

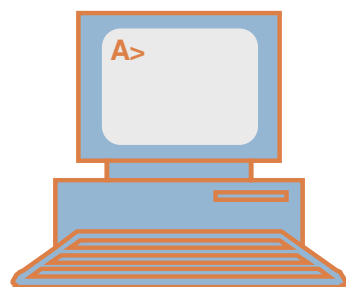
# AETERNUM

## AVR CP/M



Joe G.

Dokumentation



## ÄNDERUNGSVERZEICHNIS

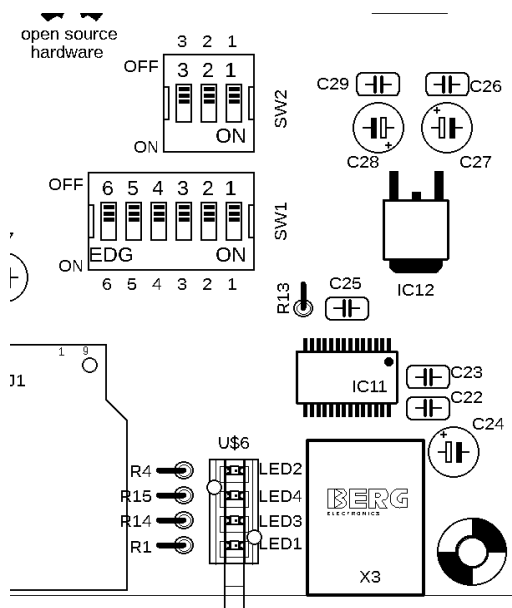
Änderung			geänderte Kapitel	Beschreibung	Autor	neuer Zustand
Nr.	Datum	Version				
1	13.11.2018	V01.00		Startversion	Joe G.	OK
2	27.02.2019	V01.01	Bestückung	Bestückung USB-Schnittstelle	Joe G.	OK
3	01.03.2019	V01.02	Bestückung	CP/M-System	Joe G.	OK

# Aeternum

## BESTÜCKUNG

### USB-Schnittstelle

Wir bestücken zuerst die Bauteile für die USB-Kommunikation auf der Oberseite der Platine. Die entsprechende Orientierung liefert Bild 1.



Wir beginnen mit den LEDS:

[ ]	LED1	grün	SMD 0603
[ ]	LED2	gelb	SMD 0603
[ ]	LED3	rot	SMD 0603
[ ]	LED4	grün	SMD 0603

Achtung, auf die richtige Polarität der LED's achten! Die Anode zeigt in Richtung der Widerstände.

Es folgt der USB-Seriell-Wandler und die USB-B Buchse:

[ ]	IC11	FT232RL	SSOP28
[ ]	X3	USB-B	

**Bild 1:** USB-Kommunikation

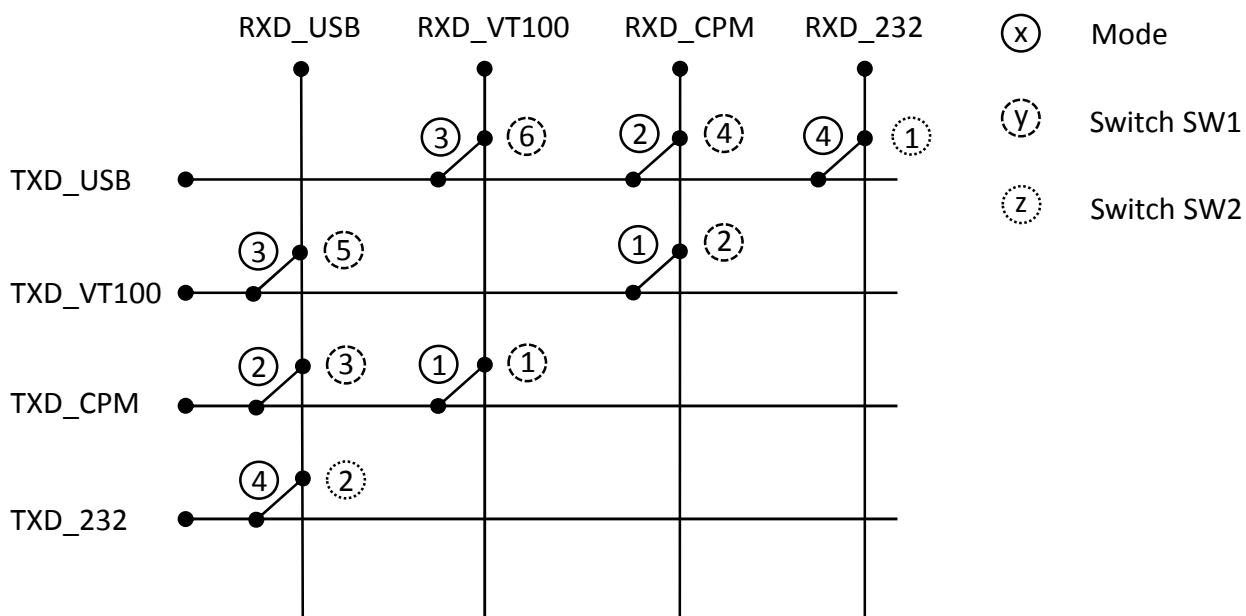
Nach der Bestückung des USB-Seriell-Wandlers und der USB-Buchse kann schon die Grundfunktion des Wandlers überprüft werden. Dazu ist die USB-B Buchse (X3) mit dem PC zu verbinden. Nun wird (im Windows Betriebssystem) automatisch der dazu notwendige Treiber (virtueller COM-Port) installiert. Anschließend ist bei korrekter Installation im Gerätemanager unter **Anschlüsse (COM & LPT)** der zugeordnete COM-Port aufgeführt. Ist das nicht der Fall, oder erscheint dort ein fehlerhafter COM-Port, sind die Lötverbindungen des FT232RL nochmals zu überprüfen. Bei einer fehlerfreien Installation des virtuellen COM-Ports werden die folgenden Bauelemente bestückt:

[ ]	R1	270 Ohm	THT
[ ]	R14	390 Ohm	THT
[ ]	R15	270 Ohm	THT

Die Widerstände sind so bemessen, dass bei einer Betriebsspannung von 3.3V unabhängig von der Flussspannung der LED immer ein Strom von 4.4mA fließt. Da die 3.3V noch nicht erzeugt werden, sind die LED's noch ohne Funktion. Anschließen folgen die Bauelemente:

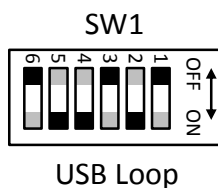
[ ]	C22	10n	THT
[ ]	C23	100n	THT
[ ]	C24	10u	THT
[ ]	C25	100n	THT
[ ]	R13	10 kOhm	THT
[ ]	IC12	LP2950CDT-3.3	DPACK
[ ]	C26	100n	THT
[ ]	C27	10u	THT
[ ]	C28	10u	THT
[ ]	C29	100n	THT

Wird jetzt die USB-Buchse (X3) mit dem PC verbunden, muss LED1 dauerhaft grün leuchten (Betriebsspannungskontrolle). LED3 (rot) und LED4 (grün) blinken beim Anstecken des USB-Kabels kurz auf. Damit ist ein erster Bestückungsschritt abgeschlossen und die korrekte Funktion der USB-Verbindung kann überprüft werden. Zu diesem Zwecke wurden alle seriellen Verbindungen über eine Schaltmatrix konfigurierbar gestaltet (Bild 2).



**Bild 2:** Schaltmatrix

Um die TXD\_USB Leitung mit der RXD\_USB Leitung zu verbinden (USB Loop) sind also die Schalter 2, 4 und 5 von SW1 auf ON zu stellen (Bild 3). Dazu wird SW1 [ ] bestückt.

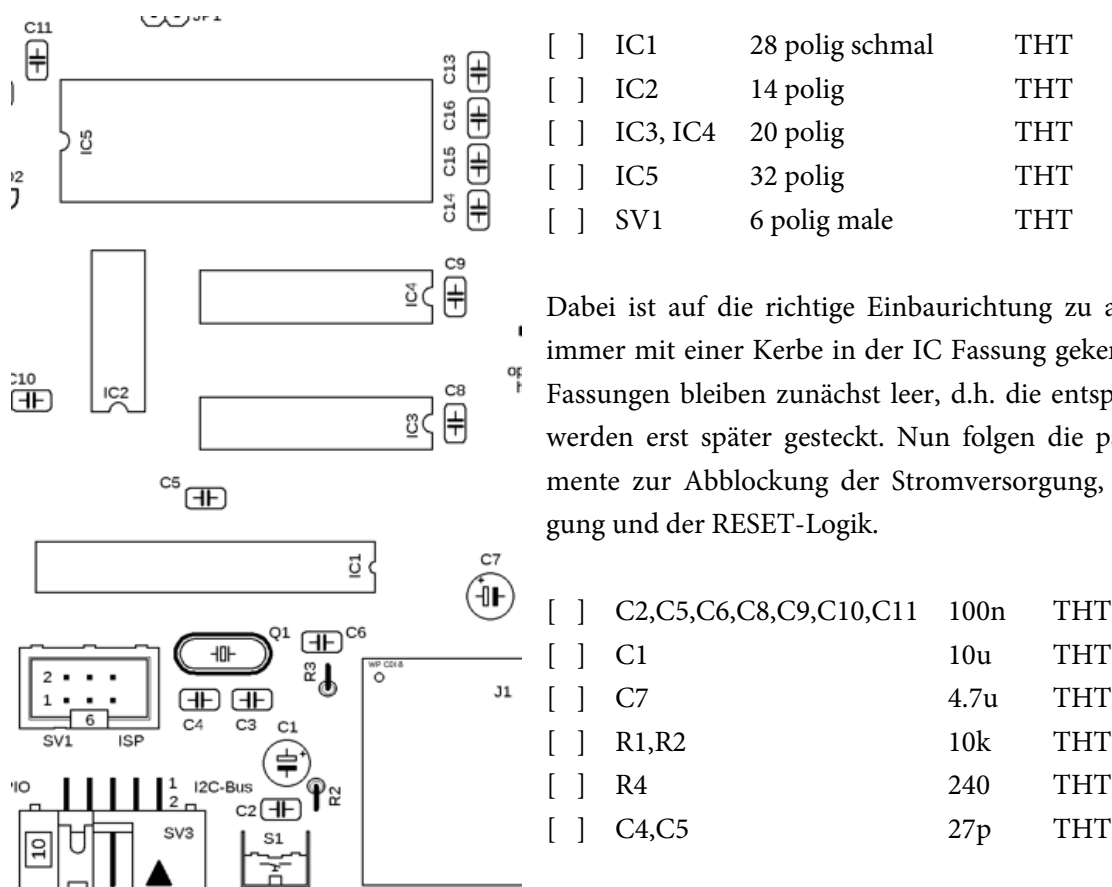


**Bild 3:** Schalterstellung für USB Loop

Nun kann mit einem Terminalprogram (z.B. HTerm) die Kommunikation mit der USB-Schnittstelle überprüft werden. Unabhängig von der eingestellten Baudrate müssen alle gesendeten Zeichen auch im Empfangsfenster wieder angezeigt werden (Echo). Wird vom PC ein Zeichen gesendet, so ist die TXD Leitung des FT232RL aktiv. Das wird durch LED3 (rot) signalisiert. Da durch die Schleife (TXD\_USB verbunden mit RXD\_USB) das Zeichen gleichzeitig an RXD anliegt, ist auch LED4 (grün) aktiv. Nun können alle Schalter von SW1 wieder in die Stellung OFF geschaltet werden. Der erste Schritt der Inbetriebnahme ist geschafft!

## CP/M-System

Im nächsten Schritt nehmen wir das eigentliche CP/M System in Betrieb. Da bis auf die SMD-Bauelemente alle IC's gesockelt sind, bestücken wir zunächst die zugehörigen IC-Fassungen (Bild 4).



Flash-Speicher des ATmega gebracht. Die dazu notwendige Datei **avrcpm.hex** befindet sich im Verzeichnis **/System/AVR**. Weiterhin ist eine korrekte Fuse Konfiguration des ATmega notwendig.

### Fuse Configuration

Hierbei ist das Fuse High Byte, das Fuse Low Byte und das Extended Fuse Byte richtig zu konfigurieren.

#### Fuse High Byte

Fuse High Byte	Bit	Set	Description	Default Value
RSTDISBL	7	1	External reset disable	1 (unprogrammed) RSTDISBL disabled
DWEN	6	1	debugWIRE Enable	0 (programmed) debugWIRE enabled
SPIEN	5	0	Enable Serial Program and Data Downloading	0 (programmed) SPI prog. Enabled
WDTON	4	1	Watchdog Timer always on	1 (unprogrammed)
EESAVE	3	1	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed) EEPROM not preserved
BOOTSZ1	2	0	Select Boot Size	0 (programmed)
BOOTSZ0	1	0	Select Boot Size	0 (programmed)
BOOTRST	0	1	Select Reset Vector	1 (unprogrammed)

Fuse **High Byte** erhält den Wert: **0xD9**

#### Fuse Low Byte

Fuse High Byte	Bit	Set	Description	Default Value
CKDIV8	7	1	Divide clock by 8	0 (programmed)
CKOUT	6	1	Clock output	1 (unprogrammed)
SUT1	5	1	Select start-up time	1 (unprogrammed)
SUT0	4	0	Select start-up time	0 (programmed)
CKSEL3	3	0	Select Clock source	0 (programmed)
CKSEL2	2	1	Select Clock source	0 (programmed)
CKSEL1	1	1	Select Clock source	1 (unprogrammed)
CKSEL0	0	1	Select Clock source	0 (programmed)

Fuse **Low Byte** erhält den Wert: **0xE7**

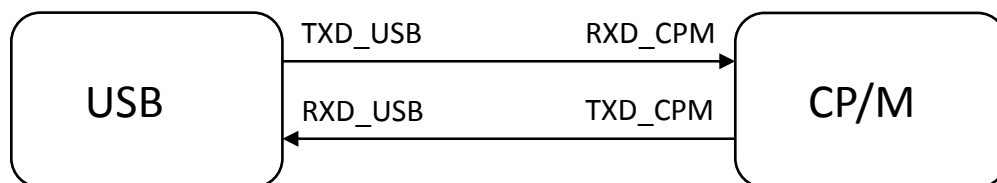
#### Extended Fuse

Fuse High Byte	Bit	Set	Description	Default Value
BODLEVEL2	2	1	Brown-out Detector trigger level	1 (unprogrammed)
BODLEVEL1	1	1	Brown-out Detector trigger level	1 (unprogrammed)
BODLEVEL0	0	1	Brown-out Detector trigger level	1 (unprogrammed)

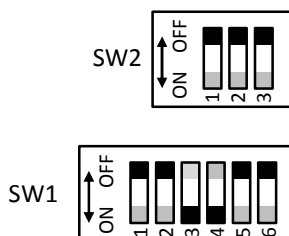
**Extended Fuse** Byte erhält den Wert: **0xFF**

Um zu kontrollieren, ob der Prozessor korrekt arbeitet, leiten wir noch die serielle Schnittstelle des ATmega 328p (TD\_CPM und RXD\_CPM) auf die USB-Schnittstelle um (Mode 2). Somit können alle Ausgaben über ein Terminalprogramm verfolgt werden. Dieses ist natürlich entsprechend zu konfigurieren:

Port: virtueller COM-Port des zuvor installierten USB-Treibers  
 Baud Rate: 115200  
 Data: 8 Bit  
 Parity: none  
 Stop: 1 Bit  
 Flow control: none



### Mode 2



### Schalterstellungen für Mode 2

Arbeitet die Software korrekt, erscheint im Terminal nach einem kurzen Resetimpuls von S1 (CP/M Reset) die folgende Ausschrift:

```

CPM on an AVR, v3.5 r241M
Testing RAM: fill...wait...reread...
Addr xx yy
0100 01 00 -----X -----
0101 00 01 -----X -----
0102 03 02 -----X -----
...
...
...
010C 0D 0C -----X -----
010D 0C 0D -----X -----
010E 0F 0E -----X -----
010F 0E 0F -----X System halted!
  
```

Das System überprüft im ersten Schritt die korrekte Funktionsweise des angeschlossenen DRAM. Dazu wird er mit einem Bitmuster beschrieben und wieder ausgelesen. Da im derzeitigen Zustand noch kein DRAM angeschlossen ist, schlägt dieser Test natürlich fehl. Wie wir an der Terminalausgabe jedoch erkennen kön-

nen, arbeitet der ATmega korrekt. Somit kann der fehlende DRAM [ ] IC5 bestückt werden. Weiterhin werden noch die Adress- und Datenmultiplexer [ ] IC3 und IC4, sowie die Auswahllogik [ ] IC2 benötigt. Auch hier ist besonders darauf zu achten, dass alles IC's in der korrekten Richtung in die Fassungen gesteckt werden. Selbstverständlich ist vor der Bestückung die Spannungsversorgung zu trennen. Wird nun erneut der Reset-Taster S1 betätigt, muss der RAM-Test erfolgreich absolviert werden:

---

```
CPM on an AVR, v3.5 r241M
Testing RAM: fill...wait...reread...
Initing mmc...
No bootable CP/M disk found! Please change
MMC/SD-Card.
```

---

Wie die Terminalausgabe zeigt, wird zunächst der DRAM geschrieben und nach einer kurzen Wartepause wieder gelesen. Stimmen Ein- und Ausgabe im RAM-Test überein, folgt die Initialisierung der SD-Card und der eigentliche Bootvorgang des CP/M-Systems. Da wir noch keine SD-Card installiert haben, muss dieser Test natürlich fehlschlagen. Der SD-Card Halter [ ] wird an der Position J1 (Bild 4) bestückt. Anschließend muß ein bootfähiges CP/M Image **CPMDSK\_A.IMG** auf eine SD-Card kopiert werden. Vorgefertigte CP/M Image befinden sich im Verzeichnis **/System/CPM**. Anschließend wird wiederum ein der RESET-Taster S1 betätigt.

---

```
CPM on an AVR, v3.5 r241M
Testing RAM: fill...wait...reread...
Initing mmc...

A: FAT16 File-Image 'A' at: 8451, size: 256KB.
Partinit done.
Ok, Z80-CPU is live!

ipl
62k cp/m vers 2.2

A>
```

---

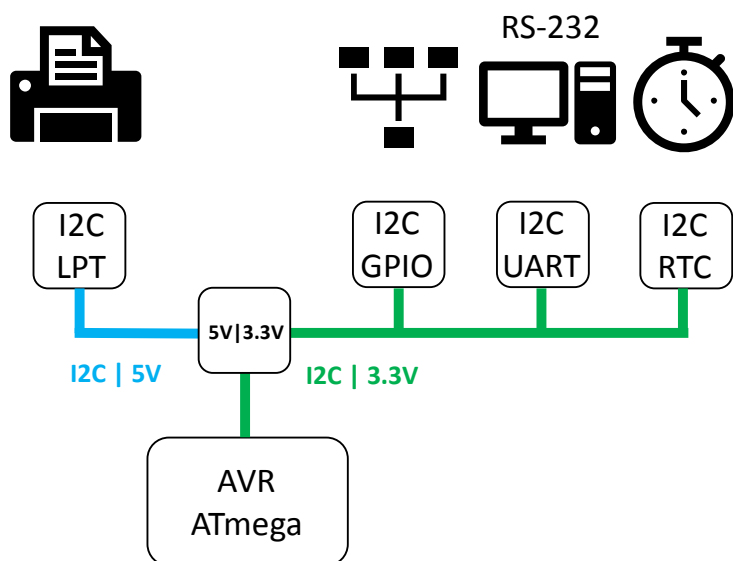
Mit dieser Terminalausgabe ist ein weiter Inbetriebnahmeschritt erfolgreich absolviert! Das CP/M System bootet korrekt von der SD-Card und kann für die folgenden Erweiterungen genutzt werden.

## I2C-Baugruppen

I<sup>2</sup>C, für englisch Inter-Integrated Circuit, ist ein 1982 von Philips Semiconductors entwickelter serieller Datenbus. Er wird im CP/M-System geräteintern für die Kommunikation zwischen den unterschiedlichen Baugruppen genutzt (Bild 5), also zwischen dem Prozessor (ATmega) und Peripherie-ICs. Atmel führte aus li-



zensrechtlichen Gründen die heute auch von einigen anderen Herstellern verwendete Bezeichnung TWI (Two-Wire-Interface) ein, technisch sind TWI und I<sup>2</sup>C aber identisch.



**Bild 5:** I2C Baugruppen des CP/M-Systems

Wie Bild 5 zeigt, arbeitet das Bus-System mit zwei unterschiedlichen Spannungen (3.3V und 5V). Die Pegelanpassung zwischen diesen beiden unterschiedlichen Busspannungen wird durch einen Pegelwandler (IC10) realisiert. Zusätzlich benötigt der I2C-Bus für jeden Spannungsstrang eine Terminierung der Busleitungen (R9 – R12). Für die Inbetriebnahme des I2C-Busses bestücken wir also den erforderlichen Pegelwandler und die Bus Terminierungen.

[ ] IC10	PCA9517	SO-08
[ ] R9,R10,R11,R12	10k	THT

Als erstes Bus-Gerät kann nun die Real-Time-Clock (RTC) in Betrieb genommen werden. Dazu müssen wir die zugehörigen Bauelemente sowie die 3V Stützbatteriefassung auf der Leiterplatte bestücken.

[ ] IC6	PCF8583P	DIL8
[ ] Q2	32.768 kHz	TC26H
[ ] C12	6-20pf	Trimmer
[ ] D1,D2	BAT43	THT
[ ] G1	CR2032VKZH	THT

Wie schon bei den vorherigen DIL-Bauelementen ist beim RTC auf die korrekte Steckrichtung in der IC-Fassung zu achten. Die variable Kapazität (Trimmer C12) stellen wir auf eine Mittelstellung. In die Fassung der Stützbatterie wird abschließend eine 3V Zelle (CR2032) gesteckt. Auch hier muss unbedingt auf die richtige Polarität geachtet werden. Dazu ist an den Stirnseiten der Fassung jeweils eine Polaritätskennzeichnung (+) / (-) angebracht. Nach dieser Bestückung booten wir das System wie gewohnt über den RESET-Taster S1. Zur Identifikation vorhandener I2C-Busgeräte dient das Programm **I2CSCAN.COM** welches sich schon auf

dem bootfähigen CP/M Image **CPMDSK\_A.IMG** befindet. Dieses wird einfach im Terminal gestartet und scannt den internen I2C-Bus nach vorhandenen I2C-Geräten.

---

```
A>i2cscan
Scanning I2C bus for devices...
*** Device found at slave address 50h = 80d (8-bit: A0h = 160d)
Done.

A>
```

---

Die Terminalausgabe zeigt uns an, dass ein I2C-Gerät unter der Slave-Adresse hex 0x50 gefunden wurde. Diese Adresse ist hardwareseitig dem RTC-IC PCF8583P zugeordnet. Jetzt muss nur noch die Echtzeituhr in das CP/M-System integriert werden. Das erfolgt mit dem Programm **LDTIM.COM**, welches sich ebenfalls schon im Bootimage befindet.

---

```
ZSDOS Time Stamp Loader, Ver 1.1
  Copyright (C) 1988 by H.F.Bower /
  C.W.Cotrill

...loaded at D8F8H
Clock is : AVRCPM Clock          0.2

A>
```

---

Damit ist die Uhr einsatzbereit. Im Initialisierungszustand ist die Uhrzeit und das Datum natürlich noch nicht korrekt gestellt. Mit dem Programm TD.COM kann Uhrzeit und Datum verändert und dann der RTC übergeben werden.

---

```
A>td
Mar 01, 2019 09:56:24
A>
```

---

Damit ist das erste I2C Gerät für unser CP/M-System erfolgreich installiert.

Für E/A Experimente sind am 3.3V-Bus 8 GPIO-Pins auf den Steckverbinder SV2 herausgeführt. Diese können extern, unter Beachtung der Anschlussbedingungen, für eigene Erweiterungen verwendet werden. Ein zweiter Steckverbinder (SV3) beinhaltet zusätzlich die Versorgungsspannungen 3.3V und 5V sowie den I2C-Bus mit den jeweiligen Logikpegeln. Dazu sind die Steckverbinder SV2 und SV3 [ ] zu bestücken sowie IC9 in die zugehörige IC-Fassung zu stecken. Zu beachten sind zwei unterschiedliche Ausführungen des 8-Bit

I/O Busexpanders. Während der PCF8574 unter der Slave-Adresse hex 0x20 angesprochen wird, wird der PCF8574A unter der Adresse hex 0x38 adressiert. Da das IC gesockelt ist, kann wahlweise die passende Variante verwendet werden. Nach einem erneuten I2C-Bus-Scan muss der Busexpander in der Liste der Slave Devices auftauchen.

---

```
A>i2cscan
Scanning I2C bus for devices...
*** Device found at slave address 38h = 56d (8-bit: 70h = 112d)
*** Device found at slave address 50h = 80d (8-bit: A0h = 160d)
Done.

A>
```

---

Als letztes internes 3.3V I2C-Bus-Gerät wird ein Wandlerbaustein verwendet, der den seriellen I2C-Bus in einen seriellen RS232-Bus konvertiert. Dazu sind die folgenden SMD-Bauelemente zu bestücken.

[ ] IC8	SC16IS740	TSSOP16
[ ] Q3	14.7456 MHz	SMD-5x3
[ ] C17,C18	18pf	0603
[ ] R6,R8	1k	0805

Wie schon den Bauteilbezeichnungen entnommen werden kann, handelt es sich bei diesen Bauelementen ausschließlich um SMD Bauelemente. Mit etwas Übung können jedoch auch diese Bauelemente gut per Hand bestückt werden. Nach korrekter Bestückung meldet sich der Schnittstellenwandler unter der Slave-Adresse hex 0x48.

---

```
A>i2cscan
Scanning I2C bus for devices...
*** Device found at slave address 38h = 56d (8-bit: 70h = 112d)
*** Device found at slave address 48h = 72d (8-bit: 90h = 144d)
*** Device found at slave address 50h = 80d (8-bit: A0h = 160d)
Done.

A>
```

---

Mit diesem Schritt ist die Bestückung des I2C-Busses auf der 3.3V Seite abgeschlossen. Wie Bild 5 zeigt, ist die Druckerschnittstelle auf der 5V Seite des I2C-Busses angeschlossen. Weiterhin benötigt ein Drucker zur korrekten Ansteuerung im Parallelbetrieb mehr als 8-Bit. Aus diesem Grunde wird ein 16-Bit I/O Expander (U1) eingesetzt. Dieser muss nur korrekt in die entsprechende IC-Fassung gesteckt werden, um seine Funktion aufzunehmen. Ein erneuter Scan des I2C-Busses bestätigt die Funktion der Centronics-Schnittstelle unter der Adresse hex 0x27. Abschließend bestücken wir den Pegelwandler IC7, die Kapazitäten C13-C16, den Widerstand R5 sowie die Steckverbinder X1 und X2.

---

```
A>i2cscan
Scanning I2C bus for devices...
*** Device found at slave address 27h = 39d (8-bit: 4Eh = 78d)
*** Device found at slave address 38h = 56d (8-bit: 70h = 112d)
*** Device found at slave address 48h = 72d (8-bit: 90h = 144d)
*** Device found at slave address 50h = 80d (8-bit: A0h = 160d)
Done.

A>
```

---

Um die Funktionsweise der seriellen Schnittstelle überprüfen zu können, benötigen wir ein Nullmodemkabel und ein zweites Terminal. Dieses wird mit der RS232-Schnittstelle unter den folgenden Parametern verbunden:

Port:	COM-Port des zweiten Terminals
Baud Rate:	9600
Data:	8 Bit
Parity:	none
Stop:	1 Bit
Flow control:	none

Über die *logischen* und *physikalischen* E/A-Einheiten des CP/M-Systems kann die Konsole auf die RS232-Schnittstelle umgelenkt werden. Nähere Erläuterungen dazu finden sich unter dem Abschnitt Ein-/Ausgab-Einheiten. Für einen vorläufigen Test der gerade bestückten Hardware wird die Konsole wie folgt umgelenkt:

---

```
A>STAT CON:=CRT:
```

---

Terminal 1 ist nun wirkungslos und das CP/M-System kann nur noch über Terminal 2 bedient werden. Das wiederum zeigt die korrekte Funktion der RS232-Schnittstelle an. Selbstverständlich kann von Terminal 2 wieder zurück auf Terminal 1 geschaltet werden.

---

```
A>STAT CON:=TTY:
```

---

Somit ist der ursprüngliche Zustand hergestellt. Damit ist der Aufbau des CP/M-Systems erfolgreich abgeschlossen und die letzte Baugruppe, das VT100-Terminal kann in Angriff genommen werden.

## VT100-Terminal

Das VT100 Video Terminal ist ein Computerterminal, welches 1978 von der Digital Equipment Corporation (DEC) aus dem VT52 entwickelt wurde. Es hat sich im Laufe der Jahre zum de facto Standard für Terminals entwickelt. Noch heute unterstützen zahlreiche Terminal-Emulatoren diesen Standard. Erst im Jahre 1983 wurde das VT100 von einem verbesserten Standard wie dem VT220 ersetzt.

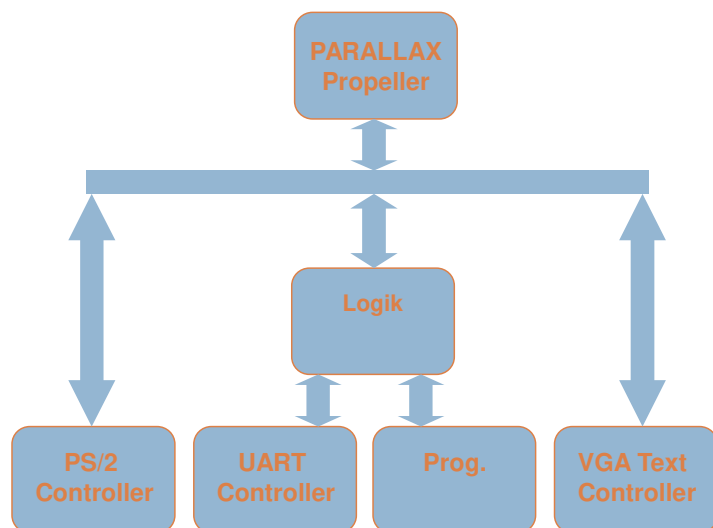
Die Hardware des ersten VT100 Terminals basierte auf einem Industrie-Standard-Mikroprozessor, dem Intel 8080. Optional gab es zusätzliche Anschlüsse für einen Drucker oder für eine Grafikerweiterung. Auf seinem 12 Zoll Monochrom-Bildschirm konnte das Terminal 24 Zeilen zu 80 Zeichen oder alternativ 14 Zeilen zu 132 Zeichen darstellen. Die Ausgabe über eine Semigrafik wurde durch Sonderzeichen und ANSI-Escape-Sequenzen ermöglicht. Das Terminal verfügte über eine erweiterte Schreibmaschinen-Tastatur (83 Tasten, QWERTY-Belegung) mit einigen Funktionstasten. Weiterhin existierten Folgeprodukte mit unterschiedlicher Hardwareausstattung. Erwähnenswert ist hier z. B. das VT180 (Codename "Robin"). Es beinhaltete zusätzlich noch einen Zilog Z80 Mikroprozessor und konnte direkt CP/M ausführen.

### Hardware-Plattform

Das hier bestückte VT100-Terminal baut auf einem Parallax Propeller (P8X32A) auf und emuliert die ursprünglichen VT100 Terminalcodes. Der verwendete Mikrocontroller beinhaltet 8 unabhängig arbeitende 32-Bit-RISC CPU Kerne. Sie übernehmen dabei die Funktionen:

- Videodarstellung (VGA-Text-Mode)
- PS/2-Tastaturcontroller
- UART
- GPIO

Wie aus Bild 5 ersichtlich, kann der Controller über eine Logik immer nur auf einen externen seriellen Kanal zugreifen. Dabei dient die Programmierschnittstelle eigentlich nur der Programmierung und der Inbetriebnahme. Im tatsächlichen Terminalbetrieb wird der serielle Kanal des Terminals auf das CP/M-System geschaltet.



**Bild 5:** Baugruppen des VT100-Terminals



## EIN / AUSGABE - EINHEITEN

CP/M verfügt über vier *logische* E/A-Einheiten. Eine *logische* E/A-Einheit entspricht dabei einem symbolischen Namen für eine Gruppe von E/A-Einheiten, die alle von CP/M bedient werden können. Diese Gruppen sind:

**CON:** für „Console“ oder ein Terminal das Bildschirm und Tastatur besitzt (Eingabe/Ausgabe)

**RDR:** Eingabe vom Lochstreifenleser

**PUN:** Ausgabe auf Lochstreifenstanzer

**LST:** für „List“ Geräte wie z.B. Drucker

Über das I/O-Byte, kann jedem der *logischen* E/A-Einheiten eine tatsächliche *physikalische* vorhandene E/A-Einheit zugewiesen werden. Das I/O-Byte befindet sich im Arbeitsspeicher auf Adresse 0003h und ist folgendermaßen aufgebaut:

**Tab. 1:** Aufbau des I/O-Byte

LST		PUN		RDR		CON	
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	x	x	x	x	x

Jedes Feld (zwei Bit) kann somit einen Wert zwischen 0 und 3 einnehmen und bestimmt dadurch die Zuordnung zu einer bestimmten, tatsächlich existierenden, *physikalischen* E/A-Einheit. CP/M erlaubt die folgenden *physikalischen* Einheiten:

**TTY:** Teletype device (slow speed console)

**CRT:** Cathode ray tube device (high speed console)

**BAT:** Batch processing (console is current RDR; output goes to current LST: device)

**UC1:** User-defined console

**PTR:** Paper tape reader (high speed reader)

**UR1:** User-defined reader #1

**UR2:** User-defined reader #2

**PTP:** Paper tape punch (high speed punch)

**UP1:** User-defined punch #1

**UP2:** User-defined punch #2

**LPT:** Line printer

**UL1:** User-defined list device #1

Welche E/A-Einheiten tatsächlich implementiert sind, hängt von der jeweiligen Hardware des eigenen Computers ab. Im aktuellen System existiert die folgende Zuordnung zu den *physikalischen* E/A-Einheiten:

**TTY:** AVR-UART    Console    (input and output)

**CRT:** I2C-UART    Console    (input and output)

<b>BAT:</b>	AVR-UART	Console	(input and output)
<b>UC1:</b>	AVR-UART	Console	(input and output)
<b>TTY:</b>	AVR-UART	Reader	(input only)
<b>PTR:</b>	I2C-UART	Reader	(input only)
<b>UR1:</b>	AVR-UART	Reader	(input only)
<b>UR2:</b>	AVR-UART	Reader	(input only)
<b>TTY:</b>	AVR-UART	Punch	(output only)
<b>PTP:</b>	I2C-UART	Punch	(output only)
<b>UP1:</b>	I2C-GPIO	Punch	(output only)
<b>UP2:</b>	AVR-UART	Punch	(output only)
<b>TTY:</b>	AVR-UART	Printer	(output only)
<b>CRT:</b>	I2C-UART	Printer	(output only)
<b>LPT:</b>	I2C-GPIO	Printer	(output only)
<b>UL1:</b>	AVR-UART	Printer	(output only)

Über den **STAT-Befehl** (status) kann der Zustand des eigenen Systems abgefragt und auch verändert werden. Die Abfrage zeigt, dass die Console (Bildschirmausgabe und Tastatureingabe) über die AVR-UART-

---

```
A>stat dev:
CON: is TTY:
RDR: is PTR:
PUN: is PTP:
LST: is LPT:
```

---

Schnittstelle (**TTY**) bedient wird. **RDR** und **PUN** werden nicht bedient, da **PTR** keinem *physikalischen* Gerät zugeordnet ist. Weiterhin können mit dem **STAT-Befehl** alle möglichen Zuordnungen des I/O-Bytes abgefragt werden.



---

```
A>stat val:

Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS
$DIR
Disk Status   : DSK: d:DSK:
User Status   : USR:
Iobyte Assign:
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
```

---

So kann der Console z.B. die tatsächlich vorhandene **CRT**-Einheit (I2C-UART) zugeordnet werden. Alle Tastatureingaben und Bildschirmausgaben erfolgen ab dem Zeitpunkt der Zuordnung über die I2C-UART Schnittstelle.

---

```
A>STAT CON:=CRT:
```

---

Ist die **CRT**-Einheit nicht korrekt angeschlossen oder arbeitet fehlerhaft, so kann das CP/M-System nicht mehr über die Tastatur bedient werden. Bei korrekter Arbeitsweise lässt sich jedoch die nun geänderte Gerätezuordnung überprüfen.

---

```
A>STAT DEV:
CON: is CRT:
RDR: is PTR:
PUN: is PTP:
LST: is LPT:
```

---

Ein entsprechend umgekehrter Befehl schaltet die Console wieder auf die ursprüngliche E/A-Einheit zurück (AVR-UART).

---

```
A>STAT CON:=TTY:
```

---

CP/M bietet auch die Möglichkeit die *physikalischen* Geräte direkt anzusprechen. Dazu werden die Gerätenamen direkt angesprochen.

---

```
A>PIP TTY:=readme.txt
```

---

Der Inhalt der Datei „readme.txt“ wird unabhängig von der jeweiligen Zuordnung des I/O-Byte direkt auf **TTY** (AVR-UART) gesendet. Der Befehl

---

```
A>PIP LPT:=readme.txt
```

---

sendet die Datei „readme.txt“ direkt, unter Umgehung des I/O-Byte, an den Drucker (**LPT**).

## QUELLENANGABEN

- [1] [http://www.mikrocontroller.net/articles/AVR\\_Bootloader\\_FastBoot\\_von\\_Peter\\_Dannegger](http://www.mikrocontroller.net/articles/AVR_Bootloader_FastBoot_von_Peter_Dannegger)
- [2] [http://www.mikrocontroller.net/articles/AVR\\_Bootloader\\_FastBoot\\_von\\_Peter\\_Dannegger/Tutorial\\_ATtiny13](http://www.mikrocontroller.net/articles/AVR_Bootloader_FastBoot_von_Peter_Dannegger/Tutorial_ATtiny13)
- [3] <http://www.leo-andres.de/2012/09/updateloader-benutzeroberfläche-für-avr-bootloader/>
- [4] <http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Components/General/SDSpec.pdf>
- [5] <http://elm-chan.org/fsw/ff/en/filename.html>
- [6] <https://github.com/Boregard/FBoot-Linux>
- [7] [http://www.nxp.com/documents/application\\_note/AN10911.pdf](http://www.nxp.com/documents/application_note/AN10911.pdf)
- [8] [https://www.sdcard.org/downloads/pls/simplified\\_specs/](https://www.sdcard.org/downloads/pls/simplified_specs/)

## ANHANG A

### Steckverbinder (noch offen !!!)

#### CON0 – RS-232

Data Terminal Equipment (DTE)

Name	Funktion	Pin (DB-9) male	Pin (DB-10) male	DTE	connect
<b>DCD</b>	(Data) Carrier Detect	1	1	IN	x
<b>RxD</b>	Receive Data	2	3	IN	x
<b>TxD</b>	Transmit Data	3	5	OUT	x
<b>DTR</b>	Data Terminal Ready	4	7	OUT	x
<b>GND</b>	Ground	5	9	-	x
<b>DSR</b>	Data Set Ready	6	2	IN	x
<b>RTS</b>	Request to Send	7	4	OUT	x
<b>CTS</b>	Clear to Send	8	6	IN	x
<b>RI</b>	Ring Indicator	9	8	IN	-
	Common Ground	NC	10	-	x

#### CON1 – Terminal

Data Terminal Equipment (DTE)

Name	Funktion	Pin (DB-9) male	Pin (DB-10) male	DTE	connect
<b>DCD</b>	(Data) Carrier Detect	1	1	IN	-
<b>RxD</b>	Receive Data	2	3	IN	x
<b>TxD</b>	Transmit Data	3	5	OUT	x
<b>DTR</b>	Data Terminal Ready	4	7	OUT	x
<b>GND</b>	Ground	5	9	-	x
<b>DSR</b>	Data Set Ready	6	2	IN	x
<b>RTS</b>	Request to Send	7	4	OUT	-
<b>CTS</b>	Clear to Send	8	6	IN	-
<b>RI</b>	Ring Indicator	9	8	IN	-
	Common Ground	NC	10	-	x

#### CON2 – RS-232 (USB)

Data Terminal Equipment (DTE)

Name	Funktion	Pin (DB-6)	DTE	connect
------	----------	------------	-----	---------

		female		
<b>DTR</b>	Data Terminal Ready	1	OUT	-
<b>RxD</b>	Receive Data	2	IN	x
<b>TxD</b>	Transmit Data	3	OUT	x
<b>VCC</b>	Voltage	4	-	x
<b>CTS</b>	Clear to Send	5	IN	x
<b>GND</b>	Ground	5	-	x

### CON3 – Terminal (USB)

Data Terminal Equipment (DTE)

Name	Funktion	Pin (DB-6) female	DTE	connect
<b>DTR</b>	Data Terminal Ready	1	OUT	-
<b>RxD</b>	Receive Data	2	IN	x
<b>TxD</b>	Transmit Data	3	OUT	x
<b>VCC</b>	Voltage	4	-	x
<b>CTS</b>	Clear to Send	5	IN	-
<b>GND</b>	Ground	5	-	x