

RETROCOMPUTING

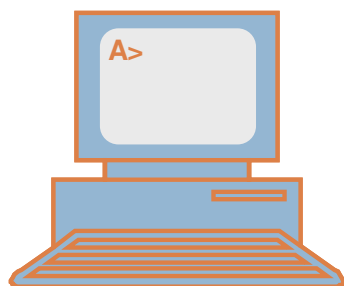
Z180 - STAMP

AVR - STAMP



Joe G.

Dokumentation



ÄNDERUNGSVERZEICHNIS

Änderung			geänderte	Beschreibung	Autor	neuer
Nr.	Datum	Version	Kapitel			Zustand
1	12.09.2014	V01.00		Startversion	Joe G.	OK
2	26.09.2014	V01.01	Z180	Versorgungsspannungen am Z180-Stamp Modul, (Pin2, Pin3)	Joe G.	OK
3	28.09.2014	V01.02	Z180	Versorgungsspannungen am Z180-Stamp Modul, (Pin2, Pin3)	Joe G.	OK
4	22.12.2014	V01.03	Inbetriebnahme	Monitorbeschreibung	Joe G.	OK
5	23.03.2015	V01.03	Bootloader	Extended Fuse	Joe G.	OK
6	25.04.2015	V01.04	RAM Monitor	RAM Test Monitor	Joe G.	OK
7	24.05.2015	V01.05	Anhang B	Steckverbinder	Joe G.	i.A.
8	30.05.2015	V01.06	Anhang A	Busverbinder	Joe G.	OK
9	05.06.2015	V01.07	Monitor	Scriptarbeitung	Joe G.	OK
10	28.09.2015	V01.08	Stromversorgung	Stromversorgung Hardware V1.1	Joe G.	OK
11	10.05.2016	V01.09	Anhang A	Busverbinder	Joe G.	OK
12	18.05.2016	V01.10	Monitor	Enviromentvariablen	Joe G.	OK
12	31.03.2018	V01.11	Monitor	Enviromentvariablen	Joe G.	OK

Retrocomputing

STROMVERSORGUNG

Sowohl das AVR-Stamp als auch das Z180-Stamp Modul besitzen bis auf wenige Ausnahmen ein identisch belegtes Anschlussystem. Über dieses Anschlussystem werden auch alle benötigten Versorgungsspannungen geführt. Wird eine ausschließlich externe Spannungsversorgung gewählt, reicht die Beschaltung der entsprechenden Versorgungspins. Zu Debug- oder Entwicklungszwecken können jedoch auch beide Module über alternative Beschaltungen versorgt werden. Da im Laufe der Zeit unterschiedliche Hardwareversionen entstanden sind, ist für das weitere Verständnis die jeweilige Hardwareversion zu betrachten.

Hardwareversion 1.0

AVR-Stamp V1.0

Das AVR-Stamp Modul benötigt ausschließlich 3.3V. Die auf dem Modul zusätzlich vorhandene 5V Leitung wird nur über das Stecksystem durchgeschleift. Weiterhin kann diese 5V Leitung über die auf dem AVR-Modul vorhandene USB-Buchse mit 5V versorgt werden. Wird JP1 geschlossen, liegt die 5V USB-Spannung gleichzeitig auf den Anschluss B2 (*Achtung! Das Stamp-Modul wird damit noch nicht mit 3.3V versorgt*). Weiterhin sollte die maximale Belastbarkeit der USB-Schnittstelle beachtet werden.

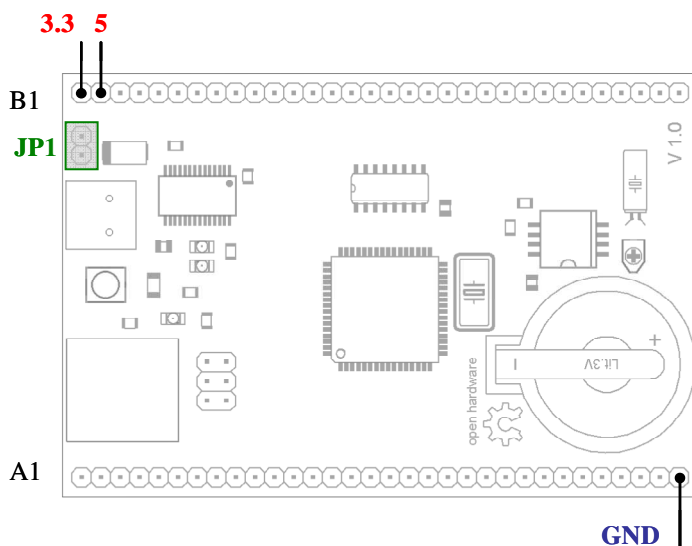


Abb.1: Versorgungsspannungen am AVR-Stamp Modul

Z180-Stamp V1.0

Das Z180-Stamp Modul besitzt neben den schon erwähnten Versorgungspins der Anschlussleisten A und B noch zusätzliche Versorgungsvarianten.

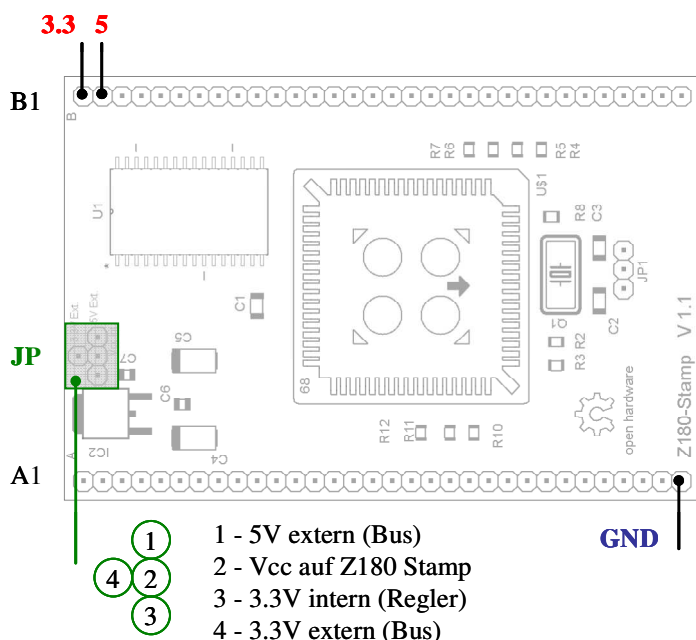


Abb.2: Versorgungsspannungen am Z180-Stamp Modul

Damit sind die folgenden Versorgungsvarianten möglich:

A – 3.3V Spannungsversorgung von Außen

Die Versorgungsspannung von 3.3V wird über das äußere Stecksystem an Pin B1 angelegt. Damit der Z180 versorgt wird, sind die Kontakte 2 und 4 zu schließen. Der interne Regler wird nicht benötigt.

B – 5V Spannungsversorgung von Außen, 5V intern

Die Versorgungsspannung von 5V wird über das äußere Stecksystem an Pin B2 angelegt. Damit der Z180 versorgt wird, sind die Kontakte 1 und 2 zu schließen. Der interne Regler wird nicht benötigt. Das Modul läuft nun mit 5V (!) Versorgungsspannung. Dieser Betrieb ist NICHT in Kombination mit dem AVR-Stamp zulässig, da dieses Modul ausschließlich mit 3.3V betrieben wird.

C – 5V Spannungsversorgung von Außen (z.B. USB) , 3.3V intern

Die Versorgungsspannung von 5V wird über das äußere Stecksystem an Pin B2 angelegt. Zur internen 3.3V Versorgung der Z180 wird der interne Spannungsregler genutzt. Dazu sind die Kontakte 2 und 3 zu schließen. Soll auch das AVR-Modul versorgt werden, sind zusätzlich die Kontakte 2 und 4 zu schließen. Damit werden den intern erzeugten 3.3V auf das äußere Stecksystem (Pin B1) gelegt. Läuft das AVR-Modul mit einer 5V USB-Spannung (JP1), erfolgt in dieser Beschaltungsvariante auch die 3.3V Versorgung über den internen Regler.

Hardwareversion 1.1

AVR-Stamp V1.1

Das AVR-Stamp Modul - Version 1.1 kann mit 3.3V oder mit 5V betrieben werden. **ACHTUNG** eine Mischbestückung AVR (3.3V) und Z180 (5V) oder umgekehrt ist nicht zulässig! Die 3.3V Versorgungsspannung wird nicht auf dem AVR-Stamp erzeugt und muss in jedem Fall über Pin B1 von außen zugeführt werden. Die 5V Versorgungsspannung kann über Pin B2 extern zugeführt werden oder sie wird intern über die USB-Schnittstelle gewonnen. Hierbei ist die Strombelastbarkeit des speisenden USB-Port zu beachten.

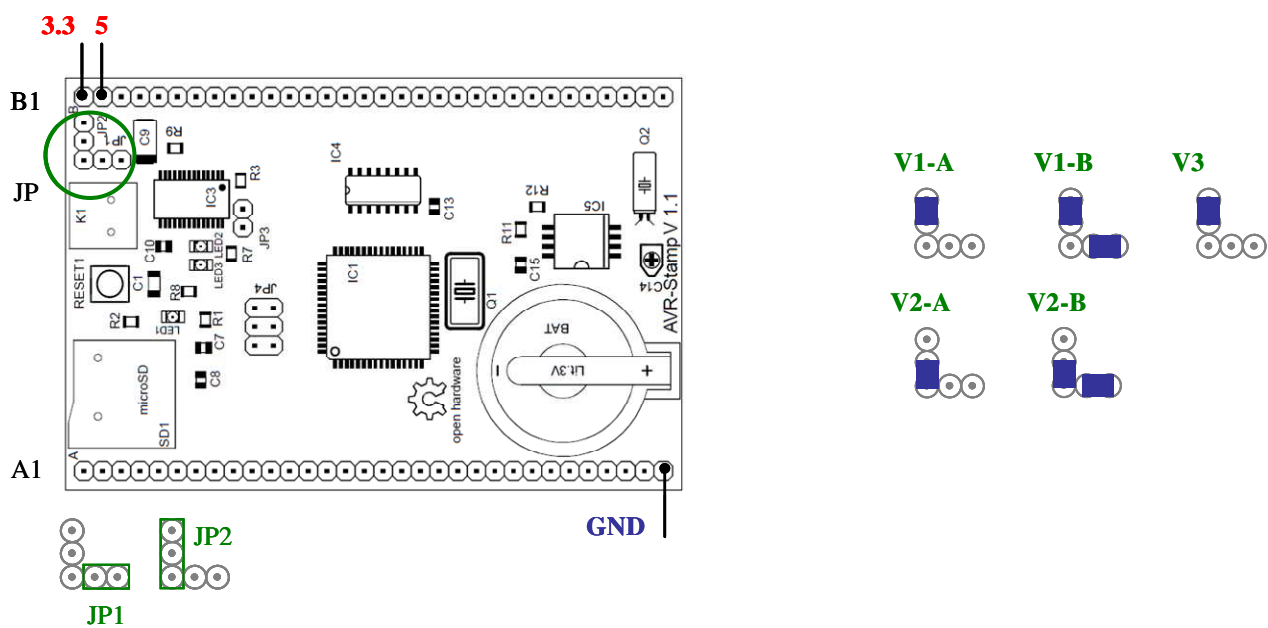


Abb.3: Versorgungsspannungen am AVR-Stamp Modul

Stromversorgungsvarianten für das AVR-Stamp Modul der Version 1.1

Variante	USB	extern	intern
V1-A	-	5V	3.3V
V1-B	5V	-	3.3V
V2-A	-	5V	5V
V2-B	5V	-	5V
V3	-	3.3V	3.3V

Z180-Stamp V1.1

Das Z180-Stamp Modul – Version 1.1 besitzt neben den schon erwähnten Versorgungspins der Anschlussleisten A und B noch zusätzliche Versorgungsvarianten.

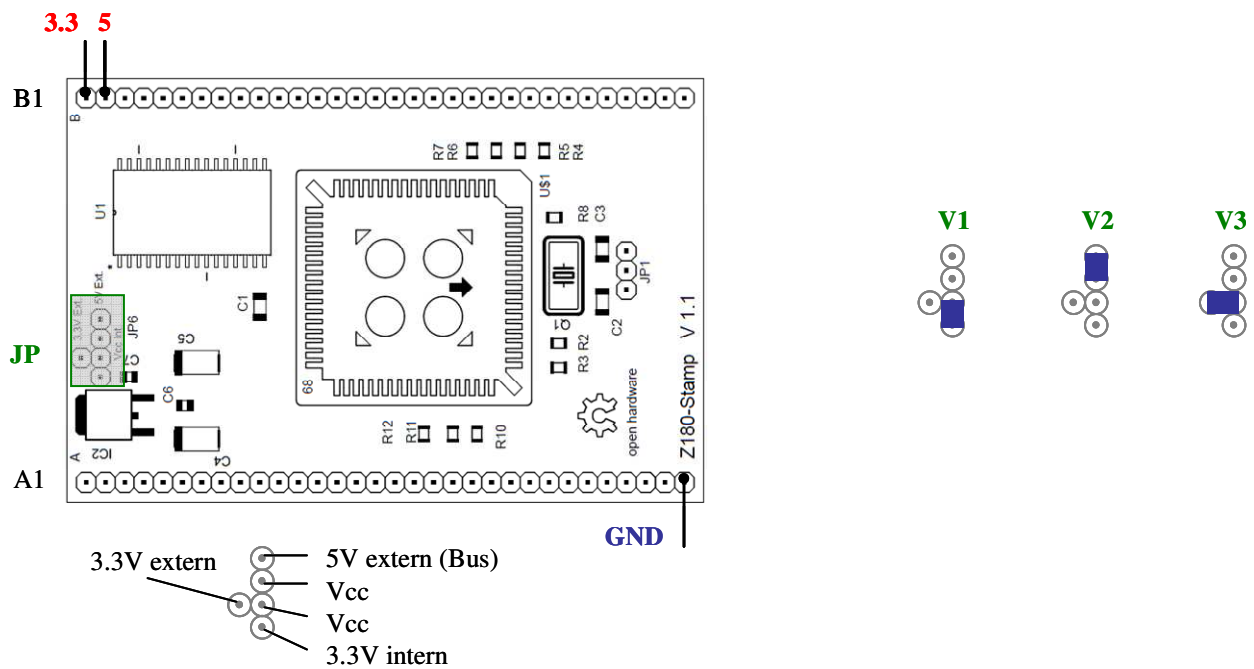


Abb.4: Versorgungsspannungen am Z180-Stamp Modul

Damit sind die folgenden Versorgungsvarianten möglich:

Variante	extern	intern
V1	5V	3.3V
V2	5V	5V
V3	3.3V	3.3V

BOOTLOADER

Der Bootloader ist ein spezielles Programm im AVR, dessen Aufgabe nur darin besteht, das eigentliche Controllerprogramm in den Flashspeicher des Controllers zu laden. Der Ladevorgang des Anwenderprogrammes erfolgt über die serielle Schnittstelle des Controllers. Normalerweise startet der Controller die Abarbeitung seiner Programmierung an der Stelle 0x0000. Da der Bootloader-Bereich des AVR jedoch am Ende des Flash-Speichers liegt, ist der AVR so zu konfigurieren, dass er nach einem RESET von der Bootloaderadresse gestartet wird. Diese Konfiguration ist wie alle wichtigen und grundlegenden Konfigurationen über die Fuses des AVR geregelt.

Boot Size Configuration

Der ATmega1280/1281 besitzt einen maximalen Bootspeicher von 4096 Words (Bootadresse 0xF000). Für den verwendeten Bootloader *FastBoot* von Peter Dannegger [1] reichen jedoch 512 Words aus. Damit liegt die Bootadresse bei 0xFE00. Diese Konfiguration ist über die beiden Fuses **BOOTSZ1** (Bit 2) und **BOOTSZ0** (Bit 1) festzulegen. Beide erhalten den Wert **1**. Weiterhin muss dem AVR mitgeteilt werden, dass er nach einem RESET direkt in die Bootloader Sektion springt. Das wird mit der Fuse **BOOTRST** (Bit 0) festgelegt. Auch diese muss den Wert **1** erhalten. Bei Unklarheiten hilft ein Blick in das Datenblatt des ATmega1280/1281.

Fuse Configuration

Für den generellen Betrieb im CP/M System sind weitere Fuse Einstellungen notwendig. Dazu ist das Fuse High Byte, das Fuse Low Byte und das Extended Fuse Byte richtig zu konfigurieren.

Fuse High Byte

Fuse High Byte	Bit No	Description	Default Value
OCDEN	7	Enable OCD	1 (unprogrammed) OCD disabled
JTAGEN	6	Enable JTAG	0 (programmed) JTAG enabled
SPIEN	5	Enable Serial Program and Data Downloading	0 (programmed) SPI prog. Enabled
WDTON	4	Watchdog Timer always on	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed) EEPROM not preserved
BOOTSZ1	2	Select Boot Size	0 (programmed)
BOOTSZ0	1	Select Boot Size	0 (programmed)
BOOTRST	0	Select Reset Vector	1 (unprogrammed)

Fuse High Byte erhält den Wert: **0xD6**

Fuse Low Byte

Fuse High Byte	Bit No	Description	Default Value
CKDIV8	7	Divide clock by 8	0 (programmed)
CKOUT	6	Clock output	1 (unprogrammed)
SUT1	5	Select start-up time	1 (unprogrammed)
SUT0	4	Select start-up time	0 (programmed)
CKSEL3	3	Select Clock source	0 (programmed)
CKSEL2	2	Select Clock source	0 (programmed)
CKSEL1	1	Select Clock source	1 (unprogrammed)
CKSEL0	0	Select Clock source	0 (programmed)

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

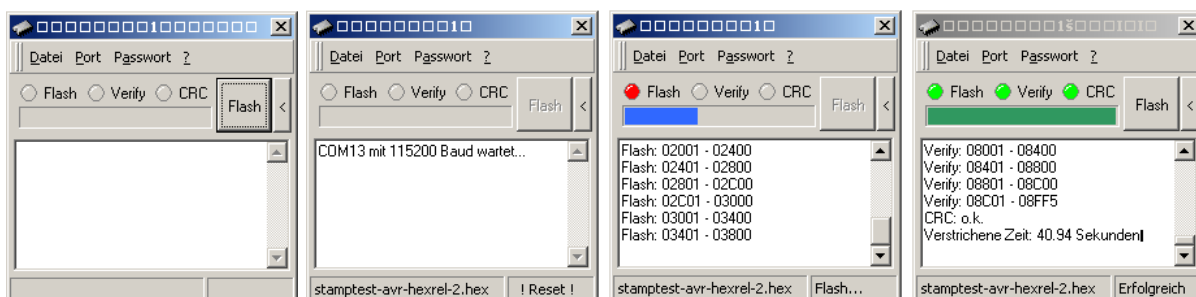
Extended Fuse

Fuse High Byte	Bit No	Description	Default Value
BODLEVEL2	2	Brown-out Detector trigger level	1 (unprogrammed)
BODLEVEL1	1	Brown-out Detector trigger level	1 (unprogrammed)
BODLEVEL0	0	Brown-out Detector trigger level	1 (unprogrammed)

Extended Fuse Byte erhält den Wert: **0xF5**

Host Software

Um auf der PC-Seite das AVR Anwenderprogramm mittels serieller Schnittstelle zu übertragen, ist eine geeignete Software notwendig. Unter dem Betriebssystem WINDOWS bietet sich die Software *AVRFlash* [2] oder *Updateloader* [3] an. Für Linux oder OS X ist eine äquivalente Hostsoftware unter [6] zu finden. Die Bedienung von AVRFlash ist sehr einfach. Unter dem Menüpunkt Datei wird die HEX-Datei (Intel-Hex-Format) gewählt, die auf den Controller übertragen werden soll. Unter dem Menüpunkt Port werden der COM-Port und die Baudrate eingestellt. Als Baudrate ist 115200 zu wählen. Nun kann durch das Betätigen des Flash Buttons der Download ausgelöst werden. Das Programm wartet nun auf einen RESET am AVR System. Nach dem RESET startet der Download. Ein erfolgreicher Download wird über die drei grünen LEDs signalisiert. Gleichzeitig kann der eigentliche Downloadvorgang über die LED2 und LED3 auf dem AVR-Stamp beobachtet werden.



INBETRIEBNAHME

Die Inbetriebnahme der einzelnen Funktionen wie RTC, RAM oder SD-Card werden durch den *Stamp-Monitor* unterstützt. Der komplette Funktionsumfang, sowie die Bediensyntax kann über den Befehl *help* bzw. *?* abgefragt werden. Zur Kommunikation mit dem *Stamp-Monitor* kann ein beliebiger Terminal-Emulator wie z.B. *Tera Term* verwendet werden. Dazu sind die folgenden Kommunikationsparameter zu wählen.

Baud rate	115200
Data	8 bit
Parity	none
Stop	1 bit
Flow control	none

Nach dem betätigen des Reset-Tasters auf dem AVR-Stamp-Modul muss sich der *Stamp-Monitor* mit einer Startinfo melden. Läuft die Startsequenz vollständig durch, so können ab dieser Stelle die einzelnen Monitorfunktionen genutzt werden.

RTC

Auf dem AVR-Stamp-Modul befindet sich eine Echtzeituhr (RTC) welche über eine Pufferbatterie gestützt wird. Die Kommunikation der RTC (IC4) mit dem AVR erfolgt über einen I2C-Bus. Dazu muss zunächst hardwareseitig die korrekte I2C Busadresse der RTC eingestellt werden. Standardmäßig ist die Adresse A0h für Write und A1h für Read vorgesehen. Dazu ist das Pin 3 (A0-RTC) von IC4 auf GND zu legen. Jetzt hat der *Stamp-Monitor* Zugriff auf die RTC.

Abfragen der Uhr

Der Befehl **date** ohne Argumente liefert das aktuelle Datum und die Uhrzeit der RTC.

```
=>date
```

```
Mon Dec 22 10:30:00 2014
```

Stellen der Uhr

Das Stellen der Uhr erfolgt über das Kommando **date MMDDhhmmCCYY.ss**

Bsp.: 22.12.2014 um 10:30 Uhr

```
=> date 122210302014.00
```

Der Monitor quittiert eine korrekte Eingabe mit der Meldung des Datums und der Zeit.

```
Mon Dec 22 10:30:00 2014
```

SD-Card

Das AVR-Stamp-Modul verfügt über zwei Varianten eine SD-Card anzuschließen. Die Basisvariante besteht aus einem Micro-SD-Sockel direkt auf dem AVR Board. Für eine Erweiterung kann eine zweite SD-Card an der externen Stiftleiste des AVR-Stamp-Moduls bestückt werden.

Funktion	Port-Pin	Steckverbinder
CS	PG4	A12
WP	PG5	A13
SCK	PB1	A14
MOSI	PB2	A15
MISO	PB3	A16
CD	PG3	A21

Um das CS-Pin (PG4/A12) zur Kartenerkennung zu nutzen, ist an dieser Stelle ein Pulldown Widerstand von ca. 330k vorzusehen. In Kombination mit dem internen Pullup Widerstand der SD-Card ergibt sich eine einfache Möglichkeit eine gesteckte SD-Card zu detektieren.

Test einer SD-Card

Die Low-Level Funktionen der SD-Card werden über den `sd` Befehl bereitgestellt.

Mit dem Befehl `sd status x`

`x=0` On-Board-Card

`x=1` externe Card

kann der aktuelle Status des jeweiligen SD-Sockels abgefragt werden. Der Monitor antwortet mit einer entsprechenden Statusmeldung.

Code	Meldung
00	Drive initialized
01	Drive not initialized
02	No medium in the drive
04	Write protected
08	Fast SPI clock

Nach einem Monitor-Reset oder einem Kartenwechsel sollte der Befehl

```
=> sd status 0
```

bei eingelegter Karte die Antwort

```
Socket status: 01
```

ergeben (Drive not initialized).

Im nächsten Schritt kann die Karte mit dem Befehl

```
=> sd init 0
```

initialisiert werden. Als Antwort erhält man bei erfolgreicher Initialisierung den Disk-Status (siehe Statusabelle). Die Antwort

```
rc=00
```

zeigt eine erfolgreiche Initialisierung an.

Mit dem Befehl

```
=> sd info 0
```

kann nun der Zustand der SD-Card ermittelt werden. Die Monitorausgabe beinhaltet eine Vielzahl von Informationen. Nähere Auskunft über die Registerbelegungen sind im SanDisk SD Card Product Manual, Version 2.2 zu finden [4]. Vertiefende Informationen zu SD-Karten findet man unter [7] und [8]. Verlaufen alle diese Schritte erfolgreich, kann die SD-Card bzw. darauf aufbauende Kommandos verwendet werden.

Filesystem

Neben den Low-Level Funktionen der SD-Card beherrscht der Monitor auch das FAT16 und FAT32 Dateisystem. Das ermöglicht dem AVR oder dem Z180 einen eleganten Dateizugriff. Zunächst kann der Status des auf der Karte implementierten Filesystems ermittelt werden.

```
=> fatstat 0:
```

```

FAT type:                2
Bytes/Cluster:           16384
Number of FATs:          2
Root DIR entries:        512
Sectors/FAT:             242
Number of clusters:      61935
FAT start (lba):         1
DIR start (lba,cluster): 485
Data start (lba):        517
Volume name:
Volume S/N:              1234-5678

```

```

66 files, 924663 bytes.
6 folders.
990960 KB total disk space.
989120 KB available.

```

Über **fatls 0**: werden alle Dateien und Verzeichnisse im Wurzelverzeichnis angezeigt.

```

=> fatls 0:
----A 2014/12/18 15:37    199238 stamp-monitor-avr-hexrel-4.hex
----A 2014/09/13 22:17    103673 stamp-test-avr-hexrel-2.hex
----A 2014/12/21 15:28    199258 stamp-monitor-hexrel-4.1.hex
D---- 2014/12/21 18:51         0 stamp-test-hexrel-1
----A 2014/10/14 21:45    113565 stamp-monitor.hex
   4 File(s),    615734 bytes total
   1 Dir(s),    989120K bytes free

```

Die erweiterte Syntax für Laufwerke, Namen und Verzeichnisse ist unter [5] nachzulesen.

RAM

Das Z180-Stamp-Modul verfügt in der derzeitigen Ausbaustufe über einen 512k x 8 Bit CMOS SRAM (Adressbereich von 0 -7FFFF). Der gesamte RAM-Bereich kann über die Adressleitungen A0 - A18 durch die Z180 CPU adressiert werden. Bei der Inbetriebnahme ist ein RAM-Test sehr hilfreich. Hier können

RAM-Fehler oder kalte Lötstellen bzw. Brücken auf den Adress- und Datenleitungen identifiziert werden. Eine Möglichkeit dazu bietet der Monitorbefehl **mtest x y z**.

x – Startadresse

y – Endadresse

z – Testiterationen

Soll also der gesamte verfügbare RAM-Bereich z. B. über 10 Iterationen überprüft werden, so kann das mit dem nachfolgenden Befehl erfolgen.

```
=> mtest 0 7FFFF A
```

Bei erfolgreichem RAM-Test antwortet der Monitor mit der Befehlsantwort

```
Testing 00000 ... 7ffff:
```

```
Tested 10 iteration(s) with 0 errors.
```

MONITOR

Das AVR-Stamp-Modul besitzt ein eigenes Monitorprogramm. Dieses Programm läuft direkt auf dem AVR und stellt sowohl für den AVR als auch für den Z180 Test- und Debugging – Funktionen zur Verfügung. Die Bedienung des Monitors erfolgt vollständig über die Konsolenschnittstelle. Nach einem Stamp-Reset meldet sich zunächst der Monitor mit seiner Startmeldung:

```
=====< (RE)START DEBUG >=====
### Reset reason(s): External.
### Setting I2C clock Frequency to 100000 Hz.

ATMEGA1281+Z8S180 Stamp Monitor

### main_loop entered: bootdelay=3

### main_loop: bootcmd="reset; loadf; go ${startaddr}"
Hit any key to stop autoboot: 0
## CPU now in reset state.
Loading Z180 memory...
  From: 0x00000 to: 0x001B9   ( 442 bytes)
  From: 0x001C2 to: 0x02D9E   (11229 bytes)
  From: 0x02DAE to: 0x03194   ( 999 bytes)
## Starting application at 0x0000 ...
=>
```

Jetzt ist der Monitor einsatzbereit. Eine Übersicht über alle Monitorkommandos erhält man mit dem Befehl **help** oder **?**.

Befehlssatz / Kommandos

cmp - memory compare

Das Memory Compare Kommando vergleicht den Inhalt zweier Speicherzellen miteinander.

Syntax	cmp	x	y	z
	x	RAM Adresse 1		
	y	RAM Adresse 2		
	z	Anzahl der Speicherzellen die verglichen werden		

md - memory display

Das Memory Display Kommando listet den Speicherinhalt des RAM sowohl in hexadezimaler als auch in ASCII-Darstellung auf.

Syntax	md	x	[y]
	x	Startadresse im RAM	
	y	Anzahl der Speicherzellen	

mdc - memory display cyclic

Das zyklische Memory Display Kommando listet den Speicherinhalt des RAM sowohl in hexadezimaler als auch in ASCII-Darstellung auf. Dabei wird die Auflistung periodisch wiederholt. Das Kommando kann mit dem Befehl **CTRL+C** abgebrochen werden.

Syntax	mdc	x	y	z
	x	Startadresse im RAM		
	y	Anzahl der Speicherzellen		
	z	Wiederholzeit in ms		

mw - memory write

Das Memory Write Kommando beschreibt den Speicherinhalt des RAM's.

Syntax	mw	x	y	[z]
	X	Startadresse im RAM		
	y	Speicherinhalt		
	z	Anzahl der Speicherzellen		

mwc - memory write cyclic

Das zyklische Memory Write Kommando beschreibt zyklische eine Speicherzelle. Das Kommando kann mit dem Befehl **CTRL+C** abgebrochen werden.

Syntax	mwc	x	y	Z
	X	RAM Adresse		
	y	Speicherinhalt		
	z	Wiederholzeit in ms		

Skript Abarbeitung des Monitors

Der Monitor gestattet die automatisierte Abarbeitung von Befehlsfolgen. So müssen komplexe Befehlskombinationen nicht mühsam jedes Mal per Hand wiederholt werden. Die Befehlsausführung erfolgt über den Befehl

```
=> run Var
```

wobei die Variable selbst wieder mehrere Befehle enthalten darf. Exemplarisch sollen die Befehlsfolgen zum Test einer SD-Card in ein ausführbares Skript gebracht werden. Um eine SD-Card zu testen und sich den Inhalt ihres FAT-Filesystems anzeigen zu lassen, sind mehrere Befehle notwendig:

```
=> sd status 0
```

```
=> sd init 0
```

```
=> sd info 0
```

```
=> fatstat 0:
```

```
=> fatls 0:
```

Wir wollen nun alle SD Low-Level Befehle zu einer Gruppe mit dem Variablennamen **test_sd** zusammenfassen. Dazu wird die Environmentvariable **test_sd** mit dem Befehl

```
=> setenv test_sd 'sd status 0; sd init 0; sd info 0'
```

gefolgt vom Befehlsanweisungen angelegt. Eine Befehlskette wird in Hochkomma eingeschlossen und die einzelnen Befehle selbst durch Semikolon getrennt.

Über den Befehl

```
=> printenv
```

kann der Inhalt der Variable überprüft werden. Ist diese korrekt, erfolgt die Befehlsausführung über den Befehl

```
=> run test_sd
```

Hierbei werden nun alle drei Befehle hintereinander abgearbeitet. Den gleichen Weg beschreiben wir nun für den Test des FAT-Filesystems indem wir diese Befehle geeignet zusammenfassen.

```
=> setenv test_fat 'fatstat 0;; fatls 0:'
```

Auch diese Variable kann nun einzeln mit

```
=> run test_fat
```

getestet werden. Waren alle Konfigurationen erfolgreich, werden die neu angelegten Environmentvariablen mit dem Befehl

```
=> saveenv
```

dauerhaft im EEPROM gespeichert. Die komplette Überprüfung der SD-Card kann anschließend mit dem einfachen Befehl

```
=> run test_sd test_fat
```

ausgeführt werden.

Einige Befehle legen selbst eine Environmentvariable an. So lädt der Befehl

```
=> loadc
```

das CP/M 3 BIOS vom FAT-Filesystem der SD-Card und erzeugt gleichzeitig die Environmentvariable **startaddress** mit der Kaltstarteinsprungadresse des CP/M. Diese Adresse kann nun dem **go** Befehl als Startadresse mitgegeben werden:

```
=> go ${startaddress}
```

Die Variable **bootcmd** hat für den Monitor eine besondere Bedeutung. Sie wird während des Autobootvorganges des Monitors automatisch ausgeführt. Hier können also alle Befehlssequenzen für einen kompletten CP/M Neustart vereinbart und ausgeführt werden. Mit

```
=> setenv bootcmd 'reset; loadf; loadc; go ${startaddress}'
```

```
=> run bootcmd
```

startet also das CP/M auf der Default-Konsole. Soll die Konsolenausgabe direkt im Monitor erfolgen, so ist **bootcmd** wie folgt zu setzen:

```
=> setenv bootcmd 'reset; loadf; loadc; go ${startaddress}; connect'
```

Im CP/M 3 muss natürlich **PROFILE.SUB** entsprechend angepasst werden.

```
PROFILE.SUB
```

```
device CON:=USB0
```

So könnte z.B. eine komplette Bootumgebung aussehen:

```
attach_drives=at dsk0 1:/cpm3_a.dsk;at dsk1 1:/cpm3_b.dsk;at dsk2
0:/cpm3_c.dsk;at dsk3 1:/z180-stamp-cpm3-bios.dsk;at dsk4 1:/z180.dsk
```

```
baudrate=115200
```

```
bootcmd=reset; loadc; run attach_drives; go ${startaddress}
```

```
bootdelay=3
```

```
cpm3_commonbase=f000
```

```
cpm3_file=1:/cpm3_0.6.8-23.sys
```

```
cpm3_scb=1e800
```

```
esc_char=0x40
```

```
pin_alias=0:PG5,1:PG4,2:PB4,3:PB5,4:PB6,5:PB7,6:PG3,7:PG2,8:PG1,9:PG0
,10:PE7
```

```
singlestep=1
```

```
startaddress=ad00
```



```
test_fat=fatstat 0:; fatls 0:
test_sd=sd status 0; sd init 0; sd info 0
```

Eine komplette Übersicht der derzeit verwendeten Environmentvariablen zeigt die nachfolgende Tabelle.

Name	Type	Default	defaultenv
baudrate	Dezimal	"115200"	"115200"
bootcmd	String	-	"pin \${pins}; loadcpm3; go \${startaddress}"
bootdelay	Dezimal	"4"	"3"
cpm3_bankedbase	Hex	"0"	
cpm3_commonbase	Hex	-	
cpm3_file	String	"0:/cpm3.sys"	"0:/cpm3.sys"
dsk0 ... dsk4	String	"1E"	
pin_alias	String	-	"0:PG5,1:PG4,2:PB4,3:PB5,4:PB6,5:PB7,6:PG3,7:PG2,8:PG1,9:PG0,10:PE7"
pins	String	-	"2,8 low 9 high 3 2"
singlestep	Bool	-	
startaddress	Hex	-	"0"

Ist durch den Anwender noch kleine entsprechenden Variable definiert, wird die **Default** Spalte benutzt um diese Variable zu erzeugen. Ist beim Monitorstart das EEPROM noch leer oder fehlerhaft beschrieben, werden die Defaultvariablen mit dem Inhalt **defaultenv** beschrieben. Diese Belegung wird auch bei der Ausführung des Monitorbefehls **defaultenv** durchgeführt.

Bool-Variablen (singlestep) sind "wahr", wenn sie existieren und das erste Zeichen "1" oder "y" oder "Y" oder "t" oder "T" ist, sonst sind sie "unwahr".

































































cpm3_commonbase und startadress werden durch loadcpm3 gesetzt bzw. überschrieben.

QUELLENANGABEN

- [1] 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
- [3] <http://www.leo-andres.de/2012/09/updateloader-benutzeroberflache-fur-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
- [8] https://www.sdcard.org/downloads/pls/simplified_specs/

ANHANG A

Pinzuordnung der Stamp Module

AVR	Z180	all	A	B	all	Z180	AVR
nc	DESEL	DESEL	 1	1	 +3.3V	+3.3V	+3.3V
/ARESET	nc	/ARESET	 2	2	 +5.0V	+5.0V	+5.0V
nc	A19	A19	 3	3	 A18	A18	PE4
PF0	D0	D0	 4	4	 A17	A17	PE3
PF1	D1	D1	 5	5	 A16	A16	PE2
PF2	D2	D2	 6	6	 A15	A15	PC7
PF3	D3	D3	 7	7	 A14	A14	PC6
PF4	D4	D4	 8	8	 A13	A13	PC5
PF5	D5	D5	 9	9	 A12	A12	PC4
PF6	D6	D6	 10	10	 A11	A11	PC3
PF7	D7	D7	 11	11	 A10	A10	PC2
PG5	/RTS0	do not stack	 12	12	 A9	A9	PC1
PG4	/CTS0	do not stack	 13	13	 A8	A8	PC0
SCK	/DCD0	do not stack	 14	14	 A7	A7	PA7
MOSI	TXA0	do not stack	 15	15	 A6	A6	PA6
MISO	RXA0	do not stack	 16	16	 A5	A5	PA5
PB4	CKA0	do not stack	 17	17	 A4	A4	PA4
PB5	TXA1	do not stack	 18	18	 A3	A3	PA3
PB6	RXA1	do not stack	 19	19	 A2	A2	PA2
PB7	CKA1	do not stack	 20	20	 A1	A1	PA1
PG3	TXS	do not stack	 21	21	 A0	A0	PA0
PG2	RXS	do not stack	 22	22	 /NMI	/NMI	nc
PG1	CKS	do not stack	 23	23	 /ZRESET	/RESET	PD5
PG0	/DREQ1	do not stack	 24	24	 /BUSREQ	/BUSREQ	PD7
SDA	/TEND1	do not stack	 25	25	 /BUSACK	/BUSACK	PD6
SCL	/HALT	do not stack	 26	26	 do not stack	/WAIT	RXD0
nc	/RFSH	/RFSH	 27	27	 do not stack	PHI	TXD0
PE7	EXTAL	CLKO	 28	28	 /RD	/RD	PD3
nc	/INT0	/INT0	 29	29	 /WR	/WR	PD2
nc	/INT1	/INT1	 30	30	 /M1	/M1	nc
nc	/INT2	/INT2	 31	31	 /MREQ	/MREQ	PD4
GND	GND	GND	 32	32	 /IORQ	/IORQ	nc

ANHANG B

Steckverbinder

CON0 – RS-232

Data Terminal Equipment (DTE)

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

CON2 – RS-232 (USB)

Data Terminal Equipment (DTE)

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

CON3 – Terminal (USB)

Data Terminal Equipment (DTE)

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

Pin [Hz]	Name	Port	Timer	Mode	max div	max div	min f
0		PG5	OC0B	PWM	$(2^{**}8) * 1024$	262144	70.31
1		PG4					
2	CLK2	PB4	OC2A	Toggle	$(2^{**}8) * 1024 * 2$	524288	35.16
3	ZCLK	PB5	OC1A	PWM	$(2^{**}16) * 1024$	67108864	0.2746
4		PB6	OC1B	PWM	$(2^{**}16) * 1024$	67108864	0.2746
5		PB7	OC0A	Toggle	$(2^{**}8) * 1024 * 2$	524288	35.16
6		PG3					
7		PG2					
8		PG1					
9		PG0					
10	CLKO	PE7					

```

=> pin
Pin Config Level Divider Frequency/Hz
-----
0 Input Low
1 Input Low
2 Input High
3 Input High
4 Input Low
5 Input Low
6 Input Low
7 Input Low
8 Input Low
9 Input Low
10 Input High
=> pin 2 Low 3 9MHz
=> pin 2,3
Pin Config Level Divider Frequency/Hz
-----
2 Output Low
3 Clock 2 9216000
=>

```

```

=> printenv
baudrate=115200
bootcmd=reset; loadc; run run_step; go ${startaddress}
bootdelay=3
cpm3_commonbase=F000
cpm3_file=1:/cpm3_v64.sys
dsk0=1:/cpm3_a.dsk
dsk1=1:/cpm3_b.dsk
dsk2=0:/cpm3_c.dsk
esc_char=0x40
pin_alias=0:PG5,1:PG4,2:PB4,3:PB5,4:PB6,5:PB7,6:PG3,7:PG2,8:PG1,9:PG0,10:PE7
run_step=pin 8 low
startaddress=cd00
test_fat=fatstat 0;; fatls 0:
test_sd=sd status 0; sd init 0; sd info 0

Environment size: 387/1597 bytes
=>

```

Eine Aufbau- und Inbetriebnahmeanleitung ist für die alten Hasen die damals Weihnachten neben der Modelleisenbahn auch einen Z80 gefunden haben, nicht notwendig. Doch für alle diejenigen, die mit 76h nichts mehr anfangen können, vielleicht sehr hilfreich.

Die Inbetriebnahme sollte mit den boardeigenen Mitteln durchführbar sein.

Schritt 1: AVR-Stamp

AVR-Stamp vollständig bestücken. Dazu gehört auch der 18.432 MHz Quarz. Die Z180 CPU leitet später ihren Takt von diesen Quarz ab. Die 5V Stromversorgung kann über die USB-Schnittstelle gewonnen werden. Die 3.3V müssen zur Inbetriebnahme noch extern zugeführt werden. Ohne die 3.3V laufen die interne SD-Card, LED1, LED2 und LED3 nicht!

JP1 und JP2 entsprechend dem Handbuch stecken und den AVR Bootloader über die ISP Schnittstelle (JP4) flashen. JP3 bleibt offen. Die konkreten Fuseeinstellungen des AVR's sind dem Handbuch zu entnehmen.

USB-Schnittstelle mit dem Host verbinden. Ist der entsprechende Treiber im jeweiligen Betriebssystem installiert, erscheint nun ein zusätzlicher COM-Port. Anschließend kann die aktuelle AVR-Firmware in den AVR eingespielt werden.

Start eines Terminalprogramms auf dem Host mit folgenden Einstellungen:

Port:	COMx
Baud rate:	115200
Data:	8 bit
Parity:	none
Stop:	1 bit

Wird nun der Reset-Taster betätigt, meldet sich das Monitorprogramm mit einer Einschaltmeldung. Unter Zuhilfenahme der Monitorbefehle können nun die Funktion des RTC, der SD-Card und der Bussteuerung überprüft werden.

Schritt 2: ECB-Bus

Eine Bestückung des ECB-Bus Karte erleichtert die weite Inbetriebnahme, ist jedoch nicht zwingend notwendig. Achtung, Steckrichtung und Steckplatz des AVR-Stamp beachten!

Zunächst kann die Funktion der zweiten SD-Card überprüft werden. Sie sollte sich äquivalent zu der internen SD-Card verhalten. An JP3 (ECB-Card) können die Taktsignale für die Z180 CPU gemessen werden. Weiterhin sollte die korrekte Funktion von /ZRESET überprüft werden. Da ohne den Z180-Stamp /HALT noch offen ist, müssen bei der DUO-LED (D2) beide Farben gleichzeitig leuchten (Mischfarbe). LED2 (WAIT) darf nicht leuchten. Das wird über den entsprechenden Monitor-Pin-Befehl realisiert.

Schritt 3: Z180-Stamp

Z180-Stamp bestücken. Achtung, der Z180 bekommt keinen eigenen Quarz! Die Stromversorgung kann nun wie gewünscht (5V/3.3V) über die jeweiligen Jumper gewählt werden. Abschließend wird der Z180-Stamp neben den AVR-Stamp auf den ECB-Bus gesteckt. Über das Monitorprogramm können nun die Funktionen des Z180-Stamp geprüft werden. In erster Linie betrifft das die korrekte Funktion des CMOS-RAM und der CPU. Dazu existiert eine RAM-Testroutine im Monitor. Weiterhin kann der RAM bis auf die vorletzte Speicherzelle mit 00h (NOP) beschrieben werden. Die letzte Speicherzelle erhält den Wert 76h (Halt). Wird nun die CPU ab Adresse 00h gestartet, führt sie bis zum RAM-Ende NOP Befehle aus und geht dann in den Halt (DUO-LED D2 nur rot). Auf dem Adressbus können in dieser Zeit die Adresssignale gemessen werden (verhalten sich wie Binärteiler).

Damit ist der Funktionstest abgeschlossen und die weitere Inbetriebnahme wird mit dem CP/M Plus Betriebssystem vorgenommen.