Der MOS 6567/6569 Videocontroller (VIC-II)

und seine Anwendung im Commodore 64

Originaltext von Christian Bauer
<bauec002@goofy.zdv.uni-mainz.de>

28.Aug.1996

Grafiken von Herwig Siebenhofer

Layout von fieserWolF / Abyss-Connection, Juli 2019

Inhaltsverzeichnis

.

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Dieser Artikel ist der Versuch, die Ergebnisse der zahlreichen Untersuchungen über den Grafikchip "6567/6569 Video Interface Controller (VIC-II)" (im folgenden einfach "VIC" genannt), der im legendären Commodore 64 zum Einsatz kommt, zu ordnen und damit eine umfassende Referenz über dessen spezifizierte und nicht spezifizierte Eigenschaften zur Verfügung zu stellen. Er richtet sich erster Linie an C64-Programmierer und Autoren von C64-Emulationen, allerdings sollte die Lektüre auch für "Außenstehende", die sich für Hardwaredesign und -programmierung und die Ausnutzung eines Computers bis zum letzten Bit interessieren, aufschlussreich sein. Aus diesem Grund wurden auch einige Grundlagen mit aufgenommen, die für erfahrene C64-Programmierer ein alter Hut sind (z.B. die Speicheraufteilung).

Die Beschreibung der nicht spezifizierten Eigenschaften basiert auf Messungen, die Marko Mäkelä, Andreas Boose, Pasi Ojala, Wolfgang Lorenz und ich (und zahlreiche Ungenannte) im Laufe der letzten Jahre vorgenommen haben. Dabei werden auch interne Register und Abläufe im VIC angesprochen.
Da die Innenschaltung des VIC nicht zur Verfügung steht, kann es sich dabei natürlich nur um Spekulationen handeln, aber es wurde in allen Fällen ein Modell gewählt, das die beobachteten Phänomene mit dem geringsten Schaltungsaufwand erklärt. So wurde z.B. beim Videomatrixzähler (VC) einem Modell mit zwei einfachen Zählern einem aufwendigeren mit einem +40-Addierer der Vorzug gegeben.

Obwohl einige Messungen mit einem Oszilloskop direkt am Chip stattfanden, beruhen die meisten Erkenntnisse jedoch auf Testprogrammen auf dem C64 und deren Vergleich mit der Implementierung in Einzelzyklusemulationen wie "Frodo SC".

2 Die Architektur des Commodore 64

Dieses Kapitel behandelt den grundlegenden Hardwareaufbau des C64 und die Anbindung des VIC an den Rest des Systems.

2.1 Übersicht

Der C64 besteht im Wesentlichen aus folgenden Baugruppen:

- 6510 8-Bit Mikroprozessor
- 6567/6569 VIC-II Grafikchip
- 6581 SID Soundchip
- Zwei 6526 CIA I/O-Chips
- 64KB DRAM (64K*8 Bit) als Hauptspeicher
- 0,5KB SRAM (1K*4 Bit) als Farb-RAM
- 16KB ROM (16K*8 Bit) für Betriebssystem und Basic-Interpreter
- 4KB ROM (4K*8 Bit) als Zeichengenerator

Die meisten Chips sind in NMOS-Technologie gefertigt.

2.2 6510 Prozessor

Der 6510-Mikroprozessor [1] besitzt einen 8-Bit-Daten- und 16-Bit-Adressbus und ist zum bekannten 6502 binärkompatibel. Er verfügt über zwei externe Interruptmöglichkeiten (eine

maskierbare (IRQ) und eine nicht maskierbare (NMI)) und als Besonderheit einen 6 Bit breiten bidirektionalen I/O-Port. Er wird im C64 mit einer Taktfrequenz von ca. 1MHz betrieben.

ø2	Prozessortakt Ausgang Dieses Taktsignal ist die Grundlage des gesamten Bus-Timing. Seine Frequenz beträgt 1022,7 kHz (NTSC-Rechner) bzw. 985,248 kHz (PAL-Rechner). Eine Periode dieses Signals entspricht einem Taktzyklus, der aus zwei Phasen besteht: Während der ersten Taktphase ist ø2 Low, während der zweiten Phase High (daher auch der Name 'ø2' für "Phase 2"). Der 6510 greift nur während der zweiten Taktphase auf den Bus zu, der VIC normalerweise während der ersten Phase.
R/W	Dieses Signal zeigt einen Lese- (R/W High) oder Schreibzugriff (R/W Low) an.
IRQ	Ist dieser Eingang auf Low-Pegel, wird eine Interruptbearbeitung ausgelöst, sofern der Interrupt über ein Bit im Statusregister freigegeben wurde. Die Unterbrechung erfolgt frühestens nach zwei Taktzyklen beim Erreichen des nächsten Befehls. Mit diesem Pin kann der VIC einen Interrupt im Prozessor auslösen. Interrupts werden nur erkannt, wenn RDY high ist.
RDY	Ist diese Leitung während eines Lesezugriffs Low, hält der Prozessor an und gibt auf Adressleitungen die Adresse, auf die zugegriffen werden sollte, aus. Bei Schreibzugriffen wird das Signal ignoriert. Im C64 wird RDY benutzt, um den Prozessor anzuhalten, wenn der VIC für den Zeichenzeiger- und Spritedatenzugriff zusätzliche Buszyklen benötigt. Es ist direkt mit dem Signal BA des VIC verbunden.
AEC	Mit diesem Pin werden die Adressleitungen hochohmig geschaltet. Dies dient dazu, den ProzessorAdressbus bei Zugriffen des VIC lahmzulegen. Das Signal ist mit dem Ausgang AEC des VIC verbunden.
P0-P5	Dies ist der eingebaute 6-Bit-I/O-Port. Jede Leitung kann einzeln als Ein- oder Ausgang programmiert werden. Dazu sind ein Datenrichtungsregister bei Adresse 0 und ein Datenregister bei Adresse 1 prozessorintern in den Adressraum eingeblendet. Daher sollte man annehmen, dass der Prozessor nicht auf die RAM-Adressen 0 und 1 zugreifen kann (sie werden ja vom I/O-Port überlagert), aber dazu später mehr...

Tabelle 1: wichtige Signale des 6510 Prozessor

2.3 6567/6569 Grafikchip

Die Grafikchips der 656*-Reihe von MOS Technologies wurden ursprünglich für den Einsatz in Videospielen und Grafikterminals entwickelt. Da der Absatz in diesen Märkten jedoch eher bescheiden war, entschloss sich Commodore, als sie eigene Heimcomputer planten, die Chips dafür zu verwenden.

Im C64 fand der "Video Interface Controller II (VIC-II)" [2] Verwendung, der 3 textbasierte (40×25 Zeichen mit je 8×8 Pixeln) und 2 bitmapbasierte (320×200 Pixel) Videomodi beherrscht, über 8 Hardwaresprites und eine feste Palette von 16 Farben verfügt und bis zu 16KB dynamisches RAM verwalten kann (inklusive der Erzeugung von RAS und CAS und dem Refresh des RAM).
Außerdem besitzt er noch einen Eingang für Lichtgriffel und die Möglichkeit, Interrupts auszulösen.

Zwei VIC-Typen kommen im C64 vor: Der 6567 in NTSC-Rechnern und der 6569 in PAL-Rechnern. Von beiden Typen gibt es mehrere Maskenrevisionen, wovon allerdings bis auf den 6567R56A die Unterschiede relativ belanglos sind.
In neueren C64-Versionen kommen die gleichwertigen Bausteine 8562 (NTSC) und 8565 (PAL) zum Einsatz. Im folgenden ist aber immer von 6567/6569 die Rede. Alle Aussagen lassen sich auf

die 856*-Chips übertragen. Es gibt außerdem noch den 6566, der zum Anschluss an statisches RAM entworfen wurde, aber für den C64 keine Rolle spielt.

A0-A13	Der 14-Bit-VideoAdressbus, mit dem der VIC 16KB Speicher Adressieren kann. Dabei sind die Adressbits A0-A5 und A8-A13 jeweils paarweise (d.h. A0/A8, A1/A9 etc.) auf einem Pin gemultiplext. Die Bits A6-A11 sind (zusätzlich) als einzelne Leitungen ausgeführt.
D0-D11	Ein 12 Bit breiter Datenbus, über den der VIC auf den Speicher zugreift. Die unteren 8 Bit sind mit dem Hauptspeicher und dem Prozessordatenbus verbunden, die oberen 4 Bit mit einem besonderen 4 Bit breiten statischen Speicher (1024 Adressen, A0-A9), in dem Farbinformationen gespeichert werden, dem Farb-RAM.
IRQ	Dieser Ausgang ist an den IRQ-Eingang des Prozessors angeschlossen und gibt dem VIC die Möglichkeit, Interrupts zu erzeugen. Der VIC hat vier Interruptmöglichkeiten: Beim Erreichen einer bestimmten Rasterzeile (Rasterinterrupt), bei der Kollision von zwei Sprites untereinander, bei der Kollision von Sprites mit Grafikdaten und bei einer negativen Flanke am Lichtgriffel-Eingang.
BA	Mit diesem Signal zeigt der VIC an, dass der Bus für den Prozessor in der zweiten Taktphase (ø2 High) zu Verfügung steht. Normalerweise ist BA High, da der VIC meistens nur während der ersten Phase zugreift. Für die Zeichenzeiger- und Spritedatenzugriffe benötigt der VIC den Bus jedoch auch während der zweiten Phase. In diesem Fall geht BA drei Zyklen bevor der VIC zugreift auf Low, danach bleibt AEC auch während der zweiten Phase Low und der VIC greift zu. Warum drei Zyklen? Wie bereits beschrieben, ist BA mit der RDY-Leitung des Prozessors verbunden, aber diese Leitung wird nur bei Lesezugriffen abgefragt, der Prozessor kann bei Schreibzugriffen nicht unterbrochen werden. Allerdings führt der 6510 nie mehr als drei Schreibzugriffe hintereinander aus (siehe dazu [5]).
AEC	Dieser Pin ist mit den gleichnamigen Signal des Prozessors verbunden (siehe dort). Es gibt den Status der Daten- und Adressbustreiber des VIC wieder. Bei High befinden sie sich im Tri-State. Normalerweise ist AEC während der ersten Taktphase (ø2 Low) Low und während der zweiten Phase High, entsprechend dem Schema, dass der VIC während der ersten Phase auf den Speicher zugreift und der 6510 während der zweiten Phase. Wenn der VIC auch während der zweiten Taktphase zugreift, bleibt AEC Low.
LP	Dieser Eingang ist für den Anschluss eines Lichtgriffels vorgesehen. Bei einer negativen Flanke wird die aktuelle Position des Rasterstrahls in den Registern LPX und LPY gelatcht. Da dieser Pin im C64 eine Leitung der Tastaturmatrix mitbenutzt, lässt er sich auch per Software ansprechen.
øIN	Hier liegt der Pixeltakt von 8,18 MHz (NTSC) bzw. 7,88 MHz (PAL) an, der aus der Quarzfrequenz gewonnen wird. Pro Bustaktzyklus (ø2) werden acht Pixel ausgegeben.
ø0	Aus dem Pixeltakt an øIN erzeugt der VIC durch Teilung durch acht den Systemtakt von 1,023 MHz (NTSC) bzw. 0,985 MHz (PAL), der an diesem Pin ausgegeben wird und an den Prozessor geht. Dieser erzeugt daraus das Signal ø2.

Tabelle 2: wichtige Signale des 6567/6569 Grafikchip

2.4 Speicher

Für die Grafik spielen drei Speicherbereiche im C64 eine Rolle:

- Die 64KB Hauptspeicher
- Das 1K*4 Bit Farb-RAM

- Das 4KB Zeichengenerator-ROM (Char-ROM)

In den folgenden beiden Abschnitten wird dargestellt, wie sich diese Speicherbereiche den Adressraum aus Sicht der CPU und des VIC teilen. Anschließend wird auf die Grundlagen des Speicherzugriffs und des DRAM-Handling eingegangen.

2.4.1 Speicheraufteilung aus Sicht des 6510

Der 6510 kann mit seinen 16 Adressleitungen 64KB linear Adressieren. Mit Hilfe eines speziellen PAL-Bausteins im C64 können über die Leitungen des 6510-I/O-Ports und über Steuersignale am Erweiterungsport viele verschiedene Speicherkonfigurationen eingestellt werden (siehe dazu [3]).
Hier soll jedoch nur die Standardeinstellung beschrieben werden. Die anderen Konfigurationen ändern die Lage der verschiedenen Bereiche nicht, sondern blenden nur zusätzliche Bereiche des Hauptspeichers ein.

Hier also die Speicherkarte aus Sicht des 6510:

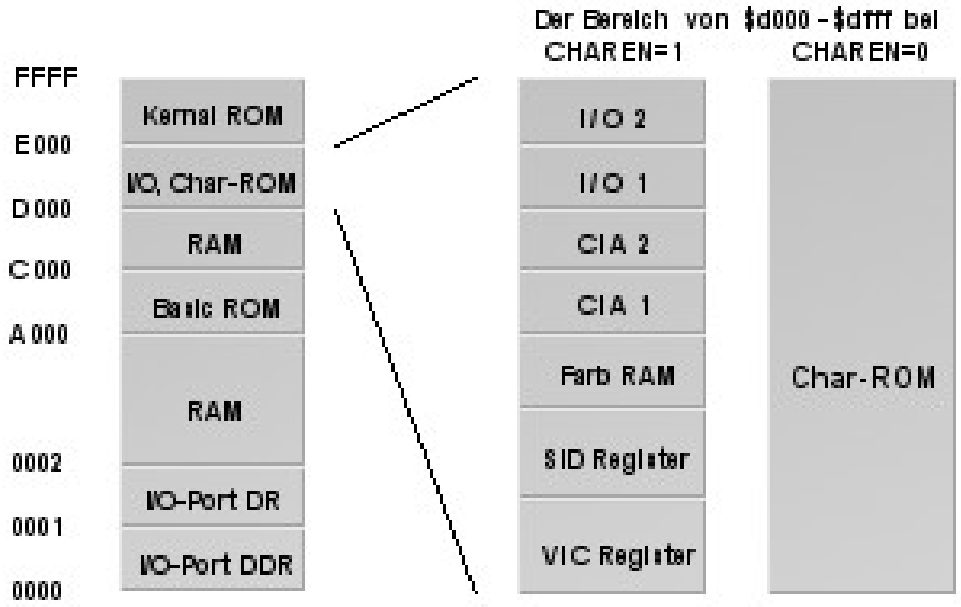

Abbildung 1: Speicherkarte aus Sicht des 6510

Die 64KB Hautpspeicher stehen im Prinzip linear zur Verfügung, aber sie sind an verschiedenen Stellen von ROM- und Register-Bereichen überlagert.
Ein Schreibzugriff auf einen ROM-Bereich speichert das Byte im "darunterliegenden" RAM. Bei Adresse $0000 und $0001 liegen Datenrichtungsregister und Datenregister des 6510-I/O-Ports.

Im Bereich von $d000-$dfff erscheinen wahlweise die Register der I/O-Chips und das Farb-RAM oder das Zeichengenerator-ROM, je nach Zustand des Signals CHAREN (dies ist Bit 2 des 6510-I/O-Ports). Das Farb-RAM ist im Adressbereich $d800-$dbff untergebracht und mit den unteren 4 Datenbits verbunden. Die oberen 4 Datenbits sind offen und liefern beim Lesen "zufällige" Werte. Die beiden mit "I/O 1" und "I/O 2" bezeichneten Bereiche sind für Erweiterungskarten reserviert und normalerweise ebenfalls offen, ein Lesezugriff liefert auch hier "zufällige" Daten (dass diese Daten gar nicht so zufällig sind, wird in Kapitel 4 noch ausführlich erklärt. Ein Lesen von offenen Adressen liefert nämlich auf vielen C64 das zuletzt vom VIC gelesene Byte zurück!).

Die 47 Register des VIC sind ab $d000 in den Adressraum eingeblendet.
Aufgrund der unvollständigen Dekodierung wiederholen sie sich alle 64 Bytes im Bereich von $d000-$d3ff.

2.4.2 Speicheraufteilung aus Sicht des VIC

Der VIC besitzt nur 14 Adressleitungen, kann also nur 16KB Speicher Adressieren. Er kann trotzdem auf die kompletten 64KB Hauptspeicher zugreifen, denn die 2 fehlenden oberen Adressbits werden von einem der CIA-I/O-Chips zur Verfügung gestellt (es sind dies die invertierten Bits 0 und 1 von Port A der CIA 2). Damit kann jeweils eine von 4 16KB-Bänken für den VIC eingestellt werden.

Die (erweiterte) Speicherlandschaft aus Sicht des VIC sieht so aus:

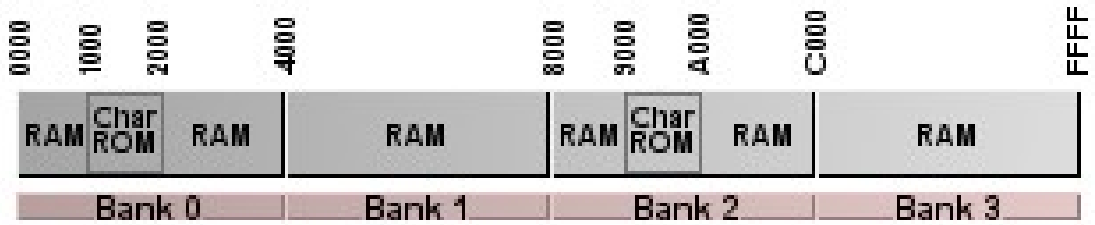

Abbildung 2: Speicherlandschaft aus Sicht des VIC

Das Char-ROM wird in den Bänken 0 und 2 jeweils an den VIC-Adressen $1000-$1fff eingeblendet (in Bank 2 erscheint es in dem Diagramm oben an Adresse $9000, aber der VIC weiß ja nichts von den beiden Adressbits, die noch für ihn erzeugt werden. Aus seiner Sicht ist das Char-ROM auch in Bank 2 an $1000-$1fff).

Der aufmerksame Leser wird bereits bemerkt haben, dass das Farb-RAM nirgendwo auftaucht, aber erinnern wir uns: Der VIC besitzt einen 12-Bit-Datenbus dessen obere 4 Bit mit dem Farb-RAM verbunden sind.
Überhaupt dienen die oberen 4 Bit des VIC-Datenbus nur dazu, das Farb-RAM auszulesen. Das Farb-RAM wird über die unteren 10 Bit des VIC-Adressbusses Adressiert und steht daher in allen Bänken an allen Adressen zur Verfügung.

2.4.3 Speicherzugriff von 6510 und VIC

6510 und VIC beruhen beide auf einem recht einfachen, festverdrahteten Design. Beide Chips führen in JEDEM Taktzyklus einen Speicherzugriff aus, auch wenn dies gar nicht notwendig ist. Wenn z.B. der Prozessor in einem Taktzyklus mit einer internen Operation wie der IndexAdressierung beschäftigt ist, die eigentlich keinen Zugriff erforden würde, wird trotzdem ein Lesezugriff ausgeführt, dessen Ergebnis verworfen wird. Der VIC greift nur lesend auf den Speicher zu, der 6510 lesend und schreibend.

Es gibt keine Waitstates, keine internen Caches und keine aufwendigen Zugriffsprotokolle auf den Bus, wie sie bei modernen Prozessoren üblich sind. Jeder Zugriff wird in einem einzigen Zyklus ausgeführt.

Der VIC erzeugt die Taktfrequenzen des Systembus und die RAS- und CAS-Signale für den Zugriff auf das dynamische RAM (auch für den Prozessor). Er hat daher die primäre Kontrolle über den Bus und kann den Prozessor gelegentlich "betäuben", wenn er zusätzliche Zyklen für Speicherzugriffe benötigt. Außerdem sorgt der VIC für den Refresh des DRAM, indem er in jeder Rasterzeile aus 5 Refresh-Adressen liest.

Die Aufteilung der Zugriffe zwischen 6510 und VIC ist zunächst einmal statisch: Jeder Taktzyklus (eine Periode des ø2-Signals) ist in zwei Phasen aufgeteilt. In der ersten Phase (ø2 Low) greift der VIC zu, in der zweiten Phase (ø2 High) der Prozessor. Das AEC-Signal schwingt parallel zu ø2 mit.

So könnten 6510 und VIC beide abwechselnd den Speicher benutzen, ohne sich gegenseitig zu stören.

Jedoch benötigt der VIC manchmal mehr Zyklen, als dieses Schema ihm zuteilen würde. Das ist dann der Fall, wenn der VIC auf die Zeichenzeiger und auf die Sprite-Daten zugreift. Im ersten Fall werden 40, im zweiten Fall pro Sprite 2 zusätzliche Zyklen benötigt. Dann geht BA 3 Zyklen bevor der VIC den Bus komplett übernimmt auf Low (3 Zyklen sind die maximale Anzahl aufeinanderfolgender Schreibzugriffe beim 6510) und nach 3 Zyklen bleibt AEC während der zweiten Taktphase Low, damit der VIC seine Adressen anlegen kann.

Das folgende Diagramm illustriert den Vorgang der Busübernahme:

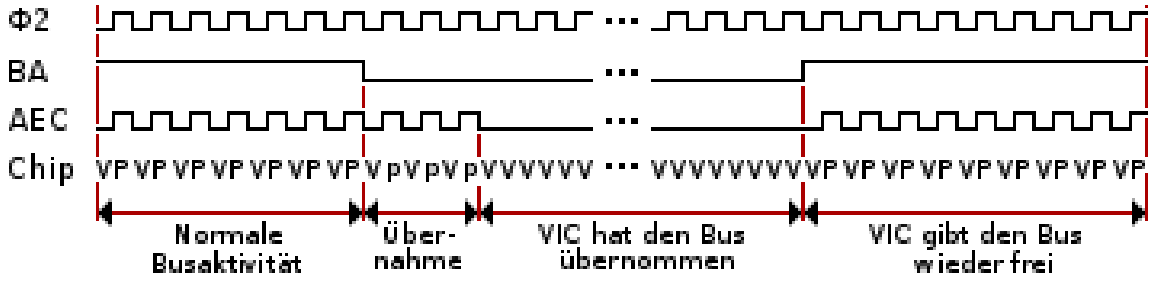

Abbildung 3: Vorgang der Busübernahme

Die Zeile "Chip" gibt an, welcher Baustein gerade auf den Bus zugreift (es findet ja, wie gesagt in jedem Zyklus ein Zugriff statt). "V" steht für den VIC, "P" für den 6510. Die mit "p" gekennzeichneten Zugriffe des Prozessors werden nur ausgeführt, wenn sie Schreibzugriffe sind. Der erste "p"-Lesezugriff hält den 6510 an, spätestens nach dem dritten "p", da ja beim 6510 höchstens 3 Schreibzugriffe aufeinanderfolgen können. Bei einem "p"-Lesezugriff werden aber trotzdem die ProzessorAdressen auf den Bus gegeben, denn AEC ist ja noch High.

Das Diagramm beschreibt den normalen Verlauf der Busübernahme. Durch geeignetes Verändern des VIC-Registers $d011 ist es möglich, eine Busübernahme zu außerordentlichen Zeitpunkten zu erzwingen. Dies sowie das komplette Bus-Timing einer VIC-Rasterzeile werden in Kapitel 3 erläutert.

3 Funktionsweise des VIC

Dieses Kapitel beschäftigt sich mit den einzelnen Funktionsgruppen im VIC, deren Arbeitsweise und Zusammenspiel und deren nicht spezifiziertem Verhalten und den Einblicken, die dadurch in die internen Abläufe im VIC gewonnen werden können.

3.1 Blockbild

Das folgende Blockschaltbild gibt eine Übersicht über den inneren Aufbau des VIC und die unabhängig voneinander arbeitenden Funktionsgruppen:

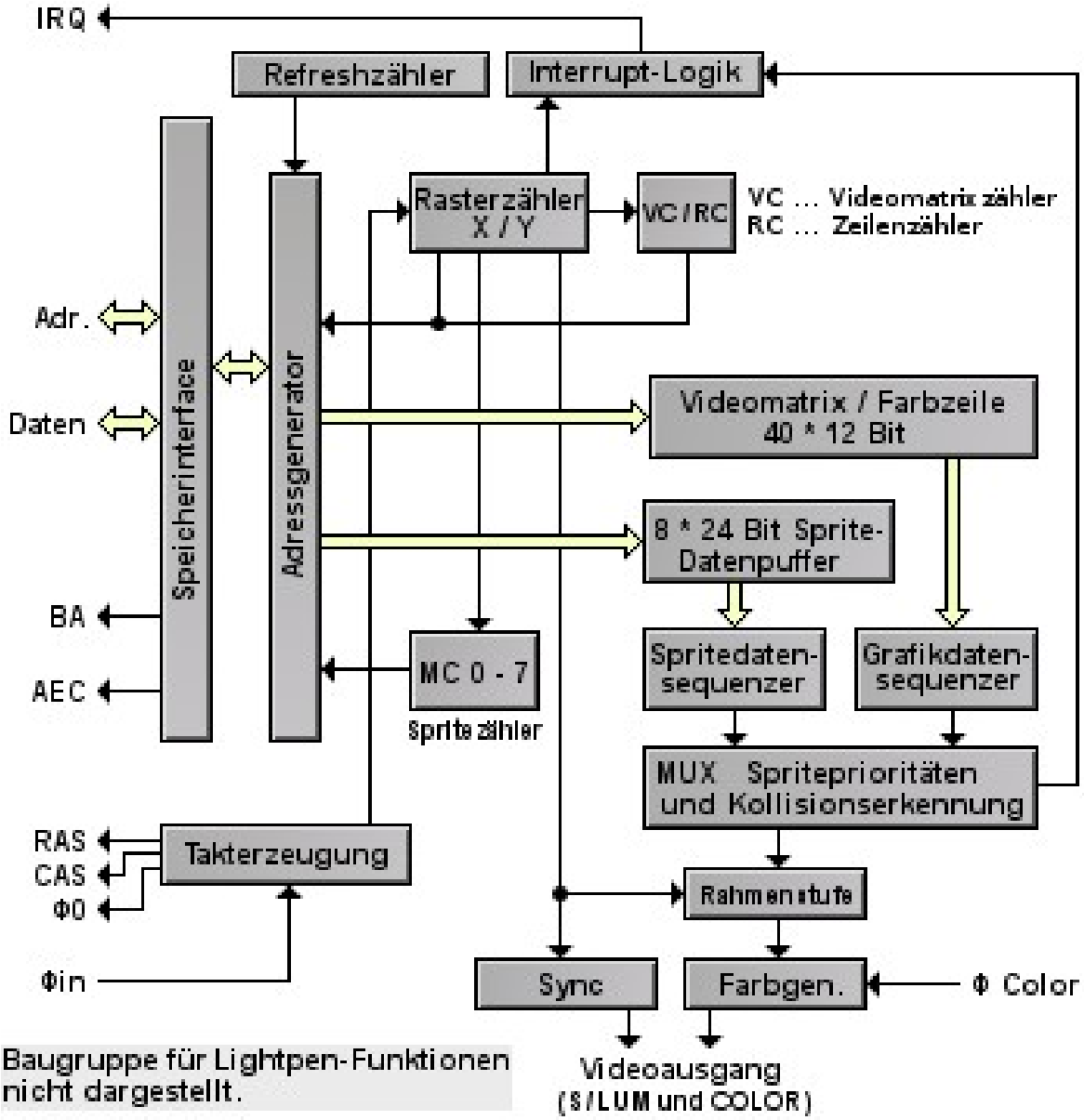

Abbildung 4: Übersicht über den inneren Aufbau des VIC und die unabhängig voneinander arbeitenden Funktionsgruppen

Die Baugruppe für die Lightpen-Funktionen ist nicht dargestellt.

Wie man sieht, kommt dem "Rasterzähler X/Y" eine zentrale Aufgabe zu. Logisch, denn sämtliche Darstellungen am Bildschirm und alle Buszugriffe orientieren sich darüber.

Wichtig ist, zu bemerken, dass sowohl bei den Sprites als auch bei der Grafik die Funktionen für die Darstellung und die dafür nötigen Speicherzugriffe getrennt voneinander sind. Zwischen den beiden steht jeweils ein Datenpuffer, der die gelesenen Grafikdaten aufnimmt und für die Verwendung durch die Schaltkreise zur Darstellung zwischenspeichert. Im normalen Betrieb des VIC ist die Funktion der beiden Kreise so aufeinander abgestimmt, dass sie wie ein einziger Funktionsblock wirken. Man kann jedoch durch geeignete Programmierung die Kreise voneinander entkoppeln und z.B. Grafik darstellen lassen, ohne dass vorher Daten gelesen wurden (in diesem Fall werden die noch im Puffer stehenden Daten angezeigt).

3.2 Register

Der VIC besitzt 47 Schreib-/Leseregister zur Steuerung seiner Funktionen durch den Prozessor:

Adr.	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Funktion
d000	M0X								X-Koordinate Sprite 0
d001	M0Y								Y-Koordinate Sprite 0
d002	M1X								X-Koordinate Sprite 1
d003	M1Y								Y-Koordinate Sprite 1
d004	M2X								X-Koordinate Sprite 2
d005	M2Y								Y-Koordinate Sprite 2
d006	M3X								X-Koordinate Sprite 3
d007	M3Y								Y-Koordinate Sprite 3
d008	M4X								X-Koordinate Sprite 4
d009	M4Y								Y-Koordinate Sprite 4
d00a	M5X								X-Koordinate Sprite 5
d00b	M5Y								Y-Koordinate Sprite 5
d00c	M6X								X-Koordinate Sprite 6
d00d	M6Y								Y-Koordinate Sprite 6
d00e	M7X								X-Koordinate Sprite 7
d00f	M7Y								Y-Koordinate Sprite 7
d010	M7X8	M6X8	M5X8	M4X8	M3X8	M2X8	M1X8	M0X8	MSBs der X-Koordinaten
d011	RST8	ECM	BMM	DEN	RSEL	YSCROLL			Steuerregister 1
d012	RASTER								Rasterzähler
d013	LPX								Lichtgriffel X
d014	LPY								Lichtgriffel Y
d015	M7E	M6E	M5E	M4E	M3E	M2E	M1E	M0E	Sprite angeschaltet
d016	–	–	RES	MCM	CSEL	XSCROLL			Steuerregister 2
d017	M7YE	M6YE	M5YE	M4YE	M3YE	M2YE	M1YE	M0YE	Sprite Y-Expansion
d018	VM13	VM12	VM11	VM10	CB13	CB12	CB11	–	Speicherzeiger
d019	IRQ	–	–	–	ILP	IMMC	IMBC	IRST	Interruptregister
d01a	–	–	–	–	ELP	EMMC	EMBC	ERST	Interrupt angeschaltet
d01b	M7DP	M6DP	M5DP	M4DP	M3DP	M2DP	M1DP	M0DP	Sprite-Daten-Priorität
d01c	M7MC	M6MC	M5MC	M4MC	M3MC	M2MC	M1MC	M0MC	Sprite Multicolor
d01d	M7XE	M6XE	M5XE	M4XE	M3XE	M2XE	M1XE	M0XE	Sprite X-Expansion
d01e	M7M	M6M	M5M	M4M	M3M	M2M	M1M	M0M	Sprite-Sprite-Kollision
d01f	M7D	M6D	M5D	M4D	M3D	M2D	M1D	M0D	Sprite-Daten-Kollision
d020	–	–	–	–	EC				Rahmenfarbe
d021	–	–	–	–	B0C				Hintergrundfarbe 0
d022	–	–	–	–	B1C				Hintergrundfarbe 1
d023	–	–	–	–	B2C				Hintergrundfarbe 2
d024	–	–	–	–	B3C				Hintergrundfarbe 3
d025	–	–	–	–	MM0				Sprite Multicolor 0
d026	–	–	–	–	MM1				Sprite Multicolor 1
d027	–	–	–	–	M0C				Farbe Sprite 0
d028	–	–	–	–	M1C				Farbe Sprite 1
d029	–	–	–	–	M2C				Farbe Sprite 2
d02a	–	–	–	–	M3C				Farbe Sprite 3
d02b	–	–	–	–	M4C				Farbe Sprite 4
d02c	–	–	–	–	M5C				Farbe Sprite 5
d02d	–	–	–	–	M6C				Farbe Sprite 6
d02e	–	–	–	–	M7C				Farbe Sprite 7
Adr.	**Bit7**	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**	**Funktion**

Tabelle 3: VIC Register

Hinweise:

- Die mit '-' gekennzeichneten Bits sind unbelegt. Beim Lesen liefern sie eine "1"
- Die VIC-Register wiederholen sich im Bereich $d000-$d3ff alle 64 Bytes, d.h. Register 0 ist an Adresse $d000, $d040, $d080 etc. verfügbar
- Die nicht belegten Adressen $d02f-$d03f liefern beim Lesen den Wert $ff, ein Schreibzugriff ist ohne Wirkung
- Die Register $d01e und $d01f sind nicht beschreibbar und werden bei einem Lesezugriff automatisch gelöscht
- Das RES-Bit (Bit 5) von Register $d016 ist bei den bisher untersuchten VIC 6567/6569 ohne Funktion. Beim 6566 dient dieses Bit dazu, den VIC zu stoppen.
- Bit 7 in Register $d011 (RST8) ist Bit 8 von Register $d012. Beide zusammen werden im folgenden mit "RASTER" bezeichnet. Ein Schreibzugriff in diese Bits legt die Vergleichszeile für den Rasterinterrupt fest (siehe Abschnitt 3.12.).

3.3 Farbpalette

Der VIC besitzt eine fest vorgegebene Palette von 16 Farben, die über 4 Bit kodiert sind:

0	schwarz
1	weiß
2	rot
3	cyan
4	lila
5	grün
6	blau
7	gelb
8	orange
9	braun
10	hellrot
11	dunkelgrau
12	mittelgrau
13	hellgrün
14	hellblau
15	hellgrau

Tabelle 4: Farben des VIC

3.4 Bildaufbau und Ausmaße des Bildausschnittes

Wie bei der Ansteuerung von Kathodenstrahlröhren üblich baut der VIC das Videobild zeilenweise auf. Die Zeilenzahl und die Anzahl Taktzyklen pro Zeile sind bei jedem VIC-Typ konstant. Der VIC arbeitet zeichenbasiert, jedes Zeichen besteht aus einer Matrix von 8×8 Pixeln, entsprechend eine Textzeile aus 8 Pixelzeilen. In den textbasierten Modi werden 40×25 Textzeichen dargestellt, in den Bitmap-Modi 320×200 oder 160×200 Pixel.

Die Angabe einer Position auf dem Bildschirm erfolgt in diesem Artikel durch die Nummer der Rasterzeile als Y-Koordinate (RASTER, Register $d011/$d012) und eine X-Koordinate entsprechend dem Sprite-Koordinatensystem. Bei der Angabe des Zeitpunktes eines VIC-Speicherzugriffs oder einer internen Operation im VIC werden die Rasterzeilennummer als Y-Koordinate und die Nummer des Taktzyklus innerhalb der Zeile als X-Koordinate benutzt. Wie bereits erwähnt, kommen auf einen Taktzyklus 8 Pixel, die Angabe einer Sprite-X-Koordinate ist also achtmal so genau wie die der Zyklusnummer.

Die Grafikdarstellung findet in einem nicht verschieblichen Fenster in der Mitte des sichtbaren Bildbereiches statt, dem "Anzeigefenster". Der Bereich außerhalb des Anzeigefensters bildet den Bildschirmrahmen und wird in der Rahmenfarbe (EC, Register $d020) dargestellt. Man kann den Rahmen mit einem Trick auch ganz oder teilweise ausschalten; dann erkennt man, dass das Anzeigefenster Teil einer "Anzeigespalte" ist, die sich aus der geradlinigen Erweiterung des Anzeigefensters nach oben und unten ergibt.
Damit kann man auch den Rahmen in einen oberen/unteren und einen linken/rechten Rahmen aufteilen. Der sichtbare Bildbereich ist horizontal und vertikal von Austastlücken umgeben, in denen die Videoausgabe abgeschaltet ist und in denen der Rasterstrahl an den Anfang der nächsten Zeile bzw. an den Anfang des Bildes zurückkehrt

Die folgende Abbildung (nicht maßstäblich) illustriert den letzten Absatz:

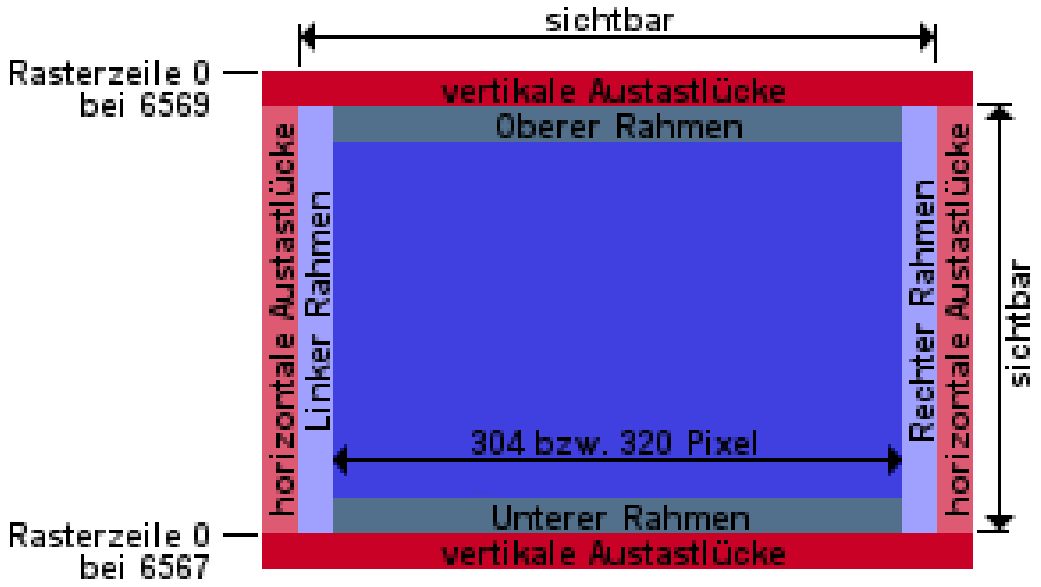

Abbildung 5: Layout des Anzeigefensters

Über die Bits RSEL und CSEL in den Registern $d011 und $d016 lassen sich für die Höhe und Breite des Anzeigefensters jeweils zwei verschiedene Werte einstellen:

RSEL	Höhe des Anzeigefensters	Erste Zeile	Letzte Zeile
0	24 Textzeilen/192 Pixel	55 ($37)	246 ($f6)
1	25 Textzeilen/200 Pixel	51 ($33)	250 ($fa)

Tabelle 5: RSEL Werte

CSEL	Breite des Anzeigefensters	Erste X-Koo.	Letzte X-Koo.
0	38 Zeichen/304 Pixel	31 ($1f	334 ($14e)
1	40 Zeichen/320 Pixel	24 ($18)	343 ($157)

Tabelle 6: CSEL Werte

Bei RSEL=0 werden der obere und untere Rahmen um jeweils 4 Pixel in das Anzeigefenster hinein verbreitert, bei CSEL=0 der linke Rahmen um 7 Pixel und der rechte um 9 Pixel. Die Position der Grafik im Anzeigefenster und deren Auflösung ändert sich dabei nicht, RSEL/CSEL schalten nur Start und Ende der Rahmendarstellung um. Auch die Größe der Videomatrix bleibt konstant bei 40×25 Zeichen.

Mit XSCROLL/YSCROLL (Bits 0-2 der Register $d011 (XSCROLL) und $d016 (YSCROLL)) kann man die Position der Grafik innerhalb des Anzeigefensters pixelweise um bis zu jeweils 7 Pixel nach rechts und unten verschieben. Damit lässt sich weiches Scrolling realisieren. Die Position des Anzeigefensters selbst ändert sich dabei nicht. Um die Grafik bündig mit dem Fenster zu halten, sind bei 25 Zeilen/40 Spalten als X/YSCROLL die Werte 0 und 3, bei 24 Zeilen/38 Spalten jeweils 7 zu wählen.

Die Ausmaße der Videodarstellung bei den verschiedenen VIC-Typen sind wie folgt:

Typ	Videonorm	Anzahl Zeilen	Sichtbare Zeilen	Zyklen/Zeile	Sichtbare Pixel/Zeile
6567R56A	NTSC-M	262	234	64	411
6567R8	NTSC-M	263	235	65	418
6569	PAL-B	312	284	63	403

Tabelle 7: Ausmaße der Videodarstellung 1

Typ	Erste V-Blank-Zeile	Letzte V-Blank-Zeile	Erste X-Koo. einer Zeile	Erste sichtbare X-Koordinate	Letzte sichtbare X-Koordinate
6567R56A	13	40	412 ($19c)	488 ($1e8)	388 ($184)
6567R8	13	40	412 ($19c)	489 ($1e9)	396 ($18c)
6569	300	15	404 ($194)	480 ($1e0)	380 ($17c)

Tabelle 8: Ausmaße der Videodarstellung 2

Wer sich bei den ersten und letzten sichtbaren X-Koordinaten wundert, dass scheinbar die erste Koordinate nach der letzten liegt: Dies kommt daher, dass als Referenzpunkt für den Beginn einer Rasterzeile das Auftreten des Raster-IRQ gewählt wurde, das aber nicht mit der X-Koordinate 0 zusammenfällt, sondern mit der unter "Erste X-Koo. einer Zeile" angegebenen. Die X-Koordinaten laufen innerhalb der Zeile bis $1ff (beim 6569 nur bis $1f7), dann erst kommt X-Koordinate 0. Bei der Erklärung des Aufbaus einer Rasterzeile wird dies noch genauer erklärt.

3.5 Bad Lines

Wie schon erwähnt, benötigt der VIC beim Zugriff auf die Zeichenzeiger (d.h. die Zeichencodes einer Textzeile aus der Videomatrix) 40 zusätzliche Buszyklen, denn die 63-65 dem VIC pro Rasterzeile für einen transparenten (vom Prozessor unbemerkten) Zugriff während der ersten Taktphase zur Verfügung stehenden Zyklen reichen nicht aus, um für die 40 Zeichen einer Zeile die Zeichenzeiger und auch noch die Pixeldaten der Zeichen aus dem Speicher zu lesen.

Aus diesem Grund benutzt der VIC während der ersten Pixelzeile jeder Textzeile den in Abschnitt 2.4.3. beschriebenen Mechanismus, um den Prozessor für 40-43 Zyklen zu "betäuben", um die Zeichenzeiger lesen zu können. Die Rasterzeilen, in denen dies geschieht, werden gewöhnlich "Bad Lines" genannt ("Bad", weil sie den Prozessor anhalten und den Rechner damit langsamer machen und zu Problemen führen, wenn es auf das genaue Timing eines Programms ankommt, wie bei der Datenübertragung zum Diskettenlaufwerk).

Der Zeichenzeigerzugriff findet auch in den Bitmap-Modi statt, denn dort werden die Daten der Videomatrix für die Farbinformation benutzt.

Normalerweise ist innerhalb des Anzeigefensters beginnend mit der ersten Zeile der Grafik jede achte Rasterzeile eine Bad Line, jeweils die ersten Rasterzeilen jeder Textzeile. Daher ist die Position der Bad Lines vom YSCROLL abängig. Wie sich noch zeigen wird, basiert der gesamte Grafikaufbau und das Zugriffsschema auf den Grafikspeicher auf der Position der Bad Lines.

Aus diesem Grund ist es notwendig, eine allgemeinere Definition einzuführen, nämlich die des "Bad-Line-Zustands":

Ein Bad-Line-Zustand liegt in einem beliebigen Taktzyklus vor, wenn an der negativen Flanke von ø0 zu Beginn des Zyklus RASTER >= $30 und RASTER <= $f7 und die unteren drei Bits von RASTER mit YSCROLL übereinstimmen und in einem beliebigen Zyklus von Rasterzeile $30 das DEN-Bit gesetzt war.

Diese Definition ist wörtlich zu nehmen. Man kann durch Verändern von YSCROLL mehrfach innerhalb einer beliebigen Rasterzeile im Bereich $30-$f7 einen Bad-Line-Zustand erzeugen und wieder wegnehmen und damit jede Rasterzeile innerhalb des Anzeigefensters ganz oder teilweise zur Bad Line machen oder die anderen Funktionen auslösen oder unterdrücken, die mit dem Auftreten des Bad-Line-Zustands zusammenhängen. Ist YSCOLL=0 tritt in Rasterzeile $30 ein Bad-Line-Zustand auf, sobald das DEN-Bit (Register $d011, Bit 4) gesetzt wird (für näheres über das DEN-Bit, siehe Abschnitt 3.10.).

Die folgenden drei Abschnitte beschreiben die Funktionsgruppen, die zur Darstellung der Grafik benutzt werden. In Abschnitt 3.6. geht es um das Speicherinterface, mit dem die Grafikdaten gelesen werden, und das Timing der Zugriffe innerhalb einer Rasterzeile. Abschnitt 3.7. handelt von der Anzeigestufe, die die Daten der Text- und Bitmap-Grafik in Farben umsetzt und die Adressen für den Speicherzugriff erzeugt, Abschnitt 3.8. behandelt die Sprites und ihre Adresserzeugung.

3.6 Speicherzugriff/Timing einer Rasterzeile

3.6.1 Die X-Koordinaten

Bevor das Timing der Speicherzugriffe innerhalb einer Rasterzeile erläutert wird, soll noch kurz darauf eingegangen werden, woher die angegebenen X-Koordinaten stammen. Der VIC besitzt nämlich zum RASTER-Register, das die aktuelle Y-Koordinate angibt, kein passendes Gegenstück für die X-Achse, man kann die X-Koordinate also nicht einfach direkt mit dem Prozessor auslesen. Jedoch zählt der VIC intern die X-Koordinaten sehr wohl mit, denn die horizontale Sprite-Positionierung basiert ja darauf und bei einem Impuls am Lichtgriffel-Eingang LP wird die aktuelle X-Position im Register LPX ($d013) gelatcht.

Die Bestimmung der absoluten X-Koordinaten von Ereignissen innerhalb einer Rasterzeile ist nicht trivial, denn man kann nicht einfach z.B. ein Sprite an eine definierte X-Koordinate bringen und aus den Textzeichen, die an der gleichen X-Position dargestellt werden, auf die X-Koordinate der zu diesen Textzeichen gehörenden Speicherzugriffe schließen. Der Speicherzugriff und die Darstellung sind getrennte Funktionsgruppen und die gelesenen Grafikdaten werden nicht direkt auf dem Bildschirm ausgegeben (es besteht eine Verzögerung von 12 Pixeln).

Daher wurde anders vorgegangen und eine einzige X-Koordinate mit Hilfe des LPX-Registers in ihrer absoluten Position innerhalb der Rasterzeile bestimmt und die anderen X-Koordinaten relativ dazu ermittelt. Dazu wurde der IRQ-Ausgang des VIC mit dem LP-Eingang verbunden und der VIC auf einen Rasterzeileninterrupt programmiert. Weil die negative Flanke von IRQ als Beginn einer Rasterzeile festgelegt wurde, konnte so die absolute X-Position des Rasterzeilenbeginns ermittelt werden. Die Position der negativen Flanke von BA während einer Bad Line wurde ebenfalls mit

dieser Methode bestimmt und die so erhaltene Position war mit dem relativen Abstand von IRQ und BA zueinander konsistent. Auf der Grundlage dieser beiden Messungen wurden die X-Koordinaten aller anderen Ereignisse innerhalb einer Rasterzeile relativ dazu bestimmt (siehe [4]). Jetzt erst wurden die Sprite-X-Koordinaten herangezogen, um den Zeitpunkt der Bilderzeugung der Textzeichen bestimmen zu können.

Dabei wird natürlich implizit angenommen, dass die LPX-Koordinaten mit den Sprite-X-Koordinaten übereinstimmen. Es gibt jedoch keinen Hinweis darauf und somit auch keinen Grund zur Annahme, dass sie es nicht täten (eine Übereinstimmung der Koordinaten wäre auch schaltungstechnisch die einfachste Lösung).

3.6.2 Zugriffsarten

Der VIC erzeugt zwei Arten von Grafik, die Zugriffe auf den Speicher erfordern: Die Text-/Bitmapgrafik (auch oft "Hintergrundgrafik" oder einfach "Grafik" genannt) und die Spritegrafik. Beide erfordern Zugriffe auf jeweils zwei getrennte Speicherbereiche.

Für die Text-/Bitmapgrafik:

- Die Videomatrix, ein 1000 VideoAdressen (40×25 mit je 12 Bit) großer Bereich, der sich mit den Bits VM10-VM13 aus Register $d018 in 1KB- Schritten innerhalb des 16KB-Adressraums des VIC verschieben lässt. Dort sind in den Textmodi die Zeichencodes und deren Farbe und in den Bitmap- Modi die Farbinformationen für 8×8-Pixel-Blocks abgelegt. Das Farb-RAM ist Teil der Videomatrix, es liefert die oberen 4 Bit der 12-Bit-Matrix.
  Die aus der Videomatrix gelesenen Daten werden in einem internen Puffer des VIC, der 40×12 Bit Videomatrix-/Farbzeile gespeichert.
- Den Zeichengenerator bzw. die Bitmap, ein 2048 Bytes (Bitmap: 8192 Bytes) großer Bereich, der sich mit den Bits CB11-CB13 (Bitmap: nur CB13) aus Register $d018 in 2KB-Schritten (Bitmap: 8KB-Schritte) innerhalb des VIC-Adressraums verschieben lässt. Dort sind in den Textmodi die Pixeldaten der Zeichen und in den Bitmap-Modi die Bitmap gespeichert. Der Zeichengenerator hat zunächst einmal nichts mit dem Char-ROM zu tun. Das Char-ROM enthält lediglich vorgefertigte Bitmuster, die als Zeichengenerator dienen können, aber man kann den Zeichengenerator auch im normalen RAM unterbringen um das Aussehen des Zeichensatzes zu ändern.

Für die Sprites:

- Die Spritedatenzeiger, 8 Bytes hinter dem Ende der Videomatrix, die für jedes Sprite einen von 256 64-Byte-Blocks innerhalb des VIC-Adressraums für die Spritedaten auswählen.
- Die Spritedaten, ein 63 Byte großer Bereich, der die Pixeldaten der Sprites enthält und für jedes Sprite einzeln mit Hilfe des Spritedatenzeigers in 64-Byte-Schritten verschoben werden kann.

Entsprechend macht der VIC 4 verschiedene Arten von Grafikzugriffen:

1. Auf die Videomatrix ("c-Zugriff", 12 Bit breit).
2. Auf die Pixeldaten, also Zeichengenerator oder Bitmap ("g-Zugriff", 8 Bit breit).
3. Auf die Spritedatenzeiger ("p-Zugriff", 8 Bit breit).
4. Auf die Spritedaten ("s-Zugriff", 8 Bit breit).

Darüber hinaus führt der VIC noch zwei weitere Arten von Zugriffen durch:

5. Zugriffe zum Refresh des dynamischen RAM, 5 Lesezugriffe pro Rasterzeile.
6. Idle-Zugriffe. Wie beschrieben, greift der VIC in jeder ersten Taktphase zu, obwohl es einige Zyklen gibt, in denen gerade kein anderer der oben beschriebenen Zugriffe ansteht. In diesem Fall führt der VIC einen Idle-Zugriff aus, einen Lesezugriff von VideoAdresse $3fff (also je nach Bank an Adresse $3fff, $7fff, $bfff oder $ffff), dessen Ergebnis verworfen wird.

3.6.3 Timing einer Rasterzeile

Die Abfolge der VIC-Speicherzugriffe innerhalb einer Rasterzeile ist fest vorgegeben, unabhängig vom Grafikmodus und für jede Rasterzeile gleich. Als Zeilenbeginn wurde die negative Flanke von IRQ bei einem Rasterinterrupt festgelegt (dies ist auch der Zeitpunkt, an dem das RASTER-Register erhöht wird). Rasterzeile 0 bildet allerdings eine Ausnahme: Dort finden IRQ und Erhöhen (bzw. Nullsetzen) von RASTER einen Zyklus später statt als in allen anderen Zeilen. Der Einfachheit halber wird hier jedoch von konstanten Zeilenlängen ausgegangen und der Beginn von Rasterzeile 0 als einen Zyklus vor dem Auftreten des IRQ definiert.

Hier zunächst die Timing-Diagramme, die Erklärung folgt danach:

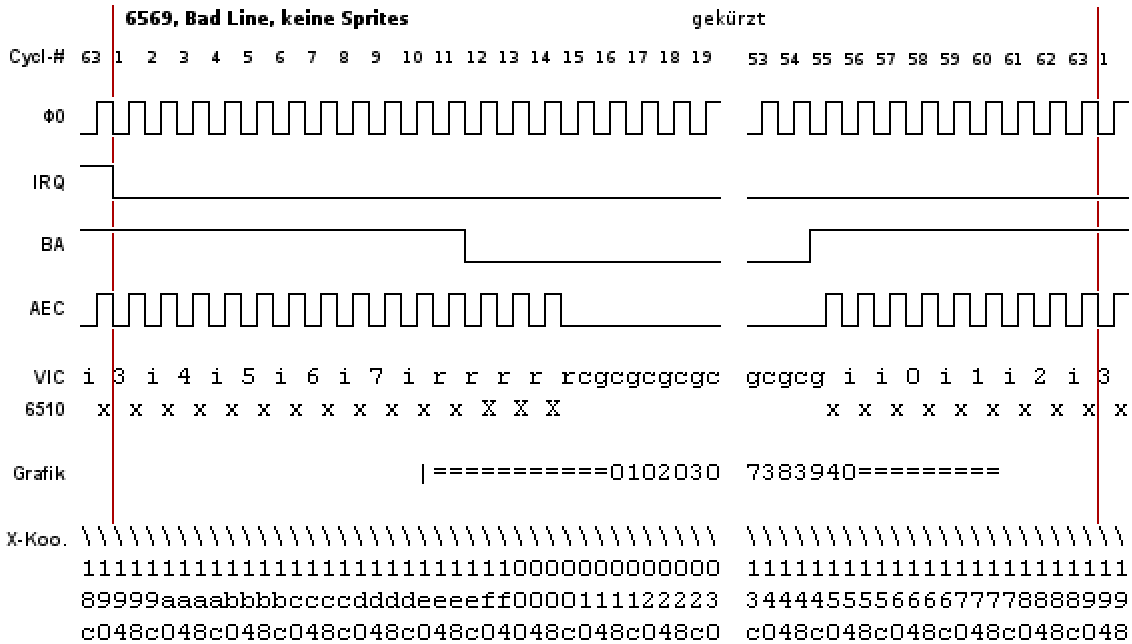

Abbildung 6: Timing einer Rasterzeile bei einer Bad Line

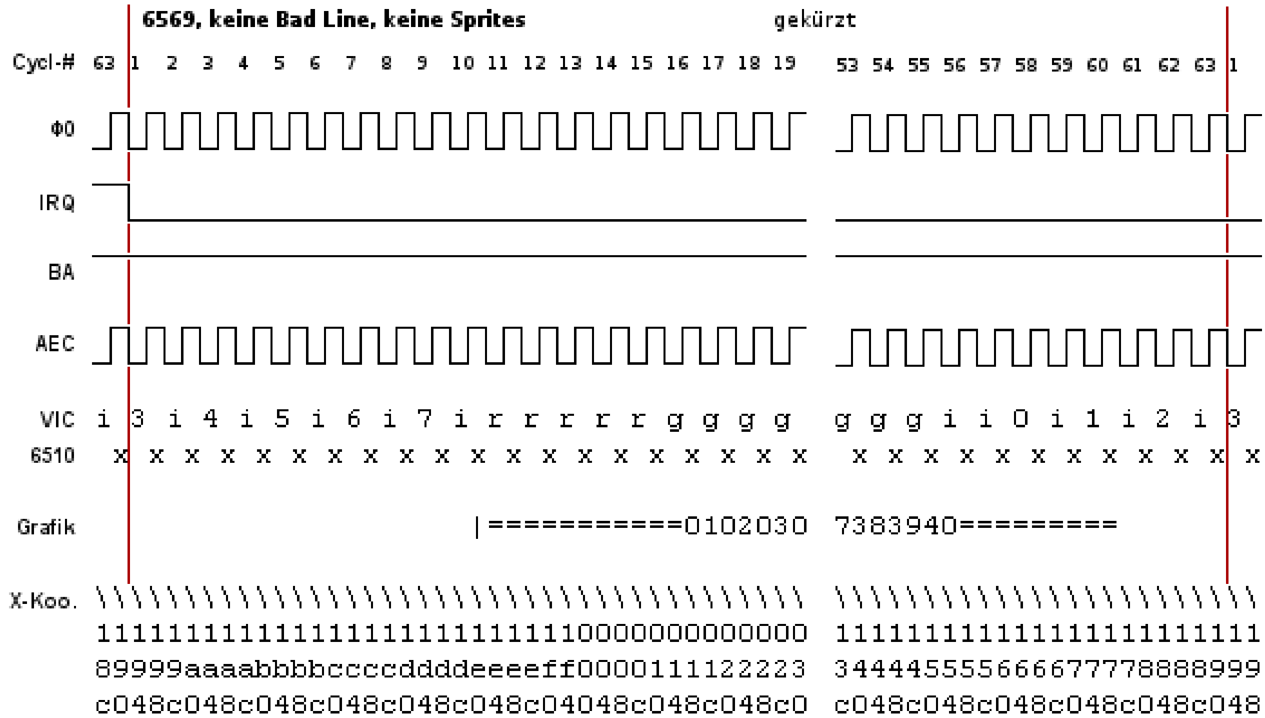

Abbildung 7: Timing einer Rasterzeile bei einer normalen Zeile (keine Bad Line)

In der Zeile "Zykl-#" ist die Nummer des Taktzyklus innerhalb der Rasterzeile aufgetragen. Die Zeile beginnt mit Zyklus 1 und besteht beim 6569 aus 63, beim 6567R56A aus 64 und beim 6567R8 aus 65 Zyklen. Zur Übersicht wurden noch der letzte Zyklus der vorangehenden und der erste Zyklus der nächsten Rasterzeile in jedes Diagramm mit aufgenommen.

Die Zeilen "ø0", "IRQ", "BA" und "AEC" geben den Zustand der gleichnamigen Bussignale wieder. In der ersten Phase jedes Zyklus ist ø0 Low, in der zweiten Phase High.

Die Symbole in den Zeilen "VIC" und "6510" geben an, welche Art von Zugriff VIC bzw. 6510 in der jeweiligen Taktphase durchführen (für eine Erklärung der verschiedenen Zugriffsarten des VIC siehe Abschnitt 3.6.2.):

c	Zugriff auf die Videomatrix und das Farb-RAM (c-Zugriff)
g	Zugriff auf den Zeichengenerator oder die Bitmap (g-Zugriff)
0-7	Lesen des Spritedatenzeigers für Sprite 0-7 (p-Zugriff)
s	Lesen der Spritedaten (s-Zugriff)
r	DRAM-Refresh
i	Idle-Zugriff
x	Schreib- oder Lesezugriff des Prozessors
X	Evtl. Schreibzugriff des Prozessors, der erste Lesezugriff stoppt den Prozessor (BA ist Low und damit auch RDY)

Tabelle 9: Symbolerklärung zum Timing einer Rasterzeile

Die Zeile "X-Koo." enthält die X-Koordinaten des Beginns jeder Taktphase (daher die "\\\" als Gedankenstütze) und die Zeile "Grafik" ist eine Projektion des 40-Spalten-Anzeigefensters und des Rahmens auf diese Koordinaten, zum Ausrichten von Sprites. Dies entspricht jedoch NICHT dem

Signal am Videoausgang des VIC. Aus der "Grafik"-Zeile kann man auch nicht ablesen, wann die Rahmenstufe den Rahmen erzeugt. Dies geschieht ca. 8 Pixel später als in der "Grafik"-Zeile angegeben.

Um beim Programmieren die Zugriffe des Prozessors innerhalb der Rasterzeile zeitlich bestimmen zu können, orientiert man sich am besten an den g-Zugriffen des VIC, indem man vom 6510 ein Byte im Grafikspeicher ändert und am Bildschirm beobachtet, an welchem Zeichen die Änderung frühestens wirksam wird. Der Schreibzugriff des Prozessors muß dann in Taktphase direkt davor erfolgt sein. Dann kann man mit Hilfe der Diagramme ermitteln, in welchem Taktzyklus der Zugriff stattfand und alle weiteren Zugriffe des Prozessors relativ dazu einfach abzählen.

3.7 Text-/Bitmapdarstellung

3.7.1 Idle-Zustand/Display-Zustand

Die Text-/Bitmap-Anzeigelogik im VIC befindet sich zu jedem Zeitpunkt in einem von zwei Zuständen: Dem Idle-Zustand oder dem Display-Zustand.

- Im Display-Zustand finden c- und g-Zugriffe statt, die Adressen und die Interpretation der Daten sind vom gewählten Anzeigemodus abhängig.

- Im Idle-Zustand finden nur g-Zugriffe statt. Der Zugriff erfolgt immer an VideoAdresse $3fff ($39ff bei gesetztem ECM-Bit in Register $d016).
  Die Grafik wird vom Sequencer genau wie im Display-Zustand dargestellt, wobei aber die Videomatrix-Daten als "0"-Bits angesehen werden.

Der Übergang vom Idle- in den Display-Zustand erfolgt, sobald ein Bad-Line-Zustand auftritt (siehe Abschnitt 3.5.). Der Übergang vom Display-in den Idle-Zustand erfolgt in Zyklus 58 einer Zeile, wenn der RC (siehe nächsten Abschnitt) den Wert 7 hat und kein Bad-Line-Zustand vorliegt.

Solange an Register $d011 innerhalb des Bildes keine Veränderungen vorgenommen werden, befindet sich die Anzeigelogik innerhalb des Anzeigefensters im Display-Zustand und außerhalb davon im Idle-Zustand.
Wenn man bei einem 25-Zeilen-Anzeigefenster einen anderen YSCROLL als 3 einstellt und in $3fff einen Wert ungleich Null ablegt, kann man am oberen bzw. unteren Rand des Fensters die Streifen sehen, die der Sequenzer im Idle-Zustand produziert.

In [4] werden sowohl die Idle-Zugriffe als auch die g-Zugriffe während des Idle-Zustand als "idle bus cycle" bezeichnet. Beide Phänomene haben jedoch nichts miteinander zu tun. Die in den Diagrammen in [4] mit "+" gekennzeichneten Zugriffe sind normale g-Zugriffe. Der Begriff "Idle-Zugriff" wird in diesem Artikel ausschließlich für die in den Diagrammen in Abschnitt 3.6.3. mit "i" gekennzeichneten Zugriffe benutzt, nicht jedoch für die g-Zugriffe während des Idle-Zustands.

3.7.2 VC und RC

Das vielleicht wichtigste Ergebnis der VIC-Untersuchungen ist die Entschlüsselung der Funktionsweise der internen Register "VC" und "RC" des VIC. Mit ihrer Hilfe erzeugt der VIC die Adressen für den Zugriff auf Videomatrix und Zeichengenerator/Bitmap.

Genaugenommen sind es drei Register:

VC	Video Counter	ein 10-Bit-Zähler, der mit dem Wert aus VCBASE geladen werden kann.
VCBASE	Video Counter Base	ein 10-Bit-Datenregister mit Löscheingang, das mit dem Wert

		aus VC geladen werden kann.
RC	Row Counter	ein 3-Bit-Zähler mit Löscheingang.

Tabelle 10: VC, VCBASE und RC Register

Außerdem gibt es noch einen 6-Bit-Zähler mit Löscheingang, der die Position innerhalb der internen 40×12 Bit Videomatrix-/Farbzeile angibt, an der gelesene Zeichenzeiger abgelegt bzw. von dort wieder gelesen werden, und den ich hier mit "VMLI" (Video Matrix Line Index) bezeichnen möchte.

Diese vier Register verhalten sich nach folgenden Regeln:

1. Irgendwo einmal außerhalb des Bereiches der Rasterzeilen $30-$f7 (also außerhalb des Bad-Line-Bereiches) wird VCBASE auf Null gesetzt.
   Vermutlich geschieht dies in Rasterzeile 0, der genaue Zeitpunkt ist nicht zu bestimmen, er spielt aber auch keine Rolle.
2. In der ersten Phase von Zyklus 14 jeder Zeile wird VC mit VCBASE geladen (VCBASE->VC) und VMLI gelöscht. Wenn zu diesem Zeitpunkt ein Bad-Line-Zustand vorliegt, wird zusätzlich RC auf Null gesetzt.
3. Liegt in den Zyklen 12-54 ein Bad-Line-Zustand vor, wird BA auf Low gelegt und die c-Zugriffe gestartet. Einmal gestartet, findet in der zweiten Phase jedes Taktzyklus im Bereich 15-54 ein c-Zugriff statt. Die gelesenen Daten werden in der Videomatrix-/Farbzeile an der durch VMLI angegebenen Position abgelegt. Bei jedem g-Zugriff im Display-Zustand werden diese Daten ebenfalls an der durch VMLI spezifizierten Position wieder intern gelesen.
4. Nach jedem g-Zugriff im Display-Zustand werden VC und VMLI erhöht.
5. In der ersten Phase von Zyklus 58 wird geprüft, ob RC=7 ist. Wenn ja, geht die Videologik in den Idle-Zustand und VCBASE wird mit VC geladen (VC->VCBASE). Ist die Videologik danach im Display-Zustand (liegt ein Bad-Line-Zustand vor, ist dies immer der Fall), wird RC erhöht.

Im Normalfall sorgen diese Regeln dafür, dass VC innerhalb des Anzeigefensters die 1000 Adressen der Videomatrix einmal durchzählt und RC innerhalb jeder Textzeile die 8 Pixelzeilen einer Textzeile. Das Verhalten von VC und RC wird maßgeblich vom (Nicht-)Auftreten des Bad-Line-Zustandes bestimmt und da man mit dem Prozessor über YSCROLL diesen beinflussen kann, kann man so auch VC und RC in bestimmten Grenzen steuern.

3.7.3 Grafikmodi

Der Grafikdatensequenzer beherrscht 8 verschiedene Grafikmodi, die über die Bits ECM, BMM und MCM (Extended Color Mode, Bit Map Mode und Multi Color Mode) in den Registern $d011 und $d016 ausgewählt werden (von den 8 möglichen Bitkombinationen sind 3 "ungültig" und erzeugen die gleiche Ausgabe, nämlich nur die Farbe Schwarz). Der Idle-Zustand ist ein Spezialfall, da darin keine c-Zugriffe stattfinden und der Sequenzer "0"-Bits als Videomatrix-Daten verwendet.

Der Sequenzer gibt die Grafikdaten in jeder Rasterzeile im Bereich der Anzeigespalte aus, sofern das vertikale Rahmenflipflop gelöscht ist (siehe Abschnitt 3.9.). Außerhalb der Anzeigespalte und bei gesetztem Flipflop wird die letzte aktuelle Hintergrundfarbe dargestellt (dieser Bereich ist normalerweise vom Rahmen überdeckt). Kernstück des Sequenzers ist ein 8-Bit-Schieberegister, das mit jedem Pixel um 1 Bit weitergeschoben und nach jedem g-Zugriff mit den gelesenen Pixeldaten geladen wird. Mit XSCROLL aus Register $d016 lässt sich das Laden des Schieberegisters um 0-7 Pixel verzögern und dadurch die Anzeige um bis zu 7 Pixel nach rechts verschieben.

Der Adressgenerator für die Text-/Bitmap-Zugriffe (c- und g-Zugriffe) besitzt bei den g-Zugriffen im wesentlichen 3 Modi (die c-Zugriffe erfolgen immer nach dem selben Adressschema). Im Display-Zustand wählt das BMM-Bit entweder Zeichengenerator-Zugriffe (BMM=0) oder Bitmap-Zugriffe (BMM=1) aus, im Idle-Zustand erfolgen die g-Zugriffe immer an VideoAdresse $3fff.
Bei gesetztem ECM-Bit schaltet der Adressgenerator bei den g-Zugriffen die Adressleitungen 9 und 10 immer auf Low, bei ansonsten gleichem Adressschema (z.B. erfolgen dann die g-Zugriffe im Idle-Zustand an Adresse $39ff).

Die 8 Grafikmodi werden nun einzeln behandelt und die erzeugten Adressen und die Interpretation der gelesenen Daten bei c- und g-Zugriffen beschrieben. Anschließend folgt noch eine Beschreibung der Besonderheiten des Idle-Zustands. Um das Nachschlagen zu erleichtern, wurden bei jedem Modus die Adressen explizit hingeschrieben, obwohl z.B. die c-Zugriffe immer gleich ablaufen.

3.7.3.1 Standard-Textmodus (ECM/BMM/MCM=0/0/0)

In diesem Modus (wie in allen Textmodi) liest der VIC aus der Videomatrix 8-Bit-Zeichenzeiger, die die Adresse der Punktmatrix des Zeichens im Zeichengenerator angibt. Damit ist ein Zeichensatz von 256 Zeichen verfügbar, die jeweils aus 8×8 Pixeln bestehen, die in 8 aufeinander folgenden Bytes im Zeichengenerator abgelegt sind. Mit den Bits VM10-13 und CB11-13 aus Register $d018 lassen sich Videomatrix und Zeichengenerator im Speicher verschieben.

Im Standard-Textmodus entspricht jedes Bit im Zeichengenerator direkt einem Pixel auf dem Bildschirm. Die Vordergrundfarbe ist für jedes Zeichen im Farbnybble aus der Videomatrix angegeben, die Hintergrundfarbe wird global durch Register $d021 festgelegt.

c-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Daten

11	10	9	8	7	6	5	4	3	2	1	0
Farbe von "1"-Pixeln				D7	D6	D5	D4	D3	D2	D1	D0

g-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CB13	CB12	CB11	D7	D6	D5	D4	D3	D2	D1	D0	RC2	RC1	RC0

Daten

7	6	5	4	3	2	1	0
8 Pixel (1 Bit/Pixel) „0”: Hintergrundfarbe 0 ($d021) „1”: Farbe aus Bits 8-11 der c-Daten							

3.7.3.2 Multicolor-Textmodus (ECM/BMM/MCM=0/0/1)

Dieser Modus ermöglicht es, auf Kosten der horizontalen Auflösung vierfarbige Zeichen darzustellen. Ist Bit 11 der c-Daten Null, wird das Zeichen wie im Standard-Textmodus dargestellt, wobei aber nur die Farben 0-7 für den Vordergrund zur Verfügung stehen. Ist Bit 11 gesetzt, bilden jeweils zwei horizontal benachbarte Bits der Punktmatrix ein Pixel. Dadurch ist die Auflösung des

Zeichens auf 4×8 reduziert (die Pixel sind doppelt so breit, die Gesamtbreite der Zeichen ändert sich also nicht).

Interessant ist, dass nicht nur die Bitkombination "00", sondern auch "01" für die Spritepriorität und -kollisionserkennung zum "Hintergrund" gezählt wird.

c-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Daten

11	10	9	8	7	6	5	4	3	2	1	0
MC-Flag	Farbe von „11“-Pixeln			D7	D6	D5	D4	D3	D2	D1	D0

g-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CB13	CB12	CB11	D7	D6	D5	D4	D3	D2	D1	D0	RC2	RC1	RC0

Daten

7	6	5	4	3	2	1	0	
8 Pixel (1 Bit/Pixel) "0": Hintergrundfarbe 0 ($d021) "1": Farbe aus Bits 8-10 der c-Daten								MC-Flag = 0
4 Pixel (2 Bit/Pixel) "00": Hintergrundfarbe 0 ($d021) "01": Hintergrundfarbe 1 ($d022) "10": Hintergrundfarbe 2 ($d023) "11": Farbe aus Bits 8-10 der c-Daten								MC-Flag = 1

3.7.3.3 Standard-Bitmap-Modus (ECM/BMM/MCM=0/1/0)

In diesem Modus (wie in allen Bitmap-Modi) liest der VIC die Grafikdaten aus einer 320×200-Bitmap, in der jedes Bit direkt einem Punkt auf dem Bildschirm entspricht. Die Daten aus der Videomatrix werden für die Farbinformation benutzt. Da die Videomatrix weiterhin nur eine 40×25-Matrix ist, können die Farben nur für Blöcke von 8×8 Pixeln individuell bestimmt werden (also eine Art YC-8:1-Format). Da die Entwickler des VIC-II den Bitmap-Modus mit sowenig zusätzlichem Schaltungsaufwand wie möglich realisieren wollten (der VIC-I hatte noch keinen Bitmap-Modus), ist die Bitmap etwas ungewöhnlich im Speicher abgelegt: Im Gegensatz zu modernen Videochips, die die Bitmap linear aus dem Speicher lesen, bilden beim VIC jeweils 8 aufeinanderfolgende Bytes einen 8×8-Pixelblock auf dem Bildschirm. Mit den Bits VM10-13 und CB13 aus Register $d018 lassen sich Videomatrix und Bitmap im Speicher verschieben.

Im Standard-Bitmap-Modus entspricht jedes Bit in der Bitmap direkt einem Pixel auf dem Bildschirm. Für jedem 8×8-Block können Vorder- und Hintergrundfarbe beliebig eingestellt werden.

c-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Daten

11	10	9	8	7	6	5	4	3	2	1	0
unbenutzt				Farbe von „1"-Pixeln				Farbe von „0“-Pixeln			

g-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CB13	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0	RC2	RC1	RC0

Daten

7	6	5	4	3	2	1	0
8 Pixel (1 Bit/Pixel) "0": Farbe aus Bits 0-3 der c-Daten "1": Farbe aus Bits 4-7 der c-Daten							

3.7.3.4 Multicolor-Bitmap-Modus (ECM/BMM/MCM=0/1/1)

Ähnlich wie beim Multicolor-Textmodus bilden auch in diesem Modus jeweils zwei benachbarte Bits ein (doppelt so breites) Pixel. Die Auflösung reduziert sich damit auf 160×200 Pixel.

Genau wie beim Multicolor-Textmodus wird die Bitkombination "01" für die Spritepriorität und -kollisionserkennung zum "Hintergrund" gezählt.

c-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Daten

11	10	9	8	7	6	5	4	3	2	1	0
Farbe von „11"-Pixeln				Farbe von „01"-Pixeln				Farbe von „10“-Pixeln			

g-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CB13	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0	RC2	RC1	RC0

Daten

7	6	5	4	3	2	1	0
4 Pixel (2 Bit/Pixel) „00”: Hintergrundfarbe 0 ($d021)							

„01”: Farbe aus Bits 4-7 der c-Daten „10”: Farbe aus Bits 0-3 der c-Daten „11”: Farbe aus Bits 8-11 der c-Daten

3.7.3.5 ECM-Textmodus (ECM/BMM/MCM=1/0/0)

Dieser Textmodus entspricht dem Standard-Textmodus, erlaubt es aber, für jedes einzelne Zeichen eine von vier Hintergrundfarben auszuwählen. Die Auswahl geschieht über die oberen beiden Bits des Zeichenzeigers. Dadurch reduziert sich der Zeichenvorrat allerdings von 256 auf 64 Zeichen.

c-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Daten

11	10	9	8	7	6	5	4	3	2	1	0
Farbe von „1"-Pixeln				Wahl des Hintergr.		D5	D4	D3	D2	D1	D0

g-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CB13	CB12	CB11	0	0	D5	D4	D3	D2	D1	D0	RC2	RC1	RC0

Daten

7	6	5	4	3	2	1	0
8 Pixel (1 Bit/Pixel) „0”: Je nach Bits 6/7 der c-Daten 00: Hintergrundfarbe 0 ($d021) 01: Hintergrundfarbe 1 ($d022) 10: Hintergrundfarbe 2 ($d023) 11: Hintergrundfarbe 3 ($d024) „1”: Farbe aus Bits 8-11 der c-Daten							

3.7.3.6 Ungültiger Textmodus (ECM/BMM/MCM=1/0/1)

Das gleichzeitige Setzen der ECM- und MCM-Bits wählt keinen der "offiziellen" Grafikmodi des VIC, sondern erzeugt nur schwarze Pixel.
Nichtsdestotrotz erzeugt der Grafikdatensequenzer auch in diesem Modus intern gültige Grafikdaten, die die Spritekollisionserkennung triggern können. Über den Umweg der Spritekollisionen kann man die erzeugten Daten auch auslesen (sehen kann man nichts, das Bild ist schwarz). Man kann so allerdings nur Vordergrund- und Hintergrundpixel unterscheiden, die Farbinformation lässt sich aus den Spritekollisionen nicht gewinnen.

Die erzeugte Grafik entspricht der des Multicolor-Textmodus, allerdings ist der Zeichenvorrat genau wie im ECM-Modus auf 64 Zeichen eingeschränkt.

c-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0

VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Daten

11	10	9	8	7	6	5	4	3	2	1	0
MC-Flag	unbenutzt					D5	D4	D3	D2	D1	D0

g-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CB13	CB12	CB11	0	0	D5	D4	D3	D2	D1	D0	RC2	RC1	RC0

Daten

7	6	5	4	3	2	1	0	
„0”: Schwarz (Hintergrund) „1”: Schwarz (Vordergrund)								MC-Flag=0: 8 Pixel (1 Bit/Pixel)
„00”: Schwarz (Hintergrund) „01”: Schwarz (Hintergrund) „10”: Schwarz (Vordergrund) „11”: Schwarz (Vordergrund)								MC-Flag=1: 4 Pixel (2 Bit/Pixel)

3.7.3.7 Ungültiger Bitmap-Modus 1 (ECM/BMM/MCM=1/1/0)

Dieser Modus erzeugt nur ebenfalls nur ein schwarzes Bild, die Pixel lassen sich allerdings auch hier mit dem Spritekollisionstrick auslesen.

Der Aufbau der Grafik ist im Prinzip wie im Standard-Bitmap-Modus, aber die Bits 9 und 10 der g-Adressen sind wegen dem gesetzten ECM-Bit immer Null, entsprechend besteht auch die Grafik - grob gesagt - aus vier "Abschnitten", die jeweils viermal wiederholt dargestellt werden.

c-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Daten

11	10	9	8	7	6	5	4	3	2	1	0
unbenutzt											

g-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CB13	VC9	VC8	0	0	VC5	VC4	VC3	VC2	VC1	VC0	RC2	RC1	RC0

Daten

7	6	5	4	3	2	1	0
8 Pixel (1 Bit/Pixel) „0”: Schwarz (Hintergrund) „1”: Schwarz (Vordergrund)							

3.7.3.8 Ungültiger Bitmap-Modus 2 (ECM/BMM/MCM=1/1/1)

Der letzte ungültige Modus liefert auch ein schwarzes Bild, das sich jedoch genauso mit Hilfe der Sprite-Grafik-Kollisionen "abtasten" lässt.

Der Aufbau der Grafik ist im Prinzip wie im Multicolor-Bitmap-Modus, aber die Bits 9 und 10 der g-Adressen sind wegen dem gesetzten ECM-Bit immer Null, was sich in der Darstellung genauso wie beim ersten ungültigen Bitmap-Modus wiederspiegelt. Die Bitkombination "01" wird wie gewohnt zum Hintergrund gezählt.

c-Zugriff
Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Daten

11	10	9	8	7	6	5	4	3	2	1	0
unbenutzt											

g-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CB13	VC9	VC8	0	0	VC5	VC4	VC3	VC2	VC1	VC0	RC2	RC1	RC0

Daten

7	6	5	4	3	2	1	0
4 Pixel (2 Bit/Pixel) „00”: Schwarz (Hintergrund) „01”: Schwarz (Hintergrund) „10”: Schwarz (Vordergrund) „11”: Schwarz (Vordergrund)							

3.7.3.9 Idle-Zustand

Im Idle-Zustand liest der VIC die Grafikdaten von Adresse $3fff (bzw. $39ff bei gesetztem ECM-Bit) und stellt sie im ausgewählten Grafikmodus dar, wobei aber die Videomatrix-Daten (normalerweise in den c-Zugriffen gelesen) nur aus "0"-Bits bestehen. Es wird also immer wiederholt das Byte an Adresse $3fff/$39ff ausgegeben.

c-Zugriff

Es werden keine c-Zugriffe ausgeführt.

Daten

11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0

g-Zugriff

Adressen (ECM=0)

13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Adressen (ECM=1)

13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	1	1	1	1	1	1	1	1	1

Daten

7	6	5	4	3	2	1	0	
8 Pixel (1 Bit/Pixel) „0”: Hintergrundfarbe 0 ($d021) „1”: Schwarz								Standard-Textmodus/ Multicolor-Textmodus/ ECM-Textmodus
8 Pixel (1 Bit/Pixel) „0”: Schwarz (Hintergrund) „1”: Schwarz (Vordergrund)								Standard-Bitmap-Modus/ Ungültiger Textmodus/ Ungültiger Bitmap-Modus 1
4 Pixel (2 Bit/Pixel) „00”: Hintergrundfarbe 0 ($d021) „01”: Schwarz (Hintergrund) „10”: Schwarz (Vordergrund) „11”: Schwarz (Vordergrund)								Multicolor-Bitmap-Modus
4 Pixel (2 Bit/Pixel) „00”: Schwarz (Hintergrund) „01”: Schwarz (Hintergrund) „10”: Schwarz (Vordergrund) „11”: Schwarz (Vordergrund)								Ungültiger Bitmap-Modus 2

3.8 Sprites

Zusätzlich zu der Text-/Bitmapgrafik kann der VIC acht unabhängige, 24×21 Pixel große, frei bewegliche Objekte darstellen, die "Sprites" (in [2] "MOBs" genannt (Movable Object Blocks)).

Die Sprites können beliebig auf dem Bildschirm positioniert werden, man kann sie mit den Bits in Register $d015 (MxE) einzeln ein- und ausschalten, sie mit Register $d017/$d01d in X- und/oder Y-Richtung um den Faktor 2 vergrößern (bei gleicher Auflösung von 24×21 Pixeln), mit Register $d01c (MxMC) den Standard- oder Multicolor-Darstellungmodus wählen, mit Register $d01b (MxDP) die Anzeigepriorität in Bezug zur Text-/Bitmapgrafik festlegen und jedem Sprite eine eigene Farbe zuordnen (Register $d027-$d02e).
Außerdem besitzt der VIC die Möglichkeit, Kollisionen zwischen Sprites untereinander oder zwischen Sprites und der Text-/Bitmapgrafik zu erkennen und ggf. einen Interrupt auszulösen (siehe 3.11.).

Die Position der linken oberen Ecke eines Sprites wird mit den zugehörigen Koordinatenregistern angegeben (MxX, MxY). Für die Y-Koordinate stehen 8 Bit, für die X-Koordinate 9 Bit zur Verfügung (die höchsten Bits aller Sprite-X-Koordinaten sind in Register $d010 gesammelt).

3.8.1 Speicherzugriff und Darstellung

Die zur Darstellung von 24×21 Pixeln notwendigen 63 Bytes an Spritedaten sind linear im Speicher abgelegt: Jeweil 3 aufeinanderfolgende Bytes bilden eine Zeile des Sprites.

Diese 63 Bytes lassen sich in 64-Byte-Schritten innerhalb des 16KB-Adressraumes des VIC verschieben. Dazu liest der VIC in jeder Rasterzeile für jedes Sprite aus den letzten 8 Bytes der Videomatrix einen Spritedatenzeiger (p-Zugriff), der bei den Spritedatenzugriffen (s-Zugriffe) die

oberen 8 Bits der Adresse bildet. Die unteren 6 Bits stammen aus einem Spritedatenzähler (MC0-MC7, für jedes Sprite einen), der für die Sprites eine ähnliche Aufgabe übernimmt wie der VC für die Videomatrix. Da die p-Zugriffe in jeder Rasterzeile stattfinden und nicht nur dann, wenn das jeweilige Sprite gerade dargestellt wird, kann man durch ändern des Spritedatenzeigers das Aussehen eines Sprites mitten innerhalb
der Darstellung ändern.

Wenn für ein Sprite s-Zugriffe notwendig sind, finden diese innerhalb der Rasterzeile in den drei Halbzyklen direkt nach dem zu dem jeweiligen Sprite gehörenden p-Zugriff statt. Der VIC benutzt dazu ebenfalls die BA- und AEC-Signale (wie in den Bad Lines), um in der zweiten Taktphase auf den Bus zugreifen zu können. Auch in diesem Fall geht BA drei Zyklen vor dem eigentlichen Zugriff auf Low. Die s-Zugriffe finden in jeder Rasterzeile statt, in der das Sprite sichtbar ist (bei den Sprites 0-2 jeweils in der Zeile davor, siehe die Timing-Diagramme in Abschnitt 3.6.3.), für jedes Sprite in festgelegten Zyklen innerhalb der Zeile.

Wie die Text- und Bitmap-Grafik, so besitzen auch die Sprites einen Standard-Modus und einen Multicolor-Modus. Im Standardmodus entspricht jedes Bit direkt einem Pixel auf dem Bildschirm. Ein "0" Pixel ist transparent und lässt die darunterliegende Grafik durchscheinen, ein "1" Pixel stellt die zum jeweiligen Sprite gehörende Spritefarbe aus den Registern $d027-$d02e dar. Im Multicolor-Modus bilden je zwei benachbarte Bits ein Pixel, wodurch die Auflösung des Sprites auf 12×21 sinkt (die Pixel werden doppelt so breit).

Außerdem lassen sich die Sprites in X- und Y-Richtung getrennt in ihrer Ausdehung auf dem Bildschirm verdoppeln (X-/Y-Expansion). Dabei wird jedes Sprite-Pixel einfach doppelt so breit und/oder hoch dargestellt, die Auflösung ändert sich nicht. Ein Pixel eines X-expandierten Multicolor-Sprites ist also insgesamt viermal so breit wie ein Pixel eines nicht-expandierten Standard-Sprites. Obwohl beide Expansionen ähnlich aussehen, sind sie doch auf vollkommen verschiedene Weise im VIC implementiert. Die X-Expansion weist einfach den Spritedatensequenzer an, die Pixel mit halber Frequenz auszugeben. Die Y-Expansion hingegen bewirkt, dass der Sprite-Adressgenerator in je zwei aufeinanderfolgenden Zeilen von denselben Adressen liest, so dass jede Spritezeile doppelt ausgegeben wird.

Zu jedem Sprite gehört ein eigener Spritedatensequenzer, dessen Kernstück ein 24-Bit-Schieberegister ist. Darüberhinaus gehören zu jedem Sprite noch zwei Register:

- "MC" (MOB Data Counter) ist ein 6-Bit-Zähler, der mit dem Wert aus MCBASE geladen werden kann.
- "MCBASE" (MOB Data Counter Base) ist ein 6-Bit-Zähler mit Löscheingang.

Außerdem gibt es pro Sprite noch ein Expansions-Flipflop, dass die Y-Expansion steuert.

Die Darstellung eines Sprites geschieht nach den folgenden Regeln (die Zyklusangaben gelten nur für den 6569):

1. Das Expansions-Flipflop ist gesetzt, solange das zum jeweiligen Sprite gehörende Bit MxYE in Register $d017 gelöscht ist.
2. In der ersten Phase von Zyklus 55 wird das Expansions-Flipflop invertiert, wenn das MxYE-Bit gesetzt ist.
3. In den ersten Phasen von Zyklus 55 und 56 wird für jedes Sprite geprüft, ob das entsprechende MxE-Bit in Register $d015 gesetzt und die Y-Koordinate des Sprites (ungerade Register $d001-$d00f) gleich den unteren 8 Bits von RASTER ist. Ist dies der Fall und der DMA für das Sprite noch ausgeschaltet, wird der DMA angeschaltet, MCBASE gelöscht und, wenn das MxYE-Bit gesetzt ist, das Expansions-Flipflop gelöscht.
4. In der ersten Phase von Zyklus 58 wird für jedes Sprite MC mit MCBASE geladen (MCBASE->MC) und geprüft, ob der DMA für das Sprite angeschaltet und die Y-

Koordinate des Sprites gleich den unteren 8 Bits von RASTER ist. Ist dies der Fall, wird die Darstellung des Sprites angeschaltet.

5. Ist der DMA für ein Sprite angeschaltet, finden in den entsprechenden, für jedes Sprite festgelegten Zyklen (siehe die Diagramme in Abschnitt 3.6.3.) drei aufeinanderfolgende s-Zugriffe statt (die p-Zugriffe finden immer statt, auch wenn das Sprite abgeschaltet ist). Die gelesenen Daten des ersten Zugriffes werden in den oberen 8 Bit des Schieberegisters gespeichert, die des zweiten in den mittleren 8 Bit und die des dritten in den unteren 8 Bit. Nach jedem s-Zugriff wird der MC um 1 erhöht.
6. Ist die Sprite-Darstellung für ein Sprite angeschaltet, wird, sobald die aktuelle X-Koordinate des Rasterstrahls mit der X-Koordinate des Sprites (gerade Register $d000-$d00e und $d010) übereinstimmt, mit jedem Pixel das Schieberegister um ein Bit nach links geschoben und das "herausgefallene" Bit dargestellt. Ist das zum Sprite gehörige MxXE-Bit aus Register $d01d gesetzt, wird nur jedes zweite Pixel geschoben und das Sprite erscheint doppelt so breit. Wenn das Sprite im Multicolor-Modus ist, werden jeweils zwei Bits zu einem Pixel zusammengefaßt.
7. In der ersten Phase von Zyklus 15 wird geprüft, ob das Expansions-Flipflop gesetzt ist. Wenn ja, wird MCBASE um 2 erhöht.
8. In der ersten Phase von Zyklus 16 wird geprüft, ob das Expansions-Flipflop gesetzt ist. Wenn ja, wird MCBASE um 1 erhöht. Dann wird geprüft, ob MCBASE auf 63 steht und bei positivem Vergleich der DMA und die Darstellung für das jeweilige Sprite abgeschaltet.

Dadurch, dass der Test in Punkt 3. gegen Ende der Rasterzeile gemacht wird, müssen die in den Registern angegebenen Sprite-Y-Koordinaten um 1 kleiner als die gewünschte Y-Position der ersten Sprite-Zeile sein, da die Darstellung der Sprites erst in der folgenden Zeile beginnt, nachdem die ersten Spritedaten gelesen wurden (sofern das Sprite nicht über die X-Koordinate $164 (Zyklus 58, siehe Punkt 4.) hinaus positioniert wird).

Sprites lassen sich vertikal "wiederverwenden": Wenn man während oder nach erfolgter Darstellung eines Sprites dessen Y-Koordinate auf eine spätere Rasterzeile setzt, so dass die in Punkt 1. und 2. genannten Vergleiche nochmal ansprechen, wird das Sprite ab dieser Y-Koordinate noch einmal dargestellt (wobei man natürlich X-Koordinate und den Spritedatenzeiger beliebig ändern kann). So ist es möglich, mehr als 8 Sprites auf dem Bildschirm erscheinen zu lassen.

Horizontal ist dies nicht möglich. Nach 24 dargestellten Pixeln ist das Schieberegister leergelaufen und auch wenn man die X-Koordinate innerhalb der Zeile so ändert, dass der Vergleich von Punkt 4. erneut anspricht, werden keine Spritedaten mehr angezeigt. Man kann also innerhalb einer Rasterzeile nur maximal 8 Sprites gleichzeitig darstellen.

Hier noch einmal das Schema der p- und s-Zugriffe im Überblick:

p-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
VM13	VM12	VM12	VM10	1	1	1	1	1	1	1	Spr.-Nummer		

Daten

7	6	5	4	3	2	1	0
MP7	MP6	MP5	MP4	MP3	MP2	MP1	MP0

s-Zugriff

Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
MP7	MP6	MP5	MP4	MP3	MP2	MP1	MP0	MC5	MC4	MC3	MC2	MC1	MC0

Daten

7	6	5	4	3	2	1	0	
„0”: Transparent „1”: Spritefarbe ($d027-$d02e)								MxMC=0: 8 Pixel (1 Bit/ Pixel)
„00”: Transparent „01”: Sprite Multicolor 0 ($d025) „10”: Spritefarbe ($d027-$d02e) „11”: Sprite Multicolor 1 ($d026)								MxMC=1: 4 Pixel (2 Bit/ Pixel)

3.8.2 Priorität und Kollisionserkennung

Sobald sich mehrere Grafikelemente (Sprites und Text-/Bitmapgrafik) auf dem Bildschirm überlappen, muss entschieden werden, welches Element im Vordergrund dargestellt werden soll. Dazu wird jedem Element eine Priorität zugeordnet und nur das Element mit der höchsten Priorität dargestellt.

Die Sprites haben untereinander eine feste Rangfolge: Sprite 0 hat die höchste und Sprite 7 die niedrigste Priorität. Wenn sich zwei Sprites überlappen, wird das Sprite mit der höheren Nummer nur dort dargestellt, wo das andere Sprite ein transparentes Pixel hat.

Die Priorität der Sprites zur Text-/Bitmapgrafik lässt sich in gewissen Grenzen regeln. Zunächst einmal muss man bei der Text-/Bitmapgrafik zwischen Vordergrund- und Hintergrundpixeln unterscheiden. Welche Bitkombinationen zum Vorder- oder Hintergrund gehören, entscheidet das MCM-Bit in Register $d016 unabhängig von Zustand des Grafikdatensequenzers und von den BMM- und ECM-Bits in Register $d011:

	MCM=0	MCM=1
Bit/Pixel	1	2
Pixel/Byte	8	4
Hintergrund	„0“	„00“, „01“
Vordergrund	„1“	„10“, „11“

Im Multicolor-Modus (MCM=1) gehören also die Bitkombinationen "00" und "01" zum Hintergrund und "10" und "11" zum Vordergrund, während im Standard-Modus (MCM=0) einfach gelöschte Pixel zum Hintergrund und gesetzte zum Vordergrund gehören. Es sollte noch bemerkt werden, dass dies auch für die im Idle-Zustand erzeugte Grafik gilt.

Mit den MxDP-Bits aus Register $d01b lässt sich nun für jedes Sprite getrennt angeben, ob es vor oder hinter den Vordergrundpixeln dargestellt wird (die Tabelle in [2] ist falsch):

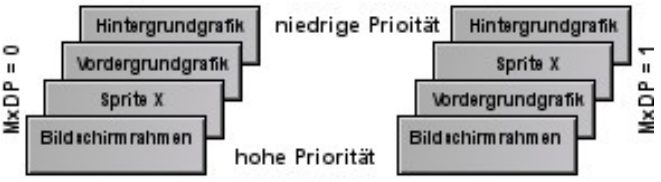

Abbildung 8: Spritepriorität

Auch hier werden die Grafikelemente, die eine niedrigere Priorität als ein darüberliegendes Sprite haben, nur an den Stellen sichtbar, wo das Sprite ein transparentes Pixel hat.

Wenn man einen der ungültigen Grafikmodi einstellt, sind nur die Sprites sichtbar (Vorder- und Hintergrundgrafik sind schwarz, siehe Abschnitte 3.7.3.6.-3.7.3.8.), aber schaltet man die Sprites so, dass sie hinter der Vordergrundgrafik erscheinen, sieht man die Vordergrundgrafik als schwarze Pixel über den Sprites.

Gleichzeitig mit der Prioritätsvergabe ist der VIC in der Lage, Kollisionen zwischen Sprites untereinander und zwischen Sprites und Vordergrundpixeln der Text-/Bitmapgrafik zu erkennen.

Eine Kollision zwischen Sprites untereinander wird erkannt, sobald beim Bildaufbau zwei oder mehrere Spritedatensequenzer gleichzeitig ein nicht-transparentes Pixel ausgeben (dies kann auch irgendwo außerhalb des sichtbaren Bildausschnittes geschehen). In diesem Fall werden im Register $d01e die MxM-Bits aller betroffenen Sprites gesetzt und (falls erlaubt, siehe Abschnitt 3.12.) ein Interrupt ausgelöst. Die Bits bleiben gesetzt, bis das Register vom Prozessor gelesen wird und werden durch den Lesezugriff automatisch gelöscht.

Eine Kollision zwischen Sprites und anderen Grafikdaten wird erkannt, sobald beim Bildaufbau ein oder mehrere Spritedatensequenzer ein nicht-transparentes Pixel und der Grafikdatensequenzer ein Vordergrundpixel ausgeben. In diesem Fall werden im Register $d01f die MxD-Bits der betroffenen Sprites gesetzt und (fall erlaubt, siehe Abschnitt 3.12.) ein Interrupt ausgelöst. Wie bei der Sprite-Sprite-Kollision bleiben die Bits gesetzt, bis das Register vom Prozessor ausgelesen wird.

Wenn das vertikale Rahmenflipflop gesetzt ist (also normalerweise innerhalb des oberen/unteren Rahmens, siehe nächsten Abschnitt), ist der Ausgang des Grafikdatensequenzers abgeschaltet und löst keine Kollisionen aus.

3.9 Die Rahmenstufe

Der VIC benutzt zwei Flipflops, um den Rahmen um das Anzeigefenster herum zu erzeugen: Ein Haupt-Rahmenflipflop und ein vertikales Rahmenflipflop.

Das Haupt-Rahmenflipflop steuert die Darstellung des Rahmens. Ist es gesetzt, stellt der VIC die in Register $d020 angegeben Farbe dar, sonst die Farbe, die der Prioritätsmultiplexer vom Grafik- oder Spritedatensequenzer durchschaltet. Der Rahmen überlagert also sowohl die Text-/Bitmapgrafik als auch die Sprites. Er hat die höchste Anzeigepriorität.

Das vertikale Rahmenflipflop dient zur Unterstützung bei der Darstellung des oberen/unteren Rahmens. Ist es gesetzt, kann das Haupt-Rahmenflipflop nicht gelöscht werden. Außerdem steuert das vertikale Rahmenflipflop den Ausgang des Grafikdatensequenzers. Dieser liefert nur bei gelöschtem Flipflop Daten, ansonsten stellt er die Hintergrundfarbe dar. Dies dient vermutlich dazu, im Rahmen Sprite-Grafik-Kollisionen zu verhindern.

Zu jedem der beiden Flipflops gehören 2×2 Komparatoren. Diese Komparatoren vergleichen die X-/Y-Position des Rasterstrahls mit einem von zwei festverdrahteten Werten (je nach Zustand der CSEL/RSEL-Bits) um die Flipflops zu steuern. Die Vergleiche fallen nur dann positiv aus, wenn der jeweilige Wert genau erreicht wird. Es findet kein Vergleich mit einem Intervall statt.

	CSEL=0	CSEL=1
Links	31 ($1f)	24 ($18)
Rechts	335 ($14f)	344 ($158)

Tabelle 11: Rahmenstufe: horizontale Vergleichswerte

	RSEL=0	RSEL=1

Oben	55 ($37)	51 ($33)
Unten	247 ($f7)	251 ($fb)

Tabelle 12: Rahmenstufe: vertikale Vergleichswerte

Die Flipflops werden nach den folgenden Regeln geschaltet:

1. Erreicht die X-Koordinate den rechten Vergleichswert, wird das Haupt-Rahmenflipflop gesetzt.
2. Erreicht die Y-Koordinate den unteren Vergleichswert in Zyklus 63, wird das vertikale Rahmenflipflop gesetzt.
3. Erreicht die Y-Koordinate den oberern Vergleichswert in Zyklus 63 und ist das DEN-Bit in Register $d011 gesetzt, wird das vertikale Rahmenflipflop gelöscht.
4. Erreicht die X-Koordinate den linken Vergleichswert und die Y-Koordinate den unteren, wird das vertikale Rahmenflipflop gesetzt.
5. Erreicht die X-Koordinate den linken Vergleichswert und die Y-Koordinate den oberen und ist das DEN-Bit in Register $d011 gesetzt, wird das vertikale Rahmenflipflop gelöscht.
6. Erreicht die X-Koordinate den linken Vergleichswert und ist das vertikale Rahmenflipflop gelöscht, wird das Haupt-Flipflop gelöscht.

Die Y-Koordinate wird also ein- oder zweimal innerhalb jeder Rasterzeile geprüft: In Zyklus 63 und wenn die X-Koordinate den linken Vergleichswert erreicht.

Man kann durch geeignetes Umschalten der CSEL/RSEL-Bits verhindern, dass einer oder mehrere der Vergleichswerte erreicht werden und damit den Rahmen ganz oder teilweise abschalten (siehe 3.14.1.).

3.10 Display Enable

Das DEN-Bit (Display Enable, Register $d011, Bit 4) dient dazu, die Text-/Bitmapgrafik ein- oder auszuschalten. Im normalen Betrieb ist es gesetzt. Das Bit hat Auswirkungen auf zwei Funktionen des VIC: Die Bad Lines und die vertikale Rahmenstufe:

- Ein Bad-Line-Zustand kann nur auftreten, wenn irgendwann in Rasterzeile $30 das DEN-Bit wenigstens einen Zyklus lang gesetzt war (siehe Abschnitt 3.5.).
- Ist das DEN-Bit gelöscht, wird der Reset-Eingang des vertikalen Rahmenflipflops deaktiviert (siehe Abschnitt 3.9.). Der obere/untere Rahmen wird dann nicht abgeschaltet.

Normalerweise bewirkt das Löschen des DEN-Bits also, dass keine Bad Lines (und damit auch keine c- und g-Zugriffe) auftreten und der ganze Bildschirm die Rahmenfarbe annimt.

3.11 Lightpen

Bei einer negativen Flanke am LP-Eingang wird die aktuelle Position des Rasterstrahls in den Registern LPX ($d013) und LPY ($d014) gelatcht. LPX enthält die oberen 8 Bit (von 9) der X-Position und LPY die unteren 8 Bit (ebenfalls von 9) der Y-Position. Daher ist die horizontale Auflösung des Lightpens auf 2 Pixel begrenzt.

Es wird pro Bild nur eine einzige negative Flanke an LP registriert. Treten mehrere Flanken an LP auf, werden alle nachfolgenden ignoriert. Erst in der nächsten vertikalen Austastlücke wird die Triggerung wieder freigegeben.

Da der LP-Eingang des VIC wie alle Leitungen der Joystick-Ports an die Tastaturmatrix angeschlossen ist, lässt er sich auch per Software steuern.

Dazu dient Bit 4 von Port B von CIA-A ($dc01/$dc03). Dies gibt die Möglichkeit, die aktuelle X-Position des Rasterstrahls durch Auslösen einer LP-Flanke und anschließendem Lesen von LPX zu ermitteln (der VIC besitzt kein Register, aus dem man die X-Position direkt auslesen könnte). Damit kann man z.B. Rasterinterrupt-Routinen zyklusgenau synchronisieren.

Die Werte, die man im LPX-Register enthält kann man aus den Sprite-Koordinaten der Timing-Diagramme in Abschnitt 3.6.3. ermitteln. Man muss sich jeweils am Ende des Zyklus, in dem die LP-Leitung gesetzt wird, orientieren. Triggert man LP z.B. in Zyklus 20, so erhält man in LPX den Wert $1e, der der Sprite-Koordinate $03c entspricht (LPX enthält die oberen 8 Bit der 9-Bit-X-Koordinate).

Der VIC kann auch zusätzlich einen Interrupt beim Auftreten einer negativen Flanke am LP-Pin auslösen (siehe nächsten Abschnitt), ebenfalls nur einmal pro Strahldurchlauf.

3.12 VIC-Interrupts

Der VIC besitzt die Möglichkeit, beim Auftreten bestimmter Ereignisse einen Interrupt für den Prozessor zu erzeugen. Dazu dient der IRQ-Ausgang, der direkt mit dem IRQ-Eingang des 6510 verbunden ist. Die VIC-Interrupts sind daher mit dem I-Flag im Statusregister des Prozessors maskierbar.

Im VIC gibt es vier Interruptquellen. Zu jeder Quelle gehört ein Bit im Interrupt-Latch (Register $d019) und ein Bit im Interrupt-Enable-Register (Register $d01a). Beim Auftreten eines Interrupts wird das entsprechende Bit im Latch gesetzt. Um es zu löschen, muss der Prozessor dort "von Hand" eine "1" hineinschreiben, der VIC löscht das Latch nicht selbsttätig.

Wenn mindestens ein Latch-Bit und das zugehörige Bit im Enable-Register gesetzt sind, wird die IRQ-Leitung auf Low gelegt und damit der Interrupt im Prozessor ausgelöst. Mit den Enable-Bits lassen sich also die vier Interruptquellen getrennt ein- und ausschalten. Da der VIC - wie beschrieben - das Latch nicht selbst löscht, muss dies der Prozessor tun, bevor er das I-Flag löscht bzw. aus dem Interrupt zurückkehrt, denn sonst wird sofort wieder ein Interrupt ausgelöst (der IRQ-Eingang des 6510 ist pegelgetriggert).

Die folgende Tabelle beschreibt die vier Interruptquellen und ihre Bits in den Latch- und Enable-Registern:

Bit	Name	Auslöser
0	RST	Erreichen einer bestimmten Rasterzeile. Die Zeile wird durch Schreiben in Register $d012 und $d011, Bit 7 festgelegt und vom VIC intern für den Vergleich gespeichert. Der Test auf das Erreichen der Interrupt-Rasterzeile findet in jeder Rasterzeile in Zyklus 0 (in Zeile 0 in Zyklus 1) statt.
1	MBC	Kollision von mindestens einem Sprite mit der Text-/Bitmapgrafik (ein Spritedatensequenzer liefert zum selben Zeitpunkt ein nicht-transparentes Pixel an dem der Grafikdatensequenzer ein Vordergrundpixel ausgibt)
2	MMC	Kollision von zwei oder mehr Sprites (zwei Spritedatensequenzer liefern gleichzeitig nicht-transparente Pixel)
3	LP	Negative Flanke am LP-Eingang (Lightpen)

Tabelle 13: Interruptquellen

Bei den MBC- und MMC-Interrupts löst jeweils nur die erste Kollision einen Interrupt aus (d.h. wenn die Kollisionsregister $d01e bzw. $d01f vor der Kollision den Inhalt Null hatten). Um nach einer Kollision weitere Interrupts auszulösen, muss das betreffende Register erst durch Auslesen gelöscht werden.

Das Bit 7 im Latch $d019 gibt den invertierten Zustand des IRQ-Ausgangs des VIC wieder.

3.13. DRAM-Refresh

In jeder Rasterzeile führt der VIC fünf Lesezugriffe zum Refresh des dynamischen RAM durch. Es wird ein 8-Bit Refreshzähler (REF) zur Erzeugung von 256 DRAM-ZeilenAdressen benutzt. Der Zähler wird in Rasterzeile 0 mit $ff gelöscht und nach jedem Refresh-Zugriff um 1 verringert.

Der VIC greift also in Zeile 0 auf die Adressen $3fff, $3ffe, $3ffd, $3ffc und $3ffb zu, in Zeile 1 auf $3ffa, $3ff9, $3ff8, $3ff7 und $3ff6 usw.

Refresh-Adressen

13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	Ref7	Ref6	Ref5	Ref4	Ref3	Ref2	Ref1	Ref0

3.14 Effekte/Anwendungen

In den folgenden Abschnitten sollen einige grafische Effekte beschrieben werden, die sich durch die Anwendung der in den Abschnitten zuvor beschriebenen Funktionsweise des VIC erzielen lassen.

3.14.1 Hyperscreen

Wie in Abschnitt 3.9. erläutert, erzeugt der VIC den Bildschirmrahmen durch das Vergleichen der Strahlkoordinaten mit Start- und Stoppositionen, die sich durch die CSEL/RSEL-Bits auswählen lassen. Der Rahmen wird also nicht grundsätzlich innerhalb eines bestimmten Koordinatenbereichs dargestellt, sondern bei bestimmten Koordinaten ein- und ausgeschaltet. Sorgt man nun durch geeignetes Umschalten von CSEL/RSEL dafür, dass der Koordinatenvergleich niemals anspricht, so wird der Rahmen z.B. nicht eingeschaltet und man kann auch die Grafik im Randbereich sehen, die normalerweise durch den Rahmen überdeckt wird. Diese Technik wird als "Hyperscreen" oder "Öffnen des Rahmens" bezeichnet.

Allerdings beschränkt sich die im Randbereich darstellbare Grafik hauptsächlich auf Sprites, da der Grafikdatensequenzer in diesem Bereich im Idle-Zustand ist, da außerhalb der Y-Koordinaten $30-$f7 keine Bad Lines auftreten können (siehe Abschnitt 3.5.). Man kann aber auch mit der im Idle-Zustand erzeugten Grafik etwas sinnvolles darstellen.

Um den oberen/unteren Rahmen auszuschalten, geht man wie folgt vor:

1. Irgendwo in der oberen Hälte des Bildschirms setzt man das RSEL-Bit und schaltet damit auf den 25-Zeilen-Rahmen.
2. Nun wartet man, bis RASTER einen Wert im Bereich 248-250 erreicht hat.
   Das vertikale Rahmenflipflop ist noch gelöscht, denn bei RSEL=1 spricht der Komparator erst in Rasterzeile 251 an.
3. Jetzt löscht man das RSEL-Bit. Der Komparator wird umgeschaltet und setzt das vertikale Flipflop nun bei Zeile 247. Diese Zeile ist aber schon vorbei und der VIC "vergisst" deshalb, den vertikalen Rahmen einzuschalten.
4. Nach Rasterzeile 251 setzt man das RSEL-Bit wieder und wiederholt das Ganze bei Punkt 2.

Wenn man den oberen/unteren Rahmen mit dieser Methode öffnet, bleibt aber weiterhin der linke/rechte Rahmen in dem "freigewordenen" Bereich aktiv.
Schaltet man im Bereich der Rasterzeilen 52-54 von RSEL=0 auf RSEL=1, so wird der Rahmen nicht mehr ausgeschaltet und überdeckt den gesamten Bildschirm (dies entspricht der Darstellung

bei gelöschtem DEN-Bits, jedoch mit dem Unterschied, dass Bad Lines stattfinden), was aber wenig sinnvoll ist.

Der linke/rechte Rahmen lässt sich mit dem CSEL-Bit auf eine ähnliche Weise abschalten. Das Timing ist jedoch wesentlich kritischer. Hat man beim vertikalen Rahmen noch 4 Rasterzeilen Zeit für die Umschaltung, so muss beim horizontalen Rahmen der Wechsel CSEL=1 nach CSEL=0 auf den Zyklus genau erfolgen, und zwar in Zyklus 56. Ganz analog lässt sich das Abschalten des horizontalen Rahmens verhindern, indem man in Zyklus 17 von CSEL=0 auf CSEL=1 schaltet.

Will man den linken/rechten Rahmen im oberen/unteren Randbereich öffnen, so muss man entweder damit anfangen, bevor das vertikale Rahmenflipflop gesetzt wird, also noch außerhalb des oberen/unteren Rahmens, oder den oberen/unteren Rahmen ebenfalls öffnen, denn das Haupt-Rahmenflipflop kann nur gelöscht werden, wenn auch das vertikale Flipflop gelöscht ist.
Vergleicht man beide Methoden, kann man sich auch davon überzeugen, dass das vertikale Flipflop den Ausgang des Grafikdatensequenzers steuert: Bei der ersten Methode ist im freigewordenen oberen/unteren Rahmenbereich nur die Hintergrundfarbe sichtbar, bei der zweiten Methode wird die Grafik des Idle-Zustands dargestellt.

3.14.2 FLD

Beim Aufbau der Grafik nach Textzeilen orientiert sich der VIC ausschließlich am Auftreten der Bad Lines: Eine Bad Line gibt das "Startsignal" für die Darstellung einer Textzeile. Durch geeignetes Ändern von YSCROLL (in Register $d011) kann man den Bad-Line-Zustand unterdrücken (siehe 3.5.) und beliebig verzögern. Man kann so genau steuern, in welchen Rasterzeilen Bad Lines auftreten sollen und damit, ab welchen Rasterzeilen der VIC jeweils eine Textzeile darstellen soll. So lässt sich der Abstand zwischen zwei Textzeilen beliebig vergrößern, wenn man die nächste Bad Line nur lange genug zurückhält. Dieser Effekt wird als "Flexible Line Distance" (FLD) bezeichnet.

Lässt man z.B. im ganzen Bildschirm nur drei Bad Lines bei den Rasterzeilen $50, $78 und $a0 zu, so stellt der VIC auch nur drei Textzeilen jeweils an diesen Positionen dar. Dazwischen ist der Sequenzer im Idle-Zustand.

Verzögert man nur das Auftreten der ersten Bad Line, kann man die komplette Grafikdarstellung um große Distanzen nach unten scrollen, ohne auch nur ein Byte im Grafikspeicher zu verschieben.

3.14.3 FLI

Anstatt wie beim FLD-Effekt das Auftreten der Bad Lines zu verzögern, kann man auch künstlich zusätzliche Bad Lines erzeugen, bevor der VIC die aktuelle Textzeile beendet hat. Besonders interessant ist dies in den Bitmap-Modi, denn dort werden die Daten aus der Videomatrix (die ja in den Bad Lines gelesen werden) für die Farbinformation benötigt, weshalb in den Bitmap-Modi normalerweise nur einzelne 8×8-Pixelblöcke individuell eingefärbt werden können. Macht man jedoch durch geeignetes Ändern von YSCROLL jede Rasterzeile zur Bad Line, liest der VIC in jeder Rasterzeile aus der Videomatrix und holt damit auch für jede Rasterzeile neue Farbinformationen.

Dadurch ist man nun in der Lage, innerhalb der 4×8 Pixel eines Blocks im Multicolor-Modus jedes Pixel einzeln einzufärben. Dieser durch Software erzeugte, neue Grafikmodus wird als "Flexible Line Interpretation" (FLI) bezeichnet und stellt vermutlich das herausragendste Beispiel "unkonventioneller" VIC-Programmierung dar.

Ein Problem gibt es allerdings: Wenn man eine neue Bad Line erzeugt, noch bevor die aktuelle Textzeile beendet wurde, wird VCBASE nicht erhöht (siehe 3.7.2.). Der VIC liest also von den gleichen Adressen aus der Videomatrix, wie in der Zeile zuvor. Da man die Videomatrix mit dem Prozessor nicht schnell genug ändern kann, muss man mit den Bits VM10-VM13 aus Register $d018 die BasisAdresse der Videomatrix umschalten (das Farb-RAM lässt sich leider nicht umschalten, daher ist die Farbwahl der Pixel nicht vollkommen frei).

Außerdem darf man den Zugriff auf $d011 zum Erzeugen der Bad Lines erst ab Zyklus 14 jeder Rasterzeile machen, denn sonst wird der RC in jeder Zeile gelöscht und die Darstellung der Bitmap nicht wie gewünscht. Dies hat aber auch zur Folge, dass die ersten drei c-Zugriffe des VIC in jeder Zeile keine gültigen Daten liefern, denn der erste c-Zugriff in Zyklus 15 erfordert, dass BA schon in Zyklus 12 auf Low gegangen sein muss, damit AEC in Zyklus 15 Low bleibt (AEC bleibt prinzipiell immer erst drei Zyklen nach der negativen Flanke von BA auf Low, dies lässt sich nicht umgehen). Da aber die Bad Line erst in Zyklus 14 erzeugt wurde, ist zwar in Zyklus 15 beim ersten c-Zugriff BA auf Low, aber AEC ist High und damit sind auch die internen Datenbustreiber D0-D7 des VIC geschlossen und da der Chip in NMOS gefertigt ist, liest er den Wert $ff und nicht die Videomatrix-Daten (die Datenbustreiber D8-D11 sind allerdings offen, jedoch wird dies in Abschnitt 3.14.6. noch näher erläutert), was am linken Bildrand als 24 Pixel breite Streifen sichtbar wird.

In der Praxis legt man im Speicher acht Videomatrizen an, die nach folgendem Schema benutzt werden: In der ersten Rasterzeile die erste Zeile der ersten Matrix, in der zweiten Zeile die erste Zeile der zweiten Matrix, usw..., in der achten Zeile die erste Zeile der achten Matrix, in der neunten Zeile die zweite Zeile der ersten Matrix, usw. Mit diesen acht Matrizen kann man eine komplette Bitmap zeilenweise abdecken.

Vom FLI-Modus gibt es noch einige Abarten, z.B. AFLI (Advanced FLI), das den Standard-Bitmap-Modus benutzt und Farbmischungen durch ähnlich gefärbte, nebeneinanderliegende Pixel simuliert, und IFLI (Interlaced FLI), das in einer Art Interlace-Verfahren abwechselnd zwei Halbbilder darstellt.

3.14.4 Linecrunch

Die Manipulation von YSCROLL bietet noch mehr Möglichkeiten, auf Bad Lines Einfluss zu nehmen. Man kann auch eine Bad Line vor ihrer korrekten Beendigung abbrechen, indem man durch Änderung an YSCROLL den Bad-Line-Zustand innerhalb einer angefangenen Bad Line vor Zyklus 14 wieder wegnimmt. Das hat mehrere Konsequenzen:

- Der Grafikdaten-Sequenzer geht in den Display-Zustand, es wird also Grafik dargestellt.
- Der RC wird nicht zurückgesetzt. Wenn man gleich die erste Bad Line eines Bilds auf diese Art abbricht, steht der RC noch von der letzten Zeile des vorigen Bildes auf 7.
- In Zyklus 58 der Zeile ist RC immer noch 7, darum geht der Sequenzer in den Idle-Zustand und VCBASE wird mit VC geladen. Da der Sequenzer aber innerhalb der Zeile im Display-Zustand war, wurde VC nach jedem g-Zugriff erhöht, also ist VCBASE jetzt effektiv um 40 erhöht worden. Der RC macht keinen Überlauf, er bleibt auf 7.

Mit diesem Verfahren hat man also die Darstellung einer Textzeile auf deren letzte Rasterzeile reduziert, denn weil VCBASE um 40 erhöht wird, geht der VIC anschließend zur nächsten Zeile über. Daher wird dieser Effekt als "Linecrunch" bezeichnet: Man kann damit einzelne Textzeilen "wegcrunchen".

Wenn man nun in jeder Rasterzeile so vorgeht, so steht der RC immer auf 7 und es finden keine c-Zugriffe statt, aber VCBASE wird in jeder Zeile um 40 erhöht. Dies führt irgendwann dazu, dass VCBASE die 1000-Byte-Grenze der Videomatrix überschreitet und der VIC auch die letzten, normalerweise unsichtbaren 24 Byte der Matrix darstellt (in denen u.a. die Spritedatenzeiger abgelegt sind) und bei Erreichen von 1024 wieder bei Null anfängt.

Dadurch, dass man ganze Textzeilen auf je eine Rasterzeile crunchen kann, hat man eine Möglichkeit, den Bildschirminhalt schnell um große Distanzen nach oben zu scrollen, ohne den Grafikspeicher zu ändern, ähnlich wie man ihn mit FLD nach unten scrollen kann. Der einzige störende Nebeneffekt dabei ist, dass sich die weggecrunchten Zeilen am oberen Bildschirmrand ansammeln, was unschön aussieht. Hier kann man aber durch Anwendung eines der ungültigen Grafikmodi recht praktisch im Bereich dieser Zeilen auf schwarz schalten.

3.14.5 Verdoppelte Textzeilen

Normalerweise ist die Darstellung einer Textzeile nach 8 Rasterzeilen zu Ende, denn dann ist RC=7 und in Zyklus 58 der letzten Zeile geht der Sequenzer in den Idle-Zustand (siehe Abschnitt 3.7.2.). Erzeugt man jetzt aber einen Bad-Line-Zustand zwischen Zyklus 54-57 der letzten Zeile, so bleibt der Sequenzer im Display-Zustand und der RC wird nochmals erhöht (und läuft damit auf Null über). Dann beginnt der VIC in der nächsten Zeile nochmal mit der Darstellung der vorigen Textzeile. Da keine neuen Videomatrixdaten gelesen wurden, wird die vorige Textzeile einfach doppelt dargestellt.

3.14.6 DMA-Delay

Die trickreichste Bad-Line-Manupulation besteht darin, innerhalb von Zyklus 15-53 einer Rasterzeile des Anzeigefensters, innerhalb der der Grafikdatensequenzer im Idle-Zustand ist, einen Bad-Line-Zustand zu erzeugen, z.B. durch Ändern von Register $d011 derart, dass YSCROLL gleich den unteren drei Bit von RASTER ist.

Der VIC setzt dann sofort im nächsten Zyklus BA nach Low, geht in den Display-Zustand und beginnt mit dem Lesen aus der Videomatrix (der Prozessor ist nun angehalten, denn BA ist Low und er will den nächsten Opcode lesen). Allerdings schwingt AEC erst noch drei Zyklen lang mit ø2 mit, bevor es ebenfalls auf Low bleibt. Dieses Verhalten (AEC erst drei Zyklen nach BA) ist im VIC festverdrahtet und lässt sich nicht umgehen.

Trotzdem greift der VIC auf die Videomatrix zu, oder versucht es zumindest, denn solange AEC in der zweiten Taktphase noch High ist, sind die Adressbustreiber und Datenbustreiber D0-D7 des VIC im Tri-State und der VIC liest statt der Daten aus der Videomatrix in den ersten drei Zyklen den Wert $ff an D0-D7. Die Datenleitungen D8-D13 des VIC haben allerdings keinen Tri-State-Treiber und sind immer auf Eingang geschaltet. Allerdings bekommt der VIC auch dort keine gültigen Farb-RAM-Daten, denn da AEC High ist, kontrolliert offiziell der 6510 noch den Bus und sofern dieser nicht zufällig gerade den nächsten Opcode vom Farb-RAM lesen will, ist der Chip-Select-Eingang des Farb-RAMs nicht aktiv. Stattdessen stellt ein 4-Bit Analogschalter (!), U16, eine Verbindung zwischen den Datenbits D0-D3 des Prozessors und den Datenbits D8-D13 des VIC her. Diese Verbindung besteht immer bei AEC High und soll dem Prozessor ggf. den Zugriff auf das Farb-RAM ermöglichen. Lange Rede, kurzer Sinn: Der VIC liest in den ersten drei Zyklen, nachdem BA auf Low gegangen ist als Zeichenzeiger $ff und als Farbinformation die untersten 4 Bit des Opcodes nach dem Zugriff auf $d011.
Erst danach werden wieder reguläre Videomatrixdaten gelesen.

Diese Daten werden ganz normal ab dem Anfang der internen Videomatrix-/Farbzeile abgelegt und nach jedem nachfolgenden g-Zugriff (mit dem Erzeugen der Bad Line wurde auch der Sequenzer in den Display-Zustand geschaltet) wird VC erhöht. Die c- und g-Zugriffe werden bis Zyklus 54 fortgesetzt. Dadurch, dass mit den Zugriffen aber erst mitten in der Zeile begonnen wurde, fanden weniger als 40 Zugriffe statt, also wurde auch VC insgesamt um weniger als 40 in dieser Rasterzeile erhöht und hat keinen durch 40 teilbaren Wert mehr, wie dies normalerweise am Ende einer Rasterzeile immer der Fall ist. Aufgrund der Arbeitsweise des VC (siehe Abschnitt 3.7.2.) setzt sich diese "Verstimmung" in allen folgenden Zeilen fort. Der ganze Bildinhalt erscheint dadurch nach rechts gerollt, und zwar um soviele Zeichen, wieviele Zyklen der $d011-Zugriff nach

Zyklus 14 gemacht wurde. Da die c-Zugriffe innerhalb der Zeile erst später als in einer normalen Bad Line einsetzen, wird dieses Verfahren als "DMA-Delay"
bezeichnet.

Damit ist es möglich, den kompletten Bildschirm um große Distanzen seitlich zu verschieben (dies funktioniert mit Bitmap-Grafiken genauso wie mit Textschirmen, denn der VC wird auch zum Zugriff auf die Bitmap-Daten benutzt), ohne den Grafikspeicher mit dem Prozessor umkopieren zu müssen.
Kombiniert man den DMA-Delay mit FLD und Linecrunch, ist es möglich, bildschirmfüllende Grafiken ohne nennenswerten Rechenzeitverbrauch um fast beliebig große Distanzen in alle Richtungen zu scrollen.

Das Experimentieren mit dem DMA-Delay (und mit Bad-Line-Effekten generell) ist auch die beste Methode, die interne Funktionsweise des VIC, insbesondere von RC und VC, zu ergründen und zu bestimmen, in welchen Taktzyklen bestimmte Vorgänge im VIC ablaufen.

Es sollte noch erwähnt werden, dass DMA-Delay nicht nur durch Manipulation von YSCROLL, sondern auch mit dem DEN-Bit aus Register $d011 erzielt werden kann. Dazu muss man YSCROLL auf Null setzen, damit Rasterzeile $30 zur ersten Bad Line wird und DEN mitten in Zeile $30 von gelöscht auf gesetzt schalten. Bad Lines können nämlich nur auftreten, wenn in Zeile $30 das DEN-Bit mindestens einen Zyklus lang gesetzt war, und wenn YSCROLL Null ist, tritt in Zeile $30 der Bad-Line-Zustand ein, sobald DEN gesetzt ist.

3.14.7 Sprite-Stretching

Da die Sprites einfacher aufgebaut sind als die Textgrafik, sind mit ihnen nicht so viele besondere Effekte möglich, aber darunter ist ein sehr interessanter Effekt, der die Funktionsweise der Sprite-Y-Expansion ausnutzt: Durch Ändern der MxYE-Bits in Register $d017 ist es nämlich nicht nur möglich, für jede Spritezeile einzeln zu bestimmen, ob sie verdoppelt werden soll, sondern man kann auch einzelne Zeilen drei- oder mehrmals wiederholen lassen und damit ein Sprite oder Teile davon um beliebige Faktoren vergrößern.

Dieser Effekt lässt sich wie folgt verstehen (siehe dazu Abschnitt 3.8.1.):

Angenommen, wir befinden uns in Zyklus 55 einer Rasterzeile, in der Sprite 0 angeschaltet ist und deren Y-Koordinate gleich der Y-Koordinate des Sprites ist, also in der Zeile bevor das Sprite dargestellt wird. Das M0YE-Bit sei gesetzt. Dann schaltet der VIC den DMA für Sprite 0 an und löscht MCBASE und das Expansions-Flipflop. BA geht nach Low, damit der VIC in Zyklus 58 und 59 in den zweiten Taktphasen zugreifen kann. In Zyklus 58 wird MC mit MCBASE geladen und dadurch ebenfalls gelöscht, und der p-Zugriff für das Sprite gemacht. Anschließend werden die drei s-Zugriffe ausgeführt und MC nach jedem Zugriff erhöht, steht also dann auf 3.

Nun wartet man auf Zyklus 16 der folgenden Zeile. Da das Expansions-Flipflop gelöscht ist, bleibt MCBASE weiterhin auf Null. Jetzt löscht man zunächst das M0YE-Bit und setzt damit das Flipflop, setzt das M0YE-Bit aber danach gleich wieder. In Zyklus 55 wird das Flipflop dann invertiert, da M0YE wieder gesetzt ist, und ist nun also gelöscht (hätte man das M0YE-Bit nicht gelöscht, wäre das Flipflop nun gesetzt). Dies ist aber genau der Zustand, in dem der VIC auch in Zyklus 55 der vorigen Zeile war. Der VIC "glaubt" daher, sich erst in der ersten Rasterzeile einer expandierten Spritezeile zu befinden und wird (da MC immer noch Null ist) die erste Spritezeile noch zweimal aus dem Speicher lesen, insgesamt jetzt also dreimal: Die erste Spritezeile wurde verdreifacht.

Ein weiterer interessanter Effekt lässt sich erzielen, wenn man genau wie oben vorgeht, aber das M0YE-Bit nicht nach Zyklus 16 löscht, sondern in der zweiten Phase von Zyklus 15. Dann wird MCBASE um 1 erhöht und die nächste Spritezeile wird mit MC=1..3 aus dem Speicher gelesen, also ein Byte höher als vorgesehen. Diese "Verstimmung" setzt sich komplett in der Darstellung

des Sprites fort. Daher wird auch die Bedingung MC=63 für das Abschalten des Sprite-DMA in Zyklus 16 nicht erreicht und das Sprite wird effektiv zweimal direkt hintereinander dargestellt. Erst nach Ende der zweiten Darstellung steht MC auf 63 und der DMA wird abgeschaltet.

4. Die Adressen 0 und 1 und der $de00-Bereich

Der Adressbereich $de00-$dfff des 6510 (siehe 2.4.1.) ist für externe Erweiterungen des C64 reserviert und normalerweise mit keinem anderen Baustein (RAM, I/O) verbunden. Ein Lesezugriff von diesen Adressen liefert scheinbar zufällige Daten. dasselbe gilt für die oberen Nybbles der Adressen $d800-$dbff (das Farb-RAM).

Auf einigen C64 sind diese Daten jedoch gar nicht "zufällig", sondern sind vielmehr mit den Daten identisch, die der VIC in der ersten Phase des Taktzyklus aus dem Speicher gelesen hat. Dieser Effekt ist jedoch nicht auf allen Geräten und nicht immer präzise nachvollziehbar.

Davon abgesehen, dass man dadurch die Möglichkeit hat, das VIC-Timing allein per Software zu vermessen (die Timing-Diagramme aus [4], auf denen auch die Diagramme in diesem Artikel basieren, sind z.B. mit dieser Methode erstellt worden), kann man den 6510 auch Programme im $de00-Bereich oder im Farb-RAM ausführen lassen, wenn man den VIC eine derartige Grafik darstellen lässt, dass der 6510 aus den vom VIC gelesenen Grafikdaten gültige Opcodes erhält.

Mit einem ähnlichen Effekt kann man auch vom Prozessor aus die RAM-Adressen 0 und 1 beschreiben. Diese sind eigentlich nicht zugänglich, da an diesen Adressen prozessorintern das Datenrichtungsregister und das Datenregister des 6510-I/O-Ports eingeblendet werden und bei einem Schreibzugriff die Datenbustreiber im Tri-State bleiben. Allerdings wird die R/W-Leitung auf Low gelegt (dies ist damit zu erklären, dass der I/O-Port erst nachträglich in das existierende Design des 6502 integriert wurde) und dadurch gelangt das in der ersten Taktphase vom VIC gelesene Byte ins RAM. Will man also einen bestimmten Wert an Adresse 0 oder 1 schreiben, so muss man nur ein beliebiges Byte an diese Adressen schreiben und dafür sorgen, dass der VIC in der Phase zuvor den gewünschten Wert aus dem RAM gelesen hat.

Die Adressen 0 und 1 lassen sich natürlich auch wieder vom Prozessor aus lesen. Entweder über den $de00-Bereich oder mit Hilfe von Spritekollisionen. Dabei lässt man den VIC eine Bitmap ab Adresse 0 darstellen und bewegt ein Sprite, das nur aus einem Pixel besteht, über die einzelnen Bits der ersten beiden Bytes der Bitmap. Je nachdem, ob eine Kollision registriert wurde oder nicht, kann man so den Zustand der einzelnen Bits herausfinden und zu einem Byte zusammensetzen.

Anhang A: Literaturverzeichnis

[1] Commodore Büromaschinen GmbH, "Alles über den Commodore 64", Anhang L: "Datenblatt Mikroprozessor 6510", 1984

[2] dto., Anhang N: "6566/6567 Video Interface Controller (VIC-II) Chip Specifications"

[3] dto., Kapitel 5, Abschnitt "Speicherverwaltung beim Commodore 64", S.256-263

[4] Marko Mäkelä, "The memory accesses of the MOS 6569 VIC-II and MOS 8566 VIC-IIe Video Interface Controller" (AKA: Pal timing), 15.07.1994

[5] John West, Marko Mäkelä, "Documentation for the NMOS 65xx/85xx Instruction Set" (AKA: 64doc), 03.06.1994

Anhang B: Danksagungen

Mein Dank geht an

- Marko Mäkelä, Andreas Boose, Pasi Ojala und Wolfgang Lorenz für die Arbeit, die sie in die Untersuchung des VIC gesteckt haben
- Kaspar Jensen für das Korrekturlesen der englischen Version dieses Textes und für seine Verbesserungsvorschläge
- Adam Vardy <abe0084@InfoNET.st-johns.nf.ca> für den Hinweis, dass meine Beschreibung der Grafikdarstellung im Idle-Zustand falsch war