Sprites ohne Esoterik

Auch fortgeschrittenen Programmierern bleibt es meist rätselhaft, wie der C 64 die Sprites auf den Bildschirm zaubert. Wir wollen dieses Thema einmal ohne Geheimniskrämerei (Esoterik) angehen.

Noch vor fünf Jahren war alles ganz einfach. Im guten alten PET 2001 wurde das Bild durch ein mittleres TTL-Bergwerk erzeugt. In den Nachfolgern CBM 30xx und so weiter wird diese Aufgabe durch einige TTL-ICs und den Bildschirmcontroller MC6845 erledigt. Bis dahin war alles überschaubar.

Da Commodore die Halbleiterfirma MOS Technology besitzt, liegt es nahe, eigene Video-Controller zu entwickeln. So treibt im VC 20 der noch relativ einfache VIC I (Video-Interface-Chip I) sein Unwesen. Im C 64 wirft der sehr komplexe VIC II die »Flammenschrift« auf den Bildschirm.

Die Funktionsweise dieses Video-Controllers soll dieser Bericht ein wenig enträtseln. Leider muß man es größtenteils durch Überlegung lösen.

Zunächst ein paar Worte zum Bildschirmformat: Es entspricht weitgehend der normalen Fernsehnorm. Je 312 Rasterzeilen ergeben fünfzig Bilder in der Sekunde. Jede Rasterzeile (Bild 1) ist 64 μs lang und beginnt mit einem Synchronisationsimpuls, der dem Monitor mitteilt, wann er mit der neuen Zeile beginnen soll. Ohne diese Synchronisation gäbe es nur Bildsalat und durchlaufende Bilder. Der Synchronisationsimpuls entspricht in der Helligkeitsskala der Videosignale einem Dunkelschwarz, so daß man den Strahlrücklauf nicht sehen kann. Es gibt übrigens nur fünf Helligkeitsstufen:

1. Schwarz
2. Rot, Blau, Braun, Grau 1
3. Violett, Grün, Orange, Hellrot, Grau 2, Hellblau
4. Türkis, Gelb, Hellgrün, Grau 3
5. Weiß

Es ist empfehlenswert, für die Vorder- und Hintergrundfarbe Farben aus verschiedenen Helligkeitsstufen zu wählen, da man sonst auf farblosen Monitoren nicht viel sehen kann...

40 μs der Rasterzeile werden für das eigentliche Bild benützt. Bei der Taktfrequenz (des VIC) von zirka 8 MHz ergeben sich 512 Punkte auf der ganzen Rasterzeile, was mit der horizontalen Auflösung der Positionen der Sprites übereinstimmt. Aus den 64μs ergibt sich eine Zeilenfrequenz von 15625 Hz, die für das unangenehme Pfeifen verantwortlich ist, das man bei jedem Bildschirm hören kann, sofern man noch gute Ohren hat. Das horizontale Scrolling ist übrigens sehr einfach zu verwirklichen. Statt dem ganzen Bild wird kurzerhand der Synchronisationsimpuls und das durch den Rand gebildete »Fenster« verschoben.

Das vertikale Format ist folgendermaßen aufgebaut:
4 Zeilen Schwarz, damit man den Strahlrücklauf nicht sieht
3 Zeilen Synchronisationsimpuls
4 Zeilen Schwarz
51 Zeilen Rand
200 Zeilen Bild
51 Zeilen Rand

Den aufmerksamen Lesern wird aufgefallen sein, daß ich hier von 312 Zeilen pro Bild rede und nicht von 625 Zeilen, wie es beim normalen Fernseher der Fall ist. Beim Fernseher werden nacheinander zwei »Halbbilder«, die zueinander um eine halbe Rasterzeile versetzt sind, geschrieben, so daß die Auflösung höher ist. Nachteilig ist bei diesem Verfahren die geringe Bildwiederholfrequenz von 25 Hz, bei der das Bild leicht flimmert. Beim C 64 sind beide Bilder identisch und nicht versetzt (non-interlace), die Bildwechselfrequenz beträt 50 Hz.

Um ein Zeichen darzustellen, muß der VIC in einer einzigen Mikrosekunde folgende Informationen lesen:

1. POKE-Wert des Zeichens aus der Videomatrix; zeigt in den Zeichengenerator
2. Farbe aus der Farbmatrix
3. Daten aus dem Zeichengenerator.

Das heißt, zusammen mit den Zugriffen der CPU müßte der Speicher mit einer Zyklusfrequenz von 4 MHz (!) betrieben werden. Das ist nicht möglich. Dieses Problem wird auf eine andere Weise gelöst.

Die CPU 6510 gibt an ihrem Ausgang Pin 2 einen Takt von zirka 1 MHz ab (Bild 2). Wenn Pin 2 Low ist, ist der Bus frei, wenn Pin 2 High ist, benötigt die CPU den Bus für ihre Speicherzugriffe.

Der VIC liest, während Pin 2 Low ist, die Daten aus dem Zeichengenerator oder aus der hochauflösenden Grafik.

Das Farb-RAM wird vom VIC parallel zum normalen RAM gelesen. Der VIC hat also einen 12-Bit-Datenbus.

Zum Lesen der Videomatrix ist keine Zeit mehr übrig. Deswegen muß der VIC regelmäßig die CPU anhalten, um die Daten lesen zu können. Durch dieses »Kaltstellen« wird der Prozessor natürlich verlangsamt. Damit die CPU nur um etwa zehn Prozent und nicht um die Hälfte verlangsamt wird, hat der VIC intern Puffer für die Video- und Farbdaten der aktuellen Zeile.

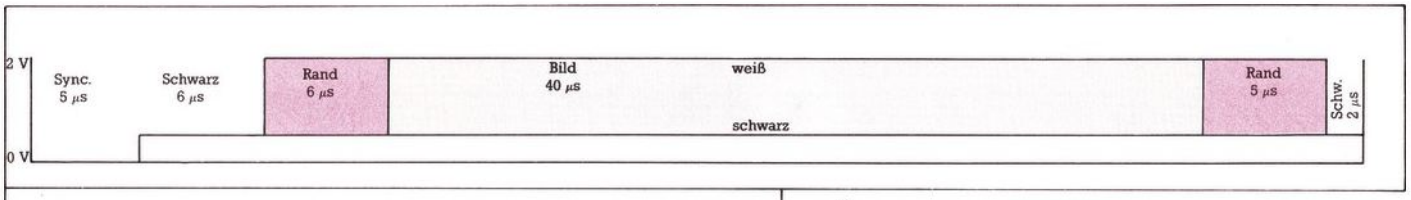

Bild 1. Videosignal einer Rasterzeile

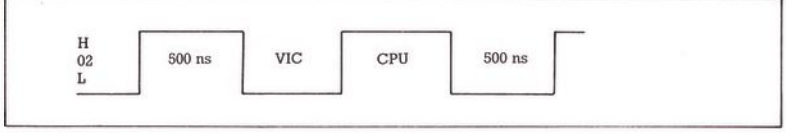

Bild 2. Takt der CPU 6510

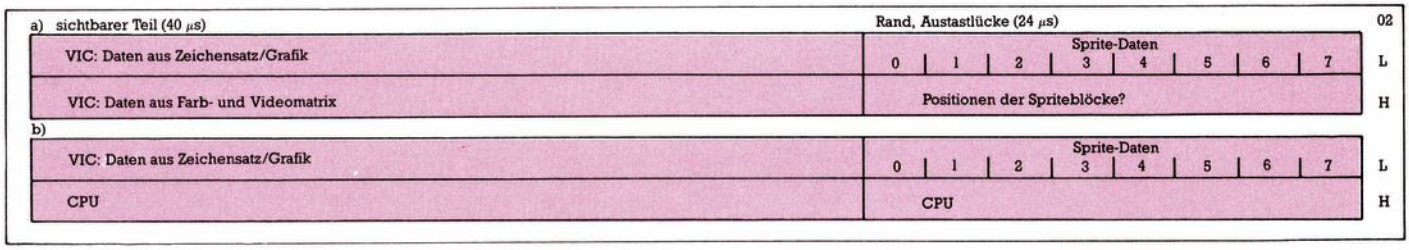

Bild 3. Zeitliche Anordnung der Speicherzugriffe

Daraus ergibt sich die auf Bild 3 gezeigte Reihenfolge der Zugriffe:
a) Letzte Rasterzeile der vorherigen Zeile: Die CPU wird angehalten, in die Puffer werden die Daten für die neue Zeile gelesen.
b) Während den sieben ersten Rasterzeilen der neuen Zeile wird die CPU nicht angehalten.
a) Wieder von vorn.

Leider hat auch dieses Verfahren Nachteile. Erstens wird der Prozessor verlangsamt, zweitens geraten bei zeitkritischen Programmen durch das Anhalten des Prozessors die Zeitverhältnisse aus dem Lot. Deswegen wird zum Beispiel beim Laden von Programmen von Kassette der Bildschrim abgeschaltet!

Und nun zu den Sprites. Unbegrenztes Vertrauen in die Leistungsfähigkeit der CPU 6510 ist nicht angebracht. Im VIC ist kein Maschinenprogramm versteckt. Es wäre ganz einfach zu langsam. Statt dessen werden die Sprites durch aufwendige Hardware erzeugt. Dieser Aufwand macht aber nicht so viel aus, da eben alles in einem Chip versammelt ist. Schwierig wird die Entwicklung eines solchen Video-Controllers vor allem durch die notwendige hohe Geschwindigkeit: Der IC hat immerhin eine Taktfrequenz von 8 MHz zu verkraften. Deswegen wird der VIC gewaltig heiß und residiert in einem unter kühlendem Blech verborgenen Keramikgehäuse.

Trotz allem ist die Logik für die Sprites eigentlich verblüffend einfach und elegant. Da Digitalelektroniker auch beim Einschlafen noch Gatter zählen, die über Schafe springen, und auch sonst mit jedem Gatter geizen (einfache ICs sind billiger), kann man schließen, daß wahrscheinlich auch hier der einfachste Weg benützt wird.

Die Daten der Sprites müssen zuerst gelesen werden. Die Logik zur Errechnung der Adressen der Sprites soll hier nicht erklärt werden, da sie nicht besonders interessant ist. Es ist mir leider nicht bekannt, wann die Spritezeiger (am Ende der Videomatrix) gelesen werden.

In jeder Rasterzeile müssen die Daten aller Sprites gelesen werden, also 3 x 8 Byte, die nahtlos in das »gemütliche Eckchen« in der Austastlücke passen. Jetzt wissen wir auch, wieso die Sprites ein so unmögliches Format (24 x 21 Pixel) haben...

Ich möchte nun anhand von Bild 4 erklären, wie die Sprites dort angezeigt werden, wo sie hingehören. Im Schema ist nur die Schaltung für ein Sprite gezeichnet, die anderen Sprites sind gleichartig aufgebaut.

Die Spritedaten werden in einen Puffer gelesen. Ein Zähler gibt die Nummer der aktuellen Rasterspalte, also die X-Koordinate, an. Davon wird die X-Koordinate der Sprites abgezogen, und man erhält eine auf den »Ursprung« des Sprites bezogene Koordinate. Die Ausdehnung von Sprites in der X-Richtung ist sehr einfach: Die Koordinate wird einfach durch zwei geteilt (in Binärsystem sehr einfach: rechts schieben). Das gleiche geschieht übrigens auch bei der Ausdehnung von Sprites in der Y-Richtung. Ein Multiplexer gibt das, durch die so erhaltene Koordinate, gewählte Bit (oder Bitpaar bei mehrfarbigen Sprites) aus. Falls die Koordinate nicht im Bereich des Sprites liegt, gibt der Multiplexer einfach den Wert für »Sprite transparent« — also 0 — aus.

Wie werden die Sprites nach der Priorität geordnet und Kollisionen von Sprites mit anderen Sprites oder mit dem Vordergrund des normalen Bilds erkannt?

Jedes Sprite gibt ein Signal von sich, das angibt, ob das Sprite jetzt transparent oder »deckend« ist. Eine relativ einfache Schaltung (in TTL nur 28 Gatter: 74LS148) erzeugt die Nummer des Sprites mit der höchsten Priorität, das gerade deckend ist oder zeigt an, daß gar kein Sprite deckend ist.

Nun muß noch entschieden werden, wer jetzt Vorfahrt hat (in der Reihenfolge der Prioritäten): Der Rand, Sprites im Vordergrund, das normale Bild, Sprites im Hintergrund oder der Hintergrund. Entsprechend dieser Entscheidung wird der richtige Farbcode ausgewählt und an den PAL-Codierer weitergegeben, der das Helligkeitssignal (Videosignal) und das Farbsignal (Chroma) erzeugt.

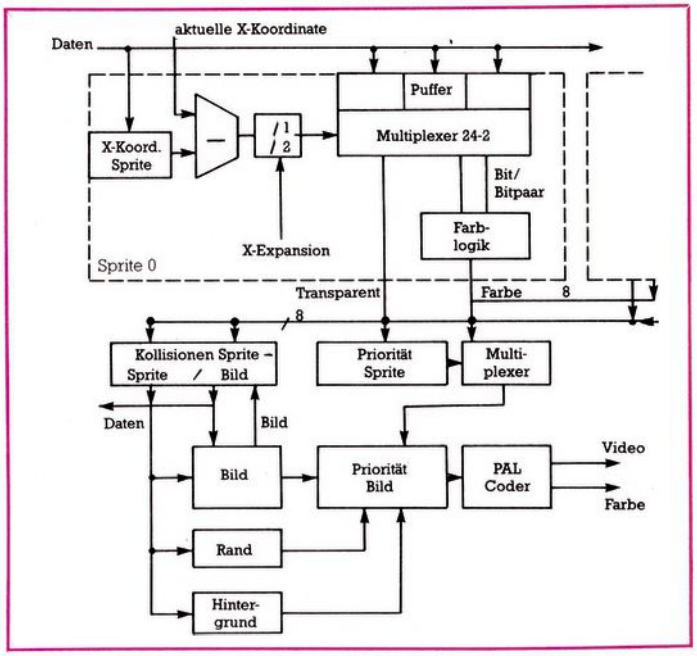

Bild 4. Schema der Sprite-/Bildlogik (stark vereinfacht)

Die Erkennung von Kollisionen ist keine schwierige Sache mehr. Wenn sowohl das Bild als auch ein oder mehrere Sprites nicht transparent sind, werden im Kollisionregister für Sprite-Bild-Kollisionen die Bits der nicht transparenten Sprites gesetzt.

Sprite-Sprite-Kollisionen sind etwas schwieriger auszuwerten, auch hier bleibt der Aufwand aber im Rahmen.

Ich hoffe, daß ich bei den Lesern mit diesem Artikel jeglichen Geisterglauben ausgetrieben habe. (Pascal Dornier/aa)