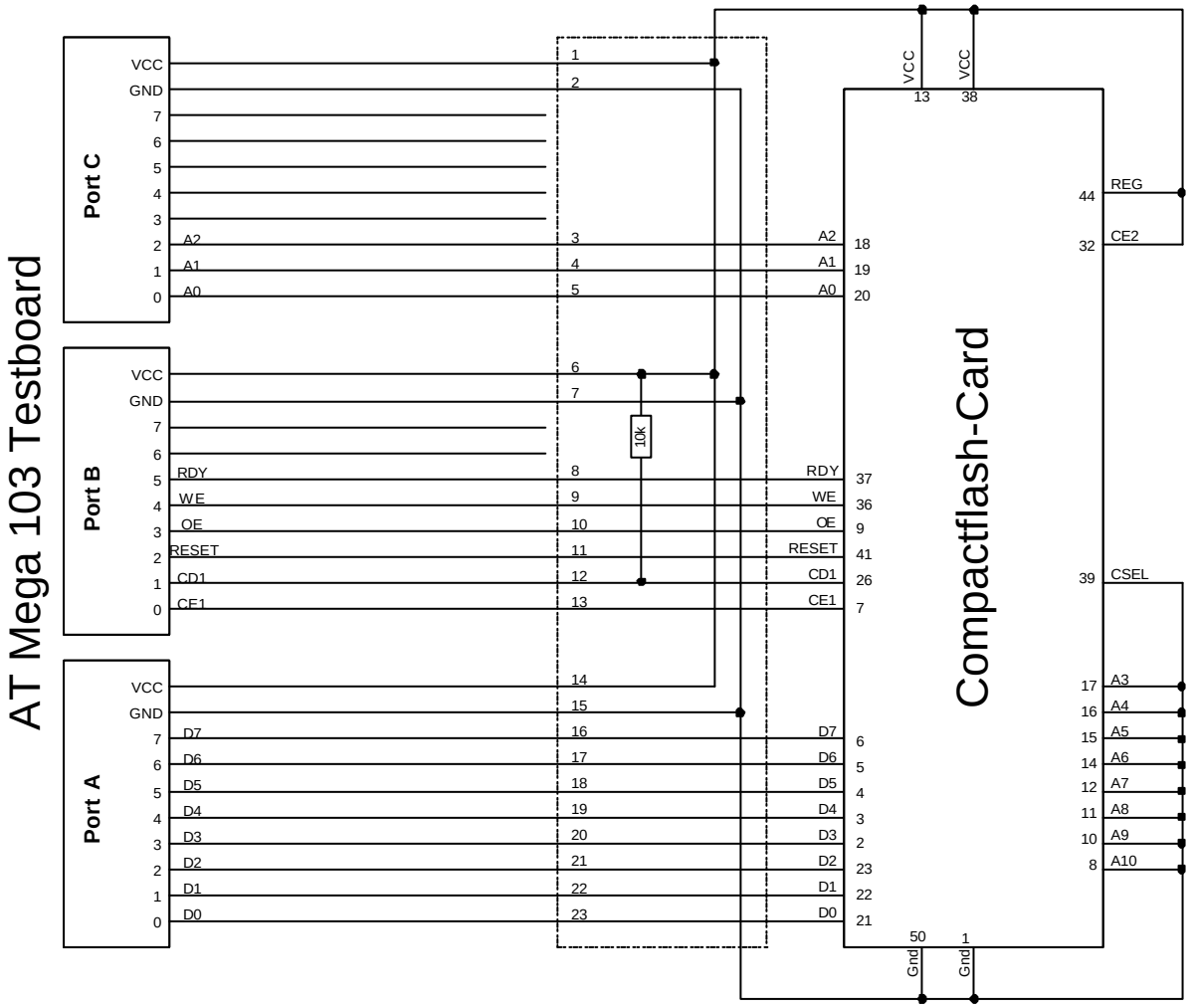
Accessing a Compact Flash Card from BASCOM

After the development of my glass house (Wintergarten) controlling system, I wanted collect the climatic data, such as outside and interior temperature, sunshine intensity and rain in order to be able to evaluate it later, building some statistisc, diagrams and so on.
First I send the data online to a terminal program on a PC and stored it there in a text-file. But this solution have the disadvantage, that I need always a PC (or Laptop) on the embedded system. Next I thougth about to store the data in a comprimized form in the EEPROM on the ATMega103, but even the 4 kByte EEPROM can hold only the data of few days.
So I searched the Internet for other solution for long time logging on an embedded system.
I found a very interesting article for embedded systems on
http://www.circuitcellar.com/echips-pdfs/0201/c0201mspdf.pdf about to store data in a compact flash card.
With a CompactFlash Card you have a huge amount of memory to a low price.
The Compactflash Memory Card can be used in three mode:

- Memory Mode
- I/O Mode
- IDE Mode

I decided to use it in **Memory Mode** to keep things simple and use lowest count of pins of the controller to attach the flashcard.
The 17 pins are 8 for data, 3 for register addressing and 6 for controlling.
You can see in the schematic, that the connection is very simple, only one pullup resistor is needed.



I used a ATMEGA103 Test-Board from RIBU Electronic Austria. But it should be no problem to use any other developing system. This boards provides VCC and Gnd on every port, but extra lines for VCC and Gnd will do it as well.

I transfered the PIC-Routines from the above mentioned article to AVR and build it modular, so they can be used in a easy way from BASCOM.

The routines are:

- **ReadSector**: Read one or more Sectors (512 Bytes) from the CF-Card to SRAM
- **WriteSector**: Write one or more Sectors (512 Bytes) from SRAM to the CF-Card
- **GetDriveIdentity**: Read CF-Card Identity Information to SRAM
- **DriveInit**: Set Input and Out-Ports in Controller and resets the CF-Card
- **DriveReset**: Resets the CF-Card
- **DriveCheck**: Checks, if CF-Card is plugged in

I wrote the routines in ASM to save code-space and execution time.
This functions are located in FlashCardDrive.Bas. There are the ASM-routines and the BASCOM-Interface routines, which allows to call the ASM-routine from BASCOM as normal Sub of Function.

Now the Usage of the Functions/Subs:

Sub **ReadSector** (SRAMPointer, SectorNumber, SectorCount)

SRAMPointer (Type Word): Pointer to memorylocation to which the transfer from the CF-Card is written
SectorNumber (Type Long): Sectornumber on the CF-Card
SectorCount (Type Byte): Count of sectors (each 512 bytes) to read from the CF-Card (hightest value is 127, which can be handled by the routine)

Reads 1 to 127 Sectors from the CF-Card and stores it in the SRAM starting at a desired address.

Example:

```
$Include "FlashCardDrive.bas"

Dim TransferBuffer(512) as Byte          ' Hold Sector to and from CF-Card
Dim wSRAMPointer as Word                 ' Address-Pointer for read and write
Dim lSectorNumber as Long                ' Sector Number

DriveInit           ' Set pins to CF-Card and reset card, only needed at start

' give Address of first Byte of the 512 Byte Buffer to Word-Variable
wSRAMPointer = VarPtr(TransferBuffer(1))

' Set Sectornumber, sector 32 normally holds the Boot record sector of first
' partition
lSectorNumber = 32

' Now read in sector 32 (1 Sector) from CF-Card
ReadSector wSRAMPointer , lSectorNumber , 1
' Now Sector number 32 is in Byte-Array TransferBuffer
```

Sub **WriteSector** (SRAMPointer, SectorNumber, SectorCount)

SRAMPointer (Type Word): Pointer to memorylocation from which the transfer to the CF-Card is written
SectorNumber (Type Long): Sectornumber on the CF-Card

SectorCount (Type Byte): Count of sectors (at 512 bytes) to read from the CF-Card (Hightest value is 127, which can be handled by the routine)

Writes 1 to 127 Sectors to the CF-Card from the SRAM starting at a desired address.

Example:

```
$Include "FlashCardDrive.bas"

Dim TransferBuffer(512) as Byte          ' Hold Sector to and from CF-Card
Dim wSRAMPointer as Word                 ' Address-Pointer for read and write
Dim lSectorNumber as Long                ' Sector Number

DriveInit           ' Set pins to CF-Card and reset card, only neede ar start

' give Address of first Byte of the 512 Byte Buffer to Word-Variable
wSRAMPointer = VarPtr(TransferBuffer(1))

' Set Sectornumber to 2
lSectorNumber = 2

' Now write Content of Byte-Array TransferBuffer to CF-Card at Sector 2
ReadSector wSRAMPointer , lSectorNumber , 1
```

Sub **GetDriveIdentity** (SRAMPointer)

SRAMPointer (Type Word): Pointer to memorylocation to which the transfer from the CF-Card is written

Store the special Card Information (512 Bytes) into SRAM starting at a desired address. Check Compact-FlashCard Specification for meaning of this information.

Example:

```
$Include "FlashCardDrive.bas"

Dim TransferBuffer(512) as Byte          ' Hold Sector to and from CF-Card
Dim wSRAMPointer as Word                 ' Address-Pointer for read and write
Dim lSectorNumber as Long                ' Sector Number

DriveInit           ' Set pins to CF-Card and reset card, only needed at start

' give Address of first Byte of the 512 Byte Buffer to Word-Variable
wSRAMPointer = VarPtr(TransferBuffer(1))

' Now read in Card Identity Information
GetCardIdentity wSRAMPointer
' Now 512 Byte of Card Identity Inforation is in Byte-Array TransferBuffer
```

Sub **DriveInit**

Inits the Input and Output – Ports of the Controller and resets the CF-Card

Sub **DriveReset**

Resets the CF-Card (Hardware-Reset)

Function **DriveCheck()** as Byte

Checks whether the CF-Card is plugged in (Pin CD1 is Low). It return 1, it CD1 is Low, otherwise it return 0

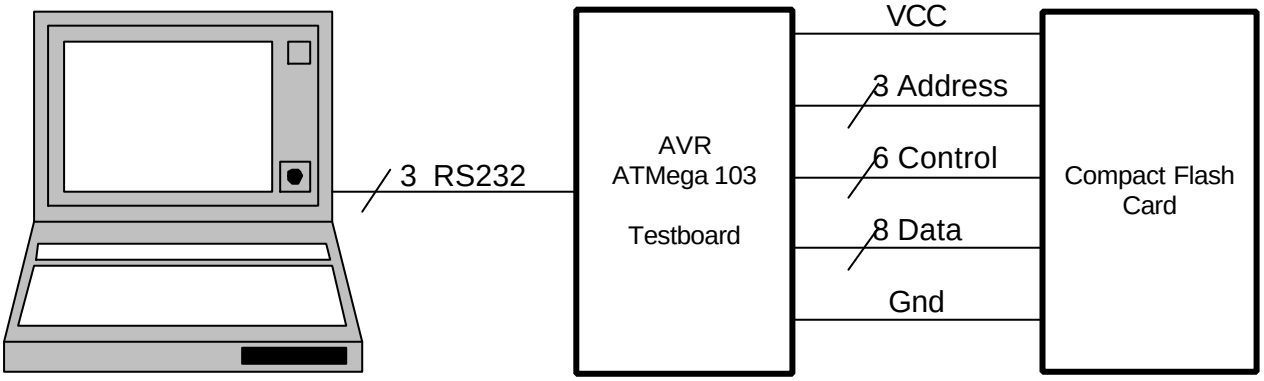
```
$Include "FlashCardDrive.bas"

if DriveCheck() = 1 then
   DriveInit
else
    print "Card not inserted, check Card!"
end if
```

A compact flash provide a huge array of 512-Byte Sectors. With the above mentioned routines reading and writing of every sectors is possible. A small card with 16 MB have for example approxemitely 31.250 Sectors.
I you want to use the previous described functions in a application, you have to $Include "FlashCardDrive.bas"

Testing enviroment

To test the Sub/Functions and get first experiences with the Compact-Flash Card i wrote an additional tiny interpreter, so the user can read and write sectors of the Card with short comands in a terminal program on a PC connected to the controller via a RS232. Preparing a test-sector in SRAM is also possible.



The Commands are:

CFI

*[**C**ompact **F**lash **I**dentity]: Reads the Card Identity Information and dumps it to the terminal. It shows also the available Number of Sectors on the Card and the Memory in Bytes*

CFR

*[**C**ompact **F**lash **R**eset]: Resets the Compactflash-Card*

MBR

*[**M**aster **B**oot **R**ecord]: Reads the first sector (sector 0) of the Card (Master Boot Record) and dumps it to the terminal. It also reads the partition table located in this sector and shows installed partitions with first sector, last sector and number of sectors in a partition and the code for the installed file system.*

SD <SectorNumber**>**

*[**S**ector **D**ump]: Read <Sectornumber> from Compactflash Card and dumps it to Terminal*

SD <SectorNumberStart> <SectorNumberEnd>

*[**S**ector **D**ump]: Dumps Sectors from <SectornumberStart> to <SectorNumbersEnd> to Terminal*

MD [<SRAMStart>] [<SRAMEnd>]

*[**M**emory **D**ump]: Dumps the SRAM Buffer to the Terminal. In the first 512 Bytes of the defined SRAM buffer(with 1024 Bytes to work with 2 sectos too) for the interpreter all readings from the CF are stored. With parameter <SRAMStart> every desired SRAM area can be dumped. With <SRAMEnd> it dumps to this address, otherwise it dumps 512 Bytes (1 Sector). The address shown at the beginning of the line is always relatively to the starting address.*

SW <SectorNumberStart> <SectorCount> [<SRAMPointer>]

*[**S**ector **W**rite]:Write <SectorCount> Sectors from SRAM to the Compactflash Card starting at <SectorNumberStart>. Without parameter <SRAMPointer> the defined SRAM-TransferBuffer is used, with the parameter <SRAMPointer> you can write from any SRAM memory-location.*

MB <Byte1> [<Byte2>] [<Byte2>] ..... [<Byte8>]

*[**M**emory **B**yte]: Write single Bytes to the Transfer Buffer. For <Byte> the ASCII – Code must be typed. For exampe 65 or $41 for Letter ‚A'. Writing to Transfer Buffer starts at address of MemroyPointer. Up to 8 Bytes are possible with one MB command.*

MF <Byte> [<BufferStart>] [<BufferEnd>]

*[**M**emory **B**yte]: Fill the Transfer Buffer with <Byte> from <BufferStart> to <BufferEnd>. For <Byte> the ASCII – Code must be typed. For exampe 65 or $41 for Letter ‚A'. Without <BufferStart> and <BufferEnd> the buffer from the MemoryPointer in the Transfer Buffer to End of the Buffer is filled. You can start also filling at <BufferStart> and end at <BufferEnd>.*

MT <Text>

*[**M**emory **T**ext]: Fill the SRAM-Transfer Buffer with <Text> starting at the position of the memorypointer.*

MP <MemoryPointer>

*[**M**emory **P**ointer]: Adjust the SRAM-Transfer Buffer Pointer for next writing with MB, MT or MF*
The actual Memory-Pointer is shown with the Prompt.

All values can written in decimal or hex. If hex is used the value must be preceeded by a $-sign like $3B.
With the commands MF, MB and MT you can prepare a test sector with desired content to write it to Compactflashcard and read it back.

A short example with some comands with a flash Cards which is 'Formatted' as a Harddisk.
User-input is **bold**, added comments are in **green.**

```
0000>cfi                          ' Get Card Identity Information

Read Card Info
0000  8A 84 D4 03 00 00 08 00 00 00 40 02 20 00 03 00   Š„Ô.......@. ...
0010  00 D4 00 00 20 20 20 20 30 30 32 33 39 30 31 47   .Ô..    0023901G
0020  30 32 4F 32 38 34 39 33 02 00 02 00 04 00 64 56   02O28493......dV
0030  20 67 2E 38 34 31 61 53 44 6E 73 69 20 6B 44 53    g.841aSDnsi kDS
0040  46 43 2D 42 32 31 20 38 20 20 20 20 20 20 20 20   FC-B21 8
0050  20 20 20 20 20 20 20 20 20 20 20 20 20 20 01 00                 ..
0060  00 00 00 02 00 00 00 01 00 00 01 00 D4 03 08 00   ............Ô...
0070  20 00 00 D4 03 00 00 01 00 D4 03 00 00 00 00 00    ..Ô.....Ô......
0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 20   ..............
0110  20 20 20 20 20 20 30 30 32 33 39 30 31 47 30 32         0023901G02
0120  4F 32 38 34 39 33 00 00 00 00 00 00 00 00 00 00   O28493..........
0130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
250880 Sectors = 128450560 Bytes
CF-Buffersize = 2 Sectors = 1024 Bytes

0000>mbr                          ' Get Master Boot Record (Sector 0)

Read Master Boot Record ... done
0000  FA 33 C0 8E D0 BC 00 7C 8B F4 50 07 50 1F FB FC   ú3À?Ð¼.|‹ôP.P.ûü
0010  BF 00 06 B9 00 01 F2 A5 EA 1D 06 00 00 BE BE 07   ¿..¹..ò¥ê....¾¾.
0020  B3 04 80 3C 80 74 0E 80 3C 00 75 1C 83 C6 10 FE   ³.?<?t.?<.u.ƒÆ.þ
0030  CB 75 EF CD 18 8B 14 8B 4C 02 8B EE 83 C6 10 FE   ËuïÍ.‹.‹L.‹îƒÆ.þ
0040  CB 74 1A 80 3C 00 74 F4 BE 8B 06 AC 3C 00 74 0B   Ët.?<.tô¾‹.¬<.t.
0050  56 BB 07 00 B4 0E CD 10 5E EB F0 EB FE BF 05 00   V»..´.Í.^ëðëþ¿..
0060  BB 00 7C B8 01 02 57 CD 13 5F 73 0C 33 C0 CD 13   ».|¸..WÍ._s.3ÀÍ.
0070  4F 75 ED BE A3 06 EB D3 BE C2 06 BF FE 7D 81 3D   Ouí¾£.ëÓ¾Â.¿þ}•=
0080  55 AA 75 C7 8B F5 EA 00 7C 00 00 49 6E 76 61 6C   UªuÇ‹õê.|..Inval
0090  69 64 20 70 61 72 74 69 74 69 6F 6E 20 74 61 62   id partition tab
00A0  6C 65 00 45 72 72 6F 72 20 6C 6F 61 64 69 6E 67   le.Error loading
00B0  20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74 65    operating syste
00C0  6D 00 4D 69 73 73 69 6E 67 20 6F 70 65 72 61 74   m.Missing operat
00D0  69 6E 67 20 73 79 73 74 65 6D 00 00 00 00 00 00   ing system......
00E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

```
0120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 80 01   ..............?.
01C0  01 00 06 07 E0 D2 20 00 00 00 E0 D2 03 00 00 00   ....àÒ ...àÒ....
01D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA   ..............Uª

Partition-Table

Partition 1 Sector: 32 to 250624 = 250592 Sectors; File-System Type: 06

0000>mp $100                    ' Place Memory Write pointer to HEX 100


0100>mt Hello World             ' Write Text to adjusted Memory Pointer


010B>mb 32 33 34 $41 $42 $43    ' Store some bytes


0111>mf $55 $118 $12F           ' fill Hex 118 to 12F with Hex 55 ('U')


0130>sw 3 1                     ' Write Transfer Buffer to Sector 3

Write 1 Sector(s) to 3 at CF-Card from Transfer-Buffer ...  done

0130>sd 3                       ' Show sector 3

Read Sector: 3 ...  done
0000  FA 33 C0 8E D0 BC 00 7C 8B F4 50 07 50 1F FB FC   ú3À?Ð¼.|‹ôP.P.ûü
0010  BF 00 06 B9 00 01 F2 A5 EA 1D 06 00 00 BE BE 07   ¿..¹..ò¥ê....¾¾.
0020  B3 04 80 3C 80 74 0E 80 3C 00 75 1C 83 C6 10 FE   ³.?<?t.?<.u.ƒÆ.þ
0030  CB 75 EF CD 18 8B 14 8B 4C 02 8B EE 83 C6 10 FE   ËuïÍ.‹.‹L.‹îƒÆ.þ
0040  CB 74 1A 80 3C 00 74 F4 BE 8B 06 AC 3C 00 74 0B   Ët.?<.tô¾‹.¬<.t.
0050  56 BB 07 00 B4 0E CD 10 5E EB F0 EB FE BF 05 00   V»..´.Í.^ëðëþ¿..
0060  BB 00 7C B8 01 02 57 CD 13 5F 73 0C 33 C0 CD 13   ».|¸..WÍ._s.3ÀÍ.
0070  4F 75 ED BE A3 06 EB D3 BE C2 06 BF FE 7D 81 3D   Ouí¾£.ëÓ¾Â.¿þ}•=
0080  55 AA 75 C7 8B F5 EA 00 7C 00 00 49 6E 76 61 6C   UªuÇ‹õê.|..Inval
0090  69 64 20 70 61 72 74 69 74 69 6F 6E 20 74 61 62   id partition tab
00A0  6C 65 00 45 72 72 6F 72 20 6C 6F 61 64 69 6E 67   le.Error loading
00B0  20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74 65    operating syste
00C0  6D 00 4D 69 73 73 69 6E 67 20 6F 70 65 72 61 74   m.Missing operat
00D0  69 6E 67 20 73 79 73 74 65 6D 00 00 00 00 00 00   ing system......
00E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0100  48 65 6C 6C 6F 20 57 6F 72 6C 64 20 21 22 41 42   Hello World !"AB
0110  43 00 00 00 00 00 00 00 55 55 55 55 55 55 55 55   C.......UUUUUUUU
0120  55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55   UUUUUUUUUUUUUUUU
0130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

```
0190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 80 01   ..............?.
01C0  01 00 06 07 E0 D2 20 00 00 00 E0 D2 03 00 00 00   ....àÒ ...àÒ....
01D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA   ..............Uª

' The previous manipulated range is yellow marked

0000>sw 4 8 0                  ' Write all ATMEGA103 SRAM (from Address 0
                               ' with 8 sectors (=4096 Bytes) to Card
                               ' starting at sector 4

Write 8 Sector(s) to 4 at CF-Card from SRAM Address 0000 ...  done

0000>
```

How to start:

- Load FlashDisk.bas, FlashCardDrive.bas and Interpreter.bas in same directory.
- If not using Hardware-UART 0, 9600 Baud, 4 MHz Cristall, ATMEGA103 adjust them in FlashCard.bas.
- IF not using same schematic between AVR (Port A for Data, Port B for Controlling and Port C for register-addressing) and FlashCard as shown in this application note, adjust hardware-settings in FlashCardDrive.bas.
- Compile FlashCard.bas
- Program AVR
- Connect AVR to PC via a RS232
- Start a terminal program
- (Re-)Start AVR
- Inside Terminal program you can make tests with the Card as shown above.

I made tests with 15 MB Card (Canon FC-15M), and a 128 MB Card (SanDisk). Both have a buffersize of 2 Sector (=1024 Bytes). I made the experience, that if I write on the 15MB Card only one sector (=512 Bytes) at once a second sector (normally the sector after the written one) lost the data. Writing 2 Sectors (=1024 Bytes) at once will work right. On the 128 MB Card I don't have detected such behaviour. I would be glad to hear some experiences from other user of Cards with a buffersize of 2 sectors.

Comments and experiences are welcomed to josef.voegel@aon.at

To get more information I suppose following Links:

http://www.circuitcellar.com/echips-pdfs/0201/c0201mspdf.pdf
http://www.sandisk.com/download/Product%20Manuals/cf_r7.pdf

```
' ================================================================================
' File: Flashdisk.Bas
' ================================================================================

$Regfile = "M103Def.dat"
$Crystal = 4000000


' Use Serial 0 for communication with PC-Terminal
$Baud = 9600
Config Serialin = buffered , Size = 20
Open "Com1:" for Binary as #1
' If you use a software-UART, define it with #1
Enable Interrupts


goto BehindIncludes
$Include "FlashCardDrive.bas"
$Include "Interpreter.bas"

BehindIncludes:

' Init Port and Card

Print #1 , "Setup Port and Reset Card ... ";

if DriveCheck() = 1 then
   DriveInit
else
    print #1 , "Card not inserted, check Card!"
end if


Print #1 , "OK"
print #1 , "Ready for Commands"

PrintPrompt

Dim gbInp as Byte                           ' holds user input

' Handling communication with user
gbPCInputPointer = 1
do
  gbInp = inkey(#1)                         ' get user input
  if gbInp <> 0 then                        ' something typed in?
     GetInput gbInp                         ' give input to interpreter
  end if
loop                                        ' do forever


End                                         'end program
```

```
' ================================================================================
' File: Interpreter.Bas
' ================================================================================

' Getting USER-Input and executes Commands

Declare Sub DoCommand()
Declare Sub ExtractToken()
Declare Function GetNextTokenStr(ByVal pbLen_Max as Byte ) as String
Declare Function GetNextTokenLong(byVAl plMin as Long , byVal plMax as Long ) as Long
Declare Sub PrintParameterErrorl(plParamLow as Long , plParamHigh as Long)
Declare Sub PrintParameterCountError(byVal psParm_Anzahl as String)
Declare Sub GetInput(byVal pbByte as Byte)
Declare Sub PrintPrompt()
Declare Function GetLongFromBuffer(pbSRAMArray as Byte , byVal pbPos as Word) as Long
Declare Function GetWordFromBuffer(pbSRAMArray as Byte , byVal pbPos as Word) as Word
Declare Sub SRAMDump(pwSRAMPointer as Word , byVal pwLength as Word)


Const cpToken_max = 10                          ' Count of Tokens in USER-Input
Const cpStrsep = " "                            ' Blank: Separator between tokens
Const cpNo = 0                                  '
Const cpYes = 1
Const cPCInput_Len = 40                         ' max. length of user-Input
Dim gsPCInput as String * 40                    ' holds user-input
Dim gbPosStrParts(cpToken_max) As Byte          ' for analysing user-input
Dim gbLenStrParts(cpToken_max) As Byte          '
Dim gbCntToken As Byte                          ' found tokens in user-input
Dim gbToken_Actual as Byte                      ' actual handled token of user-input
Dim gbPCInPutError as Byte                      ' holds error-code during analysing user-input
Dim gbPCInputPointer as Byte                    ' string-pointer during user-input
gbPCInputPointer = 1

Dim CF_TransferBuffer(1024) as Byte             ' Transferbuffer von und zur CF for 2 Sectors
Dim TransferBuffer_Write as Word



End


Sub GetInput(pbByte as Byte)
    ' stores bytes from user and wait for CR (&H13)
   Print #1 , chr(pbByte);                      ' echo back to user
   select case pbByte
      Case &H0D                                 ' Line-end?
         print chr(&H0A)
         DoCommand                              ' analyse command and execute
         gbPCInputPointer = 1                   ' reset for new user-input
         gsPCInput = ""
         PrintPrompt
      case &H08                                 ' backspace ?
         decr gbPCInputPointer
      case Else                                 ' store user-input
         mid(gsPCINput , gbPCInputPointer , 1) = pbByte
         incr gbPCInputPointer
         mid(gsPCInput , gbPCInputPointer , 1) = &H00       ' string-terminator
         if gbPCInputPointer > cPCInput_Len then       'don't exceed input-string
            gbPCInputPointer = cPCInput_Len
         end if
   end Select
End Sub


Sub DoCommand
   ' interpretes the user-input and execute
   ' Local variables
   Local lByte1 as Byte , lByte2 as Byte
   Local lInt1 as Integer , lInt2 as Integer
   Local lWord1 as Word , lWord2 as Word , lWord3 as Word
   Local lLong1 as Long , lLong2 as Long , lLong3 as Long , lLong4 as Long
   Local lbPos as Byte

   Local lsToken as String * 20                 ' Hold Tokens
   Local lbLen as Byte
   Local lwSRAMPointer as Word

   ExtractToken                                 ' token analysing
```

```
gbToken_actual = 0                          ' reset to beginn of line (first token)
gbPCInputError = cpNo
If gbcntToken > 0 Then                      ' is there any input

   lsToken = GetNextTokenStr(20)            ' get first string-token = command
   lstoken = ucase(lsToken)                 ' all uppercase
   lwSRAMPointer = varptr(Cf_Transferbuffer(1))
                                  ' Pointer to SRAM Buffer
   Select Case lsToken
      Case "CFI"                            ' Show CF-Card Information Block
           Print #1 , "Read Card Info"
           GetDriveIdentity lwSRAMPointer   ' read Info to SRAM
           TransferBuffer_Write = 0
           SRAMDump lwSRAMPointer , 512     ' Dump SRAM
           ' Get Count of Sectors in Compactflash-Card
           lLong1 = GetLongFromBuffer(CF_Transferbuffer(1) , 120) : lLong2 = lLong1 * 512
           print lLong1 ; " Sectors = " ; lLong2 ; " Bytes"
           ' Get Buffersize of Compactflash-Card
           lWord1 = GetWordFromBuffer(CF_TransferBuffer(1) , 42)
           lLong2 = lWord1 * 512
           print "CF-Buffersize = " ; lWord1 ; " Sectors = " ; lLong2 ; " Bytes"

      Case "CFR"                            ' Reset Compactflash Card
           DriveReset

      Case "MBR"                            ' Show Masterboot record = Sector 0
         lLong1 = 0
         print #1 , "Read Master Boot Record ... " ;
         ReadSector LWSRAMPointer , lLong1 , 1       ' read Sector to CF_Transferbuffer
         TransferBuffer_Write = 0
         print #1 , "done"
         SRAMDump lwSRAMPointer , 512        ' show CF_Transferbuffer
         print #1 , " " : Print #1 , "Partition-Table" : print #1 , " "
         lWord1 = 446                        ' first partition entry starts at 446
         for lByte1 = 1 to 4
            lWord2 = lWord1 + 1
            if CF_TransferBuffer(lWord2) > 0 then
               print #1 , "Partition " ; lByte1 ; " " ;
               lWord2 = lWord1 + 8
               lLong1 = GetLongFromBuffer(CF_Transferbuffer(1) , lWord2)
               lWord2 = lWord1 + 12
               lLong2 = GetLongFromBuffer(CF_Transferbuffer(1) , lWord2)
               lLong3 = lLong1 + lLong2
               print #1 , "Sector: " ; lLong1 ; " to " ; lLong3 ; " = " ; lLong2 ; " Sectors; ";
               lWord2 = lWord1 + 5
               lByte1 = Cf_TransferBuffer(lWord2)
               print #1 , "File-System Type: " ; hex(lByte1)
            end if
            lWord1 = lWord1 + 16
         next

      Case "SD"                             ' Sector Dump
         If gbCntToken = 2 then
            lLong1 = GetNextTokenLong(0 , 2000000)
            lLong2 = lLong1
         elseif gbCntToken = 3 then
            lLong1 = GetNextTokenLong(0 , 2000000)
            lLong2 = GetNextTokenLong(lLong1 , 2000000)
         else
             PrintparameterCountError "1, 2 "
             exit sub
         end if
         If gbPCInputError = cpNo then
            for lLong3 = lLong1 to lLong2
                print #1 , "Read Sector: " ; lLong3 ; " ... " ;
                ReadSector lwSRAMPointer , lLong3 , 1
                TransferBuffer_Write = 0
                Print #1 , " done"
                SRAMDump lwSRAMPointer , 512
            next
         End if

      Case "MD"                             ' Memory Dump
           lWord2 = 512
           if gbCntToken = 1 then
           elseif gbCntToken = 2 then
              lLong1 = GetNextTokenLong(0 , &HFFFF)
              lwSRAMPointer = lLong1        ' assign to word
```

```
      elseif gbCntToken = 3 then
         lLong1 = GetNextTokenLong(0 , &HFFFF)
         lwSRAMPointer = lLong1          ' assign to word
         lLong2 = GetNextTokenLong(lLong1 , &HFFFF)
         lWord2 = lLong2
      else
          PrintParameterCountError "0, 1, 2 "
      end if
      if gbPCInputError = cpNo then
         SRAMDump lwSRAMPointer , lWord2       ' Show 512 Bytes
      End if

Case "SW"                                ' Sector Write
   If gbCntToken = 3 then
      lLong1 = GetNextTokenLong(0 , 2000000)
      lLong2 = GetNextTokenLong(1 , &H7F)
      lByte1 = lLong2
      lLong3 = lLong1 + lLong2
   elseif gbCntToken = 4 then
      lLong1 = GetNextTokenLong(0 , 2000000)
      lLong2 = GetNextTokenLong(1 , &H7F)
      lByte1 = lLong2
      lLong3 = lLong1 + lLong2
      lLong4 = GetNextTokenLong(0 , &HFFFF)
      lwSRAMPointer = lLong4
   else
      PrintparameterCountError "2, 3 "
      exit sub
   end if
   If gbPCInputError = cpNo then
      print #1 , "Write " ; lByte1 ; " Sector(s) to " ; lLong1 ; " at CF-Card from " ;
      if gbCntToken = 4 then
         print #1 , "SRAM Address " ; hex(lwSRAMPointer) ; " ... " ;
      else
         print #1 , "Transfer-Buffer ... " ;
      end if
      WriteSector lwSRAMPointer , lLong1 , lByte1
      print #1 , " done"
   End If

Case "MT"                                ' Fill Memory with Text
   if gbCntToken > 1 then
      lByte1 = gbPosStrParts(2)
      do
        incr TRANSFERBUFFER_WRITE
        lsToken = mid(gsPCInput , lByte1 , 1)
        lByte2 = ASC(lsToken)
        if lByte2 = 0 then               ' String Terminator
           exit do
        end if
        CF_Transferbuffer(Transferbuffer_Write) = lByte2
        incr lByte1
      loop until Transferbuffer_Write > 1023
      decr TRANSFERBUFFER_WRITE          ' 1 based to 0 based
  End IF

Case "MP"                                ' Memory Pointer for MB and MT
      if gbCntToken = 2 then
         lLong1 = GetNextTokenLong(0 , 1023)
         if gbPCInputError = cpNo then
            TransferBuffer_Write = lLong1
         end if
      else
          PrintParameterCountError "1 "
      End if

Case "MB"                                'Fill Memory with Same Byte
   if gbCntToken > 1 then
      for lByte1 = 2 to gbCntToken
          lLong1 = GetNextTokenLong(0 , 255)
          if gbPCInputError = cpNo then
             incr TransferBuffer_Write
             lByte2 = lLong1
             CF_TransferBuffer(TransferBuffer_Write) = lByte2
             if TransferBuffer_Write >= 1023 then
                exit for
             end if
          else
```

```
                    exit for
                end if
            next
        end if

    Case "MF"

        lLong2 = TransferBuffer_Write : lLong3 = 1023
        if gbCntToken = 2 then
            lLong1 = GetNextTokenLong(0 , 255)
        elseif gbCntToken = 3 then
            lLong1 = GetNextTokenLong(0 , 255)
            lLong2 = GetNextTokenLong(0 , 1023)
        elseif gbCntToken = 4 then
            lLong1 = GetNextTokenLong(0 , 255)
            lLong2 = GetNextTokenLong(0 , 1023)
            lLong3 = GetNextTokenLong(lLong2 , 1023)
        else
             printParameterCountError "1, 2, 3 "
             exit sub
        end if
        If gbPCInputError = cpNo then
            lByte1 = lLong1
            incr lLong2 : lWord2 = lLong2
            Incr lLong3 : lWord3 = lLong3
            for lWord1 = lWord2 to lWord3
                CF_Transferbuffer(lWord1) = lByte1
            next
            TransferBuffer_Write = lWord1 - 1
        end if

    Case Else

        Print #1 , "Command '" ; gsPCInput ; "' not recognized"

  End Select
  if Transferbuffer_write > 1023 then
     Transferbuffer_write = 0
  End if

 End If
End Sub



Sub ExtractToken
' Counts the Token in the Input-String: gsPCInput
' following variable and arrays are filled
' cntToken: Cont of Token
' PosStrParts: positions, where the tokens start
' LenStrParts: Count of bytes of each token

 Local Lstrlen As Byte
 Local LparseEnd As Byte
 Local Lpos1 As Byte , Lpos2 As Byte
 ' Init arrays with 0
 For Gbcnttoken = 1 To cpToken_max
    Gbposstrparts(gbcnttoken) = 0 : Gblenstrparts(gbcnttoken) = 0
 Next

 Gbcnttoken = 0
 gsPCInput = Trim(gsPCInput)
 Lstrlen = Len(gsPCInput)                    ' how long is string
 If Lstrlen = 0 Then                         'no Input ?
    Exit Sub
 End If
 LparseEnd = 0
 Lpos1 = 0
 For Gbcnttoken = 1 To cpToken_max
    Incr Lpos1
    Lpos2 = Instr(lpos1 , gsPCInput , cpStrSep)       ' find next blank
    If Lpos2 = 0 Then                        ' no more found?
       Lpos2 = Lstrlen : Incr Lpos2 : LparseEnd = 1
    End If
    Gblenstrparts(gbcnttoken) = Lpos2 - Lpos1 ' Lenght of token
    Gbposstrparts(gbcnttoken) = Lpos1
    If LparseEnd = 1 Then
       Exit For
```

```
      End If
      Lpos1 = Lpos2
   Next
End Sub


Function GetNextTokenStr(ByVal pbLen_Max as Byte ) as String
   ' Returns next String-token from Input
   ' Parameter: pbLen_Max: Limit for string-length
   Local lbPos as Byte
   Local lbLen as Byte
   incr gbToken_actual                          ' switch to new/next token
   lbPos = gbPosStrParts(gbToken_actual)        ' at which position in string
   lbLen = gbLenStrParts(gbToken_actual)        ' how long
   If lbLen > pbLen_Max Then lbLen = pbLen_Max  ' to long?
   GetNextTokenStr = mid(gsPCInput , lbPos , lbLen)       ' return string
End Function


Function GetNextTokenLong(byVAl plMin as Long , byVal plMax as Long ) as Long
   ' returns a Long-Value from next Token and check for inside lower and upper limit
   ' plMin: minimum limit for return-value
   ' plMax: maximum limit for return-value
   Local lbPos as Byte
   Local lbLen as Byte
   Local lsToken as String * 12
   incr gbToken_actual                          ' switch to new/next token
   lbPos = gbPosStrParts(gbToken_actual)        ' at which position in string
   lbLen = gbLenStrParts(gbToken_actual)        ' how long
   If lbLen > 12 Then lbLen = 12                ' to long?
   if mid(gsPCInput , lbPos , 1) = "$" then     ' Is input a HEX vlue?
      incr lbPos : decr lbLen                   ' adjust pointer to jump over $
      lsToken = mid(gsPCInput , lbPos , lbLen)
      GetNextTokenLong = hexval(lstoken)
   Else
       lsToken = mid(gsPCInput , lbPos , lbLen)
       GetNextTokenLong = val(lsToken)
   End if
   Select Case GetNextTokenLong                 ' check for limits
      Case plMin to plMax                       ' within limits, noting to do
      Case Else
         gbPCInputError = cpYes                 ' Set Error Sign
         Print #1 , Spc(lbPos) ; "^ " ; "Parameter Error ";
         PrintParameterErrorL plMin , plMax     ' with wanted limits
   End Select
End Function


Sub PrintParameterCountError(byVAl psParm_Anzahl as String * 10)
   ' User message for wrong count of parameter
   Print #1 , "? " ; psParm_Anzahl ; " " ; "Parameter " ; "expected "
End Sub

Sub PrintParameterErrorl(plParamLow as Long , plParamHigh as Long)
   ' Print Limits at wrong Input - value
   Print #1 , " [ " ; plParamLow ; " ] - [ " ; plParamHigh ; " ] " ; "expected "
End Sub


Sub PrintPrompt()
    Print #1 ,
    Print #1 , hex(TransferBuffer_Write) ; ">" ;
End Sub


Function GetLongFromBuffer(pbSRAMArray as Byte , byVal pbPos as Word) as Long
   ' Extract a Long-Value from a Byte-Array
   ' pbSRAMArray: Byte-array, from which the Long-value should be extracted
   ' pbPos: Position, at which the Long-Value starts (0-based)
   loadadr pbSRAMArray , Z
   loadadr pbPos , X
   ld r24, x+
   ld r25, x+
   add zl, r24
   adc zh, r25
   loadadr GetLongFromBuffer , X
   ldi r24, 4
   !Call _Mem_Copy
```

```
End Function


Function GetWordFromBuffer(pbSRAMArray as Byte , byVal pbPos as Word) as Word
    ' Extract a Word-value from a Byte-Array
    ' pbSRAMArray: Byte-array, from which the Word-value should be extracted
    ' pbPos: Position, at which the Word-Value starts (0-based)
    loadadr pbSRAMArray , Z
    loadadr pbPos , X
    ld r24, x+
    ld r25, x+
    add zl, r24
    adc zh, r25
    loadadr GetWordFromBuffer , X
    ldi r24, 2
    !Call _Mem_Copy
End Function


Sub SRAMDump(pwSRAMPointer as Word , byVal pwLength as Word)
     ' Dump a Part of SRAM to Print-Output #1
     ' pwSRAMPointer: (Word) Variable which holds the address of SRAM to dump
     ' pwLength: (Word) Count of Bytes to be dumped (1-based)
     Local lsDump as String * 16
     Local lByte1 as Byte , lByte2 as Byte
     Local lWord1 as Word , lWord2 as Word
     if pwLength > 0 then
       Decr pwLength
       for lword1 = 0 to pwLength
          lWord2 = lWord1 mod 16
          if lWord2 = 0 then
             if lWord1 > 0 then
                Print #1 , "  " ; lsDump
             end if
             Print #1 , hex(lWord1) ; "  " ;
             lsDump = "                "
             lByte2 = 1
          end if
          lByte1 = Inp(pwSRAMPointer)
          incr pwSRAMPointer
          Print #1 , hex(lByte1) ; " " ;
          if lByte1 > 31 then
             mid(lsdump , lByte2 , 1) = lByte1
          else
              mid(lsdump , lByte2 , 1) = "."
          end if
          incr lByte2
       next
       Print #1 , "  " ; lsDump
     End if
End Sub

' -----------------------------------------------------------------------------
' copy Memory from (Z) nach (X)
' counts of bytes in r24
_Mem_Copy:
    ld r25, z+
    st x+, r25
    dec r24
    brne _Mem_Copy
    ret
```

```
' ================================================================================
' File: FlashCardDrive.Bas
' ================================================================================

' This files contains the Routines to access the FlashCard


Declare Sub ReadSector(pwSRAMPointer as Word , plSectorNumber as Long , byVal pbSectorCount as Byte)
Declare Sub WriteSector(pwSRAMPointer as Word , plSectorNumber as Long , byVal pbSectorCount as
Byte)
Declare Sub GetDriveIdentity(pwSRAMPointer as Word )
Declare Sub DriveInit()
Declare Sub DriveReset()
Declare Function DriveCheck() as Byte


' Register - definitions for ASM-Routines

rByteCount_Low alias r20
rByteCount_High alias r21
rCF_SectorCount alias r21
rCF_Reg alias r22
rData alias r23


Sub ReadSector(pwSRAMPointer as Word , plSectorNumber as Long , byVal pbSectorCount as Byte)
    ' Read Sector(s) from CF-Card to SRAM
    ' pwSRAMPointer: (Word) Variable which holds the address of SRAM to transfer Sector(s) from Card
    ' plSectorNumber: (Long) CF-Startingsector from which data should be transfered
    ' pbSectorCount: (Byte) Count of 512-Byte Sector(s) to transfer (largest Value = &H7F)
    loadadr pbSectorCount , X
    ld rCF_SectorCount, X
    loadadr pwSRAMPointer , X
    ld zl, X+
    ld zh, X+
    loadadr plSectorNumber , X
    !call _CF_Read_Sector
End Sub



Sub WriteSector(pwSRAMPointer as Word , plSectorNumber as Long , byVal pbSectorCount as Byte)
    ' Write Sector(s) from CF-Card to SRAM Byte
    ' pwSRAMPointer: (Word) Variable which holds the address of SRAM to transfer Sector(s) from Card
    ' plSectorNumber: (Long) CF-Startingsector from which data should be transfered
    ' pbSectorCount: (Byte) Count of 512-Byte Sector(s) to transfer (largest Value = &H7F)
    loadadr pbSectorCount , X
    ld rCF_SectorCount, X
    loadadr pwSRAMPointer , X
    ld zl, x+
    ld zh, x+
    loadadr plSectorNumber , X
    !call _CF_Write_Sector
End Sub



Sub GetDriveIdentity(pwSRAMPointer as Word )
    ' Read the Identity Drive from CF-Card to SRAM Byte
    ' pwSRAMPointer: (Word) Variable which holds the address of SRAM to transfer Sector(s) from Card
    ' 512 Bytes will be written
    loadadr pwSRAMPointer , X
    ld zl, x+
    ld zh, x+
    !call _CF_Read_DriveIdentity
End Sub



Sub DRIVEINIT()
    ' Setup Pins to CF-Card for Input/Output and reset CF-Card
    !Call _CF_Setup
End Sub



Sub DRIVERESET()
    ' Reset CF-Card
```

```
    !Call _CF_Reset
End Sub



Function DriveCheck() as Byte
    ' Check, if drive is ready (plugged in)
    !Call _CF_CheckCard
    loadadr DriveCheck , X
    st x, r24
End Function



' ================================================================================
' ASM - Routines
' --------------------------------------------------------------------------------

' Hardware Definitions
' Port und Pin - Definitionen

' Data Port
CF_Data_Out alias PortA
CF_Data_In alias PinA
CF_Data_DDR alias DDRA

' Register - addressing
CF_ADDR_Out alias PortC
' CF_Addr_in alias PinC                ' Port C nur Output
' CF_Addr_DDR alias DDRC
' remove Comment-mark ' if bidirectional Port is used

' Control - Port
CF_Control_Out alias PortB
CF_Control_In alias PinB
CF_Control_DDR alias DDRB

' Pins at Control-Port
CF_CE1 alias 0                                  ' Card enable
CF_CD1 alias 1                                  ' Card detect
CF_RESET alias 2                                ' Reset-Pin
CF_OE alias 3                                   ' Output enable
CF_WE alias 4                                   ' Write enable
CF_RDY alias 5                                  ' Card ready

' Input: CF_CD1, CF_RDY: Input
Const CF_Control_Direction = &B00011101         ' I/O Richtungen am Control-Port

Const CF_Control_Init = &B00011000              ' Setup-Ausgabe  für Control-Port
Const CF_Control_Dir_Mask = &B11000000          'Maske für belegte Pin's am Controlport

' CF-Card Register
' addresses uses 3 LSB Bit x x x x x A2 A1 A0

' Register addresses
Const Data_Reg = &H00
Const Error_Reg = &H01
Const Feature_Reg = &H01
Const Sec_Cnt_Reg = &H02
Const Sec_Num_Reg = &H03
Const Cyl_Lo_Reg = &H04
Const Cyl_Hi_Reg = &H05
Const Head_Reg = &H06
Const Status_Reg = &H07
Const Command_Reg = &H07

' Commands for Compact Flash Card

Const CF_Identify = &HEC
Const CF_Write_Sec = &H30
Const CF_Read_Sec = &H20

'--------------------------------------------------------------------------------
' Read Identity Info from the CF-Card
_CF_Read_DriveIdentity:
   ldi rCF_SectorCount, 1                       ' 512 Bytes = 1 Sector to read
   ldi rData, CF_Identify                       ' Command for Card identity
   rcall _CF_Write_CommandReg                   ' send to card
```

```
   rjmp _CF_Read_Sector_LBASet                  ' Read Info to SRAM (mit return)


'-------------------------------------------------------------------------------
' Pin CF_Rdy is Low at Ready
_CF_Check_Ready:
   sbis CF_Control_In, CF_Rdy                   ' no timeout!!!
   rjmp _CF_Check_Ready
   Ret

'-------------------------------------------------------------------------------
' give a pulse to WE to force Compactflash Card to read Data
' Registeraddress (A0-A2) and Data (D0-D7) must be set
_CF_Write:
   rcall _CF_Check_ready
   cbi CF_Control_Out, CF_WE                    ' WE Line to Low
   nop                                          ' 600 ns at 3.3 V VCC is enough
   nop
   sbi CF_Control_Out, CF_WE                    ' WE Line to High
   ret

'-------------------------------------------------------------------------------
' Read 1 Byte from the CF
' Register address must be set.
_CF_Read:
   rcall _CF_Check_Ready
   ldi _temp1, &H00                             ' change data-port to Input
   !out Cf_Data_DDR , _temp1
   cbi CF_Control_out, CF_OE                    ' OE Line to Low
   nop
   nop
   nop
   in rData,CF_Data_in                          ' Read Byte
   sbi CF_Control_Out, CF_OE                    ' OE - Line to High
   ldi _temp1, &HFF                             ' change Data-port back to OUTPUT
   !out CF_Data_DDR, _temp1
   ret

'-------------------------------------------------------------------------------
' Set register address, and leave pins 3 - 7 unchanged
' two entry-points to routine
' 1: CF_Set_RegAddr: Register address must passed in rCF_Reg
' 2: CF_Set_RegAddrData: register address is Data
_CF_Set_RegAddrData:
   ldi rCF_Reg, Data_Reg
_CF_Set_RegAddr:
   in _temp1, CF_Addr_Out                       ' get current output at Port
   andi _temp1, &B11111000                      ' CF-Address-bits masked out
   or _temp1, rCF_Reg                           ' add CF-Address with OR
   !out CF_Addr_Out, _temp1
   ret


' -------------------------------------------------------------------------------
' Read Sector(s) from CF-Card to SRAM
' Entry with following parameters set:
' Register rCF_SectorCount: Count of Sector(s) to transfer
' Register X: Pointer to Sectornumber (LONG)
' Register Z: SRAM-Address, where the data to transfer
_CF_Read_Sector:
   rcall _CF_Set_CntLBA                         ' LBA-Sector to CF
   ldi rData, CF_Read_Sec                       ' read command
   rcall _CF_Write_CommandReg                   ' to Compactflash
_CF_Read_Sector_LBASet:
   rcall _CF_Set_RegAddrData                    ' turn register-address to data
   clr rByteCount_Low                           ' Low-Byte of Byte-Count always zero
   lsl rByteCount_High                          ' Change Sector-Count to HighByte of Transferlenght
_CF_Read_Sector1:
   rcall _CF_Read                               ' read 1 Byte
   st z+, rData
   dec rByteCount_Low
   brne  _CF_READ_SECTOR1
   dec rByteCount_High
   brne  _CF_READ_SECTOR1                       ' all bytes read?
   Ret


' -------------------------------------------------------------------------------
```

```
' write Sector(s) to CF-Card
' Entry with following parameter set:
' Register rCF_SectorCount: Count of sector(s) to transfer
' Register X: Pointer to Sectornumber (LONG)
' Register Z: SRAM-Address, at which Data to transfer starts
_CF_Write_Sector:
   rcall _CF_Set_CntLBA                         ' LBA-Sector to CF
   ldi rData, CF_Write_Sec                      ' write command
   rcall _CF_Write_CommandReg                   ' to compactflash
_CF_Write_Sector_LBASet:
   rcall _CF_Set_RegAddrData                    ' turn register-address to data
   clr rByteCount_Low                           ' Low-Byte of Byte-Count alwas zero
   lsl rByteCount_High                          ' Change Sector-Count to HighByte of Transferlenght
_CF_Write_Sector1:
   ld rData, z+
   !out CF_Data_out, rData                      ' Byte to data port
   rcall _CF_Write                              ' force CF to read byte
   dec rByteCount_low
   brne  _CF_Write_SECTOR1
   dec rByteCount_High
   brne  _CF_Write_SECTOR1                      ' last byte written?
   Ret

'-------------------------------------------------------------------------------
' write a value to a specified register address, value is passed in rData
' two different entry points
' 1: CF_Write_CommandReg: write value to command-register (default)
' 2: CF_Write_Reg: Register passed in rCF_Reg
_CF_Write_CommandReg:
   ldi rCF_Reg, Command_Reg                     ' Default register: Command
_CF_Write_Reg:
   !out CF_Data_out, rData                      ' set data to data port
   rcall _CF_Set_RegAddr                        ' set register address
   rjmp _CF_Write                               ' force CF to accept register and data

' -----------------------------------------------------------------------------
' Write count of sector(s) (read or write) and Sectornumber (LBA) to CF-Card
' following parameter must be set:
' Register CF_SectorCount: Count of Sector(s) (max &7F)
' Register X: Pointer to sectornumber
'
_CF_Set_CntLBA:
   ldi rCF_Reg, Sec_Cnt_Reg                     ' Start with Sector-Count Register in CF (2)
   mov rData, rCF_SectorCount                   ' prepare for output
   rjmp _CF_Set_LBA2
_CF_Set_LBA1:
   ld rData, X+                                 ' load next byte of sectornumber
_CF_Set_LBA2:
   rcall _CF_Write_Reg                          ' set register-address
   inc rCF_Reg                                  ' set to next register address
   cpi rCF_Reg, 6
   brmi  _CF_Set_LBA1                           ' Reg. 6 reached? (LBA/Disk/Header)
   ld rData, X+                                 ' need special handling
   andi rData, &B00001111                       ' mask upper nibble (LBA-Sign and drive)
   ori rData, &HE0                              ' set to LBA
   rjmp _CF_Write_Reg                           ' write to CF; incl Ret

' -----------------------------------------------------------------------------
' Setup the pins needed for the CF-Card
_CF_Setup:
   ' Data port to OUTPUT
   ldi _temp1, &H00                             ' all Data pins to low
   !out CF_Data_Out, _temp1

   ldi _temp1, &HFF
   !Out CF_Data_DDR , _temp1                    ' Data pins to output

   ' Controlport: prepare 6 pins, leave other two unchanged
   in _temp1, CF_Control_Out                    '
   andi _temp1, CF_Control_Dir_Mask             '
   ori _temp1, CF_Control_Init
   !Out CF_Control_Out, _temp1

   in _temp1, CF_Control_DDR                    ' read direction of pins at control port
   andi _temp1, CF_Control_Dir_Mask             ' mask out used pins
   ori _temp1, CF_Control_Direction             ' set direction in and Out
   !Out CF_Control_DDR, _temp1                  ' set new direction configuration
```

```
   ' Address port: attention: adjust if not on Port C at ATMega103
   ' ATMega103 Port C have only output, so here no configuration is necessary
   'in _temp1, CF_ADDR_DDR
   'ori _temp1, &B00000111
   '!Out CF_Addr_DDR, _temp1
   waitms 1
   rjmp _CF_Reset

'--------------------------------------------------------------------------------
' force CF-Card to a Hardware-reset
_CF_Reset:
   sbi CF_Control_Out, CF_Reset
   waitms 1
   cbi CF_Control_Out, CF_Reset
   waitms 500
   ret

'--------------------------------------------------------------------------------
' Checks, whether Card is plugged in: Pin CD1 is LOW
' if OK r24 = 1; otherwise 0
_CF_CheckCard:
   ldi r24, 1
   sbic CF_Control_In, CF_CD1                  ' flashcard plugged in?
   ldi r24, 0
   ret
```