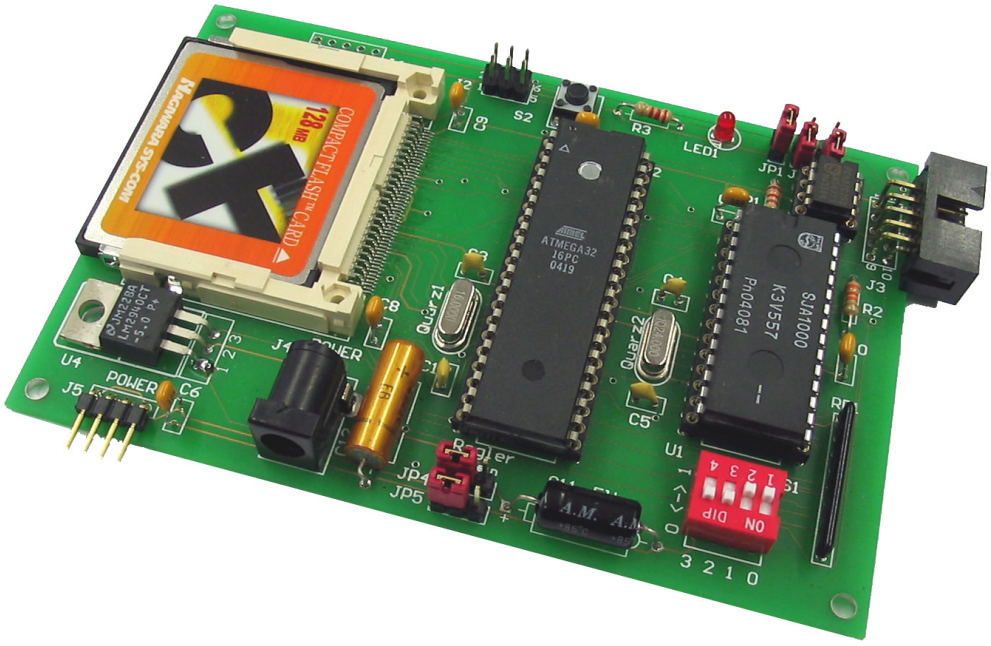
Nutzerhandbuch CF-Modul

Version 1.00



Fachhochschule Brandenburg, Fachbereich Informatik und Medien, 2005

Inhaltsverzeichnis

1 Einleitung

Das CompactFlash - Speichermodul (kurz CF-Modul) ist Teil des RCUBE Systems. Es bietet die Möglichkeit, im autonomen Betrieb größere Datenmengen auf einem FAT16-Dateisystem der CompactFlash-Karte zu speichern bzw. zu lesen. Damit können Sensor-, Lern- oder andere Daten über einen Stromausfall hinweg erhalten werden. Die Daten der CF-Karte lassen sich in einem üblichen Kartenlesegerät in den PC einlesen.

Die Verbindung mit den anderen Boards erfolgt über den CAN-Bus. Andere Boards des Systems können so über CAN-Pakete Daten lesen oder schreiben.

1.1 Funktionsumfang

Auf dem CF-Modul ist ein rudimentärer FAT-16 Dateisystemdienst implementiert. Folgende Dateioperationen werden unterstützt:

- Lesen von Sektoren
- Schreiben von Sektoren
- Lesen von Dateien
- Erzeugen von Dateien
- Schreiben von Dateien
- Anhängen an existierende Dateien
- Verzeichnisliste (ls)
- Verzeichniswechsel (cd)
- Herstellerinformation des CompactFlash-Mediums

1.2 Einschränkungen

Es ist das FAT-16-Dateisystem implementiert. Zu jedem Zeitpunkt kann immer nur eine Datei geöffnet sein - d.h. ein Mehrfachzugriff durch mehrere Prozesse ist nicht möglich. Das Formatieren des Dateisystems und das Löschen von Dateien sind nicht möglich, so dass die Karte vorher am PC formatiert werden sollte.

2 Hardware

2.1 Aufbau

Das Board besteht aus einer ATMEL-AVR-Mega32 CPU mit 32 kByte Flash und 2 kByte RAM, dem CompactFlash-Speicher und den CAN-Bus Interface. Speichermodule bis 128 Mbyte wurden im Betrieb getestet.

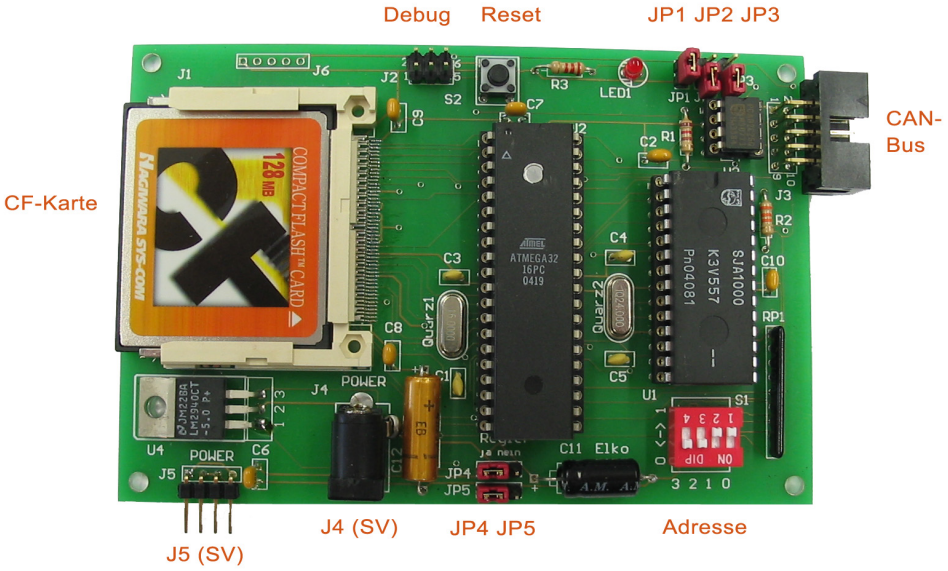


Abbildung 2.1: Anschlüsse des CF-Moduls

2.2 Konfiguration

Im Folgenden wird die Bedeutung der Jumper beschrieben, die Beschreibung der Einstellung der CAN-Adresse ist im Kapitel 3 zu finden.

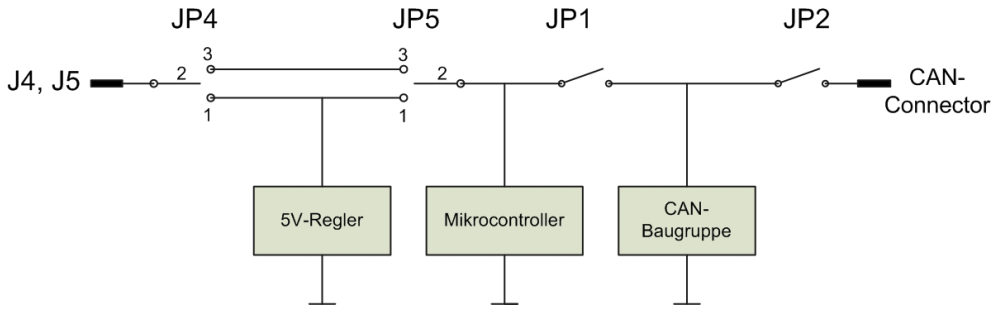


Abbildung 2.2: Jumper des CF-Moduls bezüglich Spannungsversorgung

JP1	JP2	Funktion
0	0	CAN-Baugruppe abgeschaltet
0	1	CAN-Baugruppe wird vom CAN-Connector versorgt
1	0	CAN-Baugruppe wird vom Modul versorgt
1	1	Spannung des Boards wird mit dem CAN-Connector verbunden Zwei Varianten: 1. Das Modul hat keine eigene Stromversorgung über J4 oder J5 und wird von einem anderen Board mit 5V versorgt. Hierbei auch den Spannungsregler abschalten (siehe JP4, JP5). 2. Das Board hat eine eigene Stromversorgung über J4 oder J5 und speist seine Spannung 5V in den CAN-Connector, so dass sich andere Boards über ihren CAN-Connector versorgen können. **Achtung:** Bei beiden Varianten ist darauf zu achten, dass nur ein einziges Board in den CAN-Connector einspeist.

Tabelle 2.1: JP1, JP2 - Anschluss des Boards an 5V-Leitung des CAN-Connectors

JP3	Funktion
0	Kein Abschlusswiderstand
1	Abschlusswiderstand aktiv

Tabelle 2.2: JP3 - Abschlusswiderstand CAN-Bus

JP4	JP5	Funktion
1-2	1-2	Spannungsregler zugeschaltet: An J4 oder J5 kann eine Spannung von 5.5 bis 9 V angelegt werden
2-3	2-3	Spannungsregler abgeschaltet: An J4 oder J5 kann eine Spannung von 5V angelegt werden (z.B. vom AKSEN)

Tabelle 2.3: JP4, JP5 - Anschluss 5V-Spannungsregler

3 CAN-Interface

Die CAN-Nachrichten benutzen das erweiterte ID-Format (29 Bit), jede Nachricht kann 8 Byte Nutzdaten enthalten. Über diese Grundkommunikationsstruktur wurde ein Push-Poll Protokoll für die Dateisystemzugriffe implementiert, das in der Regel aus einer Folge von CAN-Basisnachrichten besteht. Der allgemeine Aufbau sieht wie folgt aus.

Listing 3.1: CF_MSG_HDR

```
1
2  struct CF_MSG_HDR {
3     unsigned char cmd;          // Kommando
4     unsigned char retAdr;       // Partneradresse
5     unsigned int  lng;         // wenn Länge != 0, folgen Datenbytes in
6                                 // weiteren CAN-Nachrichten
7     unsigned int  cs;          // Prüfsumme
8     unsigned char info[3];      // 3 Byte Info
9  };
```

Die ID einer Nachricht an das CompactFlash-Board muss in den 5 höchsten Bits (ID.28 - ID.24) die Adresse des CompactFlash-Boards enthalten, die 8 niedrigsten Bits (ID.7 - ID.0) werden durch das Kommando *cmd* der oben angegebenen Struktur gebildet. Die Adresse ergibt sich aus der Stellung der 4 DIP-Schalter, wobei ID.28 immer 0 ist, ID.27 dem 4. Schalter entspricht, u.s.w..

Die Bits ID.28 - ID.8 müssen der Adresse entsprechen, sonst wird die Nachricht durch Filterregeln im CAN-Controller verworfen. Die Bits ID.7 - ID.0 können beliebig gesetzt sein, sie werden immer von dem CAN-Controller empfangen und an den Microcontroller zur Auswertung übergeben.

Das Feld *retAdr* gibt die CAN-ID an, die für die Rückantwort benutzt wird. Es werden nur die unteren 4 Bits genutzt und in die Bits ID.28 - ID.24 der CAN-ID analog zu oben geschrieben.

Die Länge *lng* ist die Datenlänge der Nachricht ohne die 8 Byte der ersten Nachricht, d.h. eine Nachricht mit der Länge 0 besteht aus genau einer CAN-Basisnachricht.

Die Prüfsumme *cs* ergibt sich als 16-Byte Summe über die Bytes der gesamten Nachricht.

Das 3-Byte Infofeld *info[3]* wird für Kurzinformationen benutzt, die in einer CAN-Basisnachricht gesendet werden.

Die folgende Tabelle listet alle verfügbaren Kommandos.

Zu jedem dieser Kommandos gibt es eine Antwort-Nachricht, diese hat jeweils zusätzlich das Bit 7 gesetzt (z.B. *CF_INIT_RESP 0x89*).

Name	Code
INFO	2
ROOT	4
LIST_DIR	5
READ_SECTOR	6
WRITE_SECTOR	8
CF_INIT	9
CF_OPEN	0xa
CF_READ	0xb
CF_WRITE	0xc
CF_CLOSE	0xd
CHANGEDIR	0xe

Tabelle 3.1: Kommandos des CF-Moduls

3.1 CAN-Kommunikation

Die Kommunikation mit dem CompactFlash-Modul besteht aus einem einfachen Push-Pull- Protokoll. Zur Zeit wird von einer fehlerfreien Kommunikation ausgegangen, das Wiederaufsetzen einer abgebrochenen Kommunikationsfolge ist nicht möglich. Das Compact-Flash-Modul arbeitet die einzelnen Nachrichten im Polling-Verfahren ab und bedient aufgrund der Kommunikation die CompactFlash-Karte. Es ist möglich, durch sehr schnelles Senden von CAN-Nachrichten den Microcontroller zu überfordern - hier sind einige Tests erforderlich. Das mitgelieferte Testprogramm für das AKSEN-Board zeigt beim Senden von CAN-Nachrichten die notwendigen Verzögerungen. Im folgenden sind nicht die CAN-Nachrichten beschrieben, sondern die mitgelieferte Bibliothek, die notwendigen CAN-Nachrichten sind daraus leicht im Quellcode des Testprogramms ersichtlich.

3.2 Bibliothek

Die im folgenden beschriebenen Bibliotheksfunktionen wurden für das AKSEN-Board implementiert. Das Interface ist auf die absolut notwendige Funktionalität beschränkt. Auch die Größe des internen RAM bedeutet einige Einschränkungen an die maximal mögliche Nachrichtenlänge. Ein beispielhaftes Testprogramm ist mitgeliefert.

3.2.1 Init CompactFlash Board

Das CompactFlash-Board kann durch den Aufruf CFinit() neu initialisiert werden.

Aufruf: *unsigned char CFinit (unsigned int debug)*

Der übergebene Wert steuert Debug-Verhalten und künstliche zeitliche Verzögerungen beim Sendevorgang des CompactFlash-Boards.
Oberes Byte = 0 : Keine Debug-Ausgaben

Oberes Byte = 1 : Debug-Ausgaben über eine 115200 Baud-Schnittstelle.
Unteres Byte: zeitliche Verzögerung beim Senden: falls Debug nicht gesetzt ist, gibt der Wert die Verzögerung in 100 usec-Einheiten an, im Fall, dass Debug gesetzt ist, wird der angegebene Wert als Verzögerung in 1 millisec-Einheiten interpretiert (wegen der Debugausgaben sind größere Verzögerungen notwendig).

Für das AKSEN-Board als Gegenstelle ist in dem Beispielprogramm etwa 70 in beiden Fällen notwendig, dieses entspricht 7 millisec bzw. 70 millisec (Debug aus / Debug an) als künstliche Sendeverzögerung.

Rückgabewert: TRUE - ok , sonst FALSE, wobei TRUE = 1 und FALSE = 0 definiert sind

Testprogramm: Eingabe von *i*;

3.2.2 Goto Root Directory

Das Kommando bewirkt, das das aktuelle Verzeichnis das Wurzelverzeichnis ist - entsprechend dem UNIX-Kommando *cd /*.

Aufruf: *unsigned char gotoRootDir (void)*

Rückgabewert: TRUE - ok , sonst FALSE, wobei TRUE = 1 und FALSE = 0 definiert sind

Testprogramm: Eingabe von *r*;

3.2.3 Change Directory

Das Kommando bewirkt den Wechsel in das nächsthöhere oder -niedere Verzeichnis. Ein zusammengesetzter Pfad kann nicht angegeben werden, sondern muß durch mehrfache Aufrufe emuliert werden. Der Übergabeparameter zeigt auf eine String, der das Pfadelement enthält. Entsprechend der MSDOS-Konvention kann der Name nur aus 8.3 Zeichen bestehen - d.h. 12 Zeichen inkl. des Punktes.

Aufruf: *unsigned long changeDir (unsigned char * pathname)*

Rückgabewert: Die aktuelle Clusternummer des neuen Verzeichnisses, "0ïm Fehlerfall.

Testprogramm: Eingabe von cdDATEI ohne Leerzeichen, auch cd..ïst möglich.

3.2.4 Open CompactFlash File

Eine Datei des aktuellen Verzeichnisses wird geöffnet. Es kann mithilfe von *type* eine Lese- oder Schreibzugriff bestimmt werden. Gibt es die Datei noch nicht, wird sie angelegt. Der Positionszeiger wird beim Öffnen immer an den Anfang gesetzt Das Erzeugungdatum ist immer gleich (z.Z. 10.5.03), da das CF-Modul keine Uhr besitzt. Es kann immer nur eine Datei zur Zeit geöffnet sein.

Aufruf: *unsigned char CFopen (unsigned char* filename, unsigned int type)*

F_READ = 1 / F_WRITE = 2

Rückgabewert: TRUE - ok , sonst FALSE

3.2.5 Close CompactFlash File

Die aktuelle geöffnete Datei wird geschlossen, der letzte geschriebene Sektor wird geschrieben..

Aufruf: *unsigned char CFclose (void)*

Rückgabewert: TRUE - ok , sonst FALSE, wobei TRUE = 1 und FALSE = 0 definiert sind

3.2.6 Read File

Aus des aktuell geöffneten Datei werden *laenge* Bytes in den angegebenen Puffer gelesen. Die Position in der Datei verschiebt sich entsprechend.

Aufruf: *unsigned char CFread (unsigned char * puffer, unsigned int laenge, unsigend int * returnLaenge)*

Rückgabewert: TRUE - ok , sonst FALSE, wobei TRUE = 1 und FALSE = 0 definiert sind *returnLaenge*: tatsächlich gelesene Anzahl Bytes

Testprogramm: Eingabe von *rfDATEI* gibt die gesamte Datei auf dem Bildschirm aus. Es werden dafür *Cfopen()*, *Cfread()* und *Cfclose* benutzt.

3.2.7 Write File

An die aktuell geöffnete Datei wird Inhalt des *Puffers* mit der *laenge* Bytes angehängt.

Aufruf: *unsigned char Cfwrite(unsigned char * puffer, unsigned int laenge, unsigend int * returnLaenge)*

Rückgabewert: TRUE - ok , sonst FALSE, wobei TRUE = 1 und FALSE = 0 definiert sind *returnLaenge*: tatsächlich gelesene Anzahl Bytes

Testprogramm: Eingabe von *wfDATEI* erzeugt eine große Datei, die eine ASCII-Zeichenfolge enthält. Es werden dafür *Cfopen()*, *Cfwrite()* und *Cfclose* benutzt.

3.2.8 Sektor lesen

Ein einzelner Sektor des Dateisystems wird gelesen, das Ziel ist ein Puffer, der 512 Bytes groß sein muss.

Aufruf: *unsigned char CFReadSector (unsigned long sectorNr, unsigned char * puffer)*

Rückgabewert: TRUE - ok , sonst FALSE, wobei TRUE = 1 und FALSE = 0 definiert sind

Testprogramm: rs12345 - liest den entsprechenden Sektor aus. Abbildung 3.1 gibt die Ausgabe des Testprogramm nach *rs32* an:

3.2.9 Sektor schreiben

Ein einzelner Sektor des Dateisystems wird geschrieben, die Quelle ist ein Puffer, der 512 Bytes groß sein muss.

Aufruf: *unsigned char CFReadSector (unsigned long sectorNr, unsigned char * puffer)*

Rückgabewert: TRUE - ok , sonst FALSE, wobei TRUE = 1 und FALSE = 0 definiert sind

Testprogramm: ws12345 - beschreibt den entsprechenden Sektor (hier zum Test mit den Daten 0 - 255).

3.2.10 Dateiverzeichnis ausgeben

Das aktuelle Dateiverzeichnis wird in einem festen Format ausgegeben, der Aufruf liest max 512 Bytes in den angegebenen Puffer, d.h. der angegebene Puffer muss eine Größe von 512 Bytes

```
EB3C904D53444F53352E300002020200  ë<□MSDOS5.0.....
0200020000F8F7003F00FF0020000000  .....ø÷.?.ÿ. ...
E0EF0100000029C66D52404E4F204E41  àï....)ÆmR@NO NA
4D4520202020464154313620202033C9  ME    FAT16   3É
8ED1BCF07B8ED9B800208EC0FCBD007C  □Ñ¼ð{□Ù¸. □Àü½.|
384E247D248BC199E83C01721C83EB3A  8N$}$‹Á™è<.r.ƒë:
66A11C7C26663B07268A57FC750680CA  f¡.|&f;.&ŠWüu.€Ê
0288560280C31073EB33C98A461098F7  .^V.€Ã.së3ÉŠF.˜÷
661603461C13561E03460E13D18B7611  f..F..V..F..Ñ‹v.
608946FC8956FEB82000F7E68B5E0B03  `‰Fü‰Vþ¸ .÷æ‹^..
C348F7F30146FC114EFE61BF0000E8E6  ÃH÷ó.Fü.Nþa¿..èæ
00723926382D741760B10BBEA17DF3A6  .r9&8-t.`±.¾¡}ó¦
6174324E740983C7203BFB72E6EBDCA0  at2Nt.fÇ ;ûræëÜ
FB7DB47D8BF0AC9840740C487413B40E  û}´}‹ð¬˜@t.Ht.´.
BB0700CD10EBEFA0FD7DEBE6A0FC7DEB  »..Í.ëï ý}ëæ ü}ë
E1CD16CD19268B551A52B001BB0000E8  áÍ.Í.&‹U.R°.»..è
3B0072E85B8A5624BE0B7C8BFCC746F0  ;.rè[ŠV$¾.|‹üÇFð
3D7DC746F4297D8CD9894EF2894EF6C6  =}ÇFô)}ŒÙ‰Nò‰NöÆ
06967DCBEA030000200FB6C8668B46F8  .–}Ëê... .¶È f‹Fø
6603461C668BD066C1EA10EB5E0FB6C8  f.F.f‹ÐfÁê.ë^.¶È
4A4A8A460D32E4F7E20346FC1356FEEB  JJŠF.2ä÷â.Fü.Vþë
4A525006536A016A10918B4618969233  JRP.Sj.j.‘‹F.–’3
D2F7F691F7F64287CAF7761A8AF28AE8  Ò÷ö‘÷öB‡Ê÷v.ŠòŠè
C0CC020ACCB80102807E020E7504B442  ÀÌ..Ì¸..€~..u.´B
8BF48A5624CD136161720B4075014203  ‹ôŠV$Í.aar.@u.B.
5E0B497506F8C341BB000060666A00EB  ^.Iu.øÃA»..`fj.ë
B04E544C44522020202020200D0A4461  °NTLDR      ..Da
74656E7472846765722065 6E74666572  tentr„ger entfer
6E656EFF0D0A4D656469656E6665686C  neny..Medienfehl
6572FF0D0A4E6575737461727 43A2054  erÿ..Neustart: T
617374652064728 1636B656E0D0A0000  aste dr□cken....
00000000000000000000000ACC4D355AA  ...........¬ÄÓUª
```

Abbildung 3.1: Gelesener Sektor

haben. Ist der Rückgabewert *more* nicht 0xFF, sind noch nicht alle Daten des Verzeichnisses gelesen worden und der Aufruf soll wiederholt werden.

Aufruf: *unsigned char listDir (unsigned char * more, char *puffer)*

Rückgabewert: TRUE - ok , sonst FALSE, wobei TRUE = 1 und FALSE = 0 definiert sind

more : Nicht 0xFF: weitere Daten lesbar; 0xFF: Ende der Verzeichnisdaten erreicht.

Testprogramm: ls gibt das Verzeichnis in einem festen Format aus, es wird der Dateityp, die Attribute, das Erzeugungsdatum, die Länge in Bytes und das Anfangscluster ausgegeben.

```
D .            ----  04.04.04  08:51 <DIR>      Clu:         2
D ..           ----  04.04.04  08:51 <DIR>      Clu:         0
F BASIC.LST    ---A  02.04.04  16:40     24399  Clu:         3
F CANC~1.BAK   ---A  02.04.04  18:16     18209  Clu:        27
F CAN~1.C_S    ---A  03.04.04  11:18        73  Clu:        45
F CANH~1.BAK   ---A  02.04.04  15:43      4137  Clu:        46
F CAN~1.H_S    ---A  28.03.04  15:32        73  Clu:        51
F CAN.C        ---A  04.04.04  08:38     18210  Clu:        52
F CAN.D        ---A  04.04.04  08:38        55  Clu:        70
F CAN.H        ---A  02.04.04  15:46      4159  Clu:        71
F CAN.O        ---A  04.04.04  08:38     18236  Clu:        76
```

```
F SIO.C        ---A  01.04.04  23:38      2369  Clu:        94
F SIO.D        ---A  02.04.04  16:39        38  Clu:        97
```

3.2.11 CompactFlash Information ausgeben

Die auf der CompactFlash gespeicherte Herstellerinformation wird ausgegeben, der angegebene Puffer muss eine Größe von 512 Bytes haben.

Aufruf: *unsigned char ReadDriveInformation (unsigned char * more, unsigned char *puffer, unsigned int * laenge)*

Rückgabewert: TRUE - ok , sonst FALSE, wobei TRUE = 1 und FALSE = 0 definiert sind *more* : Nicht 0xFF : weitere Daten lesbar; 0xFF: Ende der Informationen erreicht. *laenge*: die tatsächlich bereitgestellte Anzahl Bytes.

Testprogramm: *f* gibt die CF-Herstellerinformation in einem festen Format aus. Die hier wiedergegebene Ausgabe entstand durch eine 64 MByte Samsung-CompactFlash.

```
CF-Modul gk 3/2004 V1.0
SAMSUNG CF/ATA // Herstellerangaben
S1.18.4 // Revisionstext/Seriennummer
C/H/S 248/16/126976 //Cluster/Head/Sektoren
63488kB // Größe
00010100060F20F720000000E0EF0100 // Partitionstabelle
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
Boot: 0x0
Typ: 0x6 // Dateisystemtyp
Btsec: 32 // Bootsektoren
S/C: 2 // Sektoren pro Cluster
Res: 2 // Reserve
FATs: 2 // Anzahl FATs
S/FAT: 247 // Sektoren / FAT
#Secs (16Bit): 0  (32Bit): 126944 // Nutzbare Sektoren
RDE 512 // Root Directory Entry
FRS: 528 // First Root Sector
FDS: 560 // First Data Sector
DS: 126416 // Data Sectors of Partition
DC: 63208 // Data Clusters of Partition
1.FAT: 34 // 1. FAT Sector
FAT16 // FAT-Type
OEM: MSDOS5.0 // BootSector - OEM-Name
```