RCBD UDOS

Ralf-Peter Nerlich
rio@early8bitz.de

10. Dezember 2024

Zusammenfassung

Handbuch RCBD

UDOS auf RomWBW unterstützter Hardware

Dieses Dokument wurde mit LaTeX 2_ε formatiert.

Inhaltsverzeichnis

1 Sicherheitshinweise **5**

2 Einleitung **5**

3 Begriffserklärung **6**

4 Systemvoraussetzungen **9**
4.1 8 bit Rechner . . . 9
4.2 Serielles Terminal . . . 9

5 UDOS auf RomWBW/HBIOS **10**
5.1 CPU . . . 10
5.1.1 Unterstützte CPUs . . . 10
5.1.2 UDOS auf Z180 - geht das? . . . 10
5.2 Hauptspeicher . . . 10
5.3 Stack . . . 12
5.4 Massenspeicher . . . 12
5.5 Hierarchie der RCBD Komponenten . . . 13

6 Systemstart **14**
6.1 Start RomWBW . . . 14
6.2 Laden des RCBD-ROM über die serielle Konsole . . . 15
6.3 Starten des RCBD-ROM mit dem RomWBW Loader . . . 16
6.4 Initialisierungsmeldungen des RCBD-ROM . . . 17
6.5 Arbeiten im ROM Monitor . . . 17
6.6 Start von UDOS . . . 18

7 Diskmapping **20**
7.1 Dienstprogramm 'md' . . . 20
7.1.1 Diskmapping anzeigen . . . 20
7.1.2 Diskmapping aufheben . . . 21
7.1.3 Diskmapping erstellen . . . 21
7.1.4 Fehlermeldungen von 'md' . . . 22
7.1.5 Nutzung der RAM-Disk . . . 22
7.1.6 Initiales Mapping . . . 23
7.1.7 Entnahme der Systemdiskette . . . 24

8 RCBD spezifische UDOS Kernkomponenten **26**
8.1 RCBD-ROM . . . 26
8.2 OS . . . 26
8.3 NDOS . . . 27
8.4 PATCHOS . . . 27
8.5 $PRTC . . . 27

9 Neue und modifizierte Dienstprogramme **28**
9.1 format . . . 28
9.2 status . . . 28
9.3 md . . . 29
9.4 copy.disk . . . 29
9.5 y2kdate . . . 30
9.6 set.date . . . 31
9.7 do / ndo . . . 31
9.8 sys.link / sysrd.link . . . 32
9.9 edit . . . 34
9.10 vi . . . 34
9.11 ttysize . . . 35
9.12 xm . . . 36
9.12.1 XMODEM senden . . . 36
9.12.2 XMODEM empfangen (receive) . . . 36

A Einstellungen Terminalprogramm (TeraTerm) **38**

B Speicherbelegung **39**

C Datenträgergrößen **41**

D Systemstart auf Z80-SBC **42**

E Systemkonfiguration SC126 Z180-SBC **44**

F Systemkonfiguration SC720 Z80-SBC **45**

G System- und Fehlermeldungen **47**
G.1 Lade- und Initialisierungsmeldungen . . . 47
G.2 UDOS Startmeldungen . . . 48
G.3 Meldungen von PATCHOS . . . 49
G.4 Fehlermeldungen von 'md' . . . 49
G.5 Sonstige Meldungen . . . 50

H Kommandos des RCBD-ROM-Monitors **51**

I Aufruf des Treibers $PRTC **52**

J RomWBW Datenträger vorbereiten **53**

K RCBD auf Datenträger kopieren **54**

1 Sicherheitshinweise

Beachten Sie beim Umgang mit Halbleitern und elektronischen Baugruppen die allgemeinen Richtlinien zum Umgang mit elektrostatisch empfindlichen Bauelementen (ESD sensitive Devices).

RC2014, RCBus und ähnliche Systeme, die als Zielplattform für diese Software dienen, haben keine Hotplug Fähigkeiten. Der Wechsel von Baugruppen oder Datenträgern muss daher unbedingt im spannungslosen Zustand erfolgen.

2 Einleitung

RCBD ist eine Implementierung des 8bit Betriebssystems *UDOS* 3.1 des EAW Computers P8000 auf moderne Z80 und Z180 basierte modulare oder Single Board Computer. Die Implementierung des Betriebssystems setzt auf der *HBIOS* Schnittstelle von *RomWBW*[1] auf, das für viele Retro-Computing Systeme verfügbar ist.

Der Name *RCBD* leitet sich von **RCB**us-**D**USSEL ab. DUSSEL war der Name meines Selbstbau Computers aus den 80er Jahren[2], der bzgl. *UDOS* ebenfalls annäherend P8000 kompatibel war.

Prinzipiell sollte die *RCBD* Implementierung auf jeder Z80 oder Z180 basierenden Hardware lauffähig sein, die von *RomWBW* unterstützt wird und die unter Punkt 4.1 gelisteten Minmalvoraussetzungen erfüllt.

Um *RCBD* auf einen *RomWBW* Rechner zu bringen, sind einige Grundkenntnisse im Umgang mit der *RomWBW* Architektur notwendig. Das betrifft z. B. das Generieren einer kundenspezifischen *RomWBW* ROM Version, *RomWBW*-Firmware flashen und updaten, Präparierung von Datenträgern zur Nutzung unter *RomWBW*. Daher sei hier auf die ausführliche Dokumentation zu *RomWBW* verwiesen, die im Git Repository[1] des Projekts als PDF zu finden ist.

- *RomWBW* System Guide
- *RomWBW* User Guide

[1] https://github.com/wwarthen/RomWBW/
[2] https://rio.early8bitz.de/__fps2/fps2-computer.htm

3 Begriffserklärung

`RomWBW`

Die *RomWBW* Software[1] bietet eine vollständige Implementierung von CP/M (und ähnlichen) Betriebssystemen und Standalone Anwendungen für moderne Z80/Z180/Z280 Retro-Computing Hardwaresysteme.

`HBIOS`

RomWBW isoliert konzeptgemäß alle hardwarespezifischen Funktionen im ROM-Chip selbst. Das ROM bietet eine Hardwareabstraktionsschicht, das *HBIOS*, so dass alle Betriebssysteme und Standalone Anwendungen auf einem Diskdevice auf jedem *RomWBW* basierten System laufen.

`HBIOS-Proxy (HBX)`

Während des *HBIOS* selbst versteckt in einer separaten Speicherbank ausgeführt wird, bildet der permanent verfügbare *HBIOS*-Proxy (kurz HBX) die Aufrufschnittstelle für alle *HBIOS*-Funktionen.

`RomWBW-Monitor`

Standalone Programm, das direkt vom *RomWBW*-Bootprompt gestartet werden kann. Es ermöglicht das Laden von Programmen (Intel-HEX oder XMODEM) in den Speicher von *RomWBW* und das Manipulieren des Speicherinhaltes.

`RomWBW-Loader`

RomWBW ermöglicht das Starten von binär vorliegenden Programmimages direkt aus einem Slice eines unterstützten Datenträgers. Dazu muss der Binärcode des zu startenden Programmes zusammen mit einer Datenstruktur (Prefix), die die Ladeinformationen enthält, im betreffenden Slice abgelegt werden.

`Slice`

Zentrales Element zur Verwaltung *RomWBW* unterstützter Datenträger in kleineren Einheiten zur Präsentation als Laufwerk an die laufenden Betriebssysteme oder Standalone Programme. In Punkt 5.4 weiter erläutert.

`RIO`

RIO ist ein diskettenorientiertes Betriebssystem, das Zilog Mitte der 70er Jahre zusammen mit seinen Mikrorechner-Entwicklungssystemen (ZDS, MCZ, PDS) auf den Markt brachte. Der Fokus dieser Systeme lag in der Bereitstellung von Hardware- und Software-Entwicklungswerkzeugen für die eigenen Prozessorfamilien Z80, Z8 und Z8000. Für den Einsatz als Personalcomputer im Büroumfeld waren diese Geräte mangels officespezifischer Anwendungsprogramme weniger geeignet.

`UDOS`

UDOS ist die Portierung von *RIO* auf in der damaligen DDR hergestellte Computerhardware, die mit der Z80 kompatiblen Prozessorfamilie U880 ausgestattet war[3]. Die ersten *UDOS*-Versionen arbeiteten mit dem übernommenen *RIO* Dateisystemtreiber `$ZDOS`, der ein propritäres Sektorformat zur Verlinkung von physischen Sektoren zu logischen Dateien verwendete. Spätere *UDOS*-Versionen benutzten ein standardisiertes Sektorformat sowie Zeigerblöcke, um die zu einer Datei gehörenden physischen Sektoren zu verwalten. Zu diesen Versionen gehört *UDOS* 5 auf dem A5120 oder PC1715W und *UDOS* 3.1 auf dem EAW P8000.

`RCBD`

RCBD ist ein Hobbyprojekt, dass es zum Ziel hat *UDOS* auf heute im Retro-Computer Umfeld verwendete Hardware zu portieren. Ausgangspunkt ist dabei die *UDOS* 3.1 Implementierung des EAW P8000.

`RCBD-ROM`

RCBD-ROM ist die residente Komponente des *RCBD*. Sie entspricht funktionell dem 3 KiB ROM der Zilog Systeme bzw. den 3 KiB residenter Software, die bei den DDR-*UDOS*-Implementierungen von speziell reservierten Sektoren der Systemdiskette oder einem versteckten ROM in den unteren Teil des Hauptspeichers geladen wurde.

RCBD-ROM enthält die residententen Treiber `$FLOPPY` und `$PCON`.

`RCBD-UDOS`

RCBD-UDOS ist die ladbare Komponente des *RCBD*. Sie entspricht im wesentlichen dem vom P8000 bekannten *UDOS* 3.1. Einige Systemkomponenten und Dienstprogramme mussten für die Portierung modifiziert oder neu geschrieben werden. Diese Komponenten werden in den Kapiteln 8 und 9 beschrieben.

Da es aus Sicht der Betriebssystemphilosophie keine gravierenden Unterschiede zwischen *RIO*, *UDOS* und *RCBD-UDOS* gibt, soll im weiteren nur noch der Begriff *UDOS* für das ladbare bzw. geladene Betriebssystem mit seinen Dienst- und Anwendungsprogrammen benutzt werden.

`$FLOPPY`

Low Level (Sektor Level) Treiber zur physischen Bedienung der Diskettenlaufwerke. Im *RCBD* übernimmt `$FLOPPY` auch das Bedienen der RAM-Disk. Im *RCBD* spricht `$FLOPPY` die Hardware der Massenspeicher

[3] https://www.robotrontechnik.de/html/software/udos.htm

nicht direkt an, sondern wandelt die Requests in entsprechende *HBIOS*-Aufrufe zum Zugriff auf die tatsächlichen Datenträger um. `$FLOPPY` ist Bestandteil des *RCBD-ROM* und somit unmittelbar nach dem Starten des *RCBD* aktiv.

`$NDOS`

Treiber zur logischen Verwaltung des *UDOS* Dateisystems. `$NDOS` ist hardwareneutral, es benutzt Aufrufe an `$FLOPPY` zum Zugriff auf die einzelnen Sektoren der Disketten. `$NDOS` ist eine separate Datei (**'NDOS'**) die während des Bootvorgangs von *UDOS* in den Speicher geladen wird.

`$PCON`

Treiber zur physischen Bedienung der seriellen Konsolenschnittstelle. Im *RCBD* spricht `$PCON` die Hardware der seriellen Schnittstelle nicht direkt an, sondern wandelt die Requests in entsprechende *HBIOS*-Aufrufe zum Zugriff auf die tasächlichen seriellen Bausteine um. `$PCON` ist Bestandteil des *RCBD-ROM* und somit unmittelbar nach dem Starten des *RCBD* aktiv.

`$CON`

Erweiterter Treiber für die serielle Konsolenschnittstelle. `$CON` erweitert den Funktionsumfang des `$PCON` Treibers. `$CON` ist hardwareneutral, es benutzt Routinen aus `$PCON` beim Zugriff auf die serielle Hardware. `$CON` ist integrierter Bestandteil der Betriebssystemexekutive (Datei **'OS'**) und somit erst nach Laden von *UDOS* aktiv.

`$PRTC`

Treiber zum Zugriff auf die Datums- und Zeitinformationen der Echtzeituhr. Im *RCBD* spricht `$PRTC` die Hardware der Echtzeituhr nicht direkt an, sondern wandelt die Requests in entsprechende *HBIOS*-Aufrufe zum Zugriff auf die tatsächliche RTC-Hardware um. `$PRTC` ist Bestandteil des *RCBD-ROM* und somit unmittelbar nach dem Starten des *RCBD* aktiv.

4 Systemvoraussetzungen

4.1 8 bit Rechner

Mindestens notwendig sind:

- Z80 CPU oder Z180 CPU im Interruptmode 2
- *RomWBW* unterstützer Paged ROM
- *RomWBW* unterstützer Paged RAM; Empfohlen 512 KiB Paged RAM (16 32 KiB Seiten) für Nutzung einer RAM-Disk (Standard RAM Bank Layout). 128 KiB Paged RAM (4 32 KiB Seiten) möglich, dann keine RAM-Disk (Tiny RAM Bank Layout).
- *RomWBW* unterstützter Memory Manager (MMU)
- Mind. ein serieller Kanal (SIO, ACIA, UART, interner ASCI bei Z180) für Systemkonsole
- Mind. ein interruptfähiger Timer (CTC bei Z80, interner Timer bei Z180)
- Mind. ein Massenspeicher, der von *RomWBW* als Harddisk reportet wird (CF-Karte, SD-Karte, pATA (Parallelport IDE), USB-Sticks)
- *RomWBW*-Firmware Version 3.4.0 oder höher

Weiterhin kann *RCBD* bisher folgende optionale Hardware nutzen:

- Echtzeituhr (RTC)

Noch nicht implementiert ist die Einbindung folgender Hardware in *RCBD*

- Zweiter serieller Kanal zum Anschluss eines Druckers

Alle weitere Hardware, die von *RomWBW* unterstützt wird, ist unter *RCBD* nicht zugänglich.

4.2 Serielles Terminal

UDOS und so auch *RCBD* arbeitet konsoleorientiert. Zur Bedienung ist ein serielles Terminal notwendig, das bzgl. Baudrate auf die genutzte Schnittstelle einstellbar sein muss. Das kann ein Hardware Terminal sein, oder ein externer Computer mit Terminalemulationsprogramm. Die Einstellungen für die Übertragungsparameter stehen im Anlage A.

In *RCBD-UDOS* existiert eine Implementierung des XMODEM Programms über die serielle Konsolenschnittstelle, um Dateien zwischen dem *RCBD*-System und einem externen Computer zu übertragen. Zur Nutzung muss das verwendete Terminalemulationsprogramm das XMODEM Protokoll implementiert haben, was z. B. bei TeraTerm der Fall ist.

5 UDOS auf RomWBW/HBIOS

5.1 CPU

5.1.1 Unterstützte CPUs

Obwohl *RomWBW* neben den Z80 und den Z180 basierten Rechnern auch einige Systeme mit eZ80 und Z280 unterstützt, beschränkt sich die *RCBD* Implementierung auf Rechner mit den beiden CPUs Z80 und Z180.

Die Technologie (NMOS/CMOS) und die Taktfrequenzen der eingesetzten CPUs spielen für *RCBD* nur eine untergeordnete Rolle. Generell ist natürlich schneller und stromärmer immer die bessere Wahl.

5.1.2 UDOS auf Z180 - geht das?

Ja. Die Z180 CPU ist weitestgehend befehlskompatibel zur Z80 CPU. Die zusätzlichen Befehle der Z180 CPU spielen nur bei der Systeminitialisierung und einigen hardwarenahen Passagen eine Rolle, was vollständig durch das *RCBD-ROM* und das *HBIOS* verdeckt ist. Das *UDOS*-Betriebssystem und die Anwendungs- und Dienstprogramme benötigen diese Befehle nicht. Somit ist die *RCBD*-Systemdiskette zwischen Z80 und Z180 basierten Rechnern uneingeschränkt austauschbar.

Wie sieht es mit den undokumentierten Befehlen der Z80 CPU aus? Diese sind nicht in der Z180 CPU implementiert. Alle nicht dokumentierten Opcodes führen beim Z180 zu einer Ausnahmebehandlung (Trap), die im *RCBD-ROM* abgefangen und gemeldet wird. Zilog hat in seinem *RIO*-Betriebssystem nie die undokumentierten Befehle verwendet, und auch bei der Portierung des P8000 *UDOS* zum *RCBD* ist mein Z180 System noch nie in einen Trap gelaufen.

Durch *UDOS*-Anwender selbst geschriebene Software könnte natürlich undokumentierte Z80 Befehle enthalten. Aber *UDOS* selber mit seinen eigenen Programmen scheint diesbezüglich 'sauber' zu sein.

5.2 Hauptspeicher

UDOS kennt nur 64 KiB linearen Hauptspeicher. In der *RomWBW* Umgebung wird dies durch die User-Bank in den unteren 32 KiB des Hauptspeichers (`0x0000..0x7FFF`) bereitgestellt. Die oberen 32 KiB (`0x8000..0xFFFF`) werden durch die Common-Bank gebildet. Diese Speicherpräsentation darf sich während der Laufzeit von *UDOS* Code nicht ändern.

Während die Common-Bank permanent im oberen Adressbereich eingeblendet ist, kann die User-Bank temporär durch eine andere Bank ersetzt werden (Bank Switching). Das passiert z. B. immer, wenn die Anwendungssoft-

ware (in unserem Fall ein für *RCBD* angepasster *UDOS*-Treiber) mit der Hardware kommunizieren will. Dann wird über einen Aufruf einer *HBIOS*-Funktion der entsprechende *RomWBW*-Treiber benutzt. Diese Treiber liegen in der *HBIOS*-Bank, die zu diesem Zweck in die unteren 32 KiB eingeblendet wird. D. h. der untere Teil des *UDOS*-Speichers ist während dessen weg.

Damit der Aufruf und vor allem die Rückkehr in das aufrufende *UDOS*-Programm funktioniert, existiert eine Zwischenstation in der permanent verfügbaren Common-Bank, der sogenannte *HBIOS*-Proxy (HBX). Dieser liegt am oberen Ende des 64 KiB Adressraumes und belegt 512 B. Bei einem Aufruf einer *HBIOS*-Funktion passiert etwa folgendes:

- Das aufrufende Programm (egal wo es im Hauptspeicher liegt) ruft eine pasende *HBIOS*-Funktion mit den notwendigen Parametern über den HBIOS-Proxy auf.
- Der HBX schaltet die unteren 32 KiB des Hauptspeichers von der User-Bank auf die *HBIOS*-Bank um.
- Der HBX ruft die dem Funktionsaufruf entsprechende Treiberroutine in der *HBIOS*-Bank auf.
- Am Ende führt der *HBIOS*-Treiber ein Return zur rufenden Instanz im HBX durch.
- Der HBX schaltet die unteren 32 KiB des Hauptspeichers zurück von der *HBIOS*-Bank auf die User-Bank, damit sind die 64 KiB von *UDOS* wieder komplett.
- Der HBX führt ein Return zum rufenden Programm durch, welches sich jetzt wieder an der ursprünglichsen Stelle im Hauptspeicher befindet.

Da *UDOS*-Programme die Hardware nicht direkt ansprechen, sondern über entsprechende *UDOS*-Treiber, ist das Verhalten für das *UDOS*-Betriebssystem (OS) und reine *UDOS*-Dienstprogramme transparent. Im wesentlichen betrifft dies die beiden Treiber `$FLOPPY` für physische Diskettenzugriffe und `$PCON` für die Ein- und Ausgabe über die serielle Konsolenschnittstelle. Beide Treiber sind im residenten Teil von *UDOS* dem *RCBD-ROM* enthalten und verdecken die *RomWBW* Anpassung vor dem eigentlichen *UDOS*-Betriebssystem und seinen Dienst- und Anwendungsprogrammen.

Einen Überblick über die Speicheraufteilung im *RCBD* gibt Anlage B.

5.3 Stack

Die Konsequenz aus dem Paging der unteren 32 KiB zwischen *UDOS*-Speicher und *HBIOS* ist, dass der Stackpointer der CPU stets auf einen permanent verfügbaren Speicher, also auf eine Adresse in den oberen 32 KiB (Common-Bank ab `0x8000`) zeigen muss. Für *RCBD-ROM*, **OS** und **NDOS** ist das sicher gestellt. Ladbare Dienst- und und Anwendungsprogramme, die sich gemäß *UDOS*-Konzept den Stack vom **OS** dynamisch zuteilen lassen, sind unverändert lauffähig, da die Zuteilungsalgorithmen im **OS** (Punkt 8.2) angepasst wurden. Ladbare Programme, die einen hart kodierten Stack in den unteren 32 KiB verwenden, müssen manuell gepatched werden. Der Stack dieser Programme muss in den extra dafür reservierten 256 B Bereich `0xFB00..0xFBFF` verlegt werden (siehe Anlage B).

5.4 Massenspeicher

UDOS arbeitet ausschließlich mit Disketten und kann 8 Diskettenlaufwerke (Laufwerke 0...7) addressieren. Der als Vorbild dienende P8000 hat einen 8272-basierten Diskettenkontroller, welcher nur vier Laufwerke (0...3) bedienen kann. Im *RCBD* wurde der `$FLOPPY` Treiber um die Möglichkeit erweitert, über die Laufwerksnummern 4...7 RAM-Disks anzusprechen, wobei in der mir bekannten RomWBW kompatiblen Hardware maximal eine RAM-Disk existieren kann. Laufwerk 4 ist im *RCBD* eine RAM-Disk, sofern die Hardware mit einem 512 KiB RAM-Baustein ausgestattet ist (bei kleinerem RAM sind nicht genug Bänke für eine RAM-Disk verfügbar).

RomWBW organisiert den Massenspeicher in sogenannten Slices zu je 8 MiB. Das hat seinen Ursprung in den in *RomWBW* serienmäßig mitgelieferten CP/M Betriebssystemen, die bis zu 8 MiB pro Laufwerk nutzen können. Auf diese Organisation wurde bei *RCBD* aufgebaut, obwohl das Image einer *UDOS*-Diskette nur 640 KiB groß ist. Bei einer Maxmimalanzahl von 256 Slices bei einem Datenträger >= 2 GiB ist Platz für wahrscheinlich alle noch existierenden *UDOS*-Disketten. Der Platzverschleiß wurde zu Gunsten einer leichteren Umrechnung einer *UDOS*-Diskadresse (Track/Sektor) in die LBA-Adresse eines bestimmten Slices auf dem physischen Speicher in Kauf genommen. Außerdem ermöglicht erst die simple Zuordnung 'Ein Slice = Eine *UDOS*-Diskette' ein vernünftiges Sichern einzelner Diskettenimages auf einen externen Rechner oder das Einspielen eines weiteren Diskettenimages in einen freien Slice einer als Datenträger dienenden CF- oder SD-Karte.

RCBD akzeptiert nur Datenträger die

- von RomWBW als Harddisk reportet werden (CF, SD, pATA, USB) und im 'Modern Layout' (hd1k) vorbereitet sind
- von RomWBW als RAM-Disk mit mindestens 256 KiB reportet werden

- und in beiden Fällen als LBA adressierbar reportet werden

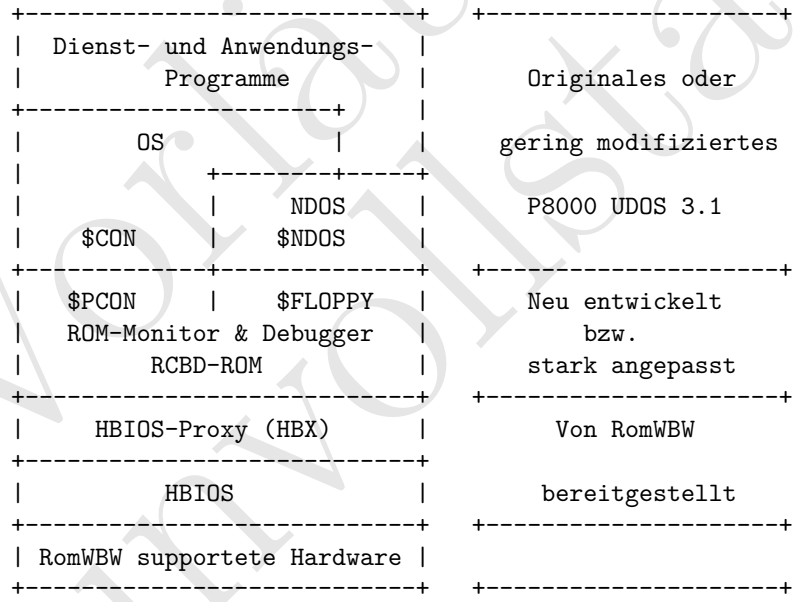
Native Floppy Disks und Harddisks, die im älteren 'Classic Layout' (hd512) präpariert sind, werden nicht unterstützt.

Grundlagen zu Datenträgern findet man im Kapitel 'Disk Layout' im '*RomWBW* System Guide' sowie für die *RomWBW* konforme Vorbereitung neuer Datenträger im Kapitel 'Disk Preparation' im '*RomWBW* User Guide'. Anlage C listet typische Datenträgergrößen auf. Die Vorbereitung eines Beispieldatenträgers ist in diesem Handbuch in Anlage J beschrieben.

Als Diskettenformat wird im *RCBD* ausschließlich der beim P8000 als Typ 5 bezeichnete Diskettentyp (80 Spuren, 2 Köpfe (Double Side), 16 Sektoren pro Kopf und Seite, 256 B pro Sektor) unterstützt. Das entspricht dem in der DDR als K5601 bzw. MFS 1.6 bekannten Laufwerk. Die Kapazität ist 640 KiB (2560 Sektoren @ 256 B) pro Diskette.

5.5 Hierarchie der RCBD Komponenten

Die einzelnen Ebenen in der Struktur von *RCBD* sind in der folgenden Grafik dargestellt.



6 Systemstart

6.1 Start RomWBW

RomWBW startet nach Power On oder Reset (Taste) und führt eine Hardwareerkennung und -initialisierung durch. Die konfigurierte u/o gefundene Konfiguration wird detailiert ausgegeben. Ebenso werden alle erkannten Massenspeicher aufgelistet.

```
------------------------------------------------------------------------
RomWBW HBIOS v3.5.0-dev.91, 2024-10-13

Small Computer SC126 [SCZ180_sc126_std] Z8S180-N @ 18.432MHz IO=0xC0
0 MEM W/S, 2 I/O W/S, INT MODE 2, Z180 MMU
512KB ROM, 512KB RAM, HEAP=0x1F82
ROM VERIFY: 00 00 00 00 PASS

LCD: IO=0xAA NOT PRESENT
AY: MODE=RCZ180 IO=0x68 NOT PRESENT
ASCI0: IO=0xC0 ASCI W/BRG MODE=115200,8,N,1
ASCI1: IO=0xC1 ASCI W/BRG MODE=115200,8,N,1
DSRTC: MODE=STD IO=0x0C Mon 2024-11-11 16:50:11 CHARGE=OFF
MD: UNITS=2 ROMDISK=384KB RAMDISK=256KB
FD: MODE=RCWDC IO=0x50 NOT PRESENT
IDE: IO=0x10 MODE=RC
IDE0: ATA 8-BIT LBA BLOCKS=0x0003D400 SIZE=122MB
IDE1: ATA 8-BIT LBA BLOCKS=0x0007A2B0 SIZE=244MB
PPIDE: IO=0x20 PPI NOT PRESENT
SD: MODE=SC OPR=0x0C CNTR=0xCA TRDR=0xCB DEVICES=1
SD0: SDHC NAME=SU04G BLOCKS=0x00762C00 SIZE=3781MB
CH0: IO=0x3E NOT PRESENT
CH1: IO=0x3C NOT PRESENT
FP: IO=0x00 NOT PRESENT

Unit       Device     Type              Capacity/Mode
---------- ---------- ----------------  --------------------
Char 0     ASCI0:     RS-232            115200,8,N,1
Char 1     ASCI1:     RS-232            115200,8,N,1
Disk 0     MD0:       RAM Disk          256KB,LBA
Disk 1     MD1:       ROM Disk          384KB,LBA
Disk 2     IDE0:      Hard Disk         122MB,LBA
Disk 3     IDE1:      CompactFlash      244MB,LBA
Disk 4     SD0:       SD Card           3781MB,LBA

Small Computer SC126 [SCZ180_sc126_std] Boot Loader

Boot [H=Help]:
------------------------------------------------------------------------
```

Hier wurde *RomWBW* auf einem SC126[4] Z180 Single Board Computer gestartet. Es finden sich zwei serielle Schnitstellen (ASCI0 und ASCI1) sowie fünf Disk Devices. Neben der RAM-Disk (MD0) und der ROM-Disk (MD1) in den Speicherchips des SBC steckt ein Compact Flash Modul SC729[5] als Erweiterungskarte auf dem SBC, bestückt mit einer 128 MB CF-Karte (IDE0) und einer 256 MB CF-Karte (IDE1). Außerdem steckt noch einen SD-Kartenadapter, der mit einer 4 GB SD-Karte bestückt ist (SD0).

Die Konfiguration des *RomWBW* ROMs steht in Anlage E.

6.2 Laden des RCBD-ROM über die serielle Konsole

Am Bootprompt von *RomWBW* kann mit dem Kommando `M[onitor]` (Groß- und Kleinschreibung ist nicht relevant) der *RomWBW*-Monitor aufgerufen werden. Er meldet sich mit der aktuellen User-Bank Nummer als Prompt, hier die Bank '8E'.

```
------------------------------------------------------------------
Boot [H=Help]: m

Loading Monitor...

Monitor Ready (? for Help)
8E>
------------------------------------------------------------------
```

Der *RomWBW* Monitor besitzt das Kommando `L[oad]` zum Laden von Intel-HEX Dateien in den RAM. Die zu ladende Intel-HEX-Datei muss dabei im Terminalprogramm ausgewählt werden. Dazu muss die Konsole über ein Terminalemulationsprogramm benutzt werden, welches eine Funktion zum Senden von Dateien ohne bestimmtes Protokoll besitzt. Bei TeraTerm geschieht das über 'Datei'->'Datei senden'->'Browsen zur Datei'->'Öffnen'->'Ok'. Das *RCBD-ROM* liegt als Intel-HEX Datei vor.

```
------------------------------------------------------------------
8E>L

Loaded
8E>
------------------------------------------------------------------
```

Alternativ kann das Monitorkommando `T[ransfer]` benutzt werden, wenn das zu ladende Programm in Binärform vorliegt. Hierbei ist zusätzlich die Angabe der Ladeadresse notwendig. Die zu ladende Binärdatei muss dabei im Terminalprogramm ausgewählt werden. Dazu muss die Konsole über ein Terminalemulationsprogramm benutzt werden, welches eine Funktion zum Übertragen von Dateien im XMODEM Protokoll besitzt. Bei TeraTerm geschieht das über 'Datei'->'Transfer'->'XMODEM'->'Senden'->'Browsen zur Datei'->'Öffnen'->'Ok'. Das *RCBD-ROM* liegt auch als Binärdatei vor.

[4] https://smallcomputercentral.com/sc126-z180-motherboard-rc2014/
[5] https://smallcomputercentral.com/sc729-rcbus-compact-flash-module/

```
------------------------------------------------------------------
8E>T 0

Monitor Ready

Loaded
8E>
------------------------------------------------------------------
```

Nachdem *RCBD-ROM* fehlerfrei in den RAM des *RomWBW*-Systems geladen wurde, wird es mit `R[un]` an der Adresse 0 gestartet. *RCBD-ROM* führt interne Initialisierungen seiner Datenstrukturen durch (wie jedes *UDOS* System). Außerdem werden die zur Verfügung stehenden Massenspeicher ermittelt und angezeigt. Falls die Hardware mit einer von *RomWBW* unterstützten Echtzeituhr (RTC) ausgestattet ist, wird das *UDOS*-Systemdatum (DATE) mit dem ausgelesenem Datum voreingestellt. Ist keine RTC vorhanden wird DATE mit dem Build-Datum der *RCBD-ROM*-Datei vorbelegt.

```
------------------------------------------------------------------
8E>R 0
Device 0 - RAM Disk
Device 2 - 0f Slice(s)
Device 3 - 1e Slice(s)
Device 4 - 40 Slice(s)
Date set from RTC
UDOS for Z180 RomWBW HBIOS 241016
>
------------------------------------------------------------------
```

Final meldet sich der *RCBD-ROM*-Monitor mit seinem Prompt `'>'`. Das Größerzeichen `'>'` ist der Prompt des *RCBD-ROM*-Monitors/Debuggers.

6.3 Starten des RCBD-ROM mit dem RomWBW Loader

Die elegantere Methode, *RCBD-ROM* zu laden und zu starten besteht darin, die *RCBD-ROM*-Datei in binärer Form zusammen mit einer *RomWBW* spezifischen Ladestruktur (*RomWBW*-Loader) als bootfähiges Betriebssystem in einem Slice eines Diskdevices abzulegen (*RCBD-ROM*-Loader). Dann kann am Bootprompt von *RomWBW* das *RCBD-ROM* direkt unter Angabe von Device- und Slicenummer geladen und gestartet werden.

```
------------------------------------------------------------------
Boot [H=Help]: 2.10

Booting Disk Unit 2, Slice 10, Sector 0x00028800...

Volume "RCBD Z180 ROM" [0x0000-0x1FFF, entry @ 0x0000]...
Device 0 - RAM Disk
Device 2 - 0f Slice(s)
Device 3 - 1e Slice(s)
```

```
Device 4 - 40 Slice(s)
Date set from RTC
UDOS for Z180 RomWBW HBIOS 241104
>
-------------------------------------------------------------------
```

Hier wird das *RCBD-ROM* von Device 2 (IDE0) Slice Nummer 10 gestartet.

6.4 Initialisierungsmeldungen des RCBD-ROM

RCBD analysiert als erstes die *RomWBW*-Umgebung und gibt Fehlermeldungen über Zustände aus, die den Start von *RCBD* verhindern. Das kann eine unsupportete Hardware-Plattform sein, eine falsche *RomWBW*-Version oder ein nicht geeignetes Bank-Layout.

Bei erfolgreicher Analyse werden nach der Initialisierung und vor der ersten Ausgabe des *RCBD-ROM*-Prompts Meldungen über die ermittelten für *UDOS* geeigneten Massenspeichergeräte angezeigt. Dabei werden die Devicenummern (die Nummern korrepondieren mit der Diskdevice Nummerierung beim Starten von *RomWBW*, siehe Punkt 6.1) und die Eigenschaften des Devices angezeigt.

In den Startmeldungen aus Punkt 6.2 bzw. 6.3 finden wir als Device 0 die RAM-Disk (unter RomWBW Disk 0) und die Devices 2 (IDE0=CF-Karte), 3 (IDE1=CF-Karte) und 4 (SD0=SD-Karte). Die Angabe der Sliceanzahl (im *RCBD-ROM*-Monitor sind alle Zahlenangaben hexadezimal) gibt Auskunft, wieviel *UDOS* Floppy-Disk-Images auf dem Datenträger untergebracht sein können (hier also 15, 30 bzw. 64 Slices in dezimaler Interpretation). Dabei handelt es sich um die maximale Anzahl von Slices, die auf der *RomWBW* Partition des Datenträgers Platz finden. Eine Nutzung von Slices für andere von *RomWBW* bereitgestellte Betriebssysteme (CP/M, BASIC) wird nicht automatisch erkannt und muss durch den Anwender selbst überwacht werden.

In Abhängigkeit von der Hardwareausstattung mit einer unterstützten Echtzeituhr (RTC) wird noch eine Meldung ausgegeben ob das *UDOS*-Systemdatum von der RTC gesetzt wurde oder ob keine RTC im System vorgefunden wurde.

In Anlage D sind die Startmeldungen für eine Z80-basierte *RCBD*-Implementierung abgebildet.

6.5 Arbeiten im ROM Monitor

An dieser Stelle kann im *RCBD-ROM*-Monitor/Debugger gearbeitet werden. Die Debugger Komandos entsprechen denen der *RIO*- bzw *UDOS*-Systeme, wurden aber im Laufe der Zeit 'perfektioniert'. Die Übersicht der implementierten Debugger Kommandos ist in Anlage H zu finden.

```
------------------------------------------------------------------
>d 0 40     [Zeilen eingekürzt]
0000 f3 c3 02 07 .... 64 c3 8e 0f 21 04 0d 18 03  >....RcBd...!....<
0010 c3 91 0f 22 bd 0f 18 03 .... fb 3e 3e 18 03  >..."........>>..<
0020 c3 97 0f cd 8b 04 .... 9a 0f 21 c9 00 18 03  >..........!....<
0030 c3 9d .... af 18 03 c3 a5 06 21 76 00 01 0e  >..........!v...<
>r
sz.h.pnc a  b  c  d  e  h  l  i  ixiy pcsp  mem
00000000 00 00 00 00 00 00 00 ff 0000 0000 f3 c3 0b 00
00000000 00 00 00 00 00 00 00 01 0000 fcd0 ff ff ff ff
>
------------------------------------------------------------------
```

6.6 Start von UDOS

Das eigentliche *UDOS*-Betriebssystem wird mit dem Kommando `os` gestartet. Ein zweistufiger Bootloader sucht die Dateien **'OS'** und **'NDOS'** auf der Diskette im Laufwerk 0, lädt sie an die vorgesehenen Adressen im Hauptspeicher und startet das Betriebssystem an.

Aus einer Datei **'os.init'** liest das **OS** weitere *UDOS*-Dienstprogramme und führt sie bei der Initialisierung aus. Dadurch können zusätzliche Systemeinstellungen getroffen oder einfach nur informative Meldungen ausgegeben werden. Nach erfolgreicher Initialisierung meldet sich *UDOS* mit der Eingabeaufforderung `rcbd:`.

```
------------------------------------------------------------------
>os
rcbdboot: Ok
osload:   Ok

    ************************
         RCBD OS 2.12
      UDOS auf RomWBW/HBIOS
    ************************

OS successful patched with HighStack Patch for RCBD
Dienstag, November 12, 2024

Drive 0: 5 1/4"  DD, DS, 80 tracks, 32 sectors per 256 byte (5)
Drive 1: 5 1/4"  DD, DS, 80 tracks, 32 sectors per 256 byte (5)
Drive 2: 5 1/4"  DD, DS, 80 tracks, 32 sectors per 256 byte (5)
Drive 3: 5 1/4"  DD, DS, 80 tracks, 32 sectors per 256 byte (5)

TTY/Screen size 137x34

| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
| 2.12 |      |      |      |      |

Drive 0   RCBD-Sys CF2.12 241108
```

```
Floppy Disk - Type 5 - 80 tracks  32 sectors
1858 sectors used
 702 sectors avialable
RIO REL 2.12
rcbd:
--------------------------------------------------------------------
```

7 Diskmapping

Das Diskmapping ist ein zentraler Bestandteil des *RCBD*. Ein *UDOS*-Rechner besitzt zwischen ein und vier Diskettenlaufwerke. Der Anwender kann in diese Laufwerke Disketten einlegen und wieder entnehmen. Außerdem können die Disketten beim Einlegen einen Schreibschutz besitzen oder beschreibbar sein.

Im *RCBD* sind die Disketten Images, die in Slices eines oder mehrerer von *RomWBW* unterstützten Speichergeräten (Devices) abgelegt sind. Das Pedant zum Einlegen einer Diskette ist im *RCBD* das Zuordnen einer Device:Slice Kombination, die ein gültiges *UDOS*-Diskettenimage enthält, zu einer *UDOS*-Laufwerksnummer. Dieser Vorgang wird als 'Diskmapping' bezeichnet.

Natürlich kann auch ein unbenutzter Slice einem *UDOS*-Laufwerk zugeordnet und dann unter *UDOS* als neue Diskette formatiert werden.

Der entgegengesetzte Vorgang, das Aufheben einer Zuordnung zwischen Device:Slice und *UDOS*-Laufwerk entspricht dann dem Entnehmen einer Diskette aus einem physischen Laufwerk.

7.1 Dienstprogramm 'md'

Für diesen Vorgang wurde das Dienstprogramm **'md'**, welches auf der *RCBD*-Systemdiskette zu finden ist, geschrieben. **'md'** ist für das Einlegen, Entnehmen und Steuern des Schreibschutzes von Disketten zuständig. Auch das Einbinden bzw. Abhängen der RAM-Disk wird von **'md'** übernommen.

7.1.1 Diskmapping anzeigen

'md' ohne Argumente zeigt das aktuelle Diskmapping an.

```
-------------------------------------------------------------------
rcbd:md
| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
| 2.12 | 4:14 |      |      | 0:00 |
-------------------------------------------------------------------
```

Die Kopfzeile ist selbsterklärend, vier Diskettenlaufwerke und ein RAM-Disk-Laufwerk. Ein leeres Feld unter dem Laufwerksbezeichner kennzeichnet ein leeres, ungemapptes Laufwerk. Einen gezielten Zugriff darauf würde *UDOS* mit einem 'Drive not ready' Fehler (Error C2) beantworten. Gemappte Laufwerke werden durch `device.slice` oder `device:slice` angezeigt.

Das Trennzeichen zwischen `device` und `slice`, Punkt oder Doppelpunkt, zeigt einen vorhandenen Schreibschutz an. Als Eselsbrücke kann man sich merken: 'Ein Punkt: Lesen, Zwei Punkte: Lesen und Schreiben'. In Laufwerk 0 steckt also die schreibgeschützte Diskette aus Slice 12 von Device 2.

In Laufwerk 1 liegt eine beschreibbare Diskette, deren Image in Slice 14 auf Device 4 gespeichert ist. Das RAM-Disk-Device von *RomWBW* (Device 0) ist beschreibbar im Laufwerk 4 gemappt. Bei der RAM-Disk wird immer Slice 0 angezeigt, die von *RomWBW* bereitgestellte RAM-Disk ist monolithisch und nicht weiter unterteilt.

Den Versuch, auf eine schreibgeschützt gemappte Diskette zu schreiben, quittiert *UDOS* mit einem 'Write protect' Fehler (Error C3)

7.1.2 Diskmapping aufheben

Das Aufheben des Diskmappings, also das Entnehmen einer Diskette, führt **'md'** durch, wenn als Argument die *UDOS*-Laufwerksnummer, gefolgt von einem Gleichheitszeichen `'='`, angegeben wird. Das Resultat wird anschließend angezeigt. Ausgehend vom oben abgebildeten Status:

```
-------------------------------------------------------------
rcbd:md 1=
| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
| 2.12 |      |      |      | 0:00 |
-------------------------------------------------------------
```

Die Diskette aus Laufwerk 1 wurde entnommen.

7.1.3 Diskmapping erstellen

Das Herstellen eines Diskmappings, also das Einlegen einer Diskette, wird durch ein dem Gleichheitszeichen folgendes `device.slice` oder `device:slice` Paar initiiert. Dabei würde ein Punkt `'.'` als Trennzeichen die Diskette schreibgeschützt einlegen, ein Doppelpunkt `':'` würde eine beschreibbare Diskette dem Laufwerk zuordnen. Das Resultat wird anschließend angezeigt. Ausgehend vom oben abgebildeten Status:

```
-------------------------------------------------------------
rcbd:md 1=2.4
| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
| 2.12 | 2.04 |      |      | 0:00 |
rcbd:md 2=4:55
| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
| 2.12 | 2.04 | 4:55 |      | 0:00 |
-------------------------------------------------------------
```

Es wurden zwei weitere Disketten eingelegt, davon eine schreibgeschützt.

RCBD achtet darauf, dass die gleiche Diskette nicht gleichzeitig in mehrere *UDOS*-Laufwerke eingelegt wird, und dass eine Diskette nur in ein leeres Laufwerk eingelegt werden kann. Das entspricht der natürlichen Arbeitsweise am physischen Rechner - bevor man eine Diskette in ein bestimmtes Laufwerk einlegt, muss man eine andere darin befindliche Diskette entnehmen. Außerdem werden Laufwerks-, Device- und Slicenummern auf Gültigkeit und Existenz überprüft.

7.1.4 Fehlermeldungen von 'md'

'md' kann folgende Fehlermeldungenen ausgeben:

```
Invalid UDOS drive number
Unsupported HBIOS device number
Unusable HBIOS device (no HD in hd1k format or no RAM disk)
Invalid slice number for device
Drive not empty (a different slice is already mapped)
Slice already mapped to another drive
Drive and device incompatible (FD in RAM drive or so)
```

Eine detailierte Beschreibung der Fehlermeldung finden Sie in Anlage G.4.

7.1.5 Nutzung der RAM-Disk

Startet *RCBD* nach einem Power On, hat der RAM-Chip, der die RAM-Disk bereitstellt, undefinierte Daten. Nach dem Mappen des RAM-Disk-Devices in das Laufwerk 4 muss die RAM-Disk formatiert werden, bevor sie als *UDOS*-Datenträger nutzbar ist.

```
------------------------------------------------------------------
rcbd:md
| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
| 2.12 |      |      |      |      |
rcbd:md 4=0:0
| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
| 2.12 |      |      |      | 0:00 |
rcbd:format d=4 id='Temporary RAM-Disk 256k' q=n
RCBD Disk Formatter 241107
rcbd:status

Drive 0   RCBD-Sys CF2.12 241108
Floppy Disk - Type 5 - 80 tracks  32 sectors
1858 sectors used
 702 sectors avialable

Drive 4   Temporary RAM-Disk 256k
   RAM Disk - Type 5 - 32 tracks  32 sectors
   9 sectors used
1015 sectors avialable
------------------------------------------------------------------
```

Die Formatierung der RAM-Disk und deren Inhalt bleiben erhalten, solange die *RCBD*-Hardware nicht ausgeschaltet wird oder im Falle eines Resets (Taste) nicht vor dem nächsten Start des *RCBD* ein anderes, von *RomWBW* bereitgestelltes Betriebssystem (z. B. CP/M) gebootet wird.

Folgende Aktivitäten verändern nicht den Inhalt der RAM-Disk:

- Unmappen und späteres neu Mappen der RAM-Disk

- Aufruf des *RCBD-ROM*-Monitors/Debuggers und Initiieren eines neuen *UDOS*-Bootvorgangs (`os` Kommando im Debugger)
- Aufruf des *RCBD-ROM*-Monitors/Debuggers und Rückkehr zum *HBIOS*-Bootprompt (`qx` Kommando im Debugger), anschließendes Laden des *RCBD-ROM* und Booten von *UDOS*
- Hardware Reset (Taste) ohne zwischenzeitliches Power Off, Laden des *RCBD-ROM* und Booten von *UDOS*

7.1.6 Initiales Mapping

Um das *UDOS* starten zu können, benötigt man ein initiales Diskmapping für das Laufwerk 0 auf ein Slice mit dem Image einer *RCBD*-Systemdiskette. Beim Start des *RCBD-ROM* wird daher dem Laufwerk 0 ein Standardslice als schreibgeschützte Diskette zugeordnet. Die Zuordnung basiert auf einer Designentscheidung, die ich getroffen habe und deren Umsetzung im *RCBD-ROM* kodiert ist.

- In der Initialisierungsphase des *RCBD-ROM* werden alle für *RCBD* nutzbaren *RomWBW*-Devices ermittelt. Das Ergebnis sieht dann z. B. so aus, wie in Punkt 6.3 gezeigt und in Punkt 6.4 beschrieben.
- Die Systemdiskette wird in dem Harddisk-Device erwartet, welches die niedrigste Devicenummer hat, im abgebildeten Beispiel also im Device 2.
- Bei Datenträgern mit weniger als 15 Slices wird die Systemdiskette in Slice 0 des Devices erwartet.
- Bei Datenträgern mit 15 oder mehr Slices muss das Image der Systemdiskette in Slice 12 abgelegt werden.

Hintergrund dieser Festlegung ist, dass bei Medien (CF-/SD-Karten etc.) höherer Kapazität in den unteren Slices Platz bleibt für die von *RomWBW* serienmäßig mitgebrachten Betriebssystemimages von CP/M, BASIC, Forth etc.. Nutzt man diese Betriebssysteme nicht, können natürlich die Slices unterhalb von 12 für *UDOS*-Disketten genutzt werden. Nur die Position der Systemdiskette in Slice 12 ist hart kodiert.

Eine weitere Festlegung, die jedoch 'weich', d. h. nicht im Programmcode festgehalten ist, ist die Lage des *RCBD-ROM*-Loaders für die Startart gemäß Punkt 6.3. Hier sieht meine (nicht bindende) Organisation wie folgt aus:

- Bei Datenträgern mit weniger als 15 Slices speichere ich den *RCBD-ROM*-Loader im höchsten Slice des Datenträgers ab und habe nur das *RCBD-ROM* für eine Prozessorarchitektur (Z80 oder Z180) auf dem Datenträger.

- Bei Datenträgern mit 15 oder mehr Slices speichere ich den *RCBD-ROM*-Loader für das Z180-*RCBD-ROM* in Slice 10 und das Z80-*RCBD-ROM* in Slice 11 ab. Damit kann ich diese Datenträger zwischen meinem Z80-Aufbau und dem Z180-Aufbau umstecken.

Siehe Anlage C für typische Datenträgergrößen.

7.1.7 Entnahme der Systemdiskette

Was tun, wenn man die Systemdiskette entnommen hat und dadurch des **'md'** Dienstprogramm nicht mehr zur Verfügung steht?

```
-----------------------------------------------------------------
rcbd:md 0=
| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
|      |      |      |      | 0:00 |
-----------------------------------------------------------------
```

Dazu gibt es mehrere Lösungsansätze. Den *RCBD* neu starten ist davon der schlechteste.

Variante 1 ist, das **'md'** Dienstprogramm auf alle Disketten, mit denen man arbeitet, zu kopieren. *UDOS* durchsucht beim Aufruf eines externen Programms alle bereiten Laufwerke in aufsteigender Reihenfolge, bis es das gesuchte Programm gefunden hat. So lange also irgend eine entsprechend vorbereitete Diskette in einem Laufwerk liegt, steht **'md'** zum erneuten Mappen der Systemdiskette ins Laufwerk 0 zur Verfügung.

Man könnte also bespielsweise **'md'** zu Beginn der Arbeit gleich auf die frisch formatierte RAM-Disk kopieren, die man während des gesamten Arbeitszyklus im allgemeinen nicht entnimmt.

```
-----------------------------------------------------------------
rcbd:md
| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
| 2.12 |      |      |      |      |
rcbd:md 4=0:0
| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
| 2.12 |      |      |      | 0:00 |
rcbd:format d=4 id='Temporary RAM-Disk 256k' q=n
RCBD Disk Formatter 241107
rcbd:copy 0/md 4/md o
-----------------------------------------------------------------
```

Variante 2 nutzt die Eigenschaft des **OS** aus, dass ein ausgeführtes Programm nach Beendigung im Speicher verbleibt und neu angestartet werden kann, solange der Speicherbereich nicht durch ein anderes Programm überschrieben wurde. Dazu wird das interne **OS** Kommando `x[ecute]` benutzt.

```
-----------------------------------------------------------------
rcbd:md 0=
```

```
| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
|      |      |      |      |      |
rcbd:md 0=2.12
Nonexistent command
rcbd:x * 0=2.12
| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
| 2.12 |      |      |      |      |
-----------------------------------------------------------------
```

Variante 3: Im *RCBD-ROM*-Monitor/Debugger wurde ebenfalls ein Kommando `md` implementiert, auf das zurück gegriffen werden kann. Dies ist funktionell identisch zum externen **'md'** Dienstprogramm, benutzt aber eine vereinfachte Syntax (bei 3 KiB Platz im residenten Teil muss man Abstriche an die Schönheit machen).

- Alle Zahlenangaben (*UDOS*-Laufwerksnummer, Device- und Slicenummern) sind hexadezimal einzugeben bzw. werden in Hex angezeigt
- Der Schreibschutz einer Diskette wird aufgehoben, indem beim Mappen die Laufwerksnummer mit gesetztem Bit 7 notiert wird.
- Device- und Slicenummer sind als 16bit Zahl zusammen gefasst, wobei das MSB die Devicenummer enthält und das LSB die Slicenummer

Die Beispiele aus den Punkten 7.1.1 bis 7.1.3 hier mit Kurzkommentaren in der `md`-Variante aus dem *RCBD-ROM*.

```
-----------------------------------------------------------------
rcbd:md                                 ; Ausgangssituation
| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
| 2.12 | 4:14 |      |      | 0:00 |
rcbd:d                                  ; Aufruf ROM-Monitor/Debugger
>md                                     ; Mapping anzeigen
2.0c 4:0e *.ff *.ff 0:00
>md 1                                   ; Mapping LW 1 löschen
2.0c *.ff *.ff *.ff 0:00
>md 1 204                               ; Mapping LW 1 Read only
RC = 00
2.0c 2.04 *.ff *.ff 0:00
>md 82 437                              ; Mapping LW 2 Read/Write
RC = 00
2.0c 2.04 4:37 *.ff 0:00
>q                                      ; Rückkehr ins OS
rcbd:md                                 ; Aktueller Zustand
| Drv0 | Drv1 | Drv2 | Drv3 | Ram4 |
| 2.12 | 2.04 | 4:55 |      | 0:00 |
-----------------------------------------------------------------
```

8 RCBD spezifische UDOS Kernkomponenten

8.1 RCBD-ROM

RCBD-ROM ist die residente Komponente des *RCBD*. Sie entspricht funktionell dem 3 KiB ROM der Zilog Systeme bzw. den 3 KiB residenter Software, die bei den DDR *UDOS*-Implementierungen von speziell reservierten Sektoren der Systemdiskette oder einem versteckten ROM in den unteren Teil des Hauptspeichers geladen wurde. Der Code von *RCBD-ROM* belegt den Speicherbereich `0x0000..0x0BFF`, zusätzlich wird 1 KiB operativer Speicher im Bereich `0x0C00..0x0FFF` benutzt. Dieser operative Speicher (System-RAM) wird nach dem Booten von *UDOS* auch von den Komponenten **OS** und **NDOS** benutzt.

Die *RCBD-ROM*-Ladedatei enthält weiteren Code, der zur Initialisierung dient und nur beim ersten Start des *RCBD-ROM* nach dem Laden einmalig ausgeführt wird. Dieser liegt ab Adresse `0x1000` im Hauptspeicher und wird anschließend durch das Booten von *UDOS* überschrieben.

Ein dritter Codeabschnitt in der *RCBD-ROM*-Ladedatei enthält Code, der während der *RCBD-ROM*-Initialisierung in eine andere RAM-Bank (bei mir Shadow Bank genannt) kopiert wird und für *UDOS* und seine Dienst- und Anwendungsprogramme unsichtbar ist. Diese Programmteile enthalten den Code, der nicht *UDOS*-typisch ist, aber für die Interaktion zwischen *RCBD-ROM* und der *HBIOS*-Schnittstelle gebraucht wird. Diese Routinen werden aus dem *RCBD-ROM* über *HBIOS*-Aufrufe angesprochen, z. B. innerhalb des `$FLOPPY`-Treibers.

8.2 OS

Die Datei **'OS'** wurde auf Byteebene minimal gepatcht. Zum einen benutzt das **OS** als Kernkomponente von *UDOS* einen statischen Stack von 128 B, der innerhalb seiner Programmstruktur `0x1000..0x20FF` und somit in der User-Bank unterhalb von `0x8000` liegt. Dieser Stack wurde in die Common-Bank verlegt.

UDOS Dienst- und Anwendungsprogramme fordern beim Laden dynamisch einen Stack an, dessen Größe in der Dateieigenschaft `stack_size` festgelegt ist und die ein ganzzahliges Vielfaches von 128 B (`0x80`) beträgt. Die originalen Routinen im **OS** weisen als Stack einen Block der angeforderten Größe am unteren Ende des größten zusammenhängenden freien Hauptspeicherblocks zu. Dieser Bereich liegt gewöhnlich unterhalb der Adresse `0x8000` und somit in der User-Bank. Die Änderung im **OS** sorgt dafür, dass dynamisch zugewiesener Stack garantiert in der Common-Bank, also oberhalb der Adresse `0x8000` liegt.

8.3 NDOS

Der Treiber `$NDOS` benötigt auf Grund der komplexeren Dateiverwaltung gegenüber dem `$ZDOS` von *RIO* und früheren *UDOS*-Versionen mehr Speicher und zusätzliche Puffer. Diese liegen ursprünglich am oberen Ende des Hauptspeichers und belegen 1280 B (`0x500`). Da im *RCBD* am oberen Ende das Hauptspeichers der *HBIOS*-Proxy und die umgelegten statischen Stacks liegen, wurde dieser Bereich um 1280 B nach unten verlagert und belegt jetzt den Adressraum `0xF600..0xFAFF`.

8.4 PATCHOS

```
Usage: PATCHOS [-R]
```

'PATCHOS' ist ein residentes Programm, welches den Algorithmus im **OS** zur Vergabe dynamischen Stacks an aufgerufene externe Programme im oberen Speicherbereich weiter optimiert. **'PATCHOS'** kann zur Laufzeit von *UDOS* geladen oder entladen werden. **'PATCHOS'** lässt sich gut in der Datei **'os.init'** unterbringen, so dass es beim Booten von *UDOS* automatisch geladen wird.

```
-----------------------------------------------------------------
rcbd:PATCHOS                                          ; Laden
OS successful patched with HighStack Patch for RCBD
rcbd:PATCHOS -R                                       ; Entladen
HighStack Patch for RCBD removed from OS
-----------------------------------------------------------------
```

Anlage G.3 enthält die vollständige Liste aller Meldungen, die **'PATCHOS'** erzeugen kann.

8.5 $PRTC

`$PRTC` ist ein Treiber ohne Vorbild im P8000 *UDOS*, der den *UDOS*-konformen Zugriff von Anwendungsprogrammen auf die Hardware einer von *RomWBW* unterstützten Echtzeituhr (RTC) ermöglicht. Der Treiber ist ROM-resident im *RCBD-ROM* untergebracht.

In Anlage I sind Aufruf- und Rückgabeparameter des Treibers beschrieben.

9 Neue und modifizierte Dienstprogramme

UDOS Dienst- und Anwendungsprogramme werden, soweit sie nicht in diesem Kapitel explizit beschrieben werden, wie in der originalen *RIO*- bzw. *UDOS*-Dokumentation benutzt.

Im *UDOS*-Dateisystem sind Dateinamen case sensitiv. D. h. dass externe Dienst- und Anwendungsprogramme, die als Datei von der Diskette geladen werden, in der Schreibweise aufgerufen werden müssen, in der sie auf der Diskette gespeichert sind. Ich habe (bereits in den 80er Jahren) auf meinen *UDOS*-Rechnern alle interaktiv aufgerufenen Dateien auf den Systemdisketten in Kleinschreibung umbenannt. Das gefiel mir besser und war vom Tastaturhandling 'natürlicher'. Bestimmte Dateien, die nicht interaktiv, sondern als Overlay von anderen Dateien aufgerufen werden, habe ich in Großschreibung belassen. So ruft beispielsweise der Z80 Assembler **'ASM'** (auf meinen Systemdisketten **'asm'**) die Overlaydateien **'ASM2'** und **'ASM3'** auf, die weiterhin in Großschreibung auf den Disketten gespeichert sind.

9.1 format

```
Usage: format [d=<drive>] [id='<max_24_char'] [q=n] [s] [ll=y]
```

'format' ist das Formatierungsprogramm für Disketten in *RIO* und *UDOS*. Für den *RCBD* wurde dieses Programm dahingehend modifiziert, dass beim Formatieren einer Diskette keine Lowlevel-Formatierung erfolgt. Das ist einerseits technisch nicht notwendig, da es sich im *RCBD* nicht um magnetische Datenträger sondern um Speicherabbilder (Images) handelt. Zum anderen ist es natürlich sinnvoll, die flashbasierten Datenträger, die die Images der Disketten enthalten (CF-/SD-Karten etc.), nicht unnötig mit Schreiboperationen zu belasten, da dies ihre Lebensdauer reduziert.

'format' auf dem *RCBD* initialisiert daher lediglich die Disk-Allocation-Map, legt ein leeres Inhaltsverzeichnis (**'DIRECTORY'**) an und kopiert optional die Systemsektoren (*UDOS*-Bootloader), wenn die Option zum Formatieren einer Systemdiskette angegeben wurde.

Eine Pseudo-Lowlevel-Formatierung (Beschreiben alle Sektoren der Diskette mit dem Füllbyte `0xE5` kann durch eine neue Kommandozeilenoption `ll=y (LowLevel=Yes)` erzwungen werden. Alle anderen Optionen arbeiten wie in der originalen Programmversion.

9.2 status

```
Usage: status [<drive>]
```

'status' zeigt den Status eines bestimmten oder aller bereiten Diskettenlaufwerke und die Belegungsstatistik der eingelegten Diskette(n) an. Für den *RCBD* wurde **'status'** um die Ausgabe des Status der RAM-Disk erweitert.

9.3 md

```
Usage: md
       md <drv>=
       md <drv>=[<device>]{.|:}<slice>
```

Dieses neue Dienstprogramm wurde bereits in Kapitel 7 beschrieben.

9.4 copy.disk

```
Usage: copy.disk s=<src_drv> d=<dst_drv> [-b|-r]
```

'copy.disk' kopiert in seiner Grundfunktion ohne die Optionen `-b` oder `-r` komplette *UDOS*-Disketten (natürlich deren Images) sektorweise von einem Laufwerk auf ein anderes. Es kann nur zwischen Diskettenlaufwerken kopiert werden. Ein Kopieren zwischen Diskettenlaufwerk und RAM-Disk Laufwerk ist auf Grund der unterschiedlichen Geometrien nicht möglich.

Quelllaufwerk `src_drv` kann jedes der vier Diskettenlaufwerke sein, als Ziellaufwerk `dst_drv` können nur die Laufwerke 1 bis 3 dienen. Das schützt die Systemdiskette in Laufwerk 0 vor Überschreiben, falls diese nicht schreibgeschützt gemappt ist.

Die Diskette im Quelllaufwerk kann schreibgeschützt gemappt sein. Im Ziellaufwerk muss die Diskette beschreibbar gemappt sein. Im Ziellaufwerk muss nicht unbedingt eine formatierte Diskette gemappt sein, auch ein leerer oder von Fremdsystemen befüllter Slice würde mit der Kopie der Diskette aus dem Quelllaufwerk überschrieben werden.

Jede Kopieroperation muss mit `y[es]` bestätigt werden, alle anderen Tasten brechen das Programm ab.

```
------------------------------------------------------------------
rcbd:copy.disk s=1 d=3
Copy the whole disk in drive 1 to drive 3 [Ny]?
------------------------------------------------------------------
```

Mit **'copy.disk'** können Sicherheitskopien ganzer Disketten gemacht werden, idealerweise auf ein Slice in einem anderen Device.

Mit der `-b` Option wird nicht die gesamte Diskette kopiert, sondern nur der primäre und der sekundäre Bootloader einer Systemdiskette. Es wird überprüft, ob im Quelllaufwerk eine Systemdiskette mit einem *RCBD*-Bootloader liegt. Es wird nicht überprüft, ob im Ziellaufwerk eine formatierte Systemdiskette liegt. Ist im Ziellaufwerk eine Nicht-Systemdiskette eingelegt, kommt es zu Datenverlust, da auf Nicht-Systemdisketten die Sektoren für den Bootloader nicht reserviert sind und somit bereits von regulären Dateien belegt sein könnten.

```
------------------------------------------------------------------
rcbd:copy.disk s=0 d=2 b
Copy UDOS boot loader from drive 0 to drive 2 [Ny]?
------------------------------------------------------------------
```

Mit der `-r` Option werden keine *UDOS*-Disketten oder Teile daraus kopiert, sondern *RCBD-ROM*-Loader-Images. Obwohl es sich bei diesen Images strukturell nicht um Datenträger im *UDOS*-Format handelt, müssen die entsprechenden Slices für den Kopiervorgang in je ein Laufwerk gemappt werden. Es wird überprüft, ob im Quelllaufwerk ein *RCBD-ROM*-Loader-Image gemappt ist. Es wird nicht überprüft, ob im Ziellaufwerk ein leeres Slice oder ein alter *RCBD-ROM*-Loader gemappt ist. Es kommt zu Datenverlust wenn im Ziellaufwerk eine reguläre *UDOS*-Diskette liegt.

```
------------------------------------------------------------------
rcbd:copy.disk s=3 d=2 r
Copy RCBD ROM loader from drive 3 to drive 2 [Ny]?
------------------------------------------------------------------
```

9.5 y2kdate

```
Usage: y2kdate [yymmdd|-c|-r|-v]
```

'y2kdate' ist eine Anpassung des ursprünglichen **'date'** Dienstprogramms an die Jahrtausendwende. *UDOS* verwaltet die Zeitspempel für die Erstellung ('Date of creation') und der letzten Änderung ('Date of Modification') einer Datei in einem 6-stelligen Datenfeld in der Form `YYMMDD`. Dabei wird die zweistellige Jahreszahl implizit den Jahren 1900 bis 1999 zugeordnet.

'y2kdate' interpretiert die Datumsangabe bei der Anzeige des Systemdatums als 1970...1999 bei den Jahreszahlen 70...99 und als 2000...2069 bei den Jahreszahlen 00...69.

Das ist ein rein kosmetischer Vorgang, nur für die Anzeige des Systemdatums. Intern werden Dienstprogramme, die Datumsangaben vergleichen, trotzdem eine Datei mit dem Änderungsdatum '241115' als älter betrachten als eine Datei mit dem Änderungsdatum '890521' da $241115 < 890521$ ist.

Mit den Optionen `-c` oder `-r` kann das Systemdatum direkt aus der RTC gesetzt werden, sofern die Hardware eine solche Komponente beinhaltet.

```
------------------------------------------------------------------
rcbd:date                       ; Anzeige mit mit Original 'date'
Sonnabend, November 15, 1924    ; Datenfeld enthält '241115'
rcbd:y2kdate                    ; Anzeige mit mit 'y2kdate'
Freitag, November 15, 2024
rcbd:y2kdate 890521             ; Manuell ein (altes) Datum setzen
Sonntag, Mai 21, 1989
rcbd:y2kdate -c                 ; Datum aus RTC setzen (wenn vorh.)
Freitag, November 15, 2024
------------------------------------------------------------------
```

Die Originalversion des Dienstprogrammes **'date'** wurde aus Kompatibilitätsgründen auf der *RCBD*-Systemdiskette belassen.

9.6 set.date

```
Usage: set.date [-c|-r|-o|-D<yymmdd>|-v] [{-j|-y}<jahr>] \
                                 [-m<monat>] [{-t|-d}<tag>]
```

Interaktives Setzen des Systemdatums. Die Datumselemente Jahr (`-j, -y`), Monat (`-m`) und Tag (`-t, -d`) können in der Kommandozeile angegeben werden, fehlende Elemente werden interaktiv abgefragt.

```
-----------------------------------------------------------------
rcbd:set.date
Tag?    (1...31) : 4
Monat?  (1...12) : 10
Jahr?   (0...99) : 24

Freitag, Oktober  4, 2024
rcbd:set.date -j24 -m11
Tag?    (1...31) : 15

Freitag, November 15, 2024
-----------------------------------------------------------------
```

Mit den Optionen `-c` oder `-r` (Datum aus RTC), `-o` (bisher eingestelltes Datum) oder `-Dyymmdd` (genau spezifiziertes Datum) kann auch ein Basisdatum gesetzt werden, von welchem mit den Jahr-, Monat- und Tag-Optionen einzelne Bestandteile selektiv verändert werden können.

```
-----------------------------------------------------------------
rcbd:set.date -c                        ; Datum aus RTC lesen

Sonnabend, November 16, 2024
rcbd:set.date -c -t10                   ; Datum aus RTC lesen und Tag
                                        ; modifizieren
Sonntag, November 10, 2024
-----------------------------------------------------------------
```

9.7 do / ndo

Das Dienstprogramm **'do'** zur Ausführung von Command Files wurde nicht verändert. Da der Adressbereich, in den **'do'** geladen wird, sich nicht mit den Adressbereichen überschneiden darf, in die die im Command File enthaltenen Programme geladen werden, ist eine zweite Version von **do** auf der *RCBD*-Systemdiskette, die auf eine höhere Ladeadresse gelinkt wurde. Diese hat den (frei gewählten) Namen **'ndo'**. Mit diesen zwei Versionen von **do** sollten alle Fälle abgedeckt sein.

```
-----------------------------------------------------------------
```

```
rcbd:cat f=l p=& *do do.obj
                  FILE RECORD RECORD FILE  START   DATE OF DATE OF
 FILENAME   DRIVE TYPE COUNT  LENGTH PROPS ADDRESS CREAT.  LAST MOD.

 do           0    P     1     0400  SF     7c00    791019  241108
 ndo          0    P     1     0400  SF     e000    870311  241108
 do.obj       0    B     6     0100                 861218  241108

    97 FILES EXAMINED
     3 FILES LISTED
    17 TOTAL SECTORS FOR LISTED FILES
-----------------------------------------------------------------------
```

Sollte der Anwender **do** auf eine weitere Adresse linken wollen, kann er die auf der Systemdiskette enthaltene Objektdatei **'do.obj'** gemäß seinen Wünschen linken. Dabei muss, abweichend von den Defaultwerten des Linkers die Stackgröße auf 256 B (`0x100`) gesetzt werden. Die Wahl der Recordlänge hat untergeordnete Bedeutung. Da **do** sich selbst aufrufen kann (ein Command File enthält den Aufruf eines anderen Command Files), muss **do** zur korrekten Ausführung die Dateieigenschaft 'Force' (`F`) erhalten.

```
-----------------------------------------------------------------------
rcbd:ndo sysrd.link do 0c800 st=100 rl=400
PLINK 4.0
LINK COMPLETE
rcbd:set properties of 4/do to f
rcbd:cat f=l p=& *do
                  FILE RECORD RECORD FILE  START   DATE OF DATE OF
 FILENAME   DRIVE TYPE COUNT  LENGTH PROPS ADDRESS CREAT.  LAST MOD.

 do           0    P     1     0400  SF     7c00    791019  241108
 ndo          0    P     1     0400  SF     e000    870311  241108
 do           4    P     1     0400  F      c800    241118  241118

   100 FILES EXAMINED
     3 FILES LISTED
    15 TOTAL SECTORS FOR LISTED FILES
-----------------------------------------------------------------------
```

9.8 sys.link / sysrd.link

```
Usage: do sys.link <object_file> <load_address> \
                        [<opt_arg1> [<opt_arg2> [<opt_arg3>]]]
       do sysrd.link <object_file> <load_address> \
                        [<opt_arg1> [<opt_arg2> [<opt_arg3>]]]
```

'sys.link' und **'sysrd.link'** sind keine Dienstprogramme, sondern Command Files (Stapelverarbeitungs- oder Batchdateien). Ein Command File ist eine Textdatei, die eine Folge von *UDOS*-Dienstprogrammen mit einem einzigen Aufruf abarbeiten kann oder die wiederholte Ausführung von

Dienstprogrammen mit einer Vielzahl von Kommandozeilenargumenten vereinfacht.

Die beiden Command Files sind dazu gedacht, eigene Programme, die unter *UDOS* laufen sollen, gegen die globalen Systemadressen des Systemkerns zu linken. **'sys.link'** und **'sysrd.link'** unterscheiden sich nur dadurch, dass das beim Linken entstehende Procedure File (ausführbare Programm) bei **'sys.link'** auf Laufwerk 1 abgelegt wird, bei **'sysrd.link'** auf Laufwerk 4 (RAM-Disk).

Als Beispiel das bekannte 'Hello world' Programm unter Nutzung einer globalen (public) Routine aus dem OS. Zuerst das Quellprogramm **'helloworld.s'**.

```
------------------------------------------------------------------
; Das Hello World Programm
; Aus OS:
        external PUTMSG         ; global in OS

        global   entry          ; Load- & start address

entry   ld      hl,hwmsg
        ld      c,(hl)
        ld      b,0             ; BC = Stringlength
        inc     hl
        call    PUTMSG          ; Call PUTMSG from OS
        ret                     ; Return to UDOS

hwmsg   deft    'Hello world'
        end
------------------------------------------------------------------
```

Die Erstellung des Procedure Files: Assemblieren und Linken. Der Assembler wird hier direkt aufgerufen, da der Aufruf sehr simpel ist. Der Linker muss die **'*.obj'**-Dateien des Systemkerns zusammen mit der Datei **'helloworld.obj'**, die beim Assemblieren entstanden ist, verbinden. Da das eine komplexe Kommandozeile ist, wurde es in ein Command File ausgelagert.

Die beiden Command Files können bis zu drei optionale Parameter an den Linker übergeben. In diesem Beispiel wurde die Recordlänge des Procedure Files spezifiziert sowie die benötigte Stackgröße. Die optionalen Parameter müssen nur angegeben werden, wenn der Linker von den Defaulteinstellungen abweichende Parameter benutzten soll.

```
------------------------------------------------------------------
rcbd:asm helloworld
ASM 5.9 RCBD
PASS 1 COMPLETE
0 ASSEMBLY ERRORS
```

```
ASSEMBLY COMPLETE
rcbd:v                      ; Verbose um zu sehen, was ausgeführt wird
rcbd:do sysrd.link helloworld 4000 rl=200 st=100
plink -rcbdrom.dummy $=1000 -os.rcbd $=2100 -con.main -con.rcbd \
          $=2600 -ndos.dummy -last_module $=4000 helloworld \
          (nom e=4000 n=$NDOS:4/helloworld rl=200 st=100
PLINK 4.0
LINK COMPLETE
rcbd:cat f=l helloworld
                  FILE RECORD RECORD FILE  START   DATE OF DATE OF
 FILENAME   DRIVE TYPE COUNT  LENGTH PROPS ADDRESS CREAT.  LAST MOD.

 helloworld   4    P     1     0200         4000    241117  241117

   103 FILES EXAMINED
     1 FILES LISTED
     3 TOTAL SECTORS FOR LISTED FILES
rcbd:helloworld
Hello world
-------------------------------------------------------------------
```

9.9 edit

'edit' ist bis auf eine winzige Änderung der originale zeilenorientierte Editor von *RIO* und *UDOS*. Die Änderung betrifft den Input Mode, in dem fortlaufend Textzeilen eingegeben werden können. Es folgt der Originaltext aus dem Manual:

```
In this mode, all text that is entered from the console is
inserted after the current line. Input mode is terminated
when a null line is entered (by typing just a carriage return).
```

Das Beenden des Input Modes durch alleinige Eingabe von `ENTER` bzw. `RETURN` am Anfang einer neuen Zeile hatte als Konsequenz, dass keine echten Leerzeilen im Text erzeugt werden konnten. Eine optisch leere Zeile im Text musste dadurch erzeugt werden, dass man am Anfang einer neuen Zeile ein einzelnes Leerzeichen eingeben musste (welches dann auch in der Datei gespeichert war), um im Input Mode zu verbleiben.

Das Verhalten wurde dahingehend geändert, dass im Input Mode das Drücken von `ENTER` bzw. `RETURN` direkt am Anfang einer neuen Zeile eine echte Leerzeile (die nur das Zeilenendezeichen beinhaltet) erzeugt. Zum Verlassen der Input Modes muss in der geänderten Editorversion in der neuen Zeile als erstes ein einzelner Punkt `'.'`, unmittelbar gefolgt von `ENTER` bzw. `RETURN` eingegeben werden.

9.10 vi

'vi' ist ein Clone des bekannten gleichnamigen Editors aus der Linuxwelt. Ei-

gentlich handelt es sich um ein bildschirmorientiertes Frontend zum *UDOS*-Editor **'edit'**, dass mit den Tastaturbefehlen des **vi** bedient wird.

Für dieses Programm wird es eine separate Dokumentation geben.

9.11 ttysize

```
Usage: ttysize [-d|-x]
       ttysize [{-r|-l|-z}<rows>] [{-c|-s}<columns>]
```

'ttysize' setzt zwei in **'rcbdrom.dummy.obj'** global vereinbarte 8 bit-Variable `TTYCOL` und `TTYLIN` im System-RAM, die die Bildschirmauflösung des verwendeten Konsolenterminals enthalten. Diese Variablen sind kein Bestandteil früherer *UDOS*-Systeme, sondern wurden im *RCBD* eingeführt, um zukünftige bildschirmorientierte Programme unter *UDOS* zu ermöglichen. Bisher werden diese Variablen nur im **vi**-Editor (9.10) benutzt.

Der Wert dieser Variablen muss mit der Auflösung des verwendeten Terminals oder den Einstellungen des verwendeten Terminalemulationsprogramms übereinstimmen und manuell gesetzt werden. Initial wird während der Initialisierungsphase des *RCBD-ROM* eine Bildschirmauflösung von 80x25 Zeichen eingestellt. Falls das verwendete Terminal eine andere Auflösung als 80x25 benutzt, kann nach dem *UDOS*-Start durch manuelle Ausführung von **'ttysize'** oder durch Einbinden von **'ttysize'** in die Datei **'os.init'** die tatsächlich benötigte Auflösung eingestellt werden.

'ttysize' ohne Optionen zeigt die aktuell eingestellte Auflösung an. Mit den Optionen `-r`, `-l` oder `-z` (Rows, Lines, Zeilen) kann die vertikale und mit `-c` oder `-s` (Columns, Spalten) die horizontale Auflösung eingestellt werden. Die beiden Werte können einzeln oder gemeinsam gesetzt werden.

```
------------------------------------------------------------------
rcbd:ttysize
TTY/Screen size 80x25
rcbd:ttysize -c137 -r34
TTY/Screen size 137x34
------------------------------------------------------------------
```

Die Option `-d`, die allein anzugeben ist, stellt die Auflösung zurück auf den Defaultwert 80x25. Die Option `-x`, die allein anzugeben ist, stellt `TTYCOL` und `TTYLIN` auf Null, was für ein nutzendes Anwendungsprogramm das Zeichen sein sollte, dass keine Auflösungsinformation zur Verfügung steht.

```
------------------------------------------------------------------
rcbd:ttysize -x
TTY/Screen size 0x0
rcbd:ttysize -d
TTY/Screen size 80x25
------------------------------------------------------------------
```

9.12 xm

```
Usage: xm -s [-k] <send_file>
       xm -r [-a|-b] [-c] [-x] <receive_file>
```

Das Dienstprogramm **'xm'** zur Datenübertragung mit dem XMODEM-Protokoll wurde von der in *RomWBW* enthaltenen CP/M-Version abgeleitet, für die im Sourcezweig der *RomWBW*-Software der Quellcode enthalten ist. Die Übertragung erfolgt über die Konsolenschnittstelle, so dass der *RCBD* mit einem Terminalemulationsprogramm bedient werden muss, welches eine XMODEM Übertragung integriert hat (z. B. TeraTerm). **'xm'** arbeitet ausschließlich im Binärmode, d. h. es erfolgt keine Veränderung des Dateiinhalts, z. B. Anpassung von Zeilenendezeichen zwischen *RCBD* und dem Wirtsrechner.

9.12.1 XMODEM senden

'xm' überträgt Daten in 128 B Paketen und optional in 1 KiB Paketen, wenn die Option `-k` angegeben ist (Erweiterung XMODEM-1k). Die Datenübertragung wird bei 128 B Paketen durch eine Prüfsumme (Checksum) oder durch CRC (Erweiterung XMODEM-CRC) gesichert, den Mode fordert der Empfänger entsprechend seiner Implementierung an. Bei XMODEM-1k ist immer die Absicherung durch CRC vorgeschrieben.

Die Sendeblockgröße ist adaptiv. *UDOS*-Dateien sind stets ein Vielfaches von 256 B (Recordlength=0x100) lang. **'xm'** sendet, sofern die `-k` Option angegeben wurde, 1 KiB Pakete, bis der verbleibende Rest weniger als 1 KiB beträgt. Dieser Rest wird in 128 B Paketen übertragen, so dass die empfangene Datei auf dem Wirtsrechner exakt die gleiche Größe hat wie auf dem *UDOS*-Rechner.

```
------------------------------------------------------------------
rcbd:xm -sk xm.s
XMODEM for RCBD 241102
(1k protocol selected)
File size: 194 records (25k)
To cancel: Ctrl-X, pause, Ctrl-X
------------------------------------------------------------------
```

Die Datei **'xm.s'** wird in 1 KiB Paketen gesendet. Auf der Empfangsseite muss Speicherort und Dateiname im XMODEM-Programm des Empfängers angegeben werden, da das XMODEM-Protokoll keine Übertragung des Dateinamens vorsieht.

9.12.2 XMODEM empfangen (receive)

Beim Empfang von Dateien mit **'xm'** kann mit den Optionen `-a` oder `-b` angegeben werden, ob die Datei als *UDOS*-Dateityp `A` (ASCII) oder `B` (Binary)

auf der *UDOS*-Diskette abgespeichert wird. `-b` ist default und kann auch weg gelassen werden. Eine vorhandene Datei gleichen Namens wird überschrieben. Die Option `-c` fordert vom Sender explizit den Checksum-Mode an, die Option den `-x` den 128 B Mode. Inwieweit die sendende XMODEM-Seite diese Wünsche berücksichtigt, ist implementationsabhängig.

Dateien können auf dem Wirtsrechner (Windows- oder Linux-PC) eine beliebige Dateilänge haben. Durch die paketweise Übertragung muss das letzte Paket der Übertragung mit Füllzeichen (`0x1A`) auf 128 B oder 1 KiB aufgefüllt werden, wenn die Dateigröße der Sendedatei nicht zufällig ein exaktes Vielfaches der Paketgröße ist. Die abgespeicherte *UDOS*-Datei wird dadurch fast immer größer sein, als das Original auf dem Wirtsrechner.

```
-----------------------------------------------------------------------
rcbd:xm -ra 4/asciidatei
XMODEM for RCBD 241102
File open - ready to receive
To cancel: Ctrl-X, pause, Ctrl-X
-----------------------------------------------------------------------
```

Die vom Wirtsrechner gesendete Datei wird unter dem Namen **'asciidatei'** mit dem Dateityp `A` auf dem *UDOS*-LAufwerk 4 abgespeichert.

A Einstellungen Terminalprogramm (TeraTerm)

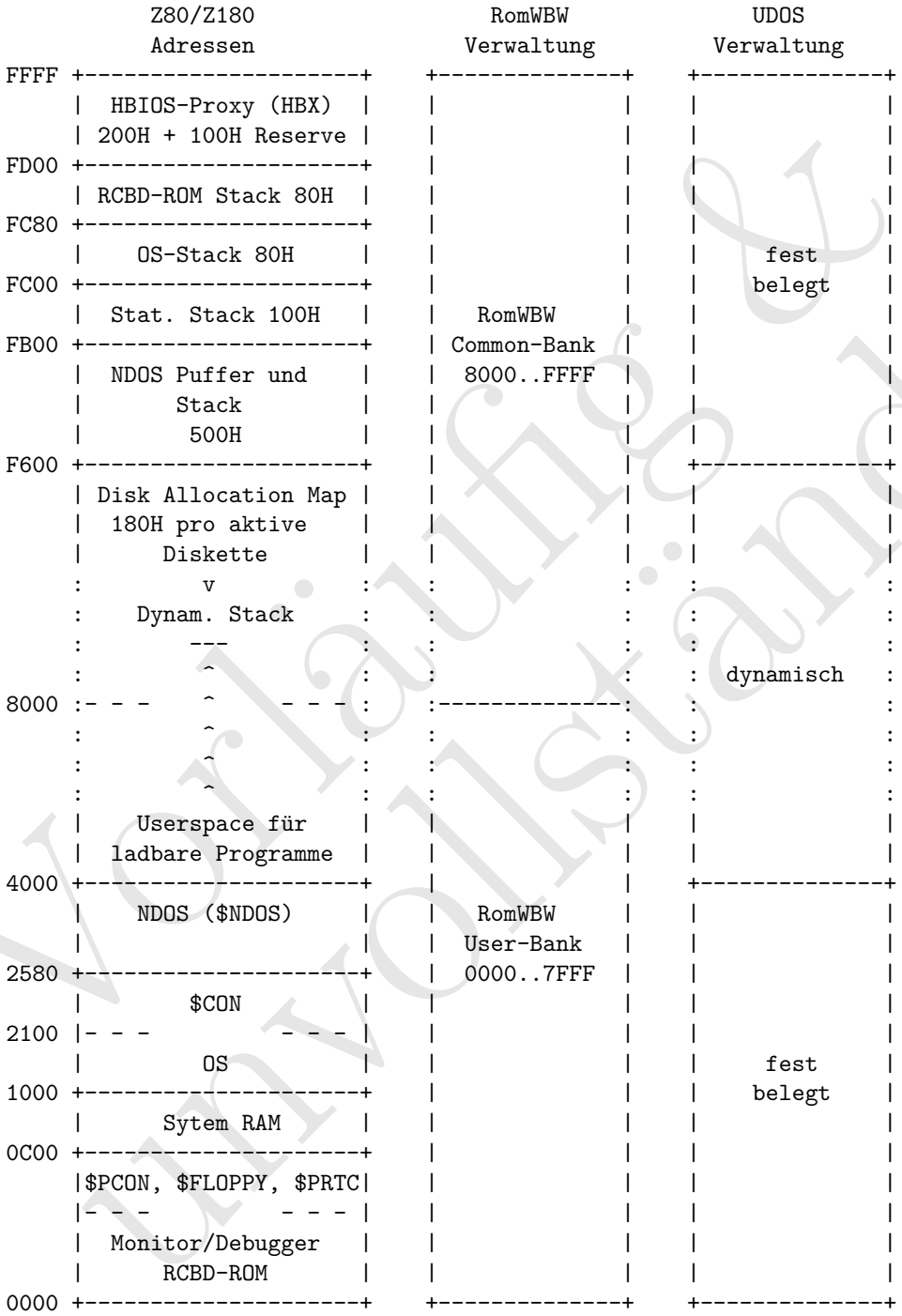
```
Übertragungsparameter
=====================================================================
Serial - COMx
Serial - 115200/8/n/1
                        (Speed entsprechend konkreter RomWBW Umgebung)
Serial - Flow Control - RTS/CTS
                    (oder 'Hardware', je nach Bezeichnung im Produkt)

In TeraTerm
=====================================================================
Serieller Port - Transmit delay: 0 msec/char, 0 msec/line
Terminal Einstellungen - Terminal-ID: VT100
Terminal Einstellungen - Lokales Echo: aus
Terminal Einstellungen - Neue Zeile: Übertrage CR, Absenden CR

Für optimale Nutzung der Cursortasten im vi-Editor muss im
Terminalemulationsprogramm (TeraTerm) eine Tastaturanpassungsdatei
geladen werden.

Die Flusssteuerung in RomWBW HBIOS erfolgt über Hardware.
Ein Download von Programmen oder Daten mit XMODEM oder Intel-HEX
vom Host-Rechner (PC) auf den RomWBW-Rechner kann vom 8-bit Rechner
nicht schnell genug verarbeitet werden. RomWBW signalisiert dem Host-
Rechner über die /RTS-Leitung die Bereitschaft zum Empfang von Daten.
```

B Speicherbelegung



Die Grundbelegung des Hauptspeichers, angezeigt mit dem Dienstprogramm **'display'**.

Die Ausgabe von **'display'** wurde hier redaktionell nachbearbeitet, um die fixe Speicherreservierung durch die Systemkomponenten darzustellen. Das `'A'` der belegten Speicherblöcke durch das gerade geladene Programm **'display'** wurden durch den Kleinbuchstaben `'d'` ersetzt. Die Belegung durch die Disk Allocation Map der Systemdiskette wurde durch den Kleinbuchstaben `'a'` ersetzt.

```
------------------------------------------------------------------------
rcbd:display
                    MEMORY ALLOCATION MAP

            0         400       800       C00
            |   |     |   |     |   |     |   |
0000        AAAAAAAA  AAAAAAAA  AAAAAAAA  AAAAAAAA
1000        AAAAAAAA  AAAAAAAA  AAAAAAAA  AAAAAAAA
2000        AAAAAAAA  AAAAAAAA  AAAAAAAA  AAAAAAAA
3000        AAAAAAAA  AAAAAAAA  AAAAAAAA  AAAAAAAA
4000        dddd....  ........  ........  ........
5000        ........  ........  ........  ........
6000        ........  ........  ........  ........
7000        ........  ........  ........  ........
8000        ........  ........  ........  ........
9000        ........  ........  ........  ........
a000        ........  ........  ........  ........
b000        ........  ........  ........  ........
c000        ........  ........  ........  ........
d000        ........  ........  ........  ........
e000        ........  ........  ........  ........
f000        ........  daaaAAAA  AAAAAAAA  AAAAAAAA

       "." => FREE    "A" => ALLOCATED
------------------------------------------------------------------------
```

C Datenträgergrößen

Kapazität	Größe RomWBW Partition	Anzahl Slices	Systemdisk in Slice	Eignung für RCBD/UDOS
8 MB		0	Medium nicht für RomWBW geeignet	
16 MB	8 MiB 8	1	0	bedingt brauchbar, nur eine UDOS-Disk möglich
32 MB	24 MiB	3	0	Systemdisk, 1 Datendisk RCBD-ROM-Loader
64 MB	56 MiB	7	0	brauchbar
128 MB	120 MiB	15	12	brauchbar
256 MB	240 MiB	30	12	brauchbar
512 MB	488 MiB	61	12	brauchbar
1GB	968 MiB	121	12	brauchbar
Maximal 2 GB RomWBW Partition (max. 256 Slices) bei großen Medien				

Datenträger haben in der Regel eine etwas geringere Kapazität als der aufgedruckte Handelswert. Auch schwankt die tatsächliche Kapazität zwischen Herstellern bzw. Modellen. Ich habe 128 MB CF-Karten unterschiedlicher Hersteller, die mit Kapazitäten zwischen 122 MiB und 125 MiB angezeigt werden.

Die *RomWBW*-Partitionierung für das hd1k Layout erfordert 1 MiB Prefix für die Partitonstabelle und ein ganzahliges Vielfaches von 8 MiB für die *RomWBW*-Partition mit den Slices.

D Systemstart auf Z80-SBC

Hardware ist hier ein SC720[6] Z80 Single Board Computer (enthält bereits einen CF-Kartenadapter) und eine CTC Erweiterungskarte SC718[7]. Ein Echtzeituhrmodul SC727[8] oder SC606[9] als Erweiterung ist geplant.

Die Konfiguration des *RomWBW* ROMs steht in Anlage F.

```
------------------------------------------------------------------------
RomWBW HBIOS v3.5.0-dev.84, 2024-10-07

RCBus [RCZ80_rcbd] Z80 @ 7.372MHz
0 MEM W/S, 1 I/O W/S, INT MODE 2, Z2 MMU
512KB ROM, 512KB RAM, HEAP=0x2FCD
ROM VERIFY: 00 00 00 00 PASS

CTC: IO=0x88
LCD: IO=DA NOT PRESENT
SIO0: IO=0x80 SIO MODE=115200,8,N,1
SIO1: IO=0x82 SIO MODE=115200,8,N,1
DSRTC: MODE=STD IO=0xC0 NOT PRESENT
MD: UNITS=2 ROMDISK=384KB RAMDISK=256KB
FD: MODE=RCWDC IO=0x50 NOT PRESENT
IDE: IO=0x10 MODE=RC
IDE0: ATA 8-BIT LBA BLOCKS=0x0000F500 SIZE=30MB
IDE1: ATA 8-BIT LBA BLOCKS=0x0003E800 SIZE=125MB
PPIDE: IO=0x20 PPI NOT PRESENT
CH0: IO=0x3E NOT PRESENT
CH1: IO=0x3C NOT PRESENT
FP: IO=0x00 NOT PRESENT

Unit         Device      Type               Capacity/Mode
----------   ----------  ----------------   --------------------
Char 0       SIO0:       RS-232             115200,8,N,1
Char 1       SIO1:       RS-232             115200,8,N,1
Disk 0       MD0:        RAM Disk           256KB,LBA
Disk 1       MD1:        ROM Disk           384KB,LBA
Disk 2       IDE0:       Hard Disk          30MB,LBA
Disk 3       IDE1:       Hard Disk          125MB,LBA

RCBus [RCZ80_rcbd] Boot Loader

Boot [H=Help]:
------------------------------------------------------------------------
```

[6] https://smallcomputercentral.com/sc781-rcbus-z80-sbc-motherboard/
[7] https://smallcomputercentral.com/sc718-rcbus-z80-ctc-module/
[8] https://smallcomputercentral.com/sc727-rcbus-rtc-module/
[9] https://smallcomputercentral.com/rcbus/sc600-series-selection/sc606-rcbus-rtc-module/

```
------------------------------------------------------------------------
Boot [H=Help]: 2.2

Booting Disk Unit 2, Slice 2, Sector 0x00008800...

Volume "RCBD Z80 ROM" [0x0000-0x1FFF, entry @ 0x0000]...
Device 0 - RAM Disk
Device 2 - 03 Slice(s)
Device 3 - 0f Slice(s)
No RTC found
UDOS for Z80 RomWBW HBIOS 241101
>
------------------------------------------------------------------------
```

Hier sind neben der RAM-Disk zwei CF-Karten (32 MB, 128 MB) gesteckt. Die angezeigte Kapazität der Karten ist 30 MiB bzw 125 MiB. Eine Echtzeituhr (RTC) ist (noch) nicht vorhanden.

E Systemkonfiguration SC126 Z180-SBC

Die Systemkonfiguration des *RomWBW* ROMs für den SC126 Z180 Single Board Computer basiert auf der Standardkonfiguration **SCZ180_sc126_std**, die unverändert übernommen werden konnte.

```
------------------------------------------------------------------------
ROM Image File: SCZ180_sc126_std.rom
    Default CPU Speed 18.432 MHz
    Interrupts Mode 2
    System Timer Z180
    Serial Default 115200 Baud
    Memory Manager Z180
    ROM Size 512 KB
    RAM Size 512 KB

Supported Hardware
X FP:       LEDIO=0x0D
  FP:       SWIO=0x00
X DSRTC:    MODE=STD, IO=0x0C
  INTRTC:   ENABLED
X ASCI:     IO=0xC0, INTERRUPTS ENABLED
X ASCI:     IO=0xC1, INTERRUPTS ENABLED
  UART:     MODE=RC, IO=0xA0
  UART:     MODE=RC, IO=0xA8
  SIO       MODE=RC, IO=0x80, CHANNEL A, INTERRUPTS ENABLED
  SIO       MODE=RC, IO=0x80, CHANNEL B, INTERRUPTS ENABLED
  SIO       MODE=RC, IO=0x84, CHANNEL A, INTERRUPTS ENABLED
  SIO       MODE=RC, IO=0x84, CHANNEL B, INTERRUPTS ENABLED
  CH:       IO=62
  CH:       IO=60
  CHUSB:    IO=62
  CHUSB:    IO=60
X MD:       TYPE=RAM
X MD:       TYPE=ROM
  FD:       MODE=RCWDC, IO=0x50, DRIVE 0, TYPE=3.5" HD
  FD:       MODE=RCWDC, IO=0x50, DRIVE 1, TYPE=3.5" HD
X IDE:      MODE=RC, IO=0x10, MASTER
X IDE:      MODE=RC, IO=0x10, SLAVE
  PPIDE:    IO=0x20, MASTER
  PPIDE:    IO=0x20, SLAVE
X SD:       MODE=SC, IO=0x0C, UNITS=1
  AY38910: MODE=RCZ180, IO=0x68, CLOCK=1789772 HZ
X = Als Hardware vorhanden
------------------------------------------------------------------------
```

F Systemkonfiguration SC720 Z80-SBC

Die Systemkonfiguration des *RomWBW* ROMs für den SC720 Z80 Single Board Computer basiert auf der Standardkonfiguration **RCZ80_std**, die jedoch für den Einsatz im *RCBD* geändert werden musste. In der Standardkonfiguration wird im Interruptmode 1 gearbeitet, während für den *RCBD* der Interruptmode 2 notwendig ist.

```
------------------------------------------------------------------
ROM Image File: RCZ80_rcbd.rom
    Default CPU Speed 7.372 MHz
    Interrupts Mode 2
    System Timer None
    Serial Default 115200 Baud
    Memory Manager Z2
    ROM Size 512 KB
    RAM Size 512 KB

Supported Hardware
X FP:       LEDIO=0x00
  FP:       SWIO=0x00
  DSRTC:    MODE=STD, IO=0xC0  (geplant)
  UART:     MODE=RC, IO=0xA0
  UART:     MODE=RC, IO=0xA8
X SIO       MODE=RC, IO=0x80, CHANNEL A, INTERRUPTS ENABLED
X SIO       MODE=RC, IO=0x80, CHANNEL B, INTERRUPTS ENABLED
  SIO       MODE=RC, IO=0x84, CHANNEL A, INTERRUPTS ENABLED
  SIO       MODE=RC, IO=0x84, CHANNEL B, INTERRUPTS ENABLED
  ACIA:     IO=0x80, INTERRUPTS ENABLED
  CH:       IO=0x3E
  CH:       IO=0x3C
  CHUSB:    IO=0x3E
  CHUSB:    IO=0x3C
X MD:       TYPE=RAM
X MD:       TYPE=ROM
  FD:       MODE=RCWDC, IO=0x50, DRIVE 0, TYPE=3.5" HD
  FD:       MODE=RCWDC, IO=0x50, DRIVE 1, TYPE=3.5" HD
X IDE:      MODE=RC, IO=0x10, MASTER
X IDE:      MODE=RC, IO=0x10, SLAVE
  PPIDE:    IO=0x20, MASTER
  PPIDE:    IO=0x20, SLAVE
X CTC:      IO=0x88
X = Als Hardware vorhanden
------------------------------------------------------------------
```

Die Anpassungsdatei im **<path>/Source/HBIOS/Config** Verzeichnis.

```
;------------------------------------------------------------------
; File:    Config/RCZ80_rcbd.asm
;==================================================================
; ROMWBW DEFAULT BUILD SETTINGS FOR RCBUS Z80 - Custom Configuration
;   - 2024-10-07 R.-P. Nerlich <early8bitz.de>
;   - Subject:   Modifications for RCBD
;   - Details:   Change interrupt mode to IM2
;==================================================================
;
#INCLUDE "Config/RCZ80_std.asm"
;
CPUOSC     .SET  7372800  ; CPU OSC FREQ IN MHZ
INTMODE    .SET  2        ; INTERRUPTS: 0=NONE, 1=MODE 1, 2=MODE 2,
;                         ;             3=MODE 3 (Z280)
;------------------------------------------------------------------
```

G System- und Fehlermeldungen

Es werden nur *RCBD*-spezifische Meldungen beschrieben. Meldungen aus der allgemeinen *RIO*- bzw. *UDOS*-Dokumentation werden hier nicht aufgeführt.

G.1 Lade- und Initialisierungsmeldungen

`Unsupported RomWBW Version, use 3.4.0 or higher`

Es wurde versucht, ein *RCBD-ROM*-Image auf einer zu alten *RomWBW*-Firmwareversion zu starten. *RCBD* wird nicht gestartet.

`Unsupported platform for Z80 RCBD-ROM detected`

Es wurde versucht, ein Z80-*RCBD-ROM*-Image auf einet Z180 basierenden Hardware zu starten.
- oder -
Es wurde versucht, ein Z80-*RCBD-ROM*-Image auf einer Z80 basierenden Hardware zu starten, die aber nicht kompatibel zu *RCBD* ist.
RCBD wird nicht gestartet.

`Unsupported platform for Z180 RCBD-ROM detected`

Es wurde versucht, ein Z180-*RCBD-ROM*-Image auf einer Z80 basierenden Hardware zu starten.
- oder -
Es wurde versucht, ein Z180-*RCBD-ROM*-Image auf einer Z180 basierenden Hardware zu starten, die aber nicht kompatibel zu *RCBD* ist.
RCBD wird nicht gestartet.

`No free CTC channel found`

Es wurde kein freier CTC-Kanal für den $FLOPPY Treiber gefunden. Alle CTC-Kanäle sind als Taktgeber für eine SIO im *HBIOS* registriert.
RCBD wird nicht gestartet.

`Init Error XX`

Fehler beim Initialisieren ohne expliziten Fehlertext. `XX` kann sein
B0 - Fehler beim Lesen der *RomWBW*-Systeminformationen
B1 - Fehler beim Lesen der *RomWBW*-Bankinformationen
B2 - User-Bank-ID und Shadow-Bank-ID identisch
B3 - Ungültige Bankgröße (nicht 32KiB)
B4 - Fehler beim Lesen von Konsolenparametern

RCBD wird nicht gestartet.

`No avialable Disk Device`

RomWBW hat kein Diskdevice bereit gestellt, dass für *RCBD* geeignet ist. Es kann im *RCBD-ROM*-Monitor/Debugger gearbeitet werden, aber es ist kein *UDOS* startbar.

`Device 0 - RAM Disk`

Ein für *RCBD* nutzbares RAM-Disk-Device wurde gefunden.

`Device 3 - 1e Slice(s)`

Ein für *RCBD* nutzbares Diskdevice wurde gefunden. Das Device enthält 30 Slices (`0x1E`). Device- und Slicenummer sind hexadezimal angegeben.

`No RTC found`

In der Hardware wurde keine von *RomWBW* unterstützte Echtzeituhr (RTC) gefunden. Die *UDOS*-Variable **DATE** wird auf das Build-Datum des *RCBD-ROM* gesetzt. Das aktuelle Datum muss nach dem Start von UDOS manuell mit den Dienstprogrammen **'y2kdate'**, **'set.date'** oder **'date'** eingestellt werden.

`Date set from RTC`

In der Hardware wurde eine von *RomWBW* unterstützte Echtzeituhr (RTC) gefunden. Die *UDOS*-Variable **DATE** wird auf das von der RTC gelesene Datum gesetzt.

G.2 UDOS Startmeldungen

Der *UDOS*-Bootloader meldet, wenn er die Dateien des Betriebssystemkerns **'OS'** oder **'NDOS'** nicht auf der Systemdikette findet.

```
------------------------------------------------------------
>os
rcbdboot: Ok
osload:   File not found: OS
------------------------------------------------------------
```

Ein fehlerfreies Laden des *UDOS*-Betriebssystems verläuft ohne Meldungen. Ausgaben können jedoch durch die Programme erzeugt werden, die zur automatischen Abarbeitung in der Datei **'os.init'** notiert sind.

G.3 Meldungen von PATCHOS

`Not running on a RCBD system`

Es wurde versucht **'PATCHOS'** auf einem anderen *UDOS*-System (z. B. P8000) zu starten.

`OS successful patched with HighStack Patch for RCBD`

Das Patch wurde erfolgreich auf das geladene **OS** angewendet.

`OS already patched with HighStack Patch for RCBD`

Es wurde versucht, das Patch ein zweites Mal anzuwenden, obwohl es schon geladen war.

`Invalid or not prepatched OS version loaded`

Beim Laden des Patches wurde festgestellt, dass die geladene **OS**-Version nicht die für den *RCBD* vorbereitete **OS**-Version ist. Das Patch wird nicht geladen.

`HighStack Patch for RCBD removed from OS`

Das Patch wurde vom geladenen **OS** erfolgreich entfernt.

`OS currently not patched with HighStack Patch for RCBD`

Es wurde versucht, das Patch zu entladen, obwohl es nicht geladen war.

G.4 Fehlermeldungen von 'md'

Das Dienstprogramm **'md'** gibt Fehlermeldungen beim Diskmapping als Text aus. Die residente Version im *RCBD-ROM* meldet die gleichen Fehler als Returncode `RC=XX`.

`Not running on a RCBD system`

Es wurde versucht **'md'** auf einem anderen *UDOS*-System (z. B. P8000) zu starten.

`Invalid UDOS drive number`

RCBD unterstützt 5 *UDOS*-Laufwerke (0...4). Diese Meldung wird ausgegeben, wenn beim Diskmapping eine Laufwerksnummer größer als 4 angegeben wird. In der *RCBD-ROM*-Version wird der Returncode `FF` zurück gegeben.

`Unsupported HBIOS device number`

RCBD kann die ersten acht ermittelten Diskdevices von *RomWBW* nutzen (Devices 0...7). Diese Meldung wird ausgegeben, wenn beim Diskmapping eine Devicenummer größer als 7 angegeben wird. In der *RCBD-ROM*-Version wird der Returncode `FE` zurück gegeben.

`Unusable HBIOS device (no HD in hd1k format or no RAM disk)`

Die angegebene Devicenummer ist ein Diskdevice, welches keine *RomWBW*-Partiton enthält (nicht im 'Modern Layout' (hd1k) vorbereitet) und ist auch keine RAM-Disk. In der *RCBD-ROM*-Version wird der Returncode `FD` zurück gegeben.

`Invalid slice number for device`

Die im Mapping angegebene Slicenummer ist höher als die höchste Slicenummer, die das Device auf Grund seiner Kapazität beherbergen kann. In der *RCBD-ROM*-Version wird der Returncode `FC` zurück gegeben.

`Drive not empty (a different slice is already mapped)`

Das *UDOS*-Laufwerk, das gemappt werden soll, ist nicht leer. Es ist bereits eine Diskette aus einem anderen Slice gemappt. Das Mapping in dem Laufwerk ist vorher zu löschen, bevor eine anderes Slice gemappt werden kann. In der *RCBD-ROM*-Version wird der Returncode `FB` zurück gegeben.

`Slice already mapped to another drive`

Das im Mapping angegebene Slice ist bereits in ein anderes Laufwerk gemappt. Ein Slice kann nicht gleichzeitig in mehrere Laufwerke gemappt sein. In der *RCBD-ROM*-Version wird der Returncode `FA` zurück gegeben.

`Drive and device incompatible (FD in RAM drive or so)`

Es wird versucht, ein Slice auf einem Diskdevice in das RAM-Disk-Laufwerk (Laufwerk 4) zu mappen. In der *RCBD-ROM*-Version wird der Returncode `F9` zurück gegeben.

G.5 Sonstige Meldungen

`Trap @ XXXX`

In einer Z180 basierenden *RCBD*-Umgebung hat die CPU versucht, einen ungültigen Opcode auf der Adresse `XXXX` auszuführen.

H Kommandos des RCBD-ROM-Monitors

ToDo !!

I Aufruf des Treibers $PRTC

`$PRTC` ist als *UDOS*-Treiber geschrieben. Der Aufruf erfolgt über den standardisierten I/O-Vektor, dessen Anfangsadresse im Register `IY` zu übergeben ist. Die Aufrufadresse ist die Adresse `0x0BDC` im Sprungverteiler am Ende des *RCBD-ROM*.

```
I/O-Vektor für den Aufruf des Treibers
IY+0      0x00  Unit, unbenutzt
IY+1      0xrr  rr = Request Code
IY+2,3  0xaaaa  aaaa = Puffer Adresse für Datenfeld
IY+4,5  0x0006  Datenlänge, wird momentan ignoriert
IY+6,7  0x0000  oder Completion return address
IY+8,9  0x0000  oder Error return address
IY+10           Completion Code (wird vom Treiber gesetzt)

Zulässige Request Codes
0x00    Initialize Request      Keine Aktion
0x02    Assign Request          Keine Aktion
0x04    Open Request            Keine Aktion
0x06    Close Request           Keine Aktion
0x0A    Read Binary             Lesen von Datum/Zeit aus der RTC
0x0E    Write Binary            Setzen von Datum/Zeit in der RTC

Mögliche Completion Codes
0x80    Operation complete      Fehlerfreie Abarbeitung des Aufrufs
0x41    Invalid or inactive device
                                Keine RTC im System
0xC1    Invalid Request         Ungülter Request Code
0xC6    Data Transfer Error     Datenübertragung von oder zur
                                RTC fehlgeschlagen
```

Das im angegebenen Puffer zurück gelieferte (Datum/Zeit lesen) bzw. zu übergebende (Datum/Zeit setzen) 6 Byte Datenfeld hat den Aufbau `YYMMDDhhmmss` im gepackten BCD-Format. D. h. zwölf Stellen werden in 6 Byte gespeichert und das Anwenderprogramm muss daraus das für die Anzeige oder Weiterverarbeitung notwendige Format selbst erzeugen.

Offset	Inhalt	Wert
0	YY Jahr	00-99
1	MM Monat	01-12
2	DD Tag	01-31
3	hh Stunde	00-24
4	mm Minute	00-59
5	ss Sekunden	00-59

Die Interpretation der zweistelligen Jahresangabe für ein bestimmtes Jahrhundert obliegt dabei dem Anwender.

J RomWBW Datenträger vorbereiten

Präparieren einer 128 MB CF-Karte mit dem Dienstprogramm **'FDISK80'**, welches unter einer in *RomWBW* enthaltenen CP/M-Version gestartet wurde. Die CF-Karte enthielt in ihrem früheren Einsatz ein Linux.

```
------------------------------------------------------------------------
B>fdisk80
FDISK80 for RomWBW, UNA, Mini-M68k, KISS-68030, SBC-188  ----
       Version 1.1-23 created 3-June-2023
                 (Running under RomWBW HBIOS)

HBIOS unit number [0..3]: 3
Capacity of disk 3:  (125M)  256000      Geom 03e81010
Nr  ---Type- A --     Start         End   LBA start  LBA count  Size
 1     Linux * 83    15:0:1   498:15:32        7680     247808  121M
 2     Linux   83     0:0:2     5:15:32           1       3071    1M
 3     Linux   83     6:0:1    14:15:32        3072       4608    2M
 4             00      *** empty ***
>>I
>>P
Nr  ---Type- A --     Start         End   LBA start  LBA count  Size
 1             00      *** empty ***
 2             00      *** empty ***
 3             00      *** empty ***
 4             00      *** empty ***
>>N1
Starting Cylinder (default 0): +1MB
Ending Cylinder (or Size= "+nnn"): +120MB
>>p
Nr  ---Type- A --     Start         End   LBA start  LBA count  Size
 1     FAT16   06     8:0:1   967:15:16        2048     245760  120M
 2             00      *** empty ***
 3             00      *** empty ***
 4             00      *** empty ***
>>T1
New type (in hex), "L" lists types: 2e
>>P
Nr  ---Type- A --     Start         End   LBA start  LBA count  Size
 1    RomWBW   2e     8:0:1   967:15:16        2048     245760  120M
 2             00      *** empty ***
 3             00      *** empty ***
 4             00      *** empty ***
>>w
Do you really want to write to disk? [N/y]: y
Okay
FDISK exit.

B>
------------------------------------------------------------------------
```

K RCBD auf Datenträger kopieren

Für diesen Vorgang ist ein Rechner (PC) erforderlich, der einen Kartenleser für die gemäß Anlage J vorbereitete Speicherkarte besitzt und das Linux Dienstprogramm **'dd'** ausführen kann. Das kann ein natives Linux System sein, ein Windows-PC mit installierter Cygwin-[10] oder MSYS2-Umgebung[11] oder ein sonstiges System, das Diskimages punktgenau auf einen externen Datenträger schreiben kann. Folgende Annahmen gelten für das Beispiel:

- Der externe Datenträger wird im Kopierrechner als `\dev\sdg` angesprochen
- Die *RomWBW*-Partition auf diesem Datenträger erscheint als `\dev\sdg1`
- Die Imagedatei des *RCBD-ROM*-Loaders heißt `rcbdloaderZ180.img` und soll gemäß meines Designs (7.1.6) in Slice 10 des Datenträgers kopiert werden.
- Die Imagedatei der *RCBD-UDOS*-Systemdiskette heißt `rcbd-sys.img` und soll gemäß meines Designs in Slice 12 des Datenträgers platziert werden.

Zuerst muss die Blockadresse für die beiden Slices berechnet werden. Kopieren wir mit einer Blockgröße von 1 MiB, dann liegt bei einer Slicegröße von 8 MiB der Beginn von Slice 10 bei Blocknummer 80 (10*8) relativ zum Start der *RomWBW*-Partition und der Beginn von Slice 12 bei Blocknummer 96 (12*8). Auszuführen wären dann also folgende Kopierbefehle:

```
-----------------------------------------------------------------
$ dd if=rcbdloaderZ180.img of=/dev/sdg1 bs=1M count=1 seek=80
0+1 records in
0+1 records out
8207 bytes (8.2 kB, 8.0 KiB) copied, 0.0101084 s, 812 kB/s

$ dd if=rcbd-sys.img of=/dev/sdg1 bs=1M count=1 seek=96
0+1 records in
0+1 records out
655360 bytes (655 kB, 640 KiB) copied, 0.262176 s, 2.5 MB/s
-----------------------------------------------------------------
```

Thats all. Dann den Datenträger in die *RomWBW*-Hardware stecken und *RCBD* gemäß Punkt 6.3 starten. Welche Devicenummer der Datenträger in der konkreten *RomWBW*-Hardware hat, erfährt man beim Start von *RomWBW* (Punkt 6.1).

[10] https://www.cygwin.com/
[11] https://www.msys2.org/