

DEC PDP-11/23 SE

Special Edition
DEC = Digital Equipment Corporation

Von: *Reinhard Heuberger*

Aufbau: 1983-1987



Restauriert: 2008

Allgemein:

Das System wurde fast ausschließlich mit DEC Standard-Komponenten aufgebaut, allerdings mit folgenden 2 **Ausnahmen**, bzw. **Besonderheiten**:

1) **Gehäuse ist Eigenbau**

2) **Q-BUS <--> CTI-BUS Converter, Eigenbau.**

Dieser Converter dient zum Ansteuern der 5MByte Hard-Disk, welche ursprünglich aus einem DEC PC350-Professional stammte. Ausführliche Infos dazu unter Punkt 3

Entstehung und Aufbau

Das System wurde **1983** erstmals aufgebaut, zuerst mit einem KD11 Processor Board , auch LSI-11/2 CPU (M7270) genannt. Ab **1985** bis **1987** wurde ein PDP-11/23 CPU Board (M8186) mit Floating-Point Unit FPF11 eingesetzt. Ende **1985** wurde das Doppel-Floppy Laufwerk RX02 mit einer 5Mbyte RD50 Hard-Disk ergänzt mit eigens entwickeltem **Q-BUS <--> CTI-BUS Converter**.

Restaurierung des Systems

Die Restaurierung des Systems wurde nach ca. 22 Jahren Lagerung in einem Kellerraum **2008** durchgeführt. Das defekte Netzteil wurde durch ein Standard-PC-Netzteil ersetzt. Allerdings musste eine eigene Logic für die Power-Up Signale und der System-Clock extra gebaut werden (Siehe Video) . Das gesamte Äußere wurde durch Einsatz von Plexi-Glas und LED's dahingehend verändert, dass ein Einblick in das Innere des Systems möglich ist.

Referenz Dokumentation:

Zusätzliche Dokumentation befindet sich im Ordner: “ **DEC-DOKU** “

Die wichtigsten 5 zusätzlichen Manuals als pdf-Datei:

BA11-N mounting box user's guide

Part-Nr.: EK-BA11N-UG-001 , Datei : **ballnug1.pdf**

digital LSI PDP 11/03 MAINTENANCE

Part-Nr.: EK-LSI11-MC-01 , Datei : **ek-lsi11-mc-001.pdf**

LSI Systems Service Manual volume 1

Part-Nr.: EK-LSIFS-SV-05 , Datei : **ek-lsifs-sv-005-vol1.pdf**

LSI Systems Service Manual volume 2

Part-Nr.: EK-LSIFS-SV-05 , Datei : **ek-lsifs-sv-005-vol3.pdf**

LSI Systems Service Manual volume 3 ,

Part-Nr.: EK-LSIFS-SV-05 , Datei : **ek-lsifs-sv-005-vol3.pdf**

Videos

Im Ordner: “ **V I D E O S** “ befinden sich 3 **aussagekräftige** Videos:

Datei **Video#3.mpg** : Architektur, Konzept und Restaurierung des Systems.

Datei **Video#4.mpg** : RT-11 Basic+Assembler Programmierung auf PDP-11/23

Datei **Video#5.mpg** : Backup von historischen Computern mit Kermit

Die wichtigsten System-Komponenten

Bild 1 : PDP-11/23 CPU Board (M8186) mit Floating-Point Unit FPF11

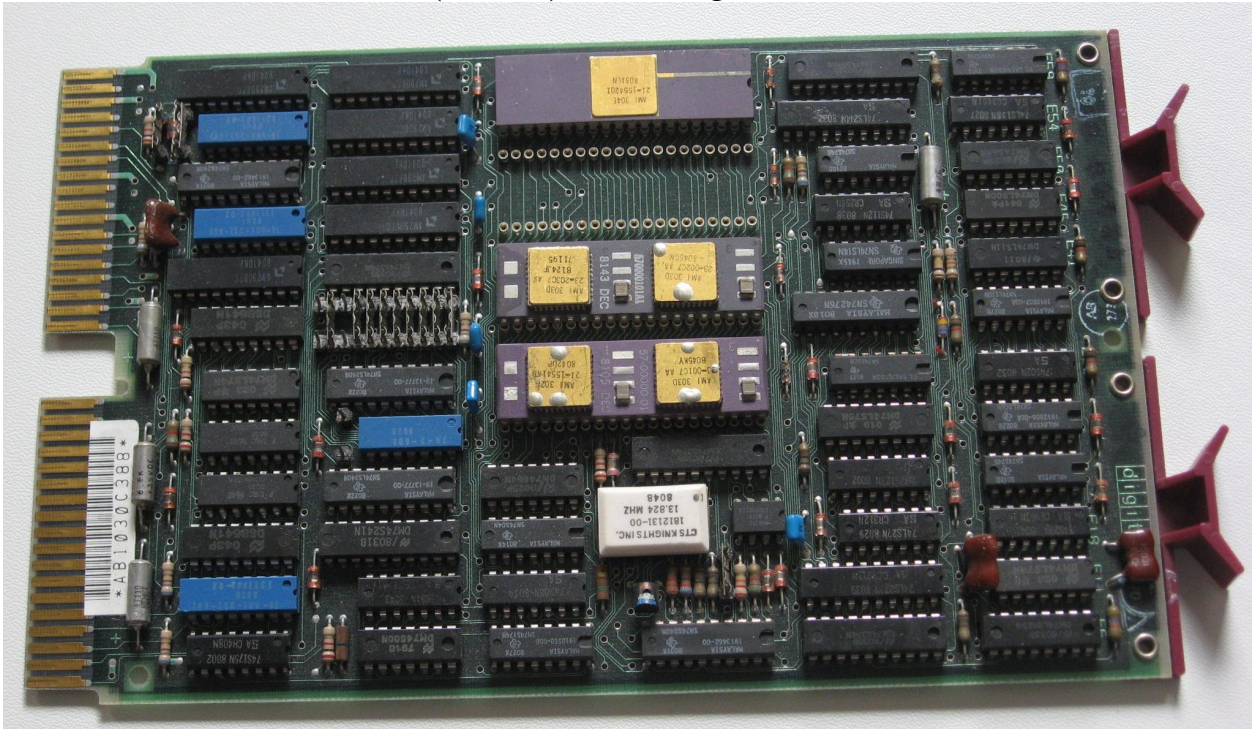


Bild 2-A : Multi-Funktions Board MXV-11 (M8047) mit 2 Serial I/O Ports und Boot-ROMs für Floppy und RH-Hard-Disk.

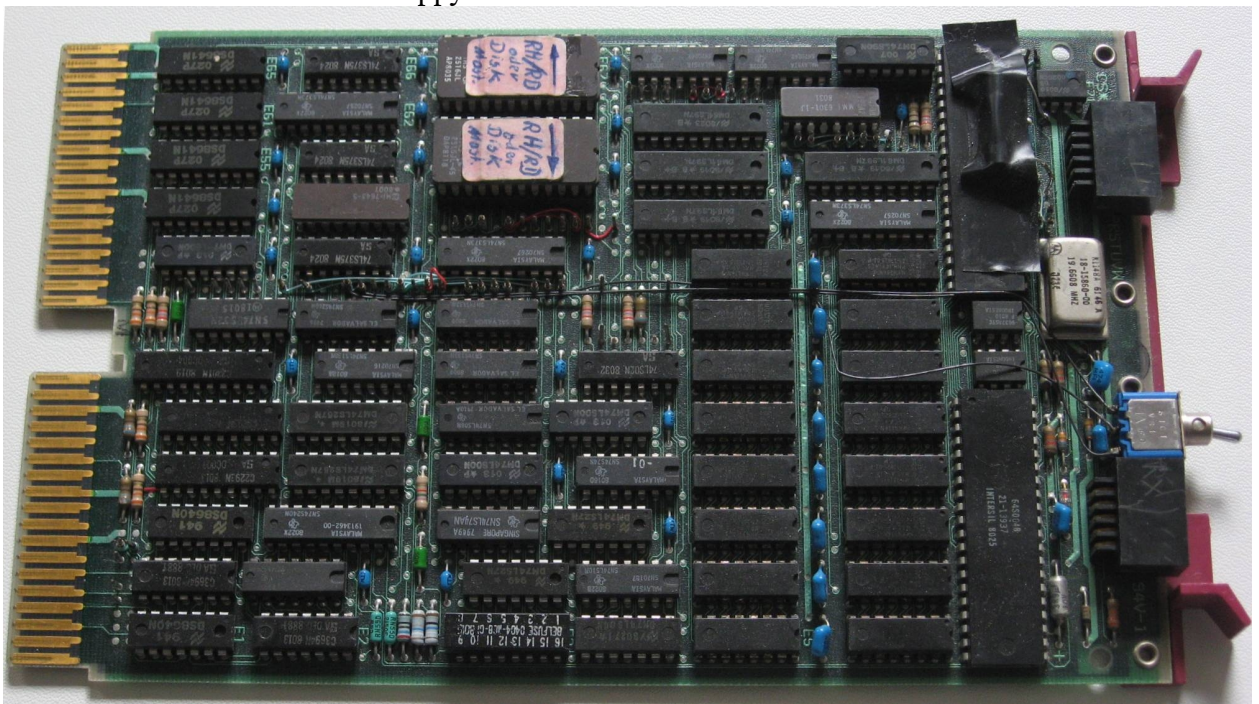


Bild 2-B : Memory-Board , M8067 , 512Kbyte (voll bestückt) . Hinweis: Bedingt durch die nur 18 Adress-Lines des H9270 Backplane können nur 128Kbyte benützt werden

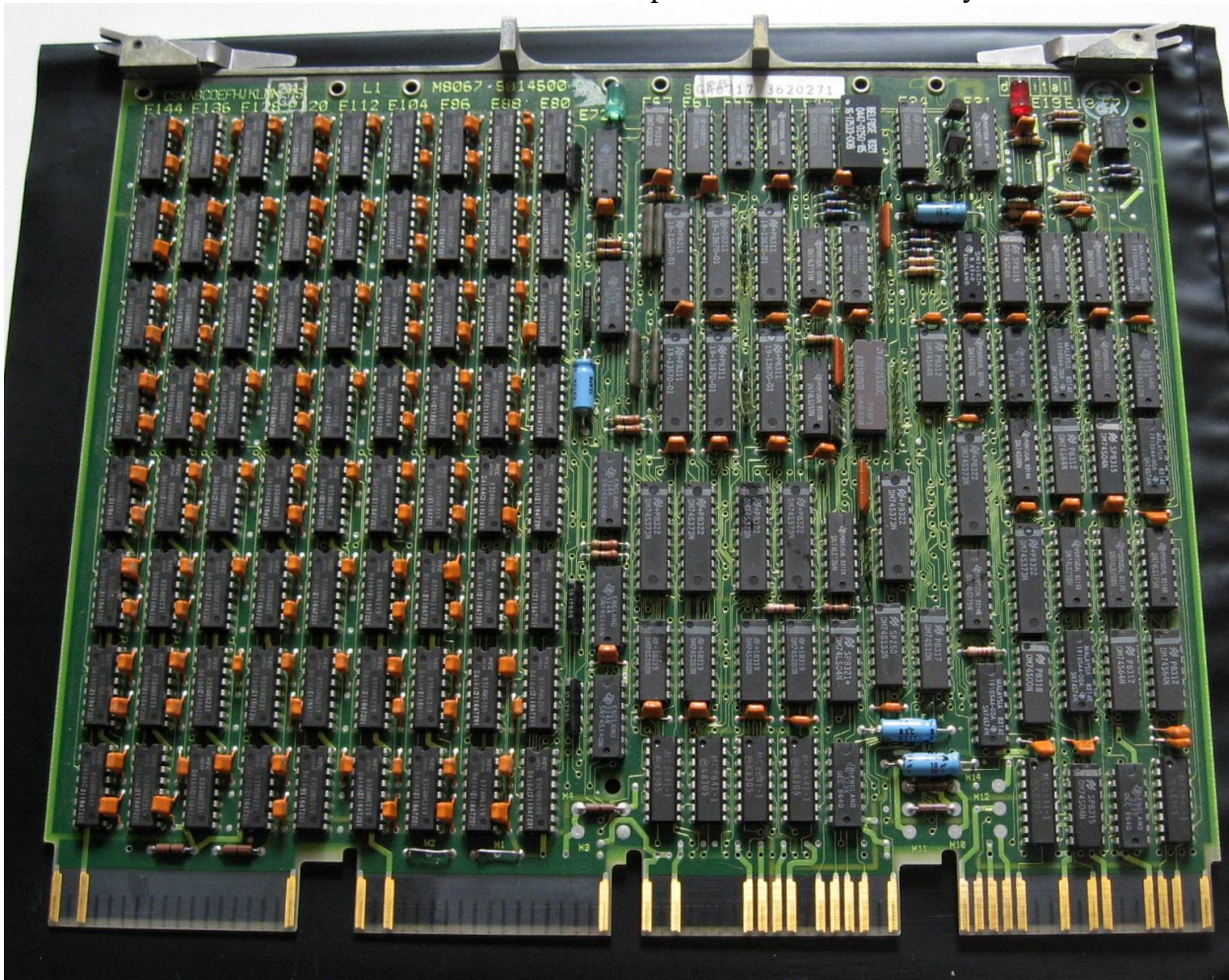
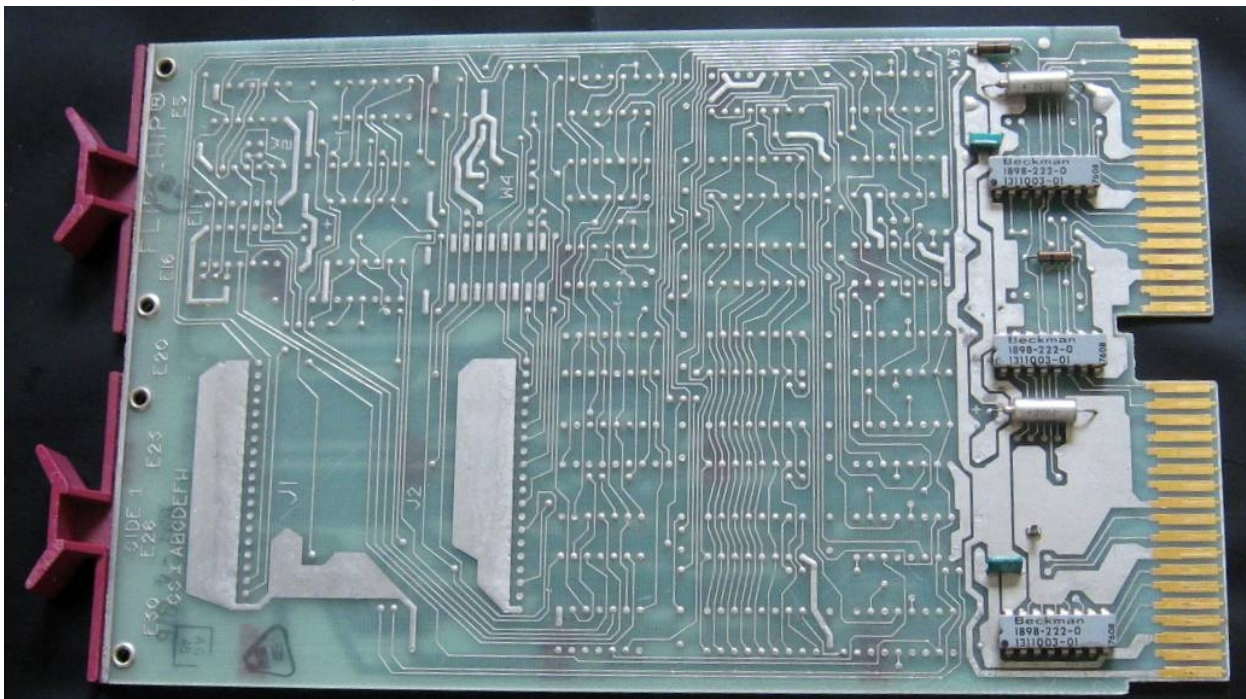


Bild 2-C : M9400 YB **Q-BUS** Terminator Module



Zu Bild 1:

Das im Bild ersichtliche CPU Module ist auf den DEC PDP-11 F-11 (Code-Name war DEC "Fonz-11") Chipset aufgebaut und zusätzlich mit der Floating Point Unit FPF11 bestückt. Dieser Chipset wurde 1979 erstmals vorgestellt und arbeitete mit einer Core Frequenz von 3.6 MHz .

Zu Bild 2-A :

Das Module MXV-11 / M8047 besteht aus 2 *SLU* (*Serial Line Unit*) und 2 Boot-ROMs. In meinem Fall wurde der Boot Code in den ROMs erweitert, um zusätzlich die RD50 über meinem entwickelten **Q-BUS <--> CTI-BUS** Konverter zu booten. Dies wurde mit den größeren 8K by 8 UV-ROMs erreicht und wie im Bild ersichtlich kann mit einen Schalter zwischen den Standard "DEC BOOT Code" und dem "BOOT Code" für die RD50 bequem umgeschaltet werden. (Zusatzinfo im Ordner " **DEC-DOKU** ", **mxv1bug1.pdf** , Seite 54) Hier noch das Listing des Assembler Programms welches sich zusätzlich in den UV-ROMs befindet:

```
.TITLE BOOT
;
;+
; PROGRAM TO BOOT THE RD50 WITH THE Q-BUS <--> CTI-BUS
; CONVERTER, TO RUN THE BOOTSTRAP AT POWER-UP
; FOR THE RD50                                From: Reinhard.Heuberger
;-
START: MOV    #10,@#174020      ; RESET CONTROLLER
BUSY:  TST     @#174020         ; CONTROLLER BUSY ??
      BMI     BUSY             ; STILL BUSY
      CLR     R4               ; SETUP POINTER
      MOV     #1,@#174006      ; SECTOR -->1
      MOV     #0,@#174012      ; CYLINDER -->0
      MOV     #0,@#174014      ; HEAD -->0
      MOV     #40,@#174016     ; SET CONTROLLER TO READ MODE
WAIT:  TST     @#174020         ; CONTROLLER BUSY ??
      BMI     WAIT             ; STILL BUSY
GET:   BIT     #1,@#174020     ; ONE BLOCK TRANSFER DONE ??
      BNE     EXE              ; YES, GOTO START SEC. BOOT
      TSTB    @#174020         ; NEXT DATA AVAILABLE ??
      BPL     GET              ; NOT YET
      MOV     @#174010,(R4)+    ; BRING ONE CHARACTER INTO MEMORY
      BR      GET              ; AND LOOP UNTIL FINISCH
EXE:   CLR     R5               ; POINT TO ADDRESS 0
      CMP     @R5,#240          ; DID WE READ A VALID BOOTSTRAP BLOCK?
      BEQ     DO               ; YES, GO TO START EXECUTION !!
      HALT                    ; PRIMARY BOOT E R R O R !!
;
DO:    MOV     #1000,SP         ; LOAD PROGRAMS STACK POINTER
      MOV     #2,R0             ; BLOCK NUMBER OF SECONDARY BOOTSTRAP
      MOV     #<4*256.>,R1      ; WORD COUNT OF 4 BLOCKS(2-5)
      MOV     #1000,R2         ; MEMORY ADDRESS OF SECONDARY BOOT
      JMP     @R5              ; AND - START -
      .END
```


Bild 3 : 5MByte Hard-Disk RD50 mit **CTI-BUS** Controller von DEC PC350 Professional
 Darunter: **Q-BUS <----> CTI-BUS Converter, Eigenbau** , basierend auf
 W951 DEC-Foundation-Module , Wire-Wrap Technologie. Beide Komponenten
 sind über ein 2mal 20-Pin Flachbandkabel verbunden, = **CTI-BUS**

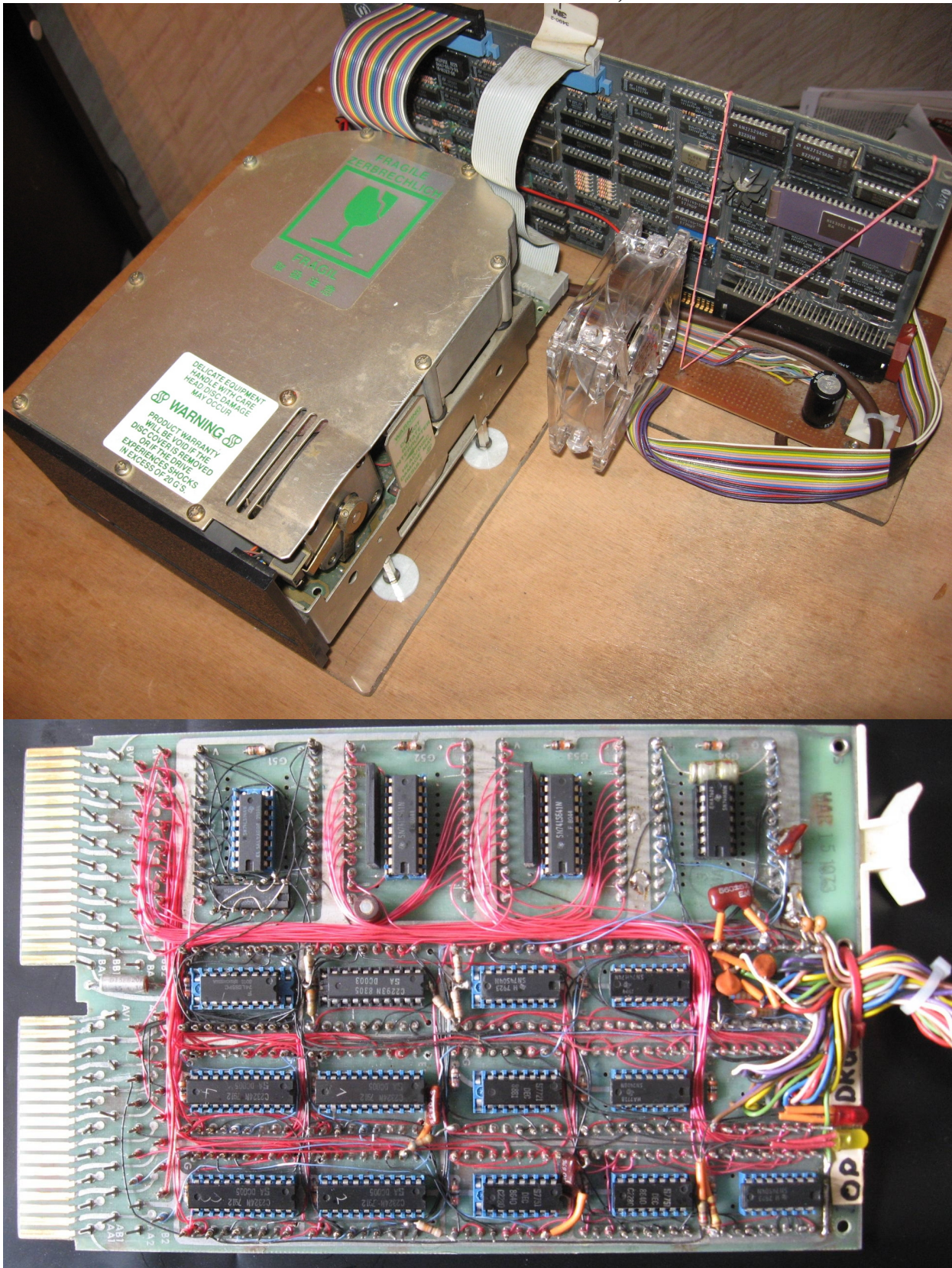


Bild 4 : Steuer-Panel, Vorderansicht und Rückansicht.

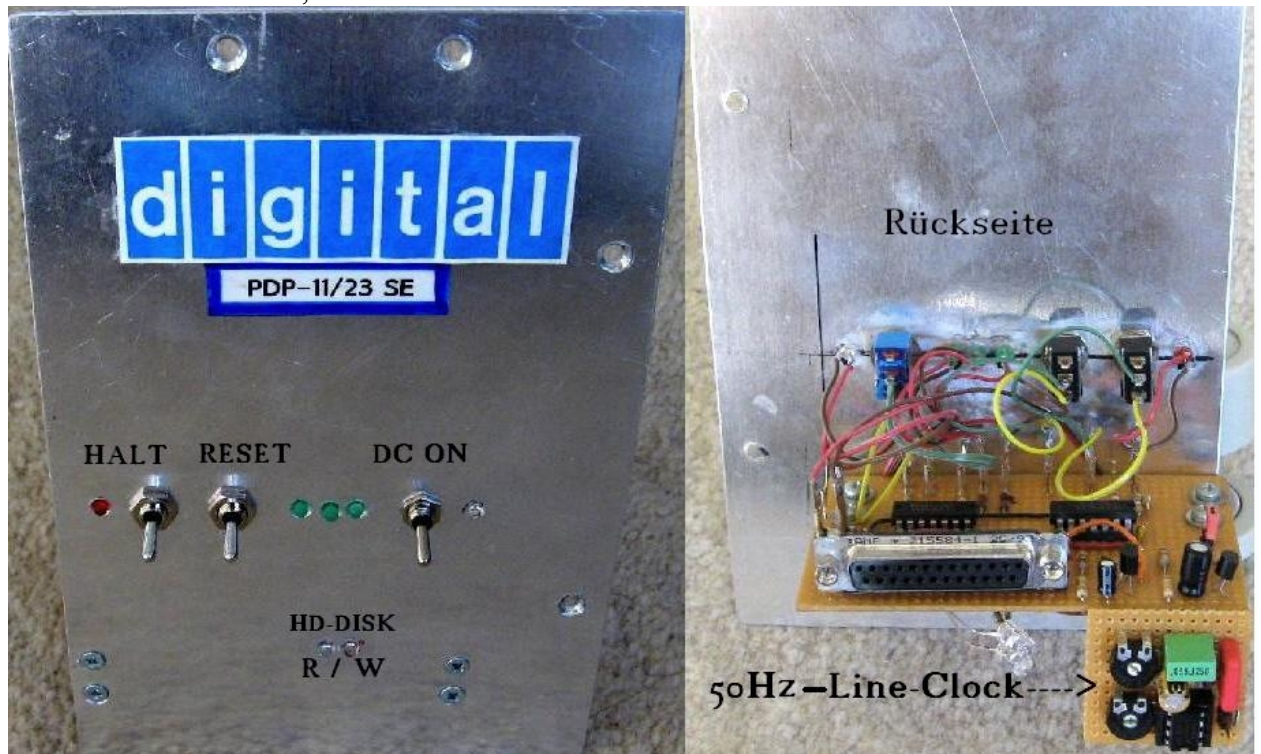
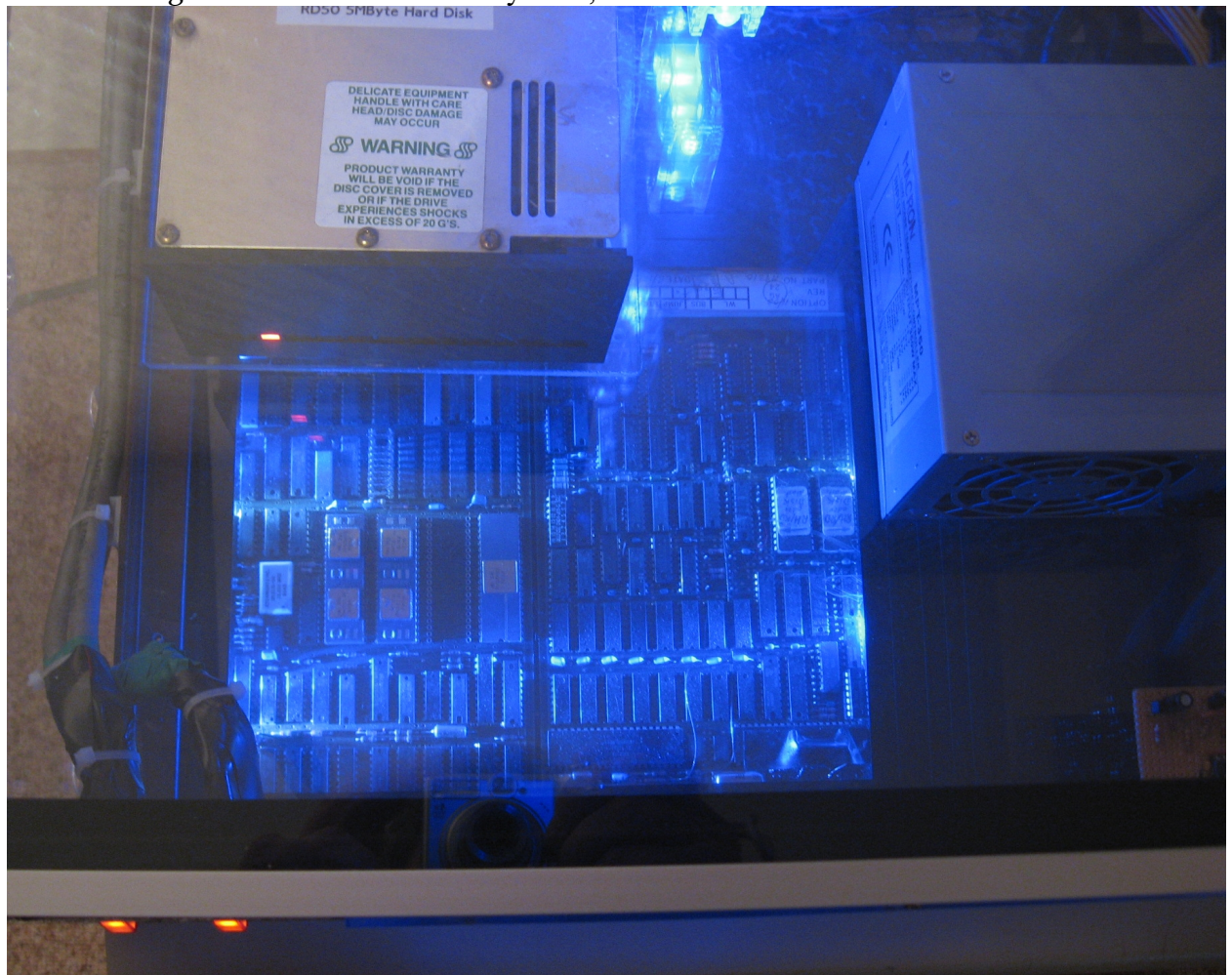


Bild 5 : Eingeschaltetes/beleuchtetes System , Ansicht von Oben durch Glas-Platte.

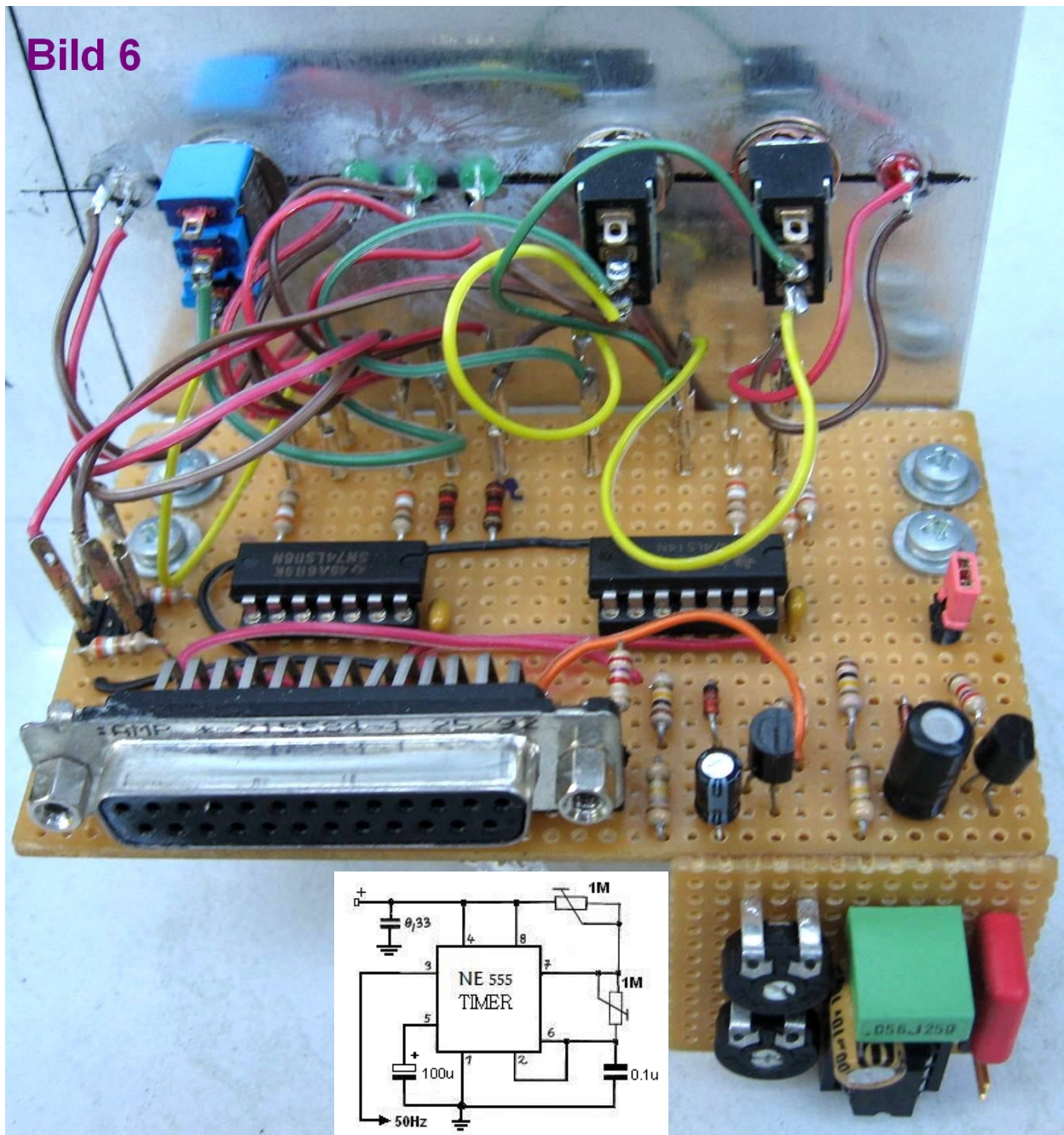


Die Restaurierung

Das gesamte System wurde 1987 in einem Kellerraum eingelagert, ohne Klimatisierung und 2008, also nach gut 21 Jahren wieder versucht in Betrieb zu nehmen. Zu meiner großen Überraschung konnte das System auch sofort gestartet werden und das Betriebs-System RT11 startete ohne Probleme von der 5 MB Hard-Disk .

Allerdings dauerte dies nur 20 Minuten und dann verabschiedete sich das Netzteil für immer. Das allgemein bekannte Problem: u.A. waren die **Elkos ausgedrocknet**. Ersatzteile gibt es nicht mehr und ich entschied mich für eine länger anhaltende und flexiblere **Lösung**: Ersatz durch ein **Standard-PC-Netzteil**. Für den Betrieb einer PDP-11/23 sind allerdings die Power-Up Signale **BPOK** und **BDCOK** als auch das System-Clock Signal **BEVNT** **unbedingt** erforderlich. Da diese Signale in einen PC -Netzteil nicht existieren mussten die Signale mit einer eigens dafür entwickelten Schaltung und Mechanik aufgebaut werden , wie im folgenden **Bild 6** ersichtlich:

Bild 6



Im letzten **Bild 6** ist der Line-Clock Generator als angebautes Module mit dem dazugehörigen Schaltplan zu erkennen. Hiermit werden 20ms / 50 Hz Impulse erzeugt. Dieses Signal wird dann über ein Open-Collector-Gate , 7406 als Signal **BEVNT** auf das H9270 Backplane geschaltet.

Power-Up-Signale **BPOK** und **BDCOK**

Diese wichtigen Signale müssen genau nach den Spezifikationen wie im folgenden **Bild 7** ersichtlich ablaufen.

Bild 7

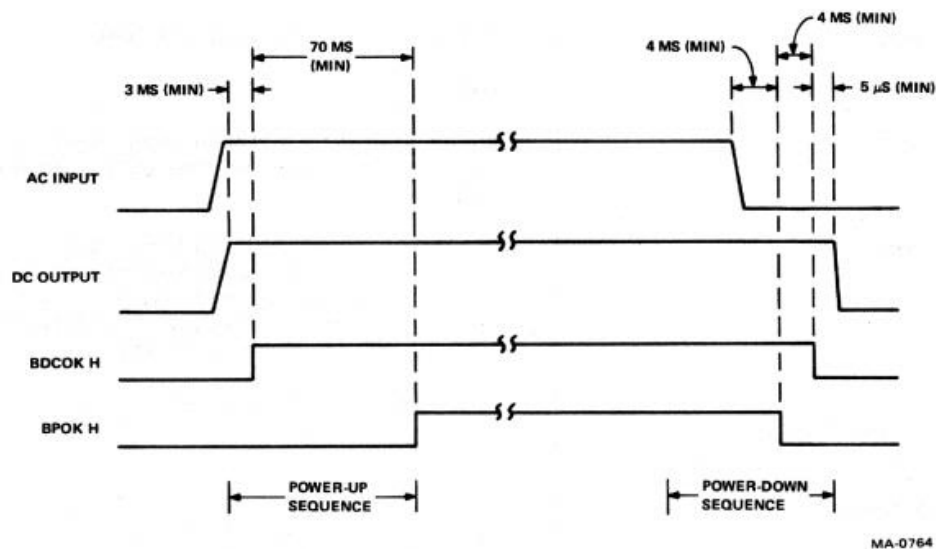
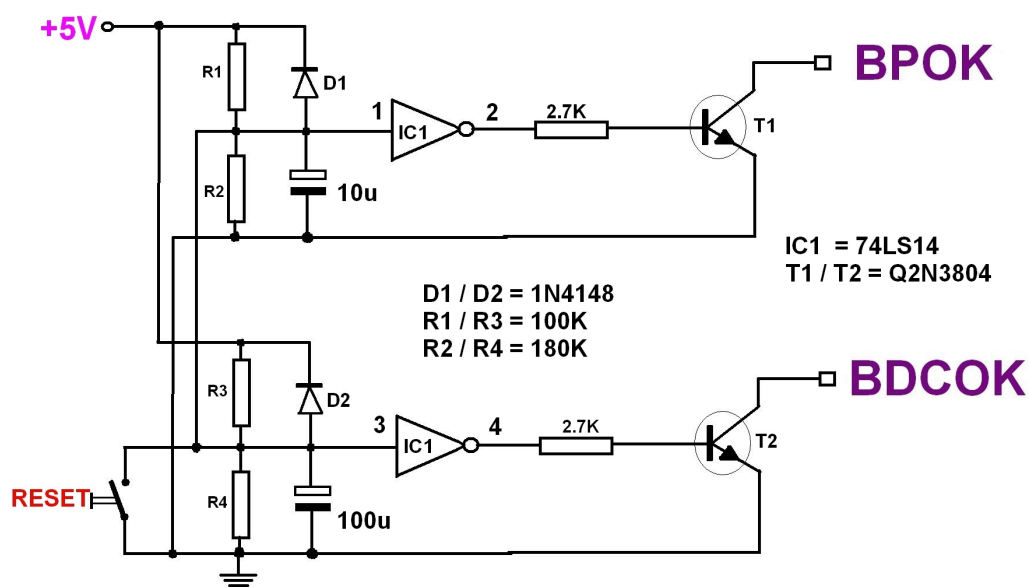


Figure 1-11 Power Up/Power Down Timing

Hier dazu der Schaltplan in **Bild 8** . Im Prinzip sind es 2 identische Schaltungen mit unterschiedlicher Zeit-Konstante durch die Elkos : BPOK=10u , BDCOK=100u .

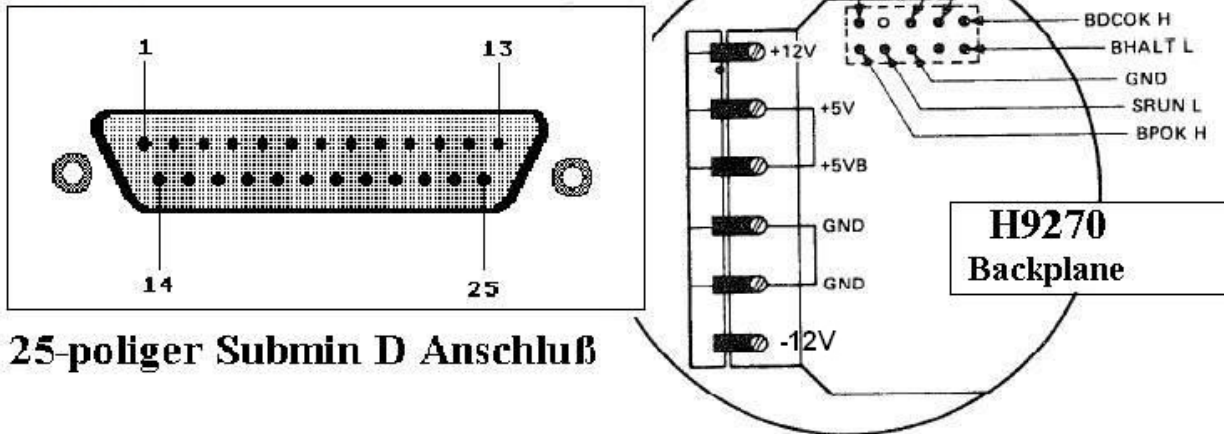
Bild 8 Erzeugung der POWER-UP Signale



Signale und Anschlüsse

Das Steuer-Panel ist über einen 25-poligen Submin-D Stecker-Anschluss mit dem H9270 Backplane und der Stromversorgung wie folgt verbunden:

Bild 9



25-poliger Submin D Anschluß

1 = +12V	2 = +5V – Standby	3 = Ground	4 = not used
5 = not used	6 = DC -START	7 = DC-START	8 = not used
9 = BHALT L	10 = not used	11 = GRND	12 = SRUN L
13 = BPOK H	14 = +5V	15 = GRND	16 = -12V
17 = not used	18 = +5V	19 = not used	20 = not used
21 = BDCOK H	22 = HD - R	23 = HD - W	24 = GRND
25 = BEVNT L (50 Hz)			

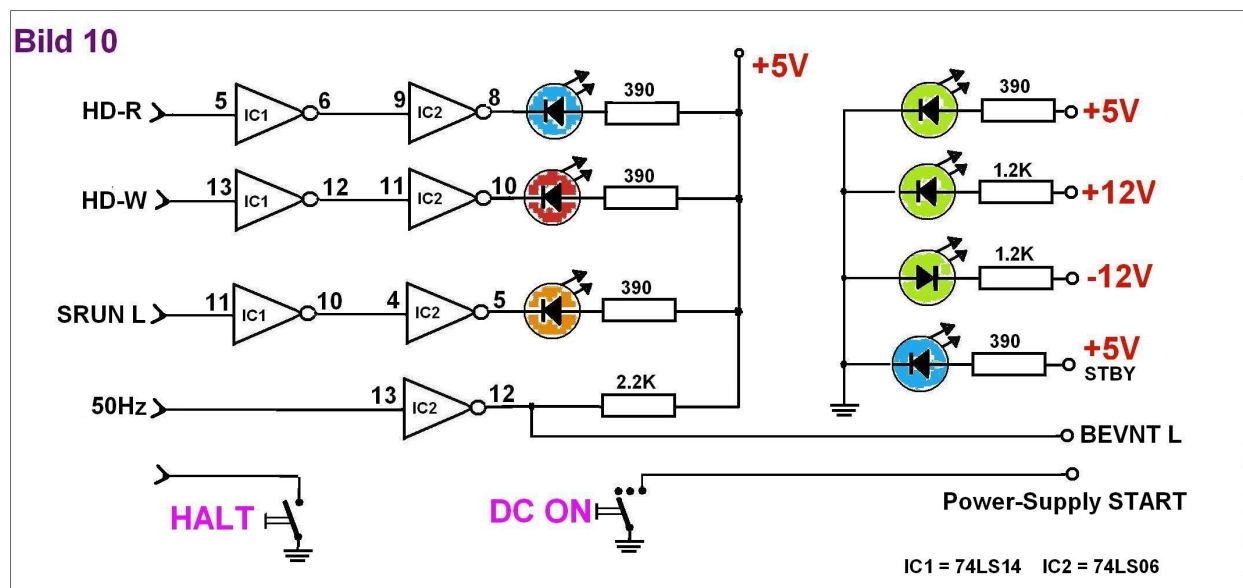
Front-Panel-Steuerung und LED's

Die Steuerung des Systems erfolgt mit 3 Schaltern: **DC ON** , **HALT** und **RESET** (Bild 8)
Die LED's zeigen den Zustand von:

HD-Read , **HD-Write** , **RUN** , **+5Volt** , **+12Volt** , **-12Volt** und **Standby-Power**

Die Schaltung dazu ist folgenden **Bild 10** ersichtlich:

Bild 10



Hinweise:**Netzteil**

Das original Netzteil wurde durch ein Standard 350 Watt (PC) -Netzteil ersetzt . Ein solches Netzteil sollte folgende Minimal-Leistung zur Verfügung stellen:

+5 Volt 25 A , +12 Volt 6 A , -12 Volt 1A , +5Volt Standby 1A

Die -12 Volt werden lediglich für die Serial Receive/Transmit Chips 9636 und 9637 auf dem MXV-11 Modul benötigt. Die +5 Volt Standby wird nur für die LED's benötigt und steht sofort zur Verfügung, wenn das PC Netzteil mit 220Volt versorgt wird . Das Netzteil wird über den Eingang "**Power Supply start**" eingeschaltet , rechter Schalter am Front-Panel, = **DC-ON** . Nach dem Einschalten sollten alle **3 grünen** LED's leuchten und zeigen somit die Verfügbarkeit von **+5V, +12V und -12 V** an.

Signale

Nach dem Einschalten werden die Power Up Signale wie in Bild 7 ersichtlich erzeugt, die CPU startet und befindet sich **RUN Mode**, angezeigt durch die rote LED im linken Teil des Front-Panel. Das Signal **SRUN L** ist ein pulsierendes Signal und ein Indikator, ob sich die CPU im Run-Mode befindet. Im Original steuert dieses Signal ein 74LS123 Mono-Flop an, um das Run-Signal gut sichtbar zu machen. In meinem Fall löste ich dies mit einer modernen leuchtstarken LED, welche über einen 74LS06 LED Treiber mit Vorschaltung eines 74LS14 Schmitt-Trigger den selben Zweck erfüllt, allerdings mit wesentlich weniger Aufwand.

System Clock Signal : Dieses wichtige Signal , **BEVNT L** wird auf das H9270 Backplane geschaltet und ist für alle zeitabhängigen Operationen erforderlich. Weil das alternative Clock Signal vom MXV-11 Module bei dem H9270 Backplane nicht verwendet werden kann und weil ein solches Signal bei einem Standard Netzteil nicht zur Verfügung steht, musste ein eigener 50Hz Oszillator gebaut werden. Wie im Bild 6 ersichtlich, ist dies mit einen NE555 Timer und nachgeschalteten Open Collector Treiber, 74LS06 realisiert worden.

HARD-DISK Anzeige: Die Signale **OP** (Operation) und **DRQ** (Data-ReQuest) von meinem **Q-BUS <--> CTI-BUS** Converter entsprechen weitgehend einer Read / Write Funktion und werden auf dem Controller selbst durch zwei LED's angezeigt. Um die RD50 Hard-Disk Operationen auch am Front-Panel sichtbar zu machen wurden die beiden LED-Signale zum Front-Panel geschaltet und über jeweils 1/6stel 74LS14 aufbereitet und dann mit 1/6stel 74LS06 die LED am Front-Panel angesteuert.

Restaurierung am MXV-11 Module:

Leider gab es immer wieder Ausfälle mit den Transmitter / Receiver Chips 9636 und 9637. Das Auswechseln ist sehr schwierig, so dass ich mich entschieden hatte, hier jeweils einen Sockel einzulöten, um im Bedarfsfall diese Chips einfachst wechseln zu können.

Der I/O Teil des Systems: 1) Floppy-DISK-SYSTEM

Bild 11 : RX02 Floppy-Disk-System , für 8 Inch Floppy Diskette , 2mal 0.5 Mbyte.

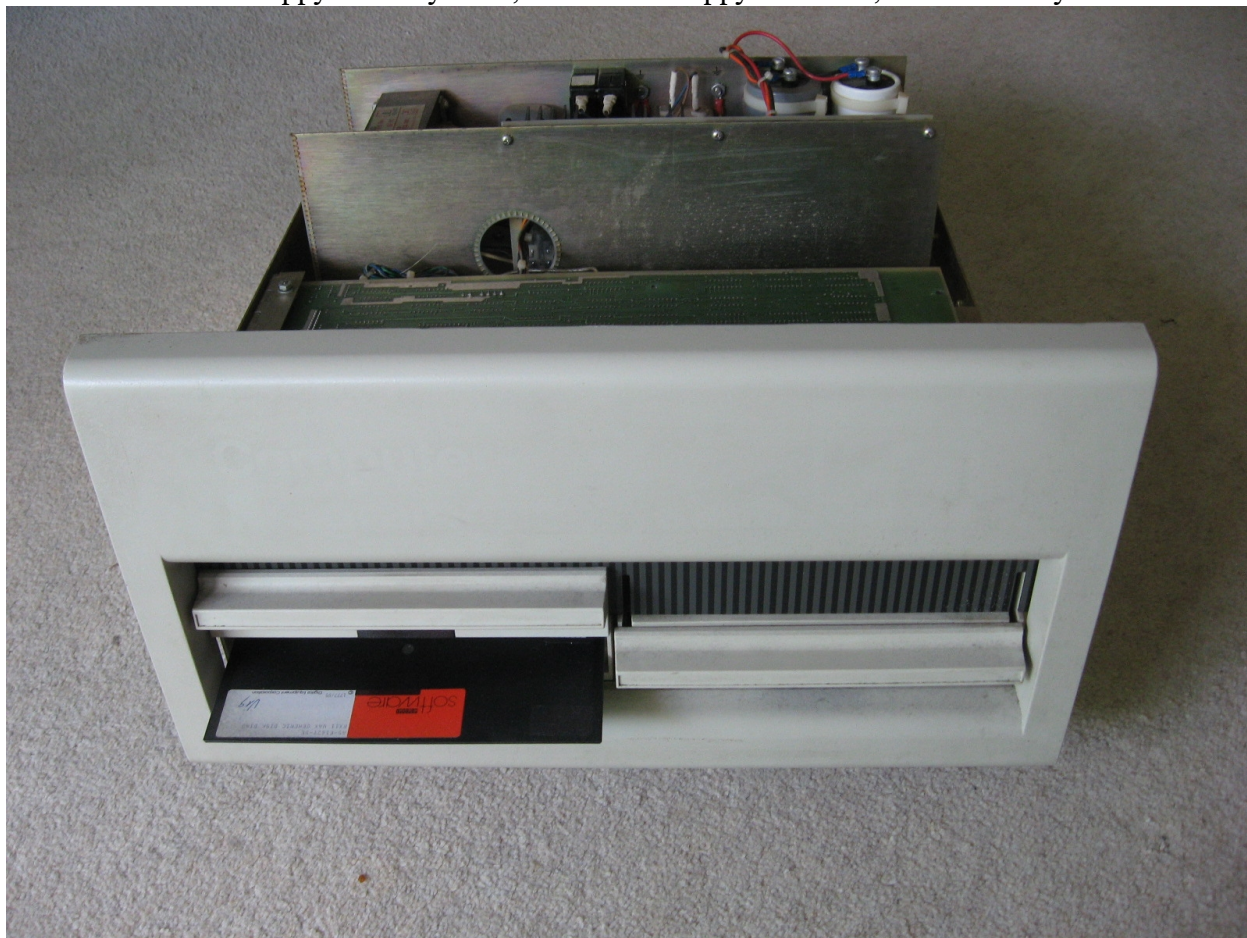


Bild 12 : RXV21 , M8029 **Q-BUS** DMA Controller für RX02



Bild 13 : Ansicht von oben , Controller Board (Rückseite) und Netzteil.



Bild 14 : Blick auf Read/Write und Stepper-Motor-Treiber Board

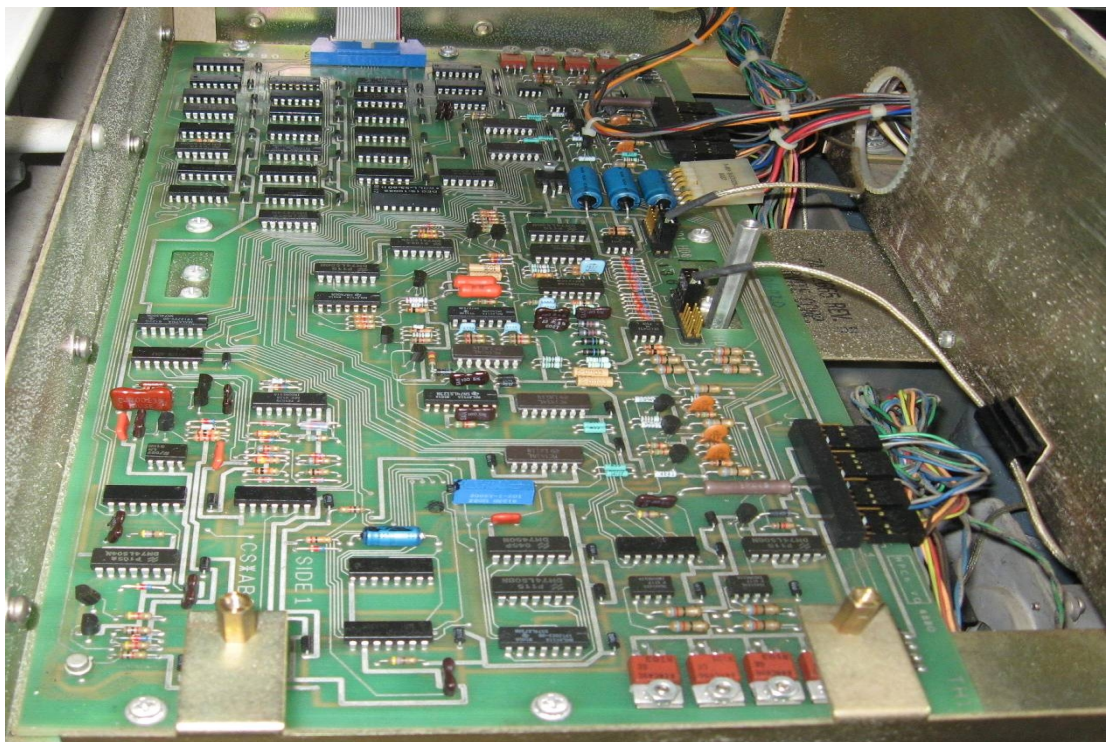


Bild 15 : Ausgebautes Floppy Laufwerk

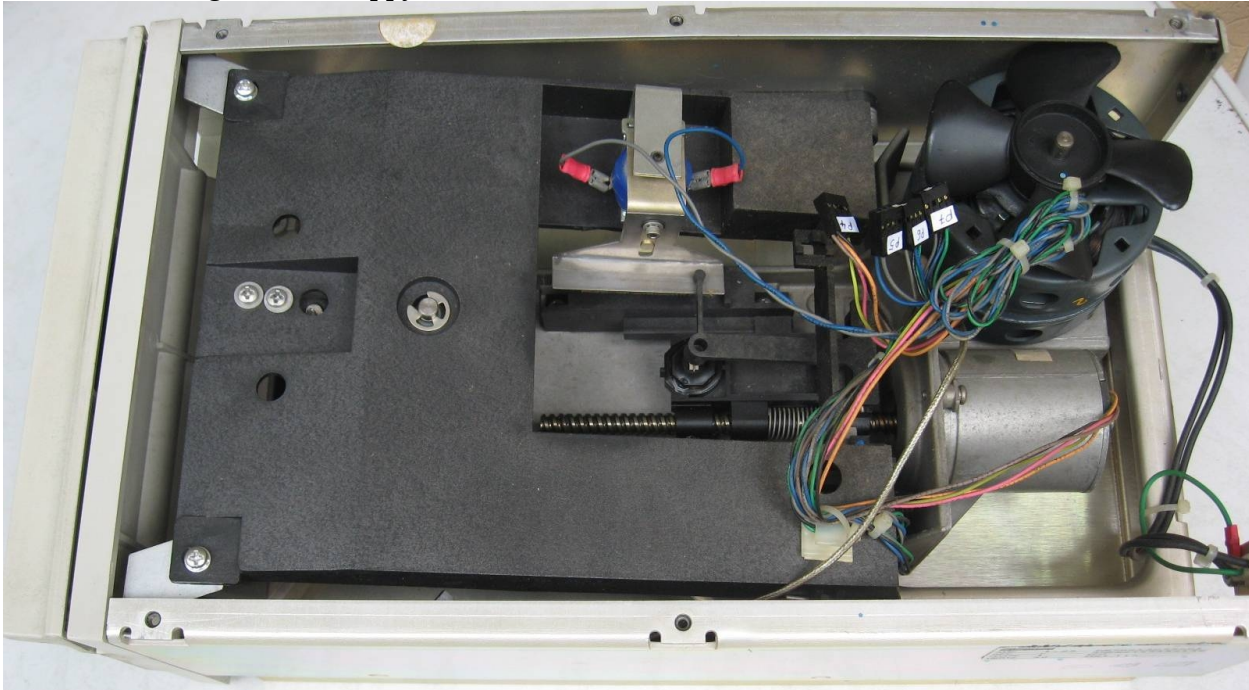
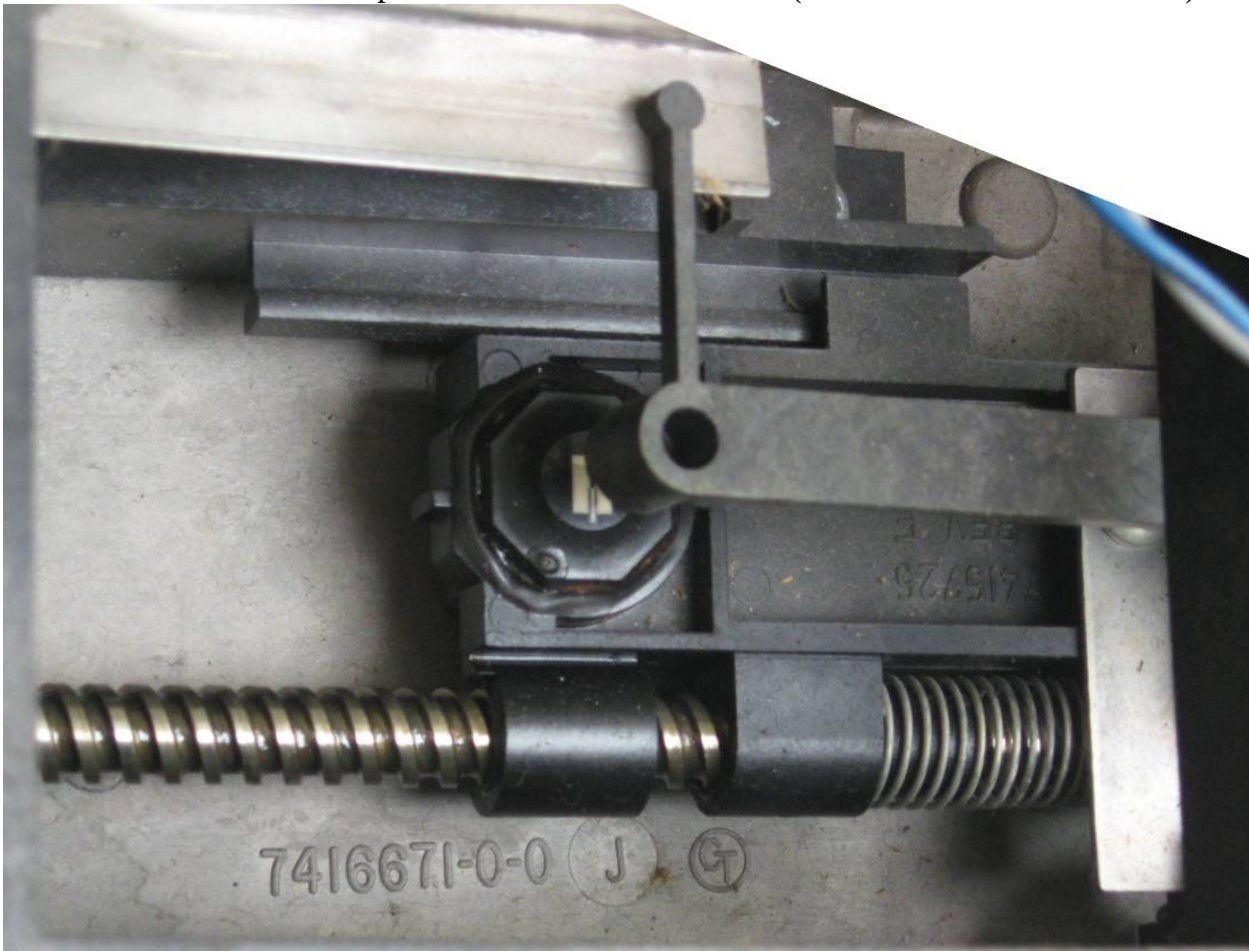


Bild 16 : Schreib/Lese Kopf mit Lade/Andruck Mechanik (HEAD LOAD AKTUATOR)



PIN Assignment , Q-BUS

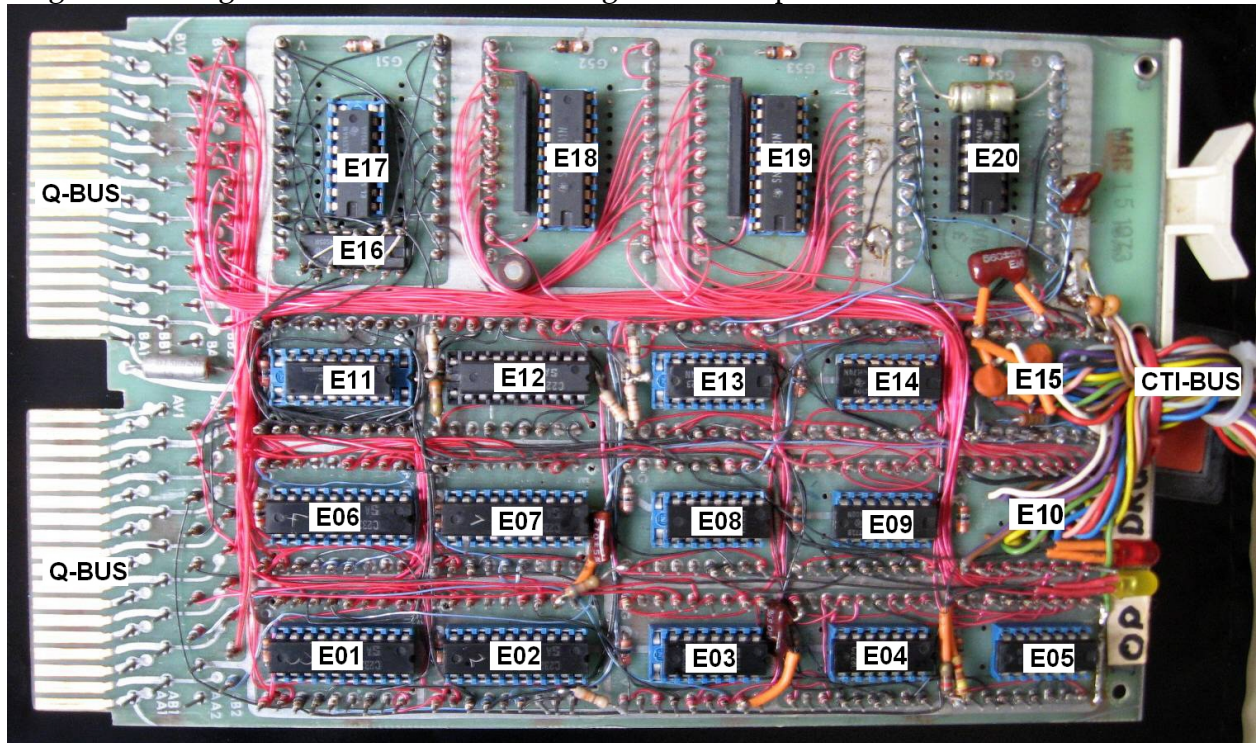
Row A (Same as Row C)		Row B (Same as Row D)	
Module Side 1 (Component Side)			
AA1	BSPARE1	BA1	BDCOK H
AB1	BSPARE2	BB1	BPOK H
AC1	BAD16	BC1	SSPARE4
AD1	BAD17	BD1	SSPARE5
AE1	SSPARE1	BE1	SSPARE6
AF1	SSPARE2	BF1	SSPARE7
AH1	SSPARE3	BH1	SSPARE8
AJ1	GND	BJ1	GND
AK1	MSPARE A	BK1	MSPARE B
AL1	MSPARE A	BL1	MSPARE B
AM1	GND	BM1	GND
AN1	BDMR L	BN1	BSACK L
AP1	BHALT L	BP1	BSPARE6
AR1	BREF L	BR1	BEVNT L
AS1	PSPARE3	BS1	PSARE4
AT1	GND	BT1	GND
AU1	PSPARE1	BU1	PSPARE2
AV1	+5B	BV1	+5
Module Side 2 (Solder Side)			
AA2	+5	BA2	+5
AB2	-12	BB2	-12
AC2	GND	BC2	GND
AD2	+12	BD2	+12
AE2	BDOUT L	BE2	BDAL2 L
AF2	BRPLY L	BF2	BDAL3 L
AH2	BDIN L	BH2	BDAL4 L
AJ2	BSYNC L	BJ2	BDAL5 L
AK2	BWTBT L	BK2	BDAL6 L
AL2	BIRQ L	BL2	BDAL7 L
AM2	BIAKI L	BM2	BDAL8 L
AN2	BIAKO L	BN2	BDAL9 L
AP2	BBS7 L	BP2	BDAL10 L
AR2	BDMGI L	BR2	BDAL11 L
AS2	BDMGO L	BS2	BDAL12 L
AT2	BINIT L	BT2	BDAL13 L
AU2	BDAL0 L	BU2	BDAL14 L
AV2	BDAL1 L	BV2	BDAL15 L

PIN Assignment CTI-BUS

BDCOK H	1
+5 Vdc	2
BPOK H	3
GND	4
BINIT L	5
-12 Vdc	6
BDAL15 L	7
BDAL13 L	8
BDAL14 L	9
BDAL12 L	10
NOT USED	11
BDAL11 L	12
BRPLY L	13
BDAL10	14
GND	15
BDAL9 L	16
BMDEN L	17
BDAL8 L	18
BWR L	19
BDAL7 L	20
NOT USED	21
BDAL6 L	22
NOT USED	23
BDAL5 L	24
BSDEN L	25
GND	26
BDAL3 L	27
BSSL	28
BDAL2 L	29
IRQ8 L	30
BDAL1 L	31
IRQAL	32
BDAL0 L	33
GND	34
GND	35
BDSL	36
+5 Vdc	37
BASL	38
+12 Vdc	39
NOT USED	40
THRU	41
	60

Hardware: Das Design

Die Hardware wurde in Wire-Wrap Technik auf einen DEC W951 **Q-BUS** Foundation Module aufgebaut. In folgenden **BILD 17** sind die eingesetzten Chips und deren Position ersichtlich:



E01, E02, E06, E07 = DEC DC005
 E03, E04 = DEC 8640
 E05 = 74S04
 E08 = DEC 8881
 E09 = 74S08
 E11 = 74LS85
 E12 = DEC DC003
 E13 = 74s04
 E14 = 74H74
 E16 = 74LS85
 E17 = 74LS74
 E18, E19 = 74LS641
 E20 = 74H00

Q-BUS Protokoll Chip, Addressing
 BUS Receiver Chip
 Hex Inverter
 Bus Driver
 Quadruple 2-Input Positive – AND Gates
 4-Bit Magnitude Comparator
Q-BUS Protokoll Chip, Vector/Interrupt
 Hex Inverter
 Dual D-Type Positive-Edge-Triggered Flip-Flops
 4-Bit Magnitude Comparator
 Dual D-Type Positive-Edge-Triggered Flip-Flops
 OCTAL BUS TRANCEIVER (17nS)
 Quadruple 2 – Input Gates Positiv – Nands Gates

E10 = CTI-BUS #1 of 2

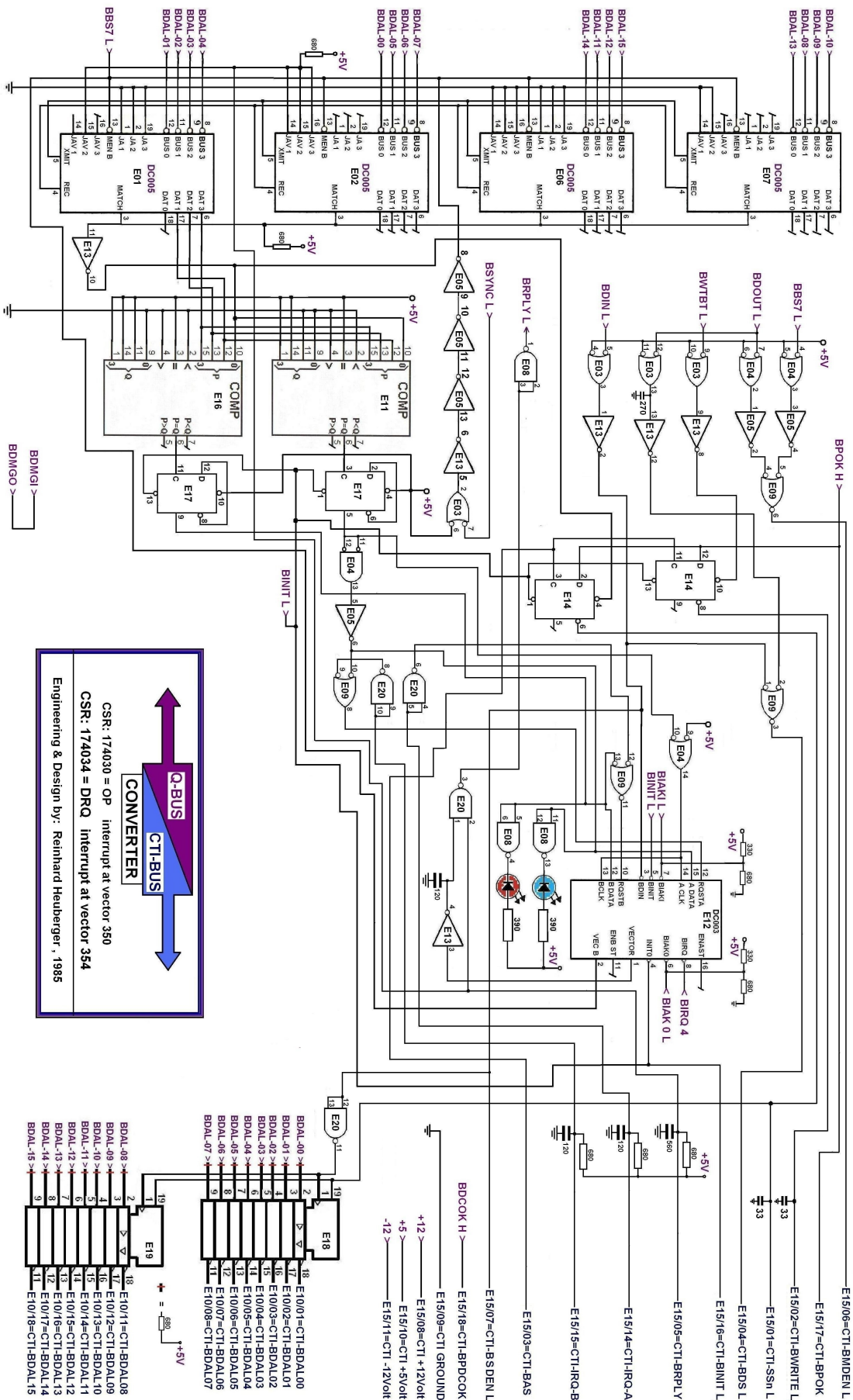
01 = CTI-BDAL-0	02 = CTI-BDAL-1	03 = CTI-BDAL-2	04 = CTI-BDAL-3
05 = CTI-BDAL-4	06 = CTI-BDAL-5	07 = CTI-BDAL-6	08 = CTI-BDAL-7
11 = CTI-BDAL-8	12 = CTI-BDAL-9	13 = CTI-BDAL-10	14 = CTI-BDAL-11
15 = CTI-BDAL-12	16 = CTI-BDAL-13	17 = CTI-BDAL-14	18 = CTI-BDAL-15

E15 = CTI-BUS #2 of 2

01 = CTI-SSn L	02 = CTI-BWRITE L	03 = CTI-BAS L	04 = CTI-BDS L
05 = CTI-BRPLY	06 = CTI-BMDEN L	07 = CTI-BSDEN L	08 = +12Volt
09 = GROUND	10 = +5Volt	11 = -12Volt	12 = NOT USED
13 = NOT USED	14 = CTI-IRQA	15 = CTI-IRQB	16 = BINIT L
17 = CTI-BPOK H	18 = CTI-BPDCOK		

Auf folgender Seite ist der komplette Schaltplan ersichtlich:

(für Details : Zoom und/oder Seite drehen oder siehe Datei: QBUS-CTIBUS-engineering-drawing.pdf)



Hinweise zum Schaltplan:**Referenzen : Timing Diagramme und Bus Spezifikationen:****Q-BUS:** Siehe Ordner “DEC-DOKU “ , ek-lsifs-sv-005-vol3.pdf**CTI-BUS:** Siehe Ordner “DEC-PRO-350 “ , EK-PC300-V1-001_pro300tecV1.pdf

Der Aufbau wurde Ende 1985 begonnen und in Wire-Wrap Technik durchgeführt.

Das Interface-Timing für den **Q-BUS** wurde mit 4mal I/O DEC Chips DC005 realisiert und die Interrupt Steuerung mit den DEC Chip DC003. Die CSR (Control Status Register) Adressen und die Interrupt Vector Adressen sind fest verdrahtet mit folgender Zuweisung:

CSR: 174030 = OP Interrupt at vector 350 (OCTAL)

CSR: 174034 = DRQ Interrupt at vector 354 (OCTAL)

Da zwei eigene Interrupt Vektoren für die dazugehörigen CSR-Adressen erforderlich sind, wurden die nötigen “Match“ - Signale mit den beiden Comparatoren E11 und E16 erzeugt. Generell: Es mussten aus den **Q-BUS** Signalen die **CTI-BUS** Signale erzeugt werden. Allerdings waren die Signale in der zeitlichen Ablauffolge verschieden und teilweise gar nicht vorhanden und/oder anders definiert. mussten also auch manchmal neu erzeugt werden, wie z.B. das CTI SS (Slot Select) Signal. Schwierig war die Erzeugung des **Q-BUS** Signals BRPLY in Abhängigkeit der **CTI-BUS** Signale. Abhilfe brachte hier eine Laufzeit-Verzögerung mit 4 Inverter Bausteinen.

Elektrisches Verhalten:

Der DEC **CTI-BUS** wurde einzig alleine für den DEC PRO-350/380 PC entwickelt und als 3-State Bus konzeptioniert mit einer maximalen Länge auf den Motherboard von ca. 10 cm. Bei meinem Design wurde der **CTI-BUS** ca. 150cm lang, also fast 15mal länger . Physikalisch wurde dieser Bus mit 2mal 20-Pin Flachbandkabel aufgebaut. Dies war unbedingt erforderlich, da sich die RD50 außerhalb des **Q-BUS** befand. Aber ... aber .. damit fingen die Probleme erst wirklich an. Lösen konnte ich sie letztendlich durch den stundenlangen Einsatz von Logic-Analyser und (Speicher)Oscilloscope und Schreiben von SCOPE-Loop Routinen in Assembler. Spikes und Signal-Übersprecher waren die Folge des 150cm langen **CTI-BUS**, welche ich dann Stück für Stück ausschalten konnte. Haupt - “Übeltäter“ waren die Spikes auf den Signalen CTI-SS (E15/1) und CTI-BWRITE (E15/2). Diese Spikes konnte ich jeweils durch den Einsatz eines Kondensator 33pf zuverlässig ausschalten.

Ab diesen Zeitpunkt ging es zügig weiter. Allerdings stellte sich noch heraus, dass die Schaltzeiten der beiden BUS Systeme bei einigen Signalen unterschiedlich waren. Um hier eine Synchronisation sicherzustellen verwendete ich Kondensatoren für das **Q-BUS** Signal BDOUT und für die **CTI-BUS** Signale IRQ-A, IRQ-B und BRPLY. Dies war eher keine professionelle Lösung, allerdings blieb mir als Privat Person aus finanziellen Gründen keine andere Wahl und letztendlich erfüllte diese Methode den Zweck die unterschiedlichen Timing zu synchronisieren.

Als Bus Treiber kamen die damals noch neu erschienen 74LS641 in Frage. Frühere Versuche mit andern Bus Treibern scheiterten, bedingt durch die Länge des **CTI-BUS** bei meinem Design

Es entstand die Schaltung wie obig dargestellt welche immer noch sehr zuverlässig arbeitet.

Hinweis: In ganz seltenen Fällen bootet Die Hard-Disk nicht , wenn die Converter-Karte aus und eingebaut werden muss. Es scheint ein Kontakt (Signal-Übersprech) Problem mit dem CTI-Bus Signal BSDEN/E15-7 zu sein und kann dadurch behoben werden, indem man den Kabelbaum in eine andere Position legt (Die Kabel sind nach 22 Jahren Keller-Aufenthalt leider schon leicht porös geworden).

SOFTWARE: Device-Treiber und Hilfsprogramme

Für den Betrieb des **Q-BUS <--> CTI-BUS Converter** wurden von mir folgende Programme entwickelt, alle in MACRO-11 Assembler programmiert:

(Source-Code aller Programme auch im Ordner "RT11-QBUS-CTIBUS-Assembler")

1) **Device Treiber für das DEC RT-11 Betriebssystem (RH.MAC)**

Sicherlich das wichtigste und auch umfangreichste Programm um sinnvoll mit meinem Design zu arbeiten. Der Programm Code wurde mit Hilfe eines Skeleton-Device-Driver Code von DEC entwickelt. Dieser Code stellt ein allgemeines Gerüst zu Verfügung um einen eigenen Device Treiber schneller zu realisieren. Der Source Code ist umfangreich kommentiert, so daß keine weiteren Erläuterungen notwendig sind.

2) **Low-Level Formatierungs Programm für RD50 Hard-Disk (RIFORM.MAC)**

Jede Hard-Disk muss natürlich formatiert werden können, so auch die RD50 . Dies wird in meinem Fall durch dieses Programm sichergestellt. Optional kann man mit diesen Programm auch interleaved-sectors formatieren, um einen Performance Gewinn zu erzielen. In der Praxis war allerdings kein Unterschied zu erkennen, sicherlich damals bedingt durch die langen Zugriffszeiten (seek time) , welche nicht annähernd im Verhältnis zum Zeitgewinn durch interleaved sectors standen und somit gar nicht zur Geltung kamen.

3) **Testprogramm für RD50 Hard-Disk (RDTEST.MAC)**

Dieses Programm testet die gesamte **Q-BUS <--> CTI-BUS Converter** Umgebung. Im Prinzip wird die RD50 mit Testpattern beschrieben und die Lese/Schreib Funktion mit RANDOM SEEKS überprüft. Im Fehlerfalle wird die genaue Ursache ausgegeben, also betroffener HEAD, CYLINDER, SECTOR und den Inhalt der Controller Status Register als auch den Inhalt des betroffenen Sectors in Form einer Liste.

4) **Boot Programm für RD50 Hard-Disk (BOOT.MAC)**

Siehe Seite 4

Im folgenden Anhang finden sie die Programme im Quellen-Code, jeweils in der Farbe wie sie obig in Punkt 1) bis 3) beschreiben wurden.

Für Fragen oder Anmerkungen können Sie mich wie folgt erreichen:

Reinhard Heuberger
Watzmannring 71
85748 Garching
Tel.: +49 89 3205134
E-Mail: Reinhard.Heuberger@gmx.de

Device Treiber für das DEC RT-11 Betriebssystem (RH.MAC

```
.MCALL .MODULE
MODULE RH,RELEASE=V05,VERSION=5,COMMENT=<Q-Bus - CTI-Bus Driver for RD50>
;
.iif ndf mmg$t, mmg$t = 1
.iif ndf tim$it, tim$it = 1
;
.sbttl description
;+
; This RT-11 handler is meant to act as either a system or non-system device-
; handler. It will support a special HW , CTI-/Q-bus converter, to connect
; the RD50 or RD51(not improved yet) with the PC300 wini controller to the
; Q-Bus. This special HW is responsible to convert the Q-Bus timing to the
; CTI-Bus timing, This means it will provide all CTI-bus handshake signals
; ( e.g. SS,M DEN,S DEN,D S,A S,...) as a result of the Q-Bus signals.
; Additional the Q-Bus/CTI-Bus converter includes 2 octal Bus Transceivers
; for reduceing the Q-bus AC-loads .
; The interrupt controll on the Q-Bus/CTI-Bus converter is realized complete
; different as on the PC-Motherboard .On the PC mother board two special
; interrupt controller are assigned to controll the interrupt requests
; from the RD50-controller.
; In general, the Q-BUS/CTI-Bus converter is doing this with the Q-Bus
; protocoll chips DC003/DC005 to provide the Q-bus interrupt standarts.
; For the I/O controll there are 7 i/o registers available . The Converter
; is able to provide 40 (174000-174036) i/o registers. 2 registers are
; additional used for interrupt controll.
; CSR: 174030 = OP interrupt at vector 350
; CSR: 174034 = DRQ interrupt at vector 354
; This register are simple flip/flop register. Every access to this
; registers will change the interrupt condition, on or off .
; This driver is not able to support error logging.
;
;
;
| |
| | |-----| |-----| |-----|
| |-----| Q-BUS/|-----|PC 350 |-----| WINI |
| |-----|CTI-BUS|-----|CONTR. |-----| RD50 |
| | |-----| |-----| |-----|
;
;
by: R.Heuberger
;
; Modified for V5.4 compatibility and fix logical to physical block
; conversion according to V5.4 releasenotes by H.M.
;
; all changes are marked by " H.M. "
;-
; The extended handler functions are as follows:
;
CODE NAME FUNCTION
---- ----
377 ABSRW Read absolute sector (block) - read any block on the
device, including physical block 0 and the last cylinder
376 ABSRW Write absolute sector (block) - write any block on the
device, including physical block 0 and the last cylinder
;
Disk Data
;
Bytes/sector 512.
Sectors/track 16.
Heads 4
Cylinders 153.
```



```

; Tracks          612.
;
; 1 sector = 1 block
;-
    .SBTTL  MACRO DEFINITIONS
;+
; Macro to generate the maximum number of entities that can be counted
; by a field the specified number of bits in size
;
; Calling sequence:
;
;     MAXVAL BITS, VAL
;
; Where:
;
;     BITS          Inputs the number of bits in the bit field
;
;     VAL           Returns the largest unsigned number that can be
;                   contained in the bit field + 1, i.e., the number
;                   of entities that the field can count
;-
.MACRO  MAXVAL BITS, VAL
VAL     = 1
.REPT   BITS
VAL     = VAL * 2
.ENDR
.ENDM
;+
; Macro to generate a bit mask for accessing ID registers
;
; Calling sequence:
;
;     BITMSK  ARG, CMASK, BITS
;
; Where:
;
;     ARG          Inputs the maximum supported value for an ID
;
;     CMASK        Returns the logical complement of value "MASK - 1",
;                   where "MASK" is the lowest number that is a power of
;                   2 and is greater than ARG. Thus, "MASK - 1" contains
;                   a right-justified field of contiguous 1's, with 0's in
;                   the higher order bits, such that:
;
;                   ARG .AND. (MASK-1) = ARG
;
;                   but
;
;                   (ARG*N) .AND. (MASK-1) <> (ARG*N)
;
;                   where N >= 2.
;
;                   "MASK - 1" is complemented to permit its use as a mask
;                   that will pass a right-justified data field in a bit
;                   clear instruction.
;
;     BITS         Returns the logarithm base 2 of MASK, that is, the
;                   number of bits that CMASK will pass.
;-
.MACRO  BITMSK ARG, CMASK, BITS
.IIF EQ ARG,      .ERROR  ;Bitmask argument must be nonzero
N       = ARG      ;Don't actually change ARG
MASK    = 1        ;Initialize mask word
BITS    = 0        ;Initialize masked bits counter
FLAG    = 0        ;Exitloop flag
.REPT   16         ;Test all 16 bits in ARG
.IF EQ FLAG      ;If exitloop has not occurred yet
MASK    = MASK * 2 ;Shift the mask

```

```

BITS      = BITS + 1      ;Increment the bits counter
N          = N / 2        ;Shift ARG
      .IF EQ N            ;If ARG -> 0
FLAG      = 1             ;Exitloop
      .ENDC ;EQ ARG
      .ENDC ;EQ FLAG
      .ENDR              ;End of loop
CMASK      = ^C<MASK - 1> ;Return the actual bit mask, inverted for BIC's
      .ENDM

;
      .SBTTL PRIMARY DECLARATIONS
;
QC$CSR     = 174004        ; Base Device address of Qbus/CTIbus converter
QC$VEC     = 350           ; Base interrupt vector address
QC$STA     = 174020        ; H.M.
QC$OPE     = 174030        ; Q-BUS/CTI-BUS CONVERTERS INTERRUPT CSR
                        ; address for OP ended interrupt. (vec:350)
QC$DRQ     = 174034        ; Q-BUS/CTI-BUS CONVERTERS INTERRUPT CSR
                        ; address for DRQ interrupt. (vec:354)
QC$PRI     = 4             ; Interrupt CONVERTERS interrupting prior.
;
;
;+
; Base addresses for this handler based on the above
;-
RH$CSR     = QC$CSR        ;Base device address for QC controller
RH$VEC     = QC$VEC        ;Base device interrupt vector address for QC
;+
;
      Device size limits and options
;
; For the RD50 , the first block and the last cylinder are reserved
; for the hardware, and cannot be accessed unless you're using the absolute
; read/write SPFUN's.
;-
RH$SEC     = 16.           ;16 sectors per track
RH$SRF     = 4.            ;4 tracks per cylinder (4 heads)
RH$C50     = 153.          ;153 cylinders per unit (RD50)
RH$C51     = 306.          ;306 CYLINDER per unit (RD50)
;+
; Absolute device size limits and their derivatives (defined by the number
; of bits, including those allocated for future expansion, in the ID device
; registers as described in the PC RD50C hardware specification).
;-
SECIDMX    = 5             ;Maximum number of bits in a Sector ID (from spec)
SRFIDMX    = 3             ;Maximum number of bits in a Head ID (from spec)
CYLIDMX    = 10.           ;Maximum number of bits in a Cylinder ID (from spec)
MAXVAL SECIDMX, MAXSEC ;Calculate maximum possible number of sectors
MAXVAL SRFIDMX, MAXSRF ;Calculate maximum possible number of heads
MAXVAL CYLIDMX, MAXCYL ;Calculate maximum possible number of cylinders
;+
; Guarantee that the absolute maximum sized fields can contain the
; values specified at sysgen
;-
.IIF GT <RH$SEC - MAXSEC> .ERROR ;Max Sector ID bit field exceeded
.IIF GT <RH$SRF - MAXSRF> .ERROR ;Max Head ID bit field exceeded
.IIF GT <RH$C50 - MAXCYL> .ERROR ;Max Cylinder ID bit field
.IIF GT <RH$C51 - MAXCYL> .ERROR ;Max Cylinder ID Bit field
;+
; Other primary definitions for the handler's preamble
;-
ASIZ50     = RH$SEC*RH$SRF*RH$C50 ;Absolute device size in blocks (RD50)
USIZ50     = <RH$SEC*RH$SRF*<RH$C50-1>>-1 ;Usable device size in blocks (RD50)
ASIZ51     = RH$SEC*RH$SRF*RH$C51 ;Absolute device size in blocks (RD51)
USIZ51     = <RH$SEC*RH$SRF*<RH$C51-1>>-1 ;Usable device size in blocks (RD51)
;
;
      .SBTTL PREAMBLE SECTION
;
;+

```



```

; I/O queue element format:
;
; NAME          OFFSET          CONTENTS
;
; Q.LINK        0              Link to next queue element
; Q.CSW         2              Pointer to channel status word
; Q.BLKN        4              Block number
; Q.FUNC        6              Function code (bits 7-0 of low order byte)
; Q.JNUM        7              Job number (bits 6-3 of high order byte)
; Q.UNIT        7              Device unit (bits 2-0 of high order byte)
; Q.BUFF        10             Virtual address of user buffer
; Q.WCNT        12             Word count (<0 for Write, >0 for Read)
; Q.COMP        14             Completion routine address code
; Q.PAR         16             PAR1 value to map user buffer
;
; The .DRDEF macro defines the following identification codes, offsets, and
; bit patterns:
;
; Base CSR and vector addresses:
;
;     RH$CSR      The address declared in the definition of RH$CSR
;     RH$VEC      The address declared in the definition of RH$VEC
;
; Offsets into the current I/O queue element, relative to the pointer that the
; monitor leaves in handler address RHCQE:
;
;     NAME          USE
;
;     Q$LINK        Offset from queue element pointer to Q.LINK
;     Q$CSW         Offset from queue element pointer to Q.CSW
;     Q$BLKN        Offset from queue element pointer to Q.BLKN
;     Q$FUNC        Offset from queue element pointer to Q.FUNC
;     Q$JNUM        Offset from queue element pointer to Q.JNUM
;     Q$UNIT        Offset from queue element pointer to Q.UNIT
;     Q$BUFF        Offset from queue element pointer to Q.BUFF
;     Q$WCNT        Offset from queue element pointer to Q.WCNT
;     Q$COMP        Offset from queue element pointer to Q.COMP
;     Q$PAR         Offset from queue element pointer to Q.PAR
;
; Bit masks for the Channel Status Word:
;
;     HDERR$        Hard error bit in the CSW
;     EOF$          End of file bit in the CSW
;
; Identification codes for the header:
;
;     RHDSIZ        Device size for .DSTATUS
;     RH$COD        Device identifier code
;     RHSTS         Device status word
;
; Bits in the device status word:
;
;     VAR$          SPFUN 373 request is valid
;     SPFUN$        Special functions supported
;     HNLDR$        Abort on job aborts
;     SPECL$        Special directory-structured device
;     WONLY$        Write-only device
;     RONLY$        Read-only device
;     FILST$        Sequential/random access device
;-
;
; .MCALL .DRDEF, .ASSUME
;
;+
; Define the standard preamble for the following parameters:
;
; Device name          -          RH
; Device identifier    -          53
; Device status        -          RANDOM ACCESS, SPFUN, VARIABLE SIZED

```

```

; Device size - RHSIZ
; Default CSR address - RH$CSR
; Default vector address - RH$VEC
;-
;
; .DRDEF RH,200,FILST$!SPFUN$!VARSZ$,USIZ50,RH$CSR,RH$VEC
;
; .DRPTR FETCH = 0 ; V5.4 add on by H.M.
; .DREST CLASS = DVC.DK ; V5.4 add on by H.M.
; .DRSPF <373> ; V5.4 add on by H.M.
; .DRSPF <376> ; V5.4 add on by H.M.
; .DRSPF <377> ; V5.4 add on by H.M.
;
; .SBTTL CONSTANT DEFINITIONS
;
SYSPTR = 54 ;Pointer to base of RMON
P1EXT = 432 ;Offset from $RMON to external routine
;
;+
; CSR read bits
;-
CMD0 = 1 ;Command 0
CMD1 = 2 ;Command 1
CMD2 = 4 ;Command 2
CMD3 = 10 ;Command 3
CMD4 = 20 ;Command 4
CMD5 = 40 ;Command 5
CMD6 = 100 ;Command 6
CMD7 = 200 ;Command 7
;+
; Define the CSR addresses for the interrupt controll -CSR that
; service the OP ENDED interrupt and the DATA REQUEST interrupt
;-
ICOPND = QC$ope ;OP ENDED handled by VECTOR: 350
ICDRQ = QC$drq ;DRQ handled by VECTOR: 354
;+
; Define the device register addresses as offsets from the device CSR
; address RH$CSR.
;-
RHERR = 0 ;ERROR/PRECOMP register (Logged)
RHPRE = 0 ;
RHREV = 2 ;BACKUP REV/SECTOR ID register
RHSEC = 2 ; (Logged)
RHBUF = 4 ;DATA BUFFER register
RHCYL = 6 ;CYLINDER ID register (Logged)
RHHEAD = 10 ;HEAD ID register (Logged)
RHST2 = 12 ;STA 2/COMMAND register (Logged)
RHCMD = 12 ;
RHSTAT = 14 ;STATUS/INIT register (Logged)
RHINIT = 14 ;
;
;+
; Define the ERROR/PRECOMP register error bits.
;-
ERHM = 0400 ;Data Mark not found during read sector
ERTRO = 01000 ;Track 0 not found during restore
ERABO = 02000 ;Illegal/aborted command
ERIDNF = 10000 ;Sector not found
ERICRC = 20000 ;ID CRC error
ERHCRC = 40000 ;Data CRC error
;+
; Define the BACKUP REV/SECTOR ID register Sector ID bits.
;-
BITMSK RH$SEC-1,SECMSK,SECBIT ;Invoke the bit mask generating macro
BITMSK MAXSEC-1,MXSECM,MXSECB ;Invoke the bit mask generating macro
.IIF NE <SECIDMX-MXSECB> .ERROR ;Bitmask macro failed
;+
; Define the HEAD ID register bits.

```



```

;-
BITMSK RH$SRF-1,SRFMSK,SRFBIT ;Invoke the bit mask generating macro
BITMSK MAXSRF-1,MXSRFM,MXSRFB ;Invoke the bit mask generating macro
.IIF NE <SRFIDMX-MXSRFB> .ERROR ;Bitmask macro failed
;+
; Define the CYLINDER ID register bits
;-
BITMSK RH$C51-1,CYLMSK,CYLBIT ;Invoke the bit mask generating macro
BITMSK MAXCYL-1,MXCYLM,MXCYLB ;Invoke the bit mask generating macro
.IIF NE <CYLIDMX-MXCYLB> .ERROR ;Bitmask macro failed
;+
; Define the STA 2/COMMAND register status bits.
;-
S2ERR = 400 ;Error status valid in ERROR/PRECOMP register
S2DRQ = 4000 ;Data transfer request
S2SEK = 10000 ;Seek complete
S2WRF = 20000 ;Write fault
S2RHY = 40000 ;Drive ready
;+
; Define the STA 2/COMMAND register command bits.
;-
CMREST = 20 ;Restore command
CMREAD = 40 ;Read sector command
CMWRIT = 60 ;Write sector command
;+
; Define the STATUS/INIT register status bits.
;-
STOPND = 1 ;Operation ended
STDRQ = 200 ;Data transfer request
STDCAP = 400 ;Drive capacity (1 = 5 Meg, 0 = 10 Meg)
STBUSY = 100000 ;Busy (controller's internal bus in use)
;+
; Define the STATUS/INIT register init bits.
;-
STINIT = 10 ;Reset/initialize
;+
; Expected value of drive capacity bit in STATUS/INIT register
;-
.IF GT <RHSIZ - <10. * 1024.>> ;If capacity .GT. 10K blocks (5Mbytes)
;RHCAP = 0 ;Drive capacity bit should be 0
;.IFF ;Else this is a 5Mbyte drive
;RHCAP = STDCAP ;Drive capacity bit should be 1
;.ENDC ;Endif
;+
; Other constants
;-
IDBITS = <CYLBIT+SRFBIT+SECBIT> ;Total size of the ID bit fields
.IIF GT <IDBITS-16.>, .ERROR ;Block number can exceed 16 bits!
RETRY = 8. ;Allow 8 tries (7 retries) per operation
WREQ = 100000 ;Bit 15 - Write request flag
;
.SBTTL INSTALLATION VERIFICATION ROUTINE
;
.ASECT ;Routine goes in block 0 of handler file
. = 200 ; at location 200
;
NOP ;Same test for system & non-system device
1$: MOV #RH$CSR,R0 ;Base CSR address
TST RHSTAT(R0) ;Is controller busy (busy bit (15) set)?
.ASSUME <STBUSY> EQ 100000
BMI 1$ ;Busy
RETURN
;
;
.SBTTL HEADER SECTION
;
;+
; Create the header
;-

```

```

        .DRBEG  RH
        ;
        BR      RHENT          ;Branch over the protection table
RHWPRO: .WORD   0              ;Unit 0
DRETRY: .WORD   RETRYS         ;Number of retries - default is 8
        ;
        ;
        .SBTTL  I/O INITIATION SECTION
        ;
RHENT:   ;-----xm-----
        .IF NE MMG$T
        MOV     @#SYSPTR,R4    ; R4 -> monitor base
        MOV     P1EXT(R4), (PC)+ ; Get address of externalization routine
$P1EXT:  .WORD   P1EXT          ; Pointer to externalization routine
        .ENDC ;NE MMG$T
        ;-----xm-----
        MOV     #RH$CSR,R4     ;Point to QC device registers
        TST     RHSTAT(R4)     ;Is controller busy (busy bit (15) set)?
        .ASSUME <STBUSY> EQ 100000
        BMI     JMPFD1         ;Fatal error - Nothing's been initiated
        ASL     (PC)+          ;Do we know what we're on?
        .WORD   100000
        BCC     10$            ;If yes, branch
        BIT     #STDCAP,RHSTAT(R4) ; RD50 ??
        BNE     10$
        MOV     #RH$C51,RH$CYL ; RD51 CYLINDER
        MOV     #ASIZ51,RHASIZ ; RD51 ABSOLUTE SIZE
        MOV     #USIZ51,RHUSIZ ; RD51 AVAILABLE SIZE
10$:     MOV     DRETRY,RETRY   ;Initialize retry count
        CLR     INREST        ;Clear restore in progress flag
        CLR     FNFLAG        ;Clear flag assuming a special function
        CLR     WRFLAG        ;Clear write fault catcher bit
        MOV     RHCQE,R5      ;Point to Q$BLKN
        MOV     @R5,R3        ;Get block number
        ;
        ;+
        ; Extract the unit number and check if legal.
        ;-
        ;
        MOV     Q$UNIT(R5),R0  ;Get the unit/job number byte
        BIC     #^C<7>,R0      ;Mask in just the unit number bits
        BNE     JMPFU1         ;Fatal user error (only unit 0 supported)
        ;
        MOV     Q$FUNC(R5),R0  ;Get the function code
        ADD     #FNEG,R0       ;Normalize to >= 0
        CMP     R0,#FNUM       ;Compare with total number of codes
        BLO     DISPAT         ;Within range (0 =< R0 < FNUM)
JMPFU1:  JMP     RHFUE          ;Relay to fatal user error
JMPFD1:  JMP     RHFDE          ;Relay to fatal device error
        ;
        ;
        .SBTTL  LOCAL VARIABLES
        ;
drqon:   .word   0              ; SW - register for DRQ-interrupt
opeon:   .word   0              ; SW - register for OP- interrupt
RETRY:   .WORD   0              ;Retry count
INREST:  .WORD   0              ;Flag that a restore operation is in progress
FNFLAG:  .WORD   0              ;Flag to interrupt routine of function type
WRFLAG:  .WORD   0              ;Flag to intercept write fault error condition
WCNT:    .WORD   0              ;Copy of Q$WCNT(R5)
CRNTCYL: .WORD   0              ;Copy of current Cyl. ID for retries & wchk
CRNTHD:  .WORD   0              ;Copy of current Head ID for retries & wchk
CRNTSEC: .WORD   0              ;Copy of current Sector ID for retries & wchk
RHASIZ:  .WORD   ASIZ50        ;Absolute device size (RD50 default)
RHUSIZ:  .WORD   USIZ50        ;Available device size (RD50 default)
RH$CYL:  .WORD   RH$C50        ;Cylinders/unit (RD50 default)
        .IF EQ MMG$T          ;If not extended memory system = SJ +F/B
BUFF:    .WORD   0              ;Copy of Q$BUFF(R5)
        .ENDC ;EQ MMG$T

```



```

;
;
.SBTTL VECTOR TABLE
;
;+
; Create table to associate vectors with interrupt entry points
; Note that both interrupts vector to the same entry point
;-
;
; .DRVTB RH,RH$VEC,RHINT ;DRQ ENDED interrupt
; .DRVTB ,RH$VEC+4,RHINT ;OP ENDED interrupt
;
;
.SBTTL COMMAND DISPATCH
;
DISPAT: ASL R0 ;Make function code into word offset
ADD PC,R0 ;Add PIC address of next instruction
ADD #FNTBL-.,R0 ;Add offset from here to FNTBL
ADD @R0,PC ;Add table entry to PC (PC -> FNTBL)
JMPREF: ;This label must follow "ADD @R0,PC"
;
FNTBL:
.WORD <DSIZ - JMPREF> ;#-5 (373) Determine device size
.WORD <RHFUE- JMPREF> ;#-4 Unused slot
.WORD 0. ;#-3 (375) Format track
.WORD <ABSW - JMPREF> ;#-2 (376) Write absolute sector
.WORD <ABSR - JMPREF> ;#-1 (377) Read absolute sector
;
;+
; READ(X)/WRIT(X) function goes here. Label must stay with code #0.
;-
;
FTBZER: .WORD <RW - JMPREF> ;#0 READ(X)/WRIT(X)
;
;+
; Any positive functions can go here, before label.
;-
;
FTBEND: ;Last table entry + 2
;
;+
; Dispatch table statistics to check for legal function code when a
; request is made (2 bytes per table entry)
;-
;
FNEG = <FTBZER - FNTBL>/2 ;Number of .SPFUN requests,
;including unused slots
FNUM = <FTBEND - FNTBL>/2 ;Total number of requests,
;
;
.SBTTL HANDLER FUNCTIONS SECTION
;
;
.SBTTL ABSRW - Read/write absolute sector
;
;+
; These two SPFUN's allow the user to access any block (sector) on the
; device. This includes the "hidden" block 0, and the last cylinder,
; which is reserved for bad block replacement.
;-
;
ABSW: BIS #100000,ABSSEC ;High bit set for absolute write
NEG Q$WCNT(R5) ;Make the word count negative for a write
ABSR: BIS #1,(PC)+ ;High bit clear, low bit set for absolute read
ABSSEC: .WORD 0 ;Nonzero for absolute read or write
;
;
.SBTTL - READ(X)/WRIT(X) FUNCTION
;

```

```

RW:    BIT      #1,Q$BUFF(R5)      ;Is user buffer word-aligned?
      BNE      JMPFUE              ;User error - Word-aligned device
      MOV      #1,FNFLAG           ;Set flag assuming a read request
      TST      Q$WCNT(R5)          ;Check sign of word count
      BMI      1$                  ;Word count is negative for a write
      BNE      RWNEXT              ;Word count is positive for a read
      JMP      RHRQS               ;No data - Exit (with no error)
      ;
1$:    BIT      #S2WRF,RHST2(R4)    ;Has a write fault occurred?
      BNE      JMPFDE              ;Yes - Can't write until next power-up
      ;
; HERE TO CHECK IF UNIT IS WRITE-PROTECTED
      ;
      ASL      (PC)+                ;Check write anyway one-shot
RHW1:  .WORD    .-                  ; 100000 means write anyway
      .ASSUME  . LE RHSTRT+1000
      BCS      10$                  ;Skip test if write anyway
      TST      RHWPRO               ;Are we write-locked?
      BNE      JMPFUE              ;If yes, branch
10$:   MOV      #S2WRF,WRFLAG       ;Set flag to catch write fault condition
      MOV      #WREQ,FNFLAG        ;Set flag for write request
      NEG      Q$WCNT(R5)          ;Get the real word count for a write
      ;
;+
; Entry points for next block after successful completion of previous
; block during a multiblock READ(X)/WRIT(X)
;-
      ;
RWNEXT: inc     r3                  ;block # log => phy    -- H.M.
      TST      ABSSEC               ;Is this an absolute sector read/write?
      BEQ      2$                   ;If not, branch
      CMP      R3,RHASIZ            ;Is the requested block on the device?
      BHIS     JMPFUE              ;User error - Block number too large
      BR       25$                  ;Merge below
      ;
2$:    CMP      R3,RHUSIZ            ;Is the requested block on the device?
      BHIS     JMPFUE              ;User error - Block number too large
25$:   mov      r3,r1                ;Copy block number    -- H.M.
      BIC      #SECMASK,R1          ;Mask out all but Sector ID
      ;
      .REPT    SECBIT               ;SECBIT shifts to get R3/RH$SEC
      ASR      R3                   ;Divide by 2
      .ENDR
      ;
      MOV      R3,R2                ;Copy Cylinder & Head ID bits
      BIC      #SRFMSK,R2           ;Mask out all but Head ID
      ;
      .REPT    SRFBIT               ;SRFBIT shifts to get R3/RH$SRF = Cyl ID
      ASR      R3                   ;Divide by 2
      .ENDR
      ;
      BIC      #CYLSK,R3            ;Remove any propagated sign bit
      CMP      R3,RH$CYL            ;Trying to seek beyond highest track?
      BHIS     JMPFUE              ;User error - Cylinder ID too large
      MOV      R3,CRNTCYL           ;Save current values of Cylinder, Head,
      MOV      R2,CRNTHD            ; and Sector ID'S so calculations aren't
      MOV      R1,CRNTSEC           ; repeated during retries & write checks
      ;
;+
; Entry point for READ(X)/WRIT(X) retries
;-
      ;
RWRTRY: MOV      CRNTCYL,RHCYL(R4)  ;Load Cylinder ID register
      MOV      CRNTHD,RHHEAD(R4)  ;Load Head ID register
      MOV      CRNTSEC,RHSEC(R4)   ;Load Sector ID register
      ;*****
1$:    tst      @#QC$STA             ; H.M.
      bmi      1$                   ;
      TST      FNFLAG               ;Read or write request?

```



```

    BMI      3$                ;Request is for a write
    TST      @#QC$drq          ; *** TURN -DRQ-INTERRUPT ON ***
    inc      drqon              ; and set SW register
    ;*****
    MOV      #CMREAD,RHCMD(R4) ;Issue the read command
    BR       4$                ;Wait for seek and read to complete
    ;*****
3$:   TST      @#QC$OPE         ; *** TURN -OP- INTERRUPT ON ***
    inc      opeon              ; and set SW register
    ;*****
    MOV      #CMWRIT,RHCMD(R4) ;Issue the write command
    JSR      R0,MTFILL          ;Fill the controller's sector buffer
    MOV      (R2)+,@R4          ;Buffer fill instruction
    CLR      @R4                ;Buffer zero-fill instruction
4$:   CLR      ABSSEC           ;Clear abs sector read/write flag
    RETURN
    ;
JMPFDE: JMP      RHFDE          ;Relay to fatal device error
JMPFUE: JMP      RHFUE          ;Relay to fatal user error
    ;
    ;
.SBTTL     DSIZ - Get disk size
    ;
;+
; DSIZ processes the SPFUN 373 request.
;-
    ;
DSIZ:
    .IF EQ MMG$T
        MOV      RHUSIZ,@Q$BUFF(R5) ;Put usable size into buffer
        ;-----xm-----
    .IFF ;EQ MMG$T
        MOV      R4,-(SP)          ;Save R4
        MOV      RHUSIZ,-(SP)      ;Put usable size on the stack
        MOV      R5,R4             ;Copy queue pointer for $PTWRD
        CALL     @SPTWRD           ;Put size in buffer
        MOV      (SP)+,R4          ;Pop the stack
    .ENDC    ;EQ MMG$T
        ;-----xm-----
        JMP      RHRQS             ;Go out
    ;
.SBTTL     EMPTY/FILL THE CONTROLLER'S SECTOR BUFFER SUBROUTINE
    ;
;
; Inputs:
;
; Word 1 of 2 in-line arguments
;
; MOV      (R2)+,@R4              ;To fill sector buffer from user buffer
;                                  (write)
; MOV      @R4,(R2)+              ;To empty sector buffer into user buffer
;                                  (read)
;
; Word 2 of 2 in-line arguments
;
; CLR      @R4                    ;To zero-fill sector buffer (write)
; MOV      @R4,R5                 ;To drop sector buffer data (read)
;
; R5      - Points to queue element at offset Q$BLKN
; R4      - Points to device registers
; R0      - Subroutine link register
;
; Outputs:
;
; BUFF      - Value of Q$BUFF(R5), incremented by 256., to point
;              at the next block in the user buffer, for
;              unmapped systems only
;
; PAR      - Value of Q$PAR(R5), incremented by 256./32., to

```

```

;                                     point Q$BUFF(R5) at the next 256. word block
;                                     in the user buffer, for mapped systems only
;
;
;      WCNT          - Value of Q$WCNT(R5), decremented by 256., or set to
;                                     zero if it is already less than 256.
;
; Registers changed:
;
;      R1, R2, R3
;-
      .ENABL  LSB
;
MTFILL: MOV      R5,-(SP)      ;Save I/O queue element pointer
      MOV      R4,-(SP)      ;Save device register pointer
      MOV      (R0)+,3$      ;Put buffer fill instruction in-line
      MOV      (R0)+,5$      ;Put buffer zero-fill instruction in-line
      MOV      R4,R3         ;Copy device register address
      ADD      #RHSTAT,R3    ;Point to STATUS/INIT register
      ADD      #RHBUF,R4     ;Point to DATA BUFFER register
      MOV      Q$BUFF(R5),R2 ;Get virtual address of user buffer
      MOV      Q$WCNT(R5),R1 ;Get word count
      BEQ      4$            ;Zero - Zero-fill/dump sector buffer
      MOV      R1,WCNT       ;Copy word count for adjusting
      CMP      R1,#256.      ;Word count > 256. ?
      BLOS     1$            ;No
      MOV      #256.,R1      ;Can only do 256. words per seek
1$:    TST      FNFLAG        ;Is this a write?
      BPL      100$          ;If not, branch
      ASL      (PC)+         ;Is this the first write?
      .WORD    100000
      BCS      10$           ;If yes, branch
100$:  SUB      R1,WCNT       ;Adjust saved copy of word count
10$:
;-----x-----
      .IF NE MMG$T          ;If extended memory system
      MOV      Q$PAR(R5),PARVAL ;Get PAR1 mapping for the user buffer
      JSR      R0,@$P1EXT    ;Let the monitor execute the following code.
      .WORD    PARVAL-      ;Number of instructions in bytes plus 2.
;-----x-----
      .IFF
      MOV      R2,BUFF       ;Copy buffer address
      TST      FNFLAG        ;Is this a write?
      BPL      200$          ;If not, branch
      ASL      (PC)+         ;Is this the first write?
      .WORD    100000
      BCS      2$            ;If yes, branch
200$:  ADD      #512.,BUFF    ;Point to next block in buffer for next time
      .ENDC
;
2$:    MOVB     @R3,R5        ;DRQ bit set?
      BPL      2$            ;Not yet
3$:    .WORD    0             ;Insert/remove a word of user data to data reg
      SOB      R1,2$         ;Repeat until word count exhausted
;-----x-----
      .IF NE MMG$T          ;If extended memory system
PARVAL: .WORD    0            ;use this value for the PAR 1 bias.
      TST      FNFLAG        ;Is this a write?
      BPL      30$           ;If not, branch
      ASL      (PC)+         ;Is this the first write?
      .WORD    100000
      BCS      300$          ;If yes, branch
30$:   ADD      #<256./32.>,PARVAL ;Map to next block in buffer next time
300$:
      .ENDC
;-----x-----
4$:    MOV      @R3,R5        ;BUSY bit set yet?
      BMI      6$            ;Yes - Done transferring data (write)
      ASR      R5            ;OP ENDED bit (bit 0) set yet?
      BCS      6$            ;Yes - Done transferring data (read)

```



```

        ASLB      R5                ;DRQ bit (bit 7) set?
        BPL       4$                ;No
5$:      .WORD    0                ;Zero-fill/dump a word in data buffer register
        BR       4$                ;Continue looping
        ;
6$:      TSTB     @R3                ;DRQ still set????
        BPL      7$                ;No
        HALT                    ;Yes - Find out why!!!!
        ;
7$:      MOV      (SP)+,R4            ;Restore device register pointer
        MOV      (SP)+,R5            ;Restore I/O queue element pointer
        RTS      R0                ;Return
        .DSABL   LSB
        ;
        ;
.SBTTL   DUMP THE CONTROLLER'S SECTOR BUFFER SUBROUTINE
        ;
        ;
DUMP:    MOV      R4,R2                ;Copy device register pointer
        ADD      #RHSTAT,R2          ;Point to STATUS/INIT register
        MOV      R4,R3                ;Copy device register pointer
        ADD      #RHBUF,R3          ;Point to DATA BUFFER register
1$:      MOV      @R2,R1              ;Get STATUS register
        ASR      R1                  ;OP ENDED bit (bit 0) set yet?
        BCS      2$                ;Yes - Done dropping data
        ASLB     R1                  ;DRQ bit (bit 7) set?
        BPL      1$                ;No
        MOV      @R3,R1              ;Drop a word from the sector buffer
        BR       1$                ;Continue until buffer is empty
        ;
2$:      TSTB     @R2                ;DRQ still set????
        BPL      3$                ;No
        HALT                    ;Yes - Find out why!!!!
        ;
3$:      RETURN
        ;
        ;
.SBTTL   INTERRUPT SERVICE SECTION
        ;
INTFDE:  JMP      RHFDE                ;Relay to fatal device error
INTRQS:  JMP      RHRQS                ;Relay to successful completion
        ;
        .DRAST   RH,QC$PRI            ;; (Generate RTS PC for abort entry)
        ;*****
        tst      drqon                ;; Test DRQ-SW- register
        beq      765$                ;; DRQ - interrupt not on,..
        tst      @#qc$drq            ;; TURN -DRQ- interrupt off !,
        clr      drqon                ;; and reset DRQ-SW register
765$:    tst      opeon                ;; Test OP-SW- register
        beq      999$                ;; OP = off ... exit
        tst      @#qc$ope            ;; Turn -OP- interrupt off !,
        clr      opeon                ;; and reset OP-SW register
        ;*****
999$:    .FORK    RHFBLK                ;;Get to FORK level immediately
        MOV      RHCQE,R5              ;Point to queue element for this call
        MOV      #RH$CSR,R4            ;Point to DW device registers
        TST      RHSTAT(R4)            ;BUSY bit set? (It must not be)
        BMI      INTFDE                ;Fatal device error
        BIT      WRFLAG,RHST2(R4)      ;Did write fault occur during this operation?
        BNE      INTFDE                ;Yes - Hard error (can't be reset by software)
        BIT      #S2ERR,RHST2(R4)      ;Error status?
        BNE      4$                ;Yes - Error
        ;
        TST      INREST                ;Did a restore just complete?
        BNE      6$                ;Yes - Continue with soft error
        ;
80$:     MOV      FNFLAG,R0              ;Test function type flag
        BEQ      INTRQS                ;Not a read or write, so it's done
        BMI      3$                ; I/O FUNCTION = WRITE

```

```

        JSR      R0,MTFILL      ;Empty the controller's sector buffer
        MOV      @R4,(R2)+      ;Buffer empty instruction
        MOV      @R4,R5         ;Drop buffer data instruction
3$:      MOV      WCNT,Q$WCNT(R5) ;Update word count
        BEQ      INTRQS         ;Word count -> zero - Request complete
        ;-----xcm-----
        .IF NE MMG$T            ;If extended memory system
        MOV      PARVAL,Q$PAR(R5) ;Update PAR1 mapping for user buffer
        ;-----xcm-----
        .IFF
        MOV      BUFF,Q$BUFF(R5) ;Update user buffer address
        .ENDC
        ;
        TST      FNFLAG         ;Is this a write?
        BPL      30$            ;If not, branch
        ASL      (PC)+          ;Is this the first write?
        .WORD    100000
        BCS      35$            ;If yes, branch
30$:      INC      @R5            ;Update block number
35$:      MOV      @R5,R3         ;Get block number for next pass
        MOV      DRETRY,RETRY    ;Reset retry counter for next block
        JMP      RWNEXT          ;Do next block of request
        ;
4$:      TST      INREST         ;Was a restore in progress?
        BNE      RHFDE          ;Yes - Hard error (even if not TR000)
        ;
        BIT      #<ERTR0!ERABO>,RHERR(R4) ;TR000 and illegal/aborted
        BNE      RHFDE          ;Command are hard errors
        ;
        TSTB     RHSTAT(R4)      ;Is the DRQ bit set?
        BPL      5$            ;No - OK to continue
        CALL     DUMP            ;Empty the sector buffer
5$:      BIT      #ERIDNF,RHERR(R4) ;Sector ID not found?
        BEQ      6$            ;No - Retry the error now
        ;
        MOV      RHST2(R4),R3    ;Get STA 2 information
        BIT      #S2WRF,R3       ;Does a write fault condition exist?
        BNE      RHFDE          ;Yes - Can't do restore, so hard error
        BIC      #^C<S2SEK!S2RHY>,R3 ;Get SEEK COMPLETE and DRIVE READY bits
        CMP      #<S2SEK!S2RHY>,R3 ;^^Are both bits set?
        BNE      RHFDE          ;No - Can't do restore, so hard error
        MOV      #-1,INREST      ;Set the restore in progress flag
        ;*****
666$:      tst      @#QC$STA      ; H.M.
        bmi      666$           ;
        TST      @#QC$ope        ; *** TURN -OP- INTERRUPT ON ***
        inc      opeon          ; and set SW-register
        ;*****
        MOV      #CMREST,RHCMD(R4) ;Issue the restore command
        RETURN                    ;Wait for the OP ENDED interrupt
        ;
6$:      DEC      RETRY          ;Decrement the retry count
        BEQ      RHFDE          ;No more retries left
        CLR      INREST         ;Clear restore in progress flag
        TST      FNFLAG         ;Read/write request?
        BEQ      8$            ;No - Special function
        BPL      7$            ;Read request - Retry it
7$:      JMP      RWRTRY        ;Try this block again
        ;
9$:      MOV      CRNTCYL,RHCYL(R4) ;Load Cylinder ID register
        MOV      CRNTHD,RHHEAD(R4) ;Load Head ID register
        MOV      CRNTSEC,RHSEC(R4) ;Load Sector ID register
        ;*****
855$:      tst      @#QC$STA      ; H.M.
        bmi      855$           ;
        TST      @#QC$drq        ;** TURN -drq- INTERRUPT ON ***
        inc      drqon          ; and set drq-SW register
        ;*****
        MOV      #CMREAD,RHCMD(R4) ;Initiate the read

```



```

        RETURN                                ;Wait for the OP ENDED interrupt
;
8$:      MOVB    Q$FUNC(R5),R0                ;Get the function code
        ADD     #FNEG,R0                     ;Normalize to >= 0
        ASL     R0                           ;Make function code into word offset
        ADD     PC,R0                        ;Add PIC address of next instruction
        ADD     #RTRYTB-.,R0                 ;Add offset from here to RTRYTB
        ADD     @R0,PC                       ;Add table entry to PC (PC -> RTRYTB)
;
RTRYTB:  .WORD   <DSIZ - RTRYTB>              ;#-5 (373) Determine device size
        .WORD   <RHFUE- RTRYTB>              ;#-4 Unused slot
        .WORD   0.                          ;#-3 (375) Format track
        .WORD   <ABSW - RTRYTB>              ;#-2 (376) Write absolute sector
        .WORD   <ABSR - RTRYTB>              ;#-1 (377) Read absolute sector
;
;
;
.SBTTL   I/O COMPLETION SECTION
;
RHFDE:
;+
; Hard user error entry point
;-
;
RHFUE:   BIS     #HDERR$,@Q$CSW(R5) ;Set hard error bit in channel status
;
;
; Successful completion entry point
;-
;
RHRQS:
;+
; Jump to the monitor's completion section
;-
        CLR     ABSSEC                      ;Clear abs sector read/write flag
        .DRFIN  RH
;
;
RHFBLK:  .WORD   0, 0, 0, 0                  ;Fork block
;
;
.SBTTL   BOOTSTRAP READ ROUTINE
;
.DRBOT   RH,BOOT1,READ,CONTROL=<CBUS>
;
. = RHBOOT + 40                            ; Put next instruction in syscom area
BOOT1:   JMP     @#<BOOT-RHBOOT> ; Go to software bootstrap
;
;+
;Bootstrap read routine
;Controller should not set DRQ but can be busy, because the
;Q-Bus and so the Q-Bus/CTI-Bus converter is working faster.
;One reason is the difference between bus-RESET sequence.
;-
.ENABL   LSB
;
. = RHBOOT + 210
READ:    MOV     #RH$CSR,R4                  ;Point to device registers
        TSTB    rhstat(r4)                  ;DRQ-bit set yet ???
        bmi     40$                          ; YEs !! ?? Fatal error
444$:    TST     RHSTAT(R4)                   ;Controller busy?
        BMI     444$                          ;Yes
        ASL     (PC)+                        ;Do we know what we're on?
        .WORD   100000
        BCC     10$                          ;If yes, branch
        BIT     #STDCAP,RHSTAT(R4)           ;RD50?

```

```

        BNE      10$      ;If yes, branch
        MOV      #RH$C51,RH$CY1 ;RD51 cylinders (ignore the last one)
10$:      MOV      R0,-(SP)      ;Save block number
        MOV      R1,-(SP)      ;Save word count
20$:      MOV      @SP,R1      ;Get current word count
        CMP      R1,#256.      ;Word count > 256. ?
        BLOS     30$      ;No
        MOV      #256.,R1      ;Can only do 256. words per seek
30$:      SUB      R1,@SP      ;Adjust word count
        MOV      2(SP),R0      ;Get current block number
        inc      r0      ; H.M.
        INC      2(SP)      ;Increment block number
        MOV      R0,R5      ;Copy block number
        BIC      #SECMASK,R5      ;Mask out all but Sector ID
        MOV      R5,RH$SEC(R4) ;Load Sector ID register
        ;
        .REPT    SECBIT      ;SECBIT shifts to get R0/RH$SEC
        ASR      R0      ;Divide by 2
        .ENDR
        ;
        MOV      R0,R5      ;Copy Cylinder & Head ID bits
        BIC      #SRFMSK,R5      ;Mask out all but Head ID
        MOV      R5,RH$HEAD(R4) ;Load Head ID register
        ;
        .REPT    SRFBIT      ;SRFBIT shifts to get R0/RH$SRF = Cyl ID
        ASR      R0      ;Divide by 2
        .ENDR
        ;
        BIC      #CYLMSK,R0      ;Remove any propagated sign bit
        CMP      R0,(PC)+      ;Trying to seek beyond highest track?
RH$CY1:   .WORD    153.
        BLO      50$      ;Within range
        ;
40$:      JMP      @#<BIOERR-RHBOOT> ;Hard error
        ;
50$:      MOV      R0,RH$CYL(R4) ;Load Cylinder ID register
        MOV      #CM$READ,RH$CMD(R4) ;Issue read command
        MOV      R4,R0      ;Copy device register pointer
        ADD      #RH$STAT,R0      ;Point to STATUS/INIT register
        MOV      R4,R5      ;Copy device register pointer
        ADD      #RH$BUF,R5      ;Point to DATA BUFFER register
60$:      TST      @R0      ;BUSY bit set?
        BMI      60$      ;Yes - Wait for seek to complete
        BIT      #S2ERR,RH$ST2(R4) ;Did the seek fail?
        BNE      40$      ;Yes - Hard error
70$:      TSTB     @R0      ;DRQ bit set?
        BPL      70$      ;No - Wait
        MOV      @R5,(R2)+      ;Transfer a word from controller to RAM
        SOB      R1,70$      ;Repeat until word count exhausted
80$:      BIT      #STOPND,@R0      ;OP ENDED bit set yet?
        BNE      90$      ;Yes - Done transferring data
        TSTB     @R0      ;DRQ bit set?
        BPL      80$      ;No
        TST      @R5      ;Drop a word from controller
        BR       80$      ;Continue
        ;
90$:      TST      @SP      ;Word count zero yet?
        BNE      20$      ;No - Go read next block
        CMP      (SP)+,(SP)+      ;Drop word count and block number
        CLC      ;Clear the carry bit
        RETURN
        ;
        .DSABL   LSB
        ;+
        ;Software bootstrap section
        ;-
        ;
        . = RH$BOOT+500
        ;

```



```

BOOT:  MOV      #10000,SP      ;Set up the stack pointer
        MOV      #2,R0         ;Block number of secondary bootstrap
        MOV      #<4*256.>,R1   ;Word count of 4 blocks (2-5)
        MOV      #1000,R2      ;Memory address of secondary boot (B$BOOT)
        JSR      PC,READ       ;Load the secondary boot
        MOV      #<READ-RHBOOT>,@#B$READ ;Store pointer to read routine
        MOV      #B$DNAM,@#B$DEVN ;Store RAD50 device name
        CLR      @#B$DEVU      ;Store unit number (always 0)
        JMP      @#B$BOOT      ;Enter the secondary boot
        ;
        .DREND  RH
        ;
.END

```

Low-Level Formatierungs Programm für RD50 Hard-Disk (RDFORM.MAC)

```

        .title  RDFORM
        ;+
        ;  Format -Program for the Q-BUS<->CTI-BUS
        ;  converter for the RD50.
        ;
        ;                                R.Heuberger
        ;-
        .mcall  .exit,.print,.INTEN,.PROTECT,.device,.TTYIN
        ;
INT1     = 174030
INT2     = 174034
id       = 174000      ; Id Register (read only )
er       = 174004      ; Error / precomp ( R-error/W-Precomp )
ba       = 174006      ; Backup REV/Sector Id ( Read/Write )
da       = 174010      ; Data Buffer ( Read/Write )
cy       = 174012      ; Cylinder ( Read/Write )
he       = 174014      ; Head ID ( Read/Write )
sta2     = 174016      ; STA2 / comand (R-STA2/ W-Command )
stat     = 174020      ; Status / Init ( R-Status/W-Init )
        ;
A        = 350         ; DRQ - Interrupt vector
B        = 354         ; OP-ended Interrupt vector
DEVPRI   = 4           ; Priority 4 on Q-Bus
PRI7     = 340         ; Priority 7 (interrupt entry)
lf       = 12
        ;
        ;
START:   .PRINT  #HALLO
        .print  #hallo1
        .print  #hallo2
        .print  #hallo3
        .print  #hallo2
        ;
SETUP:   .PROTECT #EMTLST,#A
        BCC     5$
1$:      .PRINT  #PROERR
        .EXIT
5$:      .PROTECT #EMTLST,#B
        BCS     1$
        ;
10$:    MOV      #AINT,@#A
        MOV      #PRI7,@#A+2
        MOV      #BINT,@#B
        MOV      #PRI7,@#B+2
        ;
        .device #emtlst,#indis
        ;

```

```

;+
; Ask user for using interleaved sector formatting.
;-
;
45$: .print #ask
mov #consol,r1
clr r2
50$: .ttyin (r1)+
cmpb #lf,r0
bne 50$
mov #consol,r1
bicb #200,(r1) ; disable lowercase
cmpb (r1),'N
beq noint ; no-interleaved mode
cmpb (r1),'Y
beq inter ; interleaved mode
br 45$
;
;+
; Set up the buffer with data := date 0-->17 for interleaved
; sequence: 0,10,1,11,2,12,3,13,4,14,5,15,6,16,7,17.
; The other part of the buffer must be FILL-Character .
;-
;
inter: mov #buffer,r1 ; R1 is used as pointer
mov #0,(r1)+
mov #10,(r1)+
mov #1,(r1)+
mov #11,(r1)+
mov #2,(r1)+
mov #12,(r1)+
mov #3,(r1)+
mov #13,(r1)+
mov #4,(r1)+
mov #14,(r1)+
mov #5,(r1)+
mov #15,(r1)+
mov #6,(r1)+
mov #16,(r1)+
mov #7,(r1)+
mov #17,(r1)+
br go ; skip noninterleaved mode
;
;+
; Setup for buffer if no interleaved sequence is used
;-
;
noint: mov #buffer,r1 ; R1 is used as pointer
clr count ; clear counter
10$: mov count,(r1)+ ; Mov counter contents to buffer
inc count ; and increment counter.
cmp count,#20 ; Limit for sectors reached ??
beq go ; Yes.
br 10$ ; No, not yet.
;
go: mov #buffer,point ; Prepare
clr drqok ; some values for
clr count ; correct run-down.
clr line ;
;
TST @#INT1 ; turn interrupt on
TST @#INT2 ; " " "
;
;+
; format-PROGRAM starts here with retry capability.
; In case of an Error , the program will exexecute
; a maximum of 5 Retries with last parameters untill
; the retry-Flag is set to zero.
;-

```

```

;
reset:  clr    flag
        clr    cylind
        clr    head
exe:    mov     #buffer,point
        clr    flag
        call   busy
        mov     head,@#he      ; select head
        mov     cylind,@#cy    ; set cylinder address
        mov     #120,@#sta2    ; select operation mode = FORMAT
        call   get             ; format one TRACK...
;
        tst     retry          ; should we execute a retry ??
        bne     exe            ; yes , last operation was not o.k
        cmp     #3,head        ; over head limit ??
        beq     10$           ; yes, got next adjustment
        inc     head           ; no, switch to next head
        br      exe            ; and execute operation.
10$:    clrb    head            ; reset of heads,= to head 0
        inc     line           ; Increment line count.
        cmp     line,#81.      ; over one line ??
        bne     15$           ; No, not yet.
        .print  #cr            ; Print a <return>
        clr     line
15$:    .print  #star
        inc     cylind         ; Increament Cylinder
        cmp     #231,cylind    ; Cylinder limit over ???
        bne     exe            ; no ;
;
        .print  #done
        mov     #10,@#stat     ; reset controller
        call   busy
        TST     @#INT1         ; turn interrupt oFF
        TST     @#INT2         ; " " "
        .exit   ; ----- E N D -----
;
;
;+
; Subroutine to receive 256. words from the controller.
; This subroutine includes Error handling and retries
; 5 times if a problem in a read operation occurs.
;-
get:    tst     flag           ; This is the main loop of this subroutine.
        bmi     200$          ; This loop is so long active as a interrupt
        beq     get           ; occurs from the OP ended interrupt routine
                                ; or an ERROR has ocured
;
        cmp     count,#256.    ; Do we really received 256. words?
        beq     10$           ; Yes, branch to continue.
6$:     .PRINT  #NONR          ; give a warning message and
        br      200$           ; execute a retry !
10$:    mov     #buffer,point   ; reset buffer-pointer=r3
        clr     drqok          ; prepare "drqok" and count
        clr     count          ; for next operation.
        clr     retry          ; clear retry-flag
        br      300$           ; and loop.
;+
; Error/Retry handling part of this subroutine
;-
200$:   call   busy             ; wait untill controller is ready.
        TST     @#INT1         ; turn interrupt oFF
        TST     @#INT2         ; " " "
        Mov     #10,@#stat     ; reset the controller
        call   busy           ; and wait untill reset is done.
        cmp     #5,retry       ; retry-limit reached ??
        beq     210$          ; yes -- next step
        inc     retry          ; retry counter incremented
        br      250$
210$:   clr     Retry           ; reset retry-count

```



```

        .print #ioerr
250$:   TST     @#INT1          ; turn interrupt on
        TST     @#INT2          ; " " "
300$:   return
        ;
        ;+
        ;
        ;               INTERRUPT Service Routines
        ;
        ; OP-ended Interrupt service routine , initialized
        ; if one TRACK is complete formatted .
        ; -
AINT:   .INTEN  DEVPRI          ; switch to correct priority.
        MOV     @#STA2,end      ; get status .. tranfer completed o.k ??
        bit     #400,end        ; CHECK error bit and branch
        beq     20$             ; in case of error-free to 20$
        mov     #100000,flag     ; E R R O R:-> set flag to minus
        br      30$             ; and exit !!
20$:   inc     flag
30$:   RTS PC                   ; Return....
        ;
        ;+
        ; DRQ-Interrupt service routine to to format one TRACK.
        ; -
BINT:   .INTEN  DEVPRI          ; Switch to correct priority.
        mov     @#sta2,drqok    ; get controller status
        bit     #400,drqok      ; test for an error = bit *
        beq     20$             ; not set,all O. K. =continue
        mov     #100000,flag     ; set flag for error (=minus)
        br      30$             ; and exit
20$:   mov     @point,r4         ; obtain next value
        mov     r4,@#da         ; and bring it to I/O-Buffer
        inc     count           ; Increment counter.
        inc     point           ; update pointer
        inc     point           ; to buffer
30$:   RTS PC                   ; Return.....
        ;
        ;+
        ; Subroutine to wait for ready of the controller
        ; -
busy:   tst     @#stat          ; test for bit 15
        bmi     busy            ; if set = busy than loop
        return
        ;
        ;
        .EVEN
EMTLST: .BLKW    6               ; Interrupt EMT - List
retry:  .word    0               ; ERROR-Retry lag
head:   .word    0               ; Head-select-word
cylind: .word    0               ; Cylinder select word
buffer: .blkw    256.            ; Buffer for one sector contens.
consol: .blkb    80.
point:  .word    0               ; pointer- word for I/O buffer
drqok:  .word    0               ; word to test for successfull I/O
count:  .word    0               ; counter
end:    .word    0
flag:   .word    0
temp:   .word    0
line:   .word    0
mode:   .word    0
indis:  .word    id              ; disable interrupt in case of exit
        .word    0
        .even
HALLO:  .ASCIZ<33>/[2J INTERRUPT-driven Q-BUS<->CTI-Bus FORMATTER PROGRam /
hallo1: .asciz<12><15>/          for the RD50 5-Mbyte Winchester-Drive/<12><15>
hallo2: .asciz/*****
hallo3: .asciz/***** All Data will be destroyed !!! *****/
PROERR: .ASCIZ<33>/[2J          P R O T E C T   ERROR !!!/
ask:    .asciz<12><15>/Do you want to format with interleaved Sectors?../<200>
ioerr:  .asciz<7>/ I-O tranfer ERROR,... 5 Retries without success !! /

```

```

NONR:  .ASCIZ<7>/ Illegal numbers of data-transfers received !!/
star:  .asciz/*/<200>
cr:    .asciz<15><200>
done:  .asciz<12><12><15><7>/          F O R M A T   done !!/
      .end      start

```

Testprogramm für RD50 Hard-Disk (RDTEST.MAC)

```

.title RDTEST
;+
;      Test-Program for Write/Read functionlity of
;      the Q-BUS<->CTI-BUS converter for the RD50.
;                      by:      R.Heuberger
;      ||
;      ||      |-----|      |-----|      |-----|
;      ||-----| Q-BUS/|-----|PC 350 |-----| WINI  |
;      ||-----|CTI-BUS|-----|CONTR. |-----| RD50  |
;      ||      |-----|      |-----|      |-----|
;      ||
;-
.mcall .exit,.print,.INTEN,.PROTECT,.Qset
.MCALL .gtim,.TTYOUT,.ttyin,.scca
;
OPEINT = 174030
DRQINT = 174034
id      = 174000      ; Id Register (read only )
er      = 174004      ; Error / precomp ( R-error/W-Precomp )
ba      = 174006      ; Backup REV/Sector Id ( Read/Write )
da      = 174010      ; Data Buffer ( Read/Write )
cy      = 174012      ; Cylinder ( Read/Write )
he      = 174014      ; Head ID ( Read/Write )
sta2    = 174016      ; STA2 / comand (R-STA2/ W-Command )
stat    = 174020      ; Status / Init ( R-Status/W-Init )
;
A       = 350         ; DRQ - Interrupt vector
B       = 354         ; OP-ended Interrupt vector
DEVPRI  = 4           ; Priority 4 on Q-Bus
PRI7    = 340         ; Priority 7 (interrupt entry)
lf      = 12
write   = 60          ; Write MODE
read    = 40          ; Read  MODE
;
;
START:  .PRINT  #HALLO
;
SETUP:  .PROTECT #EMTLST,#A
        BCC     5$
1$:     .PRINT  #PROERR
        .EXIT
5$:     .PROTECT #EMTLST,#B
        BCS     1$
;
10$:    MOV     #AINT,@#A
        MOV     #PRI7,@#A+2
        MOV     #BINT,@#B
        MOV     #PRI7,@#B+2
;
;
mov     #buffer,point      ; setup pointer for buffer
clr     drqok              ; and prepare
clr     count              ; some values.
;

```

```

;+
; Ask for type to test the rd50
;-
25$: .print #simple
mov #consol,r1 ; Setup
clr r2 ; Registers.
30$: .TTYIN (r1)+ ; get one character.
cmpb #lf,r0
bne 30$ ; loop
mov #consol,r1
BICB #200,(R1) ; DISABLE LOWERCASE
cmpb (r1),'N' ; check for 'N'
beq 35$
cmpb (r1),'Y' ; check for 'Y'
beq 40$
br 25$ ; in case of wrong answer
35$: clr branch ; set flag
br 45$ ; and start next question.
40$: inc branch ; set flag
;+
; Ask for writting Wini with test -pattern
;-
45$: .Print #ask ; print Question
mov #consol,r1 ; Setup
clr r2 ; Registers.
50$: .TTYIN (r1)+ ; get one character.
cmpb #lf,r0
bne 50$ ; loop
;+
; Up to here , ^C abort will is inhibit !!
;-
.qset #qelem,#1 ; Allocate another Q-Element (SJ-MONITOR)
.SCCA #marea,#SCCA ; Inhibit ^C^C action by Monitor !!
;
mov #consol,r1
BICB #200,(R1) ; DISABLE LOWERCASE
cmpb (r1),'N' ; check for 'N'
beq 55$
cmpb (r1),'Y' ; check for 'Y'
beq RANDOM
br 45$
55$: .print #wstart
TST @#DRQINT ; *** enable interrupts
TST @#OPEINT ; " "
;
reset: clr flag ; clear contents of
clr LINE ; .
clr cylind ; .
clr head ; .
clr sector ; .
CLR PATER ; . . . and
mov #write,mode ; switch to write operation.
;+
; Write-PROGRAM starts here with retry capability.
; In case of an Error , the program will exexcute
; a maximum of 5 Retries with last parameters untill
; the retry-Flag is set to zero.
;-
exe: TST SCCA ; Check if ^C^C entered...
beq 3$ ; Branch if no = Continue
jmp abo ; abort Program !
3$: clr patter ; clear test pattern word,
add cylind,patter ; to setup with
add head,patter ; new contents, depending
add sector,patter ; of actuall contents.
5$: clr flag
call get ; Write one sector contents.
tst retry ; should we execute a retry ??

```



```

    bne      5$                ; yes , last operation was not o.k
    cmp      #3,head           ; over head limit ??
    beq      10$               ; yes, got next adjustment
    inc      head              ; no, switch to next head
    br       exe               ; and execute operation.
10$:  clr     head              ; reset of heads,= to head 0
    inc      line              ; Increment line-count
    cmp      #81.,line         ; over one line ??
    bne      15$               ; no not yet
    .print   #crlf             ; print a CRLF
    clr      line              ; reset line-counter
15$:  .print   #dot             ; send a '.' for one completed
                                ; sector with all 3 Head.
    inc      sector            ; Increament sector
    cmp      #17,sector        ; sector limit over ???
    bne      exe               ; no , execute with new sector
    clr      sector            ; yes, reset sector
    inc      cylind            ; Increament Cylinder
    cmp      #230,cylind       ; Cylinder limit over ???
    bne      exe               ; no ;
    TST      @#DRQINT          ; *** DISABLE interrupts
    TST      @#OPEINT          ;      "      "
    ;
    ;+
    ; Pack is now written with Test-Pattern. The Exerciser will
    ; start now to execute random seeks and perform a read/compaare
    ; and write-back operation.
    ;-
    ;
random: .print   #test          ; print message for start
    clr      modify            ; use modify as pointer
    Mov      #10,@#stat        ; reset the controller
    call     busy              ; and wait untill ready
    TST      @#DRQINT          ; *** enable interrupts
    TST      @#OPEINT          ;      "      "
exre:  add     #10,modify        ; step 10
    cmp      modify,@#54       ; use monitor-base as max for compare
    bgt      10$               ; limit not reached yes.
    clr      modify
10$:  clr      cylind            ; clean word of cylinder
    clr      head              ;      head
    clr      sector            ;      and sector.
    mov      #read,mode        ; switch to read operation!
    call     time              ; receive ticks .. -->r1
    bic      #177774,r1        ; mask out bit 0 u. 1
    mov      r1,head           ; create random head-select
12$:  call     time              ; receive ticks
    add      @modify,r1        ; add anything
    bic      #177400,r1        ; mask out bit 0-7 +bit 6
    cmp      r1,#227           ; make sure to go not out of limit !
    blos     20$               ; random range ok. if less or equeal
    bic      #177550,r1        ; make sure.
20$:  mov      r1,cylinder       ; create random cylinder-select
25$:  call     time              ; receive ticks
    bic      #177760,r1        ; mask out bit 0-3
    cmp      r1,#16            ; make sure to be in limit
    bgt      25$               ; if not in range , try again
    mov      r1,sector
    ;
50$:  clr      flag             ;
    call     get                ; get ane sector contents
    ;
    tst      retry             ; should we execute a retry ??
    bne      50$               ; yes , last operation was not o.k
    ;
    ;+
    ; One complete sector is now read, contents is in buffer.
    ; This buffer must now be verified for correct data.
    ;-

```

```

;
clr      patter          ; clear test pattern word,
add      cylind,patter   ;   to setup with
add      head,patter     ;   new contents, depending
add      sector,patter   ;   of actual contents.
mov      patter,bapatt   ; Save contents of actual pattern
CALL     COMPARE         ; Compare data
;
Tst      SCCA            ; Check if ^C^C entered
beq      53$             ; Branch if no = continue
jmp      abo             ; Abort Program!!
;+
; Follow routines will only be executed if the user
; has additionally selected the extensive part of this
; test program. This part is doing a write/read and verify of
; the selected sector with '0' and '1'. At the end of this
; program-part the sector will be written and verified
; with the original test-pattern value.
;-
53$:     tst      branch   ; which test-mode is used ??
         beq      exre     ; short -test-mode
55$:     clr      flag
         mov      #177777,patter ; switch to all ones and
         mov      #write,mode   ;   write this on
         call     get           ;   same sector
         tst      retry         ; Should we execute a retry ?
         bne      55$          ; Yes , a error has occurred.
60$:     clr      flag
         mov      #read,mode    ; Let's read this sector
         call     get           ; with test-pattern '177777'
         tst      retry         ; Should we execute a retry ?
         bne      60$          ; Yes , a error has occurred.
         call     compar        ; Compare data !
;+
; Same procedure with test-pattern '0'
;-
65$:     clr      flag
         mov      #0,patter     ; switch to all zeros and
         mov      #write,mode   ;   write this on
         call     get           ;   same sector
         tst      retry         ; Should we execute a retry ?
         bne      65$          ; Yes , a error has occurred.
70$:     clr      flag
         mov      #read,mode    ; Let's read this sector
         call     get           ; with test-pattern '0'
         tst      retry         ; Should we execute a retry ?
         bne      60$          ; Yes , a error has occurred.
         call     compar        ; Compare data !
;+
; Now write back the original Test-pattern
;-
75$:     clr      flag
         mov      bapatt,patter ; switch to Back-UP pattern
         mov      #write,mode   ;   and write this on
         call     get           ;   same sector.
         tst      retry         ; Should we execute a retry ?
         bne      75$          ; Yes , a error has occurred.
80$:     clr      flag
         mov      #read,mode    ; Let's read this sector
         call     get           ; with test-pattern
         tst      retry         ; Should we execute a retry ?
         bne      80$          ; Yes , a error has occurred.
         call     compar        ; Compare data !
;
jmp      exre            ; and loop
;
;
;+
; Subroutine to wait for ready of the controller

```

```

;-
busy:  tst      @#stat          ; test for bit 15
      bmi      busy           ; if set = busy than loop
      return
      ;
      ;
      ;+
      ; Subroutine to receive time in ticks, neccessary to
      ; generate a random number .
      ;-
time:  Mov      #ticks,r1       ; r1 points to where to put time
      .gtim    #timare,r1      ; get ticks since midnight via .gtim
      mov      (r1)+,r0        ; r0 = lo order time
      mov      @r1,r1          ; r1 = hi order time
      return      ; exit ..
      ;
      ;
      ;+
      ; ASCII Conversion and Print subroutine for one octal
      ; word on the screen with .print
      ; Input is r4 , output = 'ocnum' - r2 used as pointer.
      ;-
ASCII: mov      #ocnum+10,r2    ; Setup r2 as pointer.
      mov      #200,-(r2)      ; Set terminator for .Print
      mov      #5, chcou       ; Setup loop ount.
10$:   mov      r4,-(sp)        ; Copy word into Stack.
      bic      #177770,@sp     ; select one octal value
      add      #'0,@sp         ; CONVERT to ASCII
      movb     (sp)+,-(r2)      ; bring one Ascii to buffer.
      asr      r4              ; Shift
      asr      r4              ;      right
      asr      r4              ;      three.
      dec      chcou           ; Test if done.
      bne     10$              ; No, do it again.
      bic      #177776,r4      ; Get last bit
      add      #'0,r4          ; and convert to ASCII.
      movb     r4,-(r2)        ; Bring last ASCII into buffer,
      movb     #' ',-(r2)      ; including a blank,
      .print   #ocnum          ; print complete oct. value
      RETURN                  ; and exit from subroutine.
      ;
      ;
      ;+
      ; Subroutine to compare the contents of one sector
      ; with the correct test-pattern. In case of error, all
      ; information about good and bad Data will be printed.
      ;-
compar: TST      @#DRQINT       ; *** DISable interrupts
      TST      @#OPEINT       ; " "
      mov      #buffer,compoi   ; compoi is used as pointer for buffer
      clr      comcou          ; r1 is used as counter
20$:   cmp      @compoi,patter   ; make one compare!!
      beq      100$            ; All o.k here !
      ;
      ; ERROR: Bad Data received on actual sector
      ;
      inc      baddat          ; Increment 'Bad-Data-Counter'
      tst      datflg          ; should we print the Header ??
      bne     50$              ; No, it was done before
      .print   #newln          ; Print a ( cr/lf ) and
      .print   #dattxt         ; print header message.
      call    errinf           ; Give Header info were we are!!
      .print   #newln
      .print   #dathea          ; Print second header
      .print   #dathe2         ; " " "
50$:   .print   #newln
      .print   #space          ; print a space
      mov      comcou,r4       ; prepare r4 to number
      call    ascii            ; of bad word.

```



```

        .print #space          ; Print a space
        mov    patter,r4       ; Prepare r4 to print
        call   ascii           ; the good data
        .print #space          ; Print a space
        mov    @compoi,r4      ; Prepare r4 to print
        call   ascii           ; the bad data
        inc    datflg          ; make sure to print the header
        ;                               ; not again.
100$:    inc    compoi          ; increment pointer twice
        inc    compoi          ; for word addressing.
        inc    COMCOU          ; INCREMENT counter( 0--->256.)
        cmp    comcou,#256.    ; over limit ?
        beq    120$            ; Yes, exit this routine.
        br     20$             ; NO, loop until finisch
120$:    clr    datflg          ; reset temp. flag
        TST    @#DRQINT        ; *** enable interrupts
        TST    @#OPEINT        ; " "
        return
        ;
        ;
        ;+
        ; Subroutine to print all information about I/o register
        ; contents in case of E R R O R.
        ;-
errinf:  .print #funk          ; Start to print I-O mode.
        bit    #20,mode        ; Was it a read or a write ??
        bne    30$             ; it was a write .
        .print #RE             ; Print : read funktion
        br     40$             ; and skip
30$:    .print #wr             ; Print : write funktion
40$:    inc    soft            ; increment soft error count
        .print #cylerr         ; Start to give info about cylinder.
        mov    cylind,r4       ; Prepare r4 and
        call   ascii           ; print cylinder ID.
        .print #secerr         ; Start to give info about sector.
        mov    sector,r4       ; Prepare r4 and
        call   ascii           ; print sector ID.
        .print #heaerr         ; Start to give info about Head.
        mov    head,r4         ; Prepare r4 and
        call   ascii           ; print head ID.
        return
        ;
        ;
        ;+
        ; Subroutine to print the contents of the ERROR/PRECOMP
        ; and STA2/COMMAND register.
        ;-
reginf:  .print #ererp          ; Start to give info about ERROR/PRECOMP reg.
        mov    erprin,r4       ; Prepare r4 to print contents of
        call   ascii           ; ERROR/PRECOMP register.
        .print #ersto          ; Start to give info about STA2/COMMAND reg.
        mov    stcoin,r4       ; Prepare r4 to print contents of
        call   ascii           ; STA2/COMMAND register.
        .print #newln          ; bring cursor to new line
        RETURN
        ;
        ;
        ;+
        ; Subroutine to receive 256. words from the controller.
        ; This subroutine includes Error handling and retries
        ; 5 times if a problem in a read/write operation occurs.
        ;-
get:     call   busy           ; wait until controller is ready
        mov    cylind,@#cy      ; set cylinder !
        mov    sector,@#ba      ; set sector !
        mov    head,@#he        ; select head !
        mov    mode,@#sta2       ; select operation mode = read or write.
        call   busy           ; wait until controller is ready
2$:      tst    flag            ; This is the main loop of this subroutine.

```

```

        bmi      200$          ; This loop is so long active as a interrupt
        beq      2$           ; occurs from the OP ended interrupt routine
                                ; or an ERROR has occurred
        cmp      count,#256.   ; Do we really received 256. words?
        beq      10$          ; Yes, branch to continue.
6$:      .PRINT  #NONR         ; give a warning message and
        call     busy          ; wait untill controller is ready.
        TST      @#DRQINT      ; *** DISABLE interrupts
        TST      @#OPEINT      ; " "
        inc      soft          ; increment soft-Error count , skip
        br       205$          ; next routine and execute a retry !
10$:     mov      #buffer,point ; reset buffer-pointer=r3
        clr      drqok         ; prepare "drqok" and count
        clr      count         ; for next operation.
        clr      retry         ; clear retry-flag
        br       300$          ; and loop.
        ;+
        ; Error/Retry handling part of this subroutine
        ; as well ERROR info handling !!!
        ;-
200$:    call     busy          ; wait untill controller is ready.
        TST      @#DRQINT      ; *** DISABLE interrupts
        TST      @#OPEINT      ; " "
        mov      @#er,erprin   ; save ERROR-PRECOMP - register
        mov      @#sta2,stcoin  ; save STA2/Command - register,
        inc      retry         ; INCREMENT retry counter
        cmp      #5,retry      ; retry-limit reached ??
        beq      210$          ; yes -- Now it is really a HARD ERROR
        .print   #newline      ; print new line and
        .print   #bad          ; a ERROR Info message
        call     errinf        ; start the ERROR-INFO subroutine
        call     reginf        ; and the Register-info routine.
205$:    Mov      #10,@#stat    ; reset the controller
        call     busy          ; and wait untill reset is done.
        br       250$
        ;+
        ; Fatal HARD - I/O Error .. give message
        ;
210$:    clr      Retry         ; reset retry-count
        inc      Hard          ; Increment Hard-Error counter
        .print   #newline
        .print   #full
        .print   #ioerr
        .print   #full
        call     errinf        ; give status about last used cylinder,
        call     reginf        ; sector, head and Error-registers
        .print   #full
250$:    TST      @#DRQINT      ; *** enable interrupts
        TST      @#OPEINT      ; " "
        clr      count         ; clear I/o tranfer counter
300$:    return
        ;
        ;
        ;+
        ; ***** INTERRUPT Service Routines *****
        ;-
        ;
        ;+
        ; OP-ended Interrupt service routine , initialized
        ; if one block of 256. words (=one sector) was
        ; completely written or read.
        ;-
AINT:    .INTEN  DEVPRI        ; switch to correct priority.
        MOV      @#STA2,end     ; get status .. tranfer completed o.k ??
        bit      #400,end       ; CHECK error bit and branch
        beq      20$            ; in case of error-free to 20$
        mov      #100000,flag    ; E R R O R:-> set flag to minus
        br       30$            ; and exit !!
20$:     inc      flag

```

```

30$:   RTS PC                      ; Return....
      ;
      ;
      ;+
      ; DRQ-Interrupt service routine to read/write one Word
      ; -
BINT:  .INTEN  DEVPRI              ; Switch to correct priority.
      mov     @#sta2,drqok         ; get controller status
      bit     #400,drqok          ; test for an error = bit *
      beq     20$                 ; not set,all O. K. =continue
      mov     #100000,flag        ; set flag for error (=minus)
      br      30$                 ; and exit
20$:   bit     #20,mode            ; Should we do a write or a read ??
      bne     25$                 ; we should do write(bit 4=set), branch to 25$
      mov     @#da,temp           ; READ: receive one of 256. words,
      mov     temp,@point         ; bring this to buffer
      br      28$                 ; and skip write part!
25$:   mov     patter,@#da        ; WRITE: move test-pattern to buffer!
28$:   inc     count              ; Increment test-counter
      inc     point              ; Uptate pointer
      inc     point              ; to buffer (+ 2 )
30$:   RTS PC                      ; Return.....
      ;
      ;
      ;+
      ; Follow routine will be executed if the user was pressing
      ; tmcice ^C^C. This routine will give a ERROR summary report
      ; and will reset the controller.
      ; -
abo:   Mov     #10,@#stat         ; reset the rd-controller
      call    busy               ; and wait untill finisch
      TST     @#DRQINT           ; *** DISABLE interrupts
      TST     @#OPEINT           ; " "
      .print  #newln             ; New line
      .print  #Hallo             ; print Info
      .print  #full
      .print  #ende
      .print  #full
      .print  #newln
      .print  #summ
      .print  #newln
      .print  #abohar            ; Print Haeder of Hard-Errors
      mov     hard,r4            ; and prepare R4 to give number
      call    ascii              ; in octal ascii
      .print  #abosof            ; Print Haeder of soft-Errors
      mov     soft,r4            ; and prepare R4 to give number
      call    ascii              ; in octal ascii
      .print  #abobad            ; Print Haeder of bad data-Errors
      mov     baddat,r4          ; and prepare R4 to give number
      call    ascii              ; in octal ascii
      .print  #newln
      .print  #full
      .exit                                ; THIS IS the only way to exit !!
      ;
      ;
      ;+
      ; ----- Local Values and ASCII messages -----
      ; -
      ;
      .EVEN
consol: .blkb  80.
EMTLST: .BLKW  6                  ; Interrupt EMT - List
timare: .blkw  2                  ; TIME-request - EMT - List
qelem:  .blkw  10                 ; Extra Q-Element
Marea:  .blkw  6                  ; .Qset EMT Argument block
SCCa:   .word  0                  ; ^C^C Status word
ticks:  .word  0,0               ; words to receive HI u. Lo Ticks
ocnum:  .blkb  10                 ; Ascii buffer to print a octal number
modify: .word  0                  ; Modifier word for random

```



```

branch: .word 0 ; flag for expanded test 1=expand
retry: .word 0 ; ERROR-Retry lag
head: .word 0 ; - Head-select-word
cylind: .word 0 ; - Cylinder select word
sector: .word 0 ; - Sector select word
buffer: .blkw 256. ; Buffer for one sector contents.
point: .word 0 ; pointer- word for I/O buffer
drqok: .word 0 ; word to test for successfull I/O
count: .word 0 ; counter to check 256. I/O word.
end: .word 0 ; TEST-word for completed sector transfer
datflg: .word 0 ; temp. flag for subroutine compar
compoi: .word 0 ; pointer for subroutine 'compar'
comcou: .word 0 ; counter for subroutine 'compar'
flag: .word 0 ; ERROr or O.K flag ( minus = ERROR )
temp: .word 0 ; temporary value only
mode: .word 0 ; word for I/O mode = READ/WRITE
patter: .word 0 ; word for test-pattern (CYL+SECT.+HEAD)
bapatt: .word 0 ; backup word for test-pattern
line: .word 0 ; character-line-count
soft: .word 0 ; SOFT-ERROR-COUNT
hard: .word 0 ; HARD-ERROR-COUNT
baddat: .word 0 ; BAD-DATA-COUNT
erprin: .word 0 ; Word after error for ERROR/PRECOMP
stcoin: .word 0 ; Word after error for STA2/COMMAND
chcou: .word 0 ; ASCII character counter
;
.even
HALLO: .ASCIZ<33>/[2J INTERRUPT-driven Q-BUS<->CTI-Bus TEST PROGRam /<12><15>
PROERR: .ASCIZ<33>/[2J P R O T E C T ERROR !!!/
ioerr: .asciz<7>/**HARD** I-O ERROR,... 5 Retries without success !! /
dot: .asciz/./<200>
crlf: .asciz<15><15><200>
wstart: .asciz<12><15>/ Writting Winchester with Test-Pattern !, please wait/
test: .asciz<12><12><15><7>/ Exerciser is now starting ..../<12><15>
ask: .asciz/ Is Winchester already be written with TESTPATTERN ? .. /<200>
simple: .asciz/ Do want a extensive write-read-verify ?.../<200>
NONR: .ASCIZ<7>/ Illegal numbers of data-transfers received !!!/
bad: .asciz/ ----- I-O E R R O R -----/
funk: .asciz/ I-O Funktion was: /<200>
re: .asciz/ R E A D /
wr: .asciz/ W R I T E /
heaerr: .asciz<12><15>/ Selected HEAD was ..... /<200>
cylerr: .asciz<12><15>/ Selected CYLINDER was ..... /<200>
secerr: .asciz<12><15>/ Selected SECTOR was ..... /<200>
ererp: .asciz<12><15>/ Contens of ERROR-PRECOMP register was.. /<200>
ersto: .asciz<12><15>/ Contens of STA2-COMMAND register was.. /<200>
dattxt: .asciz<12>/----- Follow part of Winchester contains BAD Data ! -----/
dathea: .asciz<12>/-----/<12><15>
dathe2: .asciz/ Data # good: bad:/
space: .asciz/ /<200>
newln: .ascii<12><15><200>
full: .asciz/*****
ende: .asciz/***** Program aborted through twice ^C !! *****/
Summ: .Asciz/ E R R O R summary Report of Test-Program: /
abohar: .asciz<12><15>/ Number of H A R D Errors: ..... /<200>
abosof: .asciz<12><15>/ Number of S O F T Errors: ..... /<200>
abobad: .asciz<12><15>/ number of B A D Data on RD50: ..... /<200>
;
.end start

```