MFM Emulator / Reader

deutsche Anleitung

28.06.2018

Vorwort

Bei dieser Beschreibung (geschrieben mit WinWord 2010) handelt es sich um das Sichern von Daten aus alten Festplatten, welche eine ST506 Schnittstelle haben und mit MFM formatiert wurden. Mit der im folgenden beschriebenen Hardware können von besagten Festplatten Komplettabzüge (Images) erstellt werden und dann ebenfalls mit der beschriebenen Hardware und Umstecken von Kabel / Jumper die vorher eingelesene Festplatte für den Hostcomputer emuliert werden.

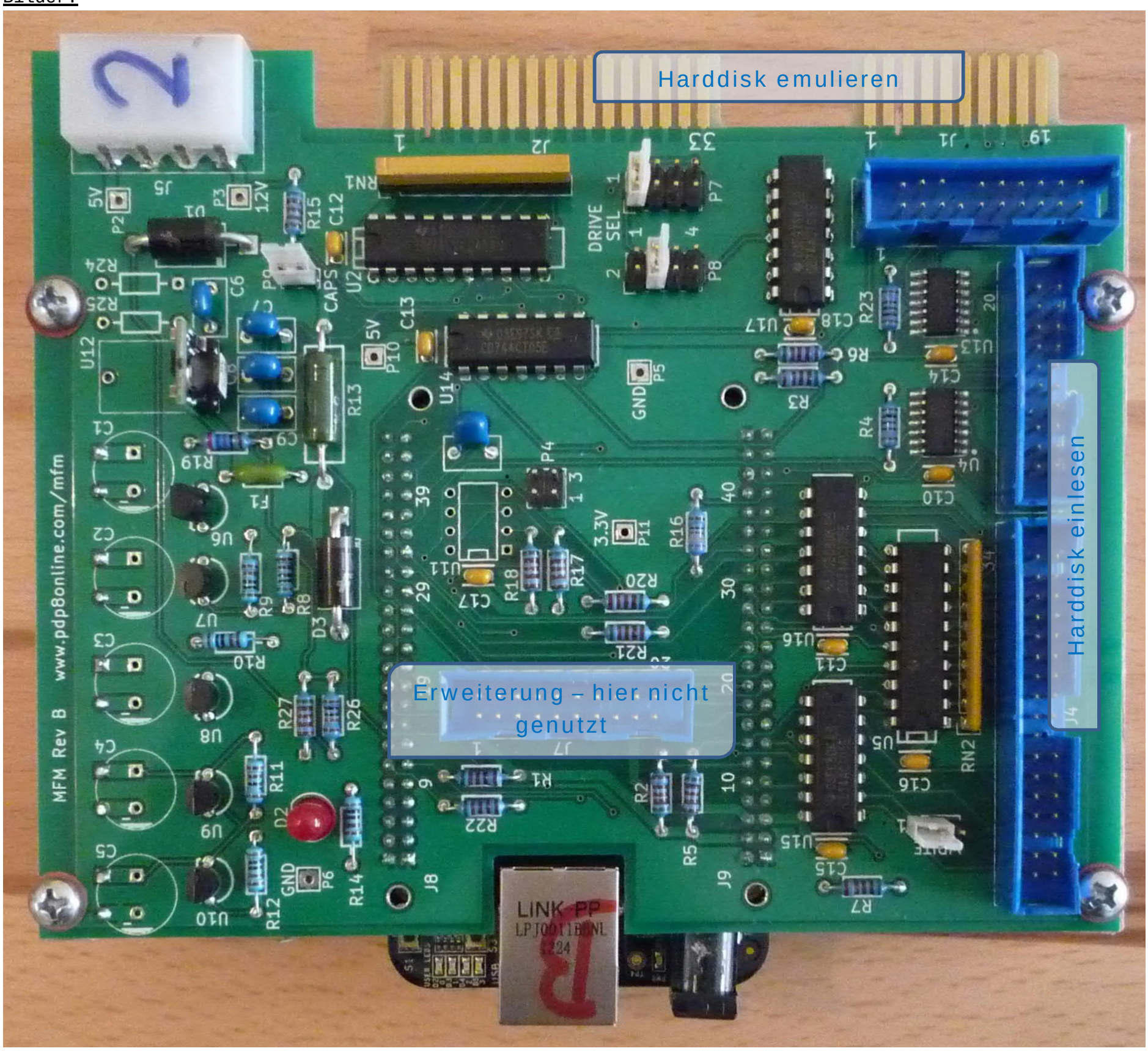
Ich möchte hier ausdrücklich meiner Begeisterung für die Entwicklung von David Gesswein kundtun.
Auf http://www.pdp8online.com/mfm/ hat David den von ihm entwickelten MFM Hard Disk Reader/Emulator beschrieben und davon eine Miniserie für Interessenten produziert.

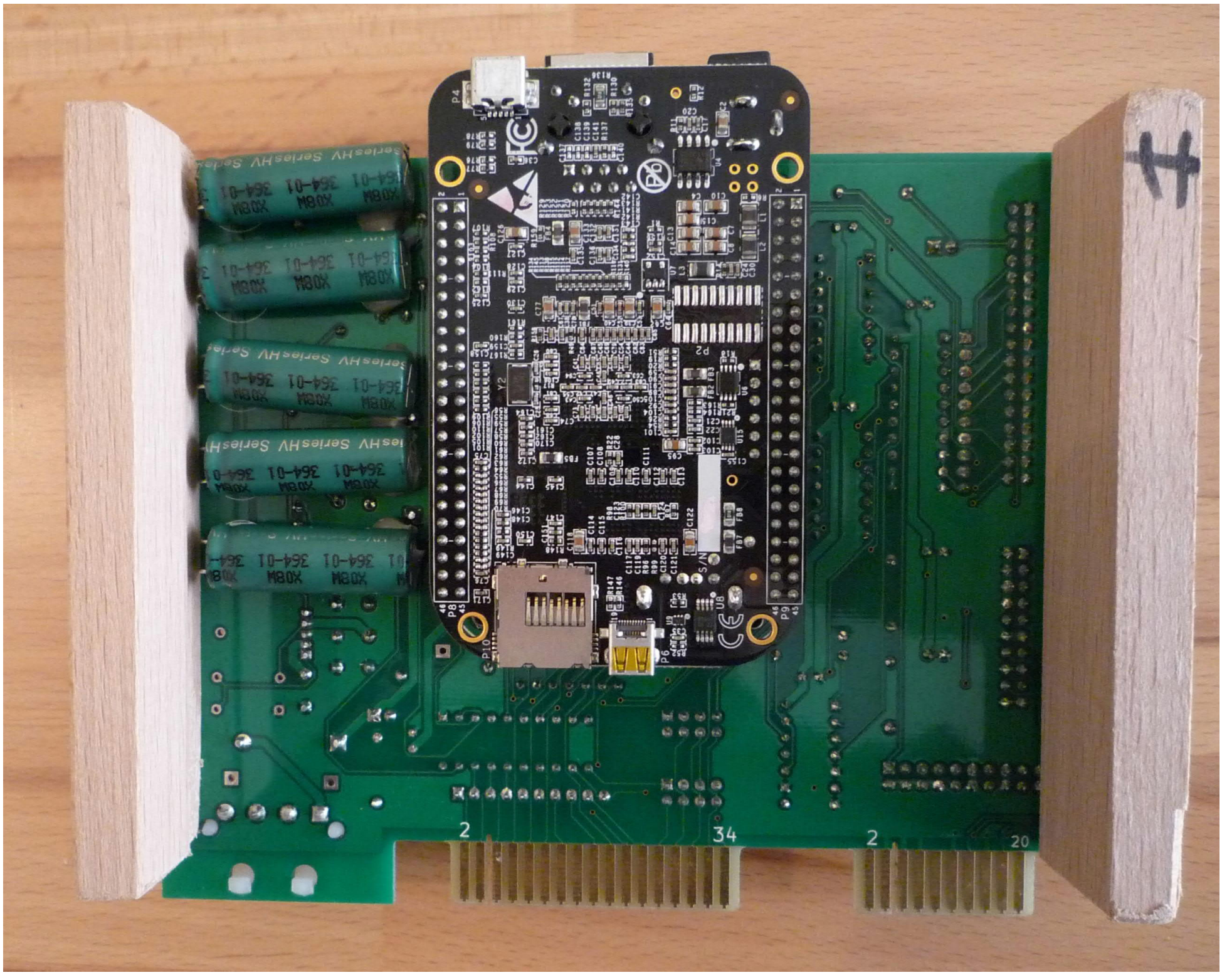
Der MFM Hard Disk Reader/Emulator besteht aus einer Platine zum Anschluss der Festplatte und wird als Adapterboard auf ein Beagle Board gesteckt. Das Beagle Board hat 2x PRU 32-bit Microcontroller (PRU= programmable real-time unit = programmierbare Echtzeit-Einheit) und 2x 46 Pin Buchsenleisten für stapelbare Erweiterungen.

Die von David geschriebene Software nutzt die im BBB vorhandenen Microcontroller (PRU), Betriebssystem ist ein DEBIAN kernel 3.8. Das Image von Davids Installation beinhaltet die Betriebssystemumgebung ohne grafische Oberfläche, ich habe für mich als GUI XFCE installiert.

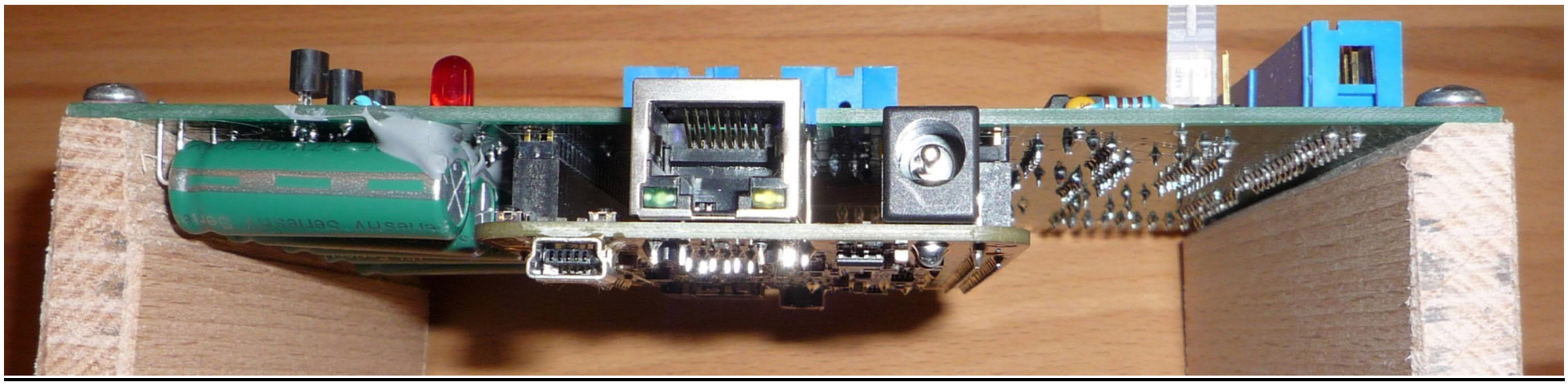
5 großen Kondensatoren dienen als unterbrechungsfreie Stromversorgung zum sauberen Herunterfahren des Betriebssystems wenn die Stromversorgung abgeschaltet wird. Die Emulation startet wieder automatisch (konfigurierbar) beim Einschalten der Versorgungsspannung, womit von der Emulation wie bei der ersetzten Festplatte das Hostbetriebssystem gebootet werden kann.

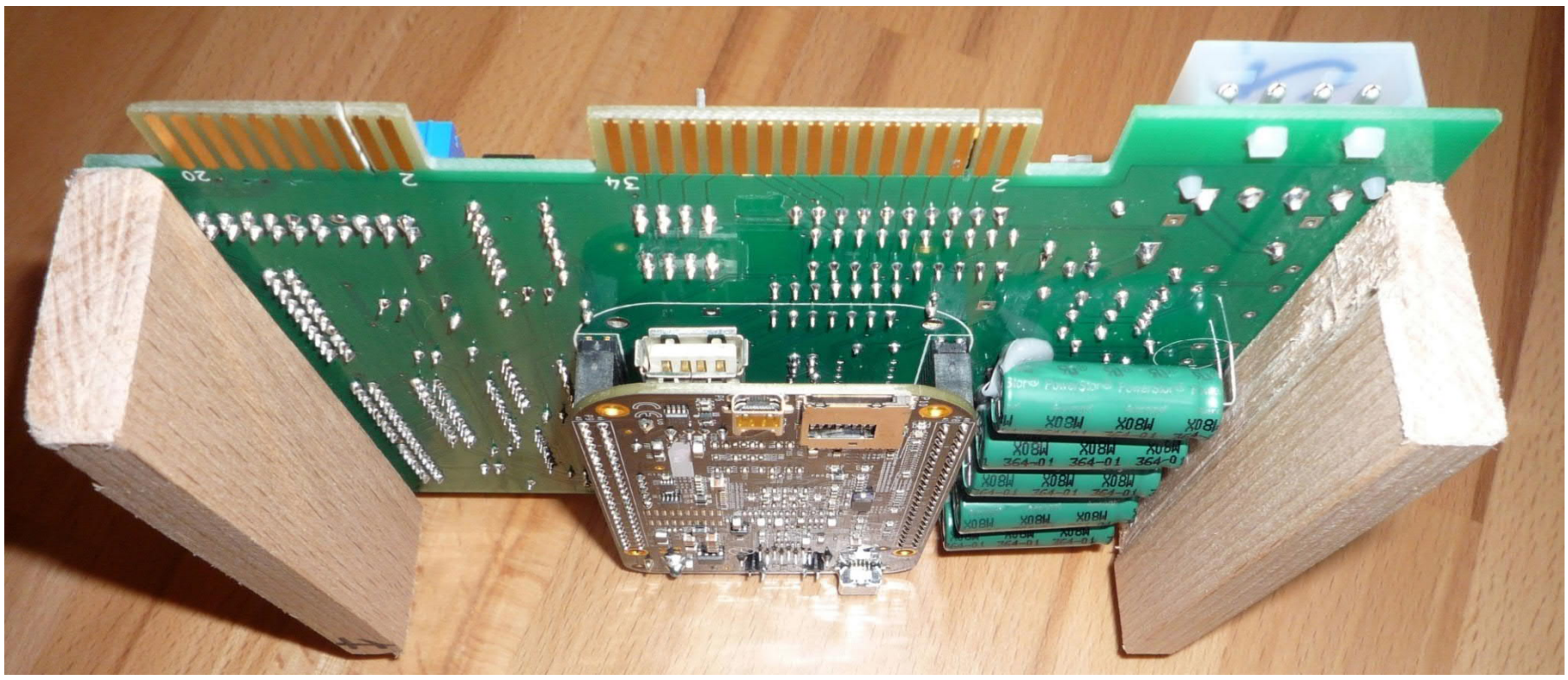
Bilder:

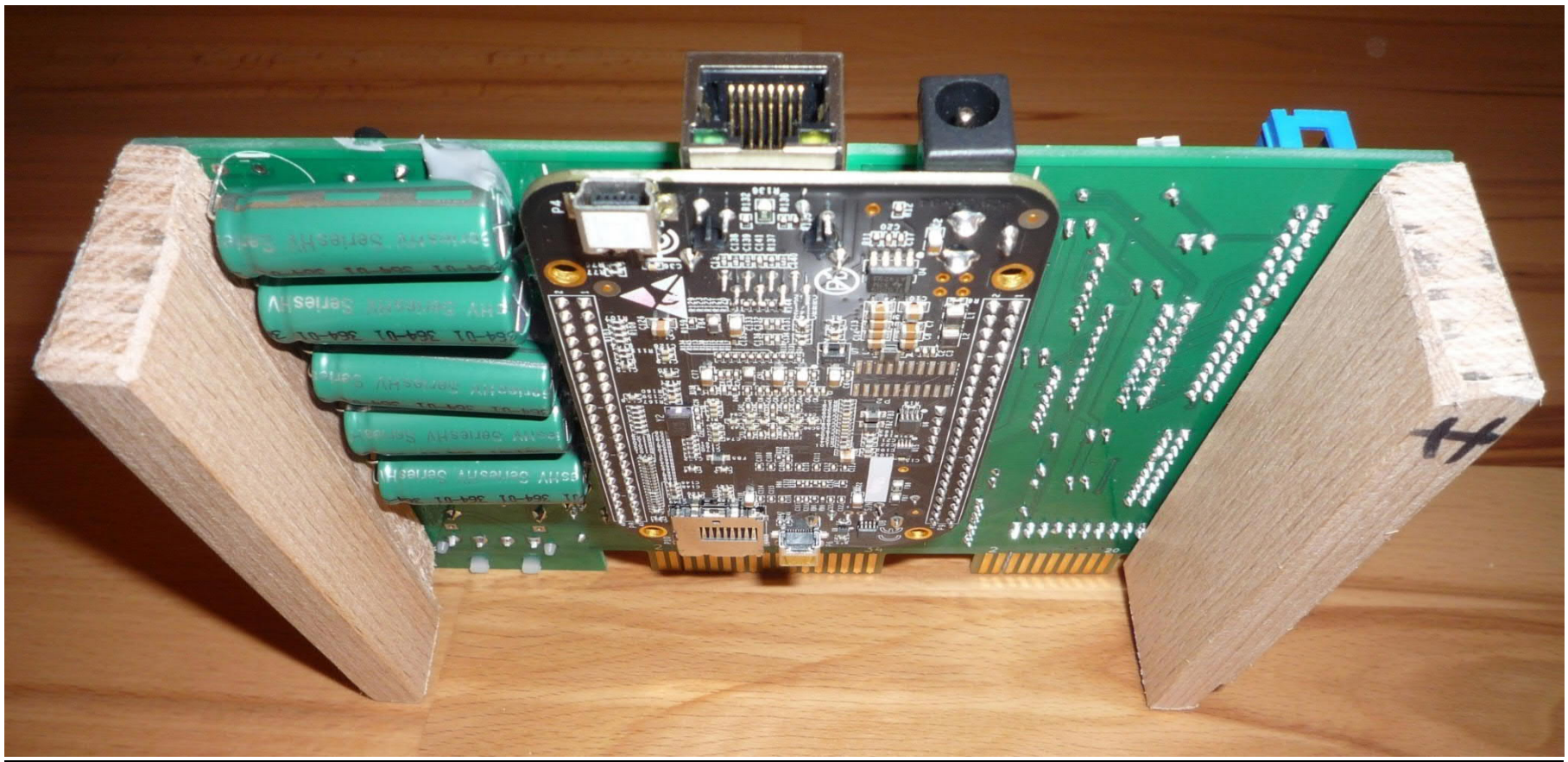




bei Fragen: freiwiederwind1@hotmail.de







http://elinux.org/Beagleboard

Quelle: http://beagleboard.org/static/images/black_hardware_details.png

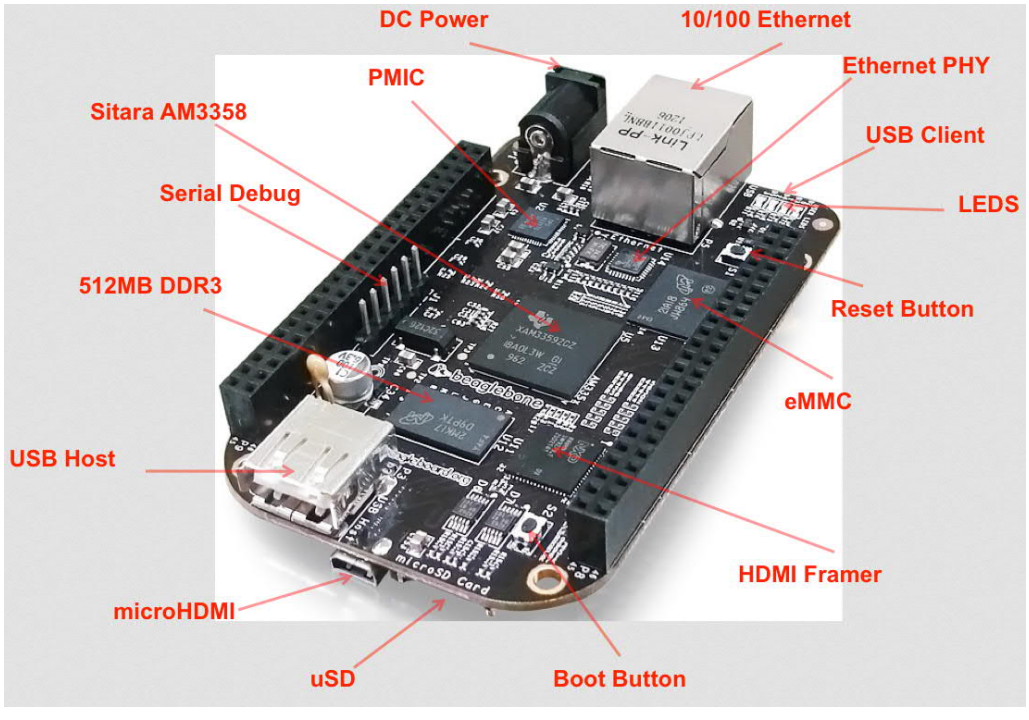
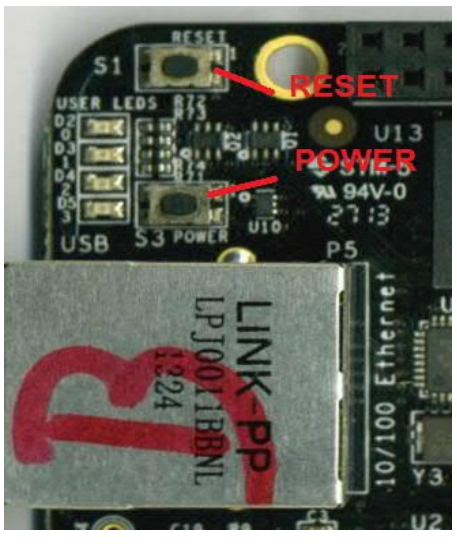


Bild links: BeagleBone Black, Revision C

Bild unten: Revision B, hier sind die Taster anders angeordnet.



Hard- und Software

aus http://www.pdp8.net/mfm/

Kurzbeschreibung Board:

Festplatte Emulieren:

```
J1 20pol. Daten HD1 | Emulation Festplatte – Hierzu den Festplattencontroller mit J1(J6) und J2
J2 34pol. Befehle   | verbinden.
J6 20pol. Daten HD2 |
```

Festplatte Einlesen oder Beschreiben(beta):

```
J3 20pol. Daten     | Festplatte einlesen – hierzu die Festplatten mit J3 und J4 verbinden
J4 34pol. Befehle   | Ein Beschreiben der Festplatte ist auch möglich, allerdings noch
                    | in Teststadium und bedarf einer Konfigurationsänderung mit Neustart.
```

Jumper P1 = WRITE – nur zum "Schreiben auf HD" stecken da sonst auch beim Einlesen Fehler auftreten können.

Jumper P9 = CAPS - Die Kondensatoren dienen als Notstromversorgung und werden bei gesetzten P9 geladen Beim Ausschalten wird die fehlenden Versorgungsspannung registriert und das Betriebssystem fährt runter. Bei eingeschalteter Versorgungsspannung wird das Betriebssystem (debian) automatisch gestartet. (den Autostart des Emulators habe ich abgestellt)

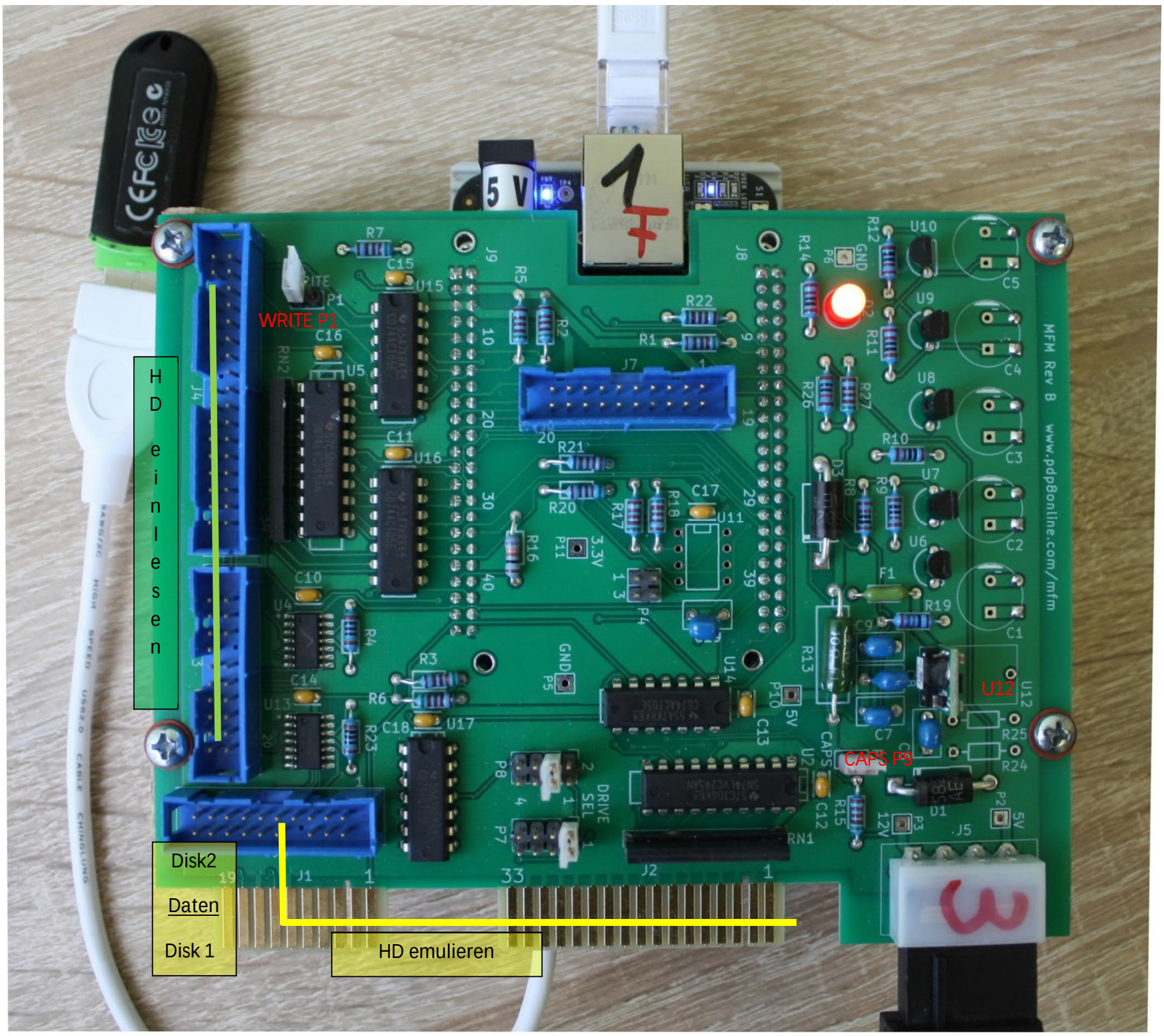
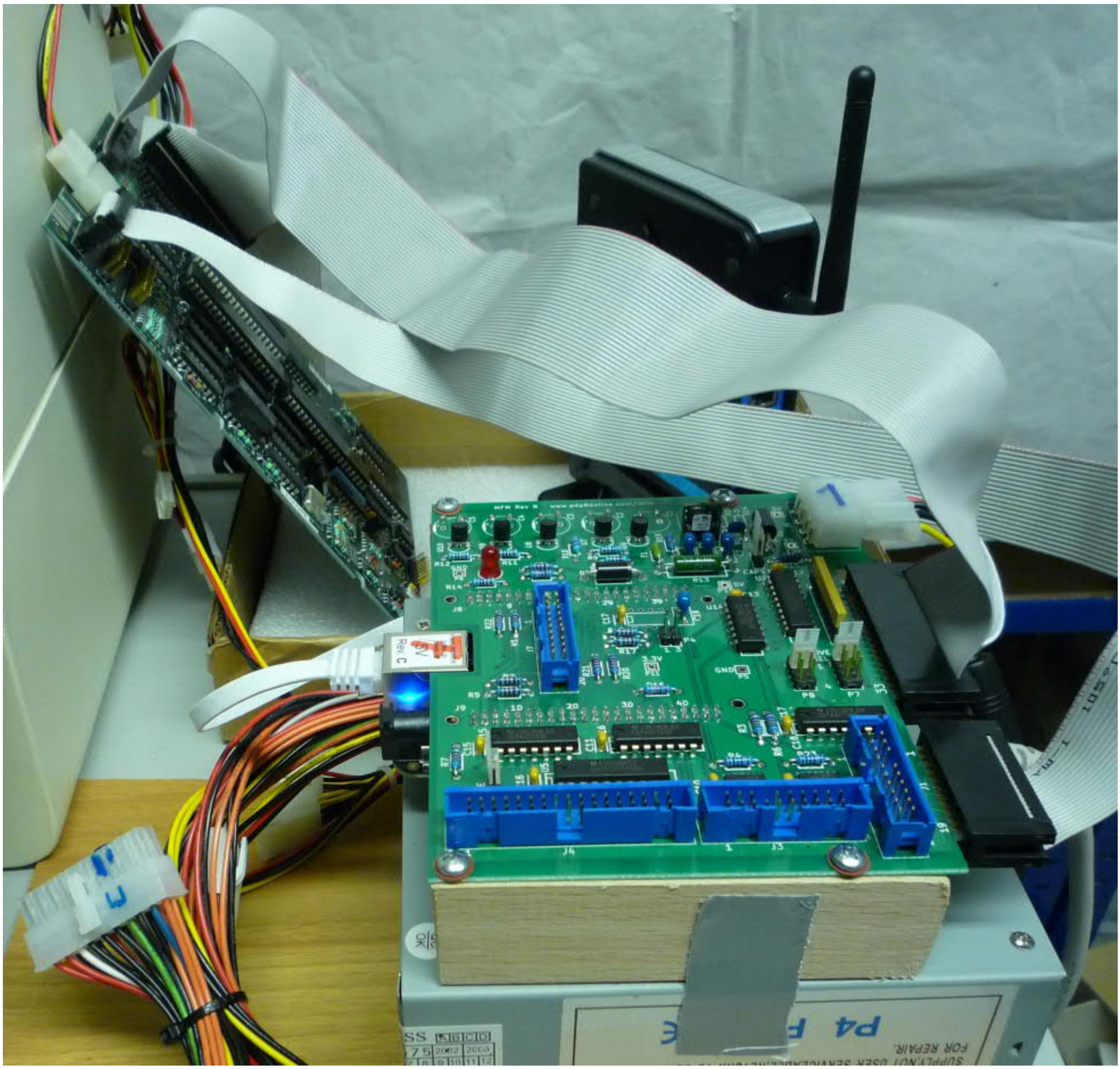


Bild aus einer HD Emulation mit NCR DMV (J9=CAPS ist hier nicht gesetzt, LED ist aus.)

Beschaltung des ATX-Netzteiles siehe https://www.elektronik-kompendium.de/sites/com/0601151.htm



Die Stromversorgung des BeagleBoard geht über den Spannungswandler U12 der 12V nach 5V wandelt. Bei gesetztem CAPS Jumper und leuchtender Diode D2 das BeagleBoard nicht von der MFM-Emu-Platine ab- oder anstecken. Bei defektem Spannungswandler (bei einem meiner MFM Boards war der Spannungswandler U12 defekt und wurde von mir getauscht) kann das BeagleBoard über seine eigene Stromversorgung betrieben werden.

OKR-T/1.5-W12-C

Enlarge

Images are for reference only

Mouser No:	580-OKR-T/1.5-W12-C
Mfr. No:	OKR-T/1.5-W12-C
Mfr.:	Murata Power Solutions
Customer No:	
Description:	Non-Isolated DC/DC Converters 12Vin, 0.591-6Vout 1.5A, Positive Logic
Datasheet:	OKR-T/1.5-W12-C Datasheet

Grundlegendes zum MFM-Emulator:

Ausführliches steht auf http://www.pdp8.net/mfm/. David Gesswein hat die Platine entwickelt und eine Miniserie gefertigt. Meine Version nutzt ein aktuelles Beagle Board Black mit 4GB 8-bit eMMC on-board "Flash Speicher" wogegen das ältere BBB nur 2GB "Flash Speicher" hatte.

Mit 4GB eMMC habe ich mich entschlossen, die Debian Version für Konsole aus dem Image von David um die XFCe Oberfläche zu erweitern. Zusammenbau, Installation und Testen beschreibt David ausführlich hier. Meine Installation beschreibe ich in einem eigenen Kapitel.

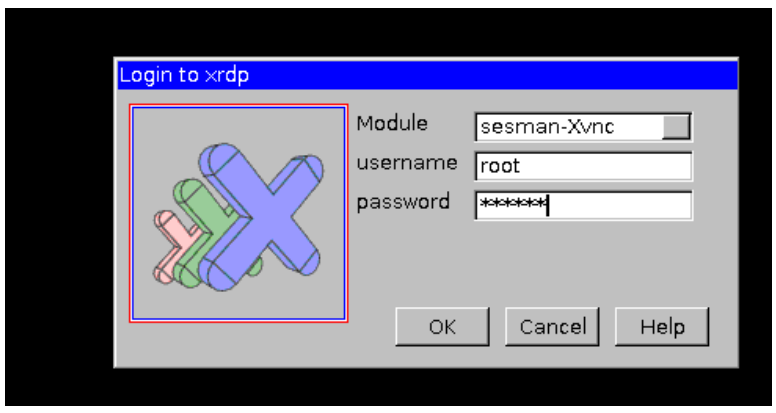
System starten und einloggen:

Kabel entsprechend der gewünschten Nutzungsart (Einlesen, Emulieren) anschließen, Netzwerkkabel anschließenu und Stromversorgung (12V) einschalten

Im Netzwerk zur MAC-Adresse (auf dem Bord aufgedruckt) die IP finden.

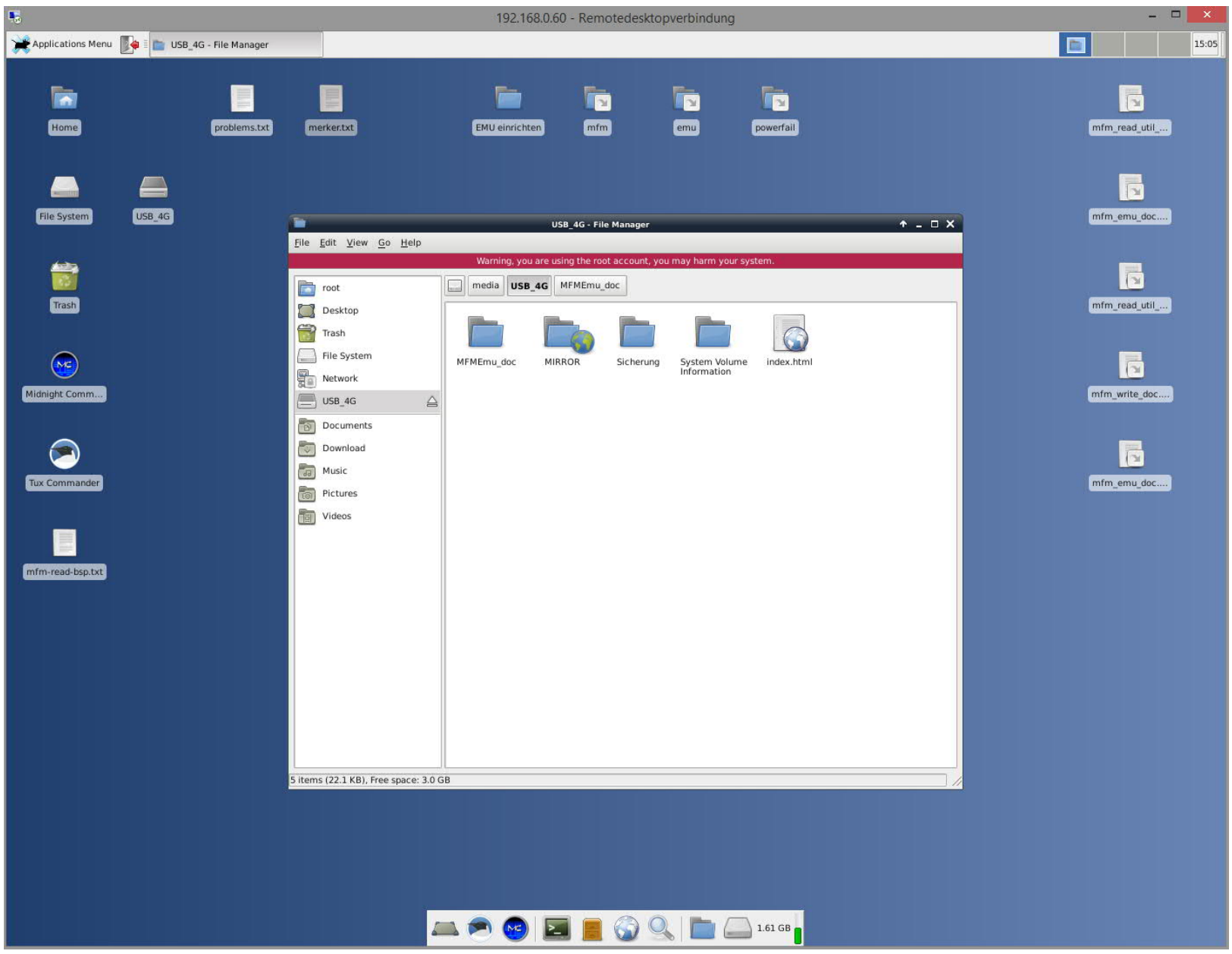
Verbinden an die entsprechende IP per Remote Desktop (RDP) oder mit z.B. PUTTY per SSH. Ich bevorzuge RDP da hiermit das Handling einfacher ist.

Aktuell bei mir als Remotedesktopverbindung zur IP 192.168.0.60 verbinden.



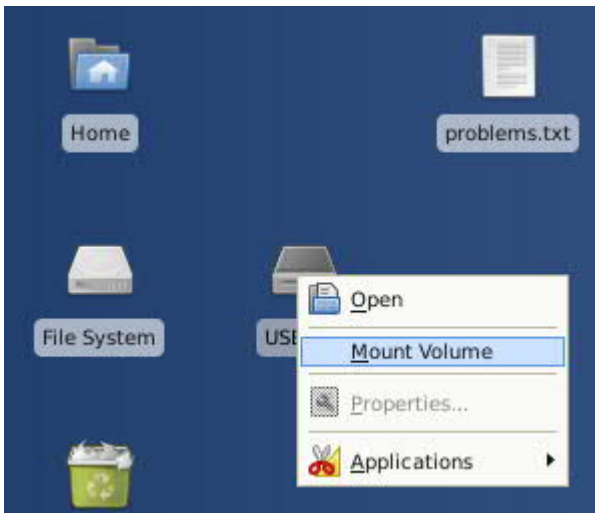
Beispielbilder

Der Cursor blinkt nicht !!



Login:
root | geheim   (mein default wegen remote Desktop login)

oder als User: debian | temppwd



Anschließend den USB Stick falls nicht gemounted manuell mounten.
Hier lassen sich die ausgelesenen Daten der Festplatte(n) speichern und dann auf den PC übertragen

Auf der nächsten Seite geht es weiter.

(Platz für Notizen)

Einlesen einer HD
(Beschreibung der Kommandozeilenbefehle in mfm_read_doc.)

Festplatte einlesen:

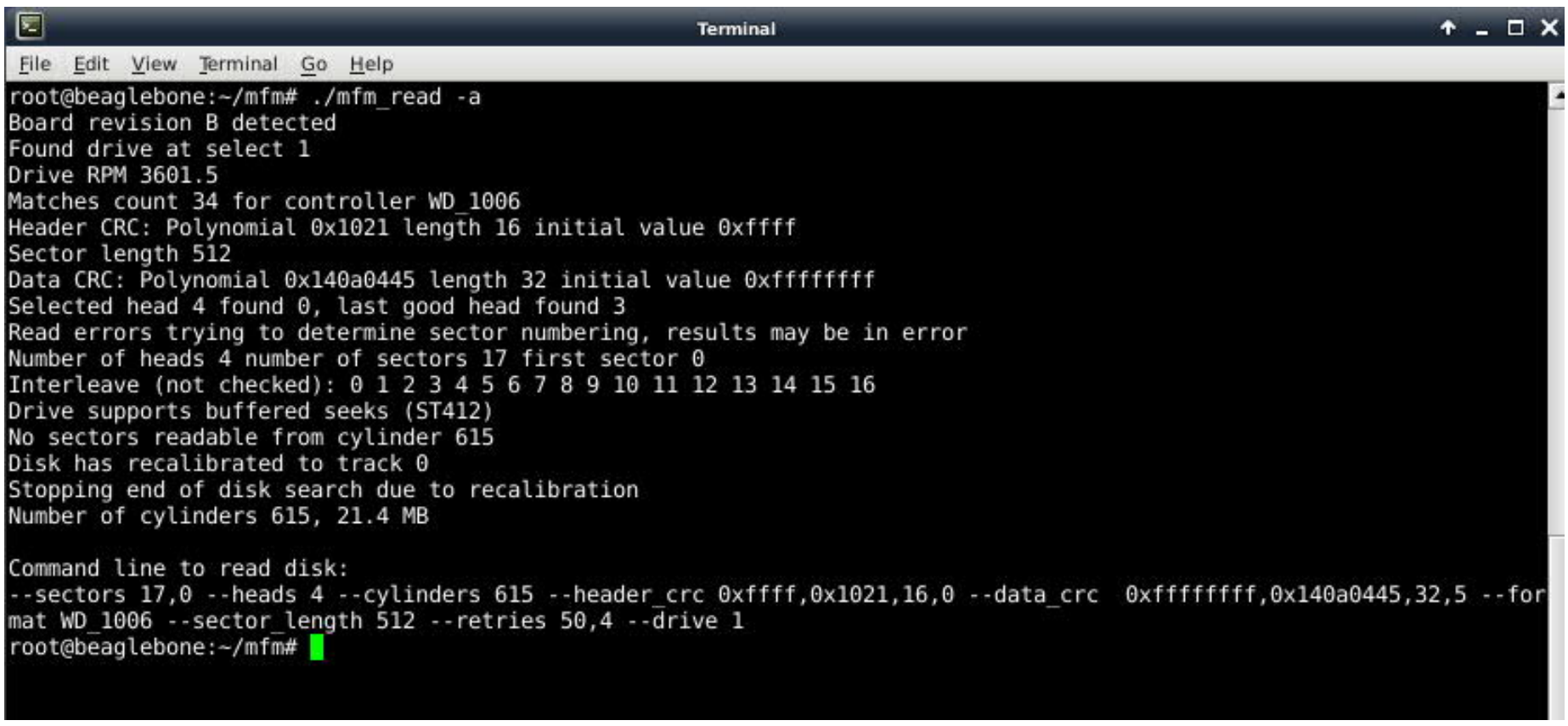
J3 20pol. Daten | Festplatte lesen – hierzu die Festplatten mit J3 und J4
J4 34pol. Befehle | verbinden.

Jumper P1 = Write abziehen.

1. Mit „cd mfm“ ins MFM-Verzeichniss wechseln.
2. Mit dem Befehl ./setup_mfm_read das Board konfigurieren. Es meldet bei mir
   als Bestätigung „Rev B Board“

3. Die Festplatte analysieren durch Eingabe von ./mfm_read –a und Feststellen der Parameter der
   angeschlossenen HD zur Kontrolle. Wird beim echten Einlesen nochmals durchgeführt.

   Hier MFM-HD von NCR DM-V CP/M-80



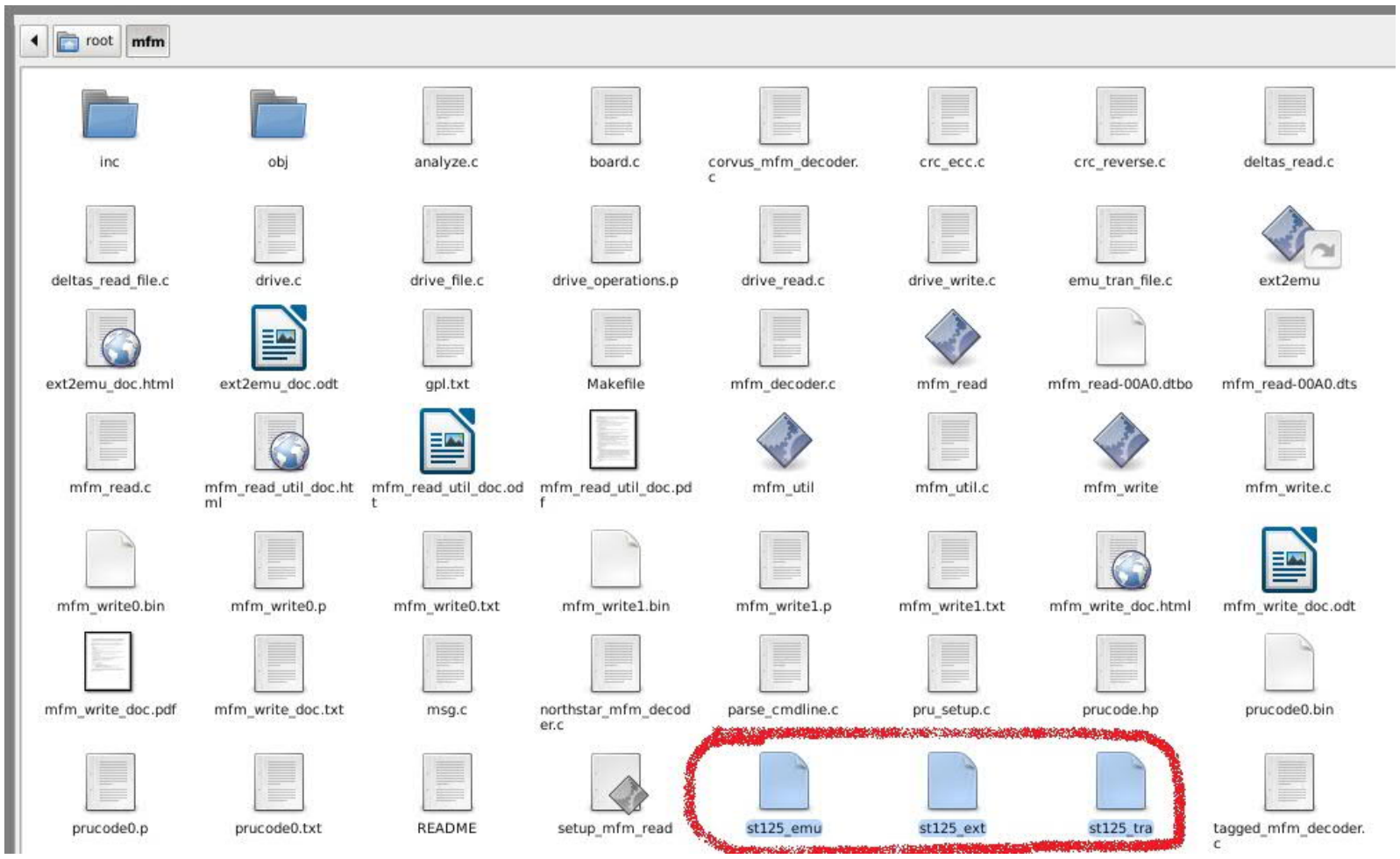
Beispiel: HD einlesen mit folgendem Kommando:

./mfm_read –a –e st125_ext –m st125_emu –t st125_tra –n "(Beschreibungstext)"

-e steht für „Extract“, d.h. das ist ein Hexdump
-m steht für „Emulator“, d.h. diese Datei ist zur Emulation der HD notwendig
-t steht für „Transitions“, also der rohe Bitstrom vom Controller.
-n steht für „Note”,   "st125_" ist hier nur beispielhaft gewählt.

Voraussetzung zum Extrahieren der Daten aus dem rohen Bitstrom ist daß der verwendete Festplattencontroller bekannt ist. Falls bei der Analyse MFM_READ –a Fehler angezeigt werden und die Analyse recht lange dauert ist aller Warscheinlichkeit nach der Festplattenkontroller unbekannt, womit die parameter zu –data_crc und –header_crc fehlen. Hier kann zur Archivierung nur mit –t „Transitions“ nur der rohe Datenstrom gelesen werden.   => siehe Unbekannter Festplattenkontroller

Nach dem Einlesen befinden sich die 3 Dateien im Verzeichnis mfm



Ausgabe während des Einlesens:

```
root@beaglebone:~/mfm# ./mfm_read -a -e st125_ext -m st125_emu -t st125_tra
Board revision B detected
Found drive at select 1
Drive RPM 3602.1
Matches count 34 for controller WD_1006
Header CRC: Polynomial 0x1021 length 16 initial value 0xffff
Sector length 512
Data CRC: Polynomial 0x140a0445 length 32 initial value 0xffffffff
Selected head 4 found 0, last good head found 3
Read errors trying to determine sector numbering, results may be in error
Number of heads 4 number of sectors 17 first sector 0
Interleave (not checked): 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Drive supports buffered seeks (ST412)
No sectors readable from cylinder 615
Disk has recalibrated to track 0
Stopping end of disk search due to recalibration
Number of cylinders 615, 21.4 MB
Command line to read disk:
--sectors 17,0 --heads 4 --cylinders 615 --header_crc 0xffff,0x1021,16,0 --data_crc
0xffffffff,0x140a0445,32,5 --format WD_1006 --sector_length 512 --retries 50,4 --drive 1

Retries failed cyl 51 head 1
Bad sectors on cylinder 51 head 1: 4
Retries failed cyl 53 head 1
Bad sectors on cylinder 53 head 1: 4
Retries failed cyl 54 head 1
Bad sectors on cylinder 54 head 1: 4
Retries failed cyl 55 head 1
Bad sectors on cylinder 55 head 1: 4
Found cyl 0 to 614, head 0 to 3, sector 0 to 16
Expected 41820 sectors got 41816 good sectors, 0 bad header, 4 bad data
0 sectors marked bad or spare
0 sectors corrected with ECC. Max bits in burst corrected 5
Track read time in ms min 27.931500 max 1684.372292 avg 44.853666

root@beaglebone:~/mfm#
```

Emulation

Emulation: (Beschreibung der Kommandozeilenbefehle in mfm_emu_doc.)

J1 20pol. Daten HD1 | Emulation Festplatte – Hierzu den Festplattencontroller mit J1(J6) und J2
J2 34pol. Befehle | verbinden.
J6 20pol. Daten HD2 |

Es wurden mit ./mfm_read –a –e TEST_ext –m TEST_emu –t TEST_tra –n "Test HD einlesen ST125" 3 Dateien geschrieben st125_ext, st125_emu und st125_tra.

st125_emu wird für die Emulation benötigt.

Entferne die Kabel zum Lesen eines Laufwerks, bevor ein Laufwerk emuliert werden soll.
Verbinde die Kabel vom Controller mit J1 und J2. (siehe Bild NCR DM-V)

Formaler Aufruf zur Emulation:

```
cd ~/emu
./setup_emu
./mfm_emu  --drive 1 --file ../emu_file
```

Im aktuellen Fall:

```
cd ~/emu
./setup_emu
./mfm_emu  --drive 1 --file ../st125_emu
```

und den Hostcomputer booten.

Falls keine Festplatte eingelesen wurde kann eine leere Emulationsdatei erstellt werden, welche anschließend mit Controller des Hostcomputers formatiert werden muß.

```
cd ~/emu
./setup_emu
./mfm_emu  --drive 1 --file ../emu_file --initialize --cylinders # --heads #

#für St125   cylinder 615 heads 4
./mfm_emu  --drive 1 --file ../emu_file --initialize --cylinders 615 --heads 4
```

Ersetze # mit den korrekten Werten für das zu emulierende Festplattenlaufwerk (Anzahl der Sektoren wird nicht benötigt da vorgegeben = 17)
Anschließend das Low Level Formatprogramm des Hostrechners, der mit dem MFM-Emulator verbunden ist, laufen lassen und die emulierte Festplatte formatieren. Es werden Informationen angezeigt und das Formatieren sollte ohne Fehler beendet werden.

Festplatte beschreiben

Festplatte beschreiben (in Arbeit): Beschreibung der Kommandozeilenbefehle in mfm_write_doc.)

Bitte bei Problemen auch die Informationen aus der e-mail von David lesen.

J3 20pol. Daten | Festplatte lesen – hierzu die Festplatten mit J3 und J4
J4 34pol. Befehle | verbinden.

Der Jumper P1 = Write muß gesteckt werden

Die Datei /mfm/mfm_read-00A0.dts ist für Schreiben und Lesen anzupassen. Nach dem Beschreiben der HD mfm_read-00A0.dts wieder für Lesen zurücksetzen und den Write-Jumper entfernen, dann Reboot.

Zum Schreiben ist in mfm_read-00A0.dts folgende Zeile auszukommentieren

```
0x190 0x07 // OUT P9_31 = gpio3_14
```

und bei folgender Zeile "//" löschen und dann erst setup_mfm_read ausführen

```
//0x190 0x2d // OUT P9_31 = pr1_pru0_pru_30_0
```

Festplatte mit emu_file beschreiben

```
cd ~/mfm
./setup_mfm

#test - das emu_file habe ich nach root kopiert

./mfm_write --emu /root/st125_emu
```

Die Datei /mfm/mfm_read-00A0.dts ist für Schreiben und lesen anzupassen. Nach dem Beschreiben der HD mfm_read-00A0.dts wieder für Lesen zurücksetzen und den Write-Jumper entfernen, dann Reboot und Festplatte zur Kontrolle als test_emu_file (Vorschlag) einlesen.

Defekte Sektoren

Das mfm_write Programm kann nicht auf defekte Sektoren testen und diese ausblenden - logisch da ja ein IMAGE zurückgeschrieben werden soll. Die zu beschreibende Festplatte also "vorher" auf defekte Sektoren testen. Entweder mit einlesen der HD mfm_read oder Format mit MSDOS und testen mit Spinrite oder ähnlichen Programmen.

(ENDE DOC)

Installation

INSTALLATION MFM Reader/Emulator mit XFCE und Remote Desktop auf Beagle Board Black 4GB eMMC

Debian System für MFM - EMU installieren
BBB flaschen aus https://www.pdp8.net/mfm/revb/revb_board_build.shtml das Betriebssystemimage sichern

Aktuelles Image ist http://www.pdp8.net/mfm/revb/BBB-mfm-emu_v1.17.img.xz wobei das IMAGE mit Endung .XZ eine komprimierte Datei ist. Entpacken unter Windows ist mit 7-Zip ab der Version 9 und WinRAR ab Version 5 möglich.
Das Image wird mit dem WIN32 DISKIMAGER sourceforge.net/projects/win32diskimager/files/latest/download auf eine 2 bis 4GB große Micro-SD-Karte geschrieben.
Anschließen die Micro-SD-Karte ins BBB einsetzen, USER SWITCH drücken und Stromversorgung einstecken.

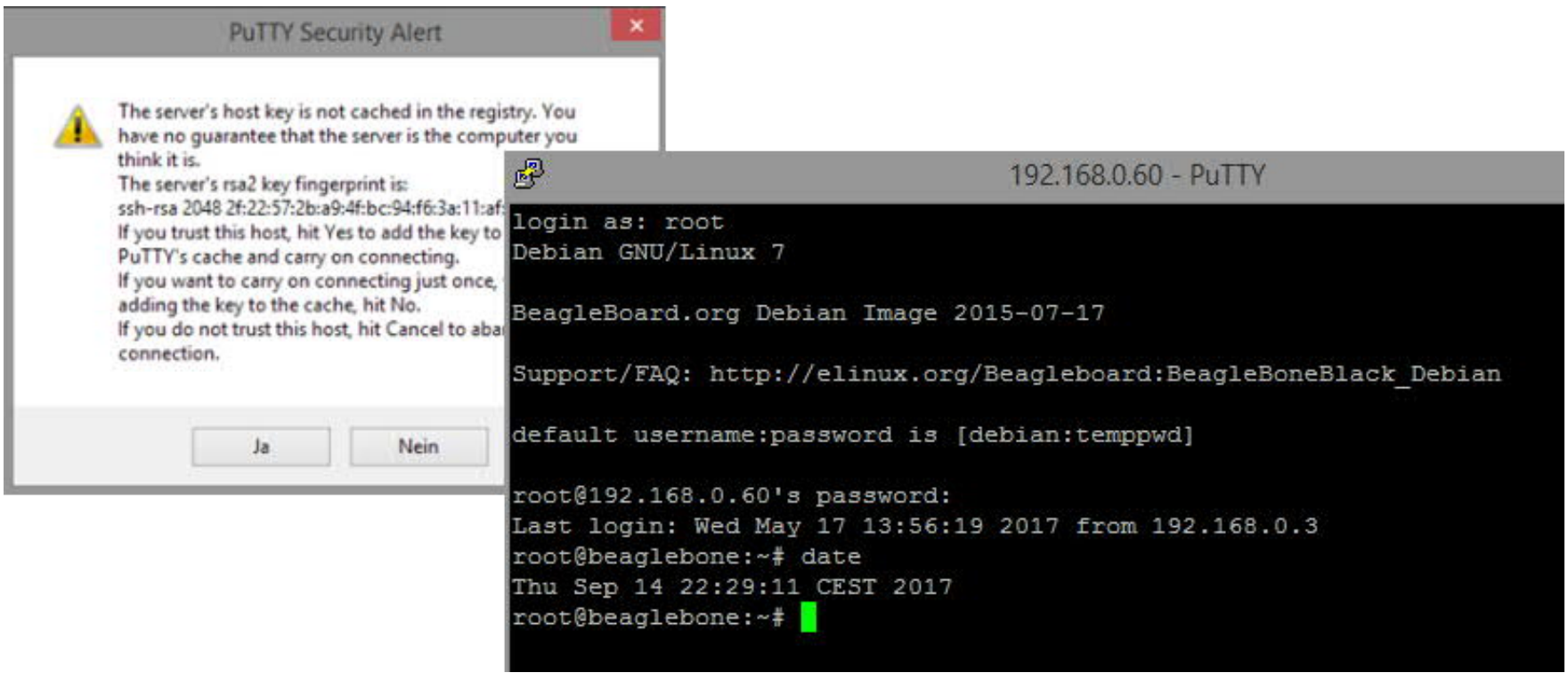
Notwendig ist die gerade erstellte Micro-SD-Karte mit dem zu flashendem Image des Betriebssystemes von David Gesswein

1. Die Stromversorgung der DC Buchse sollte mindestens 5V / 2A betragen. Alternativ kann auch ein USB-Adapter verwendet werden.
2. Trennen das Netzwerk(Ethernet)-Kabel und entfernen Sie alle sonstigen Stecker und USB-Peripheriegeräte
3. BBB ausschalten durch USB / Netzkabel physisch trennen.
4. Beschriebenen Micro-SD-Karte in den Micro-SD-Kartensteckplatz der BBB stecken
5. Halte die Boot-Taste (S2) oben rechts (in der Nähe des SD-Kartensteckplatzes) gedrückt, und während diese Taste gedrückt gehalten wird das USB / Netzkabel einstecke um die Stromversorgung einzuschalten. Halte die Taste gedrückt, bis die LEDs zu blinken beginnen. Die blauen On-Board-LEDs sollten nacheinander leuchten und dann für die nächsten 5-25 Minuten weiter blinken (abhängig von der Größe und der Geschwindigkeit der SD-Karte).
6. Warte bis die LEDs aufhören zu blinken und alle 4 LEDs leuchten. Dies kann je nach verwendetem Image 5-25 Minuten dauern. Wenn der Flashvorgang fehlschlägt - z. B. blinken keine LEDs, oder es dauert länger als 45 Minuten - dann trenne die Stromversorgung und versuchen die BBB mit der Taste S2 zu resetten.
7. Entferne die Micro-SD-Karte damit kein neuer automatischer Flash-Vorgang gestartet wird
8. Schließlich drücke den Einschalter (S3), um die Karte einzuschalten und das gerade installierte System sollte starten.

[Anleitung aus WWW im Original in englischer Sprache]

User einrichten

Das BBB ist gestartet und per Netzwerk erreichbar. IP finden -> die MAC Adresse steht seitlich auf dem BBB und zum Zugriff über USB-RS232 oder IP nutze ich PUTTY. Die IP wird meist per DHCP über Router, Firewall oder Accespoint vergeben. Beim ersten Zugriff mit PUTTY kommt eine Warnmeldung zum RSA Fingerprint - diesen bestätigen.
Und schon ist man auf der Console.



Dem ROOT ein Passwort zuweisen damit Login per Remote Desktop geht

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

login as: root

```
root@beaglebone:~# passwd
Enter new UNIX password: geheim
Retype new UNIX password: geheim
passwd: password updated successfully
root@beaglebone:~#
```

X--X

System aktualisieren

```
apt-get update
apt-get upgrade
apt-get install mc
```

Damit die Dateien per Update per "sudo apt-get update" geladen werden
in /etc/apt/apt.conf.d/o2compress-indexes
gzip auf false stellen
Acquire::GzipIndexes "true"; Acquire::CompressionTypes::Order:: "gz";
nach Acquire::GzipIndexes "false"; Acquire::CompressionTypes::Order:: "gz";
ändern.

Jetzt das System updaten (keinen neueren kernel einspielen!)

```
apt-get update      damit wird die  Paket-Datenbank aktualisiert
apt-get upgrade     damit werden die installierten Pakete aktualisiert
```

Das BBB hat leider keinen Accu oder Batterie und behält die Systemzeit nicht, weshalb die Zeit und das Datum beim Start per NTP gesetzt werden müssen.

NTP

```
root@beaglebone: apt-get install ntp
```

Anschließend gewünschten ntp-server in /etc/ntp.conf
eintragen. Ich habe local unter 192.168.0.1 einen NTP Server

X--X

```
# You do need to talk to an NTP server or two (or three).
#server ntp.your-provider.example
# localer server von fritz

server 192.168.0.1

# pool.ntp.org maps to about 1000 low-stratum NTP servers.  Your server will
# pick a different set every time it starts up.  Please consider joining the
# pool: <http://www.pool.ntp.org/join.html>
server 0.debian.pool.ntp.org iburst
server 1.debian.pool.ntp.org iburst
server 2.debian.pool.ntp.org iburst
server 3.debian.pool.ntp.org iburst
```

X--X

Manuell ntp neu starten

```
root@beaglebone:/etc/init.d# sh ntp stop
```

```
[ ok ] Stopping ntp (via systemctl): ntp.service.
root@beaglebone:/etc/init.d# sh ntp start

[ ok ] Starting ntp (via systemctl): ntp.service.

root@beaglebone:/etc/init.d#
root@beaglebone:/etc/init.d# ntpd  -g   [justiere einen großen Zeitfehler]
root@beaglebone:/etc/init.d# date
Sat Mar  4 12:08:47 UTC 2017

root@beaglebone:/etc/init.d#
X-----------------------------------------------------------------X
```

XFCE installieren

Für angenehme Benutzung XFCE und XRDP installieren
-> https://wiki.debian.org/Xfce

```
root@beaglebone:/apt-get install xfce4
irgendwann ist xfce fertig.

apt-get install xfce4-goodies
apt-get install tuxcmd
apt-get install tuxcmd-modules
apt-get install xfce4-terminal
apt-get install leafpad

nur bei 4GB FLASH - die ältere BBB Version hat nur 2GB
apt-get install libreoffice-writer

apt-get install xrdp   (für den grafischen Zugang)

dann ein REBOOT

Es dauert eine Minute bis XFCE4 eingerichtet ist und das Bild aufgebaut wird. Die DEFAULT Einrichtung
bitte bestätigen.
Ausloggen und als ROOT neu einloggen.

user: ROOT
pw: geheim

#WWW browser wählen:
sudo apt-get install midori

vielleicht auch

sudo apt-get install netsurf

#Bilderbetrachter
sudo apt-get install Ristretto

#PDF Viewer
sudo apt-get install xpdf


(Ende Installation)
```

Unbekannter Festplattenkontroller:

Im Juni 2018 hatte ich eine Festplatte zum Nixdorf 8870 Quattro/7 welche gesichert werden sollte. Das Lesen der Festplatte wurde leider nach längerem leseversuch mit einer Fehlermeldung beendet.

Befehl:

```
mfm_read --analyze --transitions_file raw_data --extracted_data_file extracted_data test
         --note "irgendetwas"
```

Mit obigen Kommando versucht die Software aus den Rohdaten der Flußwechsel entsprechnede binaere Daten zu gewinnen.

Das Format des verwendeten Nixdorf Harddiskkontrollers ist leider nicht kompatibel zu den im Programm vorhadenen Kontrollern und es erfolgte eine Fehlermeldung die allerdings nicht direkt Rückschlüsse auf das Problem gab.

```
*---------*
| Lösung: |
*---------*
```

Einlesen der Rohdaten zur Sicherung:

```
./mfm_read --transitions_file raw_data  --cylinders # --heads # --drive #
```

Die Informationen um aus den Rohdaten decoded und extracted Daten zu machen bedarf es der Analyse der --header_crc und --data_crc durch David Gesswein djg@pdp8online.com

MFM_EMU.DOC

http://www.pdp8.net/mfm/code/emu/mfm_emu_doc.html

mfm_emu emulates a MFM disk drive. It can use a data file created from a real drive by mfm_read or can create a empty emulation file.
--begin_time -b #
The number of nanoseconds to delay from index to start reading track only needed if initialize specified. Default is zero if not specified.
--cylinders -c #
The number of cylinders. Only needed if initialize specified.
--drive -d #[,#]
Drive number to emulate. For revision B boards specify 0 or 1 or 1,2. Use 0 for drive to always be selected (radial select). 1 and 2 select the first and second jumper block to use for drive select. One or two drive numbers may be specified to emulate one or two drives. Only one may be specified if drive number is 0. For revision A boards specify the drive number to emulate 1-4.
--file -f filename[,filename]
Emulation filename. First filename corresponds to first drive number specified.
--heads -h #
The number of heads. Only needed if initialize specified.
--initialize -i
If given create/overwrite specified file with empty data. Heads and cylinders must be specified.
--note -n “string”
Description to store in the emulation file. Only used if initialize specified.
--options -o “string”
Options for mfm_util in decoding such as –data_crc. This saves having to type each time you wish to decode file. Only used if initialize specified and not required. No validation is performed so use mfm_util to verify file is valid after creating.
--pool -p #,#.#
The first parameter is the size of the of track buffer pool to use. and the second is the maximum delay. The delay will increase linearly as the buffers fill to the maximum delay. Default is 75,0.6
--quiet -q #h
Bit mask to select which messages don't print. 0 prints all messages. Default is 1 (no debug messages). Higher bits are more important messages in general.
--rate -r #
Bit rate in Hz for the MFM clock and data bits. Only needed if initialize specified. Default is 10,000,000. if not specified.
--version -v
Print program version number.
Long options can be abbreviated to the shortest unique name. Long option values can't have spaces.
is a number. #h is a number which may be decimal, octal if starts with a 0, or hex starting with 0x.

The buffer pool option controls buffering used to prevent controller timeouts when writing data to the file. Writing to flash has large delays at times which the buffers hide. The maximum delay should be set shorter than the controller timeout for a seek. The maximum number of buffers needs to be low enough that they can be written out before the holdup capacitors are drained.
Begin_time is for drives that have a sector straddle the start of the index pulse. For emulation to work properly all the data must be read consecutively. Set this parameter to the time in nanoseconds the first

physical sector is delayed from the index pulse. It is needed for Corvus model H and NorthStar Advantage drives.
When this program is run it appends to logfile.txt in the current directory. It logs when it started, stopped, how long it was executing, maximum seek time, minimum free buffers, and how many seeks and writes were done. The shutdown time is from when the program was told to shut down to the emulation file closed and written to storage. The operating system will take about 5 more seconds to shut down.
If the log file minimum free buffers is zero you may wish to either increase the number of buffers or maximum delay. If the log file shutdown time is close to capacitor holdup time or not all the message were written to the log file the number of buffers should be decreased or wait 45 seconds after significant write activity before powering off the MFM emulator.

Example:
     mfm_emu –drive 1 –file disk_file
This used file disk_file to emulate a drive on select line 1.
    mfm_emu –drive 1 –file disk_file –initialize –cylinders 306 -heads 4 –note “Games disk” --options “--sectors 17,0 --heads 6 --cylinders 640 --header_crc 0x2605fb9c,0x104c981,32,0 --data_crc 0xd4d7ca20,0x104c981,32,0 --format OMTI_5510”
This creates/overwrites file disk_file with empty data for a disk with 306 cylinders and 4 heads. It then emulates a drive on select line 1. You will need to use the host computer low level format program to write the sector headers before the emulated drive can be read or written normally.
Currently the program prints various stuff to see what it's doing. This will change as testing goes on.
bad pattern count 0
Read queue underrun 0
Write queue overrun 0
Ecapture overrun 0
glitch count 0
glitch value 0
0:test 0 0
0:test 1 0
0:test 2 0
0:test 3 0
0:test 4 0
1:test 0 0
1:test 1 0
1:test 2 0
1:test 3 0
1:test 4 0
    The named values are various errors/unexpected conditions. The test values are used to show internal variables from the PRU's. See the source for current purpose.
select 0 head 0
   Current select and head line state
Waiting, seek time 3.9 ms max 3.9

   IARM is waiting for PRU. The values are the last and maximum time from PRU requesting the next cylinder to the data being returned. If seek time is zero you are using a buggy version of am335x_pru_package and data is likely to be corrupted.
Cyl 0,400 select 1, head 4 dirty 0
   Last and next requested cylinder, select and head lines. Dirty 1 indicates the sector data was written to.
I have found that if unrecoverable read errors occur operating the drive in a different orientation may allow the data to be read. I have also had some luck especially on Seagate drives with pushing on the shaft of the head stepper motor while it is retrying. This seems to get the head at a slightly different position where it can read the data. This has a risk of drive damage so make sure you have read as much as you can before trying this.

################################
Starting Emulation on Power On
################################

Using my prebuilt image the default when enabled is to emulate a single drive from /root/emufile_a. The emulation file will not be backed up on boot. If you want to start the emulator at boot with these options execute the following command

systemctl enable mfm_emu.service

If you wish to change the options edit /etc/mfm_emu.conf. The file has comments describing what the configuration variables do. For example if you wish to emulate two drives set EmuFN2 to the second file name. If your emulated drive has information you wish not to lose you may wish to enable backup. Set the Backup variable to the type of backup. Copy just copies the emulator file. rdiff and xdelta do a binary difference between the files to take less space. If you do something that changes most of the image file such as defragment the binary difference may take long enough for your computer to timeout. The straight copy is quicker but for small changes will take much more space. I didn't find a clear winner between rdiff and xdelta.
It seems to take about 12 seconds from power on until the mfm emulator is running if no backup is performed.(ENDE DOC)

MFM_READ, MFM_UTIL, MFM_WRITE Manual

(http://www.pdp8.net/mfm/code/mfm/mfm_read_util_doc.html)

mfm_read can analyze and/or read a MFM disk and write raw MFM transition data, decoded data, or emulation data to files.
mfm_util can read transition and convert to emulation or decoded data. It can also read emulation file data and convert to decoded data.
mfm_read and mfm_util both use similar command options.

--analyze -a [=cyl,head]
Analyze disk format. If cylinder and head is specified it will analyze the specified track for header format. No spaces can be in the optional parameters. --analyze=10,4
--begin_time -b #
The number of nanoseconds to delay from index to start reading track
--cylinders -c #
The number of cylinders.
--data_crc -j #h,#h,#h[,#h]
The CRC/ECC parameters for the sector data area. Initial value, polynomial, polynomial length, maximum ECC span.
--drive -d #
Drive number to select for reading. Only valid for read command. Drives are number 1 to 4.
--emulation_file -m filename
File name to write emulation bit data to. No file created if not specified
--extracted_data_file -e filename
File name to write decoded data to. No file created if not specified. If drive has metadata for the sectors it will be extracted to filename.metadata. Currently only Xerox 6085 has metadata
--format -f WD_1006 | OMTI_5510 | DEC_RQDX3 | Xebec_104786
The track format. Not all formats listed. Use --format help to list all currently supported formats. Some formats will also set header_crc, data_crc, sectors, and sector_length.
Parameters specified after format will override values.
--head_3bit -3
Selects header 3 bit head encoding used by WD 1003 controller. Default is 4 bit. This will not be detected by analyze. The wrong number of heads may be selected.
--header_crc -g #h,#h,#h[,#h]
The CRC/ECC parameters for the sector header. Initial value, polynomial, polynomial length, maximum ECC span.
--heads -h #
The number of heads.
--interleave -i # | #,#,#,#,...
The logical sector numbers from header in physical sector order or the interleave value
--note -n “string”
String is stored in header of transition and emulation file for information about image. mfm_util will display.
--retries -r #[,#]
Select number of retries on read errors. Only valid for read command. The optional second parameter specifies the number of retries before seeking off track and back. Default 50,4.
--sector_length -l #
The sector data area length in bytes. Default is 512.
--sectors -s #[,#]
The number of sectors per track, lowest sector number.
--track_words -w #
The number of 32 bit words of track data to use for emulator file. If this parameter needs to be set the messages during generation of the emulator file will specify the value to use.
--transitions_file -t filename
File name to write raw MFM transitions to. No file created if not specified. Only valid for read command.
--quiet -q #h
Bit mask to select which messages don't print. 0 is print all messages. Default is 1 (no debug messages). Higher bits are more important messages in general.
--unbuffered_seek -u
Use unbuffered/ST506 seeks. Default is buffered/ST412.
--version -v
Print program version number.
Long options can be abbreviated to the shortest unique name. Option values can't have spaces unless quoted as a string.

is a number. #h is a number which may be decimal, octal if starts with a 0, or hex starting with 0x. The Cyclic Redundancy Check (CRC) / Error Correcting Code (ECC) parameters consists of a initial value which the CRC register is set to before starting the CRC, the CRC bit length, a CRC polynomial, and a maximum ECC span. The ECC span should be zero if ECC correction should not be used.

It is advisable to generate a transition file when reading important disks since it may be possible to reprocess in the case of errors either in reading or due to errors in these programs.
32 bit or longer polynomials may be usable for ECC even if the original controller only used them for CRC. The quality of the polynomial chosen determines the miss-correction probability. Most 32 bit polynomials specify 5 bit correction though some say they can be used for up to 11 bit correction.
Emulation file is for use by the mfm_emu program to emulate a disk drive. For mfm_read it is an output. For mfm_util it is an output if a transitions file is specified otherwise it is an input.
If extract file is specified mfm_read will attempt to decode the track data. If the disk format is known an extract file should be specified even if one is not needed since mfm_read will then reread tracks that have errors until it exceeds the error count or gets a successful read of the track. The extract format is the sector data from the drive written in logical sector order. If controllers use alternate cylinders the data for those cylinders is not moved to the logical cylinder location.
Begin_time is for drives that have a sector straddle the start of the index pulse. For reading to work properly all the data must be read consecutively. Set this parameter to the time in nanoseconds the first physical sector is delayed from the index pulse. It is needed for Corvus model H and NorthStar Advantage drives.
NOTE: Some computers use different format on different tracks. Analyze can only handle one format at a time. This may cause it to determine the wrong number of cylinders or other parameters. The published specification for the drive should be checked against what analyze determine and the command arguments adjusted as needed.
For retrying read errors on some drives seeking helps to recover the error since the heads end up at a slightly different location. On a drive that is unhappy the extra seeks bad be bad. It may be advisable to do the first read with –retries 50,51 to prevent any seeks and then go back to the default or adjust the parameters to try to recover more data.
The Adaptec format uses logical block addresses (LBA) instead of cylinder head sector (CHS) in the header. This program does not have enough information to determine the CHS for the LBA in the header so if a seek error occurs it can not recover. Messages like Missing LBA address # to # where the second number is less than the first or large range may indicate a seek error. Some disks have a large jump near the end of the disk due to the spare and bad sector handling.

Examples:
To read an unknown format drive
   mfm_read --analyze --transitions_file raw_data --extracted_data_file extracted_data –note “Drive from TI Professional computer read November 7 2014”
Analyze is conservative on maximum ECC burst length for correction. You may rerun with the command line printed and change the header or data _crc last parameter to increase ECC span with higher probability of miss correction. If you expected sector doesn't match sector found on some of the tracks the drive may have different interleave on some tracks. Try rerunning with the command parameters printed except leave off –interleave.

To store raw transitions without decoding (change parameters to match your drive)
   mfm_read --transitions_file raw_data --drive 1 --heads 6 --cylinders 640
To process previously read raw transitions data if analyze determined decoding parameters or they were manually specifed when file created:
   mfm_util --transitions_file raw_out2 --extracted_data_file /tmp/decoded_out
Otherwise specify parameters for your drive like:
   mfm_util --sectors 17,0 --heads 6 --cylinders 640 --header_crc 0x2605fb9c,0x104c981,32,0 --data_crc 0xd4d7ca20,0x104c981,32,0 --format OMTI_5510 --sector_length 512 --interleave 0,3,6,9,12,15,1,4,7,10,13,16,2,5,8,11,14 --transitions_file raw_out2 --extracted_data_file /tmp/decoded_out
--emulation_file may be specified instead of --transitions_file to convert emulation file to extracted data.

Analysis typical messages. Explanation in italic:
AM33XX
File prucode0.bin open passed
   Informational messages from PRU access routines.
Found drive at select 2
   Informational. Select line drive responded to.

```
Returning to track 0
   Informational. This operation may take a while.
Drive RPM 3594.4
   Informational. Drive should be close to 3600 RPM
Matches count 34 for controller OMTI_5510


Header CRC: Polynomial 0x104c981 length 32 initial value 0x2605fb9c
Sector length 512
Data CRC: Polynomial 0x104c981 length 32 initial value 0xd4d7ca20
Number of heads 6 number of sectors 17 first sector 0
Informational. What we found about the disk format. Sometimes multiple formats will be shown. The
code will retry on a different cylinder to try to determine correct format. The  matches can either be
false matches or some drives  such as DEC RQDX3 use multiple formats. This code does not deal
well with that. You can manually read with the different formats and put the data back together.
Interleave (not checked): 0 3 6 9 12 15 1 4 7 10 13 16 2 5 8 11 14
Informational. If it can't determine interleave the disk can still be read and decoded. Interleave is
only checked if interleave is specified on the command line. Too many drives have tracks with
different interleave which generated confusing messages.
Drive supports buffered seeks (ST412)
    Informational.
No sectors readable from cylinder 640
Stopping end of disk search due to two unreadable tracks in a row
Number of cylinders 640, 33.4 MB
    Informational.  The method we used to determine the number of cylinders and size determined.
Command line to read disk:
--sectors 17,0 --heads 6 --cylinders 640 --header_crc 0x2605fb9c,0x104c981,32,0 --data_crc
0xd4d7ca20,0x104c981,32,0 --format OMTI_5510 --sector_length 512 --retries 50 --drive 2
--interleave 0,3,6,9,12,15,1,4,7,10,13,16,2,5,8,11,14
This is the options needed to decode the disk with mfm_read. For mfm_util remove --retries and
--drive from options. Change header_crc and/or data_crc last parameter if you wish to use different
ECC maximum span. Remove interleave if you get expected sector doesn't match sector found errors.
It is recommended to verify results of analysis since it can make mistakes.
Decoding messages:
AM33XX
File prucode0.bin open passed
   Informational messages from PRU access routines in mfm_read.
Returning to track 0
   Informational. This operation may take a while. Only mfm_read.
Retries failed cyl 22 head 0
   Unable to recover all data from the specified track. Only mfm_read.
Bad sectors on cylinder 22 head 0: 3 15H
Indicates that for the specified track that data of sector 3 has uncorrected errors and the header for
sector 15 has errors. If the header has an error the data will be all zero.
ECC Corrections on cylinder 56 head 5: 1(2) 13(4) 15(1H)
Informational. Indicates that for the specified track that data of sectors 1 and 13 were corrected and
the header of sector 15 corrected. The number in the parenthesis is the length of the bit pattern
corrected. Unless the ECC correction had a false correction the data is good.
Cyl 59 head 2 Missed sector between 12(4) and 1(6)
 Informational. Indicates that the sectors found didn't match the interleave specified. The first
number is the sector header number and the number in parenthesis the physical sector starting with
0. A bad sector message will be printed if the data is bad. Some disks vary the interleave. Try again
without the interleave option.
Missing LBA address 2576 to 2579


 Indicates some LBA addresses in the single value or range specified were not found. The data for
those sectors was not recovered. Some drives have an intentional jump in the LBA address near the
end of the disk when sectors marked bad or spare reported is non zero.
Good data after 20 Retries cyl 219 head 0
   Informational. Indicates we recovered good data by retrying the read.
Found cyl 0 to 639, head 0 to 5, sector 0 to 16
   Informational. Should match what was specified
Expected 65280 sectors got 65276 good sectors, 0 bad header, 4 bad data
0 sectors marked bad or spare
```

11 sectors corrected with ECC. Max bits in burst corrected 5
Summary of errors during the read. We have 4 sectors with bad data. Sectors marked bad or spare are one the drive format has either marked bad to not use or reserving for spare sectors.
Errors:
Command 4 fault 300 status 1002c
Not Write fault
Not Seek complete
Not Index
Ready
Drive selected
Not Track 0
This indicates a command failed. See cmd.h for definitions. Status is the drive status bits which are decoded in text below. This error was the drive did not give seek complete in the expected time. These are normally fatal and the program will exit. Some are recovered from during analysis and are informational

(Ende DOC)

MFM_WRITE_DOC

http://www.pdp8.net/mfm/code/mfm/mfm_write_doc.html

mfm_write can write an emulator file to a disk drive. I got it to the state I could write the disk I needed but it is not a finished program. Currently it can only write an entire disk at once. You need to edit mfm_write.c main routine to set the parameters for your drive. I have not fixed command line parsing.

These command line options will be supported at some time along with options for setting the write precompensation. Currently only emulation_file, version, and quiet work. Emulation_file must be specified.

```
--begin_time -b #
  The number of nanoseconds to delay from index to start reading track

--cylinders -c  #
  The number of cylinders.

--drive -d #
  Drive number to select for reading. Only valid for read command. Drives are number 1 to 4.

--emulation_file -m filename
  File name to write emulation bit data to. No file created if not specified

--heads -h #
  The number of heads.

--quiet -q #
  Bit mask to select which messages don't print. 0 is print all messages. Default is 1 (no debug
  messages). Higher bits are more important messages in general.

--unbuffered_seek –u
  Use unbuffered/ST506 seeks. Default is buffered/ST412.

--version –v
  Print program version number.
```

To work mfm_read-00A0.dts line 0x190 0x07 // OUT P9_31 = gpio3_14 needs to be commented out and this line uncommented before running setup_mfm_read: //0x190 0x2d // OUT P9_31 = pr1_pru0_pru_30_0

Use mfm_read to verify the disk is properly written. The first attempt had a couple tracks that seem to be written to the wrong head. The next run worked ok. This program does not do anything to avoid using bad locations on the disk.

Ausschnitt aus mfm_read-00A0.dts:

```
  // All inputs pullup

                      // All outputs fast pullup disabled
                       // These go to PRU0
                      0x034 0x06 // OUT P8_11 = pr1_pru0_pru_30_15
                      0x03c 0x35 // IN/OUT  P8_15 = pr1_ecap0
                      0x184 0x36 // IN  P9_24 = pr1_pru0_pru_31_16
                      0x1ac 0x2d // OUT P9_25 = pr1_pru0_pru_30_7
                      0x1a4 0x36 // IN  P9_27 = pr1_pru0_pru_31_5
                      0x19c 0x36 // IN  P9_28 = pr1_pru0_pru_31_3
                      0x194 0x36 // IN  P9_29 = pr1_pru0_pru_31_1
                      0x198 0x36 // IN  P9_30 = pr1_pru0_pru_31_2
               !      //Until we support write make pin GPIO
               !      0x190 0x07 // OUT P9_31 = gpio3_14   [ for write hd comment out ]
               !      //0x190 0x2d // OUT P9_31 = pr1_pru0_pru_30_0  [ for write uncomment  ]
                      0x1a8 0x2d // OUT P9_41 = pr1_pru0_pru_30_6
                      0x1a0 0x36 // IN  P9_42.1 = pr1_pru0_pru_31_4
```

(Ende DOC)

Hinweise von David:

Betreff: Re: Write image with MFM Reader/Emulator to a second Harddiskdrive
Von: David Gesswein <djg@pdp8online.com>

If you have an emulator image you are wanting to write to another drive
it is likely to work. A recent release fixed picking up the drive
geometry from the emulator file. The pre compensation and drive to
write to are still hard coded in the source and will need to be set
for your drive.

```
drive_params.write_precomp_cyl = 512;
drive_params.early_precomp_ns = 10;
drive_params.late_precomp_ns = 10;
drive_params.drive = 1;
```

(bezogen auf ICL format)
Looks like that format uses begin_time. I haven't tested that it works. If
it doesn't I can fix it.
Rereading with the emulator will give some indication the write is good though
that won't guaranteed if will work on the real computer. I think you will
be the first person to use this code other than me. You will need to
reboot after writing before a read will work. Also pull the write jumper
before the reboot (and install before trying to write).
If you are needed to create a disk from a extracted data file the format
info would need to be added to inc/mfm_decoder.h
--
#########

Betref: Re: Write image with MFM Reader/Emulator to a second Harddiskdrive
Von: David Gesswein <djg@pdp8online.com>
Datum: 07.09.2017 04:04

If you are needed to create a disk from a extracted data file the format
info would need to be added to inc/mfm_decoder.h
If one of the controllers for the systems you have to test with
use begin_time that would be the better one to test with. The begin_time
setting will print out when you read the disk.
Another issue I didn't previously say is that my code doesn't attempt
to deal with bad sectors on the disk. If the target disk has bad sectors
whatever ends up on them will get errors when read. Any flagging of bad
tracks/sectors on the original disk will still be marked bad though they
are likely good on the target disk.

###########

Betref: Re: Write image with MFM Reader/Emulator to a second Harddiskdrive
Von: David Gesswein <djg@pdp8online.com>
Datum: 10.09.2017 15:20

On Sun, Sep 10, 2017 at 03:07:27AM -0700, Fritz wrote:
>1. "controllers for the system " do you mean the MFM-Emulator hardware (I
>have several) or the MFM-Controller of the old computer?
>The controller in the old computer. The only way you would know would be
>to read the disk with the MFM emulator or use mfm_util on an emulator file.
>2. "begin_time".. with this I have to look into your documents and hope I
>understand a little bit. Please explain the "begin_time" for a novice like
>me.

It's a command line option.
http://www.pdp8online.com/mfm/code/mfm/mfm_read_util_doc.html
If you read a drive with mfm_read --analyze it will print the value in

option list when it prints "Command line to read disk:"
When reading a disk it reads the transitions from index pulse to index pulse. If a index pulse occurs in the middle of a sector it then can't decode it. Begin time shifts the actual start and end of reading by that time interval so it happens in the fill data between sectors. Index pulse is a signal that goes active one per revolution of the disk. Most controllers when formatting the disk wait for the index pulse then start writing from the first sector. Some controllers seem to be really slow getting to writing the first sector so the last sector straddles the index pulse.
The one ICL computer I have information on uses the Xebec S1410 which needs non zero begin time. Other models may use different controllers which don't need it.
The system I tried mfm_write with uses 0 begin time so non zero may never have been tested, I was just saying if one of the two systems you had to test with used non zero begin time that would be a better test.
You can also test by making an emulator file that uses begin time, writing to a drive, and reading it back. Since Xebec isn't currently supported by ext2emu you can test using northstar format.
Create a file with 16*heads*cylinders. For this example 4 heads and 100 cyl. Adjust as needed for your drive
dd if=/dev/zero of=/tmp/zeros bs=512c count=6400
ext2emu --emu /tmp/emu --ext /tmp/zeros --cyl 100 --heads 4 --format NorthStar_Advantage
Then use mfm_write to write /tmp/emu to a drive and then use mfm_read to read it.
You need to look up the drive and see what cylinder is recommended for write precompensation and edit line in mfm_write.c. Unless you have more information leave the precom_ns values as is.
Also see http://www.pdp8online.com/mfm/code/mfm/mfm_write_doc.html
for change needed to mfm_read-00A0.dts. Will need to reboot each time the file is changed. Will need to be changed back to read. Also write jumper needs to be installed for write and best to remove it for reading.

mfm_write --emu /tmp/emu
mfm_read --ana --ext /tmp/t
Should get something like
Command line to read disk:
--sectors 16,0 --heads 4 --cylinders 100 --header_crc 0x0,0x0,16,0 --data_crc 0x0,0x0,32,0 --format NorthStar_Advantage --sector_length 512 --retries 50,4 --drive 0 --begin_time 230000
And the disk reads with only the number of errors expected from the bad sectors on the disk.

##############
Betref: Re: Write image with MFM Reader/Emulator to a second Harddiskdrive
Von: David Gesswein <djg@pdp8online.com>
Datum: 10.09.2017 15:27
 (Sent off list)
FYI: I'm on vacation for the next two weeks so can't actually test anything requiring hardware until I get back. Any of the commands I just posted that need real drive haven't been verified.
Forgot to put in the real posting.
These messages are harmless.
Not all track filled, 10416 of 10418 bytes uses
(These are because of all zero data)
No more peaks in histogram
Secondary transition period 0 ns, likely RLL
RLL is not currently supported

(End of all)