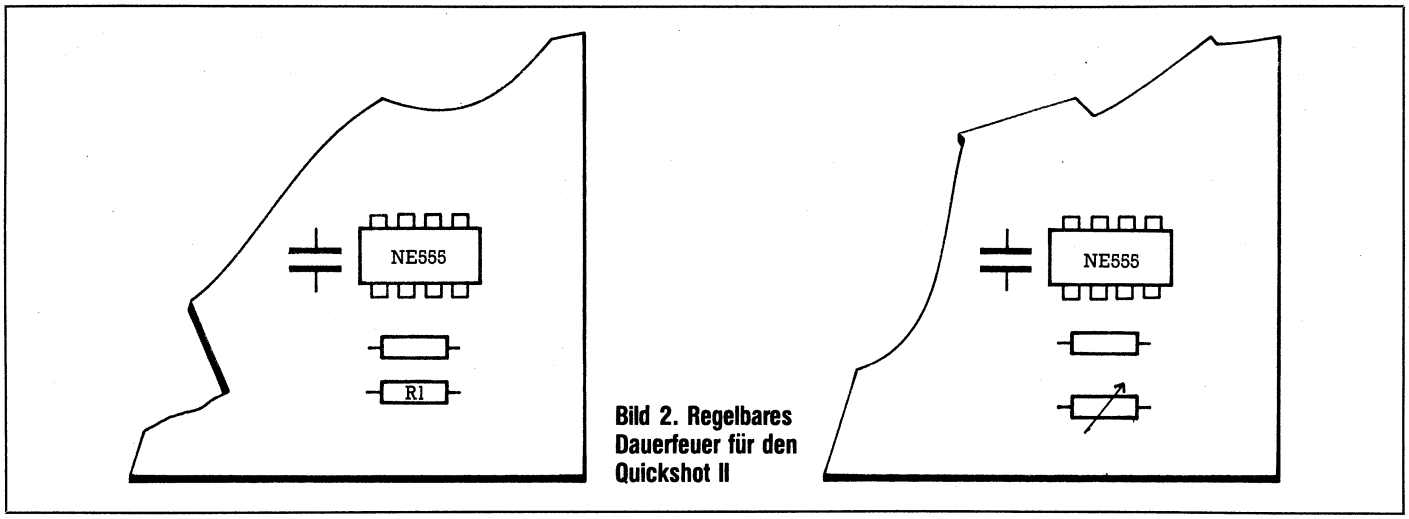


Bild 2. Regelbares Dauerfeuer für den Quickshot II

Nur so viel: es liegt daran, daß extrem viel im Speicher verschoben werden muß. Deshalb sollte man bei seinen Programmen wenn möglich folgendes beachten:
— Zur Einsparung von Speicherplatz lassen Sie alle REMs weg.
— Versuchen Sie so viel Programmtext wie möglich in eine Zeile zu packen. Man sieht es später ja nicht mehr.

Sollte Ihr Programm zu lang sein, meldet sich H.I.D.E. und sagt es Ihnen. Normalerweise sind Programme bis 32 KByte möglich. Beachten Sie auch, daß Sie das Programm jetzt nicht mehr editieren können und die Zeilen gelöscht werden, wenn Sie mit dem Cursor auf die Zeilennummer gehen und <RETURN> drücken! (Frank Hund/tr)

Regelbares Dauerfeuer

Ich habe einen Quickshot II-Joystick, der mit einfachem Dauerfeuer ausgestattet ist. Durch das Auswechseln eines Widerstandes gegen ein Poti (Bild 2) kann das Dauerfeuer geregelt werden. Hardwareschäden sind während der vier Wochen Dauertest nicht aufgetreten (K. Pichlmair/tr)

Zahlen eingeben mit dem Joystick

In Ausgabe 6/86 auf Seite 77 wurde bereits ein solches Programm veröffentlicht. Es läßt sich aber wesentlich kürzer und übersichtlicher schreiben (siehe Listing 4). Der Joystick wird in Port 1 gesteckt. Der Feuerknopf beendet die Eingabe. Danach wird die eingegebene Zahl berechnet und in der Variablen W gespeichert.

Die Eingabe kann mit Hilfe von PRINT-Befehlen beliebig positioniert werden. Der POKE 198,0 in der letzten Zeile löscht den Tastaturpuffer, der wegen des Joysticks in Port 1 wirre Zeichen enthält. (Andreas Kaiser/tr)

Tip zu PrologicDOS

Über »POKE 781,133: SYS 58551« läßt sich aus einem Basic-Programm heraus das Disketteninhaltsverzeichnis auf den Bildschirm bringen (entspricht der <F1>-Taste). Das Ganze läßt sich natürlich auch in Maschinensprache schreiben: »LDX #$85: JSR $E4B7« (tr)

Reise durch den C128 (Teil 5)

Erneut hat die Redaktion einen Bericht ihres Korrespondenten erhalten. Es ist ihm gelungen, eine Karte der Speicherlandschaft zu entwerfen, die zwar noch einige weiße Flecken aufweist, aber dennoch die Orientierung erleichtert.

Als C 128-Benutzer stolpert man meistens zuerst einmal über den BANK-Befehl. Was es damit auf sich hat, bekommt man zwar in dem Moment heraus (oder im schlimmeren Fall nicht), wenn man mit PEEK oder POKE arbeitet und auch bei Verwendung des eingebauten Monitors. Aber welche Speicher steuert man eigentlich durch die diversen BANK-Kennzahlen an und wo befinden sich die Speicher? Bild 1 gibt zunächst einen Überblick:

Sie sehen darin unten vier große (64 KByte) RAM-Bereiche, von denen die untere durch BANK 0, die darüberliegende durch BANK 1 gekennzeichnet ist. Zwei weitere 64-KByte-Bereiche sollen demnächst als Speichererweiterungen zu erwerben sein. Diese vier Bereiche werden durch BANK n angesteuert, wobei n die BANK-Nummer ist, also von 0 bis 3 reicht. Alle vier haben zwei Adressenbereiche gemeinsam: Eine »Common Area« (das heißt auf deutsch »gemeinsamer Bereich«) von Speicherstelle 0 bis 1023, also 1 KByte lang. Jedes Lesen oder Schreiben in diesen Teil findet automatisch in BANK 0 statt. Man braucht sich hier daher um keine BANK-Umschaltung zu kümmern. Ein schmaler Bereich zwischen $FF00 (dezimal 65280) und $FF04 (dezimal 65284) gehört zur sogenannten MMU (das heißt »Memory Management Unit« = »Speicherorganisations-Einheit«), um die wir uns noch kümmern werden. Dieser kleine Bereich ist deshalb allen BANKs gemeinsam, weil damit die Umschaltung von einer Speicherkonfiguration zur anderen vollzogen wird, er ist gewissermaßen der Schalter dazu. Hätten wir aber eine Speicherzusammenstellung vorliegen, in der dieser Schalter nicht enthalten wäre, dann könnten wir nicht mehr zurück in andere Konfigurationen.

Zurück zur BANK 0: Direkt oberhalb der Common Area befinden sich der normale Textbildschirm und die Sprite-Zeiger, die vom Basic 7.0 benutzt werden. Zwischen den Adressen $0800 bis $1300 liegt wieder Systemspeicher vor: Eine Wanderung durch die Memory-Map wäre hier schon ganz schön lang! Ab $1300 bis $1C00 haben wir freien Speicherplatz zur Verfügung, den wir beispielsweise für Maschinenprogramme nutzen können. Von da an wird's mehrdeutig.

Falls wir nur im Textmodus arbeiten, dürfen wir den gesamten restlichen Speicher (bis $FF00) für Basic-Programme benutzen. Sollten aber Grafik-Befehle eine Rolle spielen, dann »baut« das Betriebssystem die BANK 0 noch etwas um: Von $1C00 bis $2000 wird ein Farbspeicher eingerichtet. Zwischen $2000 und $4000 befindet sich die Bit-Map, deren Inhalt als Grafik unseren Bildschirm ziert. Erst ab $4000 steht der Basic-Programmtext. Soweit die BANK 0. Wo sind die Variablen geblieben?

Die gesamte BANK 1 ist der Tummelplatz für alle Variablen. Zwischen $0400 (dort endet ja die Common Area) und $FF00 haben sie ihren Platz. Die Tatsache, daß sie in einer getrennten BANK liegen, hat eine Reihe von Vorteilen, aber auch Nachteile. Veränderungen des Basic-Textes wirken sich nun nicht mehr — wie im C 64-Modus — löschend auf Variable aus. Andererseits wäre es manchmal wünschenswert, auch in den freien Speicherraum oberhalb des Programmes in BANK 0 Variable einzulagern, denn bei ausgiebiger Nutzung von String-Arrays hat man doch schnell die BANK 1 gefüllt. Ohne Tricks — beispielsweise mit dem Transfer-Modul aus unserem letzten Bericht — ist das aber nicht möglich.

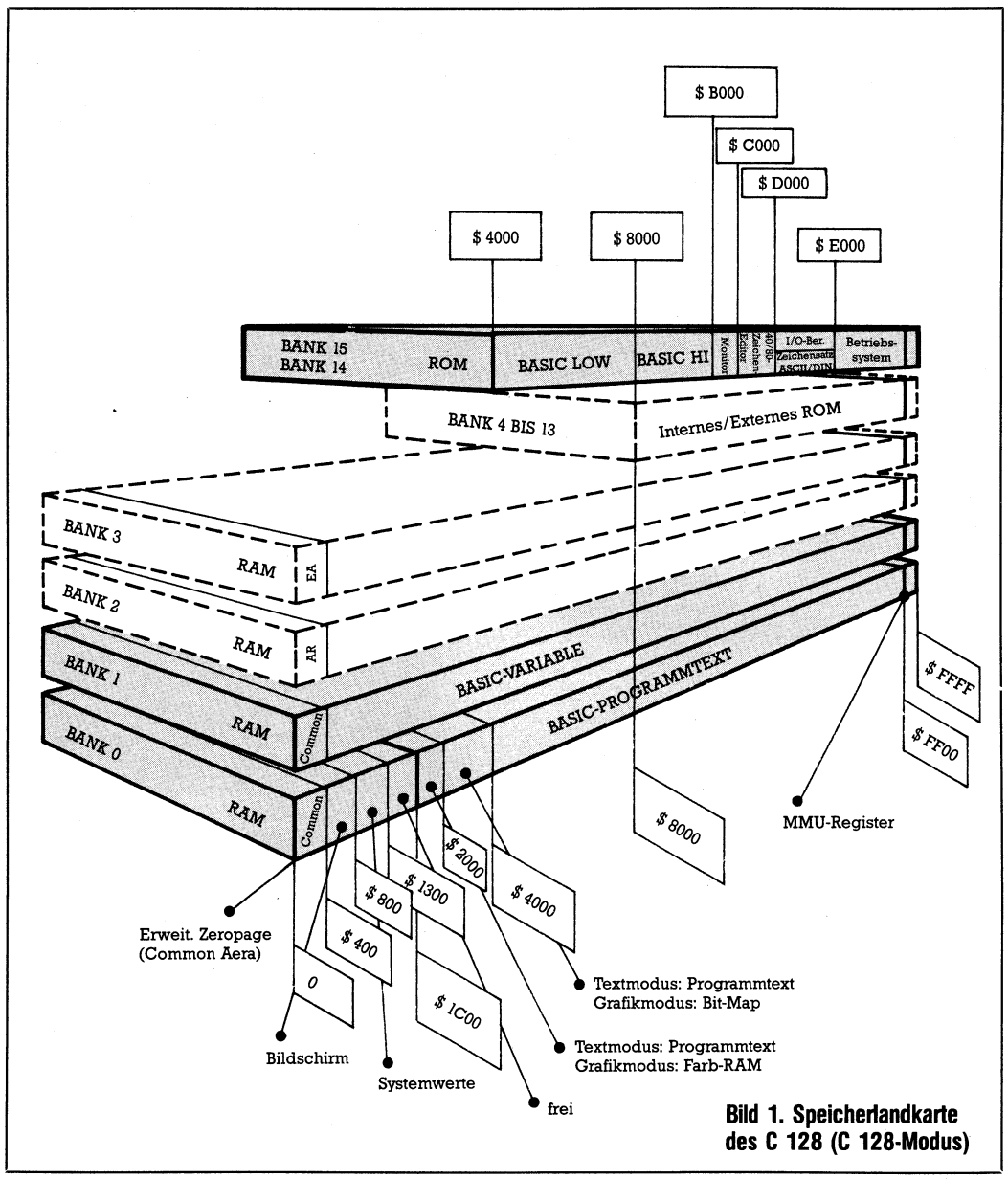


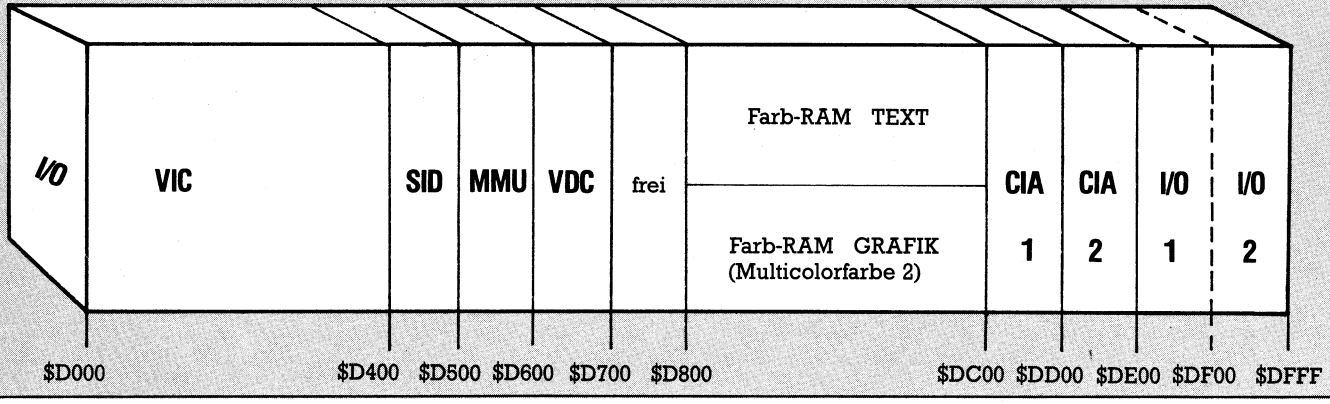
Bild 1. Speicherlandkarte des C 128 (C 128-Modus)

Zu den BANKs 2 und 3 kann noch nicht viel gesagt werden, denn es gibt noch keine Möglichkeiten, damit herumzuprobieren. Die im Basic 7.0 enthaltenen Verschiebebefehle STASH, SWAP und FETCH haben mit diesen BANKs zu tun. Durch die Basic-Befehle BANK 4 bis BANK 13 hat man Einfluß auf die sogenannten internen und externen Funktions-ROMs. Je nach BANK-Nummer stellt man sich verschiedene Kombinationen von RAM und ROM zusammen. Auch dazu soll hier nichts weiter gesagt werden, denn diese ROMs wurden noch nirgends gesehen.

Danach aber wird es noch einmal interessant: BANK 14 und BANK 15 blenden die Firmware in das RAM von BANK 0 ein. Der einzige Unterschied zwischen 14 und 15 liegt darin, daß man über BANK 14 Zugriff auf das Zeichensatz-ROM hat, wohingegen über BANK 15 der Ein- und Ausgabeteil des C 128 programmiert werden kann. In Bild 1 finden Sie den Aufbau des ganzen ROM-Paketes:

Es beginnt an der Adresse $4000 mit dem Basic-Interpreter. Dieser Teil wird als »Basic Low« bezeichnet. Sie können sich

Bild 2. Der Ein-/Ausgabebereich im C 128-Modus



Bit:	7	6	5	4	3	2	1	0
Bedeutung:	?	DIN/ASCII Umschaltung 1 = ASCII 0 = DIN		Zusammenhang 11 = keine Taste 00 = Taste	mit ?	Text-/Grafik-Modus 01 = Text 10 = Grafik		Steuert D800—DC00 1 = Farb-RAM für Text 0 = Farb-RAM für Grafik (Multicolorfarbe 2)

Bild 3. Der Prozessorport (Speicherstelle 01) im C 128-Modus

so sicher auch gleich denken, daß es ein »Basic High« gibt. Das befindet sich direkt dahinter, nämlich ab Adresse $8000. Hier beginnt das Programm, das unseren Monitor am Leben erhält, bei $C000 der 40- und 80-Zeichen-Editor. Von $D000 bis $E000 existieren — wie schon vorhin erwähnt — zwei Möglichkeiten: Zeichen-ROM oder I/O-Bereich. Ab $E000 beginnt das obere ROM, das man etwas salopp auch als Betriebssystem bezeichnen könnte.

Der I/O-Bereich

Nach diesem Überblick sehen wir uns nun auch noch den Aufbau des Ein- und Ausgabebereiches an. Der ist dem im C 64-Modus sehr ähnlich, was ja auch kein Wunder ist, denn er wird größtenteils auch in diesem Modus benutzt. Bild 2 gibt uns eine Zusammenfassung:

Zunächst finden wir den in Ausgabe 8/86 besprochenen VIC-Chip (von $D000 bis $D400). Daran schließt sich der Baustein an, der Musikliebhabern die Herzen höher schlagen läßt: der SID. Das bedeutet »Sound Interface Device«. Offenbar war zwischen $D400 und $D800 noch eine Menge Platz beim C 64, denn im C 128 finden sich nun zwei Bausteine, die allerhand leisten: Von $D500 bis $D600 ist die MMU (Memory Management Unit) und von $D600 bis $D700 der VDC-Chip enthalten. Beide sollen uns noch näher bekannt werden. Sogar ein freier Platz ist noch vorhanden. Er reicht von $D700 bis $D800. Den Rest kennen C 64-Freaks noch: Es schließt sich das Farb-RAM an, das von $D800 bis $DC00 reicht. Aber Vorsicht! Ganz so einfach macht es uns der C 128 nicht! Wenn Sie im Normalzustand mittels des Monitors mal dort hineinsehen, dann finden Sie zwar den Farb-Code für den Textbildschirm, aber löschen Sie doch mal das Bit 0 der Speicherstelle $01. Sehen Sie nun nochmal ab $D800 nach: Nun finden Sie noch ein zweites Farb-RAM, nämlich das für den Code der Multicolorfarbe 2. Klammheimlich hat Commodore also hier noch weiteren Speicherbereich parat, der sonst nicht in Erscheinung tritt. Mit den Inhalten der Speicherstelle $01, die im C 64-Modus eine große Rolle spielt (zur Organisation der ROM/RAM-Zusammenstellungen), werden wir uns ebenfalls gleich noch befassen.

Zwei CIAs — das kommt von »Complex Interface Adapter« —, die den Verkehr der Peripherie mit der Zentraleinheit steuern, befinden sich zwischen $DC00 und $DD00 (das ist der CIA1, auch IRQ-CIA genannt), sowie $DD00 und $DE00 (hier haben wir den CIA2, der auch als NMI-CIA bezeichnet wird). Über deren Aufbau und Register kann man bislang nur vermuten, daß kein gravierender Unterschied zum C 64 vorhanden sein wird, es gibt aber noch zu wenig Information, die genauere Aussagen ermöglicht. Zwischen $DE00 und $DF00, sowie zwischen $DF00 und dem Ende des I/O-Bereiches bei $E000 sind noch zwei Speicherabschnitte für zukünftige Erweiterungen freigehalten. Angeblich soll im letzteren optional ein DMA-Controller installiert werden.

Die Speicherstelle $01

Im C 64-Modus haben die beiden Speicherstellen $00 und $01 eine besondere Rolle als sogenannter Prozessor-Port. Man kann durch unterschiedliche Bit-Konstellationen ROM oder RAM an verschiedenen Plätzen ein- und ausblenden. Auch im C 128-Modus ist dies der Prozessor-Port. Wenn wir uns den Speicheraufbau ansehen, dann sollten wir auch versuchen, die Bedeutung der einzelnen Bits in diesem Modus herauszufinden. Das Ergebnis unserer Untersuchungen finden Sie zusammengefaßt in Bild 3:

Bit 0 ist, wie wir schon gesehen haben, verantwortlich dafür, welches Farb-RAM ab $D800 aktiv ist. Der normale Bitwert ist 1 und das Text-Farb-RAM eingeschaltet. Beim Wert 0 aber findet sich an der gleichen Stelle das Farb-RAM für die Multicolor-Farbe 2.

Die Bits 1 und 2 enthalten im Grafik-Modus (also nach GRAPHIC 1...4), die Belegung 10, ansonsten aber 01 in den beiden Textmodi.

Die Bedeutung der Bits 3 bis 5 konnte noch nicht ganz geklärt werden. Anscheinend dienen sie ähnlichen Zwecken wie im C 64-Modus, denn wenn eine Datasettentaste gedrückt wird, wandelt sich der Inhalt der beiden Bit 4 und 5 von »11« um in »00«. Im C 64-Modus dienen diese Bits folgendem Zweck:

Bit 3	serielle Daten werden zum Kassetten-Port gesandt. Normaler Inhalt = 0
Bit 4	1 = keine Recorder-Taste gedrückt 0 = Taste gedrückt
Bit 5	Datasettenmotor aus = 1 Motor ein = 0

Weil aber die Datasetten-Tasten immer auch den Motor einschalten, sind beide Bits (also 4 und 5) auf »11« oder »00«. Der

D50B	VR	Version-Register			
D50A	P1H	Page1-Pointer MSB			
D509	P1L	Page1-Pointer LSB			
D508	P04	Zeropage-Pointer MSB			
D507	P0L	Zeropage-Pointer LSB			
D506	RCR	RAM-Konfigurations-Reg.			
D505	MCR	Modus-Konfigurations-Reg.			
D504	PCRD	programmierbare	FF04	LCRD	Register zum Laden
D503	PCRC	Konfigurations-	FF03	LCRC	der programmierten
D502	PCRB	Register	FF02	LCRB	Konfigurations-
D501	PCRA	(A-D)	FF01	LCRA	Register (A—D)
D500	CR	Konfigurations-Register	FF00	CR	Konfigurations-Register
MMU in I/O-Bereich			MMU-Register im oberen ROM-Bereich		

Bild 4. Übersicht über die MMU-Register

Bit:	7	6	5	4	3	2	1	0
Bedeutung:	RAM-Bank-Auswahl: 00 = Bank 0 01 = Bank 1 10 = Bank 2 11 = Bank 3		Bereich C000—FFFF: 00 = ROM 01 = intern. ROM 10 = extern. ROM 11 = RAM		Bereich 8000—BFFF: 00 = Basic-ROM (high) 01 = intern. ROM 10 = extern. ROM 11 = RAM		Bereich 4000—7FFFF 0 = Basic-ROM (low) 1 = RAM	Bereich D000—DFFF 0 = I/O 1 = RAM/ ROM (H. Bit. 4 u. 5)

Bild 5. Der Aufbau des CR (Konfigurations-Register) $D500

beobachtete Normalzustand des Bit 3 im C 128-Modus war ebenfalls 0.

Interessant ist Bit 6. Im Einschaltzustand findet man dort eine »1«, wenn die ASCII/DIN-Taste nicht gedrückt ist. Hat man aber den DIN-Zeichensatz aktiviert, dann findet man eine »0«. An dieser Stelle muß noch ein Nachtrag zu den Speicherfragen gemacht werden: Blicken wir nämlich in BANK 14 (oder mit dem Monitor-Befehl »M ED000« in das Zeichensatz-RAM), dann sehen wir zwei verschiedene Inhalte bei ASCII- und bei DIN-Betrieb. Also gibt es hier ein klammheimliches weiteres Zeichen-ROM von 4 KByte Ausdehnung, das durch Bit 6 ein- oder ausgeblendet wird.

Die Bedeutung von Bit 7 konnte bislang noch nicht geklärt werden. Dort befindet sich eine »0«.

Nur die Bits 0 und 3 können durch direkten Eingriff (also durch POKE- oder Monitor-Befehle) geändert werden. Alle anderen halten ihren Wert, bis das Ereignis eintritt (also beispielsweise der Druck auf die ASCII/DIN-Taste), das ihren Inhalt neu festlegt.

Noch mehr ROM

Da wir immer noch bei Fragen der Speicherkonfiguration sind, soll noch ein ROM-Baustein von 4 KByte Länge erwähnt werden, der ebenfalls bei $D000 liegt. Wir werden ihn gleich erklären. Zuvor aber noch etwas Stoff zum Nachdenken: Die Erbauer des C 128 waren wirklich Meister im Verstecken von Speicherraum! Wie wir später beim Untersuchen des VDC-Chip feststellen werden, gibt's da ebenfalls nochmal 16 KByte Speicherraum. Insgesamt hat also unser C 128 in der Grundstufe folgenden Speicherplatz:

128 KByte in BANK 0 und 1
60 KByte in BANK 15
4 KByte Zeichen-ROM für ASCII-Zeichen
4 KByte Zeichen-ROM für DIN-Zeichen
1 KByte Farb-RAM für Multicolor
4 KByte Z80-ROM (das besprechen wir gerade)
16 KByte VDC-RAM
8 KByte Interpreter-ROM für C 64-Modus
8 KByte Kernel-ROM für C 64-Modus
233 KByte insgesamt!

Erstaunlich, nicht wahr: Unser C 128 ist ein C 233!

Nun sollen aber noch diese ominösen 4 KByte erklärt werden, die ich in der obigen Rechnung Z80-ROM genannt habe: Sicher wissen Sie, daß Sie mit dem C 128 keinen reinen 6502-Mikrocomputer gekauft haben (besser gesagt 8502. Diese Zentraleinheit ist aber der 6502 so nahe, daß wir keinen Unterschied feststellen können. Sogar ein CPU-Fehler, den die 6502 aufweist, ist dort wieder eingebaut worden!). Unser Computer hat außerdem noch einen Z80B-Mikroprozessor, der einen anderen Befehlssatz besitzt. Wenn aber der Computer eingeschaltet wird, versucht er zunächst einmal, CP/M zu booten. Dieses Boot-Programm läuft mittels des Z80 und befindet sich in dem erwähnten 4-KByte-ROM. Erst wenn es nichts zu booten gab — jedenfalls kein CP/M — wird der 8502 aktiv. Von diesem Augenblick an verschwindet dieses 4-KByte-ROM wieder aus der Speicherlandschaft.

Der Memory-Manager

Das, was die Speicherstelle $01 verwaltet, ist nur ein sehr kleiner Teil des Gedächtnisses unseres Computers. Den Gesamtüberblick hat die schon erwähnte MMU. Mittels 17 Registern hat sie — und damit auch wir, wenn wir die Bedeutung der Register kennen — alle Speichervarianten voll im Griff. Und das ist allerhand, wie Sie gleich noch sehen werden. In Bild 4 sind alle Register mit ihren Namen und Adressen vorgestellt:

Sehen wir sie uns nun im einzelnen an. Da wäre erst einmal das wohl wichtigste, das CR. CR kommt von »Configuration Register« und dieses Register steuert die aktuelle RAM-/ROM- und I/O-Zusammenstellung.

Es befindet sich sowohl an der Speicherstelle $D500 als auch — als Doppel — bei $FF00. Falls der I/O-Bereich einmal ausgeblendet ist, haben wir noch den Schalter bei $FF00. Bild 5 zeigt Ihnen den Aufbau und die Bedeutung der einzelnen Bits:

Bit 0 steuert, was sich im Bereich $D000 bis $DFFF anfindet.
1: RAM/ROM-Konfiguration gemäß den Inhalten von Bit 4 und 5
0: I/O-Bereich ist eingeschaltet.

Bit 1 richtet uns den Speicherbereich von $4000 bis $7FFF ein:
1: Hier befindet sich RAM
0: Das Basic-Low-ROM ist eingeblendet.

Bit 2 u. 3 kümmern sich um den Inhalt des Speicherraums $8000 bis $BFFF.
00: Das Basic-high-ROM liegt an der Benutzeroberfläche
01: internes ROM
10: externes ROM
beide sind für uns momentan noch ohne Interesse.
11: RAM ist eingeschaltet.

Bit 4 u. 5 Diese beiden Bit steuern den Inhalt der Speicher $C000 bis $FFFF.
00: ROM ist eingeschaltet
01: es liegt wieder internes
10: oder externes ROM vor. Beides sind ebenso wie bei den Bit 2 und 3 Erweiterungen, die noch nicht aktuell sind.
11: RAM ist eingeschaltet.
Solange mit diesen beiden Bit ROM eingeschaltet ist, liegt immer eine Lücke im Bereich $D000 bis $DFFF. In dieser Lücke befindet sich — je nach Inhalt von Bit 1 — entweder das ganze I/O-Sortiment oder aber Zeichensatz-ROM.

Bit 6 u. 7 Jedesmal, wenn bei den anderen Bit eine Kombination zur Einblendung von RAM geführt hat, entscheidet nun diese letzte Paarung, welcher RAM-Bereich gemeint ist.
00: RAM aus BANK 0
01: RAM aus BANK 1
10: (RAM aus BANK 2 und
11: RAM aus BANK 3). Die beiden letzteren sind ohne Speicherergänzung noch nicht bedeutsam. Statt dessen erfolgt die Einblendung derart, daß nur Bit 6 berücksichtigt wird.

Der Einschaltwert dieses CR-Registers ist 0. Das heißt, daß alle System-ROMs aktiv sind, ebenso der I/O-Bereich. Der verbleibende Speicher gehört zu BANK 0.

Damit endet wieder einmal die Erkundungsreise unseres eifrigen Forschers. In der nächsten Ausgabe folgt dann Teil 6, der diesen Bericht abschließt. (Heimo Ponnath/dm)