Moppel Hardware

Interface
CF-Karte

(Stand 07.04.2016)

Inhalt:

Anlagen:

- Schaltbilder:
  - ECB-Teil (jpg, Eagle)
  - AVR-Teil (jpg, Eagle)
- Software:
  - Moppel
    - Testprogramme
    - Auszug aus FDC-Routinen
  - AVR
    - Quellcode in BASCOM

Vorwort:

Mit diesem Interface erhält der Moppel Anschluß zu moderneren Massenspeicher. Es gibt viele Beschreibungen im Netz, angefangen von der direkten Ansteuerung über ein Pio-Baustein bis hin zur FPGA-Lösungen. Ich habe mich für ein Mittelweg entschieden, da ich für den einfachen Datenaustausch Moppel ↔ PC ein FAT-16 Dateisystem haben wollte, ist das nur mit einer PIO nicht getan, denn im Moppel-BIOS kann das nicht abgebildet werden - einfach zu umfangreich.

In meiner Lösung übernimmt ein Atmega644 (im weiteren Verlauf einfach als AVR bezeichnet) die Hauptarbeit, da es unter BASCOM eine entsprechende Bibliothek gibt, die das gewünschte Dateisystem bereitstellt. Zudem gibt es dann noch eine schnelle V24 Schnittstelle als Zugabe. Es können auch SD-Karten mit dem entsprechendem Modul (z.B. Conrad SD-Card Interface oder ähnliches) genutzt werden. Damit könnten viele PIN's eingespart werden, ich habe mich aber für die CF-Karten entschieden, da sie nach meinem Geschmack besser zur alten Moppel-Hardware passen.

Anmerkung: Im Laufe der Entwicklung habe ich festgestellt, dass in dieser Kombination nur CF-/SD-Karten größer 1GB laufen, die kleineren haben erhebliche Timingprobleme ! Das ist zwar Overdress, denn CP/M kann nur 8MB auf einem Datenträger verwalten. Hier könnte man noch „Jugend-forscht“ betreiben und das Timing mit den kleinen Karten untersuchen ...

Hardware:

Die Karte ist in drei Funktionsblöcken gegliedert:

1. ECB-Teil

   Bustreiber, Adressauswahl

2. Pralellinterface

   hier werden die Reste des PIO-Bausteins genutzt.

   z.B. Tastaturinterface, IO-Byte für die grundlegenden Systemeinstellungen

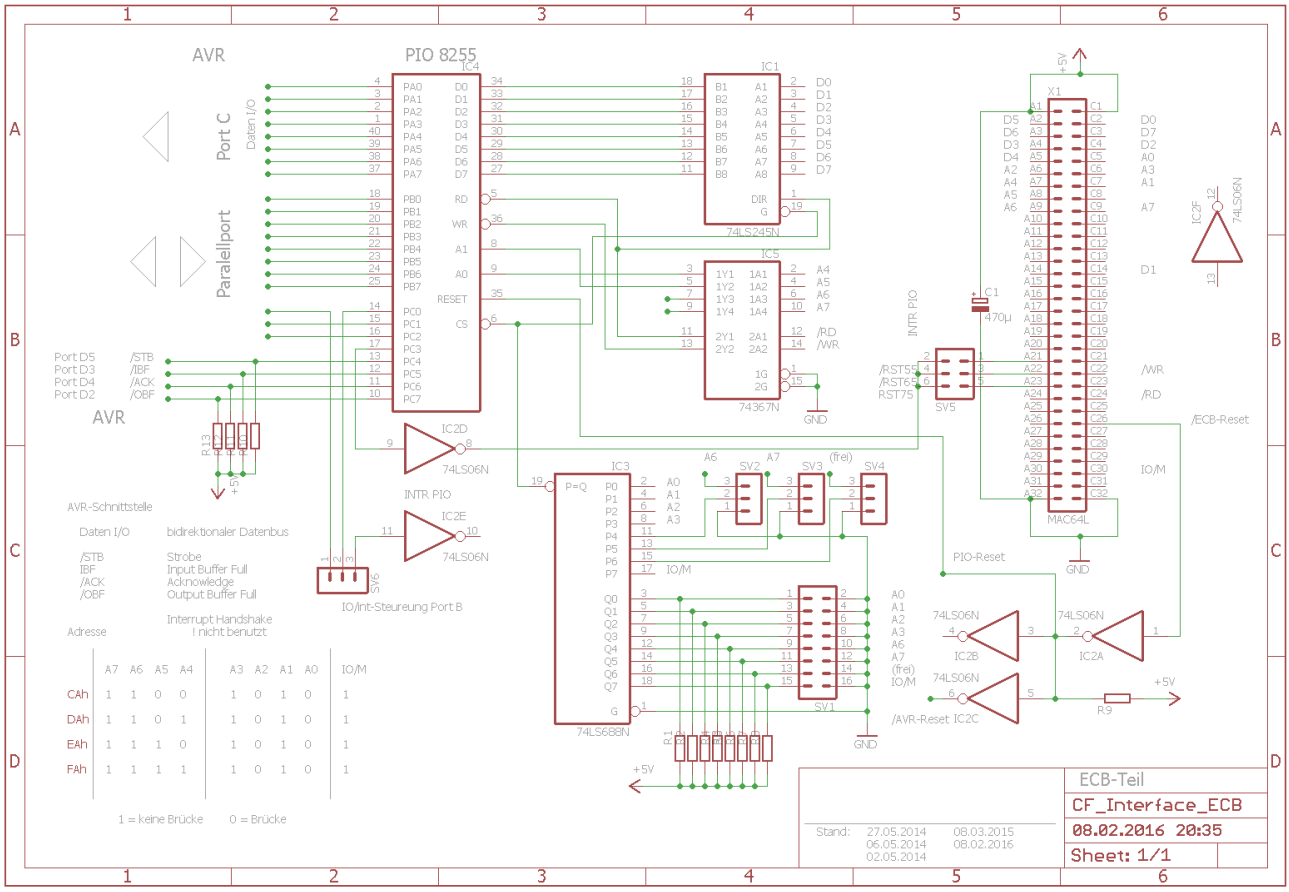
   Drucker, nicht getestet da für Busy, Error. Papierende etc. zu wenig Pin's zur Verfügung stehen.

3. AVR-Teil

   Schnittstelle zur CF-Karte (bei entsprechendem Modul SD-Karte)

   V24-Schnittstelle mit Handshake (RTS,CTS)

zu 1 ECB-Teil:



neben den Datenbustreiber ermöglicht der 74LS688 die frei Adresswahl im IO-Bereich

Die Karte (PIO) belegt 4 Adressen im IO-Bereich (CAh bis FAh)

PIO Port A	Datenaustausch mit dem AVR-Teil
PIO Port B	frei bzw. einfacher Ein-Ausgabekanal
PIO Port C	Steuer- Handshakeleitungen für Port A und B
PIO Controlregeister	

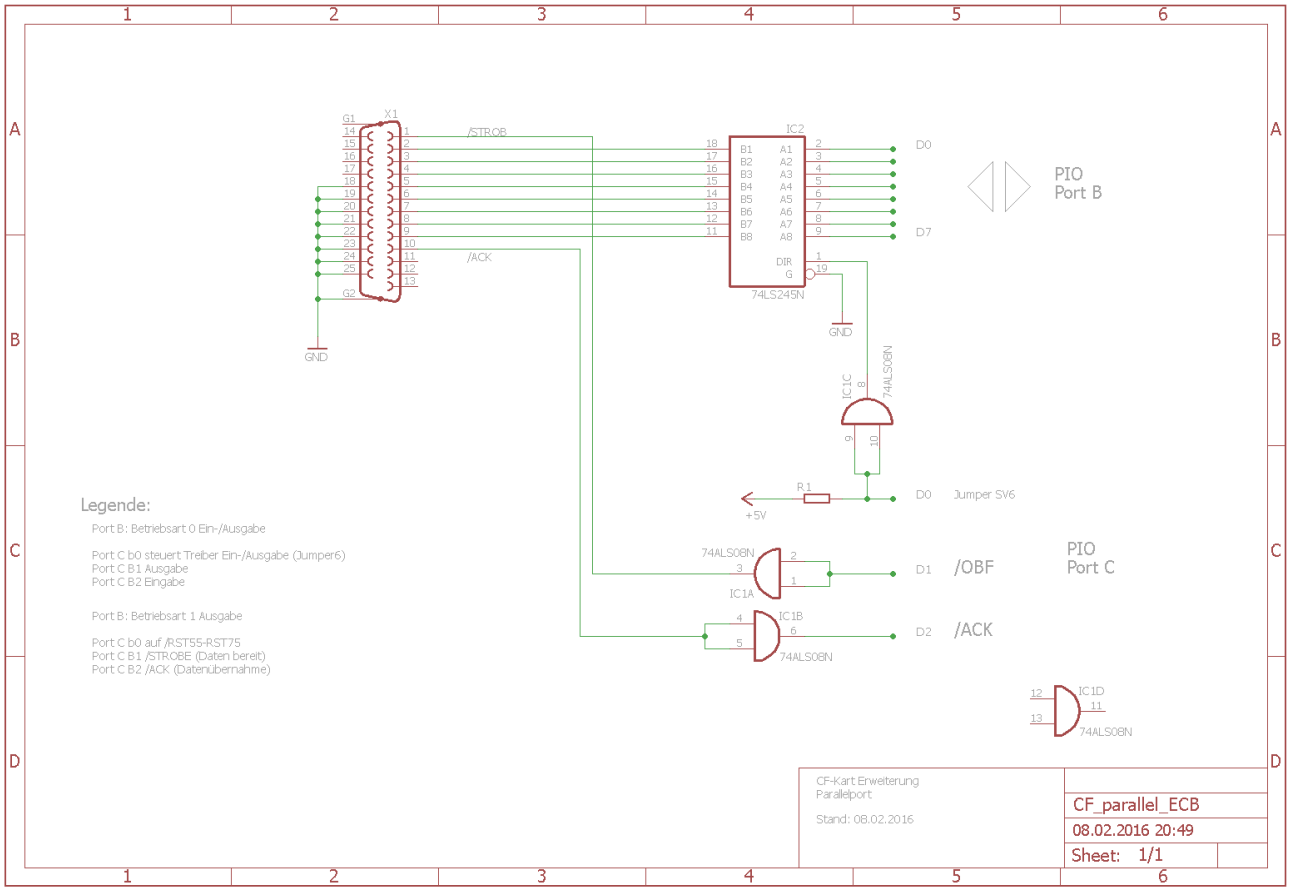
Der PIO-Baustein dient dem Datenaustausch mit dem AVR-Teil. Hierzu wird der Port A im Modus 2 (getasteter 2-Weg BUS Ein-Ausgabe) betrieben. Durch die Handshake Signale /OBF, /ACK und /STB, IBF ist die Datenübergabe synchronisiert.

Port B kann als zusätzlicher Ein-/Ausgabekanal benutzt werden, Port C Bit 0 steuert die Treiberrichtung oder in der Betriebsart 1 (getastete Ausgabe) mit den Steuerleitungen /OBF, /ACK. (Port C Bit1 und 2 – Bit0 kann über Jumper SV6 auf Int 7.5 gelegt werden.

Anmerkung:

Interruptbetrieb ist vorbereitet, Treiber und Jumper SV5, SV6 für /RST5.5 bis /RST7.5

Zu 1b Paralleles Interface:



Port B als Aus- bzw. Eingabekanal

Mit dem Steuerwort C1h ist der Kanal B als Ausgabe programmiert. Der Datentreiber wird hierzu über R1 auf Ausgabe geschaltet (Standard in meiner Software).

Die getastete Ausgabe kann mit dem Steuerwort C4h eingestellt werden. Dabei übernimmt der Port C die Steuerleitungen C1 = /OBF und C2 = /ACK.

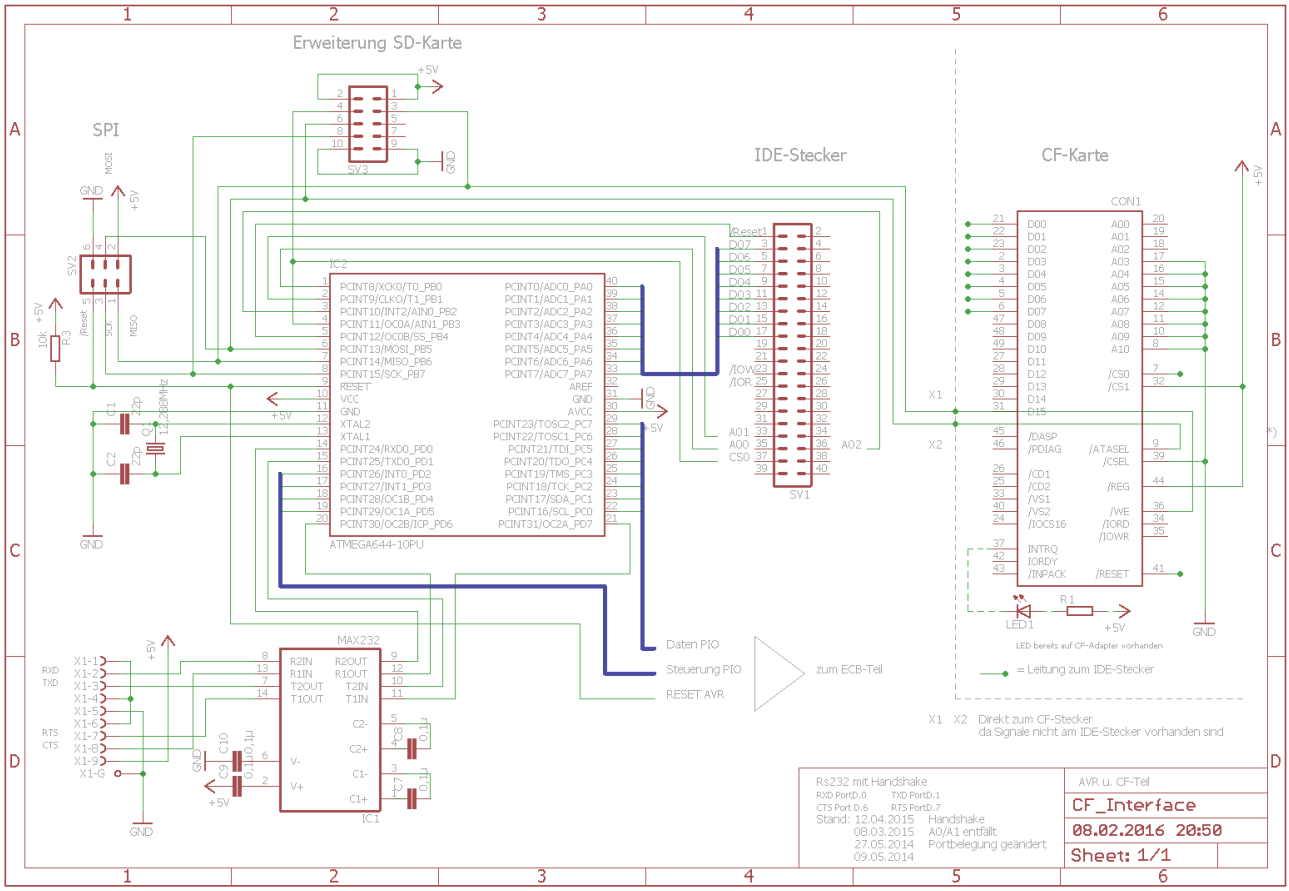
Anmerkung:

Für eine vollwertige Druckerschnittstelle fehlen die Steuerleitungen BUSY, ERROR etc.

Wenn Kanal B als Eingang genutzt werden soll, muss der Jumper SV6 auf Port-C0 gelegt werden damit der Datentreiber die Datenrichtung umschaltet. Hierbei ist das Bit0 von Kanal C auf L-Pegel zu schalten.

Oder der Pin 2 von Jumper SV6 wird auf GND gelegt, dann ist auch getastete Eingabe (Betriebsart 1) möglich um z.B. eine parallele Tastatur anzuschließen.

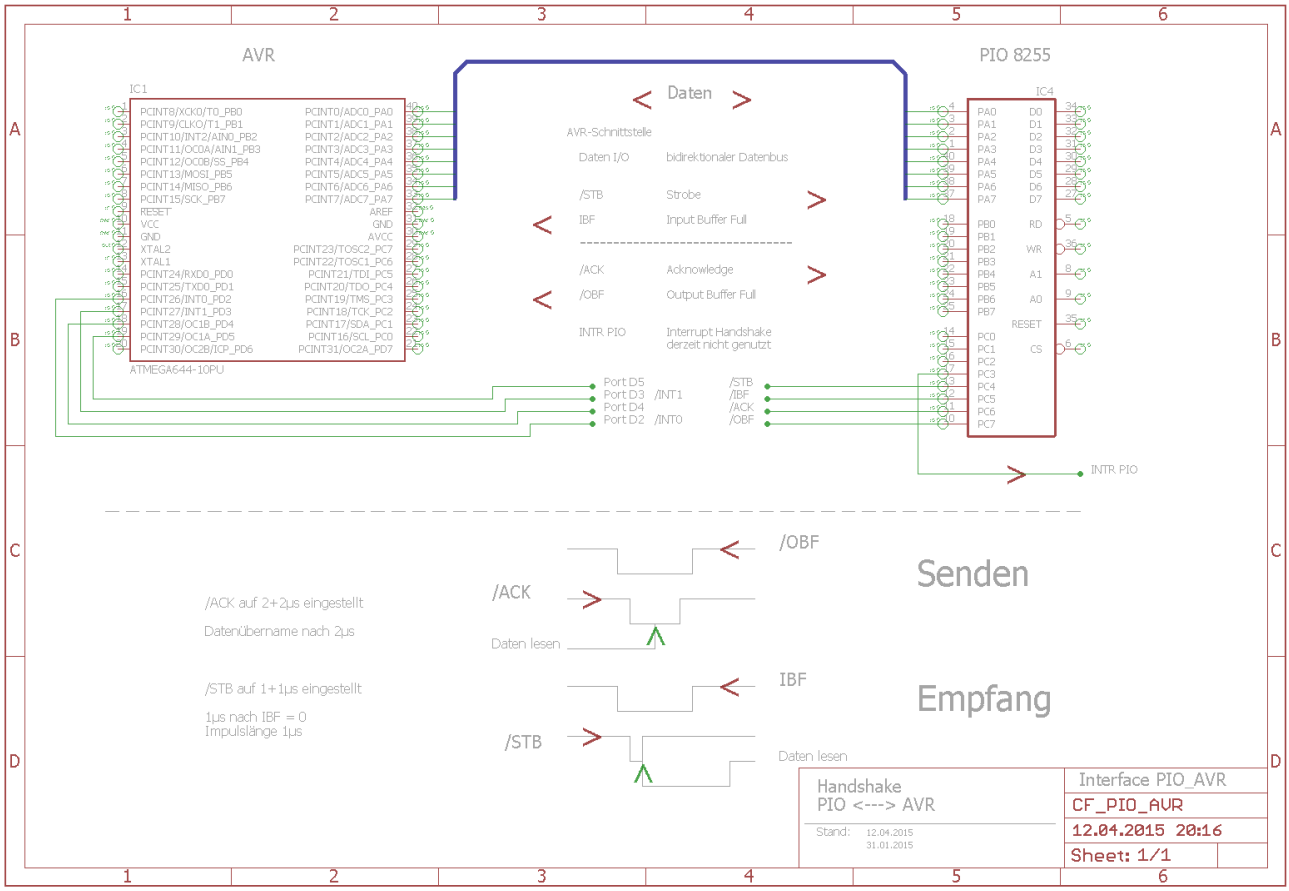
Zu 2. AVR-Teil:



Als „ intelligenten IO-Baustein“ nutze ich hier den Atmega 644, er hat mit 64kByte ROM genügend Speicherkapazität um das AVR-DOS aufzunehmen und ausreichend IO-Pins für die Anbindung der CF-Karte, sowie für den Datenaustausch mit der PIO.

Die RS232 vom Atmega ist mit den Pegeltreiber (Max232) und den Steuerleitungen RTS/CTS ausgestattet. Mit dem installiertem MCS-Boot-Loader ist die Programmierung in Sekunden schnelle erledigt und dient in der Testphase erst mal als Monitor für die Befehlsabläufe. Wenn alles eingerichtet ist, hat der Moppel hierüber einen schnellen Draht zur Außenwelt, derzeit 38kBit/s (nicht ganz - da für die Steuerung noch ein paar Befehle ausgetauscht werden müssen.

Handshake:



Senden (aus Moppelsicht)

/OBF = Output Buffer Full → Daten liegen im Ausgaberegister

/ACK = Quittungssignal vom AVR

Empfangen (aus Moppelsicht)

/STB = Strobe → Daten können übernommen werden

IBF = Input Buffer voll → Daten wurden übernommen

Moppel-Software:

Zunächst muss die PIO für den Datenaustausch entsprechend initialisiert werden, also das Steuerwort geladen werden.

```
; PIO Ansteuerung
PIO_A   equ     0cah            ; CAh   Daten Kanal A
PIO_B   equ     0dah            ; DAh   Daten Kanal B
PIO_C   equ     0eah            ; EAh   Daten Kanal C
PIO_Ctl equ     0fah            ; FAh   Steuerwort
PIO_BA equ      0c1h            ; Steuerwort
;
pioinit:push    psw             ; PIO Betriebsart einstellen
        mvi     a,PIO_BA        ;
        out     PIO_Ctl         ;
        pop     psw             ;
        ret                     ;
;
```

und hier die beiden grundlegenden Programmteile um ein Byte zu lesen bzw. zu schreiben (Moppelsicht). Im getaktetem Betrieb des Pio-Bausteins müssen die Handshake-Signale entsprechend ausgewertet werden. Um den Datentransport kümmert sich die PIO dann selbstständig. Hierdurch wird der Datenfluss synchronisiert und jeder „Gesprächspartner“ wartet auf den anderen ;-)

```
; PIO-A lesen (A)
; wartet bis Daten bereitstehen
piord:          in      PIO_C       ; Status ermitteln
                ani     00100000b   ; IBF maskieren
                jz      piord       ; warte auf Daten
                in      PIO_A       ; daten lesen
                ret                 ; (A) Daten
;
; (A) PIO-A schreiben
piowr:          push    psw         ;
pio_sts: in     PIO_C               ; Status ermitteln
                ani     10000000b   ; /OBF maskieren
                jz      pio_sts     ; warten auf Quittung /ACK
                pop     psw
                out     PIO_A       ; Daten schreiben
                ret                 ;
```

Da ich auf „Adressregister“, wie sonst bei intelligenten Peripherie-Baustein üblich, verzichtet habe, muss ein anderer Mechanismus den Datenfluss steuern. Hierzu habe ich ein paar Regeln eingeführt, damit der AVR weis was er mit den Daten machen soll.

Jede Datenübertragung wird mit einem Befehl eingeleitet, darauf folgen dann die Daten. Dies hat natürlich zur Folge, dass der Datenblock immer vollständig abgearbeitet werden muss, sonst hängt sich das System auf. Zum Beispiel ist es wichtig das genau 512Byte beim Sektor übertragen werden oder bei der Übertragung eines Strings ist dem Bufferüberlauf besonderer Aufmerksamkeit zu widmen !!!.

Befehlssatz:

```
;
AVRres      equ   00h       ; AVR zurücksetzen (Spur 0 Sektor 0 ansteuern)
AVRstat     equ   0F0h      ; AVR Status ermitteln
                            ; 00h     = OK - CF-Karte vorhanden
                            ; <> 0    = Fehler – CF-Karte fehlt
;
; CF-Karte
SetSpur     equ   10h       ; Spurnummer setzen
SetSekt     equ   20h       ; Sektornummer setzen
Rdsekt      equ   30h       ; Daten vom aktuellen Sektor lesen
Wrsekt      equ   40h       ; Daten in den aktuellen Sektor schreiben
;
; V24
v24btx      equ   52h       ; ein Byte senden
V24brx      equ   53h       ; ein Byte von V24 empfangen
v24strtx    equ   54h       ; String senden (einschließlich CR + 00h)
V24strrx    equ   55h       ; String empfangen  (einschließlich CR + 00h)
V24status   equ   5fh       ; Status der V24 Schnittstelle
                            ; Bit 0   CTS-Signal
                            ; Bit 1   RTS-Signal
                            ; Bit 5   Senderegister leer
                            ; Bit 7   Empfangsregister voll
```

Diese Status-Signale können/sollten vor der Übertragung vom Moppel ausgewertet werde damit er nicht irgendwo sinnlos wartet. Der Datenfluss wird über RTS/CTS vom AVR abgewickelt.

Das ganze ist als Halbduplexbetrieb ausgelegt, da der Moppel ohne Interrupt auskommt und nur ein Buffer definiert ist.

CF-Karte:

Die CF-Karte ist auf Moppel-Ebene als 4. Laufwerk eingebunden. Der ideale Ort für meine Ergänzungen war einmal der Kaltstart, sowie die Initialisierung für Sektor schreiben/lesen:

```
FKALT: call    cfinit          ; Test ob CF-Interface vorhanden
       .
       .
;-------------------------------------------------------------------------------------------
INIT:  DI                      ; diese Routine wird jedes mal beim schreiben/lesen
       SHLD    DATADR          ; angesprungen
       PUSH    B
       .
       .
       .
       POP     B
       call    cfq             ; Test auf Laufwerk 4 CF-Karte
       RET
.
.
;============================================================
;
; CF-karte lesen/schreiben
;  Laufwerk 4 = CF_Karte
;
; Parameterübergabe:
;
; TRKNR = Spur
; SECNR  = Sektor
; DATADR = Buffer Zieladresse
; BUFSCR = Anzahl der zu übertragenden Sektoren
;         auf 512Byte/Sektor angepasst
;
;============================================================
; cfq = Test ob Laufwerk = 4
;           wenn nein weiter mit Floppy
;      und Test auf rd/wr Sektor aus RWSUB
;
cfq:       push      psw
```

```
        lda     selbyt          ;
        cpi     1eh             ; Test ob Laufwerk 4
        jnz     cfq_e           ; nein weiter mit Floppy
        lda     rwsub1          ; RWSUB1 testen
        cpi     40h             ;
        jz      cfwrbl          ; ja CF-Karte schreiben
        cpi     66h             ;
        jz      cfrdbl          ; ja CF-Karte lesen
cfq_e   pop     psw
        ret
;
;===============================================================
```

Die CF-Karte haben eine Sektorgröße von 512Byte, die Moppellaufwerke arbeiten mit 256Byte/Sektor und CP/M mit Records zu 128Byte, hier ist also eine Umrechnung fällig. Glücklicherweise hat Herr Gößler damals das BIOS für CP/M mit einem 1kByte großen Buffer bestückt, der immer in einem Rutsch von bzw. zur Diskette geschaufelt wird. So ist das für den AVR nicht sonderlich schwierig, Division durch 2 + Rest (Modulo).

Es gibt aber ein Pferdefuß in den originalen Floppyroutinen. Die Anzahl der logischen Sektoren (Record a 128byte) ist auf 32 begrenzt. Bei CF-Karten gibt es keine Spuren sondern nur Sektoren. Diese Begrenzung ist bei der Berechnung des DPB zu berücksichtigen. Glücklicherweise ist die Spuranzahl als Word deklariert, so kommt man auf eine Gesamtkapazität von 8MB, die CP/M auch verwalten kann.

Die vollständige Beschreibung zum DPB findet ihr hier:

http://hc-ddr.hucki.net/wiki/doku.php/cpm:write_a_bios:teil_1

AVR-Software:

Auf der AVR-Seite habe ich die Software in BASCOM von MCS geschrieben, hierfür gibt es die passenden Treiber für die Ansteuerung von CF-, SD-Karten sowie das AVR-DOS mit dem FAT16 -Dateisystem.

Auch hier wieder die entsprechenden Definitionen:

```
' Daten ECB
Config Portc = Input
' Handshake ECB
Config Portd.2 = Input
Config Portd.3 = Input
Config Portd.4 = Output
Config Portd.5 = Output
Hs_obf Alias Pind.2                ' /OBF                '
Hs_ibf Alias Pind.3                ' IBF
Hs_ack Alias Portd.4               ' /ACK
Hs_stb Alias Portd.5               ' /STB
```

' AVR Befehle (Beschreibung wie oben)

```
Const Avrres = &H00                '
Const Avrstat = &H0F0
Const Setspur = &H10
Const Setsek = &H20
Const Rdsek = &H30
Const Wrsek = &H40

Const V24status  =&H5f             ; Status der V24 Schnittstelle
Const v24btx     =&H52             ; ein Byte senden
Const V24brx     =&H53             ; ein Byte von V24 empfangen
Const V24strtx   =&H54             ; String senden (einschließlich CR + 00h)
Const V24strrx   =&H55             ; String empfangen  (einschließlich CR + 00h)
```

.
.
.

Hier die grundlegenden Ein-Ausgaberoutinen auf der AVR-Seite

```
' Daten lesen (AVR-Sicht)
'
Sub Ecb_rdbyte
  Config Portc = Input
  Do                              ' warten auf Daten
  Loop Until Pind.2 = 0           ' /OBF = 0
  Reset Hs_ack                     '
  Waitus 2
  In_tmp = Pinc                    '  Daten lesen
  Waitus 2                         ' /ACK Quittung
  Set Hs_ack
'
End Sub
'
' Daten schreiben (AVR-Sicht)
'
Sub Ecb_wrbyte
  Config Portc = Output

  Do                              ' warten auf Bereit
  Loop Until Pind.3 = 0                ' IBF = 0

  Portc = Out_tmp                      ' Daten ausgeben
  Waitus 1
  Reset Hs_stb
  Waitus 1                        ' /STB ausgeben
  Set Hs_stb

End Sub
```

Der „Befehlsinterpreter“ ist eine einfache, Do – Loop Schleife und entsprechenden Case-Anweisungen arbeiten dann mit Hilfe der Unterprogramme die eingerichteten Befehle ab

```
DO
   Call Ecb_rdbyte
   'If Testflg = 1 Then
    Print "Befehl: " ; In_tmp ' Für Testzwecke, verschwindet dann in der „endgültigen Fassung“
   'End If
   Select Case In_tmp
     Case Avrres
       Call Avr_reset              ' Reset
     Case Setspur
       Call Ecb_settr              ' Spur setzen
     Case Setsek
       Call Ecb_setsek             ' Sektor setzen
  .     usw       .
    End Select
 Loop
```

Damit ist das Grundgerüst für den Datenaustausch gelegt.

Dateisystem:

Die CF-Karte habe ich unter Windows mit dem FAT-16 Dateiformat formatiert und anschließend eine 8MB-Datei mit dem YACE-AG Emulator erzeugt. So ist das CP/M-Dateisystem in einem „DOS-Container“ verpackt. Der Startsektor für diese Datei ist dann als Offset bei der Sektorenberechnung zu berücksichtigen.

```
' physikalischen Sektor berechnen
' Floppyformat:
'
'   32 logische Sektoren/Track  (16 physikalische Sektoren a 256Byte)
'   80 Spuren
'
' ph_sektor = (Track * Sektor/Track) + log Sektor + Offset
'                              :          :
'                              :          erster Sektor CP/M
'                              :          im Datenbereich
'                             aus Befehl Ecb_setsektor
```

CF-Karte bereitstellen:

Hierzu habe ich den Z80Emulator von yaze-ag in der Version 2.3 genutzt.

Yaze starten und am Systempromt folgendes eingeben:

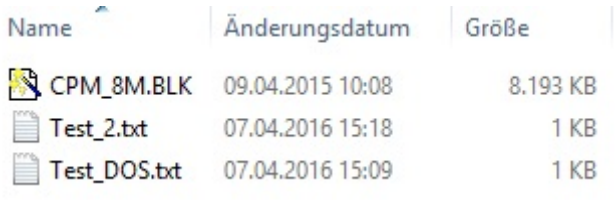
$>create disk 8m
$>mount b disk
$>mount -v

und erhält folgendes ...

A: r/o test/
dph=FD90, xlt=0000, dirbuf=FF80, dpb=FDA0, csv=0000, alv=FA66, spt=0100
bsh=05, blm=1F, exm=01, dsm=0100, drm=07EF, al=FFFF, cks=0000, off=0000
B: r/w disk
dph=FDAF, xlt=0000, dirbuf=FF80, dpb=FDBF, csv=0000, alv=FA87, spt=001A
bsh=04, blm=0F, exm=00, dsm=0FFF, drm=03FF, al=FFFF, cks=0000, off=0000
$>quit

In der Beschreibung unter Punkt 6 „CP/M Disk“ ist die Erzeugung detailliert beschrieben. Damit erhält man auf eine frisch formatierten (FAT-16) CF-Karte eine 8MB großen DOS -Container. Damit sie im Interface genutzt werden kann benötigt man noch den Offset, also die Sektornummer wo diese Datei wirklich anfängt.

Im DOS sieht es schon mal so aus



Die Datei Test_DOS.txt steht physikalisch vor dem CPM_8M.BLK und die Test_2.txt hinter dem CPM Container, damit kann dann im Hex-Editor die Lage einfacher herausgefunden werden.

und so im Hex-Editor bei Sektor 551:

```
4D 4F 50 50 45 4C 5F 31 20 20 20 08 00 00 00 00  MOPPEL_1   .....    Sektor 551
00 00 00 00 00 00 FB 78 87 48 00 00 00 00 00 00  ......ûx‡H......
E5 65 00 6E 00 74 00 2E 00 74 00 0F 00 A5 78 00  åe.n.t...t...¥x.
74 00 00 00 FF FF FF FF FF FF 00 00 FF FF FF FF  t...ÿÿÿÿÿÿ..ÿÿÿÿ
E5 4E 00 65 00 75 00 20 00 54 00 0F 00 A5 65 00  åN.e.u. .T...¥e.
78 00 74 00 64 00 6F 00 6B 00 00 00 75 00 6D 00  x.t.d.o.k...u.m.
E5 45 55 54 45 58 7E 31 54 58 54 20 00 80 05 79  åEUTEX~1TXT .€.y
87 48 87 48 00 00 06 79 87 48 00 00 00 00 00 00  ‡H‡H...y‡H......
41 54 00 65 00 73 00 74 00 5F 00 0F 00 32 44 00  AT.e.s.t._...2D.
4F 00 53 00 2E 00 74 00 78 00 00 00 74 00 00 00  O.S...t.x...t...
54 45 53 54 5F 44 4F 53 54 58 54 20 00 80 05 79  TEST_DOSTXT .€.y
87 48 87 48 00 00 27 79 87 48 02 00 18 00 00 00  ‡H‡H..'y‡H......
43 50 4D 5F 38 4D 20 20 42 4C 4B 20 00 5A AE 79  CPM_8M  BLK .Z®y
87 48 87 48 00 00 12 51 89 46 03 00 80 00 80 00  ‡H‡H...Q‰F..€.€.
E5 75 00 6D 00 65 00 6E 00 74 00 0F 00 A7 2E 00  åu.m.e.n.t...§..
74 00 78 00 74 00 00 00 FF FF 00 00 FF FF FF FF  t.x.t...ÿÿ..ÿÿÿÿ
E5 4E 00 65 00 75 00 65 00 73 00 0F 00 A7 20 00  åN.e.u.e.s...§ .
54 00 65 00 78 00 74 00 64 00 00 00 6F 00 6B 00  T.e.x.t.d...o.k.
E5 45 55 45 53 54 7E 31 54 58 54 20 00 99 21 7A  åEUEST~1TXT .™!z
87 48 87 48 00 00 22 7A 87 48 00 00 00 00 00 00  ‡H‡H.."z‡H......
41 54 00 65 00 73 00 74 00 5F 00 0F 00 38 32 00  AT.e.s.t._...82.
2E 00 74 00 78 00 74 00 00 00 00 00 FF FF FF FF  ..t.x.t.....ÿÿÿÿ
54 45 53 54 5F 32 20 20 54 58 54 20 00 99 21 7A  TEST_2  TXT .™!z
87 48 87 48 00 00 5A 7A 87 48 04 02 1A 00 00 00  ‡H‡H..Zz‡H......
```

und im Sektor 615 beginnt dann der CP/M-Container

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00 43 50 4D 5F 44 69 73 6B 80 00 00 00 00 00 00  .CPM_Disk€......    Sektor 615
01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  åååååååååååååååå
E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  åååååååååååååååå
E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  åååååååååååååååå
E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  åååååååååååååååå
E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  åååååååååååååååå
E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  åååååååååååååååå
```

Der Yaze-Emulator würde dann hier mit dem CP/M Direktory beginnen, da ich noch 4 Systemspuren reserviert habe beginnt das Moppel-Inhaltsverzeichnis bei Sektor 647

```
E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  åååååååååååååååå
00 44 44 54 20 20 20 20 20 43 4F 4D 00 00 00 26  .DDT     COM...&    Sektor 647
04 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 53 55 42 4D 49 54 20 20 43 4F 4D 00 00 00 0A  .SUBMIT  COM....
06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 53 54 41 54 20 20 20 20 43 4F 4D 00 00 00 29  .STAT    COM...)
07 00 08 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 41 53 4D 20 20 20 20 20 43 4F 4D 00 00 00 40  .ASM     COM...@
09 00 0A 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 50 49 50 20 20 20 20 20 43 4F 4D 00 00 00 3A  .PIP     COM...:
0B 00 0C 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5 E5  åååååååååååååååå
```

So bleiben 16kByte für den Bootloader und die Systemdateien CCP, BDOS und BIOS.

Dieses Prozedere mit der Erzeugung eines CP/M-Containers und die manuelle Bestimmung des Startsektors ist noch sehr umständlich. Im einfachsten Fall schreibt man einfach lustig auf einer frischen CF-Karte und der Moppel ist damit „Glücklich“. Da ich diese Karte noch ein Stückchen weiter entwickeln möchte, halte ich das aber für vertretbar.

V24 Übertragung:

Neben den Senden/Empfangen von einzelnen Bytes, können auch ganze Strings bis zu einer Länge von 64 Bytes (Einstellbar über den Parameter „MAXBUFF“) übertragen werden. Bei den Strings wird alles an ASCII-Zeichen plus CR übertragen. Dies wurde für die Übertragung von Intel-Hex-Dateien so gewählt und hat sich beim „Hexlader“ gut bewährt.

Im AVR wird der Buffer über Interupptroutinen gefüllt bzw. gesendet. Des weiteren wurde für den Datenflusssteuerung mit RTS eine Grenze bei 48Byte eingezogen (Maxrts), damit der Buffer nicht überläuft wenn der Moppel so gemächlich seine Daten abarbeitet.

Für den Fall des Bufferüberlaufs ist noch eine Notbremse eingebaut. Sobald das Bufferende erreicht ist ohne dass ein CR empfangen wurde, wird die Übertragung beendet und für den Empfänger ein CR angefügt. Was der damit macht, kann im vorgesehenem Betrieb eigendlich nicht auftreten – soweit die Theorie – ist seine Sache.

Wenn dies Probleme bereitet, muss da noch ein Status her, der Signalisiert dass die Datenübertragung einwandfrei war – ggf. durch ein Abschlußbyte – schauen wir mal …

```
' V24 Definitionen
'
$baud = 38400

Config Portd.7 = Output          ' RTS
Rts Alias Portd.7
Config Portd.6 = Input           ' CTS
Cts Alias Portd.6

' Buffer
'
const maxbuff = 64               ' Buffer Groesse Moppelbuffer
const rtsbuff = 48               ' Fuellgrad bis RTS gesetzt wird
dim v24buff(maxbuff) as byte     ' Buffer
```

Anmerkung: die Übertragung von Binärdaten ist (noch)nicht vorgesehen, dazu müssten die Daten Blockweise z.B. 256Bytes übertragen werden, da ja keine Prüfung auf CR oder 00h möglich ist. Dürfte ähnlich wie die Sektorweise Übertragung einfach zu realisieren sein. Hierzu bedarf es dann aber auf der PC-Seite ein spezielles Übertragungsprogramm ...

Schlußwort:

Bei der Grundsteinlegung dieses Projektes hätte ich nicht gedacht, dass es mich so lange beschäftigt und ein Ende ist noch nicht in Sicht. Da war mal die Idee, die Dateisysteme CP/M – DOS zu spiegeln, damit der Datenaustausch einfacher wird. Könnte mir die CF-Karte auch als FTP-Client vorstellen, wo der PC auf der AVR-Seite seine Daten ablegt bzw. abholt. Die schnelle V24 ist ja bereits verfügbar und der AVR nur zu 6% ausgelastet.

Hoffe das mit diesen Erläuterungen das Projekt nachvollzogen werden kann. Kritik wird gerne entgegen genommen …

An dieser Stelle möchte ich allen Dank sagen, die mit ihren Ideen und mich bei der einen oder anderen Problemstellung unterstützt haben.

Quellen:

http://www.prof80.de/index.html

http://www.mpm-kc85.de/index.htm

http://hc-ddr.hucki.net/wiki/doku.php/cpm:write_a_bios:teil_1

https://www.mikrocontroller.net/

http://www.forum.z80.de/

http://bascom-forum.de/

http://www.mathematik.uni-ulm.de/users/ag/yaze-ag/

und den viele kleine Ideen - Schnippsel aus dem Internet ...