

Reise durch den C 128 (Teil 6)

VDC und MMU — zwei interessante und wichtige Bausteine im C 128. Wie kann der Bildaufbau mit den VDC-Registern beeinflusst werden?

Unser Korrespondent wurde leider im letzten Teil ziemlich abrupt unterbrochen. Deshalb folgt hier zunächst die nahtlose Fortsetzung des 5. Teils aus Ausgabe 10/86.

Die Speicherstellen \$D501 bis \$D504 nennen sich PCRA bis PCRD. PCR heißt soviel wie programmierte Konfigurationsregister. Sie hängen sehr eng mit den Speicherstellen \$FF01 bis \$FF04 zusammen, die LCRA bis LCRD heißen. LCR kommt von »Lade das Konfigurations-Register«. Der Bit-Aufbau dieser acht Register ist identisch mit dem des CR, das wir in der letzten Folge besprochen haben. Was haben sie für eine Wirkung und wie benutzt man sie? Nehmen wir einmal an, daß Sie im Rahmen eines Programmes vier verschiedene Speicherkonfigurationen brauchen:

Konfiguration CR-Inhalt dazu

- 1) Alles RAM (BANK 0) \$3F (=00111111)
- 2) Alles RAM (BANK 1) \$7F (=01111111)
- 3) Alles ROM + BANK 0 + Zeichen-ROM \$01 (=00000001)
- 4) Alles ROM + BANK 1 + Zeichen-ROM \$41 (=01000001)

Nun könnten Sie natürlich jedesmal, wenn es soweit ist, den jeweiligen Code nach \$FF00 schreiben. Das wäre aber überflüssige Arbeit, denn es gibt die andere Möglichkeit: Schreiben Sie zu Beginn Ihres Programms all diese verschiedenen Speichercodes in die PCR-Register, also beispielsweise den, der zur ersten Konfiguration gehört in PCRA, den der zur zweiten gehört in PCRB und so weiter. Wenn nun eine bestimmte Speichergruppierung erforderlich ist, schreiben Sie irgendeinen Wert in das dazugehörige LCR. Das ist nämlich der Trick: Sobald man beispielsweise ins LCRA irgend etwas schreibt, dann wird automatisch der Inhalt des PCRA ins CR geschrieben. Das bedeutet, daß die neue Speicherkonfiguration aktiviert ist. Versucht man dagegen die LCRs zu lesen, dann erhält man statt des LCR-Inhaltes den Inhalt des dazu korrespondierenden PCR. Die oben verwendeten Beispiele stammen übrigens aus unserem Computer: Sehen Sie sich mal nach dem Einschalten mit dem Monitor die LCRs an.

Die LCRs sind bei jeder Speicherkonfiguration ebenso erreichbar wie das CR bei \$FF00.

Bisher hatten alle Register einen Verwandten im oberen Speicher-Bereich. Das ändert sich nun: Die folgenden existieren nur noch im unteren MMU-Teil. Da kommt als nächstes das MCR, also das Modus-Konfigurations-Register ins Spiel. In Bild 1 sehen Sie, wie es aufgebaut ist:

Bit 0 bietet eine Möglichkeit, den Computer auf den Z80-Betrieb umzustellen. Man muß dazu nur dieses Bit lö-

Bits:	7	6	5	4	3	2	1	0
Bedeutung:	40/80-Zeichen-TASTE 0 = gedrückt 1 = nicht gedrückt	Betriebsart: 0 = C 128 1 = C 64	EXROM 1	GAME 1	FSDIR Ausgabe f. schnellen seriellen Daten-Puffer	1	1	Aktive CPU 0 = Z80 1 = 8502

Bild 1. Der Aufbau des MCR (Modus-Konfigurations-Register) \$D505

sch. Allerdings wird der Computer dann abstürzen, weil er keinen gültigen Z80-Code vorfindet. Wenn man aber einen Weg dazu findet, diesen griffbereit zu halten, dann könnte man den C 128 als Z80-Computer arbeiten lassen.

Bit 1 und 2 werden noch nicht genutzt.

Bit 3 hat mit dem schnellen seriellen Daten-Transport zu tun.

Bit 4 und 5 zeigen an, ob die beiden Leitungen GAME und

EXROM belegt sind, sich also ein Steckmodul im Expansion-Port befindet. Ist das der Fall, dann wird der C 64-Modus aktiviert. C 128-Module benutzen diese Leitungen nicht. Beide Leitungen können aber auch als Ausgang (statt wie eben auf Eingang) betrieben werden. In diesem Fall haben sie mit der Verwaltung der Farb-RAM-BANK zu tun.

Bit 6 zeigt an, welche Betriebsart aktiv ist:

0 = C 128-Modus 1 = C 64-Modus

Bit 7 schließlich gibt den Zustand der <40/80-Display>-Taste an:

0 = Taste gedrückt 1 = Taste frei

Im Abschnitt über den VIC war es schon angedeutet worden: Es gibt eine Möglichkeit in den MMU-Registern, den Zugriff des VIC auf BANK 1 zu richten. Dazu dient das RCR (RAM-Konfigurations-Register). Einen Überblick über die Möglichkeiten zeigt Bild 2:

Bits:	7	6	5	4	3	2	1	0
Bedeutung:	VIC-BANKZEIGER X0 = BANK 0 X1 = BANK 1 X: Bit 7 noch ohne Bedeutung		unbenutzt (geplant als Blockzeiger)		Ort der Common Area: 00 = keine Common A. 01 = unterer RAM-Bereich 10 = oberer RAM-Bereich 11 = ein Teil im unteren, einer im oberen RAM-Bereich		Größe der Common Area: 00 = 1 KByte 01 = 4 KByte 10 = 8 KByte 11 = 16 KByte	

Bild 2. Der Aufbau des RCR (RAM-Konfigurations-Register) \$D506

Sie sehen, daß außer dem VIC-Zugriff auch die Organisation der Common Areas hier geschieht.

Bit 0 und 1

Hier wird die Größe der Common Area definiert. Sie muß also keineswegs — wie im Einschaltzustand — immer 1 KByte betragen.

00 = 1 KByte 10 = 8 KByte
01 = 4 KByte 11 = 16 KByte

Bit 2 und 3

Auch die Lage dieser Bereiche kann verändert werden:

00 = Keine Common Area!

01 = Der gemeinsame Speicher liegt im unteren RAM-Bereich, also am Speicheranfang. Das ist der Normalzustand.

10 = In dieser Konfiguration liegt die definierte Common Area am Speicherende.

11 = Der gemeinsame Speicherbereich wird aufgeteilt: ein Teil befindet sich am Anfang, der andere am Ende des Speichers.

Bit 4 und 5

Diese beiden Bits sind noch ungenutzt. Es wird daran gedacht, sie für Blockzugriffe zu verwenden. Jeder Block hätte in dem Fall 256 KByte an Speicherinhalt. Dann würden durch die vier verschiedenen Bitkombinationen insgesamt 1 MByte RAM adressierbar werden.

Bit 6 und 7

Da haben wir die VIC-Bankzeiger. In der vorliegenden Version des C 128 spielt lediglich das Bit 6 eine Rolle:

X0 = Der VIC greift auf die BANK 0 zu

X1 = BANK 1 steht dem VIC offen! Alles, was Sie im Abschnitt über den VIC erfahren, läßt sich nun auf diese BANK übertragen. Die Anzahl der möglichen Bildschirmspeicher, Bit-Maps und Zeichen-RAMs kann verdoppelt werden.

Die Variationsbreite unseres Computers ist durch die bisher kennengelernten MMU-Register schon sehr breit. Jetzt kommt es aber noch besser. Die Register \$D507 bis \$D50A können nämlich die Zeropage und den Prozessorstapel verlegen! Zwar ist das vor allem für System-Programmierer interessant, aber häufig werden gerade Grafikprogramme in Assembler geschrieben. Außerdem kommt es meist besonders bei der Grafikprogrammierung auf Geschwindigkeit an.

Wie funktioniert das? Natürlich kann an dieser Stelle nur

das Prinzip erklärt werden, mehr darüber sollte in der Assembler-Literatur erscheinen. Übrigens haben allem Anschein nach die Hersteller der Firmware des C 128 diese Möglichkeit nicht genutzt! Nun zum Prinzip. P0L (\$D507) und P0H (\$D508) sind die Page-Zeiger für eine neue Zeropage. In P0L dienen die Bits 0 bis 7 zum Einschreiben der Page in einer BANK, die nun Zeropage werden soll. Die Nummer der BANK gelangt in P0H. Dafür sind dort die Bits 0 bis 3 vorgesehen. Die Bits 4 bis 7 werden nicht benutzt und sind immer 1. Im Einschaltzustand befindet sich im P0L der Wert 0 (Page 0 ist dann die Zeropage), im P0H findet man \$F0, wobei das F von den vier gesetzten oberen Bits stammt. Ansonsten ist also die BANK 0 dafür definiert. Sie merken schon, daß man auch bei dieser Umstellung zu neuen Zeropages die Common Area im Auge behalten sollte.

Ebenso, wie wir das für die Zeropage nun betrachtet haben, gilt es für den Stapel. Hier heißen die beiden Register P1L (\$D509) und P1H (\$D50A). Der Einschaltzustand bewirkt die Inhalte: P1L = 01 (dort befindet sich ja normalerweise der Prozessorstapel) und P1H = F0, ebenso wie beim P0H.

Eine Änderung dieser Zeiger muß zuerst die Register P0H (beziehungsweise P1H) betreffen. Der eingeschriebene Wert wird so lange zwischengespeichert, bis auch P0L (beziehungsweise P1L) belegt wird.

Ein letztes MMU-Register, welches aber nur gelesen werden kann, ist das VR. VR steht für Versions-Register. Die unteren vier Byte machen eine Aussage über die vorliegende MMU-Version. In meinem C 128 steht dort 0000. Die Bits 4 bis 7 geben die Anzahl der verfügbaren 64-KByte-Banks an. Mein Computer hat dort 0010 stehen, also zwei Banks.

Einführung in den VDC-Chip

Der VDC ist ein Chip, der die Verwaltung des 80-Zeichen-Bildschirmes kontrolliert. Es wäre unfair, an dieser Stelle nun alle Register abzudrucken. Sie finden diese nämlich recht ausführlich kommentiert in Ihrem Handbuch (Anhang E, Seite 6 und folgende). Deshalb sollen nur die bedeutsamen herausgenommen und näher erläutert werden.

Weder die Register noch ein 16-KByte-Speicherbereich, der beim GRAPHIC-Befehl schon vorgestellt wurde und den VDC mit den notwendigen Bildschirminformationen versorgt, sind in irgendeiner BANK unter den normalen Adressen (beispielsweise durch PEEK oder POKE oder mittels Monitor) zu finden. Sie führen ein vornehmes Einzeldasein fernab davon. Nur zwei Speicherstellen im VDC-Bereich zwischen \$D600 und \$D700 sind die Vermittler:

\$D600 dient zur Angabe des gewünschten Registers
\$D601 übergibt den in das Register zu schreibenden oder den aus diesem Register gelesenen Wert.

Ein POKE-Befehl sähe also so aus:

BANK15:POKE DEC("D600"),Reg.Nummer:POKE DEC("D601"),Wert

Und ein PEEK wäre zu realisieren durch:

BANK15:POKE DEC("D600"),Reg.Nummer:PRINT PEEK(DEC("D601"))

Und in Maschinensprache wird es noch etwas komplexer, denn die VDC-Register sollten nur dann beschrieben werden, wenn dieser Chip nicht gerade etwas anderes zu tun hat. Zwar ist er mit seiner Arbeit so schnell, daß man in Basic kaum Störungen zu erwarten hat, in Assembler aber müssen wir daher auf ein Status-Bit achten, welches als Bit 7 der Speicherstelle \$D600 existiert. Erst wenn hier eine 1 steht, sollte \$D601 angesprochen werden.

In den 16 KByte des VDC-Speichers liegen drei für uns interessante Bereiche, auf die wir Einfluß nehmen können:

- 1) \$0000 bis \$07CF Bildschirm-Speicher
- 2) \$0800 bis \$0FCF Attribut-RAM.
- 3) \$2000 bis \$3FFF Zeichenmuster

Zum ersten Bereich fragen wir uns, wie wir dort hinein schreiben können. Außerdem ist auch hier ein Verschieben des Bildschirms eine nette Sache. Das gleiche gilt für das Attribut-RAM. Der Zeichenmusterbereich ist interessant,

weil wir durch Eingriffe eigene Zeichen definieren können.

Schließlich ist es noch unser Anliegen, Grafik auf dem 80-Zeichen-Bildschirm zu realisieren. Auch dazu gibt es Wege. Aber gehen wir der Reihe nach vor.

Die Register 18 und 19 des VDC bestimmen die Adresse aus dem 16-KByte-Speicher, in die ein Wert geschrieben wird. Dabei soll dann 19 das LSB und 18 das MSB der gewünschten Anschrift erhalten. Nehmen wir mal als Beispiel das Bildschirm-RAM von \$0000 bis \$07CF (das ist dezimal 0 bis 1999) und daraus die Adresse \$0400 (also dezimal 1024). In einem Programm können wir uns die beiden Bestandteile berechnen lassen. V sei die Adresse, LO das LSB und HI das MSB davon:

380 HI = INT(V/256):LO = V - HI*256

Wir erhalten auf diese Weise für HI den Wert 4 und für LO den Wert 0 (in Hexadezimalzahlen ist das schon auf den ersten Blick klar: 04 00). Unser Programm muß weiter lauten:

390 POKE DEC("D600"),18:POKE DEC("D601"),HI

400 POKE DEC("D600"),19:POKE DEC("D601"),LO

Nun muß natürlich noch der Wert, der in den so ausgewählten Speicherplatz hinein soll, übergeben werden. Zu diesem Zweck dient das VDC-Register 31. Sei der Wert allgemein ausgedrückt durch die Variable W, dann lautet das weitere Programm:

410 POKE DEC("D600"),31:POKE DEC("D601"),W

Es gilt nun noch zweierlei zu beachten. Es muß angegeben werden, wieviele Speicherstellen diesen Wert von der besagten Adresse an erhalten sollen. Das schreibt man als Va-

```

100 REM ***** PROGRAMM SCREENTEST *****
110 SCNCLR 5: WINDOW 0,12,79,24
120 REM +++ BILDSCHIRM-RAM VERAENDERN +++
130 : INPUT "ADRESSE BILDSCHIRM-RAM(0 BIS 19
99)";V
140 : INPUT "GEWUENSCHTER WERT(POKE-CODE)";W
150 : INPUT "WIE OFT";N
160 : GOSUB 380: SCNCLR 5
170 REM +++ ATTRIBUT-RAM VERAENDERN +++
180 : PRINT "MOECHTEN SIE EIN ATTRIBUT VERA
ENDERN? (J/N)"
190 : GET KEY A$: IF A$="N" THEN 250
200 : INPUT "ADRESSE ATTRIBUT-RAM(2048-4047)
";V
210 : INPUT "GEWUENSCHTER WERT(SIEHE BESCHRE
IBUNG)";W
220 : INPUT "WIE OFT";N
230 : GOSUB 380: SCNCLR 5
240 REM +++ ZEICHENMUSTER-RAM VERAENDERN +++
250 : PRINT "SOLLEN ZEICHENMUSTER VERAENDERT
WERDEN? (J/N)"
260 : GET KEY A$: IF A$="N" THEN 320
270 : INPUT "POKE-CODE DES ZU AENDERNDEN ZEI
CHENS";P: B=B192+16*P
280 : INPUT "DAS WIEVIELTE BYTE (0 BIS 7)";B
Y: V=B+BY
290 : INPUT "NEUER WERT";W: N=1
300 : GOSUB 380: SCNCLR 5
310 REM +++ WEITERE VERAENDERUNGEN +++
320 PRINT "WEITERE VERSUCHE?"
330 PRINT "(1=BILDSCHIRM / 2=ATTRIBUT / 3=ZE
ICHEN / 4=ENDE)"
340 GET KEY A$: IF VAL(A$)<1 OR VAL(A$)>4 TH
EN 320
350 SCNCLR 5: ON VAL(A$) GOTO 130,200,270,36
0
360 WINDOW 0,0,79,24,1: END
370 REM ----- UNTERPROGRAMM VDC-RAM BESCHREI
BEN -----
380 HI=INT(V/256): LO=V-256*HI
390 POKE DEC("D600"),18: POKE DEC("D601"),HI
400 POKE DEC("D600"),19: POKE DEC("D601"),LO
410 POKE DEC("D600"),31: POKE DEC("D601"),W
420 WAIT DEC("D600"),32
430 POKE DEC("D600"),30: POKE DEC("D601"),N
440 RETURN
    
```

Listing 1. »SCREENTEST« — Erlaubt allerlei Eingriffe in das VDC-RAM

riable N in das Register 30. Erst durch diesen Schreibvorgang wird ein Zeichenmuster oder der angegebene Wert tatsächlich in den Speicher eingetragen. Zuvor aber gibt es noch eine Kleinigkeit zu beachten: In \$D600 gibt es nämlich noch ein Bit, das anzeigt, ob ein Rasterstrahl-Rücklauf stattfindet. Darauf muß gewartet werden:

```
420 WAIT DEC("D600"),32
430 POKE DEC("D600"),30:POKE DEC("D601"),N
```

Das war's eigentlich auch schon! Damit haben wir die gesamten 16 KByte im Griff. Fassen Sie diese Programmzeilen als Unterprogramm auf und fügen Sie einen entsprechenden Rahmen dazu, dann erhalten Sie etwas ähnliches wie unser kleines Testprogramm SCREENTEST (Listing 1):

Hier eine kurze Beschreibung: Der erste Abschnitt dient dem Einschreiben von Zeichen in den Bildschirmspeicher. Zuerst wird nach der gewünschten Adresse gefragt, dann nach dem POKE-Code des Zeichens und schließlich nach deren gewünschter Anzahl. Damit die ganzen Dialoge nicht mit der Bildschirmveränderung in Konflikt kommen, ist der Textbetrieb auf ein Fenster beschränkt worden, das die untere Bildschirmhälfte einnimmt. Es ist daher sinnvoll, Adressen zu wählen, die außerhalb des Fensters liegen.

Der nächste Programmblock bedient das Attribut-RAM. Auch hier werden Sie zunächst nach der Adresse gefragt. Der Zusammenhang der Bildschirmspeicher- mit der Attribut-RAM-Adresse ist ähnlich wie bei der 40-Zeichen-Darstellung: Zu jeder Bildschirmzelle gibt es eine korrespondierende Attribut-RAM-Zelle. Bild 3 stellt die Verhältnisse dar:

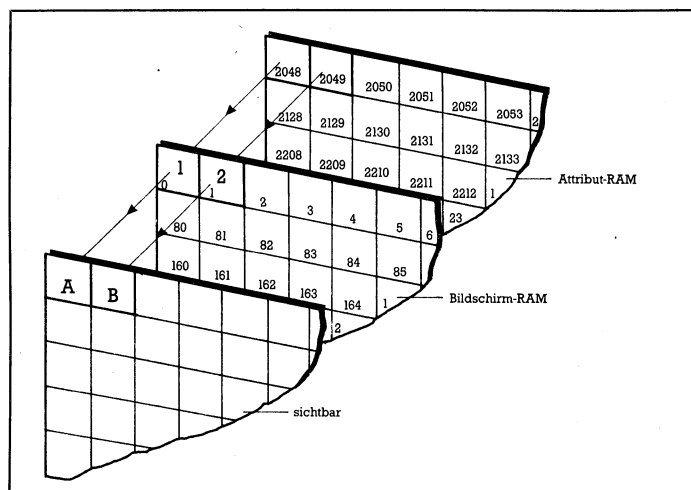


Bild 3. Zusammenspiel von Bildschirmspeicher und Attribut-RAM

Des weiteren wird der einzuschreibende Wert und die Anzahl der insgesamt davon betroffenen Speicherzellen erfragt – ebenso, wie es vorhin im ersten Teil geschah.

Der dritte Teil unseres Programms erlaubt ein Einschreiben in den Speicherbereich, aus dem der VDC seine Zeichen abrufen. Man gibt dazu den POKE-Code des zu ändernden Zeichens ein und welches der acht Byte des Zeichenmusters man meint. Dann fragt das Programm, was statt dessen dort hineingeschrieben werden soll. Falls Sie nun das betreffende Zeichen auf dem Bildschirm haben, erkennen Sie sofort das neue Gewand. Danach haben Sie die freie Wahl bis Sie <4> eingeben: Dies beendet das Programm. Sollten Sie wieder normale Zeichen benutzen wollen, brauchen Sie nur die <ASCII/DIN>-Taste betätigen. Dadurch wird der jeweils andere Zeichensatz ins Muster-RAM eingeladen und unsere Mühe ist umsonst gewesen.

Den ersten Teil unserer Fragen haben wir nun beantwortet. Außerdem wissen wir nun genau, wie wir auf den 16-KByte-Speicher des VDC zugreifen können.

Sehen wir nun mal nach, inwieweit der Bildschirmspeicher verschiebbar ist. Es existieren da nämlich zwei Register im VDC-Chip, die damit zu tun haben:

Register 13 enthält das LSB,

Register 12 das MSB der neuen Bildschirmstartadresse.

Das MSB muß außerdem noch in eine Speicherstelle des normalen Speichers geschrieben werden, nämlich in 2606. Die Bestandteile (MSB und LSB) können wieder mittels der oben genannten Formel berechnet werden. Sie wissen ja nun auch, wie man diese Werte an die beiden Register übergibt. Also steht Ihnen nun nichts mehr im Weg, den Bildschirmspeicher innerhalb der 16 KByte zu verschieben. Wohin kann geschoben werden? Gehen wir die 16 KByte einmal durch, wobei wir noch daran denken sollten, daß zum einen der Bildschirm zwar nur 2000 Speicherplätze benötigt, aber unser Computer meistens 2048 Plätze behandelt.

Oberhalb des normalen Bildschirm-RAM liegt der Platz für das Attribut-RAM. Es gibt eine Möglichkeit (die wir gleich noch kennenlernen), die Farbe nicht aus diesem RAM zu holen, sondern durch ein VDC-Register bestimmen zu lassen. Zwar haben dann der Vorder- und der Hintergrund auf dem gesamten Bildschirm jeweils den gleichen Farbcode, aber das Attribut-RAM steht uns für einen weiteren Bildschirm zur Verfügung.

Ab \$1000 ist bisher freier Platz, in den zwei komplette Bildschirme passen. Außerdem kann mit dem Zeichenmuster-RAM manipuliert werden. Es befinden sich beide Zeichensätze (Groß- und Kleinbuchstaben) komplett ab \$2000 im RAM. Kann man auf einen der beiden verzichten – man darf dann auch nicht die <ASCII/DIN>-Taste drücken –, können statt dessen noch zwei weitere Bildschirme eingerichtet werden. Insgesamt stehen uns also maximal sechs verschiedene Bildschirme zur Verfügung.

Ebenso, wie wir es eben mit dem Bildschirm gemacht haben, können wir auch das Attribut-RAM verschieben. Dazu dienen die folgenden Register:

Register 21 LSB der Startadresse und Register 20 MSB.

Wieder existiert im normalen RAM eine Speicherstelle, an die das MSB ebenfalls zu schreiben ist: 2607. Dazu braucht nun eigentlich nichts mehr gesagt werden, weil Sie alles schon vom Bildschirm-RAM her kennen.

Viel interessanter für uns ist das Register 25. Zwei Bit daraus, nämlich Bit 6 und Bit 7, werden wir uns etwas genauer ansehen. Das Bit 6 legt fest, woher die Attribute eines Zeichens auf dem Bildschirm stammen. Im Normalfall ist dieses Bit gleich 1. Das bedeutet, daß das Attribut-RAM Quelle der Attribute ist (also Farbe, Blinken, Unterstreichen, Revers, Groß- oder Kleinschreibung). Schalten wir aber statt dessen dieses Bit auf 0, dann kommt die Farbgebung durch das Register 26 zustande. Dort dienen die Bits 0 bis 3 zum Definieren der Hintergrund-, die Bits 4 bis 7 aber der Vordergrundfarbe. Der normale Inhalt von Register 25 ist 64 (es ist also lediglich Bit 6 gesetzt), so daß wir es uns einfach machen können:

```
POKE DEC("D600"),25:POKE DEC("D601"),0
```

schaltet alle Bits auf 0 und die Farben legen wir so fest: F1=Vordergrund, F2=Hintergrundfarbe. $F = 16 \times F1 + F2$ ergibt die Farbkennzahl in Register 26. Wir setzen dann die Farben so:

```
POKE DEC("D600"),26:POKE DEC("D601"),F
```

Auf diese Weise kann auch die Farbgebung im Hochauflösungs-Modus geschehen. Das ist nämlich der Zweck des Bit 7 vom Register 25. Der Inhalt 0 an dieser Stelle sorgt für die normale Textdarstellung. Schreiben wir aber eine 1 dort hinein, dann schaltet der VDC um auf die Abbildung einer Bit-Map. Im Gegensatz zur VIC-Bit-Map benötigt die VDC-Bit-Map allerdings volle 16000 Byte Speicherplatz, das heißt, daß das VDC-RAM praktisch vollständig als Bit-Map verwendet wird. Dafür ist aber die Auflösung in X-Richtung doppelt so groß, denn 80 Zeichen mal jeweils 8 Bit ergeben 640 verschiedene horizontale Bildpunkte. In der Vertikalen bleibt alles beim alten: 25 Zeilen zu je 8 Bit ergeben nach wie vor 200 Bildpunkte. Insgesamt stehen uns nun 128000 verschiedene Bildschirmpositionen zur Verfügung!

(Heimo Ponnath/dm)