NOFAT - “Datei” - System

Zweck:

Der Mikrokontroller soll auf einer SD-Speicherkarte Informationen im ASCII-Format ablegen können, ohne dass ein grosser Verwaltungsaufwand und viel Mikrokontroller - Ressourcen nötig sind. Das NOFAT-System läuft schon auf einem Mega8, während für AVR-DOS ein Mega128 (existiert nur in SMD!) nötig ist.

Lesen am PC

Da keine FAT existiert, kann ein normales Betriebssystem die Karte nicht lesen, die Daten müssen sektorweise gelesen werden. Dies ist mit speziellen Tools wie http://www.chrysocome.net/dd (Disk Dump) oder mit einem VB-Programm möglich.

Organisation der Karte

Sektor 0:
enthält wichtige Infos über Kartengrösse etc. wie bei DOS - Formatierung, NOFAT - Kennung und Nummer des nächsten freien Sektors.

Ab Sektor 1: Datenbereich im Textformat.
Es gibt natürlich keine richtigen Dateien, die Daten können aber bei Bedarf in Bereiche eingeteilt werden, die mit **“<BOF>” (Begin of File)** beginnen und mit **“<EOF>” (End of File)** enden. Diese Bereiche werden im folgenden als “Dateien” bezeichnet.
Da die Daten im ASCII-Format geschrieben werden, können sie jede Art von Daten und Text enthalten, es ist aber sinnvoll als Trennzeichen Tabulator (9) und Zeilenvorschub (10,13) zu verwenden, damit sich die Daten problemlos z.B. in eine Tabellenkalkulation übernehmen lassen.

Formatierung

Die Karte sollte am PC vorformatiert werden, damit der Bootsektor stimmt.
Anschliessend kann der Mikrokontroller den Bootsektor gezielt auf NOFAT ändern.

Routine zur Umformatierung in BASCOM:

```
Reformat:
'Karte auf NOFAT umformatieren
'ruft Clearabuffer + Changebootsector auf.

  'Info aus Boot Sektor lesen
  Gosub Readdriveinfo

  'und gezielt ändern
  Gosub Changebootsector

  Gosub Clearabuffer

 Return
'-----------------------------------------------------------------------
Clearabuffer:
'Datenpuffer mit Nullen füllen
        For I = 1 To 512
           Abuffer(i) = 0
        Next I
Return
'-----------------------------------------------------------------------------
Changebootsector:
'Bootsektor anpassen auf Dateisystem ohne FAT
'wird von Reformat aufgerufen

   'Overlay-Variablen anpassen -> Info steht in Abuffer an richtiger Stelle

   Bytespersector = 512
   Nbreservedsectors = 1
   Nbsectorspercluster = 1
   Nbfats = 0
   Nbsectorsperfat = 0

    'Für erste neue "Datei" Startsektor festlegen: hinter dem "Directory"
    Nextusablesector = 1

   Fatname = "NOFAT"
```

```
  Gbdriveerror = Drivewritesector(wsrampointer , Sectornr )
  Gosub Printerror
Return
```

Natürlich müssen alle Deklarationen und Variablen stimmen, siehe weiter hinten.

Besonders wichtig ist **Nextusablesector .** Diese Information wird jeweils nach dem Schreiben einer " Datei" aufgefrischt, damit der Controller weiss, ab welchem Sektor Platz frei ist für die nächste "Datei".

Informationen über die Karte einholen

Dies muss geschehen bevor die Karte benutzt wird , um vor allem die Grösse der Karte und den nächsten freien Sektor abzufragen.
Hierzu dient die Routine Readdriveinfo.
Diese initialisiert auch die Karte.

```
Readdriveinfo:
'Wichtige Infos aus dem Bootsector lesen

        Gosub Init_sdcard

        'Sektor 0 ist entweder Bootsektor (erkennbar an Beginn mit Sprungbefehl EB)
        'oder Partitionstabelle (beginnt mit 00 ?)
        Ltmp = 0
        Gbdriveerror = Drivereadsector(wsrampointer , Ltmp )       'dauert 8ms
        Gosub Printerror

        'Ist es  der Bootsektor oder die Partitionstabelle?
        If Abuffer(1) <> &HEB Then
          'wenn Partitionstabelle: Lage des Bootsektors aus dieser ermitteln
          'steht in Tmpbootsector nach Lesen von Sektor 0
          'und Bootsektor in Abuffer lesen
          Gbdriveerror = Drivereadsector(wsrampointer , Tmpbootsector )
          Gosub Printerror
        End If


        'Nach lesen des Bootsektors stehen
        '          Bytespersector
        '          Nbsectorspercluster
        '          Nbreservedsectors
                  'Nbfats
        '          Nbsectorsperfat
                  'Fatname
                  'NbsectorsinSmallpartition
                  'Nbsectorsinpartition
                  'NbEntriesInRoot
'       'in Abuffer() und ebenfalls (wegen OVERLAY)
        'in obigen Variablen
        ' (wenn die Karte FAT formatiert ist, sonst z.T. sinnlose Werte)

        'Es kann jetzt die Info über die Anzahl der Sektoren in
        'NbsectorsinSmallpartition  oder  Nbsectorsinpartition
        'stehen.
        'Wir benutzen Nbsectorsinpartition
        If Nbsectorsinsmallpartition <> 0 Then
          Nbsectorsinpartition = Nbsectorsinsmallpartition
        End If

        'Da Nbsectorsinpartition weg ist, wenn neuer Sektor gelesen wird,
        'die Information über die Anzahl Sektoren  in   Totalnbsectors dauerhaft speichern
        Totalnbsectors = Nbsectorsinpartition

Return
```

Readdriveinfo wurde zuerst geschrieben, um eine DOS-Partition zu lesen und enthält daher eine Menge Variablen, die hier eigentlich gar nicht nötig sind. Diese wurden aber beibehalten, sie nehmen dank OVERLAY-Technik keinen zusätzlichen Speicherplatz weg. Allerdings sthen sie auch nur temporär nach Lesen des Bootsektors zur Verfügung.

Readdriveinfo berücksichtigt auch, dass manche Karten nicht mit dem Bootsektor, sondern mit einem Master Boot Record (MBR) beginnen. In diesem Fall wird zunächst dieser und dann der Bootsektor gelesen.

Achtung: eine Karte mit MBR enthält diesen nach dem Umformatieren auf NOFAT nicht mehr !

Readdriveinfo benutzt Init_Sdcard.
Die Routine Init_Sdcard wird hier zur Sicherheit jedesmal mit aufgerufen. Ein Aufruf zu Beginn des Hauptprogramms würde auch genügen, wenn nicht, wie hier im Beispiel, die Pins des SPI-Ports auch für ein LCD-Display mitbenutzt

werden. Dann müssen sowohl SD-Karte als auch LCD vor jedem Gebrauch initialisiert werden.

```
IInit_sdcard:

'Hardware - SPI - Initialisierung:
  ' Chip Select Pin (kann SS\ des AVR sein, muss aber nicht.)
  Config Pind.7 = Output
  Mmc_cs Alias Portd.7
  Set Mmc_cs

  ' Hardware-SPI: SS\-Pin muss Ausgang und High sein, auch wenn für Chip Select der Karte
  ' ein anderer Pin benutzt wird,
  'damit der Controller als SPI-Master und nicht als Slave funktioniert!
  Config Pinb.2 = Output
  Spi_ss Alias Portb.2
  Set Spi_ss                    ' Set SPI-SS to Output and High por Proper work of
                                  ' SPI as Master

  ' HW-SPI mit maximaler Geschwindigkeit:
  Config Spi = Hard , Interrupt = Off , Data Order = Msb , Master = Yes , Polarity = High , Phase = 1 , Clockrate = 4 , Noss = 1
  Spiinit

  'keine Fremdspannung an MISO!
  Config Pinb.4 = Input
  Portb.4 = 0

  'Ein bisschen warten bis SPI bereit
  Waitms 1

  Gbdriveerror = Driveinit()  ' Init MMC/SD Card
  Gosub Printerror

  Return
```

Schreiben einer "Datei", Erklärung der Unterprogramme

Dies geschieht in 3 Teilen:

3. CreateNewFile legt eine neue "Datei" an.
4. Anschliessend werden die Daten geschrieben.
5. Dann wird mit CloseFile die "Datei" geschlossen.

6. **CreateNewFile**

```
Createnewfile:
'
 'Infos aus Boot Sektor lesen
  Gosub Readdriveinfo

  'Beginn der neuen Datei:
  Sectornr = Nextusablesector
  Print "Open new file, sector " ; Sectornr

  'Index von Abuffer initialisieren, wird vor jedem neuen Byte erhöht
   Abufferindex = 0

  'BEGIN OF FILE schreiben
     Ssbuffer = "<BOF>"
     Addstring
     Addcrlf

Return
```

Die neue "Datei" wird hinter der zuletzt geschriebenen angelegt, hierzu wird 'Sectornr auf Nextusablesector gesetzt. (Nextusablesector wird von ReaddriveInfo gelesen)

2. Schreiben der Daten

Hier ist zu beachten, dass die Karte immer sektorweise und nicht byteweise beschrieben wird.
Immer wenn ein Sektor mit 512 Byte voll ist, muss dieser sofort geschrieben werden.

Um diesen Vorgang zu vereinfachen und die Programme übersichtlicher zu machen, wurden die SUBs Addstring, AddTab und AddCRLF geschrieben.

Die Daten-Bytes werden in den Datenpuffer Abuffer() geschrieben, wenn dieser voll ist, erfolgt automatisch ein Schreiben des Sektors.

Achtung: Addstring benutzt die Variable ssbuffer (String) , die OVERLAY mit sbuffer() (Array) deklariert ist, um die Daten zu übergeben.

```
Sub Addstring
'String in ssbuffer zu Abuffer hinzufügen
' (Achtung auf die Länge!)

   Local K As Byte


      'zu Abuffer hinzufügen
      Len_sbuffer = Len(ssbuffer)
      For K = 1 To Len_sbuffer

         Abufferindex = Abufferindex + 1
         Abuffer(abufferindex) = Sbuffer(k)

         'Abuffer voll?
         Checkabuffer

      Next K
End Sub
'-------------------------------------------------------------------------
Sub Addcrlf
    'Zeilenvorschub

      Abufferindex = Abufferindex + 1
      Abuffer(abufferindex) = 13
      Checkabuffer

      Abufferindex = Abufferindex + 1
      Abuffer(abufferindex) = 10
      Checkabuffer
```

```
    End Sub
'-------------------------------------------------------------------------------
Sub Addtab
   'TAB hinzufügen
      Abufferindex = Abufferindex + 1
      Abuffer(abufferindex) = 9

      'Abuffer voll?
       Checkabuffer
End Sub
```

Alle Add...-Programme benutzen Checkbuffer zum Überprüfen von Abuffer:

```
Sub Checkabuffer()
'wenn Abuffer voll ist, Sektor abspeichern und neuen beginnen.

If Abufferindex = 512 Then
   'ABuffer voll! Abspeichern!
    'Sektor schreiben:
    Print "Sector:" ; Sectornr
     Gbdriveerror = Drivewritesector(wsrampointer , Sectornr )
        Gosub Printerror

   'neuer Sektor
    Sectornr = Sectornr + 1
    If Sectornr > Totalnbsectors Then
       Print "Card full!"
       Sectornr = 1
    End If

   'Bufferindex initialisieren
    Abufferindex = 0
 End If
End Sub
```

Wenn diese Routinen im Quelcode vorhanden sind, geht das Hinzufügen neuer Daten sehr einfach, z.B.:

```
'    'Header
     Ssbuffer = "TEST DATA"          '***
     Addstring
     Addcrlf


    '1000 Datensätze erzeugen
  For I = 1 To 1000                  '***

     'Datensatz erzeugen:
     T = I                        '***
     X = &H0F                     '***
     Y = &HF0                     '***
     Z = &HFF                     '***

     'Print T
     Addtoabuffer T               '***
     Addtab
     Addtoabuffer X               '***
     Addtab
     Addtoabuffer Y               '***
     Addtab
     Addtoabuffer Z               '***

     Addcrlf
  Next I
```

7. Schliessen der “Datei”

```
Closefile:
'Datei schliessen

'End of file - Markierung einfügen
  Ssbuffer = "<EOF>"
  Addstring
  Addcrlf

''Rest von Abuffer mit 0 füllen (nicht unbedingt nötig, aber nützlich):
Abufferindex = Abufferindex + 1
For Wtmp = Abufferindex To 512
   Abuffer(wtmp) = 0
Next Wtmp
```

```
'Letzten Datensatz noch abspeichern
Print "Save sector " ; Sectornr
Gbdriveerror = Drivewritesector(wsrampointer , Sectornr )
        Gosub Printerror

'Infos lesen + ändern:
Print "Update sector 0"
Gosub Readdriveinfo
'Infos stehen jetzt in Abuffer

'ändern:
'Aktuelle Sektornummer +1 = Nextusablesector  in Sektor 0 abspeichern
Nextusablesector = Sectornr + 1
Print "Next usable sector: " ; Nextusablesector

'schreiben
Ltmp = 0
Gbdriveerror = Drivewritesector(wsrampointer , Ltmp )
        Gosub Printerror


Return
```

Schreiben einer “Datei”, Benutzung der Unterprogramme

Beispiel:

```
'Neue Datei öffnen (Nextusablesector wurde von Readdriveinfo ermittelt)
   Gosub Createnewfile

     'Header schreiben
     Ssbuffer = "TEST DATA"          '***
     Addstring
     Addcrlf


    'Datensätze schreiben
  For I = 1 To 1000                  '***

      'Datensatz erzeugen:
      T = I                        '***
      X = &H0F                     '***
      Y = &HF0                     '***
      Z = &HFF                     '***

      'Print T
      Addtoabuffer T               '***
      Addtab
      Addtoabuffer X               '***
      Addtab
      Addtoabuffer Y               '***
      Addtab
      Addtoabuffer Z               '***

      Addcrlf
  Next I

'Datei schliessen
   Gosub Closefile
```

Die mit ‘*** versehenen Befehle müssen an die spezielle Aufgabenstelung angepasst werden.

Deklarationen und Variablen in BASCOM

Damit der oben erklärte Code lauffähig ist, sind natürlich etliche Deklarationen nötig.

Low-Level

(sektorweise Zugriff auf die Speicherkarte)
Hierzu wird die MMC-Library von A.Vögele benutzt → http://members.aon.at/voegel/index.html

```
'-------------------------------------------------------------------------------
 'DEKLARATIONEN für SD-Karte (Low Level):
'Datenpuffer, Achtung Arrayindex beginnt bei 1
Dim Abuffer(512) As Byte              ' Hold Sector to and from CF-Card
Dim Wsrampointer As Word              ' Address-Pointer for write
Wsrampointer = Varptr(abuffer(1))
Dim Sectornr As Long                  ' aktueller Sektor zum Schreiben oder Lesen
Const Cmmc_soft = 0                   'Hardware-SPI

' Errorcodes
Const Cperrdrivereset = 225           ' Error response Byte at Reset command
Const Cperrdriveinit = 226            ' Error response Byte at Init Command
Const Cperrdrivereadcommand = 227     ' Error response Byte at Read Command
Const Cperrdrivewritecommand = 228    ' Error response Byte at Write Command
Const Cperrdrivereadresponse = 229    ' No Data response Byte from MMC at Read
Const Cperrdrivewriteresponse = 230   ' No Data response Byte from MMC at Write
Const Cperrdrive = 231
Const Cperrdrivenotsupported = 232    ' return code for DriveGetIdentity, not supported yet


Dim Gbdriveerror As Byte              ' General Driver Error register
Dim Gbdriveerrorreg As Byte           ' Driver load Error-Register of HD in case of error
Dim Gbdrivestatusreg As Byte          ' Driver load Status-Register of HD on case of error
Dim Gbdrivedebug As Byte


$lib "MMC.LIB"                        ' link driver library
$external _mmc
'-------------------------------------------------------------------------------
```

High Level (NOFAT - System)

```
'Variablen für  Readdriveinfo  (alles Overlay Abuffer, brauchen also keinen zusätzlichen Speicherplatz!)
'werden nur für Readdriveinfo benötigt, und zwar auch nicht alle 'wenn nur NOFAT benutzt wird
'stehen nach Lesen von Sektor 0 in Abuffer und somit in diesen Variablen:
Dim Bytespersector As Word At Abuffer + &HB Overlay
Dim Nbreservedsectors As Word At Abuffer + &HE Overlay
Dim Nbsectorspercluster As Byte At Abuffer + &H0D Overlay
Dim Nbfats As Byte At Abuffer + &H10 Overlay
Dim Nbentriesinroot As Word At Abuffer + &H11 Overlay
Dim Nbsectorsperfat As Word At Abuffer + &H16 Overlay
Dim Nbsectorsinsmallpartition As Word At Abuffer + &H13 Overlay
Dim Nbsectorsinpartition As Long At Abuffer + &H20 Overlay
Dim Fatname As String * 8 At Abuffer + &H36 Overlay
Dim Oemname As String * 8 At Abuffer + 3 Overlay
Dim Tmpbootsector As Long At Abuffer + 454 Overlay
'-------------------------------------------------------------------------------
'Variablen und SUBs spezifisch NOFAT:
'für Sektor 0 (Drive-Infos):
Dim Nextusablesector As Long At Abuffer + &H3E Overlay

Dim Totalnbsectors As Long            ' steht auch nach Readdriveinfo + Sectornr geändert zur Verfügung

'Für Umwandlung der Daten in Strings:
Dim Ssbuffer As String * 20
Dim Sbuffer(20) As Byte At Ssbuffer Overlay
Dim Len_sbuffer As Byte
Dim Abufferindex As Word

'Für bequemes Schreiben in die "Datei":
 Declare Sub Addtoabuffer(tmp As Word)
 Declare Sub Addcrlf
 Declare Sub Checkabuffer
 Declare Sub Addtab
 Declare Sub Addstring

'Temporäre Variablen für SD-Karte und Hauptprogramm und wer immer sie braucht

Dim Wtmp As Word
Dim Btmp As Byte
Dim Ltmp As Long
```