Artur W. Klauser

Ramdisk für den NDR-Klein-Computer

Ein schnelles Laufwerk unter CP/M-68k

Daß eine RAM-Floppy mit ihrem 5...10mal schnelleren Zugriff das Arbeiten an einem Rechner deutlich angenehmer macht, braucht wohl nicht besonders betont zu werden. Wie man diesen Komfort beim NDR-Klein-Computer schon mit einer kleinen BIOS-Änderung unkompliziert nutzen kann, steht im folgenden Beitrag.

Vor allem bei massenspeicher-intensiven Aufgaben, etwa Compiler- oder Assembler-Läufen, kommt einem ohne RAM-Floppy der Rechner manchmal wie eine wild gewordenen Nähmaschine vor, mit RAM-Disk aber wird sogar der C-Compiler erträglich. Der einzige Nachteil: Vor dem Abschalten darf man eben auf keinen Fall vergessen, die Daten auf dem richtigen Laufwerk zu sichern...

Die RAM-Disk im BIOS

Wie man sicherlich bald nach Auspakken der CP/M-68k-Systemdisketten bemerkt, befindet sich auf einer davon (Disk 0) eine Routine namens ERRAM, die (wie die Datei Liesmich verrät) für das Löschen einer RAM-Disk vorgesehen ist. Dieses im NDR-BIOS implementierte Laufwerk hat allerdings seine Nachteile: Es arbeitet nur mit 256 KByte RAM, die ab Adresse $80000 untergebracht sein müssen. Außerdem muß nach jedem Kaltstart die Routine ERRAM aufgerufen werden, die dann das nötige Löschen der Speichezellen mit dem Wert $E5 besorgt.
Die hier vorgestellte Betriebssystem-Erweiterung konfiguriert die RAM-Disk (Laufwerk G:) je nach Speicherausbau des Systems, löscht sie bei jedem Kaltstart und nutzt die bereits vorhandenen Routinen zu ihrer Verwaltung. Der Hauptteil des Programmes ist in der BIOS-Routine „_init" untergebracht (*Bild 1*). Sie wird bei jedem Kaltstart automatisch aufgerufen und initialisiert sämtliche Parameter des Systems.

Zunächst wird nach einem zusammenhängenden Bereich gesucht, wobei mit der Suche erst oberhalb des CP/M-68k-Systems begonnen wird, um die TPA (transient program area) nicht zu überschreiben. Außerdem lassen sich eigene Beschränkungen des Suchbereiches (zum Schutz eines Speicherbereiches vor Überschreiben) mit den Parametern „lowram" und „highram" festlegen. Der Wert von „ramstep" gibt die Größe der kleinsten eingebauten RAM-Bausteine an und mit dem Inhalt von „minkap" wird die kleinste zusammenhängende Speichergröße definiert, die als RAM-Floppy akzeptiert wird.

Die Initialisierung

Der zweite logische Abschnitt des Programmes ist das Initialisieren des RAM-Bereichs mit $E5. Dieser Teil ist im Prinzip identisch mit der ERRAM-Routine auf der Systemdiskette, nur daß er keine Fehlermeldung auf der Konsole ausgibt. Alle Fehler führen zum Abbruch des Programmes, ohne daß eine RAM-Disk aufgebaut wird.
Den dritten Teil bildet die Initialisierung des DPBs (disk parameter block). Hier müssen je nach vorgefundenem Speicher zwei Werte verändert werden: In DSM wird die Speicherkapazität der RAM-Disk angegeben, und in DRM die

```
_init:

 move.w #%10010100,iobyte * LO, SI/SO, KEY/GDP einstellen.
 move.l #traphndl,$8c     * trap #3 handler
*                           bis hier altes bios

*
lowram  equ $20000        * niedrigste Adresse ramfloppy
highram equ $e0000        * hoechste Adresse ramfloppy
minkap  equ $10000        * min. Speicherkapazitaet fuer ramfloppy
ramstep equ $2000         * min. Speicherkapazitaet eingesetzter ram-Bausteine

*
rflinit:
 lea 0(pc),a0             * Programmcounter nach d0 holen
 move.l a0,d0
 and.l #$ffff0000,d0      * interessant ueber 64k
 add.l #$10000,d0         * naechste gerade 64k Adresse
 cmp.l #lowram,d0         * aber bis lowram fuer cp/m
 bge suchram
 move.l #lowram,d0        * suche erst ab lowram
suchram:
 jsr testram              * Ramzelle ?
 beq beginn               * wenn ja -> Anfang ramfloppy gefunden
 add.l #ramstep,d0        * sonst erhoehen
 cmp.l #highram,d0        * Ende des Suchbereichs ?
 beq endinit              * dann Testende
 bra suchram              * sonst weitersuchen
beginn:
 move.l d0,d1             * Anfang ramfloppy retten
 move.l d0,ramflo         * fuer bios
suchschl:
 jsr testram              * Ramzelle ?
 bne schluss              * wenn nein -> Ende ramfloppy gefunden
 add.l #ramstep,d0        * sonst erhoehen
 cmp.l #highram,d0        * ende des Suchbereiches ?
 beq schluss              * dann -> Ende ramfloppy
 bra suchschl             * sonst weitersuchen
schluss:
 sub.l d1,d0              * Laenge ramfloppy
```

Bild 1. So wird die Erweiterung in die Routine „_init" eingefügt

```
dpbram:
 dc.w 256        * 256 log. 128 Byte Sektoren
 dc.b 3          * BSH    1024 Bytes / Block
 dc.b 7          * BLM
 dc.b 0          * EXTNT MASK
 dc.b 0
 dc.w 255        * Kapazitaet * 1024 , max 256K patch
 dc.w 63         * Direktory Eintraege-1
 dc.w $c000      * Direktory mask (reserved )
 dc.w 0          * no check
 dc.w 0          * offset

ramflo:
 dc.l $80000
```

Bild 2. Hinter dem Label „ramflo" verbirgt sich nun eine Variable, die im Datensegment abgelegt ist

```
readram:         * RAM FLOPPY
 move.l ramflo,a1  * Quelle
 clr.l d1        * savety
 move.b track,d1
 asl.w #8,d1     * 0..n
 move.b sector,d1 * 1..256(0) ueberlauf reverse
 sub.b #1,d1     * 0..255
 asl.l #7,d1     * * 128 (da Blockgroesse)
 and.l #$fffff,d1 * max
 adda.l d1,a1    * Adresse fertig
```

Bild 3. Die unterstrichenen Zeilen zeigen die einzigen Änderungen in den Routinen „readram" und „writeram"

```
 movea.l dma,a0 * Ziel
 move #128-1,d3
ramget:
 move.b (a1)+,(a0)+
 dbra d3,ramget
 clr.w d0        * no errors
 rts


writeram:        * RAM FLOPPY
 move.l ramflo,a1  * Ziel
 clr.l d1        * savety
 move.b track,d1
 rol.w #8,d1     * 0..n
 move.b sector,d1  * 1..256(0) Ueberlauf reverse
 sub.b #1,d1     * 0..255
 asl.l #7,d1     * * 128 (da Blockgroesse)
 and.l #$fffff,d1
 adda.l d1,a1    * Adresse fertig
 movea.l dma,a0 * Quelle
 clr.b (a1)      * Speichertest kurz
 tst.b (a1)
 bne ramerr
 move.b #$ff,(a1)
 cmp.b #$ff,(a1)
 bne ramerr
 move #128-1,d3
ramput:
 move.b (a0)+,(a1)+
 dbra d3,ramput
 clr.w d0        * no errors
 rts
```

maximale Anzahl der Directory-Einträge. Diese Anzahl wird direkt von der Speicherkapazität abgeleitet. Im Beispiel wurde ein Wert von einem Eintrag pro 4 KByte gewählt, was sich aber im Programm leicht ändern läßt.

```
 move.l d0,d2            * Laenge retten
 cmp.l #minkap,d0        * min. minkak
 blt endinit             * wenn weniger: keine ramfloppy
erram:
 move.l d1,a0            * Anfang
 lsr.l #2,d0             *  / 4 weil langwort
 subq.l #1,d0            * fuer Schleife korrigieren
 move.l #$e5e5e5e5,d3    * ramfloppy mit e5 loeschen
erloop:
 move.l d3,(a0)
 cmp.l (a0)+,d3
 bne endinit             * keine ramfloppy wenn speicherzelle fehlerhaft
 dbra d0,erloop
initdpb:                 * dpb muss initialisiert werden
 lea dpbram,a0           * Anfang dpb fuer ramfloppy
 move.l #10,d0
 lsr.l d0,d2             * Laenge in 1k Bloecken
 move.w d2,6(a0)         * DSM = total storage capacity - 1
 subq.w #1,6(a0)
 lsr.w #2,d2             * Festlegung: 1 dir-Eintrag pro 4k
 move.w d2,8(a0)         * DRM = total number of dir entries - 1
 subq.w #1,8(a0)
*
endinit:
 clr.l d0                * Laufwerk A, User 0 ist Start
 rts

testram:                 * kurzer Ramtest (nur 1. langwort)
 move.l d0,a1
 move.l (a1),d4          * Inhalt retten
 move.l #$55555555,d5    * bekannte Methode ...
 move.l d5,(a1)
 cmp.l (a1),d5
 bne badram
 move.l #$aaaaaaaa,d5
 move.l d5,(a1)
 cmp.l (a1),d5
 bne badram
 move.l d4,(a1)          * Inhalt zurueckschreiben
 clr.l d4                * z-flag setzen wenn test ok
badram:                  * sonst mit z = 0 zurueck
 rts

traphndl:
```

Neben dem Einfügen der Routine „rflinit" muß auch „ramflo" geändert werden. Die Zeile
ramflo equ $80000
am Anfang des BIOS wird gelöscht und durch
ramflo dc.l $80000
im Daten-Segment des Quelltextes ersetzt (*Bild 2*). Dies ist notwendig, da die Anfangsadresse ja jetzt variabel ist.

Die letzte Änderung betrifft die Routinen „readram" und „writeram", die fast unverändert übernommen werden können. *Bild 3* zeigt die nötigen Änderungen: Es sind pro Routine lediglich zwei Zeilen umzuschreiben.

Installation

Die Installation der Erweiterung ist nicht besonders schwierig: Auf der Systemdiskette (Disk 0) ist die Datei NDRBIOS.S untergebracht, in der alle Änderungen durchzuführen sind. Die geänderte Datei (Achtung: Auch den Namen ändern, damit das ursprüngliche BIOS nicht verlorengeht!) wird dann mit dem Aufruf
AS68 NEUBIOS.S
übersetzt und mit
LO68 -R -UCPM -O CPM.REL CPMLIB NEUBIOS.O
in das neue CP/M-68k gebunden. Die Systemroutinen-Sammlung CPMLIB ist auf der Systemdiskette Disk 3 zu finden. Um nun auch vernünftig mit der RAM-Disk arbeiten zu können, muß das System noch auf eine sinnvolle Adresse

gelegt werden. Bei einem Speicherausbau von 256 KByte kann man zum Beispiel mit
RELOC -B19000 CPM.REL CPM.SYS
128 KByte für CP/M und ebensoviel Speicher für die RAM-Disk reservieren. Als Grundsatz gilt, daß der Speicher unterhalb des Betriebssystems diesem zur Verfügung steht und der Speicher oberhalb als RAM-Floppy genutzt wird. Nun steht der ersten Erprobung nichts mehr im Wege. Eine neue Diskette formatieren, mit COPY die Systemspuren initialisieren, die neue Systemdatei CPM.SYS laden und die Diskette ins Boot-Laufwerk stecken. Nach Reset und Booten müßte sich das System wie gewohnt melden. Ist das nicht der Fall, so muß man die Änderungen im BIOS prüfen, berichtigen und ein neues System erzeugen.
Die RAM-Disk kann nun als Laufwerk G: angesprochen werden. Die genauen Daten über die Speicherkapazität und das Inhaltsverzeichnis kann man jederzeit mit
STAT G: DSK:
in Erfahrung bringen.

Arithmetik für Textprogramme

Das von VDW in Seefeld für CP/M- und MS-DOS-Systeme angebotene Programm „Simpel-Rechnen" stellt eine nützliche Ergänzung zu Textverarbeitungs-Programmen dar: Es lädt eine Textdatei, die mit Eingabewerten und Markierungen für Ergebnis-Werte versehen ist, und generiert eine Ergebnis-Datei, die an den gewünschten Stellen die (evtl. gerundeten) Ergebnisse arithmetischer Operationen enthält und die man dann drucken kann. Auf diese Weise kann man zum Beispiel Rechnungen mit dem Textprogramm generieren und mit Simpel-Rechnen die summierten Werte und Mehrwertsteuer-Beiträge einfügen.
Damit das Programm weiß, nach welchen Formeln gerechnet und auf welche Stellenzahl gerundet werden soll, ist eine sogenannte Definitions-Datei nötig, die zum Beispiel so aussieht:

```
1 = ##,##
2 = ##,##
1: F1 = 10/3 : S1 = S1+F1
2: F1 = S1
```

In der Texdatei könnte schlicht folgendes stehen:

```
#1
#1
#1
#2Su.:
```

Wenn man nun Simpel-Rechnen startet, wird eine neue Datei mit folgendem Inhalt erzeugt:

```
      3,33
      3,33
      3,33
Su.: 10,00
```

Die mit einem Doppelkreuz gekennzeichneten Felddefinitionen müssen also immer am Zeilenanfang der Textdatei stehen. Pro Zeile sind maximal 50 Felder (F-Variable) möglich. An arithmetischen Operationen sind Addition, Subtraktion, Division, Multiplikation und Rundung möglich. Bei unkorrekten Angaben erfolgen Fehlermeldungen im Klartext.
Das Programm macht im Prinzip auf diese Weise aus jedem „dummen" Textprogramm eines, das auch rechnen kann.

Selbstverständlich stellt sich hier immer die Frage der effizienten Praktikabilität, denn immerhin muß man vor dem Ausdruck jedesmal erst das Textprogramm verlassen, Simpel-Rechnen starten, das Textprogramm wieder laden und die Druckfunktion aufrufen. Insofern kann diese Methode natürlich gegenüber selbst rechnenden Textprogrammen wie z. B. Euroscript nur ein Provisorium darstellen – aber ein funktionierendes. Und bekanntlich hält sich nichts so gut und lange wie ein funktionierendes Provisorium! Fe.

Connection Machine, eine neue Computergeneration

Die US-Firma Thinking Machine Corporation hat in diesem Jahr die Öffentlichkeit mit einem neuen Konzept für einen Parallelcomputer überrascht: Connection Machine. An ihrer Entwicklung waren die bekanntesten Forscher auf dem Gebiete der KI beteiligt, darunter Marvin Minsky, Daniel Hillis und David Waltz.

Die Connection Machine besteht aus vier Mikrokontrollern, die jeweils je 16384 parallel geschaltete Prozessoren steuern. Die Prozessoren sind in der „hypercube" Architektur miteinander vernetzt. Zwischen den Prozessoren kann man virtuelle Verbindungen herstellen und dadurch den Parallelcomputer an das jeweilige Problem anpassen, zum Beispiel an Gitterstrukturen für die Mustererkennung, an semantische Netze für Expertensysteme usw.

Die Steuerung der Connection Machine erfolgt entweder durch eine Symbolics 3600 oder durch eine VAX. Der Host-Rechner speichert auch Anwenderprogramme, die in CmLISP, C* oder REL-2 zu schreiben sind. CmLISP ist ein Common-LISP, das mit Prozeduren für die Parallelverarbeitung erweitert wurde, C* ist die erweiterte C-Sprache, REL-2 ist ein spezieller Assembler. Durch dieses Softwarekonzept erhielten die Sprachen LISP und C ein noch größeres Gewicht als sie es schon vorher hatten.

Durch die Parallelverarbeitung erreicht die Connection Machine mehr als 1000 MIPS und damit die 2,5fache Geschwindigkeit einer Cray. So benötigte die Symbolics 3600 für die Erkennung eines Stereobildes mehrere Minuten, die 1/4-Connection Machine erledigte das gleiche Problem in der Zeit unter einer Sekunde. Die Möglichkeit, Tausende von Knoten eines mit Wissen gefüllten Netzwerkes parallel zu verarbeiten, eröffnet in der KI-Forschung völlig neue Perspektiven. Bisher scheiterte die Bildung großer Expertensysteme an zu langer Rechenzeit bei seriellen Inferenzoperationen. Einzelheiten über die Connection Machine sind im technischen Bericht des MIT enthalten (Hillis D., The Connection Machine, MIT Press, Cambridge, Mass., 1985). Es steht noch nicht fest, ob die USA die Connection Machine nach Europa exportieren werden. *fa.*