Reise durch den C128 (Teil 4)

Unser Forscher ist weiter auf der Suche im C 128. Diesmal hat er den Video-Chip entdeckt und sofort unter die Lupe genommen.

Auf allerlei obskuren Umwegen erhielten wir wieder einen Bericht unseres Korrespondenten, der sich derzeit auf einer beschwerlichen Forschungsreise durch den unbekannten Kontinent C 128 befindet. Arg zerzaust sei er, schreibt er, und daß er voll Freude meinte, bekanntes Territorium zu betreten, als er des VIC-Chips ansichtig wurde, daß er sich aber nun verwirrt fragt, ob er einer Fata Morgana aufgesessen sei oder ob dieser Chip der Kategorie der »Rühr mich nicht an«-Wesen zugeordnet werden müsse.

Der VIC ist ein VIC — ist kein VIC...

Jetzt sind Sie schon zu Beginn ähnlich verwirrt, wie ich es war. Lassen Sie mich daher zunächst einmal die sicheren Tatsachen erzählen, damit wir wieder Boden unter den Füßen haben:

Der VIC ist nämlich allen, die den Commodore 64 kennen, ein alter Bekannter. Dort heißt er genaugenommen MOS 6566 Video-Interface-Controller. Im C 128 haben wir eine leicht geänderte Version dieses Chip vorliegen, den MOS 8564 Video-Interface-Controller, der sich aber — weil er auch im C 64-Modus verwendet wird — allem Anschein nach nur durch zwei zusätzliche Register vom 6566 unterscheidet. Bild 1 zeigt Ihnen alle Register, ihre Bedeutungen und von einigen auch die üblichen Inhalte:

Die beiden neuen Register sind 47 ($D02F, 53295) und 48 ($D030, 53296). 47 enthält — etwas rätselhaft — in den Bits 0 bis 2 eine Tastaturkennung. Interessanter scheint Register 48. Das Bit 0 dient dem Umschalten in den 2-MHz-Betrieb und das ist auch im C 64-Modus möglich. Allerdings ist dann auf dem 40-Zeichen-Bildschirm nichts mehr zu sehen (oder aber wüstes Geflimmer), weil der VIC diese doppelte Geschwindigkeit nicht mitmacht. Durch POKE 53296,1 kann der C 64 dann alle Operationen doppelt so schnell ausführen als sonst. Ein POKE 53296,0 läßt den Bildschirm wieder normal erscheinen und schaltet in den 1-MHz-Betrieb zurück.

Nun aber kommt das Verwirrspiel: Dies ist das einzige VIC-Register, das sich auf diese Art — also einfach durch POKEs oder mittels des Monitors — verändern läßt. Alle anderen Registerumbauten äußern sich bestenfalls in einem kurzen Aufflackern des 40-Zeichen-Bildschirmes... dann ist der Einheitszustand wieder hergestellt!

Offensichtlich befinden sich die VIC-Register fest in der Hand des Unterbrechungszyklus, der eine Reihe weiterer Bestandteile des äußeren und inneren Erscheinungsbildes unseres Computers ständig steuert. Das ist aber nur im C 128-Modus der Fall. Sobald Sie den C 64-Modus eingeschaltet haben, kann wieder jedes Register bedient werden. Nun ist dies hier kein Werk über die Grafik des C 64. Falls Sie mehr über die Programmierung aller Register des VIC erfahren möchten, verweise ich Sie auf mein Buch »C 64 Wunderland der Grafik«, Markt und Technik Verlag, München 1985, MT756. Das kann auch für den Assembler-Programmierer interessant sein, der es versteht, die Unterbrechungen zu beherrschen und der sich grafisch betätigen möchte.

Register	Adressen $	Adressen dez	Bit 7	6	5	4	3	2	1	0	Normaler Inhalt Text-M.	Hires-M.	Multicolor	Bemerkung
0—16	D000—D010	53248—53264	Sprite-Positionen								—	—	—	— Basic 7.0 — IRQ
17	D011	53265	msb Rasterregister	verändert. Hintergrund-Modus = 1, normal = 0	Bit-Map-Modus = 1, Text = 0	Bildschirm an = 1, aus = 0	Zeilenzahl 24 = 0, 25 = 1	Smooth Scrolling Y-Verschiebung			$1B abhängig	$BB von	$BB Bit 7	— teilweise durch Basic 7.0 — IRQ
18—21	D012—D015	53266—53269	LSB Rasterregister, Lichtgriffel, Sprites								—	—	—	— Basic 7.0 — IRQ
22	D016	53270	unbenutzt, beide Bits = 1		Reset 0 = VIC arbeitet	Multicolormode = 1	Spaltenzahl 39 = 0, 40 = 1	Smooth Scrolling X-Verschiebung			$C8	$C8	$D8	— teilweise durch Basic — IRQ
23	D017	53271	Sprite-Vergrößerung in Y-Richtung								—	—	—	— Basic 7.0 — IRQ
24	D018	53272	Bildschirmspeicherstartadresse				Text: Zeichenmusterquelle Bit-Map-Mode: 0 = Bit-Map unten, 1 = Bit-Map oben	unbenutzt		unbenutzt, immer = 1	$15	$79	$79	— teilweise durch Basic 7.0 — IRQ — Steuerbar 2064/5
25,26	D019—D01A	53273—53274	VIC-Unterberechnungs-Register								—	—	—	— Basic 7.0 — IRQ
27—46	D01B—D02E	53275—53294	Farb-Register: Hintergrund, Vordergrund, Multicolor, Sprites								—	—	—	— Basic 7.0 — teilw. IRQ
47	D02F	53295	unbenutzt (alle = 1)					Tastaturerkennung			$F8	$F8	$F8	?
48	D030	53296	unbenutzt (alle = 1)						Test (?)	Taktfrequenz: 1 MHz = 0, 2 MHz = 1	$FC	$FC	$FC	— Basic 7.0

Bild 1. Die VIC-Register im C 128

Für den C 128-Modus ist es im allgemeinen auch kaum erforderlich, VIC-Register anzusteuern. Es bleibt nur wenig übrig, was dieses kraftvolle Basic nicht vermag:
1) Der Textmodus mit verändertem Hintergrund existiert nirgends im Basic 7.0.
2) Die Anzahl der Zeilen und Spalten ist nicht veränderbar.
3) Das »smooth scrolling« kann nicht einfach programmiert werden
4) Der Bildschirmspeicher kann nicht verschoben werden, ebenso die Bit-Map und die Quelle der Zeichenmuster.

Wenn Sie nun nochmal Bild 1 ansehen, dann stellen Sie fest, daß diese Auslassungen drei Register betreffen: 17 ($D011, 53265), 22 ($D016, 53270) und 24 ($D018, 53272), von denen zweifellos das letzte das interessante ist. Was fängt man aber mit diesem »Rühr mich nicht an«-Chip an? Assembler-Spezis werden sagen, daß man eben einfach eine eigene Unterbrechungsroutine schreibt und den Vektor $314/$315 darauf richtet: Von da an können alle VIC-Spezialitäten freimütig verwendet werden. Was aber macht ein Basic-Programmierer?

Ein Ausweg aus der Sackgasse?

Ein gründliches Durchforsten der »erweiterten Zeropage« fördert schließlich einige interessante Speicherstellen zutage. Dem Register 24 nämlich — eben gerade dem erwähnten recht interessanten! — entsprechen zwei Speicherstellen:
2604 $0A2C VIC Text-Basis
2605 $0A2D VIC Bit-Map-Basis
Außerdem gibt es da noch eine:
2619 $0A3B Page des Bildschirm-Speichers
(Im C 64-Modus ist letztere die Entsprechung zur Speicherstelle 648)

Sowohl 2504 als auch 2605 entsprechen — bis auf das Bit 0, das aber auch in 53272 ohne Bedeutung ist — exakt dem Register 24 im jeweiligen Modus (also Text- oder Bit-Map-Modus). Alle drei Speicherstellen werden von den Unterbrechungen nicht behelligt — jedenfalls nicht zu unserem Nachteil wie der VIC selbst. Das, was wir in diese Speicherstellen schreiben, wird flugs nach 53272 transportiert und somit beherrschen wir den Bildschirm, die Bit-Map und die Quelle der Zeichenmuster. Oder doch nicht so ganz? Aber dazu kommen wir später noch. Sehen wir uns zunächst einmal an, was die verschiedenen Inhalte von 53272, 2604 oder 2605 zu bewirken imstande sind. Die Bits 4 bis 7 legen die Startadresse des Bildschirmspeichers fest. Dort findet sich im Einschaltzustand der binäre Inhalt 0001, in den Grafik-Modi aber der Inhalt 0111. Das Bild 2 zeigt die weiteren Möglichkeiten:

Speicherinhalt: W binär in Bits				dezimal (Bits 0 bis 3 als 0 angenommen)	Bildschirmadresse dezimal	$	Bemerkung
7	6	5	4				
0	0	0	0	0	0	0	
0	0	0	1	16	1024	400	Text-Bildschirm
0	0	1	0	32	2048	800	
0	0	1	1	48	3072	900	
0	1	0	0	64	4096	1000	
0	1	0	1	80	5120	1400	
0	1	1	0	96	6144	1800	
0	1	1	1	112	7168	1900	Grafik-Bildschirm
1	0	0	0	128	8192	2000	
1	0	0	1	144	9216	2400	
1	0	1	0	160	10240	2800	
1	0	1	1	176	11264	2900	
1	1	0	0	192	12288	3000	
1	1	0	1	208	13312	3400	
1	1	1	0	224	14336	3800	
1	1	1	1	240	15360	3900	

Bild 2. Die Bildschirmspeicher-Startadressen bei verschiedenen Inhalten der Bits 4 bis 7 in 53272 2,2 2604 oder 2605.

Erreichbar sind diese anderen Bildschirme durch die Kommandos:

```
POKE 2604,(PEEK(2604)AND15)OR W
```

im Text- und

```
POKE 2605,(PEEK(2605)AND15)OR W
```

im Grafik-Modus. W ist dabei der Dezimalwert, der in Bild 2 angegeben wurde. Bevor Sie damit anfangen, alles mögliche auszuprobieren, hier noch ein paar Tips: Speichern Sie eventuell vorhandene Programme sicherheitshalber ab. Benutzen Sie — sofern vorhanden — beide Bildschirme, denn häufig verlieren Sie aus den verschiedensten Ursachen auf dem 40-Zeichen-Bildschirm jede Kontrolle über den Cursor und Befehlstexte. Davon wird dann der 80-Zeichen-Bildschirm seltener berührt, so daß Sie relativ gute Chancen haben, ohne RESET weiterzuarbeiten. Ein letzter Tip noch: Wundern Sie sich über nichts! Die Speicherkonstruktion unseres Computers ist derart vielfältig, daß man sich nie ganz sicher sein kann, woher eigentlich manche Bildschirmdarstellungen rühren.

Sie werden bemerkt haben, daß die höchste Bildschirmstartadresse bei 15360 liegt. Pro BANK haben wir aber 64 KByte zur Verfügung. Was können wir tun, um dieses Mißverhältnis zu beheben? Dazu müssen wir erst mal wissen, daß der VIC-Chip nur über 14 Adressenleitungen verfügt (im Gegensatz zum Prozessor, der 16 davon bedienen kann). Was bedeutet das? Die Wirkung ist, daß mit 16 Bits alle Adressen im Speicherraum zwischen
0 = 0000 0000 0000 0000
und
65535 = 1111 1111 1111 1111
ansteuerbar sind, mit 14 Bit aber liegt die Höchstgrenze bei
16383 = 11 1111 1111 1111

Das entspricht 16 KByte und davon befinden sich vier in einer BANK. Es gibt nun im C 64-Modus ein Register im CIA2, dem sogenannten NMI-CIA (das ist der »Complex Interface Adapter«, also ein Ein- und Ausgabe-Baustein), welches festlegt, auf welchen 16-KByte-Bereich in einer BANK der VIC Zugriff hat. Es handelt sich um den DATA-Port A bei $DD00 (das ist dezimal 56576). Nach allen Untersuchungen liegt an derselben Stelle auch im C 128-Modus dieser Chip, so daß man das Register auch hier benutzen kann. Der VIC-Zugriff wird durch die Bits 0 und 1 gesteuert. Was man mit den verschiedenen Belegungen erreicht, zeigt Ihnen das Bild 3:

Speicherinhalt binär in Bit/ Dezimalwert			Abschnitt	Speicherstellen (dezimal)		Bemerkung
1	0	J		von	bis	
1	1	3	0	0	16383	Einschaltwert
1	0	2	1	16384	32767	
0	1	1	2	32768	49151	
0	0	0	3	49152	65535	

Bild 3. Der 16-KByte-Zugriff des VIC in Zusammenhang mit dem Inhalt der Bit 0 und 1 von Port A des NMI-CIA (Speicherstelle 56576) (CIA = Complex Interface Adapter)

Im Einschaltzustand steht hier 11, was den unteren 16-KByte-Bereich für den VIC freigibt.

Durch Kombination der Bitwerte in 2604 (oder 2605) und derer aus Bild 3 lassen sich nun theoretisch 64 Bildschirme definieren. Die Befehle zur Änderung des 16-KByte-Bereiches sind:

```
BANK15:POKE56576,(PEEK(56576)AND252)OR I
```

oder

```
BANK15:POKE56576,(PEEK(56576)AND252)OR 3
```

Dabei ist I der Dezimalwert der Bitkombination in Bild 3 und der zweite Befehl dient zur Wiederherstellung des Normalzustandes.

Sie finden hier noch das Programm BILDSCHIRMSP (Listing 1) abgedruckt, welches Ihnen das Durchprobieren sämtlicher Möglichkeiten erleichtern soll:

```
10 REM ***** BILDSCHIRMSPEICHER VERSCHIEBEN
   *****
20 INPUT "I,W";I,W: PRINT CHR$(147)
30 BANK 15: POKE 56576,(PEEK(56576) AND 252)
    OR I
40 BANK 0: POKE 2604,(PEEK(2604) AND 15) OR
   W
50 P=(W/16*1024+16348*(3-I))/256
60 POKE 2619,P: REM (BEI I=1 UND W =0 DAZUFU
   EGEN: )POKEDEC("8000"),1: POKEDEC("8001")
   ,2
70 PRINT CHR$(147)I,W: END
80 REM ***** MOEGLICHKEIT ZUM AUSPROBIEREN,
   WEITER MIT CONT *****
90 REM ***** ZURUECK ZUM NORMALEN BILDSCHIRM
    *****
100 PRINT CHR$(147)
110 BANK 15: POKE 56576,199: BANK 0: POKE 26
    04,20: POKE 2619,4
120 I=3: W=16: PRINT CHR$(147)I,W
130 END
```

Listing 1. Ein Programm zum Testen von 64 Bildschirmen

Aber: Speichern Sie es vor dem Starten unbedingt ab und beachten Sie die obigen Tips, denn da tauchen allerlei Merkwürdigkeiten bei der Benutzung auf.

Sollten Sie nur mit dem 40-Zeichen-Bildschirm arbeiten, empfiehlt es sich, alle Operationen, die mit dem neuen Bildschirm löschender- oder druckenderweise geschehen, aus dem Programm herauszunehmen. Oder aber, Sie haben die Geduld, falls etwas schiefläuft, jedesmal nach einem RESET das Programm neu zu laden. Sehen wir uns nun einige Varianten an:

I=3 und W=0 legt den Bildschirmspeicher genau auf die Speicherplätze 0 bis 999, also auf die erweiterte Zeropage. Man kann einigen Speicherstellen bei der Arbeit zusehen, beispielsweise dem Timer-Register in der Zeile 5 links. Falls Sie den 80-Zeichen-Bildschirm verwenden, können Sie dort allerlei Kommandos eingeben und zusehen, wie sich auf dem 40-Zeichen-Bildschirm Speicherstellen verändern.

I=3 und W=16 erzeugt wieder den normalen Bildschirm, wohingegen I=3 und W=32 wieder Teile der erweiterten Zeropage von 2048 (Basic-Stapel) bis 3047 (Kassettenpuffer) zeigt. Auch hier gibt es wieder veränderliche Speicherstellen wie beispielsweise 2614 (das ist ein Korrekturzähler für den 50-Hz-Betrieb). Auch die beiden nächsten Kombinationen mit I=3, nämlich die mit W=48 und W=64 zeigen noch Teile des Systemspeichers. Die Kombination I=3 und W=112 sollte eigentlich den Start des Basic-RAM ab 7168 (bis 8167) auf dem Bildschirm erscheinen lassen, was dort aber zu sehen ist, ist nicht das Programm. Eines der noch nicht gelösten Rätsel im C 128!

Alle weiteren Orte des Bildschirmspeichers (W=128 bis W=240) ergeben nutzbare Bildschirme, sofern nicht die Grafik eingeschaltet wird, die sich in diesem Gebiet tummelt. Wenn wir nun statt I=3 auch die Kombinationen mit I=2,1,0 verwenden, stellen wir fest, daß hier keine normalen Bildschirme möglich sind. Weder PRINT noch die Cursorbedienung läuft hier ohne Störung. Der eine oder andere Absturz kann auftreten. Bemüht man sich aber mal, mittels des Monitors in diese Speicherbereiche den Löschcode $20 zu schreiben (mit dem F-Kommando) und POKEt dann den Bildschirmcode in den neuen Bildschirmspeicher hinein, dann funktioniert das einwandfrei, wenn man dabei die kritischen Speicherbereiche der I/O-Bausteine zwischen $D000 und $DFFF und vor allem die MMU-Register bei $FF00 und folgende ausläßt. Dazu kommen wir nachher noch. Jedenfalls können — mit der Einschränkung, daß wir die Zeichen durch POKE-Kommandos einschreiben müssen — 51 Bildschirme in BANK 0 definiert werden.

Eigene Zeichen benutzen

Sollten Sie mal in der Verlegenheit sein, andere als die vorhandenen Zeichen unseres Computers zu benötigen, dann ist auch das möglich. In der Speicherstelle 2604 dienen die Bits 1 bis 3 der Festlegung der Startadresse des Zeichenmusterspeichers. Bild 4 schlüsselt die Zuordungen für den ersten 16-KByte-Bereich auf:

Speicherinhalt: binär in Bits/dezimal (Bit = 0)				Startadresse der Zeichenmuster		Bemerkung
3	2	1	Z	dezimal	$	
0	0	0	0	0	0	
0	0	1	2	2048	800	
0	1	0	4	4096	1000	Einschaltzustand
0	1	1	6	6144	1800	
1	0	0	8	8192	2000	
1	0	1	10	10240	2800	
1	1	0	12	12288	3000	
1	1	1	14	14336	3800	

Bild 4. Zusammenhang der Zeichenmusterquellen mit dem Inhalt der Bits 1 bis 3 in 2604

Das Umschalten auf eine andere Zeichenmusterquelle geschieht dann mittels

```
POKE 2604,(PEEK(2604)AND240)OR Z
```

Dabei ist Z die Dezimalzahl in Spalte 4 des Bildes.

Interessanterweise deutet dieses Merkmal im Einschaltzustand auf die Speicherstelle $1000 (also 4096). Wenn wir uns aber ansehen, was dort per Monitor zu finden ist, dann liegt da die erweiterte Zeropage mit allerlei wichtigen Eintragungen. Wieder eines der ungelösten Rätsel. Dieses wurde übrigens auch schon beim C 64 gestellt und immer noch nicht ganz befriedigend beantwortet. Jedenfalls holt sich der Computer im Normalzustand seine Zeichenmuster aus dem Zeichen-ROM, was es nicht möglich erscheinen läßt, diese zu verändern.

Es geht aber doch:

1) Wir kopieren den Inhalt des Zeichen-ROM in einen RAM-Bereich

2) Wir richten den Wegweiser in den Bits 1 bis 3 der Speicherstelle 2604 auf diesen RAM-Zeichenspeicher

3) Wir verändern die Muster, die ja nun im RAM liegen.

Das Programm ZEICHENSATZ (Listing 2) soll Ihnen das am Beispiel des Buchstabens A erläutern.

```
10 REM ***** ZEICHENSATZ UMSCHALTEN UND VERA
   ENDERN *****
20 REM ***** HERUNTERKOPIEREN *****
30 PRINT CHR$(147)"BITTE EINIGE ZEIT GEDULD!
   "
40 FAST
50 BANK 14
60 FOR I=0 TO 4095
70 POKE DEC("3000")+I,PEEK(DEC("D000")+I)
80 NEXT I
90 REM ***** UMSCHALTEN *****
100 SLOW
110 BANK 0
120 POKE 2604,(PEEK(2604) AND 240) OR 12
130 PRINT CHR$(147)"ABRAKADABRA"
140 SLEEP 2
150 REM **** VERAENDERN DES BUCHSTABEN A ***
    **
160 B=12296
170 POKE B,102: POKE B+1,0: POKE B+2,24: POK
    E B+3,24
180 POKE B+4,129: POKE B+5,102: POKE B+6,60:
     POKE B+7,0
```

Listing 2. Der Buchstabe A erhält ein entschieden freundlicheres Gesicht

Die Zeichen verschieben wir nach $3000 und folgende. Jeweils 8 Byte gehören zu einem Zeichenmuster. Bei 12288 beginnt der »Klammeraffe« als nulltes Zeichen, der Buchstabe A hat seine 8 Byte von 12296 bis 12303 liegen und dorthinein POKEn wir andere Inhalte. Starten Sie das Programm, dann sollten Sie sich mit etwas Geduld wappnen, denn das Kopieren der 4096 Bytes auf diesem Weg dauert eine Weile. Das Programm schaltet dann die Speicherstelle 2604 auf den neuen Zeichenbereich um und druckt auf den Bildschirm das Wort ABRAKADABRA. Einen Augenblick später erscheint das A im neuen Gewand. Gefällt es Ihnen?

Bei Ihnen hat das nicht funktioniert? Dann haben Sie alles auf dem 80-Zeichen-Bildschirm laufen lassen. Die Veränderungen spielen sich aber nur auf dem 40-Zeichen-Bildschirm ab.

Interessant ist, daß die Umschaltung auf den DIN-Zeichensatz keinen Einfluß auf die neuen Zeichen hat. Wenn vor dem

Programmlauf schon der DIN-Zeichensatz eingeschaltet war, erscheint das dazugehörige A im neuen Gewand. Merkwürdig! Welchen Zeichensatz haben wir eigentlich bei $D000 herausgelesen? Wenn Sie durch gleichzeitiges Drücken der Commodore- und der Shift-Taste auf die kleinen Buchstaben umschalten, finden Sie das normale Zeichenbild. Der Grund dafür ist, daß wir in deren Zeichenmuster nicht eingegriffen haben, denn die liegen erst ab $3800. Bild 5 zeigt Ihnen, wo sich im Zeichen-ROM (und in gleicher Reihenfolge nach dem Kopieren ab $3000) welche Zeichen befinden:

Block	Startadresse ($)	Inhalt
0	D000	große Buchstaben
	D200	Grafikzeichen
	D400	große Buchstaben (revers)
	D600	Grafikzeichen (revers)
1	D800	kleine Buchstaben
	DA00	große Buchstaben, Grafikzeichen
	DC00	kleine Buchstaben (revers)
	DE00	große Buchstaben, Grafikzeichen (revers)

Bild 5. Die Anordnung der Zeichenmuster im Zeichen-ROM (und nach dem Kopieren nach $3000)

Ein Problem ist es noch, den richtigen Ort für die neuen Zeichenmuster zu finden. Es sollte weder einer sein, der durch ein längeres Basic-Programm überschrieben werden kann (wie im Beispiel ZEICHENSATZ), noch darf er in einem anderen 16-KByte-Abschnitt liegen als der Bildschirmspeicher, noch sollte er bei 4096 beginnen, denn dort greift der Computer immer auf das Zeichen-ROM zu. Ein Weg, das Problem zu lösen ist es, den Basic-Start nach $4000 hochzulegen und die Zeichenmuster ab $2000 oder $3000. Das Hochlegen des Programms nach $4000 besorgt einfach der GRAPHIC-Befehl für uns. Wir brauchen dazu im Programm ZEICHENSATZ lediglich eine Zeile einzufügen:

```
25 GRAPHIC1,1:GRAPHIC0
```

Allerdings können wir dann keine Grafik-Befehle mehr verwenden, denn dann liegen die Zeichenmuster in der Bit-Map und jede Änderung darin wirkt sich auf die Zeichen aus. Besonders verheerend wirkt ein SCNCLR1. Aber auch ein einfacher DRAW-Befehl kann Überraschendes bewirken. Probieren Sie doch mal

```
DRAW1,9,0 TO 9,199
```

nach dem Lauf unseres veränderten Programmes.

Übrigens: Ein Druck auf die STOP- und die RESTORE-Taste rückt alle Verhältnisse wieder ins normale Dasein zurück.

Mehrere Bit-Maps

Die Speicherstelle 2605 haben wir bislang sträflich vernachlässigt. Die oberen 4 Bit haben hier dieselbe Bedeutung wie in 2604, nur daß in den Grafik-Modi der Bildschirmspeicher als Farb-RAM dient, wie wir es beim COLOR-Befehl ausführlich kennenlernen. Das Bit 3 aber hat mit der Position der Bit-Map zu tun. Bild 6 zeigt Ihnen, was dadurch bewirkt wird (bezogen auf den unteren 16-KByte-Bereich):

Bit 3	Bit-Map in Abschnitt 0 (dezimal)	
	von	bis
0	0	7999
1	8192	16191

Bild 6. Wirkung des Bit 3 in 2605 auf die Lage der Bit-Map.

Das Setzen dieses Bit 3 geschieht durch

```
POKE2605,PEEK(2605)OR8
```

das Löschen mittels

```
POKE2605,PEEK(2605)AND247
```

Nun rechnen wir rasch mal nach: Wir haben vier Bereiche zu je 16 KByte, auf die wir den VIC steuern können. In jeden dieser Bereiche passen zwei Bit-Maps (immer eine oben und eine unten). Dabei ist noch zu bedenken, daß der jeweilige Bildschirmspeicher im gleichen 16-KByte-Bereich liegen muß wie die Bit-Map. Damit Sie all diese verschiedenen Möglichkeiten ausprobieren können, wurde das Programm BIT-MAPTEST (Listing 3) geschrieben:

Nach dem RUN werden drei Zahlen abgefragt. I entspricht dem Kennzeichen des jeweiligen 16-KByte-Abschnittes, W dem der Bildschirmstartadresse (bezogen auf den jeweiligen 16-KByte-Bereich) und B kann 0 oder 1 sein und legt die Bit-Map nach unten oder oben in dem jeweiligen 16-KByte-Abschnitt. Nach der Eingabe berechnet schaltet das Programm die neue Bit-Map und den neuen Bildschirm ein, löscht den Bereich und den Bildschirmspeicher und zeichnet eine Sinuskurve. Ein Tastendruck stellt wieder die normalen Verhältnisse her.

```
10 REM ***** VERLAGERUNG VON BILDSCHIRM UND
   BIT-MAP *****
20 REM **** NEUE WERTE EINGEBEN ****
30 F=6: DEF FN A(X)=50*SIN(X/30)+100
40 PRINT CHR$(147) CHR$(17)"EINGABE DER WERT
   E" CHR$(17)
50 PRINT CHR$(17)"I(SIEHE TAB.11) ABSCHNITT-
   KENNZIFFER" CHR$(17)
60 PRINT "W(SIEHE TAB.12) BILDSCHIRM-KENNZIF
   FER" CHR$(17)
70 PRINT "B=0,BIT-MAP IM UNTEREN ABSCHNITT-B
   EREICH"
80 PRINT " =1,BIT-MAP IM OBEREN ABSCHNITT-BE
   REICH" CHR$(17)
90 PRINT CHR$(17): INPUT "I,W,B=";I,W,B: A1=
   3-I: A2=W/16*1024+A1*16384
100 A3=A1*16384+B*8192
110 P=A2/256: PRINT CHR$(17) CHR$(17)"MIT DI
   ESEN EINGABEN HABEN SIE"
120 PRINT "DEN ABSCHNITT "A1" GEWAEHLT."
130 PRINT "IHR BILDSCHIRM STARTET BEI "A2
140 PRINT "UND IHRE BIT-MAP BEI "A3
150 PRINT CHR$(17)"IST DAS SO IN ORDNUNG?(J/
   N)"
160 GET A$: IF A$="" THEN 160
170 IF A$="N" THEN 40
180 PRINT CHR$(147)
190 REM **** SPEICHERUMBAUTEN ****
200 BANK 15: POKE 56576,(PEEK(56576) AND 252
   ) OR I
210 POKE 56578,PEEK(56578) OR 3
220 BANK 0: POKE 2605,(PEEK(2605) AND 15) OR
    W
230 POKE 2619,P
240 GRAPHIC 1
250 POKE 2605,PEEK(2605) OR(8*B)
260 FOR J=0 TO 7999: POKE A3+J,0: NEXT J
270 FOR J=0 TO 999: POKE A2+J,F: NEXT J
280 FOR X=0 TO 319: Y=FN A(X)
290 BY=(X AND 504)+40*(Y AND 248)+(Y AND 7):
    BI=7-(X AND 7)
300 POKE A3+BY,PEEK(A3+BY) OR(2↑BI): NEXT X
310 GET A$: IF A$="" THEN 310
320 REM **** NORMALZUSTAND WIEDERHERSTELLEN
   ****
330 BANK 0: POKE 2604,20: POKE 2605,112: POK
   E 2619,4
340 BANK 15: POKE 56578,63: POKE 56576,199
350 GRAPHIC 0: GRAPHIC 5
360 PRINT CHR$(147): END
```

Listing 3. Wieviele Bit-Maps sind sinnvoll nutzbar?

16-KByte-Abschnitt 0, oberer Bereich

Das ist zwar die normale Bit-Map. Die kann aber dadurch interessant werden, daß wir den Bildschirmspeicher und damit das Farb-RAM ändern können.

16-KByte-Abschnitt 1, oberer Bereich

Wenn der Bildschirm an das obere Ende des unteren Bereiches gelegt wird, damit er nicht dem bei $4000 beginnenden Basic-Text in die Quere kommt, funktioniert diese Kombination einwandfrei. Der untere Bereich ist für die Bit-Map ungeeignet, weil er das Basic-Programm beinhaltet.

16-KByte-Abschnitt 2, beide Bereiche.

Hier sind sowohl unten als auch oben Bit-Maps möglich.

16-KByte-Abschnitt 3

Keiner der beiden Bereiche ist nutzbar, weil man im unteren Ein- und Ausgabebereich herumpfuscht, im oberen aber den Speicherteil ab $FF00 stört und damit unkontrollierte Verhältnisse im Computer schafft.

Der Bericht unseres Korrespondenten endet hier. Er wird sich auf die Suche nach der MMU machen, um uns dann davon zu erzählen. (Heimo Ponnath/dm)