Moppel Hardware

Interface
CF-Karte

(Stand 12.02.2016)

Inhalt:

Seite 2 …..... Hardwarebeschreibung, Portadressen
Seite 3 …..... Ablauf Handshake
Seite 4 …..... Moppel-Software
Seite 6 …..... AVR-Software
Seite 8 …..... Schaltbilder ECB/AVR Teil
Seite 9 …..... Schaltbild Parallelinterface

Anlagen:

Quellcode Moppel (Auszug aus FDC-Routinen)
AVR

Beschreibung:

Mit dieser Interfacekarte soll der Moppel den Anschluß an modernere Massenspeicher erhalten. Wenn die komplette Software erstellt ist, kann der Moppel die Karten unter CP/M schreiben/lesen und auf der anderen Seite wird alles in DOS-Format abgelegt um den Datentransport zwischen Moppel und PC zu vereinfachen.

Das ist aber noch Zukunftsmusik, hier erst mal die detaillierte Beschreibung der Hardware.

Hardware:

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 2Weg BUS Ein-Ausgabe) betrieben. Durch die Handshake Signale /OBF, /ACK und /STB, IBF ist die Datenübergabe synchronisiert.

   Port B kann als einfacher 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.

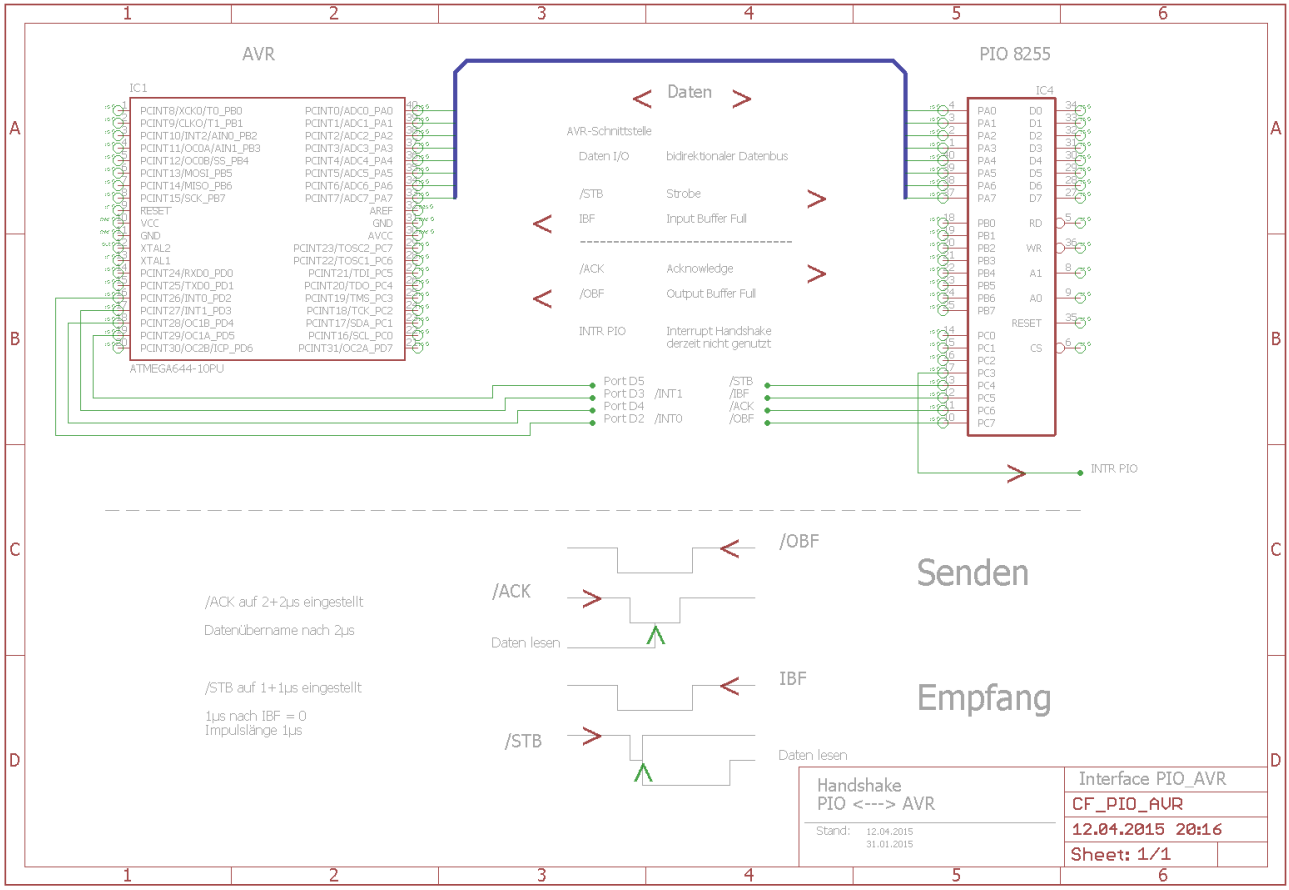
2. AVR-Teil

   Als „intelligenter 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-BootLoader ist die Programmierung in sekundenschnelle erledigt und dient in der Testphase als Monitor für die Befehlsabläufe. Später soll hiermit ein schneller Datenaustausch mit dem PC stattfinden …

   Anmerkung:

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

Handshake:



Anmerkung:

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.

```
;
;  Steuerwort: C1h
;
;  D7 D6 D5 D4 D3 D2 D1 D0
;   .  .  .  .  .  .  .  .
;   .  .  .  .  .  .  .  1  = C0-C2 als Eingang
;   .  .  .  .  .  .  .
;   .  .  .  .  .  .  0     = Kanal B als Ausgang
;   .  .  .  .  .  .
;   .  .  .  .  .  0        = Betriebart 0 für Gruppe B
;   .  .  .  .  .
;   .  .  x  x  x
;   .  .
;   1  1                    = Betriebsart 2 für Kanal A
;
;
; PIO-Kanal C lesen (EAh) Handshake
;
;  D7 D6 D5 D4 D3 D2 D1 D0
;   .  .  .  .  .  .  .  .
;   .  .  .  .  .  .
;   .  .  .  .  .  .
;   .  .  .  .  1 = INTRa   Interrupt (ISR muss Bit4-7 auswerten)
;   .  .  .  .  .
;   .  .  .  0 = /STBa      positive Flanke lädt Daten ins Eingangsregister
;   .  .  .  .
;   .  .  1 = IBFa          logisch 1 zeigt an, das Daten ins Eingangsregister
;   .  .                    geladen sind (AVR muss dies vor dem Senden auswerten
;   .  .                    wenn 0, darf mit /STB ein Byte gesendet werden)
;   .  .
;   .  .
;   .  0 = /ACKa            logisch 0 stellt die Daten ins Ausgangsregister
;   .                       sonst hochohmig
;   0 = /OBFa               logisch 0, Daten stehen im Ausgangsbuffer bereit
;                           und können vom AVR mit /ACK abgeholt 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 Programmteile um ein Byte zu lesen bzw. schreiben.

```
;
; 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                 ;
;
```

Nun müssen noch Regeln eingeführt werden, was der AVR mit den Daten machen soll, sprich ein Befehlssatz festlegen:

```
;
; AVR Befehle
;
AVRres  equ     00h             ; Spur 0 Sektor 0 ansteuern
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
AVRstat equ     0F0h            ; Status ermitteln
;
```

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 (AVR-DOS).

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

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

Hier die grundlegeneden Ein-Ausgaberoutinen

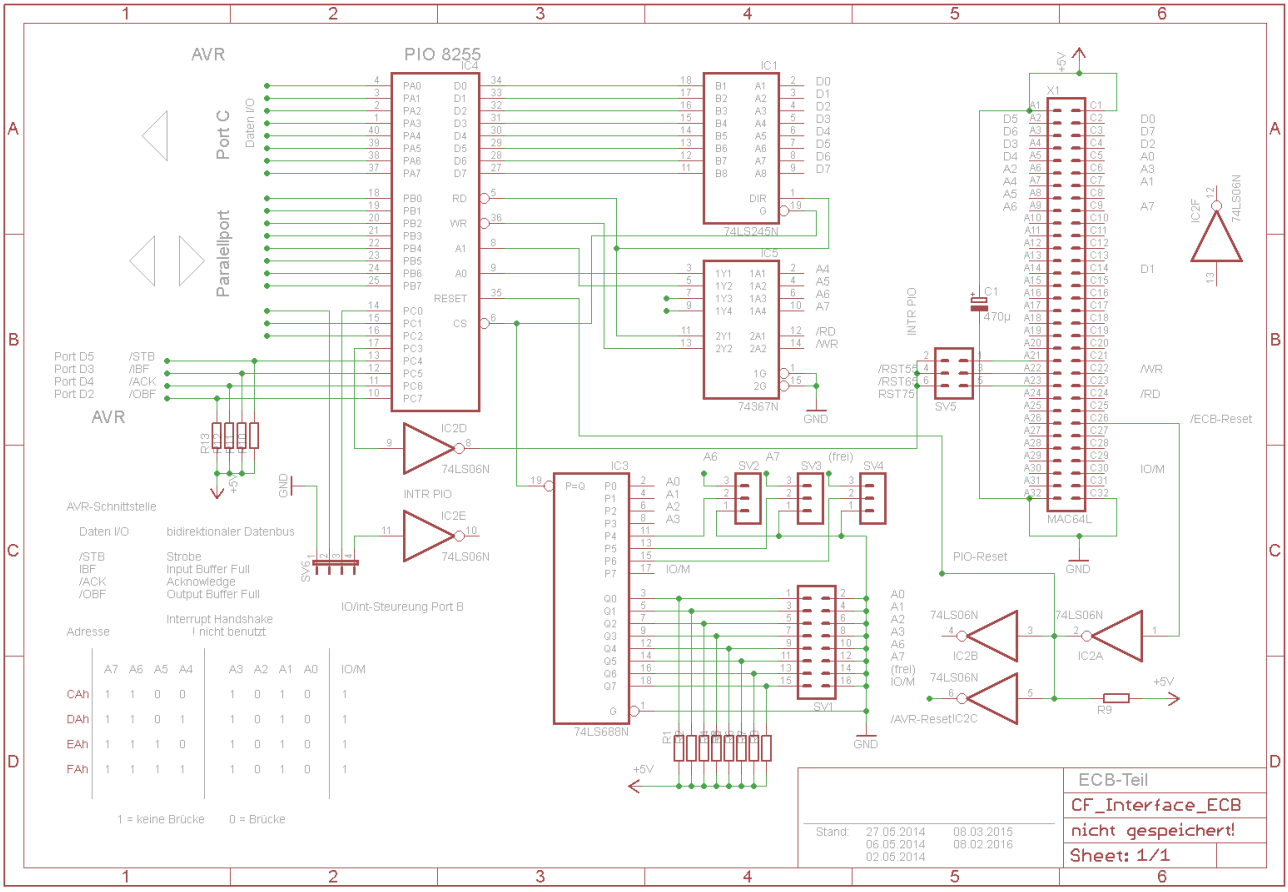
```
' Daten lesen (AVR-Sicht)
'
Sub Ecb_rdbyte
  Config Portc = Input
  Do                               ' warten auf Daten
  Loop Until Pind.2 = 0            ' /OBF = 0
  Reset Hs_ack                     '
  Waitus 2
  In_tmp = Pinc                    '  Daten lesen
  Waitus 2                      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

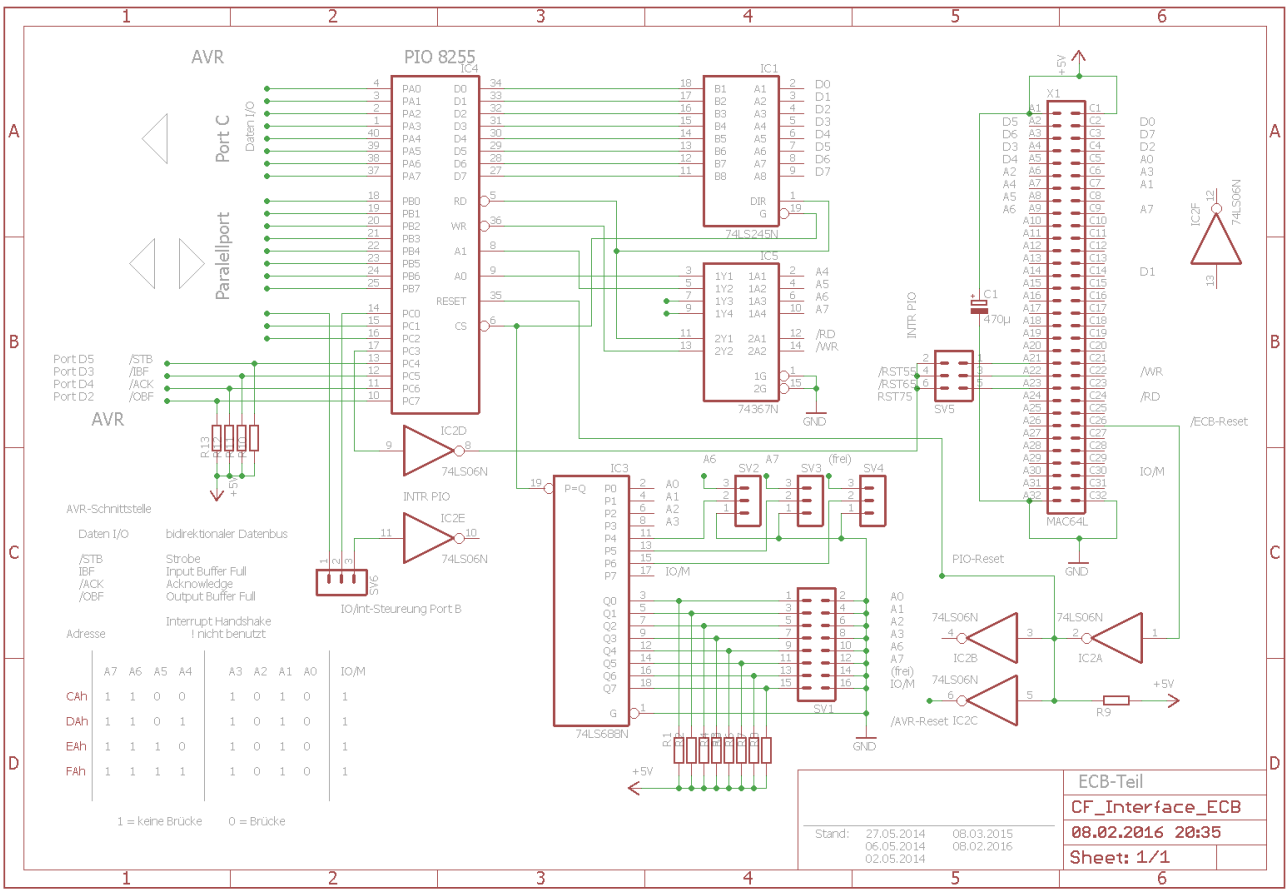
```
' Daten schreiben (AVR-Sicht)
'
Sub Ecb_wrbyte
  Config Portc = Output

  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
```

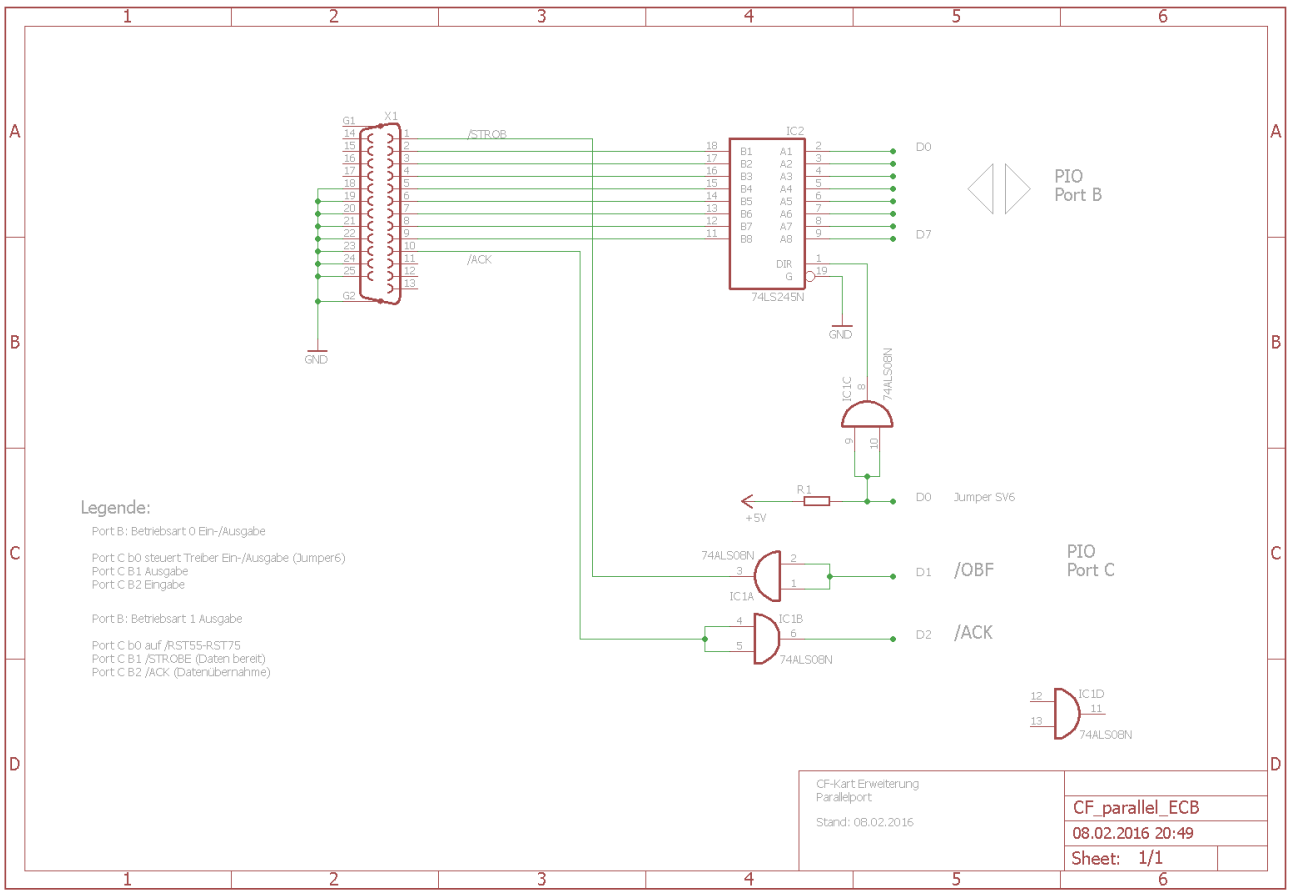
'Das Ganze noch mit dem „Befehlsinterpreter“ abrunden

```
  Do
    Call Ecb_rdbyte
    'If Testflg = 1 Then
     Print "Befehl: " ; In_tmp
    '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
      Case Wrsek
        Call Ecb_wrsek               ' Sektor schreiben
      Case Rdsek
        Call Ecb_rdsek               ' Sektor lesen
      Case Avrstat
        Call Avr_status             ' Status ausgeben
    End Select
  Loop
```

Damit ist das Grundgerüst für den Datenaustausch gelegt. Der komplette Quellcode könnt ihr den Anlagen entnehmen.


AVR
PIO 8255
Port C
Daten I/O
Parallelport
AVR
AVR-Schnittstelle
Daten I/O
bidirektionaler Datenbus
/STB
IBF
/ACK
/OBF
Strobe
Input Buffer Full
Acknowledge
Output Buffer Full
Interrupt Handshake
! nicht benutzt
IO/int-Steureung Port B
INTR PIO
PIO-Reset
/ECB-Reset
Adresse
A7 A6 A5 A4 A3 A2 A1 A0 IO/M
CAh 1 1 0 0 1 0 1 0 1
DAh 1 1 0 1 1 0 1 0 1
EAh 1 1 1 0 1 0 1 0 1
FAh 1 1 1 1 1 0 1 0 1
1 = keine Brücke 0 = Brücke
74LS245N
74367N
74LS688N
74LS06N
MAC64L
ECB-Teil
CF_Interface_ECB
nicht gespeichert!
Sheet: 1/1
Stand: 27.05.2014 08.03.2016
06.05.2014 08.02.2016
02.05.2014


AVR
PIO 8255
Port C
Daten I/O
Parallelport
AVR
AVR-Schnittstelle
Daten I/O
bidirektionaler Datenbus
/STB
IBF
/ACK
/OBF
Strobe
Input Buffer Full
Acknowledge
Output Buffer Full
Interrupt Handshake
! nicht benutzt
IO/int-Steureung Port B
INTR PIO
PIO-Reset
/ECB-Reset
Adresse
A7 A6 A5 A4 A3 A2 A1 A0 IO/M
CAh 1 1 0 0 1 0 1 0 1
DAh 1 1 0 1 1 0 1 0 1
EAh 1 1 1 0 1 0 1 0 1
FAh 1 1 1 1 1 0 1 0 1
1 = keine Brücke 0 = Brücke
74LS245N
74367N
74LS688N
74LS06N
MAC64L
ECB-Teil
CF_Interface_ECB
08.02.2016 20:35
Sheet: 1/1
Stand: 27.05.2014 08.03.2015
06.05.2014 08.02.2016
02.05.2014



Beschreibung Parallelinterface:

Port B als Aus-/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 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.