Memory Interface (EMI).



## Confidential and Proprietary Information

©Cyan Technology Ltd, 2008

This document contains confidential and proprietary information of Cyan Technology Ltd and is protected by copyright laws. Its receipt or possession does not convey any rights to reproduce, manufacture, use or sell anything based on information contained within this document.

Cyan Technology™, the Cyan Technology logo and Max-eICE™ are trademarks of Cyan Holdings Ltd. CyanIDE® and eCOG® are registered trademarks of Cyan Holdings Ltd. Cyan Technology Ltd recognises other brand and product names as trademarks or registered trademarks of their respective holders.

Any product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by Cyan Technology Ltd in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. Cyan Technology Ltd shall not be liable for any loss or damage arising from the use of any information in this guide, any error or omission in such information, or any incorrect use of the product.

This product is not designed or intended to be used for on-line control of aircraft, aircraft navigation or communications systems or in air traffic control applications or in the design, construction, operation or maintenance of any nuclear facility, or for any medical use related to either life support equipment or any other life-critical application. Cyan Technology Ltd specifically disclaims any express or implied warranty of fitness for any or all of such uses. Ask your sales representative for details.



## Revision History

Version	Date	Notes
V1.0	14/08/2006	Initial version

## Contents

1	Introduction .....	5
2	Glossary .....	5
3	CompactFlash Memory Card Overview .....	5
3.1	Operating Mode .....	6
3.2	Data Storage Addressing .....	6
3.3	Configuration and ATA Registers .....	6
4	CF Memory Card Hardware Interface .....	7
4.1	Physical to Logical Address Translation .....	8
5	CF Memory Card Software on the eGOG1k .....	9
5.1	Example Application .....	9
5.1.1	<i>Application Dependencies</i> .....	9
5.1.2	<i>Application Operation</i> .....	9
5.1.3	<i>Code Size and Memory Usage</i> .....	9
5.2	Performance .....	10
6	Design Limitations .....	10
7	References .....	10
Appendix A	CF Memory Card API Functions .....	11

## 1 Introduction

This application note describes interfacing a CompactFlash (CF) memory card to the eCOG1k microcontroller via the External Memory Interface (EMI). The CF memory card is accessed in PC ATA memory mode using a 16-bit data width, and Cylinder/Head/Sector (C/H/S) addressing mode.

A detailed reference design is shown, including hardware and software interfaces. The core software consists of a set of file system independent subroutines that perform the functions required to write and read information to/from the CF memory card.

## 2 Glossary

A table of abbreviations used in this document.

ATA	AT Attachment, Advanced Technology Attachment A standard interface for connecting storage devices inside a PC.
CF	CompactFlash
C/H/S	Cylinder/Head/Sector
ECC	Error Correcting Code
eCOG1k	Cyan Technology target microcontroller
EMI	External Memory Interface
FAT	File Allocation Table
GPIO	General Purpose Input/Output
IDE	Integrated Drive Electronics A version of ATA with drive electronics contained in the storage device.
LBA	Logical Block Address

## 3 CompactFlash Memory Card Overview

CompactFlash memory card is a removable mass storage device that electrically complies with the Personal Computer Memory Card International Association ATA (PC Card ATA) standard. The CF Memory Card also supports True IDE Mode that is electrically compatible with an IDE disk drive. This CF concept is further expanded by the CF+ specifications to include I/O devices and magnetic data storage.

The internals of a CF memory card comprise an on-card controller, buffer and varying amount of flash memory. This on-card controller provides a high level interface to the host processor, enabling the host to issue ATA commands to the CF card, primarily to read and write blocks of memory. It also features Error Correcting Code (ECC) functions to protect the integrity of the stored data. The host processor addresses the card in 512 byte sectors.

The CF memory card supports both 3.3V and 5V operation and can function at either voltage level. In this example, it has been used at 3.3V to connect to an eCOG1k device also running from a 3.3V supply.

### 3.1 Operating Mode

CF Memory Card allows three different modes of operation, namely, PC Card memory mode, PC Card I/O mode, and True IDE mode. The connection of the CF memory card in PC Card memory and I/O mode are similar to those of the True IDE mode, but with a different register interface and slight changes of interface signals.

While in True IDE mode, the CF memory card functions like a hard disk drive and can be directly connected to an IDE bus. However, unlike the other modes, True IDE mode does not support hot insertion (insert and remove while power is applied) of the CF memory card.

The CF memory card data bus width can be set to either 8-bits or 16-bits wide. A minimum of three address lines is required to access and decode the eight main control registers and initiate data transfers on the card. Access to the remaining registers and further functionality of the CF memory card requires full decoding of all the address and control signals.

Detailed descriptions for the CF memory card signals are covered in the CF specification standard [1] and CF memory card product manual [2].

This application note focuses on PC Card memory mode operation, using 16-bit data width access and full address decoding.

### 3.2 Data Storage Addressing

Data storage to and from the CF memory card must be in blocks of one or more sectors at a time, where one sector equals 512 bytes of data. Sectors on the CF memory card can be identified according to two different addressing schemes, either Cylinder/Head/Sector (C/H/S) or Logical Block Address (LBA).

When operating in C/H/S addressing mode, a sector is identified in terms of its physical location according to the three parameters of cylinder, head and sector. Note that the first sector in C/H/S addressing mode starts at 0/0/1, as 0/0/0 is undefined. In contrast, LBA addressing mode requires only one parameter, linearly mapping and numbering each consecutive sector from 0 up to  $N - 1$ , where  $N$  is the total number of sectors.

The example application in this application note uses C/H/S addressing mode as this appears to be the default mode for the range of CF memory cards tested.

### 3.3 Configuration and ATA Registers

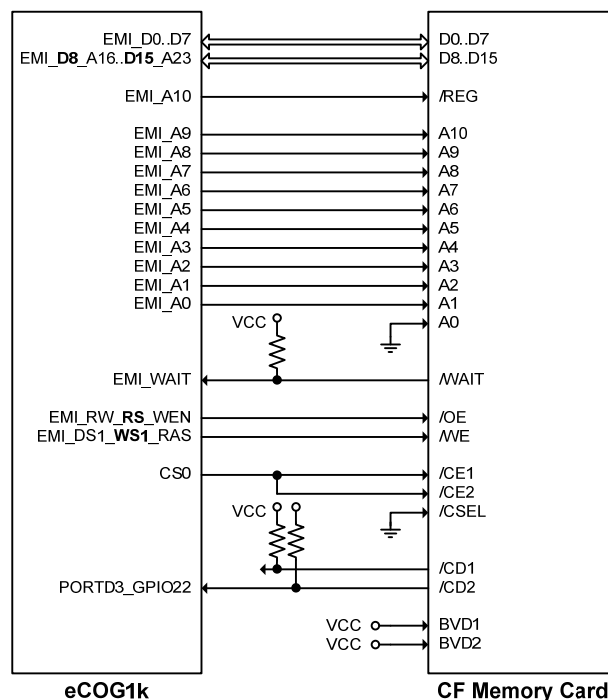
The CF memory card contains a set of configuration registers in its attribute memory and a set of ATA registers in common memory.

The configuration registers provide access to information on features and status of the CF memory card, and enable some card-specific configurations to be set up, while the latter ATA registers support data read and write functions to the CF memory card. In order to transfer data, the host processor loads the correct address into the appropriate ATA registers, and then executes the required operation by writing an ATA command to the ATA command register.

A detailed listing and explanation of the configuration and ATA registers, along with the ATA command supported, can be found in the CF specification standard [1] and the CF memory card product manual [2].

## 4 CF Memory Card Hardware Interface

The eCOG1k EMI interface can be used to interface directly to the CF memory card. This is facilitated by the inherent flexibility of the highly configurable EMI interface, which allows the required bus signals and timings to be easily set up. Figure 1 illustrates the physical connection of the signals of interest between the CF memory card and the eCOG1k.



**Figure 1. CF memory card hardware interface to the eCOG1k.**

When enabled, the CF memory card /CE1 and /CE2 pins are pulled low to accept 16-bit word size transfer. Data access to the card is then via the EMI parallel data bus. The EMI is configured as bus mode with 16-bit data width and 16-bit address width. Read and write strobe bus control signals are also enabled, along with the external wait signal. The CF memory card lowest address signal A0 is grounded as accesses to the card are in 16-bit words.

The card detect lines connected to the CF memory card /CD1 and /CD2 are externally pulled high. These lines are grounded when a card is fully inserted into its socket. Any one of the eCOG1k General Purpose Input/Output (GPIO) pins can be used to capture this card insertion event by configuring it to be interrupt-driven. In this example, port D pin 3 is chosen and is configured as GPIO22 to generate an interrupt when a falling edge is detected on the /CD2 line.

Note that in order to remain in PC Card memory mode and not enter True IDE mode during a power-up cycle, the /OE signal must be kept high during the power-up period.

## 4.1 Physical to Logical Address Translation

The CF memory card registers are directly memory mapped into the eCOG1k data space, taking up a 2Kbyte window. Access to these registers is set to start at a logical base address of 0x1000, via the MMU settings in the CyanIDE configuration tool. Table 1 provides the address decoding for the CF memory card configuration registers in its attribute memory and ATA registers in common memory.

CF memory card signal	/REG	A10	A9	A8-A4	A3	A2	A1	A0	
eCOG1k signal	A10	A9	A8	A7-A3	A2	A1	A0	Gnd	eCOG1k logical address (at base 0x1000)
Configuration Option Register read/write	0	0	1	00	0	0	0	0	0x1100
Card Status Register read/write	0	0	1	00	0	0	1	0	0x1101
Pin Replacement Register read/write	0	0	1	00	0	1	0	0	0x1102
Socket and Copy Register read/write	0	0	1	00	0	1	1	0	0x1103
Card Information Structure (CIS) read	0	0	0	XX	X	X	X	0	0x1000
CompactFlash Data read/write	1	0	X	XX	0	0	0	0	0x1400
Sector No. & Count read/write	1	0	X	XX	0	0	1	0	0x1401
Cylinder High & Low read/write	1	0	X	XX	0	1	0	0	0x1402
Status & Drive/Head write Command & Drive/Head read	1	0	X	XX	0	1	1	0	0x1403
Alt. Status & Dup. Error read Device Ctl. & Dup. Features write	1	0	X	XX	1	1	0	0	0x1406

**Table 1. CF memory card registers address decoding.**

## 5 CF Memory Card Software on the eGOG1k

The developed software is designed to illustrate how a CF memory card can be easily interfaced to the eCOG1k. Only a subset of the card-specific ATA command set is supported. The software library has been successfully tested with a range of different capacity SanDisk CF memory cards, from 64Mbytes to 256Mbytes. The set of library routines that has been developed is described in Appendix A.

### 5.1 Example Application

An example application that demonstrates the use of the CF memory card hardware and the software interface is provided. In outline, the application monitors the insertion of a CF memory card and updates the count value of a file on the card when the card insert is detected.

#### 5.1.1 Application Dependencies

The user initially needs to format the CF memory card in a suitable memory card reader (for example connected to a PC) with a FAT32 (or FAT16) file system. Next, a directory named "CYAN" must be created in the root directory. Please note that the example application requires that this directory exists as it is used for its access routines, thus it should be created after formatting the CF memory card. The CF card may then be inserted into a running eCOG1k system hosting the example application.

This example application makes use of the FAT file system support library software for the eCOG1k to handle file accesses, as described in application note AN042 [3].

The FAT library employs many recursive functions. As such, the user must manually increase the stack size in the file *cstartup.asm* to 1Kbytes (double the default value) to accommodate these recursive function calls.

#### 5.1.2 Application Operation

When a newly formatted CF memory card is inserted, the application creates a text file "COUNT.TXT" in the "CYAN" directory and writes the text "0001" into it. On subsequent card insertions, the application looks for this file and increments the count value. This text file on the card can be viewed and edited with a memory card reader.

When a CF memory card is inserted, an interrupt is generated on the falling edge of the card detect signal. The interrupt service routine then sets a global flag that instructs the firmware to perform a soft reset on the card. The firmware then queries the identification and configuration of the CF memory card to initialise the relevant variables.

Next, a search for the file "COUNT.TXT" is performed. This file is read into a buffer and the count value content is incremented. The updated value is then rewritten back into the "COUNT.TXT" file on the CF memory card. All these operations rely on multiple calls to the CF memory card sector read and sector write routines.

#### 5.1.3 Code Size and Memory Usage

The example application contains both the CF memory card library and the FAT library.

The total combined program and data space (consisting of code segments and the constant data segment) is approximately 20 Kbytes. In terms of memory usage, the application variables in RAM and scratchpad RAM take up 412 bytes. These values can be inferred from the CyanIDE report file *cf.sec*.

The FAT library uses several buffers that require a total of 1.5Kbytes of memory, while the CF library uses a 512 byte buffer that is taken from and shared with that of the FAT library buffer. In addition, a stack size of 1Kbytes is required.

## 5.2 Performance

The typical read throughput of the CF memory card read access routine is determined by transferring a sector of 512 bytes data from the CF memory card flash memory into its internal buffer, and then issuing an ATA read command to transfer this data to the host buffer. This process is repeated 2048 times (thus reading a total of 1Mbyte), with the data read from consecutive sector address of the CF memory card.

Conversely, the typical write throughput of the CF memory card write access routine is determined by transferring a sector of 512 bytes data from the host buffer into the CF memory card internal buffer, and then issuing an ATA write command to transfer this data to the CF flash memory. This process is again repeated 2048 times (giving a total of 1Mbyte), transferring the same sector data but writing to consecutive sector address in the CF memory card.

The obtained result taking the best-case value in each case is shown in Table 2.

CF Memory Card Type	Access Routines	Throughput
SanDisk 64 Mbytes	Read access	564.87 Kbytes/s
	Write access	326.53 Kbytes/s
SanDisk 256 Mbytes	Read access	480.94 Kbytes/s
	Write access	252.31 Kbytes/s
Memorex 64 Mbytes	Read access	298.34 Kbytes/s
	Write access	262.13 Kbytes/s

**Table 2. CF memory card access routines raw throughput.**

## 6 Design Limitations

The example application entails the hot insertion of the CF memory card into a running eCOG1k system hosting the example application, in which the CF memory card is physically connected directly to the eCOG1k. Ideally, the eCOG1k address, data and control lines need to be isolated from the transients (upon card insertion) produced by the CF memory card when it is undergoing a power-on reset and also while its internal pin capacitors are being charged.

Common mitigation approaches use buffers or bidirectional bus transceivers to isolate the signal lines involved, which would place these signal lines in a high-impedance state until the power-on operations are completed. These operations can take up to several hundreds of milliseconds after card insertion.

## 7 References

1. CF+ and CompactFlash Specification Revision 3.0  
<http://www.compactflash.org/>
2. SanDisk CompactFlash Memory Card Product Manual  
<http://www.sandisk.com/Assets/File/OEM/Manuals/ProdManCFlashv11.0.pdf>
3. AN042 - FAT32 (16) File System Support for the eCOG1k  
[http://www.cyantechology.com/support/app\\_notes.php](http://www.cyantechology.com/support/app_notes.php)

## Appendix A CF Memory Card API Functions

---

```
unsigned int cfChkRdyStatus(void);
```

Pool the busy and ready status of the CF memory card. The function exits when the CF memory card is not busy and ready, or when timeout occurs following a number of queries. The busy bit is 0 when the host is allowed to access the command registers and buffer, while the ready bit is 1 when the CF is ready to accept a command.

### Parameter

-	
---	--

### Return

1 if timeout variable expires.

### Example

-

---

```
unsigned int cfChkDrqStatus(void);
```

Poll the data request status of the CF memory card. The function exits when the CF memory card is ready for data transfer, or when timeout occurs following a number of queries. The data request bit is 1 when the CF requires that information be transferred either to or from the host through the data register.

### Parameter

-	
---	--

### Return

1 if timeout variable expires.

### Example

-

---

```
unsigned int cfSoftReset(void);
```

Soft reset the CF memory card, leaving the CF in the same non-configured, reset state as following power-up and hardware reset.

### Parameter

-	
---	--

### Return

0 if CF is ready following reset.

### Example

-

---

```
void cfRdDataBuffer(unsigned int *ptrData);
```

Transfer a sector of 512 bytes data from the CF memory card internal data buffer to the pointer location.

**Parameter**

<i>*ptrData</i>	Pointer to the buffer used to store the read data
-----------------	---

**Return**

Contents of CF memory card internal buffer at pointer location.

**Example**

-

---

```
void cfWrDataBuffer(unsigned int *ptrData);
```

Transfer a sector of 512 bytes data to the CF memory card internal data buffer to the pointer location.

**Parameter**

<i>*ptrData</i>	Pointer to the buffer used to store the write data.
-----------------	---

**Return**

Contents of pointer location in CF memory card internal data buffer.

**Example**

-

---

```
void cfPhySecToCHS(
    unsigned long physicalSector,
    unsigned int *ptrCylinder,
    unsigned int *ptrHead,
    unsigned int *ptrSector);
```

Translate the given physical sector address of a CF memory card into the equivalent C/H/S address, as the CF memory card is operated in C/H/S addressing mode. Global variables of parameter information on the CF memory card must be initialised prior to executing this function.

**Parameter**

<i>physicalSector</i>	CF memory card physical sector address.
<i>*ptrCylinder</i>	Pointer to the storage for the resulting cylinder value.
<i>*ptrHead</i>	Pointer to the storage for the resulting head value.
<i>*ptrSector</i>	Pointer to the storage for the resulting sector value.

**Return**

The translated C/H/S address in the respective variables.

**Example**

-

---

**void cfATACommand**

Provide a common interface for issuing ATA command to the CF memory card.

**Parameter**

<i>physicalSector</i>	CF memory card physical sector address
<i>sectorCount</i>	Number of sectors of data requested to be transferred on a read or write operation.
<i>ATACommand</i>	ATA command to be issued.

**Return**

-

**Example**

-

---

**unsigned int cfIdDrive (unsigned int \*ptrData);**

Obtain CF parameter information from the CF memory card. The relevant global variables are initialised with this information.

**Parameter**

<i>*ptrData</i>	Pointer to the buffer used to store the read parameter information.
-----------------	---

**Return**

0 if operation is successful.

CF parameter information at pointer location. The global variables of number of cylinders, heads and sectors per track are also initialised.

**Example**

-

---

```

unsigned int cfRdSector(
    unsigned long physicalSector,
    unsigned int sectorCount,
    unsigned int *ptrData);

```

Read a sector of 512 bytes data from a memory location in the CF memory card and store to the pointer location.

#### Parameter

<i>physicalSector</i>	CF memory card physical sector address to be read from.
<i>sectorCount</i>	Number of sectors of data requested to be transferred on read operation.
<i>*ptrData</i>	Pointer to the buffer used to store the read data

#### Return

0 if operation is successful.

Contents of memory location of the CF memory card at pointer location.

#### Example

```
cfRdSector(sector, 1, buffer); // Read 1 sector of data into buffer
```

---

```

unsigned int cfWrSector(
    unsigned long physicalSector,
    unsigned int sectorCount,
    unsigned int *ptrData);

```

Write a sector of 512 bytes data to a memory location in the CF memory card with source data from the pointer location.

#### Parameter

<i>physicalSector</i>	CF memory card physical sector address to be written to.
<i>sectorCount</i>	Number of sectors of data requested to be transferred on write operation.
<i>*ptrData</i>	Pointer to the buffer used to store the write data

#### Return

0 if operation is successful.

#### Example

```
cfWrSector(sector, 1, buffer) // Write buffer to the specified sector
```