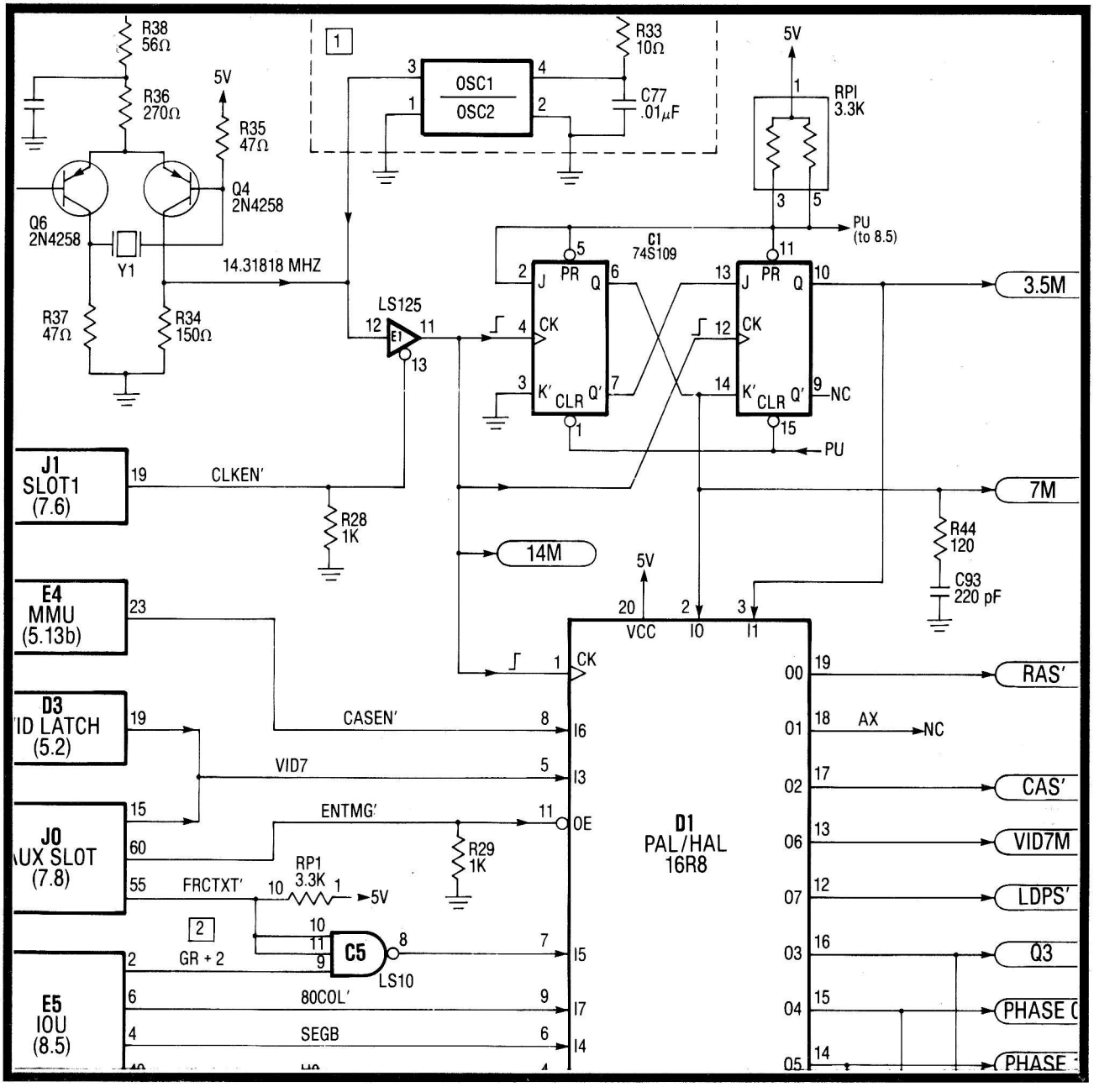
Apple *IIe* Hardware

Jim Sather



Apple //e Hardware

Jim Sather

Ampersand Verlag, Berlin

Titel der englischsprachigen Originalausgabe:
Understanding the Apple IIe

Deutsche Übersetzung: Arne Schäpers

ISBN: 3-89058-040-8

Printed in West Germany

Inhalt

Kapitel 1 - **Der Apple //e - ein Überblick** 1

Ein Abriß des Apple // 1
Der Mikroprozessor und die Busstruktur 1, Speicherbausteine, Datenorganisation und Adreßdekodierung 3, Die Steckplätze für Zusatzkarten 5, Der spezielle Steckplatz 5, MMU, IOU und HAL 6, Die Videoausgabe 8, Die Tastatur 11, Weitere I/O 12, Die Stromversorgung 14
Zusammenfassung 14

Kapitel 2 - **Die Busstruktur des Apple //e** **17**

Computerbusse und tri-State-Logik 17
Die Schubfach-Analogie 22
Die CPU, der RAM und der ROM 22
Die Adressierung des RAM und die Verteilung der Datenbits 23
Der gemultiplexte RAM-Adreßbus 24, Der Videoscanner 24, Die Verteilung der RAM-Daten 26
Die Adreßdekodierung 27
Eingabe und Ausgabe 32
Die Eingabe von der Tastatur 32, Die Steckplätze 34, Ein-/Ausgabe zusammen mit Diskettenlaufwerken 34, Direkter Speicherzugriff (DMA) 35, Der serielle Eingangsmultiplexer 35, Die seriellen Ausgänge 36
Die vollständige Busstruktur 38

Kapitel 3 - **Die Takterzeugung und der Videoscanner** **39**

Abriß des Taktgenerators 39
Die Taktsignale 40
Die genauen Frequenzen 42
Das Zeitdiagramm 43
Die Verteilung der Taktsignale 46
Detaillierte Beschreibung der Taktsignale 46
PHASE0 und PHASE1 46, 14M, 7M und COLOR REFERENCE 48, RAS', CAS' und Q3 49, LDPS' und VID7M 50
Der Aufbau eines Fernsehbildes 51
Der Videoscanner 52
Der Horizontalzähler 52, Der Vertikalzähler 54, Andere Modelle und der Videoscanner 54, Der Flash-Zähler und "Power-up" RESET 57
Der lange Zyklus 58
Die Hardware des Taktgenerators 58
Der HAL 60

Kapitel 4 - **Der Mikroprozessor 6502** **65**

Die Signale des 6502 65
Taktsignale - PHASE0, PHASE1, PHASE2 65, Die Adreßausgänge und R/W' 66, Die Verbindung zum Datenbus 66, RESET' 66, Interrupts - IRQ' und NMI' 67, READY 67, SYNC 67, SET OVERFLOW' 68
Die Verbindungen des 6502 im Apple //e 68
Die Speicherbelegung des 6502 70
Die Zeitsteuerung des 6502 im Apple //e 71
Die Programmierung des Apple 75
DMA im Apple //e 78
6502-Interrupts im Apple //e 82
RESET' 82, NMI' und IRQ' 83, Der Befehl BRK 86, Die Behandlung von IRQ' und BRK im verbesserten Apple //e 87, Die Rangfolge der Interrupts 89
Der Mikroprozessor 65C02 90
Befehlsübersicht und Unterschiede zwischen 6502 und 65C02 92

Kapitel 5 - **RAM und die Verwaltung des Speichers** 99

Die dynamischen RAMs 99
Die Verbindungen des RAMs im Apple //e 101
Die RAM-Adreßmultiplexer 104
Die Adreßlogik des Videoscanners 106, Der Scan-Prozeß für TEXT und LoRes 109, Der Scan-Prozeß für HiRes 111, Der Scan-Prozeß im Modus MIXED 113
Der RAM-Refresh im Apple //e 119
Die Speicherverwaltung 120
Die Softswitches der MMU 121, Die Schaltung des hohen RAMs ($D000-$FFFF) 123, Die Schaltung zwischen dem RAM der Hauptplatine und dem AUX-RAM 127, Der Aufbau des I/O-Bereichs ($C000-$CFFF) 129, KBD' und MD IN/OUT' 130, Der funktionale Aufbau der MMU 131, Laufzeiten in der MMU 135
Das "Timing" des RAMs im Apple //e 136
Die 80-Zeichen-Karte mit 1 kByte RAM 140

Kapitel 6 - **ROM im Apple //e** 143

ROM-Hardware 143
ROMEN1' und ROMEN2' 144
ROM in den Steckplätzen 146
Das Zeitverhalten der ROMs 147
Die Firmware des Apple 148
Das Monitorprogramm 149, Der Apple II+ 150, Der Einfluß der 16k-Sprachkarte 150, Der Apple //e 151
Der verbesserte Apple //e 152

Kapitel 7 - **Ein-/Ausgabe im Apple //e** 153

Die Dekodierung der Peripherieadressen 153
Die Softswitches der IOU 155
Der Aufbau des seriellen I/O 158
Die Tastatur des Apple //e 162
Zusammenfassung der funktionellen Eigenschaften der Tastatur 167, Das I/O STROBE'-Protokoll 174
Die Struktur der Ein-/Ausgabe
KSW und CSW 175, I/O und der Monitor des Apple 175, Die Umleitung des I/O zu anderen Geräten 176, Periphere Zusatzkarten und primäre I/O-Geräte 177
Die Zeitabläufe des I/O 178
Der spezielle Steckplatz 181

Kapitel 8 - **Der Videogenerator** 185

Farbsignale 190
Die Repräsentation der Bildinformation im Speicher 192
Die Hardware des Videogenerators 194
Die Eingänge des Video-ROMs 196, Laden und Schieben von Bitmustern 200, Die Videoausgabe im europäischen Apple //e 202
Die Softswitches für den Videomodus 206
Die Zeitsignale der Bilderzeugung 210
Die Ausgabe von TEXT 214
Die Erzeugung und Ausgabe von LoRes-Grafik 217
Die Ausgabe von LoRes40 218, Die Ausgabe von LoRes80 220
Die Erzeugung und Ausgabe von HiRes-Grafik 222
Die Ausgabe von HiRes40 222
Die Ausgabe von HiRes80 226
Die Umschaltungen im Modus MIXED 228

Kapitel 1

Der Apple //e - ein Überblick

Der folgende Überblick ist ein kurz gehaltener Abriß der Hardware des Apple //e. Er ist nicht als Beschreibung dessen gedacht, was man mit einem ausgefuchsten Programm und dem //e alles anstellen kann, sondern mehr als Beschreibung der grundlegenden Möglichkeiten, die dem Programmierer (und dem Entwurfsingenieur für Zusatzkarten) zur Verfügung stehen. Wir beginnen mit einer Einführung über Mikrocomputer im allgemeinen und einer Definition der dazugehörigen Terminologie. Leser, die gerade erst begonnen haben, sich mit den elektronischen Innereien ihres Computers zu beschäftigen, sollten sich durch die nun folgende Informationsflut nicht entmutigen lassen. Es ist auch nicht nötig, alle hier vorgestellten Punkte bis ins letzte Detail zu verstehen - die folgenden Kapitel stellen eine starke Erweiterung der einzelnen Oberpunkte dar und haben dabei auch mehr Raum für Erklärungen. Speziell Kapitel 2 beantwortet viele Fragen, die bei der Lektüre von Kapitel 1 auftauchen dürften.

Der Apple //e ist eine erweiterte und verbesserte Version des Apple II, der Mitte der siebziger Jahre von Steve Wozniak ("Woz") konstruiert wurde. Er entspricht einem Apple II mit einer 16k-Speichererweiterung in Steckplatz 0 und einer 80-Zeichen-Karte in Steckplatz 3. Dabei wurde besonderen Wert auf "Aufwärtskompatibilität" gelegt, d.h. Programme, die für den Apple II geschrieben wurden, laufen im allgemeinen auch auf einem //e - der //e bietet allerdings darüber hinaus noch einige zusätzliche Möglichkeiten. Die großen Unterschiede liegen in weiteren 64 kByte RAM, einer erweiterten Tastatur mit einigen zusätzlichen Tasten und der Möglichkeit, doppelt so hoch auflösende Grafik darzustellen wie sein Vorgänger. Außerdem enthält er noch eine ganze Reihe kleinerer Verbesserungen, die allerdings für den Programmierer zumeist von untergeordneter Bedeutung sind.

Der Grund für die Firma Apple, eine neue Version des Apple II auf den Markt zu bringen, liegt hauptsächlich in einer Reduzierung der Herstellungskosten (der //e enthält erheblich weniger Bauteile als der Apple II) sowie die Beseitigung einiger Mängel des Computers beim Einsatz als Textverarbeitungssystem. Man kann das Ergebnis ohne Übertreibung als gelungen bezeichnen: Apple hat mit dem //e eine echte Verbesserung des II bzw. II+ herausgebracht, die sich ziemlich nahtlos in die Apple II-Serie einfügt. Die folgenden Beschreibungen gelten hauptsächlich für den //e, sind in weiten Teilen aber auch für den II bzw. II+ gültig.

Ein Abriß des Apple //

Der Apple //e besteht aus fünf physikalisch voneinander getrennten Einheiten: Gehäuse, Tastatur, Stromversorgung, Lautsprecher und Hauptplatine ("motherboard"). Lautsprecher, Stromversorgung und Tastatur können dabei als (notwendige!) Zusätze betrachtet werden - sie sind in dieser oder ähnlicher Form in jedem Computer vorhanden. Der eigentliche Apple //e besteht allein aus der Hauptplatine, deshalb wird er auch manchmal als "Einplatinencomputer" bezeichnet. Auf einer einzigen Platine befinden sich der Mikroprozessor, der Speicher, die Elektronik zur Erzeugung der Videosignale inklusive der Grafik, sieben Erweiterungssteckplätze, ein spezieller Steckplatz und die notwendigen Schaltkreise zur Kommunikation mit einer Reihe von "externen" Geräten wie der bereits erwähnten Tastatur, den Spielsteuerungen etc. Alle diese Funktionen sind Teil einer organisierten Struktur, in deren Zentrum sich der Mikroprozessor befindet.

Der Mikroprozessor und die Busstruktur

Der *Mikroprozessor* ist die zentrale Einheit eines Computers, sozusagen das Gehirn der Maschine. Von außen sieht man ihm das nicht an - physikalisch ist ein Mikroprozessor nur ein schwarzer Käfer, allerdings einer mit 40 Anschlüssen. Ein Mikroprozessor (oder auch *CPU* = Central Processing Unit - "zentrale Verarbeitungseinheit") enthält eine Vielzahl von elektronischen Schaltungen und ist in der Lage, Befehle aus einem angeschlossenen Speicher zu lesen und sie in entsprechende Aktionen umzusetzen. Ein Computer, dessen zentrale Verarbeitungseinheit aus einem Mikroprozessor, d.h. aus einem einzigen Baustein besteht, wird als *Mikrocomputer* bezeichnet - im Unterschied zu Computern, bei denen diese CPU aus einem ganzen Satz von Chips besteht.

Der Apple //e verwendet einen Mikroprozessor des Typs 6502 (in den neueren Ausgaben des //e: 65C02). Dieser Chip hat 40 Pins ("Beinchen") und befindet sich ziemlich genau in der Mitte der Hauptplatine.

Jeder (digitale) Computer benötigt einen *Takt* ("Clock"), um einzelne Operationen miteinander zu synchronisieren. Das funktioniert ähnlich wie der Taktschlag in der Musik - allerdings geht das bei Mikroprozessoren erheblich rascher: der Apple benutzt etwas mehr als eine Million Takte pro Sekunde. Im Vergleich zu anderen modernen Mikrocomputern ist sogar das noch recht wenig: selbst den 6502 gibt es mittlerweile in einer Ausgabe, die mit 4 Megahertz (4 Millionen Taktschlägen pro Sekunde) arbeitet, andere Prozessoren arbeiten mit noch höheren Taktraten (bis zu 12 Megahertz). Ein allgemeiner Rückschluß der Art "je schneller die Taktfrequenz, desto schneller der Prozessor" ist nicht unbedingt richtig, weil verschiedene Arten von Mikroprozessoren unterschiedliche Anzahlen von Taktschlägen für die Ausführung eines Befehls benötigen. Richtig ist allerdings die Aussage, daß ein 6502 mit 2 Megahertz Taktfrequenz genau doppelt so schnell arbeitet wie mit einem Megahertz.

Der Apple //e ist so aufgebaut, daß der 6502 mit einer Vielzahl von anderen Bausteinen kommunizieren, d.h. Daten austauschen kann. Der Prozessor gibt zu diesem Zweck mit einem bestimmten Taktschlag eine *Adresse* aus. Über diese Adresse wird der jeweilige Baustein angesprochen, mit der der Prozessor Daten austauschen will. Die entsprechenden Leitungen werden als *Adreßbus* bezeichnet. Über den Adreßbus wird diese Adresse an alle Bausteine des Computers weitergegeben, mit denen der Prozessor Daten austauschen kann. Nach Ausgabe dieser Adresse werden Daten über einen weiteren Satz von Leitungen, dem *Datenbus*, übertragen - entweder von einem durch die Adreßausgabe aktivierten Baustein in Richtung Prozessor (Lesen) oder vom Prozessor zu einem Baustein (Schreiben).

Für beide Transferrichtungen werden dieselben Leitungen verwendet, der Datenbus wird deshalb als *bidirektional* bezeichnet. Damit über die Adreßausgabe angesprochene Bausteine wissen, ob der Prozessor Daten lesen oder schreiben will, existiert eine weitere Leitung mit der Bezeichnung R(ead)/W(rite) Control, deren Zustand ("0" oder "1") die Richtung des Datentransfers signalisiert. Diese Leitung wird vom Prozessor gesteuert und von allen anderen Bausteinen nur gelesen, d.h. ausgewertet.

Der 6502 hat 16 Adreßausgänge, über die der Zustand des Adreßbusses gesetzt wird. Der Adreßbus wird von allen anderen Bausteinen nur gelesen und hat also nur eine mögliche Richtung, er wird deshalb als *unidirektional* bezeichnet. Jede einzelne dieser Leitungen kann vom Prozessor auf "Spannung" (entspricht "1") oder "keine Spannung" (entspricht "0") gesetzt werden. Der Wert einer Adresse ergibt sich aus den Spannungswerten aller 16 Leitungen zusammen und geht folglicherweise von 0000 0000 0000 0000 (alle Adreßleitungen "0") bis 1111 1111 1111 1111 (alle Adreßleitungen "1"). In hexadezimaler Darstellung ergeben sich damit Adressen von $0000 bis $FFFF, die Umrechnung in Dezimalschreibweise ergibt mögliche Adreßnummern von 0 bis 65535.

Der 6502 verfügt über 8 Datenein- und -ausgänge, die je nach der gewünschten Richtung des Datenflusses entweder als Eingänge (Lesen) oder als Ausgänge (Schreiben) funktionieren. In derselben Weise wie bei den Adreßleitungen kann jede Datenleitung entweder den Zustand "1" oder "0" annehmen. Die Zustände aller acht Leitungen zusammen ergeben ein Datenwort des Prozessors, die darstellbaren Kombinationen gehen von 0000 0000 bis 1111 1111 ($00 bis $FF bzw. 0 bis 255).

Daten- und Adreßleitungen können nur einen von beiden Zuständen annehmen: entweder "Spannung" oder "keine Spannung". Diese Definition ist für digitale Computer charakteristisch und wird als "two-state" (zwei Zustände) oder als *binär* bezeichnet. Die gebräuchlichste Schreibweise zur Bezeichnung der beiden Zustände ist "0" oder "low" für "keine Spannung" und "1" oder "high" für "Spannung". Andere Möglichkeiten sind die Bezeichnungen aus/an oder falsch/wahr.

Der Zustand einer einzigen Leitung ("0" oder "1") ist die kleinste darstellbare Informationsmenge eines Computers und heißt Bit. Der 6502 führt jede Operation mit 8 Bit (einem Byte) auf einmal aus und wird deshalb als *8-Bit-Prozessor* bezeichnet.[1]

Durch die Ausgabe einer Adresse wird jeweils ein bestimmter Baustein innerhalb des Computers angesprochen, d.h. jedem Baustein ist eine eigene Elektronik zugeordnet, die auf bestimmte Adreßwerte reagiert und den Baustein aktiviert, wenn "seine" Adresse ausgegeben worden ist. Dieser Vorgang bzw. die entsprechende Elektronik wird als

---

1 Diese Definition ist nicht mehr ganz korrekt. Der Terminus 8-Bit-Prozessor bezieht sich auf die interne Verarbeitung von jeweils 8 Bit auf einmal. Im allgemeinen stimmen jedoch interne Wortbreite und Anzahl der Datenleitungen überein - allerdings gibt es inzwischen eine Reihe von Prozessoren, bei denen aus Kostengründen der Datenbus kleiner ausgefallen ist (z.B. 8088, 68008).

Adreßdekodierung bezeichnet. Jede Adresse wird innerhalb des Computers nur einmal verwendet und bezeichnet so eindeutig einen bestimmten Baustein.[2] Für Bausteine, die mehr als einen Speicherplatz haben, reagiert die Adreßdekodierung nicht nur auf eine einzelne Adresse, sondern auf einen Adreßbereich.

Der überwiegende Teil der möglichen Adressen innerhalb des Apple //e wird von Speicherbausteinen belegt. Jede dieser Adressen spricht eine *Speicherstelle* an. In diesen Speicherstellen befindet sich das Programm des Mikroprozessors (wie z.B. der Applesoft-Interpreter) oder ein Datum, das vom Prozessor bearbeitet wird. Der Prozessor verbringt rund die Hälfte der gesamten Laufzeit eines Programms damit, Programmspeicherstellen zu lesen. Die Befehle eines Programms sind sequentiell (d.h. hintereinander) in aufsteigender Folge der Adressen angeordnet. Dadurch besteht das Lesen eines oder mehrerer Befehle aus der einfachen Folge "Adreßausgabe, Einlesen eines Datenwortes, Erhöhung der Adresse um 1; Adreßausgabe...". Die restliche Zeit wird für Manipulation, Speicherung und Einlesen von Daten als Reaktion auf vorher gelesene Programmbefehle verwendet. Diese Befehle können auch die Veränderung des Adreßzählers für Programmbefehle zur Folge haben, d.h. ein "Sprung" innerhalb eines Programms besteht schlicht aus einer kontrollierten Veränderung der Adresse, von der aus der nächste Programmbefehl eingelesen wird.

Für die restlichen Adressen des Apple //e ist der Auswahlmechanismus für die Zuordnung "Adresse-Baustein" derselbe - allerdings "hängen" an den entsprechenden Adressen keine Speicherbausteine, sondern z.B. der Lautsprecher des Apple, der anstelle eines Speicherbausteins aktiviert wird und kein Datenwort speichert, sondern jeweils ein "Klick" produziert.

Speicherbausteine, Datenorganisation und Adreßdekodierung

Abgesehen von einigen Spezialanwendungen muß jeder Computer über zwei Arten von Speicherbausteinen verfügen: Bausteine, von denen nur gelesen werden kann (ROM), und andere, die sowohl Lesen als auch eine Veränderung ihres Inhalts zulassen (RAM).[3] RAM-Bausteine sind notwendig, damit man ein Programm schreiben und speichern kann, sowie für die Speicherung von Daten, die während eines Programmlaufs verändert werden. Mindestens ein ROM-Baustein ist erforderlich, damit der Prozessor nach Einschalten der Stromversorgung ein Programm zur Verfügung hat, das er abarbeiten kann.

Ein Speicherbaustein (ROM oder RAM) enthält eine Vielzahl von einzelnen Speicherzellen. Jeder dieser Bausteine hat eine Anzahl von Pins ("Beinchen"), über die eine vom Prozessor ausgegebene Adresse ins Innere des Bausteins gelangt, sowie weitere Anschlüsse, über die Daten hinein- bzw. hinaustransportiert werden. Einen Speicherbaustein kann man sich als eine (meist sehr große) Anzahl von Schubladen vorstellen; jede Schublade enthält ein Datum. Die Adreßdekodierung innerhalb des Bausteins sorgt dafür, daß nach Erhalt einer bestimmten Adresse jeweils die richtige Schublade geöffnet wird. Je nachdem, ob gelesen oder geschrieben werden soll (für die Leitung Read/Write Control gibt es einen weiteren Anschluß), wird der Inhalt der Schublade über die Datenanschlüsse ausgegeben (der Speicherbaustein bringt die Datenleitungen auf einen dem Schubladeninhalt entsprechenden Pegel) oder die an den Datenanschlüssen anliegenden "Nuller" und "Einser" werden in die Schublade hineingelegt. Nach Ende des Prozesses wird die Schublade wieder zugemacht, ihr Inhalt bleibt solange im Verborgenen, bis dieselbe Adresse erneut angesprochen wird.

Der Unterschied zwischen RAM und ROM liegt darin, daß RAM-Daten verändert werden können, in einem ROM gespeicherte Daten dagegen nicht: die "Schubladeninhalte" eines ROMs sind fest installiert. Dafür vergißt ein ROM seinen Inhalt auch dann nicht, wenn die Stromversorgung abgeschaltet wird; ein RAM behält seine Daten nur unter Spannung, sie gehen beim Ausschalten des Computers verloren.

Für den Prozessor sehen ROM und RAM exakt gleich aus - für das Schreiben eines Datenworts existiert keine Rückmeldung im Sinne von "erfolgreich gespeichert". Der Versuch, etwas in einen ROM hineinzuschreiben, ist völlig ungefährlich - allerdings wird man hinterher feststellen, daß die "beschriebene" ROM-Speicherzelle stur ihren alten Inhalt beibehalten hat.

---

2 Im Apple //e können verschiedene Bausteine abwechselnd in ein und denselben Adreßbereich geschaltet werden ("bank switching"). Aber auch hier gilt, daß zur gleichen Zeit immer nur ein Baustein aktivierbar ist - die Zuordnung eines Bausteins zu einer Adresse bzw. einem Adreßbereich ist dadurch nach wie vor zu jedem Zeitpunkt eindeutig.

3 Das Kürzel ROM steht für "Read Only Memory" (= nur-Lesespeicher) und stellt eine korrekte Bezeichnung dar; RAM steht für "Random Access Memory" (= Speicher mit wahlfreiem Zugriff) und ist eine der berühmtesten Bezeichnungen der sog. Murphy-Terminologie, d.h .schlicht falsch. Besser wäre es, RAM als "Random Alterable Memory" (= wahlfrei veränderbarer Speicher) zu interpretieren - schließlich kann auf einen ROM auch "wahlfrei", d.h. ab einer beliebigen Stelle zugegriffen werden. Die Abkürzung RAM stammt aus der Zeit der Lochkarten, die noch der Reihe nach (also nicht "wahlfrei") eingelesen werden mußten.

Speicherbausteine können in ihrer internen Organisation verschieden aufgebaut sein. So finden Sie z.B. an der vorderen rechten Ecke der Hauptplatine des Apple //e acht gleichartig aussehende RAM-Bausteine und etwas dahinter 2 breitere ROM-Bausteine. Dieses merkwürdige Verhältnis erklärt sich dadurch, daß jede "Schublade" eines der 8 RAM-Bausteine nur ein einziges Bit aufnehmen kann, eine "Schublade" der ROM-Bausteine dagegen 8 Bit auf einmal.

Da der 6502 immer 8 Bit auf einmal liest bzw. schreibt, ist die Adreßdekodierung des Apple //e so aufgebaut, daß jeder Zugriff auf den RAM des Computers alle acht RAM-Bausteine parallel aktiviert: Jeder RAM-Baustein ist mit allen Adreßleitungen, aber nur mit einer einzigen der acht Datenleitungen verbunden. Ein Zugriff auf den ROM aktiviert (je nach Wert der ausgegebenen Adresse) entweder den einen oder den anderen ROM-Baustein, wobei jeder ROM-Baustein mit allen acht Datenleitungen auf einmal verbunden ist.

Der Apple //e enthält 65536 Byte RAM-Speicherplatz. Diese Menge wird normalerweise als 64 kByte (64 * 1024 Byte) bezeichnet. Darüber hinaus ist auf der Hauptplatine die entsprechende Elektronik für den Einbau weiterer 64 kByte RAM ("AUX-RAM") in dem speziellen Steckplatz der 80-Zeichen-Karte vorgesehen. Außerdem enthält die Hauptplatine noch 16 kByte ROM-Speicherplatz - damit ergeben sich insgesamt 144 kByte. Die Lösung des Rätsels, wie man für 144 kByte mit insgesamt 65536 Adressen auskommt, liegt wiederum in der Adreßdekodierung: Derselbe Mechanismus, der bei einem Zugriff auf den RAM-Bereich alle acht RAM-Bausteine aktiviert und bei einem Zugriff auf den ROM-Bereich einen der beiden ROM-Bausteine, ist auch für die *Bankumschaltung* innerhalb dieser 144 kByte zuständig. Es sind immer nur 64 kByte auf einmal adressierbar - aber welche Bausteingruppe durch die entsprechenden Adressen angesprochen wird, bestimmt die Adreßdekodierung.

Die 64 kByte RAM auf der Hauptplatine des //e entsprechen einem Apple II mit einer 16 kByte-Erweiterung ("Language Card") in Steckplatz 0. Der durch Adressen im Bereich von $0000 bis $BFFF angesprochene RAM-Bereich entspricht dem des Apple II, durch Adressen im Bereich $D000 bis $FFFF werden entweder die oberen 16 kByte des RAMs oder der ROM angesprochen. Der Bereich von $D000 bis $DFFF existiert dreifach: entweder als "Bank 1" und "Bank 2" der oberen 16 kByte RAM oder als Teilbereich des ROMs. Im Unterschied zum Apple II schaltet ein RESET in diesem Bereich immer auf den ROM um. Wenn über die 80-Zeichen-Karte im speziellen Steckplatz noch weitere 64 kByte RAM angeschlossen sind, verhalten sich die oberen 16 kByte dieses RAMs in derselben Weise wie der RAM auf der Hauptplatine, d.h. ein Zugriff auf den Adreßbereich $D000 bis $FFFF selektiert entweder den ROM der Hauptplatine oder den AUX-RAM, wobei der Bereich $D000 bis $DFFF hier ebenfalls in "Bank 1" und "Bank 2" unterteilt ist. Ein Apple //e mit erweiterter 80-Zeichen-Karte hat somit das RAM-Äquivalent von zwei kompletten Apple II+, jeweils mit einer 16 kByte-Erweiterung in Steckplatz 0.

Der Apple //e benutzt sogenannte *dynamische* RAM-Bausteine, die periodisch einen *Refresh* ("Auffrischung") benötigen. Das bedeutet, daß jede Speicherzelle ihren Inhalt nach kurzer Zeit "vergißt" - wenn sie nicht jemand in gewissen Zeitabständen daran erinnert. Falls Ihnen das etwas absonderlich vorkommt - diese Art von Bausteinen wird in fast jedem Computer verwendet, in dem größere Mengen an Speicherplatz benötigt werden. Aus technischen Gründen, auf die wir hier nicht weiter eingehen wollen, bringt man in einem dynamisch aufgebauten RAM-Baustein ein Vielfaches an Speicherzellen unter als in seinem Gegenstück, dem statischen RAM. Der Preis dafür ist ein gewisser Mehraufwand an Elektronik, um für einen konstanten Refresh zu sorgen. Die Elektronik auf der Hauptplatine ist jedenfalls nicht nur für den Refresh der normalen 64 kByte, sondern auch für den der zusätzlichen 64 kByte AUX-RAM im speziellen Steckplatz eingerichtet.

In den beiden ROM-Bausteinen auf der Hauptplatine befindet sich die *Firmware* des Computers. Mit dieser Wortschöpfung soll zum Ausdruck gebracht werden, daß es sich dabei um ein Programm (Software) handelt, das aber unveränderlich ist und damit irgendwo zwischen Soft- und Hardware (= Elektronik des Computers) liegt. Die Firmware enthält das *Monitorprogramm* des Apple mit einigen zusätzlichen Routinen für den Betrieb der (optionalen) 80-Zeichen-Karte, den Applesoft-Interpreter sowie einen Selbsttest des Computers. Das Monitorprogramm enthält die Programmteile, die der Prozessor nach Einschalten der Stromversorgung durchläuft (*Boot* bzw. den ersten Teil davon), sowie die generellen Routinen zur Ein- und Ausgabe von Zeichen und das normalerweise als "Monitor" bezeichnete Programm, mit dem man einzelne Speicherstellen des Computers ausgeben und/oder verändern kann. Applesoft BASIC ist der fest eingebaute BASIC-Interpreter des Apple //e; die Routinen für die 80-Zeichen-Karte sind eine Erweiterung der bereits im Monitor vorhandenen Ein- und Ausgaberoutinen für den 40-Zeichen-Modus; das Selbsttestprogramm überprüft die Hauptfunktionen des Computers und die Datensicherheit des RAMs.

Die Steckplätze für Zusatzkarten

Die Steckplätze des Apple sind wohl einer der Hauptgründe für die Popularität der Apple II-Serie. Der //e besitzt nur noch sieben davon, der beim Apple II für die Speichererweiterung vorgesehene Steckplatz 0 ist entfallen - schließlich hat der //e bereits standardmäßig 64 kByte RAM. Alle sieben Steckplätze haben eine Verbindung zum Adreß- und Datenbus des Computers sowie zu den wichtigen Kontrollsignalen RESET', READY, NMI', IRQ' und DMA'.[4] Diese Signale finden Sie im Detail in Kapitel 4 besprochen. Wichtig ist hier nur, daß eine entsprechende Zusatzkarte über die Kontrolle dieser Signale den Apple mehr oder weniger auf den Kopf stellen kann: der 6502 kann von einer Zusatzkarte aus unterbrochen, angehalten und sogar völlig von der restlichen Elektronik des Computers getrennt werden. Damit ist es möglich, in einen der Steckplätze einen völlig anderen Prozessor zu installieren, dem sämtliche Funktionen des Apple zur Verfügung stehen. Eine CP/M-Karte mit einem Z80-Prozessor darauf ist zwar das populärste, aber bei weitem nicht das einzige Beispiel dafür. In die umgekehrte Richtung funktioniert das genauso: eine Zusatzkarte kann so konstruiert werden, daß sie auf entsprechende Signale des 6502 reagiert. Die Controller-Karte für die Diskettenlaufwerke beispielsweise reagiert auf RESET' und schaltet daraufhin den Motor der angeschlossenen Diskettenlaufwerke aus.

Jeder Steckplatz ist einem bestimmten Adreßbereich zugeordnet und kann vom Prozessor auf dieselbe Weise wie der RAM angesprochen werden. Für den Apple II und den //e existieren auf dem europäischen Markt mehr als 50 populäre Zusatzkarten sowie eine nicht abschätzbare Zahl von "Exoten".

Mit den Steckplätzen sind noch weitere Signale und Leitungen verbunden, die hier nicht im einzelnen aufgeführt werden. Sie betreffen das "Timing", also die Zeitkontrolle der Elektronik, die Stromversorgung und Auswahlsignale der Adreßdekodierung. Zusammengenommen ermöglichen alle Verbindungen die komplette Integration einer Zusatzkarte in die Elektronik des Apple //e.

Der spezielle Steckplatz

Im Gegensatz zu seinen Vorläufern verfügt der Apple //e über einen weiteren Steckplatz mit 60 (anstelle von 50) Kontakten, der auch durch seine Lage auf der Hauptplatine von den anderen Steckplätzen abgesetzt ist. Diesem Steckplatz fehlen einige Signale wie z.B. I/O SELECT'und eine vollständige Verknüpfung mit Daten- und Adreßbus.

Er ist für Zusatzkarten vorbereitet, die zusammen mit der Videoelektronik arbeiten bzw. ihre Funktionen erweitern, indem sie das Zeitverhalten der Bilderzeugung ändern. Genutzt wird dieser Steckplatz in den allermeisten Fällen von der speziellen 80-Zeichen-Karte des //e, die nicht nur 80 Zeichen, sondern auch doppelt hoch auflösende Grafik zur Verfügung stellt und in der Form als "erweiterte 80-Zeichen-Karte" weitere 64 kByte RAM ("AUX-RAM") enthält.

Damit verfügt der Prozessor des Apple insgesamt über 128 kByte RAM, die natürlich auch wieder über die Adreßdekodierung so selektiert werden, daß immer nur 64 kByte zur gleichen Zeit erreichbar sind. Funktionen, die sich auf die Generierung von Videosignalen beziehen, wie z.B. ein PAL-Modulator oder ein RGB-Ausgang, können durch "normale" Karten in einem der Steckplätze 1 bis 7 erreicht werden - bei der Konstruktion einer Karte, die die horizontale Auflösung der Bildschirmelektronik verdoppelt, dürften allerdings erhebliche Anstrengungen und Umwege notwendig sein.

Trotz des Fehlens verschiedener "typischer" Signale für normale Zusatzkarten scheint die Anwendung des speziellen Steckplatzes allerdings nicht auf 80-Zeichen-Karten mit und ohne Speichererweiterung beschränkt zu sein - momentan sind bereits Karten für diesen Steckplatz auf dem Markt, die außer einer 80-Zeichen-Darstellung auch noch einen Z80-Prozessor enthalten.

---

4 Das Apostroph (') nach dem Namen eines Signals zeigt an, daß die entsprechende Leitung *active low* ist: RESET' bedeutet, daß die RESET-Leitung *inaktiv* ist, wenn sie den Zustand "1" hat (Normalbetrieb), und aktiv, wenn sie den Zustand "0" hat. Wir folgen damit den im *Technical Reference Manual* für den Apple //e benutzten Konventionen.

MMU, IOU und HAL

Außer den bereits beschriebenen Funktionsblöcken CPU, RAM, ROM und der (noch nicht erwähnten) Eingabe/Ausgabe benötigt ein Mikrocomputer im Normalfall eine große Zahl kleinerer und einfacherer Chips, die für die reibungslose Zusammenarbeit der einzelnen Funktionsblöcke sorgen. (ATARI hat die entsprechenden Funktionen in der neuen ST-Serie sinnigerweise als "Glue" (= Klebstoff) bezeichnet). Im Apple //e wird der größte Teil dieser Funktionen von drei speziell für Apple, Inc. hergestellten Chips ("Custom-ICs") übernommen. Die Bezeichnungen dieser Bausteine lauten MMU ("Memory Management Unit" = Speicherverwaltungseinheit), IOU ("Input/Output Unit" = Ein-/Ausgabeeinheit) und HAL ("Hard Array Logic" = festverdrahtete Logik).

Die *MMU* enthält eine Reihe von Schaltern, die von einem Programm aus betätigt werden können und deshalb die Bezeichnung *Softswitches* (= Software-Schalter) tragen. Im Gegensatz zu einem physikalischen Schalter ("Hardswitch"?) besteht ein Softswitch aus elektronischen Bauteilen, die einer bestimmten Adresse zugeordnet sind und ebenfalls über Adreßdekodierung aktiviert werden. Das Ansprechen einer Adresse, an der ein Softswitch "hängt", bewirkt das Setzen eines bestimmten Zustands. Die meisten dieser Schalter sind mit mehreren Adressen verbunden: das Ansprechen einer Adresse setzt den Schalter "an", das Ansprechen einer zweiten Adresse setzt den Schalter "aus". Softswitches sind ein wichtiger Bestandteil des Apple-Konzepts.

Wie Sie vielleicht bereits erraten haben, kümmert sich die MMU hauptsächlich um die Adreßdekodierung und die Verwaltung des Speichers: die verschiedenen Konfigurationen (d.h. welcher Baustein auf einen bestimmten Adreßbereich reagiert) werden durch den Stand MMU-interner Softswitches gesetzt. Ein Beispiel dazu: Der Adreßbereich $D000...$FFFF kann entweder auf "RAM" oder "ROM" geschaltet werden, der dazugehörige Softswitch hat die Adressen $C080 ("ROM an, RAM aus") und $C081 ("RAM an, ROM aus"). Je nach Stand dieses Schalters aktiviert die MMU entweder den ROM oder den RAM, wenn eine Adresse im Bereich $D000...$FFFF vom Prozessor angesprochen wird.

Um diese Funktionen zu erfüllen, ist die MMU mit allen Adreßleitungen verbunden und sieht dem Prozessor quasi bei der Arbeit zu. Außerdem ist die MMU noch mit der Leitung R/W' Control verbunden und kann so im Zusammenhang mit einer gegebenen Adresse bestimmen, ob Daten zu einem Baustein gesendet oder von ihm gelesen werden sollen. Dadurch ist es z.B. möglich, daß der Prozessor Daten von einer Baugruppe liest und sie zu einer anderen Baugruppe schreibt, wobei Quelle und Ziel dieselbe *Adresse* haben. So verrückt das dazugehörige Programm auch aussieht - der Transfer größerer Datenmengen zwischen dem AUX-RAM und dem RAM der Hauptplatine ist nur so möglich, nämlich durch Setzen der Konfiguration "Lesen von AUX, Schreiben zum Haupt-RAM" oder umgekehrt.

Weitere Funktionen der MMU liegen in der Auswahl und Aktivierung von Zusatzkarten sowie in der Umwandlung von 16-Bit-Adressen in das von den dynamischen RAMs benötigte 8-Bit-Format. Wir werden uns in den Kapiteln 2 und 5 mit weiteren Details in dieser Richtung beschäftigen.

Die *IOU* ist primär für die Erzeugung der Videosignale zuständig. Dafür werden die folgenden Funktionsgruppen benötigt:

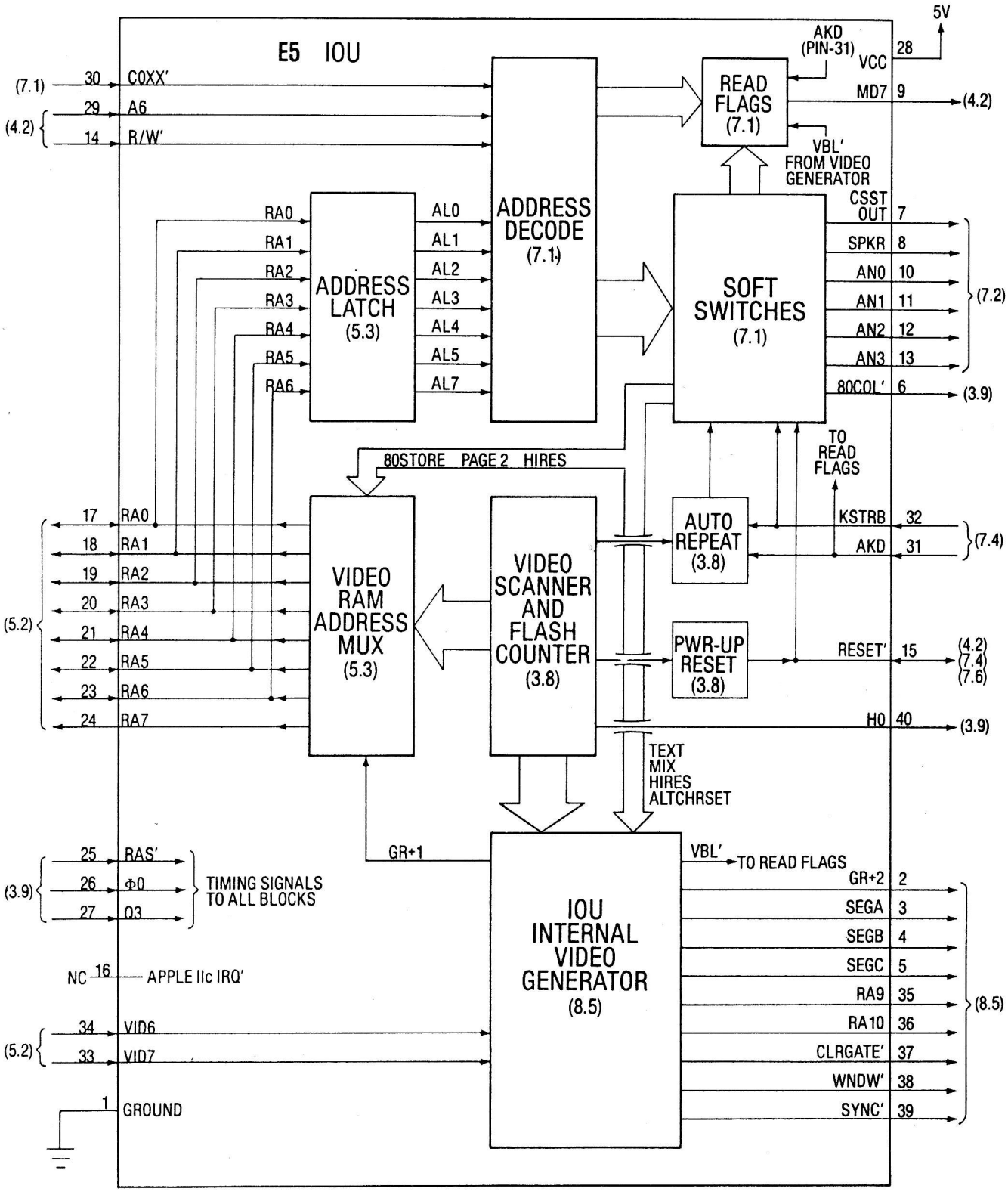
- Softswitches zum Setzen der Bildschirmmodi (TEXT, Lo-/HiRes etc.);
- ein Videoscanner, der den entsprechenden RAM-Bereich zyklisch adressiert;
- Elektronik zur Umsetzung einer Videoscanner-Adresse in das von den RAM-Bausteinen benötigte 8-Bit-Format.

Die Elektronik, die die vom RAM ausgegebenen Daten zu dem Signal VIDEO weiterverarbeitet, ist nicht in der IOU enthalten.

Beim zyklischen Durchlauf eines Speicherbereichs dürfen sich der Prozessor und die Videoscanner nicht gegenseitig in die Quere kommen, d.h. es darf immer nur einer der beiden auf den RAM zugreifen. Die IOU benutzt deshalb die Momente, in denen der Prozessor mit internen Vorgängen beschäftigt ist und nicht auf den Speicher zugreift. Dazu ist eine entsprechende Synchronisation notwendig, die hauptsächlich über die Takterzeugung und den HAL gesteuert wird.

Außerdem übernimmt die IOU die Bedienung der gesamten restlichen Ein- und Ausgabemöglichkeiten des Computers wie Tastatur, Kassettenrecorder-Schnittstelle, der "Announciators" und des Lautsprechers.

Bild 1.1 Die Funktionsblöcke der IOU und ihre Anschlußbelegung



Speziell die Schnittstelle zur Tastatur läßt sich noch in einige weitere Unterfunktionen gliedern:

- die "Autorepeat"-Funktion;
- das Signal "Any Key down" (= irgend eine Taste gedrückt);
- der Transfer des Codes einer gedrückten Taste auf den Datenbus des Computers, wenn die Tastaturschnittstelle angesprochen wird.

Einzelne Funktionen der IOU tauchen in fast sämtlichen Kapiteln dieses Buchs auf - Bild 1.1 gibt einen ungefähren Überblick über die Funktionsblöcke der IOU mit dazugehörigen Querverweisen auf die jeweiligen Abschnitte bzw. Kapitel.

Der dritte und letzte spezielle Schaltkreis auf der Hauptplatine des //e ist der HAL. Er ist für die Koordination der diversen Zeitabläufe innerhalb des Gesamtaufbaus zuständig und enthält eine große Menge "primitiver" Schaltkreise, mit denen die Zustände seiner Eingangsleitungen logisch miteinander verknüpft werden. Aus diesen Verknüpfungen werden Signale ("Ausgangsterme") zusammengesetzt, mit denen Zeittakte erzeugt und andere Baugruppen aktiviert werden. Er übernimmt z.B. einen Teil der notwendigen Synchronisation zwischen Videoscanner und Prozessor, indem er für beide Bausteine entsprechend versetzte Taktimpulse erzeugt. (Für Spezialisten: Die Abkürzung HAL steht für "maskenprogrammierter PAL-Baustein"). Details über diese Signale und ihren Zweck finden sie in Kapitel 3.

Die Videoausgabe

Die *Standardausgabe* beim Apple //e ist der Bildschirm - die Ausgabe über andere Geräte wie z.B. Kassettenrecorder oder Lautsprecher muß erst gesondert aktiviert werden, die Ausgabe eines Videosignals findet sofort nach dem Einschalten des Computers statt. Um es genau zu nehmen: man kann sie überhaupt nicht abschalten, denn sie ist sehr eng mit dem Refresh der RAMs verbunden. Die Videoelektronik erzeugt ein (Farb-)Signal, dessen Zusammensetzung der Fernsehnorm des jeweiligen Landes angepaßt ist, in dem der Computer verkauft wird.

Das vom Apple ausgegebene Videosignal kann direkt in einen farbigen oder monochromen Monitor oder in ein Fernsehgerät mit Videoeingang gespeist werden - über den Antenneneingang eines Fernsehers funktioniert das allerdings nicht, dafür wird ein dazwischengeschalteter HF-Modulator benötigt. Ein HF-Modulator erzeugt ein HF-Signal und moduliert dieses Signal mit dem Videosignal, um so dieselbe Signalart zu erzeugen wie ein Fernsehsender. Innerhalb des Fernsehers wird das HF-Signal de-moduliert, d.h. aus dem modulierten HF-Signal werden die Informationen des Videosignals wieder herausgeholt. Durch diesen doppelten Umwandlungsprozeß, zusammen mit der sowieso schlechteren Auflösung und dem stärkeren Flimmern eines Fernsehers gegenüber einem Computermonitor, haben HF-Modulatoren auch noch einen weiteren Namen bekommen, nämlich "Kopfschmerzen".

Der Apple //e verfügt über drei grundsätzlich unterscheidbare Modi der Bildausgabe: **TEXT**, **LoRes** ("Low Resolution" = niedrige Auflösung) und **HiRes** ("High Resolution" = hohe Auflösung). Im Modus TEXT werden Zeichen und Buchstaben dargestellt, im Modus LoRes (farbige) Blocks und in HiRes (farbige) Einzelpunkte. Zusätzlich sind Mischformen von Grafik und Text möglich: sowohl LoRes als auch HiRes können zusammen mit vier Zeilen Text auf den untersten Zeilen des Bildschirms dargestellt werden. Dieser "Mischmodus" wird als MIXED bezeichnet und ist für einfache Anwendungen sehr nützlich, wenn man zu einer Grafik noch Anweisungen oder Erklärungen geben will. Für die meisten ernsthaften Anwendungen sind vier Textzeilen zuwenig - die Auflösung im Modus HiRes ist aber ausreichend für die Darstellung gut lesbarer Buchstaben, und es gibt mittlerweile eine Reihe von Programmen auf dem Markt, mit denen Text auf den Grafikbildschirm gezeichnet werden kann.

Alle drei Modi können sowohl mit der vom Apple II gewohnten Auflösung (40 Zeichen TEXT, 40 LoRes-Blocks und 280 HiRes-Punkte pro Zeile) als auch mit doppelter horizontaler Auflösung (80 Zeichen TEXT, 80 LoRes-Blocks und 560 HiRes-Punkte pro Zeile) ausgegeben werden, die Voraussetzung dafür ist allerdings eine 80-Zeichen-Karte im speziellen Steckplatz.

Jeder der drei Grundmodi hat einen oder mehrere zugeordnete Speicherbereiche im RAM, deren Inhalt von der Videoelektronik zyklisch ausgelesen und in entsprechende Videosignale umgewandelt wird. Diese Speicherbereiche sind:

TEXT/LoRes	Seite 1	$0400 bis $07FF	(1 kByte RAM)
TEXT/LoRes	Seite 2	$0800 bis $0BFF	(1 kByte RAM)
HiRes	Seite 1	$2000 bis $3FFF	(8 kByte RAM)
HiRes	Seite 2	$4000 bis $5FFF	(8 kByte RAM)

Nehmen wir als Beispiel an, daß sich die Videoausgabe im Modus TEXT, Seite 1 befindet (dieser Zustand ist direkt nach dem Einschalten des Computers gegeben). In diesem Fall wird der Speicherbereich von $0400 bis $07FF rund 50 Mal pro Sekunde ausgelesen und in entsprechende Videosignale verwandelt.

Eine wichtige Konsequenz dieser Art des Bildschirmaufbaus ist, daß die Bildschirmbereiche innerhalb des RAM Programmspeicherplatz benötigen und man (speziell bei der HiRes-Grafik) darum "herumprogrammieren" muß - im Gegensatz zu einer Reihe anderer Computer, bei denen der Bildspeicher einen eigenen RAM-Bereich hat, der z.B. über eine Portadresse angesprochen wird.

Für die Unterscheidung zwischen "Seite 1" und "Seite 2" existiert ein eigener Softswitch mit dem Namen PAGE2, dessen Stellung für *alle* Video-Modi gilt. Nach Einschalten der Stromversorgung und nach einem Druck auf RESET ist Seite 1 aktiv, d.h. der Schalter PAGE2 ist zurückgesetzt.

Alle Bildschirmmodi mit einfacher Auflösung benutzen nur den RAM der Hauptplatine. Während jedes Prozessorzyklus wird ein Byte des Bildspeicherbereichs gelesen, für eine Zeile auf dem Monitor sind es insgesamt 40 Byte. Ausgehend von diesen 40 Byte pro Zeile werden wir die dazugehörigen Modi in diesem Buch als **TEXT40**, **LoRes40** und **HiRes40** bezeichnen, wenn eine Unterscheidung zu den Modi der doppelten Auflösung notwendig ist. Aus Gründen, die im nächsten Absatz erklärt werden, bezeichnen wir die Modi der doppelten Auflösung als **TEXT80**, **LoRes80** und **HiRes80**.

Alle Bildmodi mit doppelter Auflösung benutzen zum einen den bereits gezeigten RAM-Bereich auf der Hauptplatine, zum anderen exakt denselben Bereich innerhalb des AUX-RAMs. Während jedes Zyklus des Prozessors wird erst ein Byte des AUX-RAMs, danach ein Byte des Hauptplatinen-RAMs gelesen, beide Bytes haben dabei dieselbe Adresse. Auf diese Weise werden in derselben Zeit 80 anstelle von 40 Byte zu Videosignalen verarbeitet. Wenn man diese 80 Byte pro Zeile von 0 bis 79 durchnumeriert, dann befinden sich alle Bytes mit einer geraden Nummer im AUX-RAM, die Bytes mit einer ungeraden Nummer befinden sich im RAM der Hauptplatine.

Konsequenterweise ist die Benutzung der doppelt hohen Auflösung nur mit einem zusätzlichen Speicher möglich (der in dem speziellen Steckplatz installiert sein muß). Dabei kann man zwei Arten von Zusatzspeichern unterscheiden:

- Die "normale" 80-Zeichen-Karte enthält nur 1 kByte Speicher und ermöglicht damit die Darstellung von 80 Zeichen Text. Wenn man die Kontaktstifte 50 und 55 auf dieser Karte miteinander verbindet, ist der Modus LoRes80 ebenfalls möglich.
- Eine "erweiterte" 80-Zeichen-Karte enthält dagegen 64 kByte RAM, mit denen alle drei Modi in doppelter Auflösung möglich sind.[5]

Das zyklische Absuchen ("scanning") des Bildspeichers wird, wie bereits gesagt, von der IOU übernommen - der Prozessor wäre dafür auch um ein Mehrfaches zu langsam. Die IOU enthält einen Zähler, dessen Stand der Adresse des nächsten "Videobytes" im Bildspeicher entspricht und über den das Synchronisierungssignal für den Videoausgang erzeugt wird. Dieser Zähler durchläuft den durch den Videomodus gesetzten Bildspeicherbereich zyklisch und wird als *Videoscanner* bezeichnet.

Der Videoscanner spricht den Bildspeicherbereich in einer Weise an, die völlig "transparent" für den Prozessor ist, d.h. der Prozessor merkt nichts davon. In der ersten Hälfte jedes Prozessorzyklus liefert der Scanner die Adresse des nächsten "Videobytes" und "füttert" damit die Bilderzeugung - bei doppelter Auflösung werden dabei der RAM der Hauptplatine und der AUX-RAM im selben Zeitraum gelesen. Die gelesenen Daten werden zwischengespeichert und bis zum Ende des Prozessorzyklus unabhängig vom Prozessor in Videosignale verwandelt. Der 6502 führt Speicherzugriffe nur in der zweiten Hälfte jedes Prozessorzyklus aus. Durch diese Aufteilung stören sich die beiden Funktionen nicht gegenseitig.

Vom Standpunkt des Programmierers aus besteht die Erzeugung eines Bildes aus dem Setzen verschiedener Softswitches, mit denen der gewünschte Bildmodus eingestellt wird, sowie aus dem Berechnen der Speicheradressen einzelner "Bildbytes" und ihrer Veränderung.

---

5 Auf Hauptplatinen der Revision "A" ist die doppelt hoch auflösende Grafik auch mit einer erweiterten 80-Zeichen-Karte nicht möglich.

Im Modus **TEXT** werden einzelne Zeichen durch ihren ASCII-Wert innerhalb des Bildspeicherbereichs repräsentiert. Zusätzlich zum ASCII-Wert enthält jedes Byte noch eine Information über die Darstellungsart (NORMAL, INVERSE oder FLASH). Jedes Zeichen belegt dabei ein Byte. Der gespeicherte ASCII-Wert jedes Zeichens bildet eine Adresse innerhalb des Zeichen-ROMs, in dem für jedes ASCII-Zeichen eine Punktmatrix mit 5 * 7 Punkten gespeichert ist. Für NORMAL und INVERSE existieren dabei getrennte Speicherbereiche innerhalb des Zeichen-ROMs, die zueinander invertierte Punktmatrizen enthalten. Wir werden später noch einmal darauf zurückkommen. Der Apple //e verfügt über 96 darstellbare TEXT-Zeichen, die NORMAL und INVERSE dargestellt werden können. 64 davon können zusätzlich im Modus FLASH dargestellt werden. Beim verbesserten Apple //e enthält der Zeichen-ROM einen //c-kompatiblen Zeichensatz (s. Bild 8.8).

Der TEXT-Bildschirmaufbau besteht im Modus der einfachen Auflösung aus 24 Zeilen mit jeweils 40 Spalten, bei doppelter Auflösung aus 24 Zeilen mit jeweils 80 Spalten.

Die 80-Zeichen-Darstellung auf dem Apple //e ist so ausgeführt, daß sich der Computer wie ein Apple II mit einer eingebauten 80-Zeichen-Karte in Steckplatz 3 verhält. Diese Simulation geht soweit, daß man den //e als "40-Zeichen-Computer" mit einer zusätzlichen Fähigkeit zur Darstellung von 80 Zeichen betrachten kann - der //e schaltet beim Einschalten der Stromversorgung auf den 40-Zeichen-Modus und bleibt in diesem Modus solange, bis explizit die Aktivierung der 80-Zeichen-Karte befohlen wird.

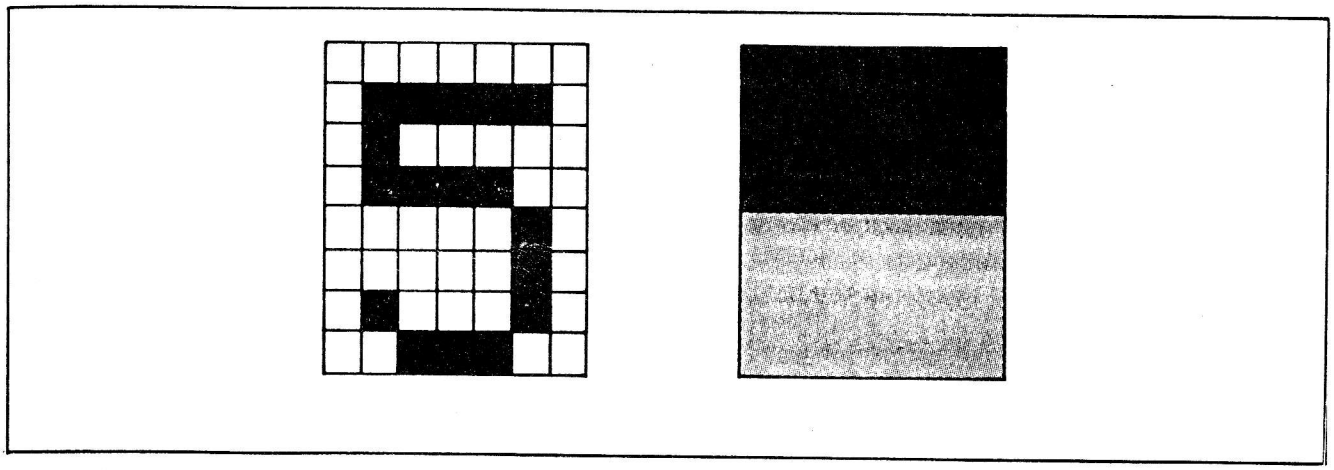
Die **LoRes-Grafik** stellt dem Programmierer einen Bildschirm zur Verfügung, der auf 48 Zeilen entweder 40 (normale Auflösung) oder 80 Spalten pro Zeile enthält (doppelte Auflösung). In jede dieser Spalten kann ein farbiger Block gesetzt werden. Technisch gesehen stehen 16 Farben zur Verfügung - praktisch sind es aber nur 15, weil die beiden Graufarben zwar durch unterschiedliche Muster im Zeichen-ROM erzeugt werden, aber auch auf einem Farbbildschirm exakt gleich aussehen. Wir werden trotzdem bei der Bezeichnung "16 Farben" bleiben.

Die LoRes-Grafik benutzt denselben Speicherbereich wie TEXT - das Arbeitsprinzip des Scanners ist also in beiden Fällen gleich. Die aus den "Bildschirmbytes" erzeugte Adresse im Zeichen-ROM wird um einen "Offset" erhöht, dadurch werden nicht Zeichenmatrizen, sondern Punktmuster adressiert. Um eine von 16 Farben zu kodieren, braucht man 4 Bit, deshalb lassen sich im Speicherplatz für ein ASCII-Zeichen (1 Byte = 8 Bit) zwei LoRes-Blöcke unterbringen und die Auflösung der LoRes-Grafik ist bei gleichem Speicherplatzbedarf doppelt so hoch wie im TEXT-Modus. Es besteht ein direkter Zusammenhang zwischen der Position eines "Buchstabenbytes" und zweier LoRes-Blöcke, der in Bild 1.2 dargestellt ist.

Im Modus **HiRes40** besteht der Bildschirm aus 192 Zeilen mit jeweils 280 Einzelpunkten. Abgesehen von der Farbdarstellung ist der Zusammenhang für den Programmierer erfreulich einfach: Für jedes gesetzte Bit im entsprechenden Speicherbereich wird ein heller, für jedes nicht gesetzte Bit ein dunkler Punkt auf dem Bildschirm erzeugt. Tatsächlich findet hier ebenfalls eine Umsetzung der "Bildschirmbytes" über den Zeichen-ROM statt, die aber den Programmierer nicht zu interessieren braucht. Jedes "Bildschirmbyte" erzeugt 7 Punkte aus D0..D6, das achte Bit wird für die Farberzeugung gebraucht. Jede Bildzeile besteht aus 40 Byte - damit ergeben sich 7 * 40 = 280 Bildschirmpunkte pro Zeile. Anstelle einer Punktmatrix, über die jedem Byte des TEXT-Bildschirms eine Matrix mit insgesamt 8 untereinanderliegenden (Video-)Zeilen zugeordnet wird, "bringt" hier jedes Byte nur eine einzige Zeile auf dem Monitor, und wir brauchen deshalb 8 * 40 Byte, um den Raum einer TEXT-Zeile auf dem Bildschirm auszufüllen. Die Auflösung in vertikaler Richtung ergibt sich damit zu 8 * 24 = 192 Bildpunkten. Soweit zu den einfacheren Dingen.

Das achte Bit jedes Bytes hat eine erheblich kompliziertere Funktion - es bestimmt die *Lage* der restlichen 7 Bits im Bild. Auf einem monochromen Monitor kann man durch Setzen dieses Bits die restlichen 7 Bildpunkte um eine halbe Punktposition verschieben, die theoretische Auflösung der einfachen HiRes-Grafik beträgt damit 560 Bildpunkte pro Zeile. Auf einem Farbbildschirm dagegen bestimmt das achte Bit eine von zwei *Farbgruppen* mit jeweils vier Farben, die ihrerseits durch die bitweise Belegung innerhalb der restlichen sieben Bit jedes Bytes bestimmt werden. Die Farbe eines einzelnen Bildpunkts ist damit sowohl vom Zustand des achten Bits als auch von der "Lage" des Punktes innerhalb des entsprechenden Bytes abhängig. So können z.B. die Farben grün und orange immer nur auf ungeraden, die Farben violett und blau nur auf geraden Horizontalpositionen gezeichnet werden. Die Auflösung für farbige Darstellung sinkt damit auf 140 * 192. Der Versuch, eine grüne Linie (Farbgruppe 0) über einen blauen Hintergrund (Farbgruppe 1) zu zeichnen, ergibt Merkwürdiges: in der Umgebung der Linie färbt sich der Hintergrund orange, weil durch das Zeichnen der grünen Linie die achten Bits gelöscht werden und damit das gesamte Byte von Farbgruppe 1 auf 0 wechselt.

Bild 1.2 TEXT- und LORES-Grafik



HiRes80 stellt einen Bildschirm mit 560 * 192 Punkten und beseitigt die vorher angesprochenen Probleme: im Schwarzweiß-Betrieb kann jeder einzelne Bildpunkt unabhängig von den anderen programmiert werden, die Position eines Punktes wird nicht mehr vom achten Bit beeinflußt. Je nachdem, wieviele Farben verwendet werden, geht die erreichbare Auflösung von 140 * 192 Punkten mit allen 16 LoRes-Farben (!) in verschiedenen Stufen bis 560 * 192 (monochrom).

Diese kurze Beschreibung enthält gerade genug Informationen, um die Komplexität des Themas erkennen zu lassen. In Kapitel 8 werden wir etwas mehr Licht in diese Abgründe bringen. Im Moment sollte die grobe Zusammenfassung ausreichen, daß die erreichbare Auflösung im HiRes-Modus des Apple //e von 140 * 192 bis 560 * 192 geht und davon abhängt, wieviele Farben benutzt werden und ob HiRes40 oder HiRes80 aktiv ist.

Wie bereits gesagt, wird für HiRes ein erheblich größerer Speicherbereich benötigt als für TEXT und LoRes: in normaler Auflösung werden 8192 Byte auf der Hauptplatine belegt, in doppelter Auflösung kommen noch einmal 8192 Byte innerhalb des AUX-RAMs dazu.

Die Tastatur

Von einigen speziellen Eingabegeräten wie der Maus und anderen einmal abgesehen, ist die Tastatur die einzige Möglichkeit für den Benutzer, seine Wünsche dem Computer verständlich zu machen. Der Prozessor des Apple verbringt folglicherweise auch den größten Teil seines Lebens in einer recht einfachen Routine mit dem Namen KEYIN (oder GETKEY, falls die 80-Zeichen-Karte aktiviert ist) und tut dabei nichts anderes, als auf einen Tastendruck zu warten. Damit es ihm nicht gar zu langweilig wird, erhöht er für jede Abfrage der Tastatur noch eine Zufallszahl um eins und bringt (auf der 80-Zeichen-Karte) einen Cursor zum Blinken. Pro Betriebsstunde kommen dabei immerhin rund 165 Millionen Tastaturabfragen zusammen, und falls Sie jemand fragt, was ihr Apple denn so tut, können Sie mit ruhigem Gewissen anworten: "Hauptsächlich prüft er, ob eine Taste gedrückt worden ist".

Die Tastatur hat 63 Tasten, welche die Buchstaben des Alphabets, die Zahlen von 0 bis 9, Sonderzeichen und einige Sonderfunktionen repräsentieren. Wenn der //e auf amerikanischen Zeichensatz umgeschaltet ist, entspricht die Tastaturbelegung der von IBM vor Urzeiten hergestellten Schreibmaschine Selectric. Über die IOU ist eine Autorepeat-Funktion realisiert, die den Code einer Taste in schneller Folge wiederholt, wenn diese Taste für längere Zeit gedrückt wird, und es kann von einem Programm aus festgestellt werden, ob irgendeine Taste gedrückt worden ist, ohne das entsprechende Zeichen "abholen" zu müssen. Die Tastatur ist mit einem sogenannten *N-Key Rollover* ausgerüstet, d.h. wenn mehrere Tasten gleichzeitig gedrückt werden, kommt dabei (im Gegensatz zu den früheren Ausgaben des Apple II) kein Unsinn heraus, sondern die Zeichen werden der Reihe nach dem Programm zur Verfügung gestellt. "Gleichzeitiges" Drücken heißt für die IOU immer noch, daß zwischen dem Kontaktschluß einer ersten und dem einer zweiten Taste ein paar hundert Mikrosekunden vergehen, und das ist ausreichend, um diese beiden Tasten "hintereinander" zu erkennen.

Über die meisten Tasten läßt sich direkt ein dazugehöriges ASCII-Zeichen erzeugen, das durch gleichzeitiges Drücken von SHIFT entsprechend verändert werden kann. Da die Tastatur "echte" ASCII-Zeichen produziert und der Bildschirm ebenfalls ASCII-Zeichen verwendet, ist die Ausgabe eines eingetippten Zeichens eine recht einfache Angelegenheit, die von den eingebauten Routinen der Firmware erledigt wird. (Es gibt immer noch Computer anderer Hersteller, bei denen das nicht der Fall ist, d.h. Tastaturcodes und/oder Bildschirmcodes nicht dem ASCII entsprechen, was vom Programmierer ausgebadet werden muß!)

Die restlichen Tasten haben spezielle Funktionen, als da wären: ESC, DELETE, RESET, TAB, CONTROL, RETURN, SHIFT, CAPS LOCK, offener und geschlossener Apfel und die vier Pfeiltasten.

ESC, DELETE, TAB, RETURN und die Pfeiltasten erzeugen dabei ebenfalls ASCII, allerdings nur durch Übersetzung in einem Programm, wobei ESC und DELETE die einzigen Möglichkeiten sind, die dazugehörigen Codes zu erzeugen. Die Codes der restlichen Tasten lassen sich auch noch auf andere Weise, nämlich durch CONTROL-I, CONTROL-M, CONTROL-U, CONTROL-H, CONTROL-J und CONTROL-K erzeugen. Alle diese Codes erzeugen keine "druckbaren" Zeichen und müssen von einem Eingabeprogramm interpretiert werden, sie werden *Steuerzeichen* genannt.

Die Taste RESET hat keinen ASCII-Wert: Wenn sie zusammen mit der Taste CONTROL gedrückt wird, bringt die IOU die Leitung RESET' auf aktiven Pegel (also "0"). Die Folge davon ist, daß nicht nur der 6502, sondern auch alle Softswitches der MMU und einige Softswitches der IOU zusammen mit allen Zusatzkarten, die auf RESET' reagieren, zurückgesetzt werden.

Die beiden SHIFT-Tasten, CAPS LOCK und CONTROL erzeugen ebenfalls keine ASCII-Zeichen, sie werden direkt von der IOU gelesen und verändern von der IOU ausgegebene Tastaturcodes für eingegebene "normale" Zeichen.

Die Tasten "offener Apfel" und "geschlossener Apfel" sind eigentlich keine Tasten im direkten Sinne und haben keine Verbindung zur IOU - stattdessen entsprechen sie den Tastern 0 und 1 der Spielsteuerungen, die über den GAME I/O (normalerweise zusammen mit einem Joystick) angeschlossen werden können. Sie sind für den Apple //e in das Tastenfeld mit aufgenommen worden, um eine Alternative für "CONTROL-Funktionen" zu bieten: ein Programm kann über das Lesen der Spielsteuerungseingänge bestimmen, ob eine dieser Tasten gedrückt wurde, und daraufhin entsprechend reagieren.

Derselbe Schalter an der Unterseite des Gehäuses, der die Bildschirmausgabe vom amerikanischen Zeichensatz auf einen nationalen Zeichensatz umschaltet, bewirkt auch eine Umschaltung der Tastatur von "amerikanischer" auf "nationale" Belegung.[6] Hinter dieser Umschaltung von einem guten Dutzend Tasten mit einem einzigen Schalter steckt wieder einmal ein ROM: Alle Codes, die durch Tastendrücke erzeugt werden können, sind im *Tastatur-ROM* gespeichert, die Zuordnung "Taste - erzeugter Code" geschieht auch hier wieder über Adressen, deren Wert ("Adreßlage") über diesen Umschalter verändert wird.

Die Tastatur des //e enthält keinen separaten Zehnerblock - allerdings befindet sich neben dem Anschluß der normalen Tastatur auf der Hauptplatine ein weiterer Steckverbinder, über den eine externe Tastatur angeschlossen werden kann. Die Erzeugung der Tastaturcodes für eine externe Tastatur geschieht ebenfalls über den Tastatur-ROMs, so daß durch eine entsprechende Umprogrammierung des ROMs auch auf einem angeschlossenen Zehnerblock beliebige Codes erzeugt werden können.

Weitere I/O

Die Abkürzung I/O steht für "Input/Output" (= Eingabe/Ausgabe). In diesem Abschnitt wollen wir uns nur auf weitere Funktionen der Hauptplatine beziehen, d.h. uns nicht dem schier unerschöpflichen Thema der Ein-/Ausgabe über Zusatzkarten wie Modems, Druckerschnittstellen, A/D- und D/A-Wandler, Digitizer, Synthesizer etc.,etc. widmen.

Außer den bereits besprochenen Funktionsblöcken "Video" und "Tastatur" enthält der Apple //e auf der Hauptplatine noch einige weitere *serielle* Ein- und Ausgabemöglichkeiten. Eine kurze Erklärung des Begriffs "seriell": Die Eingabe von der Tastatur findet *parallel* statt, d.h. die durch einen Tastendruck erzeugten ASCII-Zeichen werden

---

6 Der ab Januar 1986 verfügbare, verbesserte Apple //e hat eine leicht geänderte Tastaturbelegung und schaltet nur noch 6 Zeichen um - so bleibt z.B. bei der deutschen Version auch im amerikanischen Zeichensatz das "Z" neben dem "T".

"auf einmal" eingelesen. Ein ASCII-Zeichen wird durch 7 Bit dargestellt, folglicherweise braucht man dazu (mindestens) 7 Leitungen, auf denen genauso wie auf dem Datenbus jeweils ein Pegel liegt, der ein einzelnes Bit repräsentiert. Im Gegensatz dazu benötigt man für ein serielles Signal (theoretisch!) nur eine einzige Leitung, mit der einzelne Bits *nacheinander* übertragen werden. Im Prinzip stellt auch das Videosignal eine serielle Form der Datenübergabe dar.

Nach dieser Definition können wir kurz auf eine Ein- und Ausgabemöglichkeit eingehen, deren dazugehörige Elektronik sich zwar nicht auf der Hauptplatine des Apple //e befindet, die aber trotzdem praktisch zum Standard gehört: gemeint ist hier die Ein- und Ausgabe zu einem oder mehreren Diskettenlaufwerken. Dieser Datentransfer funktioniert über eine Zusatzkarte in einem der Steckplätze. Der Datenaustausch zwischem dem Prozessor und der Zusatzkarte findet parallel, der Austausch zwischen der Zusatzkarte und einem angeschlossenen Diskettenlaufwerk findet seriell statt. Die Kontrolle der Ein- und Ausgabe zu den Diskettenlaufwerken erfordert ein sehr umfangreiches Programm, das bekannteste trägt den Namen DOS 3.3, die neueste Version eines Disketten-Betriebssystems ("Disk Operating System") hat einige erweiterte Möglichkeiten und heißt ProDOS.

Der Apple //e verfügt über elf serielle I/O-Ports, d.h. über elf serielle Ein- und/oder Ausgänge sowie über vier analoge Eingänge.

- Der *Lautsprecherausgang* ist eine serielle Verbindung, bei der jedes "Datenwort" tatsächlich nur aus einem einzigen Bit besteht;
- Die Ein- und Ausgänge zum *Kassettenrecorder* sind echte serielle Verbindungen - hier werden größere Datenmengen Bit für Bit übertragen, wobei die erforderliche Umwandlung über Routinen des Monitorprogramms geschieht;
- Die restlichen Ausgänge des Apple arbeiten alle mit TTL-Pegeln.[7]

In der rechten hinteren Ecke der Hauptplatine befindet sich ein 16-poliger IC-Sockel, der als *GAME I/O* bezeichnet wird. Er ist primär für den Anschluß von Joysticks mit dazugehörigen "Action-Tasten" gedacht, enthält darüber hinaus aber noch die folgenden Ausgänge:

- die "Announciators", insgesamt vier an der Zahl. Der elektrische Zustand der Announciators ("0" oder "1") kann über dazugehörige Softswitches gesetzt werden.
- den "Utility Strobe", einen Ausgang, der permanent den Pegel "1" hat. Durch Ansprechen eines dazugehörigen Softswitches wird ein Impuls vom 0,5 Mikrosekunden Dauer ausgelöst, d.h. der Ausgang geht für diese Zeit auf "0" und sofort danach wieder auf "1".

Außerdem enthält der GAME I/O noch sieben Eingänge:

- drei davon reagieren auf TTL-Pegel (d.h. 0 oder 5 Volt). Zwei dieser drei Eingänge werden normalerweise als Action-Tasten zusammen mit einem angeschlossenen Joystick verwendet. Diese beiden Eingänge sind mit den Tasten "offener Apfel" und "geschlossener Apfel" verbunden. Der dritte Eingang wäre normalerweise als erste Action-Taste zusammen mit einem zweiten Joystick verbunden, ein vierter Eingang (die zweite Taste für den zweiten Joystick) fehlt. Für diesen dritten Eingang gibt es ebenfalls eine Spezialfunktion: wenn auf der Hauptplatine die mit X6 bezeichnete Lötverbindung hergestellt wird, setzt jeder Tastendruck auf SHIFT diesen dritten Eingang. Der Trick war beim Apple II notwendig, damit ein Programm erkennen konnte, ob die Taste SHIFT gedrückt wurde, und wird als "Shift-Key-Modifikation" bezeichnet.
- die restlichen vier Eingänge reagieren nicht auf TTL-Pegel, sondern auf daran angeschlossene veränderbare Widerstände. An jeden dieser Eingänge kann ein *Paddle* bzw. an jeweils zwei dieser Eingänge ein Joystick angeschlossen werden. Beide Geräte enthalten nichts weiter als Potentiometer, also veränderbare Widerstände. Der Apple enthält dazu einen Zeitgeber mit einem Kondensator, der für jeden Lesevorgang aufgeladen und über einen angeschlossenen Widerstand wieder entladen wird. Je kleiner der Widerstand des Potentiometers ist, desto schneller geht die Entladung - je größer er ist, desto länger dauert es. Das Monitor-

7 TTL steht für "Transistor-Transistor Logik" und damit für die Art und Weise, wie die meisten heutzutage käuflichen Chips kleinerer Dimension intern aufgebaut sind. Hochintegrierte Chips wie der 6502, die MMU und die IOU sind intern mit einer anderen Technik aufgebaut, sämtliche Ein- und Ausgänge halten sich aber ebenfalls an die TTL-Spezifikation: eine "0" wird durch einen Spannungswert zwischen 0 und 0,8 Volt, eine "1" durch einen Spannungswert zwischen 2,4 und 5 Volt dargestellt. Spannungen, die zwischen 0,8 und 2,4 Volt liegen, gelten als undefiniert.

programm enthält eine Routine, die diese Zeit mißt und darüber den Stand eines angeschlossenen Potentiometers feststellen kann. Ein Paddle enthält ein einziges, ein Joystick enthält zwei Potentiometer. Ein Potentiometer wird dabei durch horizontale, das andere durch vertikale Bewegungen des Joysticks eingestellt. Die Anwendungen sind dabei nicht auf Joysticks beschränkt - mit demselben Funktionsprinzip können ebenfalls Grafiktabletts und andere Eingabegeräte angeschlossen werden.

Die vier Analogeingänge und die drei TTL-Eingänge für die Action-Tasten sind zusätzlich noch über einen neunpoligen Steckverbinder auf der Rückseite des Gehäuses herausgeführt. Es ist demzufolge möglich, z.B. einen Joystick über diesen Steckverbinder anzuschließen und gleichzeitig noch ein anderes Gerät über die vom Joystick nicht belegten Anschlüsse des GAME I/O.

Die Stromversorgung

Was hierzulande aus der Steckdose kommt, ist 220 Volt Wechselspannung, auf die ein Prozessor bestenfalls mit heftigen Rauchzeichen reagiert. Die hauptsächlich benötigte Spannung in einem Computer ist +5 Volt Gleichspannung, für die entsprechende Umwandlung ist das Netzteil zuständig.

Das Netzteil des Apple //e liefert vier Gleichspannungen: +5, +12, -5 und -12 Volt (bezogen auf die Masse des Systems). Diese Spannungen sind praktisch überall auf der Hauptplatine vorhanden und zu jedem Chip geführt, der sie benötigt. Außerdem haben alle Steckplätze außer dem speziellen Steckplatz eine Zuführung sämtlicher vier Spannungen, der spezielle Steckplatz stellt nur die Spannung +5 Volt zur Verfügung.

Zusammenfassung

Der Apple //e ist ein Mikrocomputer mit einem Mikroprozessor des Typs 6502 und stellt eine Modernisierung und Erweiterung des Apple II dar. Auf der (einzigen) Hauptplatine sind Speicher und Ein-/Ausgabefunktionen wie Tastatureingang und Videoausgang enthalten. Die Erweiterungen gegenüber dem Apple II schließen die Darstellung von Kleinbuchstaben, Bildausgabe mit 80 Spalten und eine Speichererweiterung von 48 bzw. 64 kByte auf 128 kByte ein (der Apple II verfügt nur über Großbuchstaben, 40 Spalten und 48 bzw. 64 kByte RAM).

Die Hauptplatine des Apple //e enthält sieben Erweiterungssteckplätze sowie einen speziellen Steckplatz und ermöglicht den fast beliebigen Ausbau des Computers. Eingesteckte Zusatzkarten sind mit den folgenden Signalen verbunden bzw. können diese Signale kontrollieren: IRQ', NMI', RESET' und DMA'. Über das Signal DMA' ist es einer Zusatzkarte möglich, den normalen Prozessor des Apple zeitweise oder vollständig von der restlichen Elektronik des Computers abzutrennen und an seiner Stelle einen anderen Prozessor einzusetzen.

Die Kontrolle sämtlicher Konfigurationsmöglichkeiten, Softswitches und Ein-/Ausgaben findet über Adreßdekodierung statt.

Von der Hauptplatine existieren zwei Versionen - die eine enthält einen Videogenerator, der Signale nach der amerikanischen NTSC-Norm generiert, der Videogenerator der anderen Version produziert Signale, die der europäischen PAL-Norm entsprechen.

Der Videoausgang kann direkt an einen monochromen oder farbigen Videomonitor oder einen Ferseher mit Videoeingang angeschlossen werden. Über einen zusätzlichen HF-Modulator ist der Anschluß über den Antenneneingang eines normalen Fernsehers möglich, wenn auch nicht zu empfehlen.

Die Darstellung von Text kann in gemischter Groß- und Kleinschreibung erfolgen, die Zeichen sind mit einer 5 * 7 Matrix aufgebaut. Alle verfügbaren 96 Zeichen können sowohl hell auf dunklem Hintergrund als auch dunkel auf hellem Hintergrund dargestellt werden, bei 64 dieser Zeichen ist zusätzlich blinkende Darstellung möglich (FLASH mit Kleinbuchstaben ist nicht definiert). Der verbesserte Apple //e enthält anstelle blinkender Großbuchstaben grafische Sonderzeichen ("Maus-Zeichensatz").

Der Apple //e wird in den jeweiligen Ländern mit einem entsprechenden Zeichensatz und einer national angepaßten Tastaturbelegung ausgeliefert, die bei Bedarf auf den amerikanischen Standard umschaltbar ist.

Im Modus TEXT sind bei einfacher Auflösung 40, bei doppelter Auflösung 80 Zeichen auf 24 Zeilen verfügbar.

Folgende Grafikmodi stehen zur Verfügung:

- LoRes40 mit 40 * 48 Blocks in 16 Farben;
- LoRes80 mit 80 * 48 Blocks in 16 Farben;
- HiRes40 mit 140 * 192 Punkten in 8 Farben bzw. 280 * 192 Punkten schwarzweiß;
- HiRes80 mit 140 * 192 Punkten in 16 Farben bzw. 560 * 192 Punkten schwarzweiß.

Die doppelt auflösenden Modi sind nur mit einer 80-Zeichen-Karte im speziellen Steckplatz möglich, die doppelt hoch auflösende Grafik nur dann, wenn die 80-Zeichen-Karte über zusätzliche 64 kByte RAM verfügt.

Die Tastatur des Apple //e besteht aus 63 Tasten, deren Belegung zwischen dem amerikanischen Standard und einem nationalen Zeichensatz umschaltbar ist. Sie enthält zusätzliche Funktionen wie Autorepeat und N-Key Rollover und ist somit für die meisten denkbaren Anwendungen geeignet. Zusätzlich ist auf der Hauptplatine ein separater Steckverbinder für die Installation eines numerischen Zehnerblocks vorhanden.

Der 6502-Prozessor des Apple //e arbeitet mit 1.0205 MHz.

Die Hauptplatine enthält 65536 Byte dynamischen RAM und ist für die Kontrolle und Versorgung weiterer 65536 Byte RAM auf einer Zusatzkarte im speziellen Steckplatz eingerichtet. Innerhalb von 16 kByte Firmware befinden sich die Programmiersprache Applesoft BASIC, das Monitorprogramm und zusätzliche Routinen für den Betrieb mit 80 Zeichen.

Alle Speicherbereiche, die für die Speicherung von TEXT und/oder Grafik verwendet werden, befinden sich innerhalb des normalen Adreßraums (d.h. sind nicht über Ports o.ä. abgetrennt) und stehen deshalb als Programmspeicherplatz zur Verfügung, wenn der entsprechende Videomodus nicht benutzt wird. Die Kehrseite dieses Verfahrens liegt darin, daß man eventuell um diese Bereiche herumprogrammieren muß.

Für die Modi der doppelt hohen Auflösung wird der RAM mit der doppelten Taktfrequenz des Prozessors ausgelesen, also mit 2 MHz. Durch das zyklische Durchlaufen eines Speicherbereiches für die Videoausgabe wird der notwendige Refresh der dynamischen RAMs sichergestellt.

Außer Prozessor, RAM und ROM befinden sich drei weitere hochintegrierte Schaltkreise auf der Hauptplatine: die MMU enthält und kontrolliert die Adreßdekodierung und die Aktivierung verschiedener Speicherbereiche, die IOU enthält die Videoauslese, die Schnittstelle zur Tastatur und sämtliche anderen Funktionen zur Ein- und Ausgabe, der HAL koordiniert die zeitlichen Abläufe des Systems und sorgt so für eine Synchronisation der Hauptfunktionsblöcke.

Der Prozessor hat die programmgesteuerte Kontrolle über die Speicherkonfiguration und die Bildmodi, beide werden über das Setzen von programmierbaren Softswitches bestimmt.

Zusätzlich zu den Grundfunktionen Tastatur, Video und den potentiellen I/O-Möglichkeiten über die Steckplätze verfügt der Apple //e noch über Ein- und Ausgabemöglichkeiten zum eingebauten Lautsprecher, einem angeschlossenen Kassettenrecorder und einer oder mehreren Spielsteuerungen, die auch für andere Zwecke benutzt werden können. Alle diese Funktionen sind zusammen mit mehreren TTL-Ausgängen auf der Hauptplatine integriert.

Zwei der TTL-Eingänge können durch Druck auf die Tasten "offener Apfel" bzw. "geschlossener Apfel" gesetzt werden.

Das Netzteil des Apple //e stellt die Spannungen +5, +12, -5 und -12 Volt zur Verfügung. Diese Spannungen sind praktisch an jeder Stelle der Hauptplatine sowie an allen Steckplätzen außer dem speziellen Steckplatz verfügbar, der spezielle Steckplatz wird nur mit +5 Volt versorgt.

Kapitel 2

Die Busstruktur des Apple //e

Die Hauptplatine des Apple //e wird von rund drei Dutzend Signalleitungen durchzogen, über die die Kommunikation der Baugruppen untereinander und ihre Kontrolle stattfindet. Die beiden wichtigsten "Leitungsbündel" sind der *Datenbus* und der *Adreßbus*. Über den Datenbus findet die hauptsächliche Datenübergabe, über den Adreßbus die Kontrolle bzw. Aktivierung der jeweiligen Baugruppen statt. Eine der wichtigsten Voraussetzungen für das Verständnis der internen Funktionen des //e und anderer Mikrocomputer ist das Begreifen des Buskonzepts. Erfreulicherweise ist das gar nicht so kompliziert: die zugrundeliegenden Konzepte sollten jedem verständlich sein, der schon einmal einen Computer benutzt hat.

Die Busstruktur ist ein natürlicher Anfangspunkt für eine Erklärung der gesamten Elektronik: Über ihre Besprechung kommen wir zwangsläufig zu den Bausteinen, die mit ihr verbunden sind und darüber zu den einzelnen Bausteinen selber. Allerdings sollten wir jetzt erst einmal eine Erklärung liefern, was unter dem Begriff "Bus" überhaupt zu verstehen ist.

Computerbusse und tri-State-Logik

Logische Signale werden innerhalb des Apple über elektrisch leitende Verbindungen auf der Hauptplatine zu den einzelnen Bausteinen geführt. Eine Anzahl von Signalen, die logisch zusammengehören und über die viele Bausteine miteinander verbunden sind, wird als *Bus* bezeichnet. Physikalisch besteht ein Bus also aus einem "Leitungsbündel", die Anzahl der Leitungen wird als Breite bezeichnet. Der Datenbus des Apple //e ist 8 Bit breit, d.h. mit diesem Begriff werden acht Leitungen logisch zusammengefaßt. Der Adreßbus hat eine Breite von 16 Bit.

Verschiedene Bausteine, die mit einem Bus verbunden sind, arbeiten nur als **Empfänger** wie z.B. ein ROM am Adreßbus. Empfänger reagieren auf die Zustände der einzelnen Leitungen eines Busses, ohne diesen Bus selber zu beeinflussen, d.h. ein ROM liest immer nur Adressen und gibt selber nie welche aus. Elektrisch gesehen stellt ein Empfänger eine hohe Impedanz[1] dar - andere mit dem Bus verbundene Bausteine können den Inhalt des Busses verändern, ohne dabei von einem Empfänger gestört zu werden.

Irgend jemand muß allerdings erst einmal Informationen auf einen Bus legen, bevor mit diesem Bus verbundene Empfänger entsprechend reagieren können. Diese Aufgabe wird von **Sendern** übernommen. Ein Sender hat eine niedrige Impedanz und bringt die einzelnen Leitungen eines Busses auf die gewünschten Pegel. Niedrige Impedanz steht hier für "hoher Stromfluß" - ein Sender kann mehrere Empfänger gleichzeitig versorgen, ohne dabei überlastet zu werden.

Mehrere Empfänger an einem Bus sind also kein Problem - mehrere Sender dagegen schon: Wenn zwei Sender gleichzeitig versuchen, Informationen auf denselben Bus auszugeben, kommt es zu einem Kurzschluß. Um so etwas zu verhindern, sind Sender so konstruiert, daß man sie "abschalten" kann. Zu den zwei bereits bekannten Zuständen "0" und "1" kommt ein dritter Zustand dazu, nämlich "hochohmig" (= hohe Impedanz) oder "abgeschaltet". Ein Baustein, dessen Ausgänge sowohl "0" und "1" als auch den Zustand "hochohmig" annehmen können, wird als *tri-State* (drei Zustände) bezeichnet, um ihn von anderen Bausteinen, die diesen Abschaltvorgang nicht ausführen können ("two-State") zu unterscheiden. Die Ausgänge, über die ein ROM mit dem Datenbus verbunden ist, sind ein typisches Beispiel für ein tri-State-Design.

---

1 Falls Sie mit dem Begriff Impedanz nicht viel anfangen können: ein Baustein mit hoher Impedanz hat einen hohen elektrischen Widerstand und benötigt einen geringen Stromfluß, um eine Information zu erkennen.

Außer reinen Sendern und reinen Empfängern haben wir noch einen dritten Bausteintyp, der als *Transceiver* oder bidirektionaler *Treiber* bezeichnet wird. Das Kunstwort Transceiver ist eine Zusammenziehung der Begriffe *Transmitter* (Sender) und *Receiver* (Empfänger). Diese Bausteinart kann zwischen den Betriebsarten "Sender" und "Empfänger" umgeschaltet werden. Die Datenein- und -ausgänge eines Mikroprozessors sind dafür ein typisches Beispiel: Wenn der Prozessor etwas liest, funktionieren sie als Eingänge und empfangen Daten vom Datenbus, bei einer Schreibaktion dagegen als Ausgänge und senden Daten, verändern also aktiv den Zustand der einzelnen Leitungen des Datenbusses.

Bild 2.1 zeigt einen hypothetischen Datenbus mit 4 Bit Breite und führt dabei noch eine praktische Voraussetzung ein: abgesehen von sehr einfachen Mikrocomputern mit wenigen Bauteilen werden speziell für die Sendefunktion sogenannte *Leitungstreiber* benötigt. Dabei handelt es sich um einfache Verstärker für logische Signale, die aufgrund ihrer Konstruktion einen wesentlich höheren Strom als beispielsweise der Mikroprozessor liefern können (also eine niedrigere Impedanz haben). Ein Dreieck in diesem Schaltplan steht für einen einzelnen Leitungstreiber, die Spitze des Dreiecks kennzeichnet dabei die Richtung des Datenflusses. Eine Wahrheitstabelle für die als Sender geschalteten tri-State-Leitungstreiber in Bild 2.1.

Eingang	**Enable**	**Ausgang**
egal	"0"	hochohmig
"1"	"1"	"1"
"0"	"1"	"0"

Solange der Steuereingang ("Enable") den Pegel "0" hat, befindet sich der Baustein im hochohmigen Zustand, und am Eingang anliegende Informationen werden ignoriert. Wird der Baustein mit einer "1" am Steuereingang aktiviert, dann werden am Eingang anliegende Pegel innerhalb des Bausteins verstärkt und über den Ausgang weitergegeben.

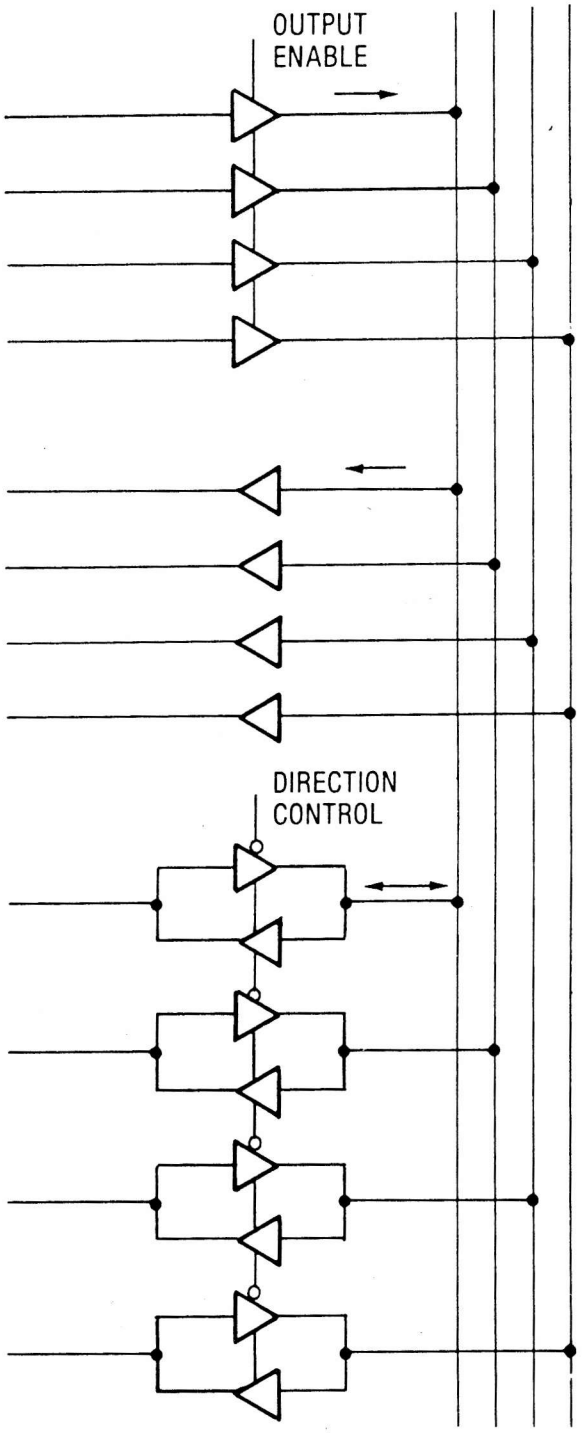
Demzufolge liegt der Schlüssel zum Aufbau einer realen Busstruktur im Steuereingang der Leitungstreiber: In einem Computer wie dem Apple //e mit einer Vielzahl von potentiellen Sendern muß sich irgendwo ein intelligenter Mechanismus befinden, der dafür sorgt, daß immer nur ein Sender über die dazugehörige Steuerleitung aktiviert ist. Wir werden uns in Kürze darum kümmern. Für den Moment sollten Sie im Kopf behalten, daß ein Vorgang wie die Ausgabe des Inhalts einer ROM-Speicherzelle auf den Datenbus nach Erhalt einer Adresse über einen tri-State-Ausgang geschieht, der zu diesem Zweck in den Zustand "aktiv" geschaltet wird und nach der Ausgabe des Datenworts wieder in den hochohmigen Zustand geht.

Bild 2.2 zeigt eine sehr grobe Vereinfachung der Busstruktur des Apple //e, bei der sich zwei Signalzweige unterscheiden lassen, nämlich der Daten- und der Adreßbus. Die Kontrolleitung R/W' ist separat geführt, über sie wird die Flußrichtung der bidirektionalen Ein-/Ausgänge des RAM gesteuert. Mit jedem Taktzyklus des Prozessors findet ein Datenaustausch zwischen dem Prozessor und dem über die Adreßleitungen selektierten Baustein statt, die Richtung des Datenflusses wird dabei über den Pegel der Leitung R/W' bestimmt, der vom Prozessor kontrolliert wird.

Bild 2.3 zeigt die beiden möglichen Formen der Benutzung des Datenbusses in einem Mikrocomputer. Bei einer Leseaktion legt der Prozessor die entsprechende Adresse auf den Adreßbus und liest danach den Datenbus. Bei einer Schreibaktion gibt der Prozessor ebenfalls eine Adresse über den Adreßbus aus, zusätzlich wird ein Datenwort auf den Datenbus ausgegeben. Der Prozessor "weiß" intern, ob er gerade lesen oder schreiben will, d.h. die Steuerung der Datenbus-Transceiver des Prozessors geschieht intern - die Steuerung der restlichen Bausteine geschieht über die Leitung R/W':

1. Niedriger Pegel auf R/W' schaltet alle Ausgänge (Sender) zum Datenbus außer denen des Prozessors in den hochohmigen Zustand, der Prozessor sendet Daten.
2. Hoher Pegel auf R/W' erlaubt die Aktivierung des Ausgangs eines Bausteins, der Prozessor empfängt Daten.

Bild 2.1 Ein hypothetischer Bus mit vier Leitungen



Sender
Anliegende Daten werden zum Bus gesendet, die Ausgänge des Bausteins sind abschaltbar.

Empfänger
Daten werden vom Bus übernommen, der Bus wird dadurch nicht belastet oder verändert.

Bidirektionaler Treiber
Kann zwischen "Sender" und "Empfänger" umgeschaltet werden.

Bild 2.2 Die Grundelemente eines Mikroprozessors

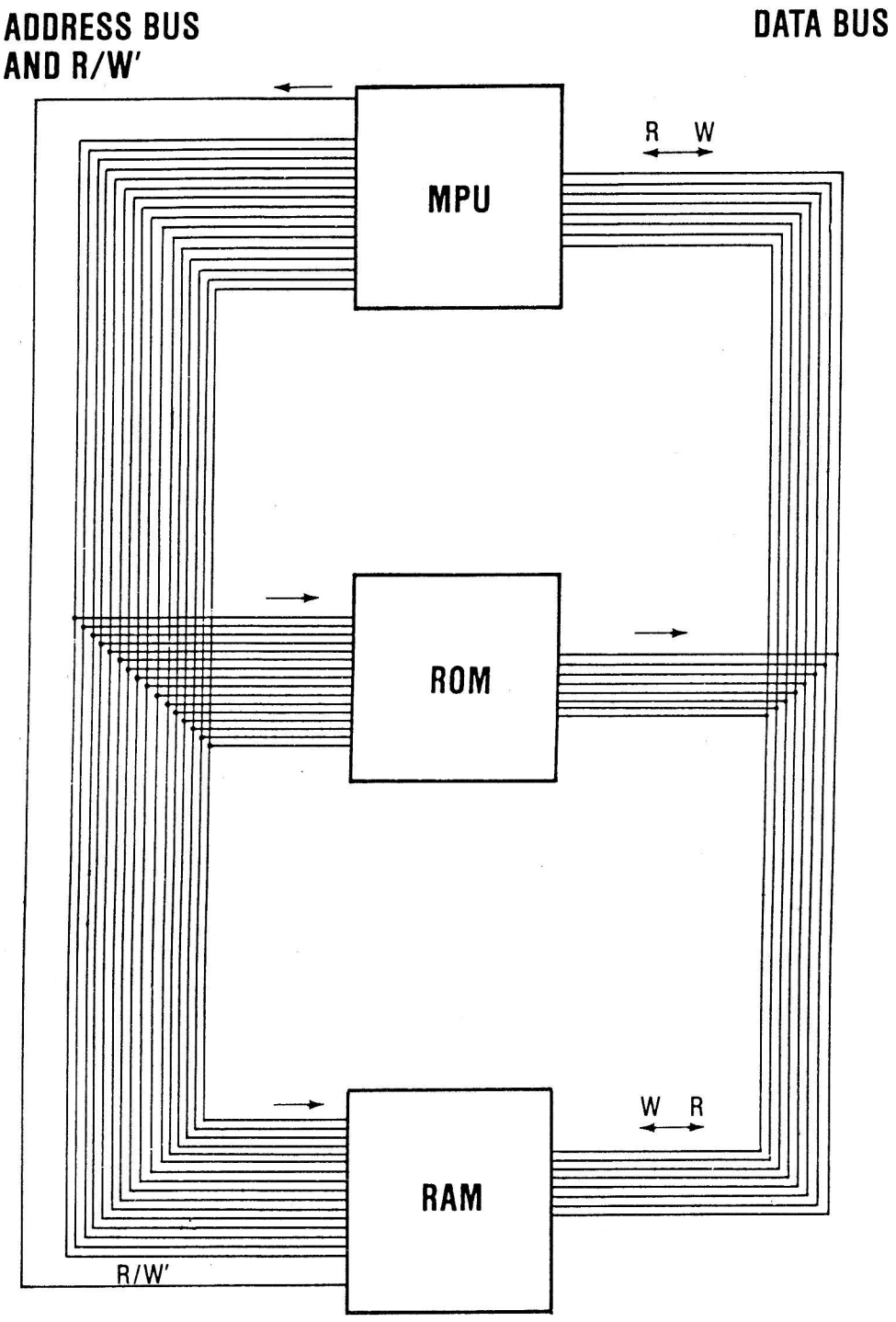
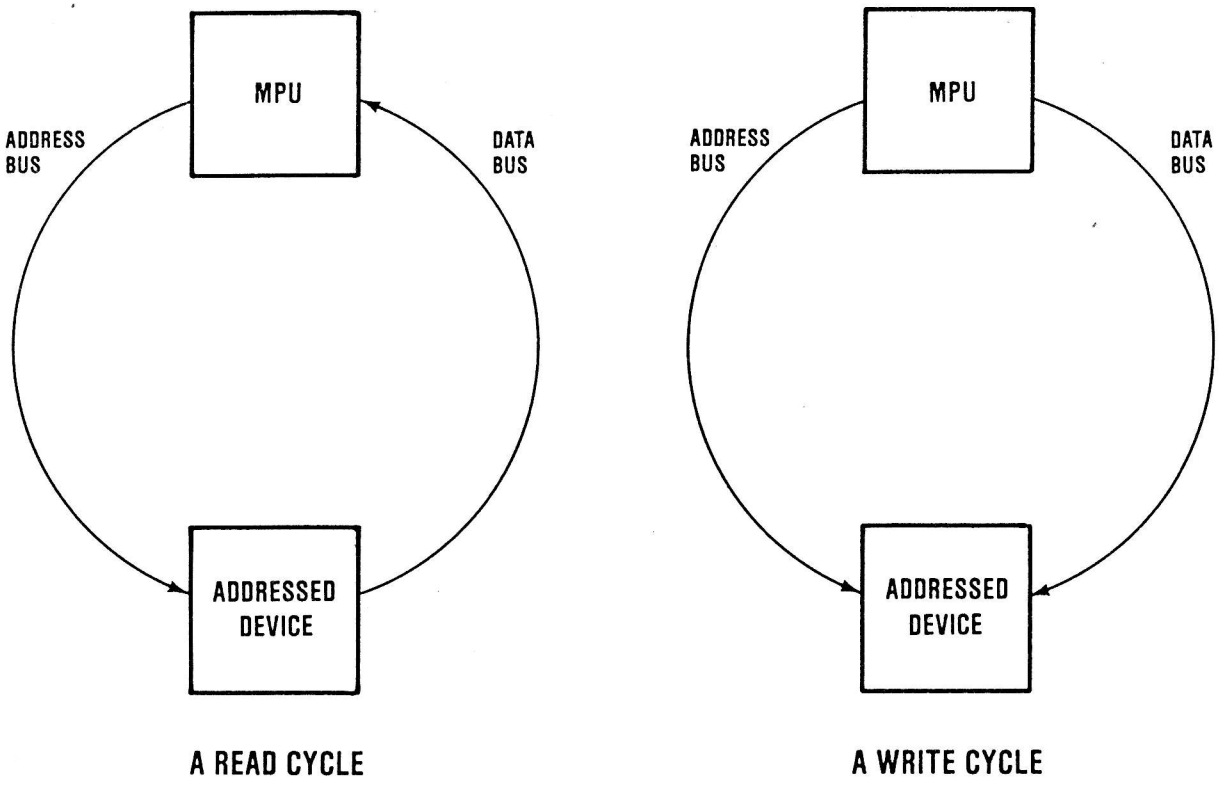


Bild 2.3 Mögliche Flußrichtungen des Datenbusses



Die weiteren notwendigen Erklärungen zur Busstruktur beschränken sich im wesentlichen auf Begriffsdefinitionen. Die Steckplätze des Apple werden manchmal als Peripheriebus oder einfach als "Applebus" bezeichnet. Tatsächlich erfüllt die Art der Verdrahtung der einzelnen Steckplätze die vorher eingeführten Bedingungen und könnte somit als Bus bezeichnet werden - es handelt sich um insgesamt 47 Leitungen, über die alle Steckplätze miteinander verbunden sind - wir haben uns bis jetzt nur der Klarheit halber um diese Bezeichnung gedrückt und werden das auch im weiteren tun.

Jeder Bus hat einen Namen und ein dazugehöriges Kürzel, z.B. "A" für den Adreßbus. Die einzelnen Leitungen eines Busses werden durchnumeriert, die Zählung beginnt mit 0. Die niederwertigste Leitung des Adreßbusses hat somit die Bezeichnung A0, die höchstwertige heißt A15.Analoges gilt für den Datenbus: MD0 bezeichnet die niederwertigste Leitung, MD7 die höchstwertige. Im folgenden eine Liste der verwendeten Bezeichnungen:

Analoges gilt für den Datenbus: MD0 bezeichnet die niederwertigste Leitung, MD7 die höchstwertige. Im folgenden eine Liste der verwendeten Bezeichnungen:

Busname	**Leitungsbezeichnungen**
Adreßbus	A0 - A15
Datenbus	MD0 - MD7
gemultiplexte RAM-Adressen	RA0 - RA7
AUX-RAM Daten	AUXD0 - AUXD7
Video-Daten	VID0 - VID7
peripherer Datenbus	D0 - D7

Die Bezeichnung MD anstelle von D folgt der von Apple, Inc. erstellten Terminologie und unterscheidet den Datenbus auf der Hauptplatine von dem der Steckplätze.

Damit sollten die Grundlagen des Buskonzepts hinreichend erklärt sein. Im folgenden werden wir uns mit der praktischen Ausführung in Mikrocomputern allgemein und im Apple //e im speziellen befassen.

Die Schubfach-Analogie

Es gibt eine sehr alte Analogie, die vielleicht nicht so sehr das Verständnis von BASIC, aber dafür umso mehr das Verständnis der Arbeitsweise eines Computers erhellt:

Stellen Sie sich eine unüberschaubare Reihe von Fächern vor. Jedes Fach enthält ein Stück Papier mit einer Anweisung darauf. Ein Mann beginnt mit dem ersten Fach, holt das Papier heraus, liest es und legt es zurück. Danach führt er die gelesene Anweisung aus und geht zum nächsten Fach, wo sich dieser Prozeß wiederholt. Sämtliche Fächer werden der Reihe nach bearbeitet - es sei denn, er erhält eine Anweisung, mit einem Fach an einer anderen Stelle der Reihe weiterzumachen.

Dieser Mann führt ein sequentiell gespeichertes Programm aus und ist damit eine (recht langsame) Analogie zu einem Mikroprozessor. Die lange Reihe der Fächer ist der Speicher des Computers, ein einzelnes Fach entspricht einer einzelnen Speicherzelle, die gelesenen Anweisungen sind die einzelnen Programmbefehle. Der Prozessor ist genau so schlau, daß er von Fach zu Fach weitergehen, die Anweisungen lesen und ausführen kann - das Vorhandensein der Anweisung ist dazu Voraussetzung. Ohne ein Programm ist mit einem Prozessor nichts anzufangen.

Die CPU, der RAM und der ROM

Der Mikroprozessor (CPU = "Central Processing Unit" = "zentrale Verarbeitungseinheit") ist das ingenieurtechnische Wunderwerk, mit dem Heim- und Personalcomputer überhaupt erst möglich geworden sind. Innnerhalb des Apple tut ein Prozessor mit der Bezeichnung 65(C)02 seine Arbeit. Von außen gesehen kann er Adressen ausgeben, die Leitung R/W' kontrollieren, Daten über den Datenbus lesen und schreiben, gelesene Daten arithmetisch und logisch manipulieren und auf die Zustände verschiedener Kontrolleingänge reagieren. Daraus ergibt sich die Fähigkeit zur Abarbeitung eines Programms - mehr ist es nicht.

Kehren wir noch einmal zur Schubfach-Analogie zurück: Der kleine Kerl, der innerhalb der CPU sitzt, hat die Kontrolle über die Leitung R/W' und die Fähigkeit, eine Adresse im Bereich von 0 bis 65535 auf den Adreßbus zu legen. Über R/W' teilt er dem Rest der Welt mit, ob er den Inhalt eines Schubfachs lesen oder als Reaktion auf eine erhaltene Anweisung mit neuen Daten beschreiben will, über den Adreßbus bestimmt er, um welches Schubfach es sich dabei handelt. Er kann noch eine ganze Reihe anderer Dinge - seine Lieblingsbeschäftigung besteht allerdings darin, den Wert des Adreßbusses um einen einzigen Schritt zu erhöhen und den Inhalt des nächsten Schubfaches zu lesen. Die meisten Schubfächer enthalten Anweisungen, einige von ihnen enthalten Daten - unterscheiden lassen sich beide Sorten von Informationen nicht! Der kleine Mann ist deshalb darauf angewiesen, daß jeder Befehl, den er liest, ihm genau erklärt, ob der nächste Inhalt des Datenbusses als Daten oder als nächster Befehl interpretiert werden soll. Falls er dabei durch einen Programmfehler einmal durcheinander kommt, wird er rücksichtslos versuchen, gelesene Daten als Befehle zu interpretieren und umgekehrt. Das Resultat nennt man dann "aufgehängt" oder "Systemabsturz".

Das Lesen von Befehlen und Daten sieht dabei für den Prozessor reichlich einfach aus: es wird eine Adresse ausgegeben, danach wird der Zustand des Datenbusses gelesen. Die Leitung R/W' bleibt dabei die ganze Zeit im Zustand "1". Der Prozessor fühlt sich leider in keiner Weise dafür verantwortlich, daß nach der Ausgabe einer Adresse irgend etwas Sinnvolles auf dem Datenbus steht - dafür ist die restliche Elektronik des Computers zuständig.

Das bedeutet: Wir brauchen einen oder mehrere Bausteine, die auf die Ausgabe einer Adresse und eine "1" auf der Leitung R/W' mit der Ausgabe eines Datenworts reagieren, also RAM- und/oder ROM-Bausteine. Damit sich mehrere Bausteine nicht gegenseitig in die Quere kommen, sind die einzelnen Adreßbereiche des Apple //e fein säuberlich verteilt: eine Adresse im Bereich von $0000 bis $BFFF spricht immer nur den RAM an, im Bereich von $D000 bis $FFFF wird (solange nicht über Softswitches etwas anderes gesetzt ist) der ROM angesprochen. Die Steckplätze werden über Adressen im Bereich von $C090 bis $CFFF angesprochen und kontrolliert; Softswitches, Tastatur, Lautsprecher und die Schnittstelle zum Kassettenrecorder liegen im Bereich von $C000 bis $C08F. Wenn der Prozessor ein im ROM gespeichertes Programm ausführt (dabei ist der RAM abgeschaltet) und den Befehl erhält, ein Datenbyte auf der Adresse $0400 zu speichern, dann legt der Prozessor die Adresse $0400 auf den Adreßbus und

bringt die Leitung R/W' auf "0". Die Adreßdekodierung schaltet als Reaktion auf die Adresse $0400 den ROM ab, den RAM an und die nun aktivierten RAM-Bausteine erkennen den Zustand der Leitung R/W', lesen den Inhalt des Datenbusses und speichern ihn auf der Adresse $0400. Alle diese Vorgänge benötigen insgesamt genau eine Mikrosekunde - wie Sie sehen, hat es der kleine Mann im Prozessor ganz schön eilig.

Der 6502 führt kontinuierlich ein Programm aus, sobald die Stromversorgung eingeschaltet wird - allerdings ist die erste Adresse, die er nach dem Einschalten der Stromversorgung ausgibt, völlig unbestimmt. Da es dem Prozessor auch noch völlig egal ist, ob auf diese Adresse überhaupt ein Baustein mit einer Ausgabe reagiert, wird er mit einer Wahrscheinlichkeit von 65535 zu 1 Unsinn produzieren.

Zur Beseitigung dieses unerfreulichen Zustands hat der Prozessor einen Kontrolleingang mit der Bezeichnung RESET', der beim Apple //e direkt mit der Leitung RESET' und der entsprechenden Taste verbunden ist. Wenn diese Leitung auf den Pegel "0" gebracht wird, unterbricht der Prozessor jede Arbeit und wartet erst einmal darauf, daß diese Leitung wieder "1" wird. Danach werden automatisch hintereinander die Adressen $FFFC und $FFFD[2] ausgegeben und dafür zwei Byte erwartet, die als Startadresse für den Prozessor interpretiert werden. Der ROM des Apple //e enthält auf den Adressen $FFFC/$FFFD den Wert $FA62 - und auf $FA62 beginnt die RESET-Routine, ein Programmteil des Monitors, der für einen definierten Start des Computers sorgt. Innerhalb dieses Programms wird der Bildschirm gelöscht, die Steckplätze werden nach einem Diskettencontroller abgesucht etc.

Beim Einschalten der Stromversorgung sorgt eine Verzögerung dafür, daß der Pegel der Leitung RESET' etwas länger auf "0" bleibt als das Netzteil braucht, um die Stromversorgung "hochzufahren". Dadurch erhält der Prozessor sofort ein RESET'-Signal und stellt zwischenzeitlich keinen Unsinn an.

Wichtig dabei ist, daß sich die Startadresse auf $FFFC/$FFFD und die RESET-Routine auf $FA62 im ROM befinden - der Inhalt des RAMs ist direkt nach Einschalten der Stromversorgung undefiniert.

Anbei: Die 16 kByte ROM enthalten natürlich noch wesentlich mehr als die RESET-Routine. Hier findet sich auch der gesamte Applesoft-Interpreter, die Firmware-Routinen für die Eingabe/Ausgabe, das Monitorprogramm etc. Ohne angeschlossene Diskettenlaufwerke kann man den Apple als "kassettengestützten" BASIC-Computer bezeichnen - und mehr war er auch nicht, bevor die Diskettenlaufwerke verfügbar waren. Das ist der Hauptgrund, warum Applesoft und das Monitorprogramm im ROM stehen - das Einlesen von 16 kByte über einen angeschlossenen Kassettenrecorder dauert eine Ewigkeit.

Die Adressierung des RAM und die Verteilung der Datenbits

Bild 2.2 zeigt zwar das Funktionsprinzip eines RAM-Zugriffes, verschweigt aber aus Gründen der Übersichtlichkeit eine ganze Reihe von Details. Die praktische Realisierung des RAM-Aufbaus fällt leider erheblich komplizierter aus: zum einen verwendet der Apple //e dynamische RAM-Bausteine, zum anderen können sowohl der 6502 als auch der Videoscanner auf den RAM zugreifen. Dazu kommt noch die Möglichkeit der Bankumschaltung, also des Ein- und Ausblendens verschiedener Speicherbereiche in denselben Adreßbereich. Bild 2.4 ist eine Erweiterung von Bild 2.2 und zeigt schematisch einen Teil der möglichen Daten- und Adreßwege, wobei schwarze Linien für den Adreßbus und graue Linien für den Datenbus stehen. Die Leitung R/W' wird in diesem Diagramm als Teil des Adreßbusses betrachtet und ist nicht mehr extra aufgeführt. Die Kontrolle des Adreßbusses liegt nach wie vor beim Prozessor.

Der Vollständigkeit halber enthält Bild 2.4 auch noch die zusätzlichen 64 kByte AUX-RAM im speziellen Steckplatz - das Zeitverhalten des Apple //e und der Videoscanner sind auf insgesamt 128 kByte RAM eingerichtet. Der AUX-RAM ist dabei so vollständig integriert, daß man meinen könnte, er sei einfach nur auf der Hauptplatine vergessen worden.

In diesem Bild finden Sie auch noch einige weitere Busse, über die Adressen oder Daten transportiert werden. Diese Busse sind nicht direkt mit dem Daten- oder Adreßbus verbunden und werden auch nicht direkt vom Prozessor kontrolliert. Hellgraue Linien stehen für Erweiterungen des Datenbusses, die mittelgrauen Linien stehen für den gemultiplexten RAM-Adreßbus, mit dem wir uns im nächsten Abschnitt näher beschäftigen werden.

---

2 Jede dieser beiden Speicherstellen enthält ein Datenwort mit 8 Bit. Für eine Startadresse mit 16 Bit müssen deshalb zwei Speicherstellen gelesen werden.

Der gemultiplexte RAM-Adreßbus

Hinter diesem erschreckenden Namen verbirgt sich die allgemein übliche Lösung des Problems "Wie verpacke ich einen Speicher mit 65536 Adressen, ohne dabei einen Chip mit 20 oder mehr Pins zu produzieren, der auf der Platine einen entsprechend großen Platz einnimmt?" In diesem Zusammenhang sollte auch nicht vergessen werden, daß auch der eigentliche Chip im Gehäuse des 6502 gerade 4 mal 4 Millimeter mißt - der Rest des Plastikgehäuses wird einzig und allein für die 40 Pins benötigt.

Um 16 Adreßleitungen, die Stromversorgung sowie Ein- und Ausgang in ein möglichst kleines Gehäuse hineinzupacken, kann man also entweder ein Gehäuse mit 20 oder 24 Pins verwenden - oder man verwendet einzelne Pins mehrfach, indem man eine 16-Bit-Adresse in zwei Hälften mit je 8 Bit aufteilt und diese beiden Hälften nacheinander zusammen mit einem Steuersignal anlegt. Der Preis ist der doppelte Zeitbedarf, dafür passen in ein Gehäuse mit 16 Anschlüssen sogar RAMs mit 256 kBit hinein. Der notwendige Vorgang des Aufteilens einer Adresse in zwei Hälften wird als Multiplexen bezeichnet und im Apple //e von der MMU übernommen. Der gemultiplexte Adreßbus ist sowohl zu den 64 kByte RAM auf der Hauptplatine als auch zum AUX-RAM im speziellen Steckplatz geführt, die einzelnen Leitungen werden mit RA0-RA7 bezeichnet.

Die beiden Adreßhälften haben ebenfalls eigene Namen: die erste Hälfte hat den Namen ROW (Reihe), die zweite den Namen COLUMN (Spalte), beide leiten sich aus dem internen Aufbau der RAM-Chips her. Dazu gibt es natürlich auch noch Signale, damit ein RAM-Chip mitbekommt, welche Hälfte einer Adresse gerade anliegt: sie werden mit RAS und CAS (Row Address Select, Column Address Select) bezeichnet.

Damit wäre die Adressierung des RAMs eigentlich bereits kompliziert genug - wir haben aber leider noch einem zweiten Apple-spezifischen Multiplexvorgang Rechnung zu tragen, der darin begründet ist, daß sowohl der Prozessor als auch der Videoscanner Adressen ausgeben können. Daher führt der gemultiplexte RAM-Adreßbus nicht nur von der MMU zu den RAM-Bausteinen, sondern geht noch weiter zur IOU, die den Videoscanner enthält. Während jedes Zyklus des 6502 gibt zuerst der Videoscanner eine Adresse aus, danach ist der Prozessor dran. Der gemultiplexte RAM-Adreßbus muß also innerhalb jedes Taktzyklus einmal zwischen MMU und IOU und für jede der beiden Adressen zweimal umgeschaltet werden. Insgesamt erhält jeder RAM-Baustein damit 4 halbe Adressen pro Taktzyklus des Prozessors:

T1 - Video ROW-Adresse	(IOU)
T2 - Video COLUMN-Adresse	(IOU)
T3 - CPU ROW-Adresse	(MMU)
T4 - CPU COLUMN-Adresse	(MMU)

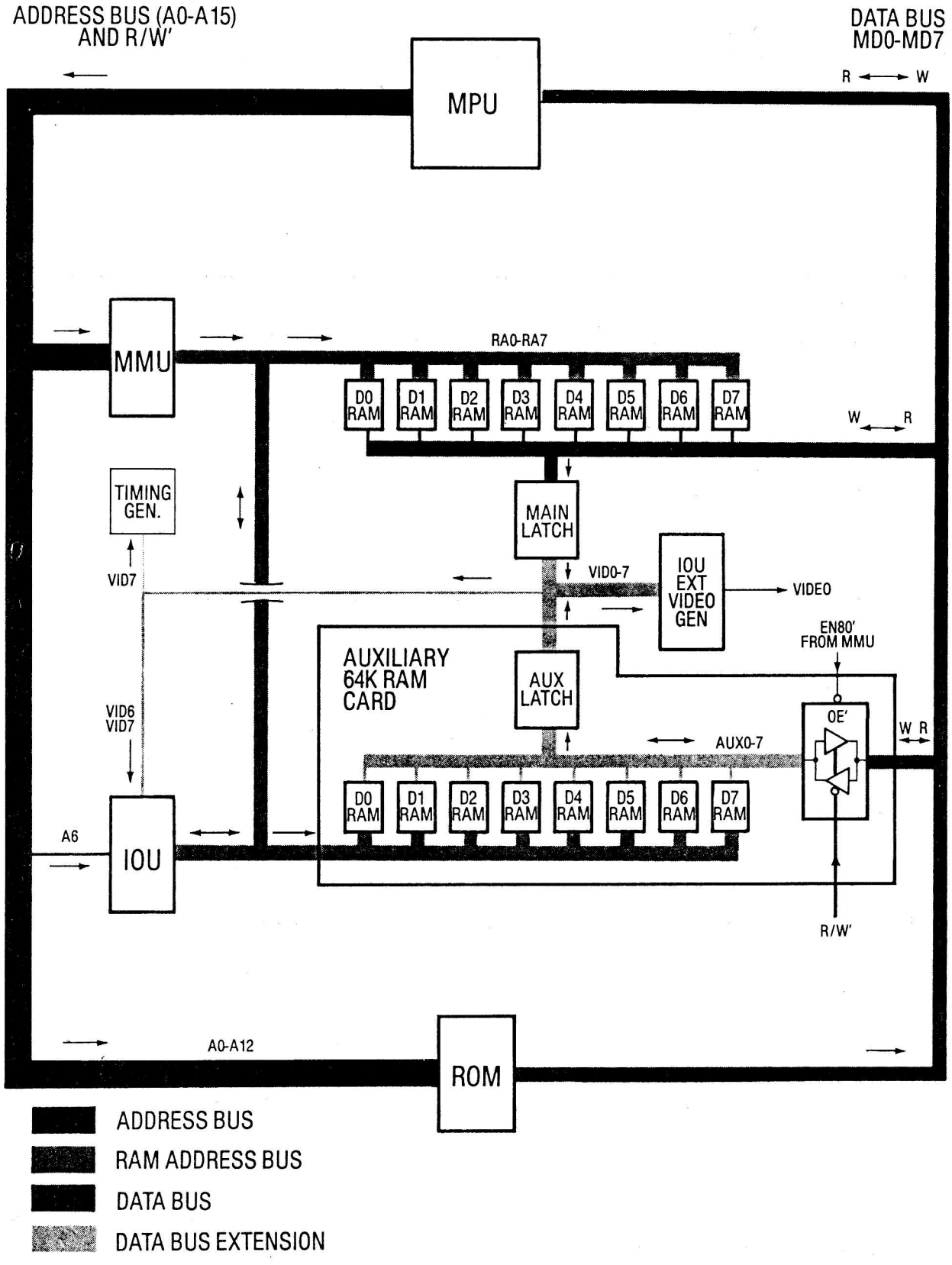
Die Verbindung der IOU zum gemultiplexten RAM-Adreßbus ist bidirektional, d.h. die IOU gibt nicht nur Adressen aus: Wenn die MMU eine Adresse ausgibt, geht die IOU auf "Empfang" und merkt sich die Werte von RA0 bis RA6 der ROW-Adresse der MMU. Mit diesem Trick, auf den wir später noch einmal zurückkommen werden, hat man sich ein paar weitere Pins der IOU gespart.

Der Videoscanner

Dieser Schaltungsteil hat keine Verbindung zum normalen Adreßbus und kann deshalb von der CPU aus nicht direkt kontrolliert werden. Der Scanner besteht hauptsächlich aus einem Zähler innerhalb der IOU und funktioniert ähnlich wie eine (recht primitive) CPU. (Falls es Ihnen mit den Kürzeln noch nicht reicht: die IOU besitzt damit eine eingebaute DMA-Funktion, mit der sie simultan zur CPU auf den gemultiplexten RAM-Adreßbus zugreift.)

Der Videoscanner ist ein reiner Adreßgenerator. Er schreibt keine Daten in den RAM hinein - er liest nicht einmal die als Reaktion auf die erzeugten Adressen vom RAM ausgegebenen Daten. Diese Daten landen statt dessen im Videogenerator selber, der sie in einen sequentiellen Strom von Videosignalen verwandelt und der mit dem Videoscanner weiter nichts zu tun hat.

Bild 2.4 Bus-Teildiagramm: RAM-Adressierung und Datenverteilung im Apple //e



Obwohl sich auf diese Art und Weise die CPU und der Videoscanner den RAM ständig teilen müssen, besteht die einzige Verbindung zwischen den beiden im Zeitablauf der Operationen: der 6502 führt jede Mikrosekunde einen Prozessorzyklus aus, der Videoscanner erzeugt jede Mikrosekunde eine neue Adresse. Beide Operationen sind um eine halbe Mikrosekunde gegeneinander versetzt. Es erscheint daher nur logisch, daß beide ihren Zeittakt aus derselben Quelle erhalten. Um noch etwas genauer zu werden: Der Zeittakt für sämtliche Abläufe innerhalb des Apple bezieht sich auf eine einzige Quelle. Die Zeitabläufe der RAM-Adressierung sind reichlich verwickelt, wir werden uns in den Kapiteln 3,5 und 8 genauer damit beschäftigen.

Die vom Videoscanner ausgegebene Adresse wird innerhalb der IOU noch weiter verwendet, nämlich zur Erzeugung einiger anderer Videosignale wie z.B. dem Taktsignal SYNC, das dafür sorgt, daß der Videomonitor im gleichen Takt wie die Bilderzeugung arbeitet. Heruntergeteilte Zählimpulse des Videoscanners werden für die Blinkfrequenz bei FLASH, für die Autorepeat-Funktion der Tastatur und im MIX-Modus verwendet.

Die Verteilung der RAM-Daten

Die 65536 Byte RAM auf der Hauptplatine bestehen physikalisch aus 8 Chips mit jeweils 65536 einzelnen Speicherzellen. Jeder RAM-Chip ist in der Form 65536 * 1 organisiert, d.h. jeder RAM-Chip hat 65536 Speicheradressen mit jeweils einem Bit Speicherplatz, einen 1 Bit breiten Dateneingang und einen 1 Bit breiten tri-State-Datenausgang. Da der 6502 immer 8 Bit auf einmal erwartet, werden 8 dieser Chips benötigt.

Jeder der RAM-Chips ist mit einer eigenen Leitung des Datenbusses verbunden - die Leitung MD7 führt also zu einem anderen Chip als die Leitung MD6 etc. Die Adreßeingänge aller 8 RAM-Chips sind dagegen parallel geschaltet, jede Adresse wird gleichzeitig an alle 8 Chips angelegt. Damit sind die 8 RAM-Chips funktionell identisch mit einem einzigen (sehr großen) Chip, der 65536 Speicherstellen mit jeweils 8 Bit Breite enthält.

Die Leitung R/W' ist direkt zu jedem RAM-Chip geführt und wird für jeden Zugriff des Videoscanners auf "1" gebracht - schließlich schreibt der Videoscanner nie Daten, sondern liest sie nur.

Die Ausgänge der RAMs führen in zwei Richtungen: einmal zum Prozessor und zum anderen zum *Videolatch*. Zu diesem Baustein hat der Prozessor keine Verbindung, obwohl beide am selben Bus angeschlossen sind. Sie werden sich vielleicht bereits denken können, wieso dem so ist: Das Videolatch wird immer nur zusammen mit dem Videoscanner aktiv und speichert die vom RAM ausgegebenen Bilddaten. Daher kann man eigentlich auch von einem gemultiplexten Datenbus sprechen: während der einen Hälfte jedes Taktzyklus fließen Daten zwischen der CPU und dem RAM, während der anderen Hälfte gibt der RAM Daten an das Videolatch ab.

Das Videolatch ist (ausnahmsweise) ein recht einfacher Baustein: zu Beginn jedes Taktzyklus gibt der Videoscanner eine Adresse aus, das Latch hält die daraufhin vom RAM ausgegebenen Daten bis zum Ende des Taktzyklus fest und stellt sie auf der "anderen Seite" dem *Videogenerator* zur Verfügung, der daraus entsprechende Videosignale formt. Der Videogenerator liegt teilweise innerhalb der IOU (VID6 und VID7) sowie außerhalb auf der Hauptplatine (VID0-VID5). VID7 hat außerdem noch eine Verbindung zum Taktgenerator und legt dort eine eventuelle Verzögerung für einzelne Siebenergruppen von HiRes-Punkten fest.

Der Aufbau des AUX-RAM gleicht dem der Hauptplatine - mit einer Ausnahme: die Datenein- und -ausgänge sind nicht direkt mit dem Datenbus verbunden, sondern über bidirektionale Treiber abgetrennt, die nur dann aktiviert werden, wenn der Prozessor auf den AUX-RAM zugreift. Damit haben wir einen weiteren Datenbus mit den Leitungen AUXD0-AUXD7. Für einen Zugriff des Videoscanners auf den AUX-RAM in einem der 80-Zeichen-Modi wird ebenfalls AUXD0-AUXD7 benutzt - außerdem existiert für die Daten ein zweites Videolatch, das deshalb auch als AUX-Videolatch bezeichnet wird. Sowohl das Videolatch auf der Hauptplatine als auch das Latch auf der Zusatzkarte haben tri-State-Ausgänge, die Steuerung dieser Ausgänge ist zeitlich so ausgelegt, daß immer eines der beiden Latches Daten an den Videogenerator liefert.

Die Zeitabläufe der RAM-Auslese sind so kompliziert, daß wir ihnen ein eigenes Kapitel gewidmet haben - Bild 2.4. sollte Ihnen aber eine ungefähre Vorstellung vermitteln. In der ersten Hälfte jedes Taktzyklus' liefert der Videoscanner eine Adresse, die sowohl den RAM der Hauptplatine als auch den AUX-RAM anspricht. Beide RAM-Gruppen geben daraufhin entsprechende Daten aus, die in den dazugehörigen Latches zwischengespeichert werden. Für die folgende halbe Mikrosekunde erhält der Videogenerator Daten über das Latch auf der Hauptplatine, für die nächste halbe Mikrosekunde Daten über das AUX-Latch. In den 40-Zeichen-Modi wird der Inhalt des AUX-Latch ignoriert und der Videogenerator läßt sich für die Verarbeitung der Daten des Latches auf der Hauptplatine eine ganze Mikrosekunde Zeit.

Zugriffe des Prozessors auf den RAM können innerhalb eines Taktzyklus stattfinden, sobald die Videodaten in den Latches zwischengespeichert sind.

Die Adreßdekodierung

Innerhalb der einzelnen RAM- und ROM-Bausteine geht es reichlich kompliziert zu, um einer angelegten Adresse die richtige Speicherzelle zuzuordnen: Jeder RAM-Chip hat intern 65536, jeder ROM-Chip hat 8192 Adressen, die alle individuell angesprochen werden können. Unnötig zu sagen, daß ein guter Teil der Chipfläche für diese Aufgabe "verbraten" wird.

In einer ähnlichen Weise muß derselbe Prozeß noch einmal auf der Hauptplatine des Apple //e stattfinden - erfreulicherweise ist der benötigte Aufwand etwas geringer: abhängig von einer gegebenen Adresse und der Stellung einiger Softswitches findet eine "Ober-Unterteilung" statt, die eine aus einem runden Dutzend Baugruppen aktiviert.

In den meisten Fällen geschieht diese Dekodierung über die MMU oder die IOU; außerdem existieren noch einige kleinere Schaltungen, die ebenfalls auf einen Adreßbereich reagieren und Steuersignale für andere Baugruppen des Apple ausgeben. Über die Adreßdekodierung werden die folgenden Funktionen gesteuert:

1. Verbindung bzw. Trennung des Datenbusses zu einzelnen Bausteinen. Das funktioniert im Normalfall über die Steuerleitung von tri-State-Treibern und schließt ROM, RAM, MMU, IOU, die Steckplätze und seriellen Eingänge ein.
2. Direkte Kontrolle der seriellen Ausgänge.
3. Kontrolle der Softswitches für die Bildschirmmodi innerhalb der IOU.
4. Kontrolle der Softswitches für die Speicherkonfiguration innerhalb der MMU.

Der 6502 verfügt im Gegensatz zu anderen Prozessoren nicht über ein spezielles Signal, um damit von der Speicheradressierung getrennte Operationen anzuzeigen (der Z80 hat z.B. eine Leitung IORQ' (In/Out Request), deren Pegel anzeigt, ob der Speicher oder eine Baugruppe zur Ein-/Ausgabe angesprochen werden soll).

Die Adressierungs- und Kontrollfunktionen des Adreßbus können deshalb im Apple //e nicht als voneinander getrennt betrachtet werden - über den Adreßbus werden auch alle Ein- und Ausgaben gesteuert.

Bild 2.5 zeigt einen Teil der Busstruktur des Apple //e unter besonderer Berücksichtigung der Adreßdekodierung; die folgende Diskussion bezieht sich auf dieses Diagramm.

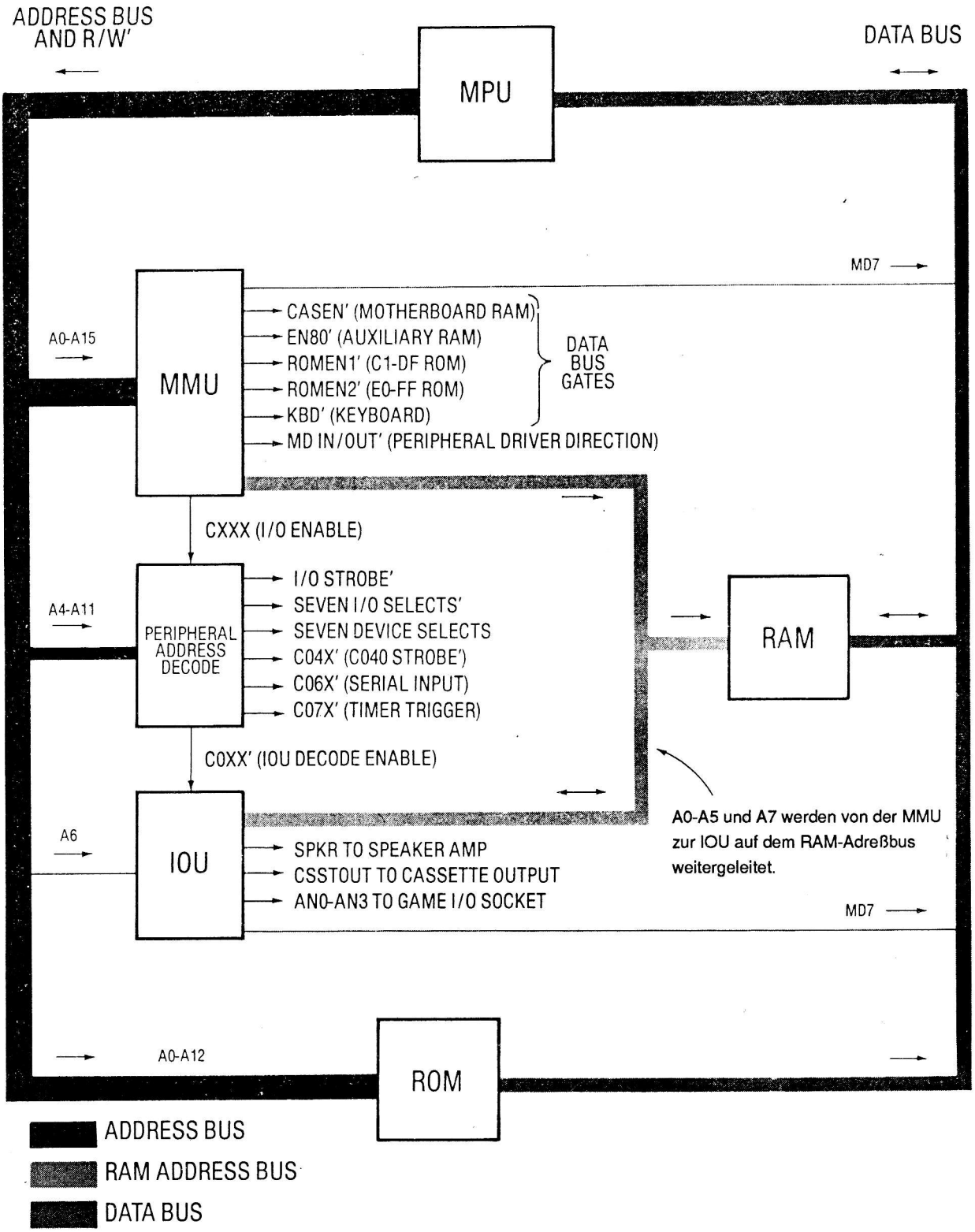
Die zentrale Steuerung der Adreßdekodierung liegt bei der MMU. Dieser Baustein ist der einzige außer dem Prozessor selber, der ständig aktiv und mit allen 16 Adreßleitungen direkt verbunden ist. Die MMU überwacht den gesamten Adreßbereich von $0000 bis $FFFF und aktiviert andere Bausteine über Steuersignale. So hat z.B. ein ROM-Baustein einen Bereich von 8192 Adressen und ist an die Leitungen A0 bis A12 angeschlossen - seine Aktivierung wird dagegen von der MMU gesteuert, die ihm über ein Steuersignal mitteilt, ob eine "seiner" 8192 Adressen angesteuert wurde. Auf diese Weise müssen die höherwertigen Adreßleitungen nur einmal dekodiert werden und es muß nicht jeder einzelne Baustein den Zustand sämtlicher 16 Adreßleitungen überwachen.

Auf dieselbe Weise findet die Aktivierung von einzelnen Ein- und Ausgängen und die Ansteuerung der IOU statt. Die IOU überwacht nur die Leitung A6 separat und erhält den Stand aller anderen Adreßleitungen indirekt über die MMU.

Die Ausgangssignale der MMU sind:

Signal	Funktion
CASEN'	Freigabe des Datentransfers zwischen dem RAM auf der Hauptplatine und der CPU
EN80'	Freigabe des Datentransfers zwischen dem AUX-RAM und der CPU
ROMEN1'	Aktivierung des ROMs $C100..$DFFF
ROMEN2'	Aktivierung des ROMs $D000..$FFFF
CXXX	Freigabe des peripheren Adreßdekoders
KBD'	Aktivierung der Tastatur
MD IN/OUT'	Kontrolle der Richtung der bidirektionalen Treiber des Peripheriebusses

Bild 2.5 Bus-Teildiagramm: Adreßdekodierungssignale des Apple //e



Innerhalb der MMU findet eine weitere Adreßdekodierung statt, die nicht unmittelbar die Ausgabe von Steuersignalen zur Folge hat: die MMU verfügt intern über eine Reihe von Softswitches, die durch das Ansprechen bestimmter Adressen gesetzt oder zurückgesetzt werden können. Der Stand eines Softswitches kann über eine weitere Adresse gelesen werden: wird diese Adresse von der MMU erkannt, gibt sie den Stand des dazugehörigen Softswitches über MD7 des Datenbusses aus.

Das Ansprechen der Adresse $C082 wird z.B. MMU-intern dekodiert und setzt dort den Schalter HRAMRD (High RAM Read = Lesen des RAMs innerhalb der oberen 16 kByte) zurück. Danach führt die Ausgabe einer Adresse im oberen Bereich (z.B. $F200) dazu, daß über ROMEN2' der ROM2 aktiviert wird. Ist dieser Schalter dagegen gesetzt, bleibt ROMEN2' inaktiv und es wird stattdessen der RAM aktiviert. Wir wollen an dieser Stelle nicht weiter auf den elektronischen Aufbau dieser Softswitches eingehen und verweisen auf Kapitel 5 - wichtig ist hier nur, daß ein 6502-Programm seine eigene Programmierumgebung manipulieren kann, indem es Softswitches innerhalb der MMU setzt oder zurücksetzt.

Die MMU reagiert also nicht nur "dumm" auf bestimmte Adreßbereiche - welche Baugruppe durch einen Adreßbereich aktiviert wird, ist zusätzlich vom Stand dazugehöriger Softswitches abhängig.

Alle Signale der MMU außer MD IN/OUT' und CXXX bewirken die Übergabe der Kontrolle des Datenbusinhalts an den aktivierten Baustein im Falle einer Leseoperation bzw. geben bei einer Schreiboperation die Dateneingänge des RAMs frei.[3]

Wenn CXXX aktiv ist ("1"), dann wird dadurch der periphere *Adreßdekoder* aktiviert. Seine Ausgangssignale sind:

- ein I/O STROBE' zu allen sieben Steckplätzen;
- ein I/O SELECT' zu jeweils einem der Steckplätze;
- ein DEVICE SELECT' zu jeweils einem der Steckplätze;
- der Ausgang C040 STROBE';
- die Freigabe $C06X' für die seriellen Eingänge;
- der Startimpuls $C07X' für den Timerbaustein (Joysticks);
- das Signal C0XX' zur IOU.

Die Signale I/O SELECT', I/O STROBE' und DEVICE SELECT' können von Zusatzkarten auf verschiedene Weise benutzt werden und sind in den Kapiteln 6 und 7 beschrieben. In den meisten Fällen bewirken sie die Freigabe des Datentransfers zwischen der CPU und einer Zusatzkarte. Der Ausgang C040 STROBE' führt zu dem GAME I/O-Steckverbinder und geht für eine halbe Mikrosekunde auf "0", wenn der Adreßbus den Wert $C040 hat. C06X' gibt die Verbindung von einem der acht seriellen Eingänge mit MD7 des Datenbusses bei einer Leseoperation des Prozessors frei. C07X' startet alle vier Timer für eine Leseoperation der Analogeingänge. Das Signal C0XX' gibt die nächste Stufe der Adreßdekodierung innerhalb der IOU frei.

Innerhalb der IOU ist die Adreßdekodierung nicht ganz so aufwendig wie innerhalb der MMU - die IOU überwacht nur einen Teil der Adressen im Bereich $C000-$C05F und setzt über diese Adressen einige Softswitches für die Bildschirmmodi oder übergibt den Stand eines Softswitches der Datenleitung MD7 bei einer Leseoperation durch den Prozessor. Außerdem übernimmt die IOU noch die direkte Kontrolle einiger serieller Ausgänge:

- die Announciators 0 bis 3 des Steckverbinders GAME I/O;
- SPKR, der Ausgang zum Lautsprecher;
- CASSO, der Ausgang zur Kassettenrecorder-Schnittstelle.

Wie in Bild 2.5 zu sehen ist, hat die IOU nur eine Verbindung mit der Adreßleitung A6. Obwohl ein großer Teil des Adreßraums bereits über den Aktivierungseingang C0XX' abgedeckt wird, reicht das noch nicht so ganz: Um zwischen "Softswitch an" und "Softswitch aus" zu unterscheiden, wird die Adreßleitung A0, um zwischen "Videomodus" und "Announciator" zu unterscheiden, wird die Adreßleitung A3 benötigt. Um es genau zu nehmen, muß die IOU für alle Funktionen tatsächlich eine Verbindung zu den Adreßleitungen A0 bis A7 haben. Sie erhält auf direktem Wege allerdings nur A6 und C0XX'. Woher kommt der Rest? Hier haben die Entwickler wieder einmal einige

---

3 Noch etwas Terminologie: $CXXX steht für $C-, also für eine beliebige Adresse im Bereich von $C000 bis $CFFF. CXXX ist im Zustand "1" aktiv, ein hypothetisches Signal $CXXX' wäre im Zustand "0" aktiv.

Tabelle 2.1a Die Adressen der Softswitches des Apple //e

FUNCTION	RW	HEX RANGE	DECIMAL RANGE	DECIMAL COMPLEMENT
RESET/SET 80STORE	W	$C000/$C001	49152/49153	−16384/−16383
RESET/SET RAMRD	W	$C002/$C003	49154/49155	−16382/−16381
RESET/SET RAMWRT	W	$C004/$C005	49156/49157	−16380/−16379
RESET/SET INTCXROM	W	$C006/$C007	49158/49159	−16378/−16377
RESET/SET ALTZP	W	$C008/$C009	49160/49161	−16376/−16375
RESET/SET SLOTC3ROM	W	$C00A/$C00B	49162/49163	−16374/−16373
RESET/SET 80COL	W	$C00C/$C00D	49164/49165	−16372/−16371
RESET/SET ALTCHRSET	W	$C00E/$C00F	49166/49167	−16370/−16369
READ KBD/KEYSTROBE	R	$C00X	49152—49167	−16384 TO −16369
RESET KEYSTROBE	RW	$C010	49168	−16368
RESET KEYSTROBE	W	$C01X	49168—49183	−16368 TO −16353
READ KBD/AKD	R	$C010	49168	−16368
READ KBD/HRAM BANK2	R	$C011	49169	−16367
READ KBD/HRAMRD	R	$C012	49170	−16366
READ KBD/RAMRD	R	$C013	49171	−16365
READ KBD/RAMWRT	R	$C014	49172	−16364
READ KBD/INTCXROM	R	$C015	49173	−16363
READ KBD/ALTZP	R	$C016	49174	−16362
READ KBD/SLOTC3ROM	R	$C017	49175	−16361
READ KBD/80STORE	R	$C018	49176	−16360
READ KBD/VBL'	R	$C019	49177	−16359
READ KBD/TEXT	R	$C01A	49178	−16358
READ KBD/MIXED	R	$C01B	49179	−16357
READ KBD/PAGE2	R	$C01C	49180	−16356
READ KBD/HIRES	R	$C01D	49181	−16355
READ KBD/ALTCHRSET	R	$C01E	49182	−16354
READ KBD/80COL	R	$C01F	49183	−16353
TOGGLE CASSETTE OUT	RW	$C02X	49184—49199	−16352 TO −16337
TOGGLE SPEAKER	RW	$C03X	49200—49215	−16336 TO −16321
C040 STROBE'	RW	$C04X	49216—49231	−16320 TO −16305
RESET/SET TEXT	RW	$C050/$C051	49232/49233	−16304/−16303
RESET/SET MIXED	RW	$C052/$C053	49234/49235	−16302/−16301
RESET/SET PAGE2	RW	$C054/$C055	49236/49237	−16300/−16299
RESET/SET HIRES	RW	$C056/$C057	49238/49239	−16298/−16297
RESET/SET AN0	RW	$C058/$C059	49240/49241	−16296/−16295
RESET/SET AN1	RW	$C05A/$C05B	49242/49243	−16294/−16293
RESET/SET AN2	RW	$C05C/$C05D	49244/49245	−16292/−16291
RESET/SET AN3	RW	$C05E/$C05F	49246/49247	−16290/−16289
READ CASSETTE IN	R	$C060,$C068	49248,49256	−16288,−16280
READ PB0	R	$C061,$C069	49249,49257	−16287,−16279
READ PB1	R	$C062,$C06A	49250,49258	−16286,−16278
READ PB2	R	$C063,$C06B	49251,49259	−16285,−16277
READ TIMER0	R	$C064,$C06C	49252,49260	−16284,−16276
READ TIMER1	R	$C065,$C06D	49253,49261	−16283,−16275
READ TIMER2	R	$C066,$C06E	49254,49262	−16282,−16274
READ TIMER3	R	$C067,$C06F	49255,49263	−16281,−16273
TRIGGER TIMERS	RW	$C07X	49264—49279	−16272 TO −16257

Tabelle 2.1b Die Adressen der Softswitches des Apple //e

FUNCTION	RW	HEX RANGE	DECIMAL RANGE	DECIMAL COMPLEMENT
HIGH RAM, BANK2				
WCNT = 0,W',R	RW	$C080,$C084	49280,49284	−16256,−16252
WCNT+1,R'	R	$C081,$C085	49281,49285	−16255,−16251
WCNT = 0,R'	W	$C081,$C085	49281,49285	−16255,−16251
WCNT = 0,W',R'	RW	$C082,$C086	49282,49286	−16254,−16250
WCNT+1,R	R	$C083,$C087	49283,49287	−16253,−16249
WCNT = 0,R	W	$C083,$C087	49283,49287	−16253,−16249
HIGH RAM, BANK1				
WCNT = 0,W',R	RW	$C088,$C08C	49288,49292	−16248,−16244
WCNT+1,R'	R	$C089,$C08D	49289,49293	−16247,−16243
WCNT = 0,R'	W	$C089,$C08D	49289,49293	−16247,−16243
WCNT = 0,W',R'	RW	$C08A,$C08E	49290,49294	−16246,−16242
WCNT+1,R	R	$C08B,$C08F	49291,49295	−16245,−16241
WCNT = 0,R	W	$C08B,$C08F	49291,49295	−16245,−16241
DEVICE SELECT' SLOT 1	RW	$C09X	49296—49311	−16240 TO −16225
DEVICE SELECT' SLOT 2	RW	$C0AX	49312—49327	−16224 TO −16209
DEVICE SELECT' SLOT 3	RW	$C0BX	49328—49343	−16208 TO −16193
DEVICE SELECT' SLOT 4	RW	$C0CX	49344—49359	−16192 TO −16177
DEVICE SELECT' SLOT 5	RW	$C0DX	49360—49375	−16176 TO −16161
DEVICE SELECT' SLOT 6	RW	$C0EX	49376—49391	−16160 TO −16145
DEVICE SELECT' SLOT 7	RW	$C0FX	49392—49407	−16144 TO −16129
I/O SELECT' SLOT 1	RW	$C1XX	49408—49663	−16128 TO −15873
I/O SELECT' SLOT 2	RW	$C2XX	49664—49919	−15872 TO −15617
I/O SELECT' SLOT 3	RW	$C3XX	49920—50175	−15616 TO −15361
I/O SELECT' SLOT 4	RW	$C4XX	50176—50431	−15360 TO −15105
I/O SELECT' SLOT 5	RW	$C5XX	50432—50687	−15104 TO −14849
I/O SELECT' SLOT 6	RW	$C6XX	50688—50943	−14848 TO −14593
I/O SELECT' SLOT 7	RW	$C7XX	50944—51199	−14592 TO −14337
I/O STROBE'	RW	$C800—$CFFF	51200—53247	−14336 TO −12289
SET INTC8ROM	RW	$C3XX (INTC3)	49920—50175	−15616 TO −15361
RESET INTC8ROM	RW	$CFFF	53247	−12289
LOWER 48 RAM ACCESS	RW	$0000—$BFFF	00000—49151	−65536 TO −16385
HIGH RAM ACCESS	RW	$D000—$FFFF	53248—65535	−12288 TO −00001
INT/SLOT ROM ACCESS	RW	$C100—$CFFF	49408—53247	−16128 TO −12289
HIGH ROM ACCESS	R	$D000—$FFFF	53248—65535	−12288 TO −00001

Pins eingespart: die restliche Adreßinformation erhält die IOU über den gemultiplexten RAM-Adreßbus und somit über die Anschlüsse, über die in der ersten Hälfte eines Taktzyklus die Adresse des Videoscanners ausgegeben wird. In der zweiten Hälfte jedes Taktzyklus haben diese Pins eine andere Funktion und lesen die erste Hälfte der RAM-Adresse - damit haben wir die Adreßleitungen A0 bis A7.

Die Kontrollfunktionen der verschiedenen Adressen sind die Grundlage des Aufbaus des Apple //e, Tabelle 2.1 enthält sie in Reihenfolge der aufsteigenden Adressen. Es ist allerdings nicht notwendig, diese Tabellen auswendig zu lernen...

Eingabe und Ausgabe

Eingabe und Ausgabe des Apple //e sind *"memory mapped"*, d.h. in den Adreßraum eingebunden. Eine Lese- oder Schreiboperation des Prozessors bewirkt je nach Adresse via MMU entweder die Aktivierung einer Speichergruppe, eines Softswitches oder eben eine direkte Ein- oder Ausgabe. Alle Adressen, über die Ein- und Ausgaben stattfinden, liegen im Bereich $CXXX. Das betrifft sowohl die bereits auf der Hauptplatine eingebauten Ein- und Ausgänge als auch eventuell eingesteckte Zusatzkarten.

Bild 2.6 zeigt wieder einen Ausschnitt der Busstruktur, diesmal mit dem Schwerpunkt auf der Ein- und Ausgabe. Wie in einem System mit "memory mapped I/O" zu erwarten, ist der Adreßbus direkt oder indirekt zu allen Ein- und Ausgabeeinheiten geführt. Zusätzlich haben die meisten Ein- und Ausgabegruppen eine Verbindung mit dem Datenbus.

Die Kontrolle einzelner Gruppen geschieht wieder über ihre Adresse. Anders gesagt: Wenn der Prozessor eine Ein-/Ausgabeeinheit adressiert, reagieren diverse Schaltkreise auf der Hauptplatine mit der Ausgabe von Aktivierungs- bzw. Freigabesignalen für diese Einheit.

Die Natur der von der Ein- oder Ausgabeeinheit erzeugten bzw. zurückgelieferten Signale hängt von ihrer Funktion ab: der Lautsprecher reagiert sicher nicht mit Daten für den Prozessor, die Adressierung des Ausgangs zum Kassettenrecorder schaltet dessen Pegel um (von "0" auf "1", bei der nächsten Adressierung wieder auf "0" usw.). Adressierung der Tastatur bewirkt dagegen einen Transfer des Codes der zuletzt gedrückten Taste auf den Datenbus, die Ergebnisse der Adressierung von Zusatzkarten hängen von den Zusatzkarten selber ab. (Eine Karte mit einem Z80-Prozessor reagiert z.B. auf eine Schreiboperation so, daß sie den 6502 stillegt!)

Die Eingabe von der Tastatur

Zu den dafür zuständigen Schaltkreisen gehören die Tastatur selber, ein *Tastaturencoder* und ein ROM mit 2048 Byte. Tastatur und Encoder wirken so zusammen, daß für eine gedrückte Taste ein Code festgehalten wird (der übrigens nicht dem ASCII-Satz entspricht). Über diesen Code wird der Tastatur-ROM adressiert, der eine tri-State-Verbindung zu den Datenbusleitungen MD0-MD6 hat. Im Tastatur-ROM sind die der entsprechenden Taste zugeordneten ASCII-Werte gespeichert - dadurch ergibt sich eine einfache Möglichkeit, die Belegung der Tastatur zu ändern. Da der Tastaturcode (also die ROM-Adresse) festgehalten wird, kann die CPU den Wert einer gedrückten Taste zu einem beliebigen Zeitpunkt lesen, auch dann, wenn der Benutzer die Taste bereits wieder losgelassen hat.

Der Prozessor kann den ASCII-Wert einer Taste über die Speicherstelle $C000 lesen. Um es genau zu nehmen, löst jedes Ansprechen einer Adresse im Bereich $C000 bis $C00F über die MMU das Signal KBD' aus und aktiviert die Ausgänge des Tastatur-ROMs.[4] Außerdem reagiert die IOU auf eine $C00X-Adresse mit der Übergabe des Status von KEYSTROBE auf MD7. Der Prozessor bekommt damit für jede Leseoperation im Bereich $C00X auf den unteren 7 Bit den ASCII-Wert der zuletzt gedrückten Taste und auf dem höchstwertigen Bit den Status von KEYSTROBE serviert.

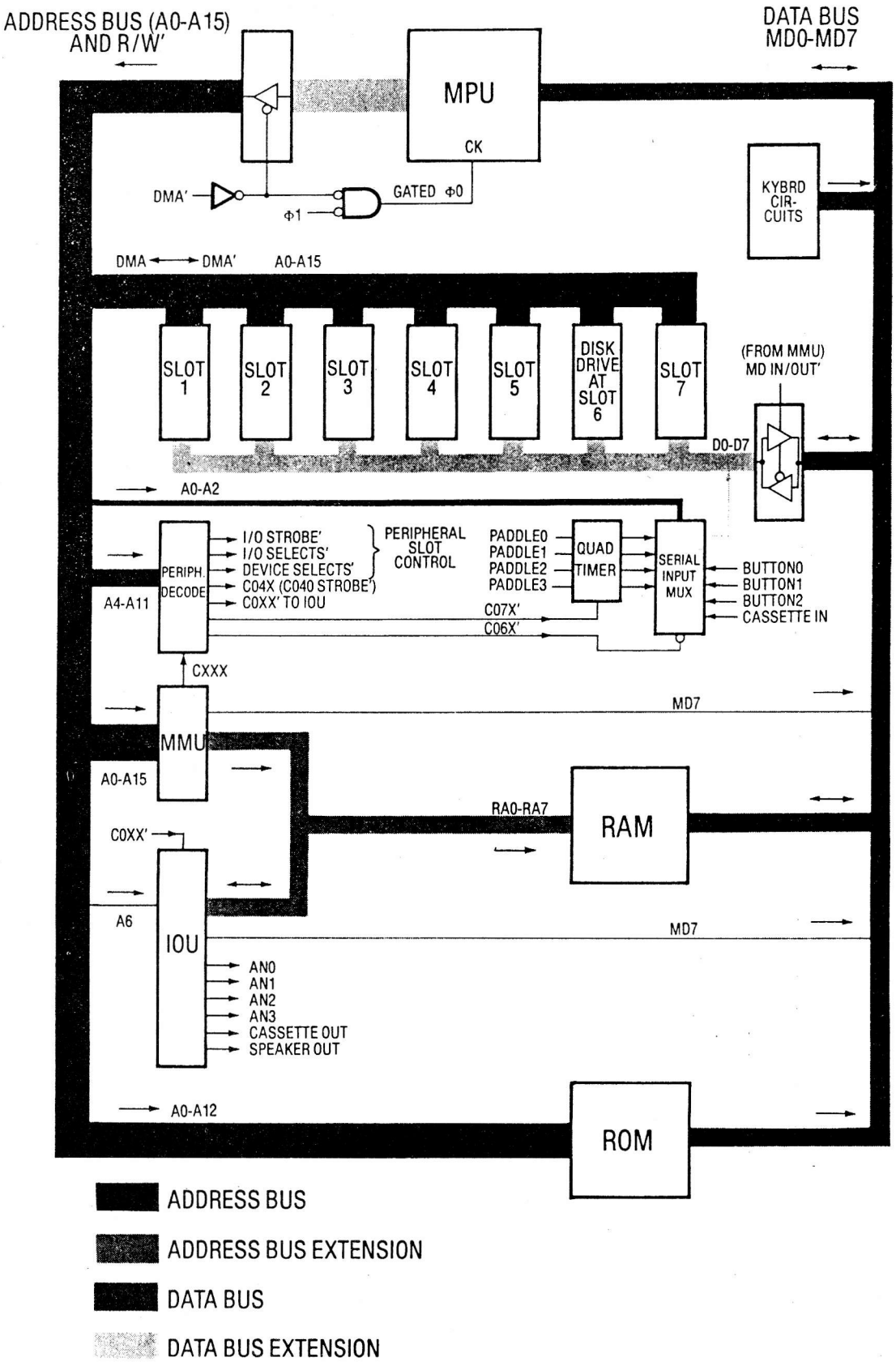
Der Tastaturencoder reagiert auf jeden Tastendruck mit der Ausgabe des Signals KSTRB an die IOU. Die IOU setzt daraufhin das Flag KEYSTROBE auf "1". KEYSTROBE wird durch eine Leseoperation von $C010 oder eine Schreibaktion auf $C01X zurückgesetzt. Auf diese Weise kann ein Programm feststellen, ob eine (weitere) Taste gedrückt wurde, indem es $C000 solange abfragt, bis das höchstwertige (durch KEYSTROBE bestimmte) Bit "1" ist. Die restlichen 7 Bit von $C000 enthalten dabei den ASCII-Wert der gedrückten Taste. Das Programm sollte unmittelbar nach der "erfolgreichen" Leseoperation via $C010 KEYSTROBE löschen, auf den eingegebenen Code entsprechend reagieren und danach $C000 erneut abfragen.

Wenn eine Taste für längere Zeit gedrückt gehalten wird, setzt die IOU KEYSTROBE 15 Mal pro Sekunde (alle 32 bzw. 48 Durchläufe des Videoscanners) automatisch neu und simuliert so für ein Programm das wiederholte Drücken dieser Taste - das Ergebnis ist die Autorepeat-Funktion der Tastatur.

---

4 KBD' wird auch dann aktiv, wenn eine Leseoperation im Bereich $C01X stattfindet, auch wenn das von Apple, Inc. nirgendwo dokumentiert ist. Wagemutige Programmierer können das ausnutzen, um die Tastatur zusammen mit AKD oder anderen IOU-Flags zu lesen - allerdings sollten Sie darauf achten, daß AKD einen Moment früher gültig ist als der ASCII-Wert des Tastatur-ROMs (s. Kapitel 7).

Bild 2.6 Bus-Teildiagramm: Ein- und Ausgabe des Apple //



Zur Tastatur gehört noch ein weiteres Flag mit dem Namen AKD (Any Key Down = "beliebige Taste gedrückt"). Der Prozessor kann dieses Flag über $C010 lesen - es handelt sich dabei übrigens um dasselbe Signal, mit dem der Tastaturencoder der IOU mitteilt, daß eine Taste gedrückt ist. Bei einer Leseoperation von $C010 legt die IOU den Stand von AKD auf MD7 und löscht gleichzeitig KEYSTROBE. Damit hat der Programmierer einige zusätzliche Freiheiten - AKD ist im Gegensatz zu KEYSTROBE nur solange "1", wie auch wirklich eine Taste gedrückt ist.

Die Steckplätze

Alle sieben Steckplätze sind direkt mit dem Adreßbus und über einen bidirektionalen Treiber mit dem Datenbus verbunden. Der primäre Zweck dieses Treibers ist die Stromverstärkung, es handelt sich also mehr um eine elektrische als eine logische Trennung vom Datenbus der Hauptplatine. Zeit- und Kontrollsignale dieses Treibers sind im Takt mit denen des normalen Datenbusses - mit einer Ausnahme: der Treiber ist während einer Schreibaktion der CPU für den Zeitraum der Ausgabe der Videosignale inaktiv, d.h. der durch den Videoscanner erzeugte Datenstrom erreicht die Zusatzkarten nicht, wenn der Prozessor im selben Zyklus eine Schreibaktion ausführt. Damit wird verhindert, daß eine für den Apple II konstruierte Zusatzkarte Videodaten fälschlicherweise mit vom Prozessor ausgegebenen Daten durcheinanderbringt.

Die MMU steuert die Richtung der bidirektionalen Treiber über den Pegel der Signalleitung MD IN/OUT'. Die Richtung wird aus dem Zustand des Adreßbusses und der Leitungen R/W', INHIBIT' und DMA' bestimmt und geht zum Datenbus (IN), wenn MD IN/OUT' "1" ist, ansonsten vom Datenbus zu den Steckplätzen (OUT).

I/O SELECT' ($C100-$C7FF), DEVICE SELECT' ($C090-$C0FF) und I/O STROBE' ($C800-$CFFF) werden auf der Hauptplatine des Apple //e dekodiert und informieren eine Zusatzkarte über einen Zugriff auf eine ihrer Adressen. Die einzelnen Steckplätze sind aber nicht auf einen Bereich von $C090-$CFFF begrenzt: Die Leitung bzw. das Signal INHIBIT' ermöglichen es jeder Zusatzkarte, den gesamten Adreßraum von $0000 bis $BFFF und von $C100 bis $FFFF für sich zu beanspruchen, indem die normalerweise durch diese Adressen angesprochenen Baugruppen auf der Hauptplatine gesperrt werden. Da jeder Steckplatz mit dem gesamten Adreßbus verbunden ist, sind die Möglichkeiten für entsprechend konstruierte Zusatzkarten praktisch unbegrenzt.

Die normalen Steckplätze und der spezielle Steckplatz unterscheiden sich bereits durch die Natur ihrer Signale. Der zusätzliche Steckplatz ist komplett in den Zeitablauf integriert und verfügt über Verbindungen zum gemultiplexten RAM-Adreßbus, dem Datenbus und dem Video-Datenbus und eignet sich dadurch in idealer Weise für 80-Zeichen-Karten und/oder Speichererweiterungen, die übrigens nicht auf 64 kByte beschränkt sein müssen.

Weitere Verbindungen, über die ebenfalls nur der spezielle Steckplatz verfügt, machen ihn zum designierten Platz für Prüfzwecke während der Produktion oder innerhalb eines Reparaturbetriebs.

Im Gegensatz dazu bieten die normalen Steckplätze ein größtmögliches Maß an Flexibilität für den Designer von Zusatzkarten, die auch durchaus etwas einfacher gestrickt sein können. Die Zeitanforderungen sind innerhalb der normalen Steckplätze nicht so kritisch, und dennoch ist sowohl der gesamte Adreß- als auch der Datenbus verfügbar.

Ein-/Ausgabe zusammen mit Diskettenlaufwerken

Diese Form von Ein- und Ausgabe ist ein gutes Beispiel für die Flexibilität der Steckplätze. Ohne eingesteckte Zusatzkarten verfügt auch der Apple //e lediglich über eine reichlich antiquierte Kassettenrecorder-Schnittstelle zur Speicherung und zum Laden von Daten und/oder Programmen. Das geht auf die schlechten alten Zeiten zurück, in denen eine derartige Schnittstelle so ziemlich das Nonplusultra überhaupt war. Nun gut - als Erinnerung daran ist auch im Apple //e diese Schnittstelle übriggeblieben; außerdem kann man sie für einige nette Programmiertricks wie Sprachanalyse und Spracherzeugung in beschränktem Rahmen benutzen.

Der Apple ist ohne Diskettenlaufwerke fast undenkbar - tatsächlich fügt sich eine entsprechende Zusatzkarte auch so nahtlos ein, als wäre sie bereits auf der Hauptplatine vorgesehen gewesen.

Der Datenweg führt bei der Ausgabe von Daten auf die Diskette vom Speicher über die CPU zum Diskettencontroller und von da auf die Diskette selber, beim Lesen von der Diskette verläuft dieser Weg in umgekehrter Richtung. Der Datentransfer zwischen CPU und dem Controller geschieht über den Datenbus in paralleler Form, der Transfer zwischen dem Controller und dem Diskettenlaufwerk seriell.

Der Diskettencontroller, d.h. die Zusatzkarte, die die entsprechende Elektronik enthält, verhält sich für die CPU ähnlich wie ein RAM-Baustein. Wenn Daten von der Diskette gelesen werden, reagiert der Controller auf einen Le-

sezugriff mit der Übergabe eines Datenbytes auf den Datenbus, während eines Schreibvorgangs auf Diskette akzeptiert der Controller von der CPU auf den Datenbus gelegte Daten. Die Adressen der Ein- und Ausgabe zum Controller sind von der Nummer des Steckplatzes abhängig, in dem der Controller steckt. Wenn wir davon ausgehen, daß sich der Controller in Steckplatz 6 befindet, dann ist die Adresse für Leseoperationen von der Diskette $C0EC, geschrieben wird über die Adresse $C0ED. Neben diesen beiden Adressen verfügt der Controller noch über die Kontrolle des Diskettenmotors, des Arms etc. Die entsprechende Adreßdekodierung wird teilweise auf der Hauptplatine, teilweise auf der Controllerkarte vorgenommen: wenn der Prozessor eine Adresse im Bereich $C0EX anspricht, wird via MMU und dem peripheren Adreßdekoder die Leitung DEVICE SELECT' von Steckplatz 6 aktiviert. Der Controller übernimmt die Dekodierung der Adreßleitungen A0-A3 sowie der Leitung R/W' selbst und reagiert entsprechend.

Die tatsächliche Programmierung der Ein- und Ausgabe zur Diskette ist eine schrecklich komplizierte Angelegenheit, es werden auf Mikrosekunden genaue Zeitintervalle, sehr komplexe Umcodierung und anderes benötigt, um das wir uns hier glücklicherweise nicht zu kümmern brauchen. Trotzdem - die gesamte Kontrolle der Diskettenlaufwerke und der Datenverkehr finden über die insgesamt 16 Adressen der Controllerkarte und über den Datenbus statt.

Auf der Hauptplatine existiert kein Programm im ROM, mit dem Daten von der Diskette gelesen werden können, wenn die Stromversorgung des Computers eingeschaltet wird. Auf der Controllerkarte findet sich ein ROM mit 256 Byte Programm im Bereich von $C600-$C6FF (solange der Controller sich in Steckplatz 6 befindet). Nach dem Einschalten der Stromversorgung sucht die Reset-Routine innerhalb des ROMs auf der Hauptplatine sämtliche Steckplätze nach einem Diskettencontroller ab. Wird dabei ein Controller gefunden, findet die Fortsetzung des Programms "auf dem Controller" statt, d.h. das im ROM auf dem Controller stehende Programm wird ausgeführt. Dieses Programm ist zwar reichlich kompliziert, kann aber nur eine einzige Funktion ausführen, nämlich das eigentliche Betriebssystem von der Diskette laden.

Falls bei der Absuche der Steckplätze keine Controllerkarte gefunden wird, startet die RESET-Routine Applesoft - ohne ein Disketten-Betriebssystem.

Direkter Speicherzugriff (DMA)

Wie in Bild 2.6 bereits gezeigt (aber noch nicht erklärt), besteht keine direkte Verbindung des Prozessors mit dem Adreßbus und der Leitung R/W', sondern dies geschieht über tri-State-Treiberbausteine. Der wesentliche Grund dafür ist eine elektrische Entlastung der CPU - allerdings kann man damit auch noch etwas mehr anfangen: die Aktivierungseingänge dieser Treiber sind mit der Leitung DMA' verbunden und können darüber gesteuert werden. Eine Zusatzkarte, die diese Leitung auf aktiven Pegel ("0") bringt, kann den 6502 damit komplett von der restlichen Elektronik des Apple //e isolieren. Außerdem werden die Taktimpulse zum 6502 gestoppt, der Prozessor schaltet sich am Datenbus auf "Eingang" und die MMU verändert die Aktivierungsprozedur der Treiber zum peripheren Datenbus. Dadurch kann ein Prozessor auf einer Zusatzkarte die vollständige Kontrolle des Apple //e übernehmen (die CP/M-Karte mit einem Z80-Prozessor darauf dürfte das bekannteste Beispiel dafür sein).

Soweit nicht ausdrücklich erwähnt, gehen alle Besprechungen in diesem Buch davon aus, daß DMA' inaktiv ist und der Computer vom 6502 kontrolliert wird.

Der serielle Eingangsmultiplexer

Zusätzlich zum Tastatureingang und den Eingabemöglichkeiten über die Steckplätze verfügt der Apple //e über vier Analogeingänge: drei Eingängen zum Lesen der Action-Tasten der Spielsteuerungen und dem Eingang der Kassettenrecorder-Schnittstelle. Jeder der Analogeingänge ist dabei mit einem Timerbaustein ("Zeitkreis") verbunden, der nach einem Startimpuls eine "1" liefert und nach einer gewissen Zeit, abhängig vom Stand der angeschlossenen Potentiometer, wieder auf "0" zurückgeht. Alle vier Analogeingänge, die Action-Tasten und der Kassettenrecorder-Eingang werden über den seriellen Eingangsmultiplexer gesteuert.

Wenn sich eine Adresse im Bereich von $C06X auf dem Adreßbus befindet, legt dieser Multiplexer den Stand eines der folgenden Eingänge auf D7 des (peripheren) Datenbusses:

Adresse	Eingang
$C060/$C068	Kassettenrecorder
$C061/$C069	Action-Taste 0
$C062/$C06A	Action-Taste 1
$C063/$C06B	Action-Taste 2
$C064/$C06C	Timer 0
$C065/$C06D	Timer 1
$C066/$C06E	Timer 2
$C067/$C06F	Timer 3

Die MMU setzt bei einer $C06X-Adresse MD IN/OUT' auf "1" (also in Richtung IN) und aktiviert gleichzeitig das Signal C06X'. Darüber wird der Eingangsmultiplexer aktiviert und der von ihm auf den peripheren Datenbus ausgegebene Wert über die Treiber auf den Datenbus der Hauptplatine geschaltet.

Im Prinzip funktioniert das für die CPU und ein Programm in derselben Weise wie ein ROM: die CPU gibt eine Adresse aus und erhält ein Datum - allerdings ist in diesem Fall nur das höchstwertige Bit von Bedeutung, alle anderen Leitungen des Datenbusses enthalten irrelevante Informationen. Das kontrollierende Programm muß die Auswertung daher allein auf das höchstwertige Bit beschränken.

Die seriellen Ausgänge

Zusätzlich zum Videoausgang und den Ausgabemöglichkeiten über die Zusatzkarten verfügt der Apple //e über sieben serielle Ausgänge auf der Hauptplatine, die allesamt über die Adreßdekodierung betrieben werden. Alle diese Ausgänge werden mit Ausnahme von C040 STROBE' von der IOU generiert. Ihre Namen und die dazugehörigen Adressen sind:

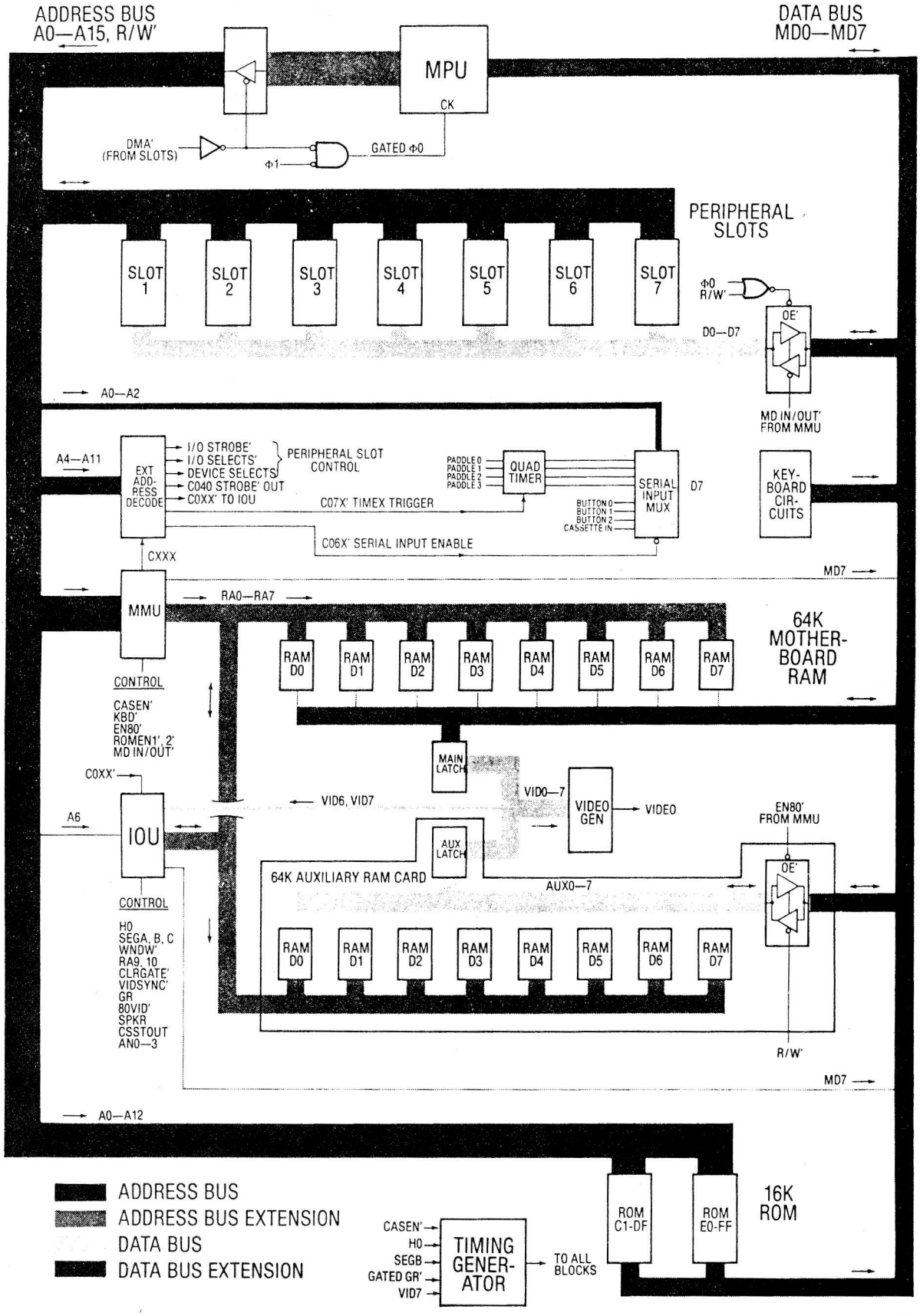
Kontrolladresse	Serieller Ausgang
$C02X	Kassettenausgang
$C03X	Lautsprecherausgang
$C04X	C040 STROBE'
$C058/$C059	ANNOUNCIATOR 0 aus/an
$C05A/$C05B	ANNOUNCIATOR 1 aus/an
$C05C/$C05D	ANNOUNCIATOR 2 aus/an
$C05E/$C05F	ANNOUNCIATOR 3 aus/an

Die seriellen Ausgänge haben strukturell gesehen eine sehr merkwürdige Eigenschaft: Die Datenübermittlung zu einem Ausgang findet nicht über den Daten-, sondern einzig und allein über den Adreßbus statt. Man sollte eigentlich meinen, daß die Informationsübermittlung nach einem ähnlichen Prinzip wie bei den Eingängen funktioniert, also durch Setzen einer der Datenleitungen - tatsächlich ist aber der Inhalt des Datenbusses völlig irrelevant, das Setzen eines dieser Ausgänge geschieht rein über die Kontrollfunktion der Adreßdekodierung. Aus diesem Grund sind z.B. auch für die Announciators zwei Adressen vorgesehen; das Ansprechen einer Adresse setzt den Announciator-Ausgang auf "0", das Ansprechen der zweiten Adresse setzt ihn auf "1". Würde es sich bei den Announciators um eine echte Datenübermittlung handeln, dann hätte jeder Announciator nur eine Adresse, in die entweder eine "0" oder eine "1" vom Prozessor geschrieben würde.

Der Lautsprecher und der Ausgang zum Kassettenrecorder haben kein derartiges Adressenpaar zugeordnet. Stattdessen wechselt der dazugehörige Ausgang bei jedem Ansprechen der Steueradresse seine Polarität: wenn er vorher "0" war, dann wird er auf "1" geschaltet und umgekehrt. Der tatsächliche elektrische Zustand läßt sich über ein Programm unter keinen Umständen bestimmen(!)

Eine dritte Art von Ausgang stellt das Signal C040 STROBE' dar: der dazugehörige Ausgang hat normalerweise den Zustand "1", bei jedem Ansprechen einer Adresse im Bereich von $C04X geht er für eine halbe Mikrosekunde auf "0" und danach wieder zurück auf "1".

Bild 2.7 Die komplette Busstruktur des Apple //e



Die seriellen Ausgänge unterscheiden sich also erheblich von den Eingängen: beim Lesen eines Eingangs findet eine echte Datenübermittlung statt, beim Setzen eines Ausgangs stattdessen eine reine Kontrollfunktion. Damit erklärt sich auch, warum eine BASIC-Anweisung wie

```
10 SPKR = PEEK (-16336)
```

überhaupt einen Sinn ergibt. Obwohl hier mit PEEK *gelesen* wird, reicht das Ansprechen der Adresse -16336 (= $C030) aus, um den Zustand des entsprechenden Ausgang zu wechseln. Der Wert der Variablen SPKR ist nach dieser Operation rein zufällig und außerdem irrelevant - schließlich wird der Datenbus dabei nur "aus Versehen" aktiviert.

Die vollständige Busstruktur

Damit haben wir den strukturellen Abriß der Busse des Apple //e abgeschlossen. Ausgehend von der einfachsten Struktur, die für einen Mikrocomputer überhaupt möglich ist (CPU/RAM/ROM), haben wir uns in diesem Kapitel schrittweise zu einer immer detaillierteren Diskussion der im Apple //e realisierten Busstruktur vorgearbeitet. Bild 2.7 gibt die komplette Struktur mit allen besprochenen Einzelheiten wieder.

Das Verständnis dieser Struktur ist ausgesprochen wichtig, so daß Sie bei Unklarheiten noch einmal einige Seiten zurückgehen möchten - alle folgenden Kapitel bauen auf dieser Besprechung auf und setzen das Verständnis des hier vorgestellten Materials voraus!

In den folgenden Kapiteln werden wir uns nicht mehr der gesamten Struktur, sondern einzelnen Funktionsblöcken des Apple //e widmen und dabei mehr in die Tiefe gehen - bis zur konkreten Ausführung mit Taktverhältnissen, einzelnen Gattern etc.

Kapitel 3

Die Takterzeugung und der Videoscanner

Auf welche Weise die einzelnen Baugruppen des Apple interagieren, haben wir im wesentlichen bereits im Zusammenhang mit der Busstruktur diskutiert. Der wichtigste Punkt ist dabei aus Gründen der Übersichtlichkeit schlicht übergangen worden, nämlich die Zeitabläufe der Operationen. Der Zeittakt (bzw. seine Ableitungen) bestimmt alles, was innerhalb des Apple passiert - ohne ihn geht überhaupt nichts. Wir müssen uns dazu allerdings auf die Ebene der kleinen verzwickten Details, der einzelnen Gatter mit Laufzeiten etc. begeben.

Die Ausführungen und Erklärungen der vorangegangenen Kapitel sind eher genereller Natur; sie sind in der einen oder anderen Form auf fast jeden Mikrocomputer anwendbar und lassen sich vor allem in einer noch zugänglichen Fachsprache formulieren. Sie ahnen bereits: Der folgende Stoff ist zäher und stellt höhere Anforderungen, wir kommen zumindest in diesem Kapitel nicht ohne Zeitdiagramme, Wahrheitstabellen und anderes Fachchinesisch aus.

Einerseits ist Ziel dieses Buchs eine komplette Dokumentation des Apple //e für Techniker, andrerseits, einem möglichst breiten Publikum Einblicke in die Funktion dieses Computers zu geben. Zur Lösung dieses Dilemmas existiert ein umfangreiches Glossar am Ende des Buchs, mit dessen Hilfe wohl jeder Leser, der schon einmal in BASIC programmiert hat, in der Lage sein sollte, den folgenden Text zu verstehen - wenn auch teilweise mit rauchendem Kopf.

Wie dem auch sei: das Fundament dieses Buchs haben Sie bereits hinter sich - falls Sie mit dem Verständnis eines der folgenden Details Schwierigkeiten haben, können Sie es wesentlich ungestrafter überlesen als in den beiden vorigen Kapiteln.

Einige extrem komplizierte Dinge wie z.B. das Innenleben des Prozessors werden überhaupt nicht besprochen, die größten Verständnisschwierigkeiten sind bei der Besprechung der RAM-Baugruppen sowie bei der Analyse von Taktgenerator und Videoscanner zu erwarten. Mit dem letzteren wollen wir uns jetzt gleich beschäftigen - also Augen zu und durch.

Abriß des Taktgenerators

Alle Zeitsignale innerhalb des Apple //e werden aus einer einzigen Quelle abgeleitet, dem Taktgenerator. Wir können uns dafür bei den Designern des Apple nur bedanken - die Verfolgung von Abläufen mit mehreren verschiedenen Taktquellen gehört neben Systemen mit mehreren Prozessoren zum Schwierigsten, was die Computerelektronik überhaupt zu bieten hat.

Zeitsignale bzw. Zeittakte lassen sich praktisch überall auf der Hauptplatine finden - die Eckbedingungen werden allerdings hauptsächlich durch die Benutzung des RAMs gesetzt, der alternierend vom Videoscanner und vom Prozessor angesprochen wird. Die Ausführung eines im RAM gespeicherten Programms und die Erzeugung eines Videosignals durch Auslese der RAM-Bildspeicherbereiche sind zwei völlig verschiedene Aufgaben, sie müssen innerhalb des Apple aber miteinander synchronisiert werden. Wie sich zeigen wird, baut der größte Teil aller Zeitabläufe im Apple //e auf dieser Forderung auf

Der Taktgenerator kontrolliert das Zeitverhalten fast aller Bausteine und -gruppen des Apple - allerdings wirken einige Baugruppen auf den Taktgenerator zurück oder werden durch andere Baugruppen beeinflußt (s. Bild 3.1):

1. Das Signal CAS' vom Taktgenerator wird über die MMU wahlweise gesperrt oder freigegeben.
2. VID7 des Videobusses und der Stand der Softswitches für den Bildschirmmodus beeinflussen die Erzeugung der Signale LDPS' und VID7M.

3. Eine Karte, die im speziellen Steckplatz installiert ist, könnte zusammen mit einer weiteren Karte in Steckplatz 1 sämtliche Taktsignale abschalten und durch eigene Signale ersetzen. Derartige Karten sind zwar momentan nicht auf dem Markt, ihre Konstruktion wäre aber möglich.
4. Ein "Feedback" vom Videoscanner sorgt dafür, daß gegen Ende jeder (horizontalen) Fernsehzeile ein Video-Ausgabezyklus kräftig verlängert wird.

Die unter Punkt 4 genannte Verlängerung ist nötig, um die erzeugten Farben von Zeile zu Zeile stabil zu halten. Da der Videogenerator mit dem Videoscanner und dem Prozessor zeitlich synchronisiert ist, erhöht sich auch die Ausführungszeit des 6502 für jeden 65. Zyklus - der Prozessor arbeitet also nicht mit einer konstanten Taktfrequenz(!). Wir werden diesen speziellen Zyklus innerhalb dieses Buchs als *langen Zyklus* bezeichnen - er ist auch daran schuld, daß der Videogenerator zusammen mit dem Taktgenerator in einem Kapitel behandelt wird.

Der Grundtakt des europäischen Apple beträgt 14.25045 MHz, die amerikanischen Ausführungen haben aufgrund der NTSC-Fernsehnorm (Bildfreuenz 60 Hz anstelle von 50 Hz) eine Grundfrequenz von 14.31818 MHz. Dieses Signal wird von einem Quartz erzeugt und als *14M* bezeichnet. Der Grund für den "krummen" Wert liegt darin, daß bei einer Teilung durch vier die Frequenz 3.5626125 MHz (bzw. 3.579545 MHz) entsteht, die für Farbmonitore als Farbträger ("Color Reference" bzw. "Color Burst") festgelegt ist. Alle weiteren Taktsignale innerhalb des Apple //e werden durch Teilung von 14M erzeugt - die Zeitabläufe des Computers werden also über die Fernsehnorm festgelegt. Allerdings funktioniert ein Quartz aufgrund thermischer Schwankungen nie auf 5 oder 6 Dezimalstellen genau - und außerdem müssen wir uns für eine Betrachtung der logischen Zusammenhänge nicht mit Zahlen wie 1.017889286 herumschlagen. Die ungefähren Frequenzen der wichtigsten Taktsignale sind:

Funktion	**ungefähre Frequenz**
6502-Zyklus	1 MHz
Erhöhung des Videoscanners um einen Schritt	1 MHz
Adreßbus-Zugriff	1 MHz
RAM-Zugriff	2 MHz
COLOR REFERENCE	3.5 MHz
Videoausgang	7 MHz maximal

Alle diese Frequenzen werden durch Teilungen von 14M gewonnen.

Der Taktgenerator besteht aus dem 14.25045-MHz-Oszillator, zwei Flipflops, die das Signal 14M jeweils durch zwei teilen und dem HAL-Chip, der einige festverdrahtete Logikfunktionen enthält. Der HAL ist maßgeblich an der Erzeugung der abgeleiteten Taktsignale beteiligt, wir werden noch auf ihn zurückkommen.

Die Taktsignale

Dieser Abschnitt enthält eine sehr kurze Beschreibung der Signale, die vom Taktgenerator ausgegeben werden. Im weiteren Verlauf dieses Kapitels werden wir detaillierter auf die einzelnen Signale eingehen.

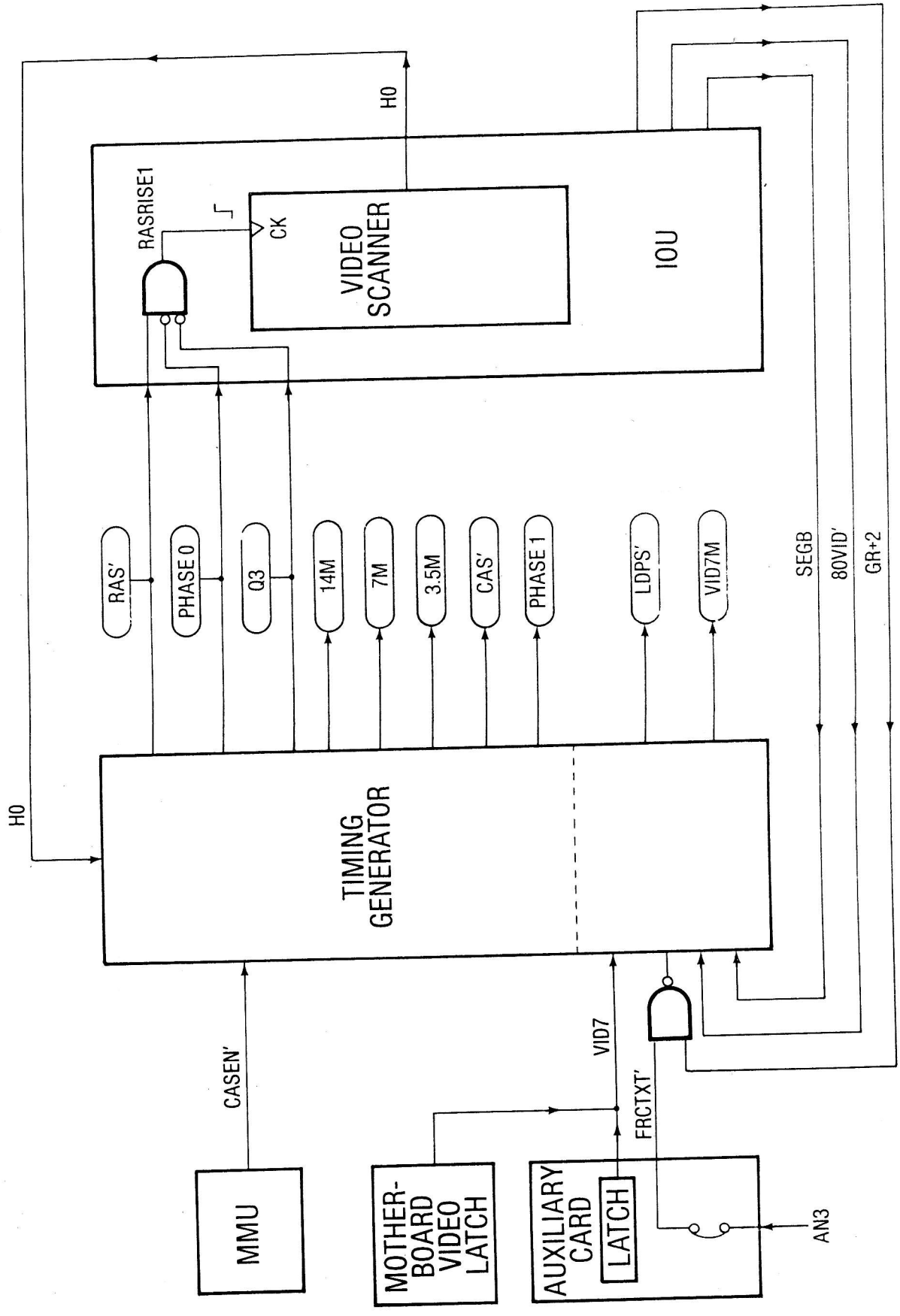
PHASE0 hat eine Frequenz von 1 MHz und ist in doppelt invertierter Form (s.u.) das Taktsignal des Prozessors. Dieses Signal wird an vielen Stellen auf der Hauptplatine sowie innerhalb der MMU und der IOU als Referenz benutzt und gibt z.B. an, wann eine vom Prozessor ausgegebene Adresse gültig ist und ob der Prozessor oder der Videoscanner den RAM adressiert. PHASE0 ist an allen Steckplätzen verfügbar.

PHASE1 ist das Komplement von PHASE0 (also PHASE0') und liefert den Takt für den 6502. PHASE1 wird invertiert und über DMA' gesteuert - wenn DMA' aktiv ist, erhält der 6502 kein Taktsignal mehr. Durch diese Invertierung erhält der Prozessor ein zu PHASE0 synchron laufendes Taktsignal (korrekt: PHASE0''). PHASE1 wird von einigen Schaltkreisen der Hauptplatine als Referenz benutzt und ist ebenfalls an allen Steckplätzen verfügbar.

COLOR REFERENCE hat eine Frequenz von 3.5 MHz und wird dazu benutzt, den "Color Burst" des Videosignals zu erzeugen - welche Farbe ein Monitor erzeugt, hängt von der Phasenlage der Signale zum Color Burst ab. COLOR REFERENCE ist nur zu Steckplatz 7 und zum speziellen Steckplatz geführt.

7M ist ein Signal mit 7 MHz und dient nur zur Erzeugung anderer (heruntergeteilter) Zeittakte. Dieses Signal ist an allen Steckplätzen verfügbar.

Bild 3.1 Funktionsschema von Taktgenerator und Videoscanner



14M ist das Ausgangssignal des Oszillators auf der Hauptplatine und die Quelle aller anderen Takte. Außerdem wird es als Zeittakt für die Schieberegister der Videodaten benutzt.

RAS' (Row Address Strobe) teilt den RAM-Bausteinen mit, daß die Reihenadresse anliegt, und dient als weitere Zeitreferenz für MMU und IOU. Unter anderem erhöht der Videoscanner seinen Zählerstand während PHASE1 auf die steigende Flanke von RAS' (also dann, wenn RAS' wieder inaktiv wird). RAS' ist während eines 6502-Zyklus zweimal aktiv - einmal für den Videoscanner und einmal für Speicherzugriffe des Prozessors.

CAS' (Column Address Strobe) steuert COLUMN und teilt den RAM-Bausteinen auf der Hauptplatine mit, daß eine Spaltenadresse (also der andere Teil einer kompletten Adresse) anliegt. CAS' wird durch CASEN' von der MMU während der aktiven Zeit von PHASE0 gesteuert - dadurch wird bestimmt, ob der RAM auf der Hauptplatine für den Prozessor erreichbar ist. Während der aktiven Zeit von PHASE1 wird CAS' immer aktiviert, d.h. der Videoscanner hat immer Zugriff auf den RAM der Hauptplatine.

Q3 hat eine Frequenz von 2 MHz und dient als Referenz für MMU und IOU. Dieses Signal ist zu allen Steckplätzen geführt.

LDPS' (LoaD Parallel in/Serial out register) ist das Signal, mit dem ein Videozyklus bestimmt wird. Während der aktiven Zeit von LDPS' werden die Daten des RAMs parallel in die Schieberegister geladen, während der restlichen Zeit werden sie seriell als Videosignal hinausgeschoben. LDPS' ist für die 40er Modi einmal pro Prozessorzyklus aktiv, für die 80er Modi zweimal.

VID7M bestimmt die Taktfrequenz der Schieberegister. In den 40er Modi wird nur jeder zweite Impuls von 14M zu den Schieberegistern durchgelassen, in den 80er Modi schiebt jeder 14M-Impuls die Videoschieberegister einen Schritt weiter. Für die Farbkonsistenz in der hochauflösenden Grafik wird dieses Signal teilweise verzögert, um die korrekte Phasenlage einzelner 7-Bit-Gruppen zu COLOR REFERENCE zu erreichen.

Die genauen Frequenzen

Darüber etwas zu sagen, ist reichlich schwierig - schließlich wird jeder 65. Taktimpuls etwas verzögert. Die Signale 14M, 7M und COLOR REFERENCE sind von dieser Verzögerung nicht betroffen; PHASE0, PHASE1, Q3, RAS' und CAS' dagegen schon.Wenn wir diesen langen Zyklus nicht hätten, würden sich die einzelnen Frequenzen durch schlichte Teilung von 14M durch 1, 2, 4, 7 oder 14 ergeben. Für 7M und COLOR REFERENCE stimmt das auch trotz des langen Zyklus noch: 14M hat eine Frequenz von 14.25045 MHz, für 7M ergibt sich 7.125225 MHz und für COLOR REFERENCE 3.5626125 MHz. Die entsprechenden Frequenzen der amerikanischen Version ergeben sich zu 14.31818 MHz, 7.15909 und 3.579545 MHz. Für die Signale mit den Frequenzen 1 und 2 MHz sieht es leider etwas komplizierter aus:

Die Zeit, die ein Signal mit 14.25045 MHz benötigt, um eine komplette Periode ("1" - "0" und wieder "1") zu durchlaufen, beträgt 1/14250450 Sekunden, also 70.1 Nanosekunden. Alle normalen und verlängerten Zyklen der Takterzeugung beziehen sich auf diesen Zeitabschnitt als "Atom", den wir für die folgende Diskussion einfach auch so nennen werden.

Ein normaler Zyklus des 6502 dauert 14 Atome oder 0.982 Mikrosekunden. Der lange Zyklus dauert 16 Atome (1.12 Mikrosekunden). Damit haben wir drei Frequenzen: die primäre Betriebsfrequenz des 6502, die für 64 von 65 Zyklen gültig ist, beträgt 1.017 MHz, die Frequenz des langen Zyklus beträgt 0.8906 MHz, die sich daraus ergebende Frequenz ist 1.0156 MHz.

Die 2 MHz-Signale verhalten sich genauso wie die 1 MHz-Signale, nur daß hier anstelle jedes 65. Zyklus natürlich jeder 130.Zyklus verlängert wird. Die Zeitdauer einer normalen Periode beträgt sieben Atome (0.491 Mikrosekunden), die Dauer eines langen Zyklus beträgt 9 Atome (und nicht 8!). Daraus ergibt sich für einen langen Zyklus die Dauer von 0.631 Mikrosekunden.

Die Frequenzen und Periodendauer der einzelnen Signale finden Sie in Tabelle 3.1 noch einmal komplett zusammengestellt - spaßeshalber bis auf die zehnte Stelle hinter dem Komma. Die angegebenen Werte sind rein theoretisch - aufgrund von thermischen Schwankungen und Bauteiletoleranzen kann man eine Genauigkeit auf 4 oder 5 Stellen erwarten.In dieser Tabelle finden Sie der Vollständigkeit halber auch die Umrechnungsfaktoren für die amerikanische NTSC-Version und die Frequenzen für einen hybriden (d.h. nicht aus einzelnen Teilen aufgebauten) Oszillator, wie er im Apple //c verwendet wird. Wie aus dieser Tabelle ersichtlich, ist ein amerikanischer Apple //e auch beim Rechnen eine Winzigkeit schneller

Tabelle 3.1 Frequenz und Periodendauer der Zeittakte

SIGNAL*	PERIODENDAUER			FREQUENZ		
	normal	langer Zyklus	gemittelt	normal	langer Zyklus	gemittelt
PHASE0	982.4251164 1.015657072		1122.771562	984.5842924	1.017889286	0.8906531251
RAS',CAS',Q3	491.212558	631.5590034	493.3717343	1.83383333	2.035778571	2.026869256
COLOR REF	280.6928904			3.562125		
7M	140.3464452			7.125225		
14M	70.1732226		14.250450			

BILDGENERIERUNG
Für jede Fersehzeile: 912 14M-Perioden
Ein Bildschirm enthält 312 bzw. 262 Zeilen (50/60 Hz)

HYBRIDER OSZILLATOR
Multiplizieren Sie alle Zeiten mit 1.000031579
Multiplizieren Sie alle Frequenzen mit 0.99968422

NTSC-AUSFÜHRUNG
Multiplizieren Sie alle Zeiten mit 0.99526965
Multiplizieren Sie alle Frequenzen mit 1.004752832

** die Tabelle ist auf PAL umgestellt!*

Genauere Beschreibungen der Differenzen zwischen den deutschen PAL-Ausführungen und der amerikanischen NTSC-Version finden Sie in Kapitel 8. Wir werden uns im weiteren hauptsächlich mit der PAL-Version beschäftigen - speziell was das Herunterzählen von Bildschirmzeilen u.ä. betrifft.

Sie werden sich wahrscheinlich bereits gewundert haben, was Sie mit diesen endlosen Nachkommastellen überhaupt anfangen sollen - in der Tat sind die exakten Zahlenangaben nur im Zusammenhang mit der jeweils verwendeten Fernsehnorm interessant. Aber - falls Sie bereits einmal eine "Softwareuhr" programmiert haben: Hier haben Sie den Grund, warum ein derartiges Programm trotz sorgfältigster Programmierung immer ein paar Sekunden (genauer: 54 Sekunden pro Stunde) zu schnell läuft. Mit dem Wissen um die tatsächliche Taktfrequenz von 1.015 anstelle von 1 MHz dürfte eine entsprechende Korrektur möglich sein - eine andere Möglichkeit wäre, die kritischen Verzögerungsschleifen so zu schreiben, daß sich ein ganzzahliges Vielfaches von 65 Taktzyklen des Prozessors ergibt.

In den allermeisten Fällen kommen wir allerdings mit ungefähren Angaben wie 1 MHz oder 3.5 MHz aus.

Das Zeitdiagramm

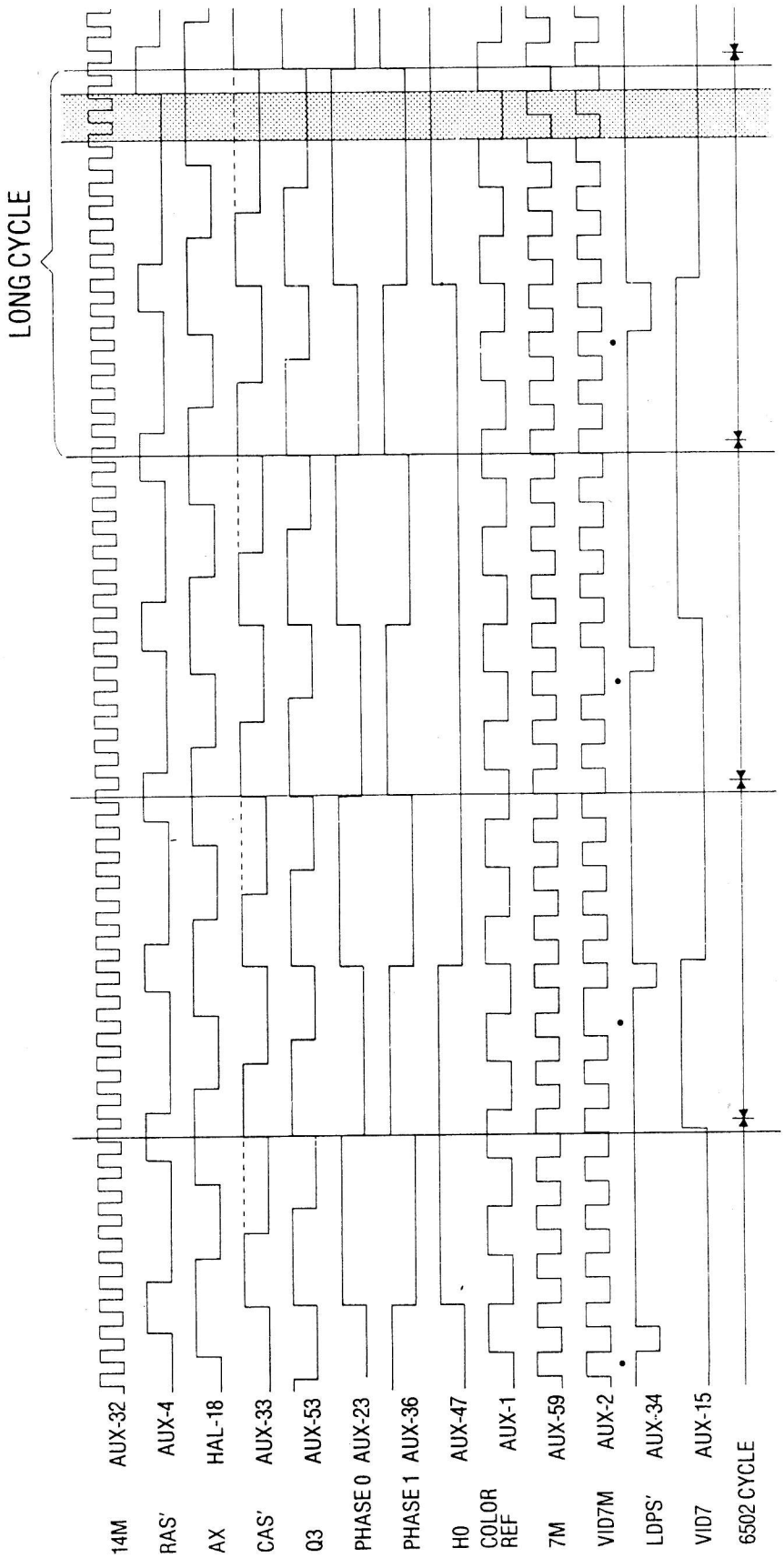
Zeitabläufe werden normalerweise in einem *Zeitdiagramm* zusammengefaßt. Bild 3.2 zeigt sämtliche Ausgänge des Zeitgenerators und einige dazugehörige Signale. In diesem Diagramm sind die einzelnen Schwingungen ("0" - "1" und wieder zurück) als Spannungsverläufe gezeichnet. Technischer: Die Y-Achse enthält die einzelnen Spannungen, die X-Achse steht für die abgelaufene Zeit, die von links nach rechts zunimmt.

In der folgenden Diskussion einzelner Taktsignale werden wir des öfteren auf dieses Diagramm Bezug nehmen - falls Sie über ein Detail innerhalb eines Ablaufs stolpern, sollte ein Blick darauf Klarheit über die Reihenfolge und das Zusammenwirken der Signale schaffen. Zeitangaben werden in Mikro- (Millionstel-) und Nano- (Milliardstel-) Sekunden gemacht.

Bild 3.2 zeigt drei Zyklen des Prozessors, zwei normale und einen langen Zyklus. Jeder normale 6502-Zyklus umfaßt eine Periode von PHASE0, zwei von RAS', CAS' und Q3, dreieinhalb von COLOR REFERENCE, sieben von 7M und 14 von 14M. Zur Erinnerung: Eine Periode von 14M dauert rund 70 Nanosekunden, eine Periode von PHASE0 rund 982 Nanosekunden.

Im Bild sind die Ausgangssignale des Taktgenerators sowie die Signale AX, H0 und VID7 dargestellt. AX (Address Multiplex) wird nur innerhalb des HAL benutzt. Im Apple II wurde mit diesem Signal noch die Gültigkeit der ROW- und COLUMN-Adressen für die RAMs angezeigt, im //e findet man es nur noch an Pin 18 des HAL.

Bild 3.2 Die Taktsignale des Apple //e



H0 ist das niederwertigste Bit der Videoscanner-Adresse (also ein Ausgang der IOU) und ebenfalls ein Eingang des HAL. Sein Zustand ändert sich innerhalb von 64 Zyklen ziemlich genau bei jeder steigenden Taktflanke von PHASE0. Bei jedem 65.Zyklus bleibt dieses Signal allerdings für eine weitere Periode auf "0". Diese doppelt lange Periode, in der H0 auf "0" bleibt, fällt zeitlich mit dem Ende einer horizontalen Fersehzeile zusammen. Achtung: Dieser 65.Zyklus ist in der Mitte von Bild 3.2 gezeigt - es handelt sich dabei *nicht* um einen langen Zyklus.

Sehen Sie sich einmal den Verlauf von PHASE0 auf der linken Seite von Bild 3.2 an: Wenn PHASE0 zum ersten Mal fällt, ist COLOR REFERENCE auf "1", am Ende der nächsten Periode ist COLOR REFERENCE dagegen "0". Diese wechselnde Beziehung zwischen PHASE0 und COLOR REFERENCE kommt daher, daß COLOR REFERENCE seinen Zustand mit der 3 1/2-fachen Geschwindigkeit ändert. Über H0 läßt sich eine konstante Beziehung herstellen: COLOR REFERENCE ist "0", wenn PHASE0 während H0' fällt, und "1", wenn PHASE0 fällt, während H0 "1" ist.

Diese Beziehung stimmt für 64 von 65 Taktzyklen von PHASE0 und muß das auch - schließlich wird darüber die Farberzeugung gesteuert. Allerdings ergibt sich aus den gesamten Zeitverhältnissen und der Fernsehnorm, daß eine Fernsehzeile 65 Zyklen von PHASE0 dauert - wenn hier keine Korrektur eingebaut wäre, würde jede zweite Fernsehzeile mit einem umgekehrten Verhältnis von H0/PHASE0 und COLOR REFERENCE durchlaufen.

Die notwendige Korrektur findet nach jeder 65.Periode statt, nämlich dadurch, daß H0 nach jedem 65.Zyklus zwei Perioden lang auf "0" bleibt. Der HAL, der H0 und die anderen Taktsignale überwacht, stellt fest, daß sich die Beziehung geändert hat und korrigiert das, indem er die Erzeugung der 1- und 2 MHz-Signale um die halbe Dauer einer Periode von COLOR REFERENCE verzögert. Diese Verzögerung verlängert den Zustand "1" der Signale PHASE0 und AX sowie den Zustand "0" von RAS', CAS' und Q3. Dadurch wird auch der laufende Prozessorzyklus etwas verlängert: wir haben einen langen Zyklus produziert. In Bild 3.2 ist die entsprechende Stelle schraffiert gezeichnet.

Abgesehen von LDPS' und VID7M durchlaufen alle Zeitsignale konstant die in Bild 3.2 dargestellten Beziehungen. LDPS' und VID7M hängen vom Darstellungsmodus ab, im Modus HiRes40 kommt noch der Einfluß von VID7 des Videodatenbusses dazu. Dieser Sachverhalt ist ebenfalls noch in Bild 3.2 illustriert: hier werden die Signale LDPS' und VID7M in Abhängigkeit von VID7 gezeigt. Kapitel 8 enthält einige weitere Diagramme, in denen andere Verläufe dieser beiden Signale gezeigt werden.

Bis jetzt haben wir mit einer idealisierten Darstellung gearbeitet. Tatsächlich benötigen alle realen Bausteine eine gewisse Zeit, um den elektrischen Zustand ihrer Ausgänge zu ändern, nämlich rund 6 Nanosekunden. Zusätzlich ergibt sich eine Verzögerung zwischen dem Wechsel eines Eingangspegels und der darauffolgenden Reaktion eines Ausgangs durch interne Umschaltungen innerhalb des Bausteins, sie wird als `Laufzeit` des Signals bezeichnet. Durchläuft ein Signal mehrere Bausteine hintereinander, spricht man von der Gruppenlaufzeit.

Die präzise Darstellung der Laufzeiten ist in einem Diagramm mit dem zeitlichen Maßstab von Bild 3.2 sehr schwierig - typische Verzögerungszeiten liegen im Bereich von 8 bis 30 Nanosekunden pro Baustein. Bild 3.3. zeigt die Hierarchie der einzelnen (Gruppen-)Laufzeiten wesentlich übersichtlicher.

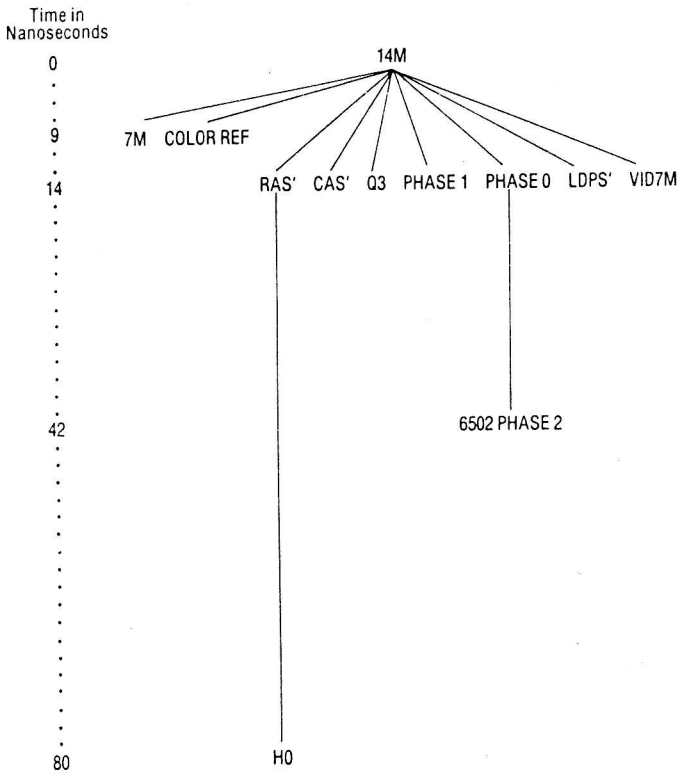
Die steigende Flanke von 14M ist der zentrale Bezugspunkt aller Abläufe. Es ergibt sich die folgende Kette:

1. RAS', CAS' Q3, PHASE0, PHASE1, LDPS', VID7M, 7M und COLOR REFERENCE werden alle über die steigende Flanke von 14M aktiviert. COLOR REFERENCE und 7M werden über ein 74S109 mit einer Laufzeit von rund 9 Nanosekunden erzeugt, die restlichen Signale über den HAL, der eine Laufzeit von rund 14 Nanosekunden hat.

2. PHASE0 wird über einen weiteren Baustein dem 6502 zugeführt. Interne Laufzeiten des Prozessors bedingen eine weitere Verzögerung, bevor der 6502 seinen "data clock" ausgibt, d.h. seinen Ausgang PHASE2 auf "0" setzt. In den Datenblättern des 6502 ist dafür kein typischer Wert angegeben, Messungen des Autors an einem 6502 von Synertek in einem Apple //e (Revision B) ergaben rund 28 Nanosekunden. Diese Verzögerung dürfte je nach Hersteller des Prozessors stark variieren.

3. Der Videoscanner innerhalb der IOU wird durch die steigende Flanke von RAS' während PHASE1 gesetzt. Die Verzögerung zwischen der steigenden Flanke von 14M und der entsprechenden Änderung von H0 betrug nach Messungen des Autors rund 80 Nanosekunden (IOU-Hersteller: AMI) und dürfte ebenfalls je nach Hersteller dieses Bausteins stark variieren. Anders gesagt: H0 ändert seinen Zustand ziemlich genau gleichzeitig mit der steigenden Flanke von PHASE1.

Bild 3.3 Gruppenlaufzeiten der Taktsignale



Die Verteilung der Taktsignale

Bild 3.4 zeigt die Verteilung der Taktsignale auf der Hauptplatine. Jeder Baustein erhält die notwendigen Signale, um mit dem Gesamtablauf synchron zu bleiben. Beachten Sie, daß *alle* Signale des Taktgenerators zu dem speziellen Steckplatz geführt sind - die "normalen" Steckplätze verfügen nur über PHASE0, PHASE1, Q3, 7M und COLOR REFERENCE. Normale Zusatzkarten haben keine Möglichkeit der Überwachung von 14M, RAS', CAS', LDPS' und VID7M.

Detaillierte Beschreibung der Taktsignale

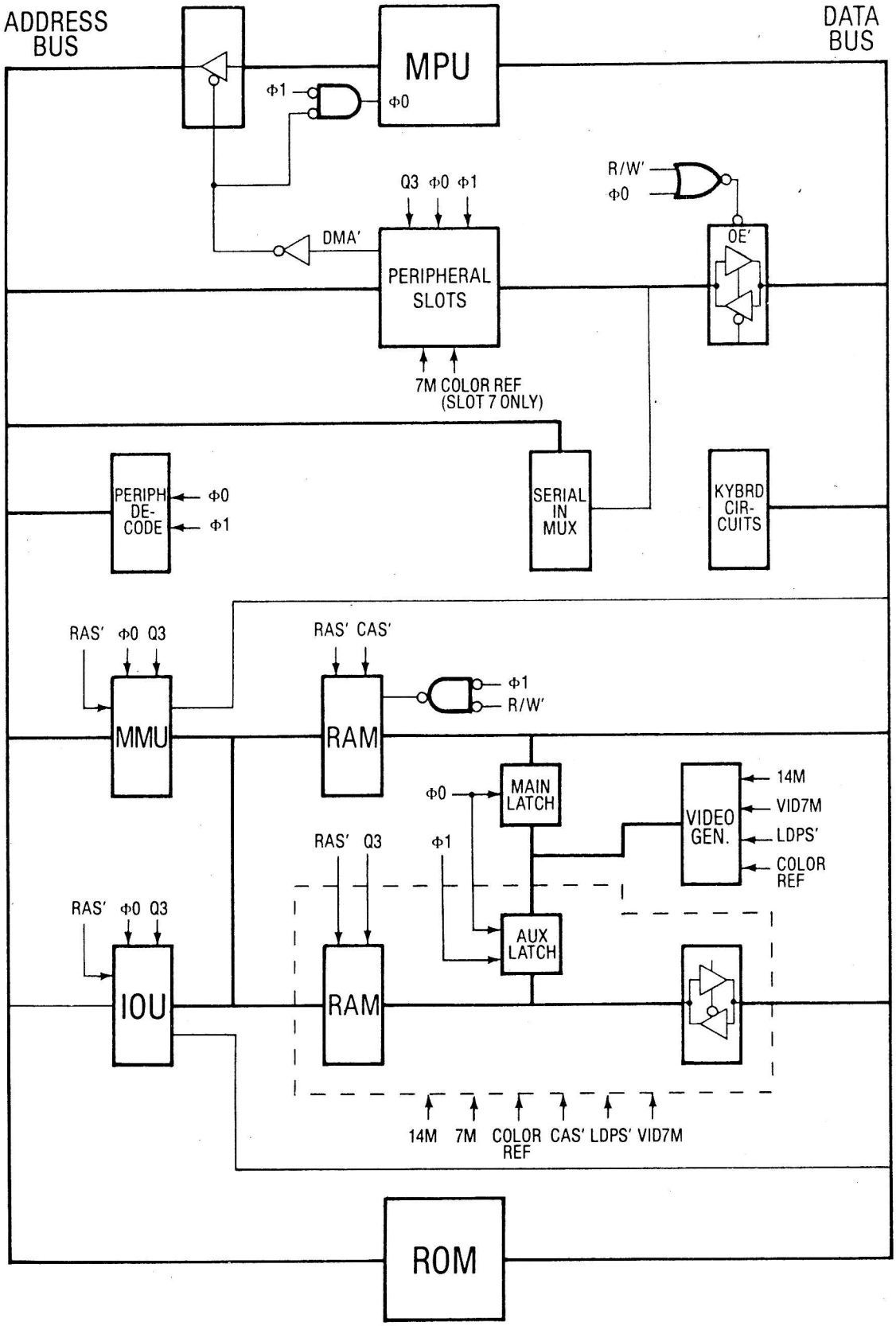
Die folgenden Abschnitte geben eine detaillierte Beschreibung der Verwendung der Taktsignale innerhalb des //e und haben die Bilder 3.2 (Zeitdiagramm) und 3.4 (Signalverteilung) zur Grundlage.

PHASE0 und PHASE1

Diese beiden Signale liefern den primären 1 MHz-Zeittakt des Apple //e. Eine korrektere Bezeichnung wäre eigentlich 1M und 1M', um Verwechslungen mit dem Eingang PHASE0 und dem Ausgang PHASE2 des 6502 zu vermeiden - wir wollen aber auch hier den "offiziellen" Bezeichnungen von Apple, Inc. folgen.

Wie bereits durch den Bezeichnungsvorschlag 1M und 1M' angedeutet, ist PHASE1 lediglich eine Inversion von PHASE0: wenn PHASE0 "1" ist, dann hat PHASE1 den Zustand "0" und umgekehrt.

Bild 3.4 - Die Verteilung der Taktsignale



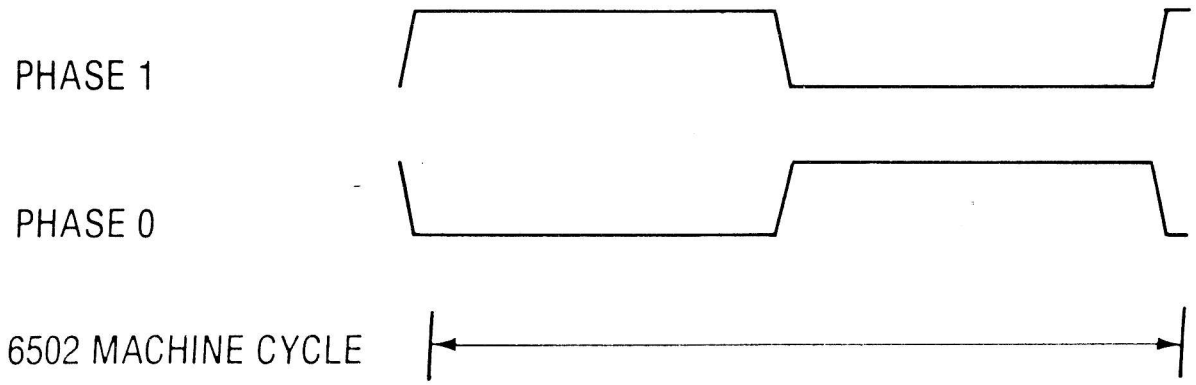
PHASE1 wird über ein von DMA' gesteuertes Gatter invertiert und dem Takteingang PHASE0 des Prozessors zugeführt. Dieses Signal bestimmt also das Zeitverhalten des 6502, die Periodendauer von PHASE0 und PHASE1 entspricht einem Zyklus des Prozessors und beträgt 0.98 Mikrosekunden für einen normalen, 1.12 Mikrosekunden für einen langen Zyklus.

Eine Periode von PHASE0 ("0" - "1" und wieder "0") fällt praktisch mit einem Prozessorzyklus zusammen, allerdings beginnt sie aufgrund der Laufzeiten einige Nanosekunden früher und endet ebenfalls etwas eher. Wenn man PHASE0 und PHASE1 als positive Steuersignale betrachtet (d.h. den Zustand "1"), dann ist PHASE1 ungefähr während der ersten Hälfte jedes Prozessorzyklus aktiv und PHASE0 während der zweiten Hälfte. Diesen Zusammenhang finden Sie in Bild 3.5 verdeutlicht.

Der 6502 reagiert an seinem Eingang nicht auf Pegel, sondern auf Flanken, d.h .auf Pegelwechsel: jeder Wechsel von PHASE1 löst prozessorinterne Aktionen aus, mit denen wir uns im nächsten Kapitel ausführlicher beschäftigen werden. Ein Wechsel von PHASE1 von "0" nach "1" startet (nach einigen Nanosekunden) innerhalb des 6502 einen neuen Maschinenzyklus.

Da PHASE0 und PHASE1 für den Beginn eines Prozessorzyklus zuständig sind, können sie auf der Hauptplatine als Referenz für Aktionen des Prozessors verwendet werden: wenn PHASE0 aktiv ist, sind vom 6502 ausgegebene Adressen gültig - folglich findet die Dekodierung von Adressen für Speicheroperationen des Prozessors nur während PHASE0 statt. Während PHASE1 arbeitet der Prozessor intern, hier finden die Speicherzugriffe des Videoscanners statt. Die Videodaten werden in dem Moment im Videolatch zwischengespeichert, in dem PHASE0 aktiv wird: der Videodatenbus enthält während PHASE0 Daten des AUX-RAMs und während PHASE1 Daten des RAMs auf der Hauptplatine. Da die Videoelektronik immer nur Daten liest und nie schreibt, wird die Schreib-/Lesekontrolleitung des RAMs während PHASE1 immer auf "Lesen" gesetzt - selbst dann, wenn der 6502 die Leitung R/W' Control auf "Schreiben" setzt.

Bild 3.5 Zeitliche Verschiebung zwischen PHASE0 und dem 6502-Maschinenzyklus



14M, 7M und COLOR REFERENCE

14M und COLOR REFERENCE (alternative Bezeichnung: 3.5M) werden beide zur Erzeugung des Videosignals gebraucht. Der Apple kann in den 80er Modi Punktraten bis zu einer Frequenz von 7 MHz erzeugen - die Voraussetzungen dafür sind natürlich entsprechend hohe Taktraten auf der Hauptplatine. Das Signal 7M wird außer für den HAL nirgendwo auf der Hauptplatine gebraucht, es ist aber zu allen Steckplätzen geführt und kann von entsprechend konstruierten Zusatzkarten verwendet werden.

Alle drei Signale werden nicht vom langen Zyklus beeinflußt, ihre Frequenzen sind konstant 14.250450, 7.125255 und 3.5626125 MHz.

14M wird (außer zur Erzeugung der anderen Taktsignale) dazu benutzt, die Schieberegister des *Videogenerators* zu takten, und läuft kontinuierlich "durch". Je nachdem, ob einer der 80er oder einer der 40er Modi gesetzt ist, schiebt jede bzw. jede zweite steigende Flanke von 14M ein Bit aus dem Schieberegister hinaus. Im Gegensatz dazu wird COLOR REFERENCE nur am Anfang jeder Zeile dem Videosignal überlagert, und liefert den Color Burst, ein kurzes "Paket" mehrerer Schwingungen mit 3.5 MHz. Ein Farbmonitor benutzt dieses Wellenpaket am Anfang jeder

Zeile, um einen eigenen Generator zu synchronisieren, der auf derselben Frequenz läuft, und erkennt Farbinformationen durch Phasenunterschiede von COLOR REFERENCE zu den einzelnen Bildpunkten. Der Apple erzeugt die Farben auf dieselbe Weise: Das Signal PICTURE wird je nach gewünschter Farbe gegenüber COLOR REFERENCE zeitlich verschoben[1].

7M ist für die Steckplätze 1 bis 7 an Kontakt 36 geführt, COLOR REFERENCE ist nur an Steckplatz 7 (Kontakt 35) vorhanden. 14M ist an keinem der normalen Steckplätze vorhanden.

RAS', CAS' und Q3

Diese drei Signale haben alle eine Frequenz von 2 MHz. Q3 wird als Zeitreferenz innerhalb von MMU und IOU benutzt und steht an allen Steckplätzen zur Verfügung. Der etwas merkwürdige Name kommt (abgesehen vom Mangel an besseren Ideen) dadurch zustande, daß diese Signalleitung über den Ausgang 3 (Q3) eines 74S195 erzeugt wird. Q3 signalisiert außerdem die Gültigkeit der COLUMN-Adresse im AUX-RAM.

RAS' und CAS' signalisieren die Gültigkeit der ROW- und COLUMN-Adressen für den RAM der Hauptplatine. Wie in Bild 3.2 zu sehen, läuft in jedem Prozessorzyklus zweimal eine RAS'/CAS'-Sequenz ab: während PHASE1 aktiv ist, findet dabei der RAM-Zugriff des Videoscanners, während PHASE0 der Zugriff des Prozessors statt.

Beide Signale liefern ein "Strobe" mit ihrer fallenden Flanke, d.h. in dem Moment, wo sich der Pegel von "1" auf "0" ändert, wird die anliegende Adresse von den RAM-Bausteinen übernommen. Der `Pegel` von RAS' informiert dabei den RAM-Chip, ob es sich bei der anliegenden Adresse um eine ROW- oder um eine COLUMN-Adresse handelt. Ein Zugriff auf eine RAM-Speicherzelle unterteilt sich damit in vier Schritte:

1. Ausgabe und Selektierung der ROW-Adresse mit RAS' auf "1";
2. Strobe der ROW-Adresse durch die fallende Flanke von RAS';
3. Ausgabe und Selektierung der COLUMN-Adresse mit RAS' auf "0";
4. Strobe der COLUMN-Adresse durch die fallende Flanke von CAS'.

RAS' ist direkt mit sämtlichen RAM-Bausteinen auf der Hauptplatine, dem AUX-RAM sowie mit der MMU und der IOU verbunden. In MMU und IOU dient dieses Signal wieder einmal als Zeitreferenz: Wenn RAS' nach einem RAM-Zugriff während PHASE1 wieder auf "1" zurückgeht, wird der Stand des Videoscanners erhöht.

CAS' ist über ein Gatter mit den acht RAM-Bausteinen der Hauptplatine verbunden und wird während PHASE1 immer aktiv - während PHASE0, also bei einem Zugriff des Prozessors nur dann, wenn die MMU CASEN' aktiv geschaltet hat. Das ist der Fall, wenn der Prozessor auf den RAM der Hauptplatine zugreift. In allen anderen Fällen hält die MMU während PHASE0 CASEN' auf "1", CAS' wird nicht aktiv, und die RAM-Bausteine bleiben damit vom Datenbus getrennt.

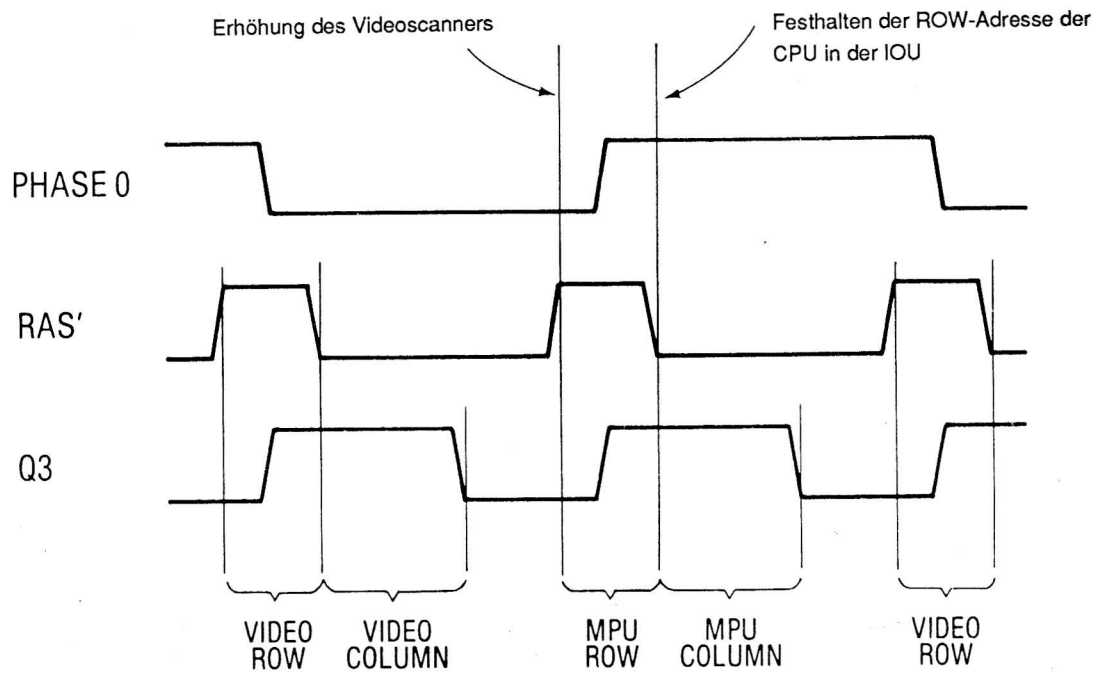
Für den AUX-RAM wird CAS' nicht als Strobe für die COLUMN-Adresse bzw. als Aktivierungssignal (s.o.) benutzt, diese Funktion übernimmt Q3, das immer geschaltet wird. Die RAM-Bausteine des AUX-RAMs erhalten also immer einen COLUMN-Strobe und werden aktiv - die Verbindung des AUX-RAMs zum Datenbus der Hauptplatine wird stattdessen über den bidirektionalen Bustreiber des AUX-Datenbusses hergestellt bzw. "gekappt", der über das Signal EN80' von der MMU gesteuert wird.

Die drei Zeitreferenz-Signale für die MMU und die IOU sind PHASE0, RAS' und Q3. Ihre zeitliche Beziehung und einige der wichtigeren Funktionen, die durch sie kontrolliert bzw. ausgelöst werden, finden Sie in Bild 3.6. Bitte denken Sie auch hier daran, daß die Laufzeiten der Signale in MMU und IOU erheblich sind - die tatsächlich ausgelösten Aktionen finden einige Dutzend Nanosekunden später statt als im Diagramm gezeichnet.

---

1 Innerhalb dieses Buchs wird das Signal, das die Intensität (Leuchtstärke) der Bildpunkte bestimmt, mit PICTURE bezeichnet. Wenn dieses Signal für einen Punkt auf dem Pegel "weiß" ist, dann ist das ausgegebene Videosignal stark genug, daß der Elektronenstrahl des Monitors an der entsprechenden Stelle im Bildschirm einen hellen Punkt erzeugt. Das komplette VIDEO-Signal des Apple //e setzt sich aus den Komponenten SYNC, COLOR REFERENCE und PICTURE zusammen.

Bild 3.6 Zeitdiagramm der MMU- und IOU-Signale



LDPS' und VID7M

Der Verlauf dieser beiden Signale hängt sehr stark vom jeweiligen Bildschirmmodus des Apple ab und ist ausführlich in Kapitel 8 beschrieben. Die folgende Diskussion beschränkt sich deshalb auf ihre Funktion.

Die Erzeugung des Signals PICTURE läßt sich in zwei voneinander getrennte Schritte unterteilen, nämlich das Laden des Video-Schieberegisters mit Daten und das bitweise Hinausschieben der geladenen Daten zum Videogenerator, der PICTURE, COLOR REFERENCE und SYNC zu einem kompletten Videosignal zusammensetzt. Die Punktfolgen werden dafür für Text *und* Grafik von einem ROM parallel geladen, dessen entsprechende Speicherzelle durch den im Videolatch festgehaltenen Wert adressiert wird. Diese Punktfolge (jeweils 7 Bit) wird danach aus dem Schieberegister seriell (d.h. jeweils 1 Bit pro Takt) herausgeschoben und stellt das Signal PICTURE dar.

Die Arbeitsweise ("Laden" oder "Schieben") des Schieberegisters wird vom Pegel des Signals LDPS' festgelegt. Wenn LDPS' "0" (also aktiv) ist, wird parallel geladen, während der restlichen Zeit werden die Bits hinausgeschoben. Die Ladezeit ist dabei gegenüber der Schiebezeit sehr kurz.

LDPS' wird am Ende jeder PHASE1 aktiv, in der 80er Modi zusätzlich noch am Ende jeder PHASE0.

VID7M steuert das Taktsignal für das Schieberegister. Wenn VID7M "0" ist, bewirkt jede steigende Flanke von 14M entweder eine Lade- oder eine Schiebeaktion. In den Modi TEXT40 und HiRes40 hat VID7M eine Frequenz von 7 MHz (daher der Name) und löst damit auf jede zweite steigende Flanke von 14M eine Operation aus. Dieses 7 MHz-Signal ist normalerweise identisch mit 7M, in den verzögerten Videozyklen von HiRes40 ist allerdings die Phasenlage um 180 Grad gedreht.

Bei LoRes40 und allen 80er Modi ist VID7M durchgehend auf "0" - damit löst jede steigende Flanke von 14M eine Operation des Schieberegisters aus, und die Bits werden mit doppelter Geschwindigkeit herausgeschoben.

Der Aufbau eines Fernsehbildes

Um die Arbeitsweise des Videoscanners zu verstehen, müssen wir uns kurz mit der Arbeitsweise eines Monitors bzw. Fernsehers beschäftigen. Eine ausführliche Darstellung folgt in Kapitel 8, hier geht es nur um die prinzipielle Funktionsweise, wobei wir auch Details wie Zeilensprungverfahren etc. einfach übergehen.

Ein komplettes Fernsehbild ist aus einzelnen Zeilen aufgebaut, die vertikal untereinander liegen. Die Innenseite eines Bildschirms ist mit einer Substanz beschichtet, die aufleuchtet, wenn sie von einem Elektronenstrahl getroffen wird. Ein Fernseher oder Monitor enthält eine Elektronenkanone, deren Strahl über eine Steuerung an- oder ausgeschaltet werden kann und der über Magnetfelder sehr schnell von links nach rechts bewegt wird. Wenn der Elektronenstrahl am Ende einer Zeile angekommen ist, beginnt ein neuer (horizontaler) Zeilendurchlauf ein kurzes Stück darunter. Wenn alle Zeilen des Bildschirms durchlaufen sind, beginnt der Prozeß in der oberen linken Ecke des Bildschirms von vorne. Die Beschichtung der Bildröhre leuchtet dabei nach einem Treffer durch den Elektronenstrahl etwas länger, als die Elektronik für einen kompletten Bilddurchlauf braucht. Dadurch entsteht ein scheinbar stehendes Bild.

Bild 3.7. verdeutlicht diesen Ablauf, aus Gründen der Übersichtlichkeit ohne die Rückläufe des Elektronenstrahls in horizontaler (nach jeder Zeile) und in vertikaler Richtung (nach jedem kompletten Bild).

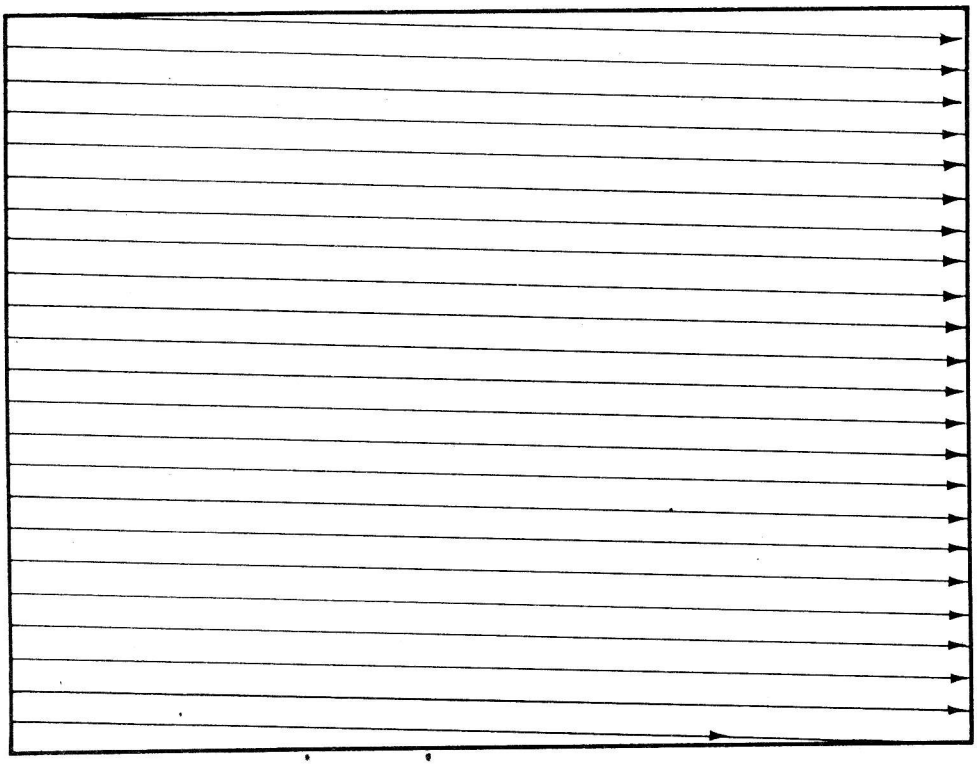
Jeder Monitor bzw. Fernseher führt diese Bewegung automatisch, d.h. auch ohne ein Eingangssignal aus und besitzt dazu einige "freilaufende" Generatoren.

Ein Videosignal muß außer der reinen Bildinformation noch zusätzliche Steuersignale enthalten, die dem Fernseher mitteilen, wann eine Zeile (horizontaler Sync) und wann ein Bild beendet ist (vertikaler Sync).

Jeder horizontale Sync-Impuls veranlaßt eine sehr schnelle Bewegung des Elektronenstrahls auf den Anfang der nächsttieferen Zeile, jeder vertikale Sync-Impuls bewegt ihn auf die oberste Zeile des Bildschirms. Dadurch ist im Normalfall eine präzise Kopplung gewährleistet, denn die Generatoren des Fernsehers laufen im "freien" Zustand immer etwas langsamer als zu erwarten wäre. Laufen sie zu schnell, dann ist der Fernseher eher mit dem Bild "fertig" als die Signalquelle - das Bild "läuft".

Die europäische PAL-Fernsehnorm gibt für den horizontalen Durchlauf eine Frequenz von 15.625 kHz und für den vertikalen Durchlauf eine Frequenz von 25 Hz bei 625 Zeilen vor, wobei ein Bild aus zwei Halbbildern mit jeweils 312.5 Zeilen besteht und in einer fünfzigstel Sekunde ausgegeben wird.

Bild 3.7 Stark vergrößerter Ausschnitt eines Fernsehbildes



Horizontaler und vertikaler Sync-Impuls werden ausgegeben, wenn das PICTURE-Signal den Wert "0" hat ("Schwarzschulter"). Der Elektronenstrahl wird durch den Sync-Impuls zum Anfang der nächsten Zeile bewegt, während der Ausgabe dieser Zeile bestimmt PICTURE die Helligkeit einzelner Punkte (d.h. beim Apple nur "an" oder "aus").

Die zeitliche Abstimmung zwischen Bildinformation, horizontalem und vertikalem Sync wird von einem Zähler übernommen, dessen Zählperioden der Länge einer Fernsehzeile bzw. der Länge eines Fernsehbildes entsprechen. Dieser Zähler ist der Videoscanner.

Der Videoscanner

Der Videoscanner ist ein Zähler innerhalb der IOU, der in derselben Weise zählt, wie ein Fernseher arbeitet (s. Bild 3.8). Die niederwertigen Bits (HPE'-H5-H4-H3-H2-H1-H0) bilden eine Zeile, die in derselben Zeit durch stetige Erhöhung des Zählerstands durchlaufen wird wie eine Fernsehzeile. Die höherwertigen Bits (V5-V4-V3-V2-V1-V0-VC-VB-VA) bilden die vertikale Position, ihr Stand wird nach jeder Fernsehzeile um 1 erhöht. Innerhalb der IOU werden aus den beiden Zählerständen die horizontalen und vertikalen Sync-Signale gebildet.

Da der Stand des Videoscanners über die durch ihn erzeugte Synchronisation den Elektronenstrahl des Fernsehers an sich bindet, ist der Vergleich statthaft, daß der Videoscanner sich wie ein Elektronenstrahl durch den Bildspeicherbereich des RAMs bewegt: zwischen beiden besteht eine direkte Zuordnung - sie durchlaufen ihren Bereich zyklisch und jedem Punkt des Bildschirms ist genau ein Bildpunkt des Speichers zugeordnet, der Elektronenstrahl befindet sich jedesmal wieder auf derselben Stelle in der Bildröhre, wenn der Videoscanner innerhalb eines Durchlaufs wieder auf dieselbe Speicherstelle zugreift.

Der Stand des Videoscanners wird bei jeder steigenden Taktflanke von RAS' während PHASE1 erhöht. Der Zähler arbeitet genau wie die CPU mit 1 MHz, der Zählerstand ist also jeweils für eine Mikrosekunde konstant. Während dieser Zeit bewegt sich der Elektronenstrahl genau um die Breite eines Zeichens im Modus TEXT40, um die Breite eines LoRes-Blocks oder um sieben HiRes40-Punkte von links nach rechts.

Tabelle 3.2 zeigt die möglichen Zustände der horizontalen und der vertikalen Zählersektion sowie einige Aktionen, die durch bestimmte Zählerstände ausgelöst werden. Damit diese Tabelle nicht über acht Seiten geht, ist nur jeder vierte Stand des Vertikalzählers gelistet. Zu den ausgelösten Aktionen gehören die Synchronisation, der Color Burst, die Umschaltung zwischen GRAPHICS und TEXT für den Modus MIXED, VBL (vertikale Schwarzschulter) und HBL (horizontale Schwarzschulter). Zweck dieser Tabelle ist ein Überblick der Kontrollfunktionen des Videoscanners für die Erzeugung des Videosignals. Die ausgelösten Aktionen sind in Kapitel 8 besprochen, die Details der RAM-Adressierung in Kapitel 5.

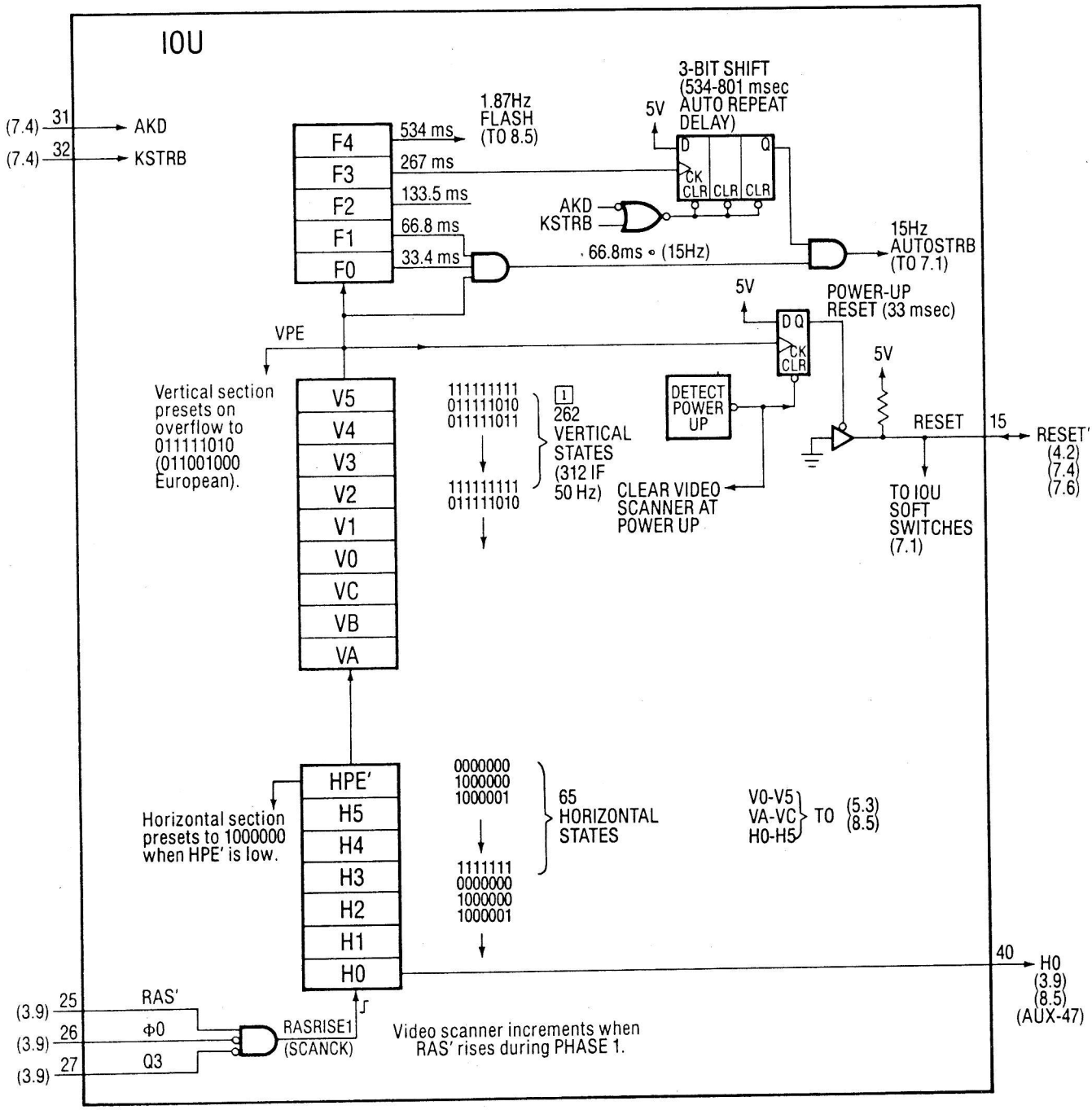
Der Horizontalzähler

Der Videoscanner läßt sich in zwei Sektionen unterteilen, den horizontalen und den vertikalen Zähler. Der horizontale Zähler besteht aus H0...H5 und HPE' (Horizontal Preset Enable = Aktivierung einer Vorgabe für den horizontalen Zählerstand). Diese sieben Bit sind die Ausgangssignale eines Zählers mit insgesamt 65 möglichen Zuständen, der bei jeder steigenden Flanke von RAS' während PHASE1 erhöht wird. Die 65 Zustände liefern die Ausgangssignale 0000000 und 1000000 bis 1111111. HPE' ist nur während einer der 65 Zustände aktiv (0000000) und setzt den Zähler mit dem nächsten Impuls auf den Zustand 1000000.

Die IOU gibt für jeden kompletten Durchlauf des Zählers einen horizontalen SYNC-Impuls ab, ein Durchlauf entspricht also genau einer Fersehzeile. Während 40 dieser Zustände werden Bildinformationen ausgegeben, während der restlichen 25 Zustände ist die Bildausgabe "schwarz": sie bestehen aus einer Schwarzperiode links vor dem Zeilenbeginn (horizontal front porch) und einer Schwarzperiode rechts nach dem Zeilenende (horizontal back porch). In dieser Zeit wird der horizontale SYNC ausgegeben und der Elektronenstrahl läuft auf die linke Bildseite zurück (retrace).Die Dauer einer horizontalen Sequenz entspricht 64 normalen Prozessorzyklen und einem langen Zyklus. Das dauert zusammen 63.99 Mikrosekunden und ergibt eine Horizontalfrequenz von 15625.49 Hz, also sehr nahe an den von der Fernsehnorm geforderten 15.625 kHz.

H0, das niederwertigste Bit des Videoscanners, wird über Pin 40 der IOU direkt ausgegeben. Die anderen Bits werden in direkter Form rein intern verwendet und nur teilweise verzögert ausgegeben (SEGA, SEGB und SEGC) sowie in gemultiplexter und umgerechneter Form als RAM-Adresse (H0-H5). Einige weitere Ausgänge der IOU werden ebenfalls über den Zustand des Videoscanners gesteuert (GR+2, WNDW').

Bild 3.8 Funktionsschema des Videoscanners



1 In the 50 Hz IOU, the vertical section presets to 011001000, resulting in 312 vertical states.

Der Vertikalzähler

Der für die Zeilenzählung zuständige Teil des Videoscanners besteht aus den Bits VA, VB, VC und V0-V5. Der daraus gebildete Zähler wird jedesmal erhöht, wenn der Horizontalzähler überläuft (d.h. am Ende jeder Fernsehzeile).

Der Vertikalzähler kann insgesamt 312 Zustände annehmen, nämlich von 011001000 bis 111111111. Im Gegensatz zum Horizontalzähler handelt es sich dabei um eine einfache binäre Zählung aufwärts - jedesmal, wenn er 111111111 erreicht, wird er mit dem nächsten Zählimpuls auf den Stand 011001000 gesetzt. Eine typische Folge von Werten für den Horizontalzähler ist:

Vertikal	**Horizontal**
111100000	1111111
111100001	0000000
111100001	1000000

Im zweiten Schritt dieser Folge ist der Horizontalzähler übergelaufen, der Vertikalzähler wird um 1 erhöht. Im dritten Schritt wird der Horizontalzähler via HPE' auf 1000000 gesetzt.

Eine typische Sequenz mit einem Überlauf des Vertikalzählers sieht so aus:

Vertikal	**Horizontal**
111111111	1111111
011001000	0000000
011001000	1000000

Einmal pro Durchlauf des Vertikalzählers gibt die IOU ein vertikales SYNC-Signal aus - die 312 Zustände der Vertikalzählers entsprechen also einem kompletten Bilddurchlauf. Wenn Sie sich jetzt fragen, wo denn die anderen 313 von den 625 Zeilen geblieben sind: Unser PAL-System arbeitet im Zeilensprungverfahren mit "Halbbildern" - in einem Durchlauf werden die Zeilen 1,3,5,7,9... und im nächsten Durchlauf die Zeilen 2,4,6,8... vom Fernseher dargestellt (genaueres dazu in Kapitel 8). Während 192 der 312 Zustände des Vertikalzählers werden Bildzeilen ausgegeben, die restlichen 120 leeren Zeilen werden für einen Freiraum am oberen Bildschirmrand (vertical front porch) und einen unteren Rand (vertical back porch) verwendet, in dem der Elektronenstrahl in die obere linke Ecke des Bildschirms zurückläuft.

Tatsächlich kommt man mit wesentlich weniger Freiraum in vertikaler Richtung aus, nämlich ungefähr 70 Zeilen anstelle von 120. Da aber der gesamte Bildschirm der amerikanischen NTSC-Norm nur aus 262 Zeilen besteht, bleiben hier gerade noch die benötigten 70 Zeilen frei und daraus erklärt sich dann das Format von 192 aktiven Bildschirmzeilen. Elektronisch gesehen wäre es kein Problem gewesen, den europäischen Ausführungen des Apple ein paar aktive Zeilen mehr zu spendieren - für Software-Autoren wäre es allerdings eine Katastrophe.

Die Darstellung eines kompletten Bildes benötigt exakt 20280 (65 * 312) Prozessorzyklen. Die damit erreichte Bildwiederholfrequenz beträgt also 50.08 Hz - nur eine winzige Idee schneller als der Standard von 50 Hz für ein Halbbild.

Ein normaler Fernseher bildet, wie bereits gesagt, innerhalb eines Halbbildes die Zeilen 1,3,5,7,9... und innerhalb des zweiten Halbbildes die Zeilen 2,4,6,8... ab und erreicht so seine effektive Auflösung von 625 Zeilen.

Der Apple kennt keine ineinandergeschachtelten Zeilen und gibt jedesmal ein gesamtes Bild mit 312 Zeilen aus. Daraus ergibt sich eine kleine Abweichung vom PAL-Standard, in dem definiert ist, daß ein kompletter Bilddurchlauf mit 2 Halbbildern 625 mal solange dauert wie eine einzelne Zeile: bei der Videoerzeugung des Apple sind es nur 624 Zeilen in zwei Halbbildern. Allerdings wird dieser Unterschied innerhalb der großen Freiräume zu allen Seiten des Fernsehbildes ausgebügelt und erzeugt damit keine unangenehmen Nebenerscheinungen wie Flimmern oder ähnliches.

Andere Modelle und der Videoscanner

Der Apple //c besitzt keinen diskret (d.h. aus einzelnen Bauteilen) aufgebauten Oszillator, sondern einen in Hybridtechnik ausgeführten Baustein, dessen Frequenz einen ganz leichten Unterschied zum //e aufweist: er schwingt mit 14.250000 anstelle von 14.250450 MHz. Auf den Hauptplatinen der Rev. B des //e ist Platz für diesen Baustein frei, die entsprechenden Leiterbahnen sind ebenfalls vorhanden - vielleicht bekommen neuere Ausgaben des //e ebenfalls einen hybriden Oszillator.

Tabelle 3.2a Die Zustände des Videoscanners

HORIZONTAL SCANNING		
HOR. SECTION P 543 210	CK NO	HOR. EVENT
1 001 100	53	BURST
1 001 101	54	BURST
1 001 110	55	BURST
1 001 111	56	BURST
1 010 000	57	
1 010 001	58	
1 010 010	59	
1 010 011	60	
1 010 100	61	
1 010 101	62	
1 010 110	63	
1 010 111	64	
1 011 000	00	HBL'
1 011 001	01	↓
1 011 010	02	
1 011 011	03	
1 011 100	04	
1 011 101	05	
1 011 110	06	
1 011 111	07	
1 100 000	08	
1 100 001	09	
1 100 010	10	
1 100 011	11	
1 100 100	12	
1 100 101	13	
1 100 110	14	
1 100 111	15	
1 101 000	16	
1 101 001	17	
1 101 010	18	
1 101 011	19	
1 101 100	20	
1 101 101	21	
1 101 110	22	
1 101 111	23	
1 110 000	24	
1 110 001	25	
1 110 010	26	
1 110 011	27	
1 110 100	28	
1 110 101	29	
1 110 110	30	
1 110 111	31	

AMERICAN VERTICAL SCANNING		
VERTICAL SECTION 543 210 CBA	DISPLAY LINE NO	VERTICAL EVENTS
111 100 1XX	228-231	
111 101 0XX	232-235	
111 101 1XX	236-239	
111 110 0XX	240-243	
111 110 1XX	244-247	
111 111 0XX	248-251	
111 111 1XX	252-255	PRESET
011 111 01X	256-257	
011 111 1XX	258-261	
100 000 0XX	000-003	VBL', GR
100 000 1XX	004-007	↓ ↓
100 001 0XX	008-011	
100 001 1XX	012-015	
100 010 0XX	016-019	
100 010 1XX	020-023	
100 011 0XX	024-027	
100 011 1XX	028-031	
100 100 0XX	032-035	
100 100 1XX	036-039	
100 101 0XX	040-043	
100 101 1XX	044-047	
100 110 0XX	048-051	
100 110 1XX	052-055	
100 111 0XX	056-059	
100 111 1XX	060-063	
101 000 0XX	064-067	
101 000 1XX	068-071	
101 001 0XX	072-075	
101 001 1XX	076-079	
101 010 0XX	080-083	
101 010 1XX	084-087	
101 011 0XX	088-091	
101 011 1XX	092-095	
101 100 0XX	096-099	
101 100 1XX	100-103	
101 101 0XX	104-107	
101 101 1XX	108-111	
101 110 0XX	112-115	
101 110 1XX	116-119	
101 111 0XX	120-123	
101 111 1XX	124-127	

EUROPEAN VERTICAL SCANNING		
VERTICAL SECTION 543 210 CBA	DISPLAY LINE NO	VERTICAL EVENTS
011 010 1XX	268-271	
011 011 0XX	272-275	
011 011 1XX	276-279	
011 100 0XX	280-283	TEXT
011 100 1XX	284-287	↓
011 101 0XX	288-291	
011 101 1XX	292-295	
011 110 0XX	296-299	
011 110 1XX	300-303	
011 111 0XX	304-307	
011 111 1XX	308-311	
100 000 0XX	000-003	VBL', GR
100 000 1XX	004-007	↓ ↓
100 001 0XX	008-011	
100 001 1XX	012-015	
100 010 0XX	016-019	
100 010 1XX	020-023	
100 011 0XX	024-027	
100 011 1XX	028-031	
100 100 0XX	032-035	
100 100 1XX	036-039	
100 101 0XX	040-043	
100 101 1XX	044-047	
100 110 0XX	048-051	
100 110 1XX	052-055	
100 111 0XX	056-059	
100 111 1XX	060-063	
101 000 0XX	064-067	
101 000 1XX	068-071	
101 001 0XX	072-075	
101 001 1XX	076-079	
101 010 0XX	080-083	
101 010 1XX	084-087	
101 011 0XX	088-091	
101 011 1XX	092-095	
101 100 0XX	096-099	
101 100 1XX	100-103	
101 101 0XX	104-107	
101 101 1XX	108-111	
101 110 0XX	112-115	
101 110 1XX	116-119	
101 111 0XX	120-123	
101 111 1XX	124-127	

Tabelle 3.2b Die Zustände des Videoscanners

HORIZONTAL SCANNING		
HOR. SECTION P 543 210	CK NO	HOR. EVENT
1 111 000	32	
1 111 001	33	
1 111 010	34	
1 111 011	35	
1 111 100	36	
1 111 101	37	
1 111 110	38	
1 111 111	39	VERT+1
0 000 000	40	HBL
1 000 000	41	↓
1 000 001	42	
1 000 010	43	
1 000 011	44	
1 000 100	45	
1 000 101	46	
1 000 110	47	
1 000 111	48	
1 001 000	49	SYNC
1 001 001	50	SYNC
1 001 010	51	SYNC
1 001 011	52	SYNC

AMERICAN VERTICAL SCANNING		
VERTICAL SECTION 543 210 CBA	DISPLAY LINE NO	VERTICAL EVENTS
110 000 0XX	128-131	
110 000 1XX	132-135	
110 001 0XX	136-139	
110 001 1XX	140-143	
110 010 0XX	144-147	
110 010 1XX	148-151	
110 011 0XX	152-155	
110 011 1XX	156-159	
110 100 0XX	160-163	TEXT
110 100 1XX	164-167	↓
110 101 0XX	168-171	
110 101 1XX	172-175	
110 110 0XX	176-179	
110 110 1XX	180-183	
110 111 0XX	184-187	
110 111 1XX	188-191	
111 000 0XX	192-195	VBL, GR
111 000 1XX	196-199	↓ ↓
111 001 0XX	200-203	
111 001 1XX	204-207	
111 010 0XX	208-211	
111 010 1XX	212-215	
111 011 0XX	216-219	
111 011 1XX	220-223	
111 100 0XX	224-227	SYNC, TXT
		↓

EUROPEAN VERTICAL SCANNING		
VERTICAL SECTION 543 210 CBA	DISPLAY LINE NO	VERTICAL EVENTS
110 000 0XX	128-131	
110 000 1XX	132-135	
110 001 0XX	136-139	
110 001 1XX	140-143	
110 010 0XX	144-147	
110 010 1XX	148-151	
110 011 0XX	152-155	
110 011 1XX	156-159	
110 100 0XX	160-163	TEXT
110 100 1XX	164-167	↓
110 101 0XX	168-171	
110 101 1XX	172-175	
110 110 0XX	176-179	
110 110 1XX	180-183	
110 111 0XX	184-187	
110 111 1XX	188-191	
111 000 0XX	192-195	VBL, GR
111 000 1XX	196-199	↓ ↓
111 001 0XX	200-203	
111 001 1XX	204-207	
111 010 0XX	208-211	
111 010 1XX	212-215	
111 011 0XX	216-219	
111 011 1XX	220-223	
111 100 0XX	224-227	TEXT
111 100 1XX	228-231	↓
111 101 0XX	232-235	
111 101 1XX	236-239	
111 110 0XX	240-243	
111 110 1XX	244-247	
111 111 0XX	248-251	
111 111 1XX	252-255	PRESET
011 001 0XX	256-259	GR
011 001 1XX	260-263	↓
011 010 0XX	264-267	SYNC

NOTE: Shaded areas indicate display blanking.

Alle westeuropäischen Staaten benutzen eine Fernsehnorm, die auf 50 Hz basiert und - mit Ausnahme von Frankreich - das PAL-System. Spezielle Versionen des //e für das in Frankreich benutzte SECAM-System gibt es nicht, die Franzosen kommen deshalb ohne eine Zusatzkarte (entweder in Steckplatz 7 oder als Erweiterung der 80-Zeichen-Karte im speziellen Steckplatz) nicht aus.

Die amerikanische NTSC-Fernsehnorm basiert auf einer Netzfrequenz von 60 Hz und weist einige Unterschiede zum PAL-System auf: es werden 262.5 anstelle von 312.5 Linien pro Halbbild verwendet, die Frequenz des Color Burst ist eine andere, die Zeit für den Durchlauf einer Fernsehzeile ist ebenfalls leicht unterschiedlich.

Die Frequenz des Signals 14M beträgt deshalb in den amerikanischen Apples 14.31818 anstelle von 14.250450 MHz. Daraus ergibt sich über eine Teilung durch 4 die Frequenz von COLOR REFERENCE: 3.57 anstelle von 3.59 MHz. Für den horizontalen Durchlauf einer Fernsehzeile werden ebenfalls 64 normale und ein langer Zyklus verwendet, aufgrund der veränderten Frequenz von 14M ergeben sich auch hier leichte Unterschiede: die Zeilenfrequenz beträgt 15.700 kHz.

Der große Unterschied liegt in der Zeilenanzahl: hier haben wir 50 Fernsehzeilen pro Halbbild weniger. Aus diesem Grund wird der Vertikalzähler des Videoscanners nicht auf 011001000, sondern auf 011111010 gesetzt, es gibt nur 262 verschiedene Zustände. Aus der Zeilenfrequenz und 262 Zeilen anstelle von 312 ergibt sich dann die Bildfrequenz von 15700 / 262 = 59.92 Hz, die damit auch recht nahe an den geforderten 60 Hz liegt.

Der Grund für die Beschränkung auf 192 aktive Zeilen ist, wie bereits gesagt, darin zu suchen: Für den oberen und unteren Bildschirmrand bleiben im amerikanischen Fernsehsystem gerade noch jeweils rund 28 Zeilen als Sicherheitsmarge (4 Zeilen werden für den vertikalen Rücklauf des Elektronenstrahls benötigt) - gerade ausreichend, um den sicheren Betrieb des Apple auch an einem nicht so ganz exakt eingestellten Fernseher zu gewährleisten.

Der Flash-Zähler und "Power-up" RESET

Der Flash-Zähler besteht aus F0 bis F4 in Bild 3.8. Dieser Zähler zählt Bilddurchläufe - ich habe ihn Flash-Zähler genannt, weil F4 dazu benutzt wird, bei blinkenden Zeichen zwischen NORMAL und INVERSE hin- und herzuschalten. Technisch gesehen ist es nicht notwendig, das Blinken von Zeichen mit der Bilderzeugung zu synchronisieren (im Apple II war das auch nicht der Fall), aber der Videoscanner stellt bereits eine schön gleichmäßig tickende Signalquelle zur Verfügung, die innerhalb der IOU als Takt für den Flash-Zähler benutzt und auch durch Interrupts, DMA' und anderes nicht beeinflußt wird. Der Flash-Zähler übernimmt noch einige weitere Funktionen: die Verzögerung, bevor eine länger gedrückte Taste wiederholt wird (Autorepeat), die Wiederholfrequenz der Autorepeat-Funktion und der Verzögerung zwischen Einschalten der Stromversorgung und Erzeugung von RESET' ("Power-up RESET").

Der Flash-Zähler wird in keiner mir bekannten Dokumentation von Apple, Inc. erwähnt, die Bezeichnungen F0...F4 stammen deshalb von mir. Der Grund für meine Annahme, daß überhaupt ein derartiger Zähler innerhalb der IOU existiert, liegt darin, daß blinkende Zeichen, die Autorepeat-Funktionen und Power-up RESET jeweils genau nach einem Überlauf des Vertikalzählers ihren Zustand ändern. Außerdem implizieren die benutzten Frequenzen dieser Funktionen, daß sie über einen einfachen binären Zähler gesteuert werden, der jeweils erhöht (oder vielleicht heruntergesetzt) wird, wenn der Vertikalzähler einen Zählzyklus beendet hat.

Ein weiterer Schaltkreis, der sich in der Dokumentation von Apple nicht finden läßt, ist die Power-up RESET-Funktion. Wenn der Apple //e eingeschaltet wird, hält die IOU die Leitung RESET' für rund 39.6 Millisekunden aktiv (d.h. auf "0"). Wenn man den Videoscanner durch Setzen von CLKEN' auf "1" davon abhält, etwas zu zählen, dann bleibt RESET' ewig aktiv. Sobald man die Aktivierung von CLKEN' zuläßt, 14M damit an die IOU gelangt und den Videoscanner eine Weile zählen läßt, wird RESET' inaktiv - soweit ich messen konnte nach einer Zeit, die einem kompletten Durchlauf des Videoscanners von 000000000/0000000 bis 1111111111/1111111 entspricht. Daraus folgt zusätzlich, daß der Videoscanner nach Einschalten der Stromversorgung auf den Stand 000000000/0000000 gesetzt wird. Der in Bild 3.8 gezeichnete Schaltkreis erfüllt diese Bedingungen.[2]

Wie in Bild 3.8 dargestellt, übernimmt der Flash-Zähler auch die Erzeugung des Signals AUTOSTRB, eines "künstlichen" Tastatur-Strobes. Der Softswitch KEYSTROBE wird entweder über KSTRB durch einen realen Tastendruck oder über AUTOSTRB gesetzt. Wenn eine Taste über einen Zeitraum von 638 bis 958 Millisekunden gedrückt gehalten wird, dann alterniert AUTOSTRB mit einer Frequenz von rund 12.5 Hz, durch jeden "1"-Pegel wird KEYSTROBE erneut gesetzt. Dadurch wird eine rasche Wiederholung der gedrückten Taste simuliert. KEYSTROBE und daranhängende Elektronik finden Sie in Bild 7.1.

F3 des Flash-Zählers dient als Taktquelle für die Verzögerung zwischen Tastendruck und dem Start des Autorepeat. Das läßt sich daraus ableiten, daß - je nachdem, in welchem Moment eine Taste gedrückt wird - die Verzögerung um maximal 320 Millisekunden variiert. Diese Verzögerungszeit kann durch einen 2-Bit-Zähler oder durch ein 3-Bit-Schieberegister erreicht werden, wie in Bild 3.8 gezeigt. In beiden Fällen muß dieser Verzögerungsgenerator

---

2 Die Dauer des von der IOU erzeugten Power-up RESET kann nur gemessen werden, wenn sich kein Diskettencontroller in einem der Steckplätze befindet - der Controller hat einen eigenen Kreis, der RESET' beim Einschalten rund 100 Millisekunden aktiv hält und so die 39 Millisekunden der IOU überdeckt.

durch KSTRB zurückgesetzt werden, damit ein tatsächlicher Tastendruck die Autorepeatfunktion solange unterbricht, bis für die neu gedrückte Taste die Verzögerungszeit wieder abgelaufen ist.

Die unterschiedlichen Verzögerungszeiten zwischen Tastendruck und Start des Autorepeat empfinde ich als Schönheitsfehler - auf einen wirklich geübten Maschinenschreiber wirken sie mehr als irritierend. Ich hätte das 3-Bit-Schieberegister durch einen 4-Bit-Zähler ersetzt, der über VPE seine Taktimpulse erhält und in derselben Weise funktioniert wie die ersten vier Bit des Flash-Zählers - abgesehen davon, daß dieser Zähler durch KBSTRB oder AKD' zurückgesetzt werden müßte. Der Überlauf dieses Zählers würde die Schwingung von AUTOSTRB aktivieren - damit erzeugte Verzögerungen lägen im Bereich von 641 bis 661 Millisekunden.

Die Dauer einzelner Aktionen, die durch den Flash-Zähler kontrolliert werden, ist im allgemeinen ein exaktes Vielfaches der Zeit, die für einen kompletten Bilddurchlauf benötigt wird. Die Ausnahme stellt die Erzeugung von RESET' dar - sie dauert sowohl auf den amerikanischen als auch auf den europäischen Ausführungen des Apple //e immer 512 Fernsehzeilen, weil hier der Videoscanner mit einem Wert von 000000000 startet. Dauer und Frequenzen der einzelnen Aktionen sind in Tabelle 3.3 gelistet, für die amerikanische Version müssen alle Zeiten außer RESET' ungefähr durch 1.2 dividiert werden.

Tabelle 3.3 Durch den Flash-Zähler kontrollierte Aktionen

Aktion	**Dauer/Frequenz**	**Anmerkungen**
Power-up RESET	32.6 msec	512 Fernsehzeilen
Flash-Zyklus	1.58 Hz	Vertikalfrequenz / 32
Autorepeat-Verz.	638-958 msec	32 - 48 Bilddurchläufe
Autorepeat-Freq.	12.52 Hz	Vertikalfrequenz / 4

Der lange Zyklus

In den vorhergegangenen Besprechungen sind wir bereits in begrenztem Maßstab auf den langen Zyklus eingegangen - warum er überhaupt benötigt wird, ist allerdings noch unklar.

Innerhalb jeder Fernsehzeile beginnt die "aktive" Ausgabe beim Stand 1011000 des Horizontalzählers (während der vorherigen Zustände wird der linke Rand des Bildes - d.h. "nichts" - erzeugt).

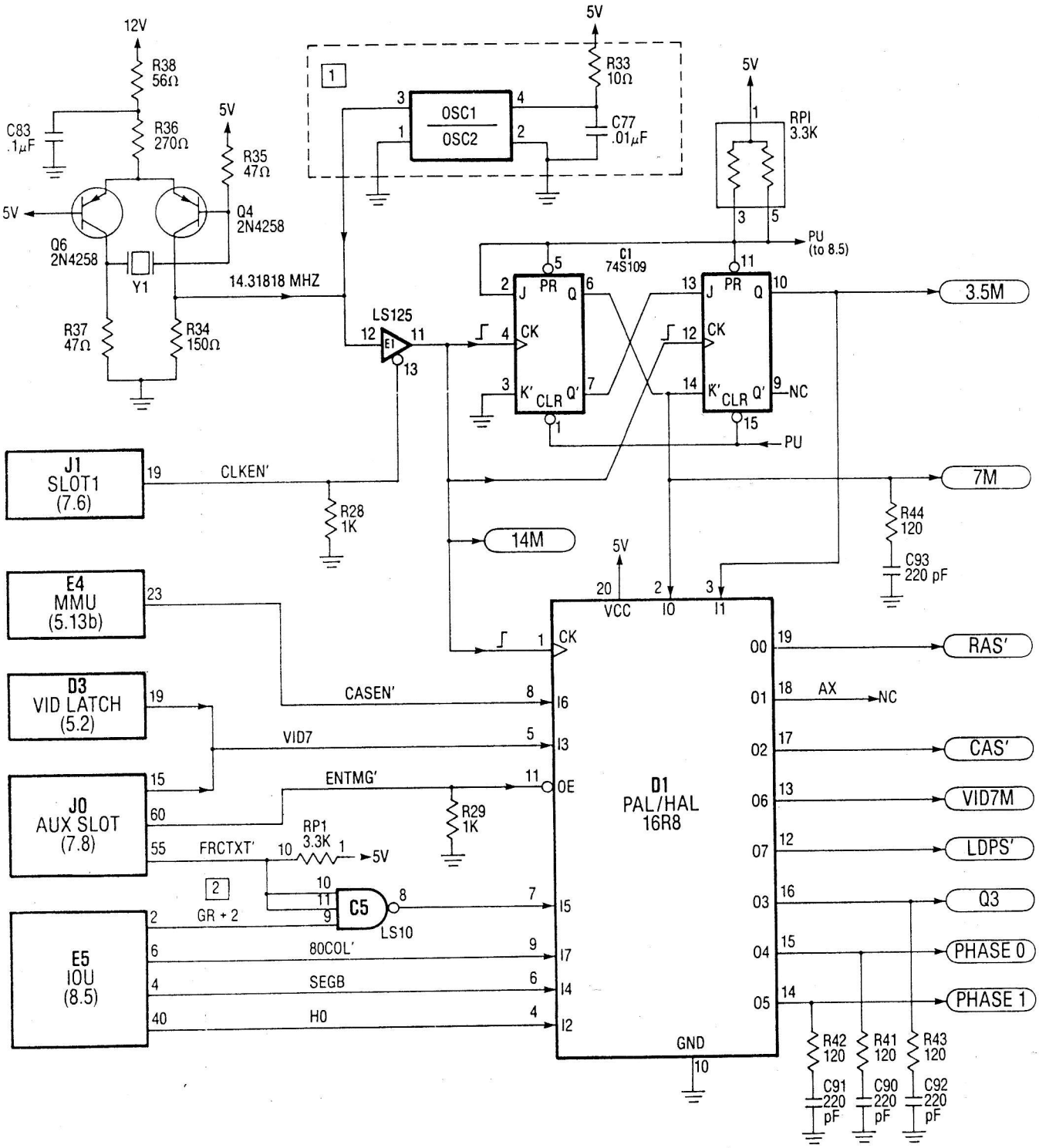
Die in HiRes40 erzeugten Farben werden durch die *Phasenlage* von PICTURE zu COLOR REFERENCE bestimmt, die sich für jedes Byte um 180 Grad ändert - das ist der Grund, warum durch Bits mit gerader Horizontaladresse andere Farben dargestellt werden können als auf ungeraden Adressen.

COLOR REFERENCE hat eine Frequenz von 3.5 MHz, der Videoscanner hat eine Frequenz von 1 MHz, pro Fernsehzeile werden 65 Zustände des Scanners und 227.5 Zyklen von COLOR REFERENCE durchlaufen. Aus dem überzähligen halben Zyklus von COLOR REFERENCE ergibt sich eine Verschiebung der Phasenlage um 180 Grad für jede neue Zeile. Zur Korrektur wird jeder 65.Zyklus um genau die Hälfte einer Periode von COLOR REFERENCE verlängert - mit dem Ergebnis, daß sich pro Fernsehzeile 228 Zyklen von COLOR REFERENCE ergeben. Als Nebeneffekt ergibt sich für alle 1- und 2-MHz-Signale eine entsprechende Verlängerung.

Die Hardware des Taktgenerators

Die Takterzeugung des Apple //e macht aus sehr wenigem sehr viel - das Signal 14M wird wiederholt geteilt, und daraus entstehen die langsameren, aber teilweise sehr komplex aufgebauten Taktsignale. Der überwiegende Teil der Erzeugung findet innerhalb des HAL statt.

Bild 3.9 Schaltplan des Taktgenerators



Das Signal 14M wird von einem Quartz-Oszillator erzeugt und ist über einen der vier tri-State-Treiber eines 74LS125 gepuffert (s. Bild 3.9). Für eine Zusatzkarte in Steckplatz 1 ist es möglich, die Weiterleitung von 14M durch Setzen von CLKEN' auf "1" komplett zu unterbinden. Eine Karte im speziellen Steckplatz könnte danach theoretisch durch ein eigenes 14M-Signal die gesamte Zeitsteuerung auf der Hauptplatine bestimmen. Solange Steckplatz 1 unbesetzt oder zumindest Kontakt 19 dieses Steckplatzes nicht verbunden ist, wird die Leitung CLKEN' durch den Widerstand R28 konstant auf "0" gehalten und damit bleibt der LS125 aktiv.

Die Leitung CLKEN', die übrigens im alten Apple II nicht vorhanden ist, könnte für spezielle Prüfkarten verwendet werden - oder auch für einen Zusatz, der so verrückt ist, daß er meine Vorstellungskraft übersteigt. Man darf gespannt sein, ob da noch etwas von Apple, Inc. auf uns zukommt.[3]

Die Erzeugung von 7M und COLOR REFERENCE geschieht durch eine einfache Teilung von 14M. 7M ist 14M durch zwei geteilt, COLOR REFERENCE ist 7M durch zwei geteilt, wobei 7M seinen Zustand auf die steigende Flanke von 14M, COLOR REFERENCE seinen Zustand auf die fallende Flanke von 7M ändert (s. Bild 3.2). Beide Teilungen werden durch Flipflops in einem 74S109 vorgenommen, der hier aufgrund seiner größeren Ausgangsleistung ("fan-out") anstelle eines 74LS109 eingesetzt wird - schließlich wird 7M direkt von diesem Baustein zu allen Steckplätzen geführt und ist nicht mehr weiter gepuffert.

14M, 7M und COLOR REFERENCE führen zu den Eingängen des HAL, der daraus *alle* restlichen Taktsignale erzeugt - PHASE0, PHASE1, RAS', CAS', Q3, LDPS' und VID7M.

Der HAL

HALs ("Hard Array Logic" = festverdrahtete Logik) und PALs ("Programmable Array Logic" = programmierbare Logik) sind relativ neue Entwicklungen der Elektronikindustrie. Ein HAL ist nichts weiter als ein bereits vom Chip-Hersteller fest vorprogrammierter PAL-Baustein, dessen Herstellungskosten bei sehr großen Stückzahlen niedriger liegen. Normalerweise probiert man einen Aufbau mit PALs aus, die einzeln programmiert werden - wenn alles funktioniert und große Stückzahlen zu erwarten sind, dann wird beim Hersteller ein fest programmierter HAL in Auftrag gegeben.

Ein HAL/PAL besteht aus einer Reihe von Invertern, Gattern, Flipflops und anderen relativ einfachen Funktionselementen auf einem Chip und einer Verbindungsmatrix. Über die Knotenpunkte der Matrix können durch Programmierung elektrische Verbindungen der Funktionselemente hergestellt werden. Je nach Anwendungszweck gibt es PAL-Bausteine mit einer unterschiedlichen Zahl von Ein- und Ausgängen und internen Funktionselementen.

Der im Apple //e benutzte HAL-Baustein hat den Typ 16R8, d.h. er verfügt über 16 Eingänge und 8 Ausgänge, deren Spezifikationen der STTL-Serie entsprechen. Er enthält acht D-Flipflops, deren Eingänge über mehrere AND- und OR-Verknüpfungen gesetzt werden können. Alle acht Flipflops erhalten ihren Taktimpuls über 14M und reagieren auf die steigende Flanke dieses Signals. Daraus folgt, daß alle erzeugten Signale ungefähr dieselbe Laufzeit (rund 14 Nanosekunden) gegenüber der steigenden Flanke von 14M haben. Alle acht Ausgänge des HAL haben tri-State-Charakteristik und können abgeschaltet werden, indem eine Karte im speziellen Steckplatz die Leitung ENTMG' ("ENable TiMinG") auf den Pegel "1" bringt. Alle vom HAL erzeugten Ausgangssignale sind zum speziellen Steckplatz geführt und könnten dort nach Abschaltung der HAL-Ausgänge theoretisch durch Signale einer entsprechenden Karte ersetzt werden. Diese Möglichkeit dürfte ebenfalls zu Prüfzwecken vorgesehen sein - mir ist keine Karte bekannt, die eine Verbindung zu ENTMG' hat.

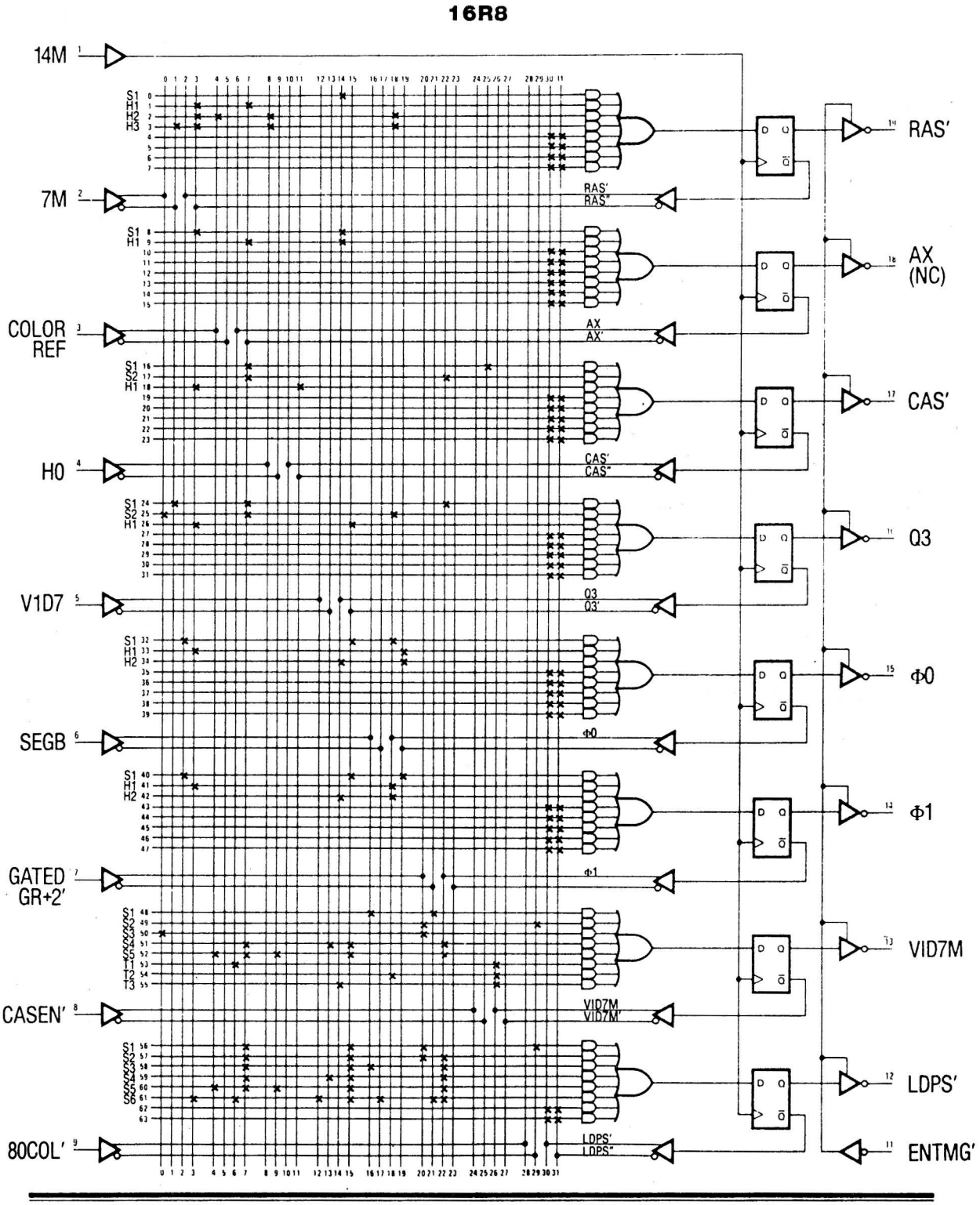
Bild 3.10 stellt die interne Schaltung eines 16R8 dar, die Programmierung (d.h. die einzelnen Verbindungen der Matrixknotenpunkte) wurden von mir so eingezeichnet, daß die resultierenden Funktionen dem HAL entsprechen, der sich auf der Hauptplatine des Apple //e, Revision B, befindet. Der HAL der Revision A ist mit Sicherheit anders programmiert, er erhält anstelle von GATED GR+2' das Signal GR+2 direkt und nicht invertiert. Die entsprechenden Verbindungen in der Matrix von GR+2 müssen deshalb umgekehrt angeordnet sein.[4]

---

3 Vielleicht die Karte mit dem 16-Bit-Prozessor, die seit längerem in der Gerüchteküche gehandelt wird?

4 Ein weiterer Unterschied besteht darin, daß die Phasenbeziehung von COLOR REFERENCE zu den anderen Signalen anders verläuft als in Bild 3.2 gezeigt. Der HAL von Rev. A würde deshalb in einer Rev. B-Hauptplatine recht merkwürdige Farben erzeugen.

Bild 3.10 Der Aufbau des HAL



Anm.:Die Zeichnung ist dem *Bipolar LSI 1984 Databook* (5. Auflage) mit freundlicher Genehmigung der Firma Monolithic Memories entnommen. Die Verbindungen der Knotenpunkte wurden von Jim Sather eingezeichnet.

Tabelle 3.4 Logische Definitionen der Ausgangsterme des HAL

PIN ASSIGNMENTS

ARRAY INPUTS	OUTPUTS	OTHER INPUTS
I0-2 = 7M	Q0'-19 = RAS'	CP-1 = 14M
I1-3 = CLR REF	Q1'-18 = AX	OE'-11 = ENTMG'
I2-4 = H0	Q2'-17 = CAS'	VCC-20 = +5V
I3-5 = VID7	Q3'-16 = Q3	GND-10 = GROUND
I4-6 = SEGB	Q4'-15 = Φ0	
I5-7 = GATED GR+2'	Q5'-14 = Φ1	
I6-8 = CASEN'	Q6'-13 = VID7M	
I7-9 = 80COL'	Q7'-12 = LDPS'	

SIGNAL	EQUATIONS	NOTES
RAS'	S1 = Q3 H1= RAS'' • AX' H2= RAS'' • CLR REF • H0 • Φ0 H3= RAS'' • 7M' • H0 • Φ0	FALL AFTER Q3 RISES RISE AFTER AX RISES LONG CYCLE DELAY LONG CYCLE DELAY
AX	S1 = RAS'' • Q3 H1= AX' • Q3	FALL AFTER RAS' FALLS RISE AFTER Q3 FALLS
CAS'	S1 = AX' • CASEN'' S2 = AX' • Φ1 H1= CAS'' • RAS''	MMU, MAY I? NUTS TO MMU DURING Φ1 RISE AFTER RAS' RISES
Q3	S1 = AX' • Φ1 • 7M' S2 = AX' • Φ0 • 7M H1= Q3' • RAS''	AX' • Φ1 • CAS' ALSO WORKS CAS' NO WORKEE RISE AFTER RAS' RISES
Φ0	S1 = Φ0 • RAS' • Q3' H1= Φ0' • RAS'' H2= Φ0' • Q3	TOGGLE AT RAS' • Q3
Φ1	S1 = Φ0' • RAS' • Q3' H1= Φ0 • RAS'' H2= Φ0 • Q3	Φ0 INVERTED
VID7M	S1 = GR'' • SEGB S2 = GR' • 80COL'' S3 = GR' • 7M S4 = VID7' • Φ1 • Q3' • AX' S5 = H0' • CLR REF • Φ1 • Q3' • AX' T1 = VID7M • AX T2 = VID7M • Φ0 T3 = VID7M • Q3	LORES GRAPHICS IS HIGH SPEED DOUBLE RES IS HIGH SPEED SAME AS 7M IF NOT HIRES FRCTXT'' • 80COL' ⟶ 7M, UNDELAYED HIRES DELAY CHECK AT Φ1 • Q3' • AX' NO DELAY AT RIGHT DISPLAY EDGE TOGGLE THROUGH AX KEEP TOGGLING THROUGH Φ0 KEEP TOGGLING THROUGH Q3
LDPS'	S1 = Q3' • AX' • 80COL'' • GR' S2 = Q3' • AX' • Φ1 • GR' S3 = Q3' • AX' • Φ1 • SEGB S4 = Q3' • AX' • Φ1 • VID7' S5 = Q3' • AX' • Φ1 • CLR REF • H0' S6 = Q3' • AX • RAS'' • Φ1 • VID7 • SEGB' • GR''	DOUBLE RES CAUSES DOUBLE LDPS' TEXT MODE LORES NOT DELAYED HIRES RIGHT DISPLAY EDGE CUTOFF HIRES DELAYED LDPS'

Bild 3.10 unterscheidet sich von der tatsächlichen Programmierung des HAL mit Sicherheit in kleineren Details - was die erzeugten Terme angeht, gibt es allerdings wenig Spielraum. Als ich mit der Analyse der inneren Verbindungen des HAL aufgrund der resultierenden Signale anfing, war es mir nicht so ganz klar, wie die benötigten Logikfunktionen mit der beschränkten Zahl von Eingängen ausgeführt werden können. Nach einigen Tagen mit rauchendem Kopf war dann klar, daß es doch geht - Hut ab vor den Leuten, die sämtliche Funktionen der Takterzeugung mit einigen Geniestreichen in einen einzigen Baustein gepackt haben.

Tabelle 3.4 listet die Ausgangsterme des HAL in Form logischer Gleichungen und dürfte von den meisten Lesern als verständlicher empfunden werden als Bild 3.10. Überhaupt sind sowohl Bild 3.10 als auch Tabelle 3.4 mehr der Vollständigkeit halber aufgenommen - in den meisten Fällen dürfte Bild 3.2 anstelle einer detaillierten Analyse ausreichend sein.

Falls Sie aber doch neugierig sein sollten, folgen hier einige interessante Details:

1. Alle Ausgänge werden über die Ausgangstreiber invertiert, d.h. sie haben den entgegengesetzten Pegel der Flipflops.
2. RAS', AX, CAS', Q3, PHASE0 und PHASE1 könnte man am besten als "set/hold"-Logik bezeichnen. Die Terme zum Setzen der dazugehörigen Flipflops werden nicht vom vorherigen Zustand des Flipflops beeinflußt (d.h. es findet kein Feedback statt); die Terme zum Halten eines Zustands dagegen werden durch den vorherigen Stand des Flipflops beeinflußt und nur dann wahr, wenn das Flipflop bereits vorher gesetzt war. Die dazugehörigen Flipflops werden gesetzt, sobald einer der "set"-Terme wahr wird und bleiben danach gesetzt, solange mindestens einer der beiden Terme "set" oder "hold" wahr ist (OR-Funktion).
3. RAS', AX, CAS' und Q3 funktionieren wie ein Schieberegister. Wenn Q3 "1" ist, dann erhalten mit den nächsten Taktimpulsen erst RAS', dann AX, dann CAS' und schließlich Q3 selber eine "0" - das Ganze mit einer speziellen Logik für Q3, weil CAS' nicht während PHASE0 aktiv wird, wenn CASEN' inaktiv ist. Wenn Q3 "0" ist, wird eine "1" zuerst zu RAS', dann zu AX und schließlich zu CAS' und Q3 gleichzeitig geschoben.
4. Die Verzögerungslogik für den langen Zyklus besteht aus den Termen H2 und H3 für das RAS'-Flipflop.
5. CASEN' von der MMU wird nicht über PHASE0 gesteuert, also muß das CAS'-Flipflop während PHASE1 unabhängig vom Zustand von CASEN' gesetzt werden (s. Term S2 des CAS'-Flipflops).
6. Die Eingänge SEGB, GATED GR+2', VID7 und 80COL' des HAL werden nur für die Erzeugung von LDPS' und VID7M benutzt. Keiner der anderen Ausgänge des HAL wird durch LDPS' und VID7M beeinflußt. Genaueres zu diesen beiden Signalen finden Sie in Kapitel 8.

Kapitel 4

Der Mikroprozessor 6502

Die 6502-CPU ("Central Processing Unit" = Zentrale Verarbeitungseinheit) ist der Schaltkreis des Apple //e, in dem die eigentliche Ausführung (sequentieller) Programme stattfindet. Physikalisch besteht er aus einem rechteckigen Plastikgehäuse mit 40 Pins (Anschlüssen). Er versteht nur eine einzige Sprache, nämlich seine eigene "Maschinensprache", und liest einzelne Befehle dieser Sprache vom Datenbus. Man kann ihn auch als das Gehirn (nicht aber als Gedächtnis!) des Apple bezeichnen.

Der 6502 wurde von der Firma MOS in der Mitte der siebziger Jahre als Teil der Mikroprozessorserie MCS6500 entwickelt. Er dürfte das bekannteste Mitglied dieser Familie sein - außer in der Apple-Serie findet man ihn in Computern der Hersteller Atari, Commodore, Ohio Scientific, Rockwell International und anderen. Der Prozessor hat ein gutes Preis-Leistungsverhältnis und ist darüber hinaus sehr zuverlässig. Er kann natürlich mit den letzten Entwicklungen wie einem V30 oder 680xx nicht mithalten, dafür kostete er bereits Anfang der siebziger Jahre kein Vermögen mehr (rund 20 Dollar). Die vom 6502 benutzte Maschinensprache ist sehr einfach - dadurch ist dieser Prozessor auch gut für Leute geeignet, die nur hobbymäßig oder gelegentlich programmieren. Womit sich ein Apple-Programmierer am meisten beschäftigen muß, ist deshalb weniger ein überaus komplizierter Befehlssatz, sondern mehr die Kunst des Programmierens an sich.

Wer 6502-Maschinenprogramme lesen und schreiben kann, dem eröffnet sich ein weites Betätigungsfeld auf dem Apple - allerdings ist diese Sprache kein Hauptthema dieses Buchs, wir werden uns dafür ausführlicher mit der dahinterstehenden Elektronik beschäftigen (über die Programmierung des 6502 sind bereits ganze Bibliotheken geschrieben worden...).

Der Grund, warum der 6502 dennoch in einem eigenen Kapitel behandelt wird, liegt in der teilweise einzigartigen Anwendung der Möglichkeiten dieses Prozessors im Apple und darin, daß über seine hardwaremäßigen Anwendungsmöglichkeiten eine gewisse Lücke im allgemeinen Literaturangebot besteht.

Die Signale des 6502

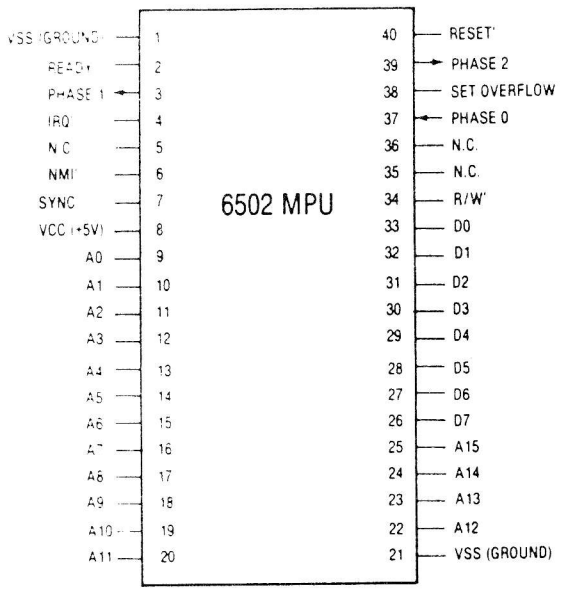
Wie bereits gesagt, verfügt der 6502 über 40 Anschlüsse, von denen drei überhaupt nicht verbunden sind. Neben den sechzehn Leitungen der Adreßausgabe und den acht Leitungen zur Datenein- und -ausgabe besitzt er vier Ausgänge (R/W', PHASE1, PHASE2 und SYNC) sowie sechs Eingänge (READY, IRQ', NMI', PHASE0, SET OVERFLOW' und RESET'). Dazu kommen noch drei(!) Anschlüsse zur Stromversorgung: über einen erhält er +5 Volt, die anderen beiden sind im Betrieb mit Masse verbunden. Bild 4.1 zeigt die Zuordnung und die Namen der einzelnen Anschlüsse, Bild 4.2 zeigt, wie der Prozessor in die Elektronik des Apple integriert ist. Die nächsten Abschnitte geben einen Abriß der Funktionen einzelner Anschlüsse mit dem Schwerpunkt auf "Apple-Spezialitäten".

Taktsignale - PHASE0, PHASE1, PHASE2

Der 6502 verfügt chip-intern über die notwendige Elektronik zur Erzeugung von Taktimpulsen und benötigt nur eine externe (einphasige) Taktquelle. Im Apple geschieht die Erzeugung der Taktimpulse unabhängig von der internen Taktelektronik, das Taktsignal wird extern erzeugt und über PHASE0 "eingefüttert".

Aus den über PHASE0 zugeführten Taktimpulsen erzeugt der 6502 die Ausgangstakte von PHASE1 und PHASE2. PHASE1 ist während der ersten Hälfte eines Prozessorzyklus "1", PHASE2 ist es während der zweiten Hälfte; sie ist aber keine schlichte Umkehrung von PHASE1, denn PHASE2 beginnt erst dann von "0" auf "1" zu steigen, wenn die fallende Flanke von PHASE1 auf "0" angekommen ist. Diese beiden Taktausgänge sind über die Pins 3 und 39 nach außen geführt - benutzt werden sie im Apple allerdings nicht, sie werden nur prozessorintern verwendet und enden sozusagen in der IC-Fassung des 6502.

Bild 4.1 Die Anschlußbelegung des 6502



Die Adreßausgänge und R/W'

Während jedes Prozessorzyklus gibt der 6502 über die Adreßausgänge eine Adresse aus. Im gleichen Zeitraum wird der Ausgang R/W' gesetzt, damit die Außenwelt mitbekommt, ob von dieser Adresse gelesen oder zu dieser Adresse geschrieben werden soll. Über die 16 Adreßleitungen kann der 6502 insgesamt 65536 (= 2 hoch 16) verschiedene Adressen erzeugen.

Die Adreßausgänge und der Ausgang R/W' haben *keine* tri-State-Charakteristik; deshalb sind sie im Apple nicht direkt, sondern über eine entsprechende Anzahl von tri-State-Treibern mit dem Rest des Computers verbunden. Die Treiber können über das Signal DMA' in den hochohmigen Zustand geschaltet werden. Auf diese Weise kann eine Zusatzkarte den Adreßbus unabhängig vom Prozessor in Beschlag nehmen.

Die Verbindung zum Datenbus

Der 6502 verfügt über acht Anschlüsse, über die Daten ein- oder ausgegeben werden können, und wird hauptsächlich deshalb als 8-Bit-Prozessor bezeichnet (auch wenn diese Definition seit dem Auftauchen von 32-Bit-Prozessoren mit acht Datenleitungen (Beispiel: 68008) etwas wacklig geworden ist). Alle acht Anschlüsse sind immer in Richtung "Lesen", also "in den Prozessor hinein" geschaltet, nur während PHASE2 können sie auf "Schreiben" gesetzt werden. Dadurch ist hier keine tri-State-Charakteristik notwendig, die Anschlüsse sind im Apple direkt mit dem Datenbus verbunden.

RESET'

Ein "0"-Impuls an diesem Eingang erzwingt einen (Neu-)Start des Prozessors. Interrupts werden gesperrt und der 6502 beginnt die Ausführung eines Programms, dessen Startadresse er von den Speicherstellen $FFFC/D liest. Solange RESET' den Pegel "0" hat, bleibt der Prozessor inaktiv, das Lesen der Startadresse von $FFFC/D beginnt erst dann, wenn der Pegel von RESET' wieder auf "1" steht.

Im Apple //e ist die Leitung RESET' mit Kontakt 31 aller "normalen" Steckplätze, mit der Taste RESET und mit Pin 15 der IOU verbunden. Im Normalbetrieb wird sie durch einen mit +5 Volt verbundenen Widerstand auf "1" gehalten. Eine Zusatzkarte kann RESET' auf "0" setzen und/oder auf den Pegel dieser Leitung reagieren. Die IOU tut das auch - zum einen reagiert sie auf RESET' und setzt einige Softswitches zurück, zum anderen erzeugt sie einen 33 Millisekunden langen "0"-Impuls auf dieser Leitung nach Einschalten der Stromversorgung.

Interrupts - IRQ' und NMI'

Ein "Interrupt" (eigentlich: Unterbrechungsanforderung) zwingt den 6502, die normale Abarbeitung eines Programms zu unterbrechen und ein spezielles Programm "dazwischenzuschieben". Interrupts werden normalerweise für Ein- und Ausgabefunktionen verwendet - anstelle einer immer wieder in ein Programm "eingestrickten" Abfrage, ob der Benutzer mittlerweile eine Taste gedrückt hat oder ob ein Drucker inzwischen bereit ist, ein neues Zeichen anzunehmen, kann man auf diese Weise den Prozessor gezielt nur dann unterbrechen, wenn ein entsprechendes Ereignis auch wirklich eingetreten ist. Im Prinzip kann jede angeschlossene Elektronik auf diese Weise beim Prozessor "anklopfen". Der Eingang IRQ' ("Interrupt Request" = Unterbrechungsanfrage) kann programmgesteuert aktiviert oder gesperrt werden, d.h. ein Programm kann entscheiden, ob derartige Unterbrechungen zugelassen werden oder nicht. Der Eingang NMI' ("Non-Maskable Interrupt" = nicht maskierbare Unterbrechung) kann nicht mittels Programmkontrolle abgeschaltet werden und erzwingt eine Unterbrechung, unmittelbar nachdem der momentan anliegende Befehl vollständig ausgeführt worden ist.

Der 6502 speichert bei einem Interrupt seinen Programmzähler (PC) und das Statusregister (P) in einem speziellen RAM-Bereich (dem Stack), sperrt automatisch den Eingang IRQ' und beginnt die Ausführung eines Programms, dessen Startadresse auf den Speicherstellen $FFFE/FFFF (IRQ') bzw. auf den Speicherstellen $FFFA/$FFFB (NMI') stehen muß.

Der Eingang NMI' reagiert auf fallende *Flanken* - um einen zweiten NMI' auszulösen, muß der Eingang zwischendurch erst wieder auf den Pegel "1" gebracht werden.

Der Eingang IRQ' reagiert dagegen auf den *Pegel* "0" und muß vor Ende der Interrupt-Behandlungsroutine wieder inaktiv werden - sonst führt der 6502 denselben Interrupt ein zweites Mal aus.

Am Ende einer durch einen Interrupt gestarteten Routine steht normalerweise der Prozessorbefehl "RTI" ("Return from Interrupt"). Der Prozessor holt daraufhin den Stand des Programmzählers und das Statusregister, stellt so den alten Zustand des IRQ-Sperrbits wieder her und macht dann im vorher unterbrochenen Programm weiter.

Die Eingänge NMI' und IRQ' sind im Apple zu allen "normalen" Steckplätzen geführt, IRQ' ist mit Kontakt 30, NMI' mit Kontakt 29 verbunden. Auf der Hauptplatine gibt es keine Baugruppen, die Interrupts erzeugen, Ein- und Ausgabe werden im Normalfall durch wiederholte Abfrage und nicht durch Interrupts behandelt.

Die "Maus" von Apple erzeugt einen IRQ', die meisten Uhrenkarten sind ebenfalls dazu in der Lage. "Programmklaukarten" ("WildCard", "Snapshot") erzeugen einen NMI'.

READY

Wenn dieser Eingang während PHASE1 auf den Pegel "0" gebracht wird, geht der 6502 in einen Wartezustand, d.h. er hält die gerade ausgegebene Adresse stabil und tut sonst nichts mehr - solange, bis READY während einer PHASE2 wieder den Pegel "1" hat. Falls READY während eines Schreibzyklus plötzlich auf "0" geht, beginnt der Wartezustand erst mit dem nächsten Zyklus.

Dieser erzwungene Wartezustand kann für langsame Speichereinheiten, schrittweises oder verlangsamtes Abarbeiten eines Programms oder für ein komplettes Anhalten des Prozessors benutzt werden.

Im Apple hat der Wartezustand des Prozessors keinen Einfluß auf die Bilderzeugung oder den notwendigen Refresh der dynamischen RAMs - der Bildschirm friert schlicht ein, wenn READY auf "0" gebracht wird. READY ist im Apple //e nur mit Kontakt 21 aller "normalen" Steckplätze verbunden und wird sonst nirgendwo genutzt - diese Fähigkeit des Prozessors ist praktisch unter den Tisch gefallen.

SYNC

Der Ausgang SYNC des 6502 geht immer auf den Pegel "1", wenn der Prozessor einen Opcode ("Operation Code" = Befehlscode) liest. Jede Anweisung hat als erstes Byte einen Opcode, der dem Prozessor sagt, wieviele Bytes die gesamte Anweisung umfaßt und was in dieser Anweisung geschehen soll. Das SYNC-Signal kann zusammen mit dem READY-Eingang für die Abarbeitung eines Programms in Einzelschritten verwendet werden - im Apple ist es lediglich zu Kontakt 39 aller "normalen" Steckplätze geführt und wird nicht weiter verwendet.

SET OVERFLOW'

Eine negative Flanke an diesem Eingang des 6502 setzt die Überlauf-Flagge ("V-Flag") im Statusregister des Prozessors. Diese Flagge wird normalerweise als Ergebnis von Rechenoperationen gesetzt oder zurückgesetzt. Ein Pegelwechsel von "1" auf "0" an SET OVERFLOW' setzt sie immer - egal, welche Art von Befehl der Prozessor gerade ausführt.

Dieser Eingang ist zu Kontrollzwecken nur sehr beschränkt brauchbar (es sei denn zusammen mit einem der Interrupt-Eingänge als Kennung der Interruptquelle). In allen anderen Fällen müßte ein komplettes Programm so geschrieben sein, daß es entweder den Stand der V-Flags ständig berücksichtigt oder sämtliche entsprechenden Befehle vermeidet, um nicht in Konflikt mit einem eventuellen Kontrollsignal zu kommen. Im Apple //e beginnt und endet die dazugehörige Leitung in der IC-Fassung des Prozessors, d.h. dieser Eingang hat keine weitere Verbindung.

Die Verbindungen des 6502 im Apple //e

Bild 4.2 zeigt sämtliche Verbindungen, die der 6502 zur restlichen Elektronik des Apple //e inklusive der Steckplätze hat.

R/W' und die Adreßausgänge sind mit dem Adreßbus der Hauptplatine über zwischengeschaltete tri-State-Treiber verbunden, die Datenanschlüsse dagegen direkt mit dem Datenbus. Das Signal PHASE1 des Taktgenerators wird über ein Gatter invertiert, kann über DMA' abgeschaltet werden und führt von da aus zum Eingang PHASE0 des Prozessors. Alle anderen Signale des 6502 sind direkt mit den Steckplätzen verbunden und haben (außer RESET') keine weitere Verbindung auf der Hauptplatine.

Sämtliche Steckplätze sind, was diese Kontrollsignale betrifft, direkt miteinander verbunden. Frage: Wie wird ein Kurzschluß verhindert, wenn eine Zusatzkarte einen Kontrolleingang des Prozessors auf "0" bringt und andere, ebenfalls mit dieser Leitung verbundene Karten eine "1" liefern?

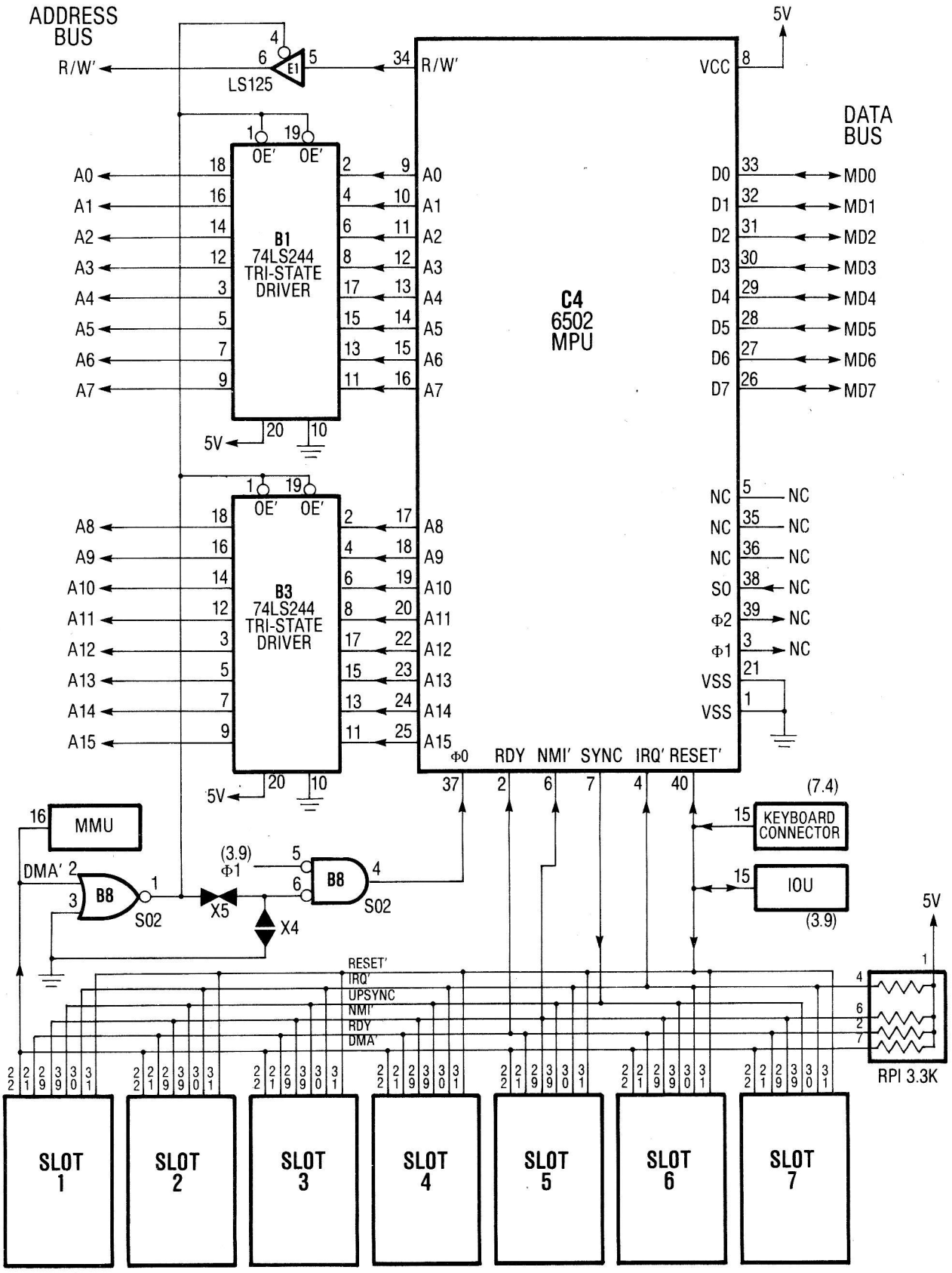
Anstelle einer gesonderten Leitung, über die die Ausgänge aller anderen Karten inaktiv geschaltet werden, und dem damit verbundenen Aufwand an Elektronik ist eine einfachere Lösung möglich, die als *wired OR* ("open collector OR" oder "verbundenes ODER") bezeichnet wird. Die Leitung wird im "Normalbetrieb" durch einen mit +5 Volt verbundenen Widerstand ("pull-up") ständig auf "1" gehalten. Alle an diese Leitung angeschlossenen Bausteinausgänge können selber keine "1" erzeugen und verhalten sich im inaktiven Zustand wie ein offener Schalter. Im aktiven Zustand dagegen wird eine Verbindung zur Masse geschaltet ("Schalter zu"), die Leitung wird auf "0" heruntergezogen ("pull-down"). Alle anderen an dieser Leitung angeschlossenen inaktiven Schalter interessiert das wenig - schließlich sind die entsprechenden Ausgänge "offen". Werden mehrere Schalter gleichzeitig aktiv, wird die Leitung eben an mehreren Stellen mit Masse verbunden, das Ergebnis ist dasselbe, als wäre nur ein einziger Schalter aktiv geworden.

In den meisten Fällen wird die entsprechende Schalterfunktion durch ein TTL-Gatter mit offenem Kollektorausgang oder von einem schlichten Transistor übernommen.

Die Datenblätter für den 6502 geben an, daß der Widerstand, der die entsprechenden Eingänge auf "1" hält, einen Wert von 3000 Ohm haben sollte. Im Apple werden Widerstände mit 3300 Ohm benutzt, die sich allesamt in einem einreihigen ("SIP") Widerstandsnetzwerk befinden.

Die tri-State-Treiber für die Adreßleitungen und für R/W' sind für DMA-Operationen notwendig, weil die entsprechenden Ausgänge des 6502 keine tri-State-Charakteristik haben. Im Apple werden dazu zwei LS244-Bausteine mit jeweils 8 Treibern für die Adreßleitungen und ein Viertel eines LS125 für R/W' benutzt.

Bild 4.2 Schaltplan der 6502-Anschlüsse im Apple //e



Die Speicherbelegung des 6502

Durch die Verwendung eines 6502 sind Teile des Speicheraufbaus des Apple festgelegt - ein 6502-System muß z.B. im Bereich der Adressen von $0000 bis $01FF immer RAM zur Verfügung haben. Die Seite 0 ($00-$FF) wird für *indirekte* und *indexierte Adressierung* verwendet. Darüber hinaus verfügt der 6502 über eine spezielle Adressierungsart für diesen Speicherbereich, die weniger Taktzyklen benötigt als der Zugriff auf andere Speicherbereiche. Programme, die oft Gebrauch von dieser Adressierungsart machen, sind kürzer und werden schneller ausgeführt. Aus diesem Grund belegen große Maschinenprogramme wie Applesoft BASIC die Seite 0 fast vollständig - überlegen Sie deshalb genau, wie und was Sie in diesen Speicherbereich hinein"POKE"n, da möglicherweise Pointer (Zeiger) und andere wichtige Systemwerte von Applesoft zerstört werden. Das folgende Programm führt mit Sicherheit zu einem Systemabsturz und sollte deshalb mit Vorsicht genossen werden:

```
10 FOR X = 0 TO 255: POKE X,0: NEXT
```

Seite 1 ist der Stack des 6502. Für jeden Aufruf eines Unterprogramms - egal, ob von BASIC aus mit GOSUB, innerhalb eines Maschinenprogramms mit JSR ("Jump to SubRoutine") oder auch durch einen Interrupt - speichert der Prozessor hier den Stand des Programmzählers, damit er weiß, an welcher Stelle des Hauptprogramms er nach Ende des Unterprogramms weitermachen soll. Der Stack ist dabei raffiniert organisiert: das, was zuletzt darin gespeichert wurde, kommt als erstes wieder heraus.

Ein Programm der Form Haupt-, Unter-,Unterunterprogramm etc. wird von oben nach unten durchlaufen. Beim Aufruf des Unterprogramms muß sich der Prozessor die Adresse im Hauptprogramm merken, mit der es nachher weitergeht. Beim Aufruf des Unterunterprogramms muß er sich zusätzlich die entsprechende Adresse im Unterprogramm merken. Die "gemerkten" Adressen stapeln sich also mit zunehmender Verschachtelung immer weiter auf.

Tatsächlich besteht der mysteriöse Stack, an dem schon viele Erklärungsversuche gescheitert sind, aus nichts weiter als einem RAM-Bereich und einem dazugehörigen Register des Prozessors, dem *Stack Pointer* ("Stapelspeicher-Zeiger" oder auch "Kellerstapelspeicherrücksprungadressenzeiger"). Dieser Zeiger zeigt am Anfang eines Programms auf die Adresse $1FF. Bei einem ersten Unterprogramm-Aufruf wird der Programmzähler auf $1FF und $1FE gespeichert, der Stack Pointer wird um zwei heruntergesetzt und zeigt jetzt auf $1FD. Falls innerhalb des Unterprogramms weitere Unterprogramme aufgerufen werden, werden die Speicherstellen $1FD und $1FC belegt, der Pointer zeigt danach auf $1FB usw. Am Ende eines Unterprogramms werden schlicht die beiden Bytes "oberhalb" des Stack Pointers wieder gelesen und der Pointer wird um zwei erhöht. Dadurch kommt immer die Adresse als erste wieder heraus, die zuletzt gespeichert wurde.

Der Stack Pointer ist ein Register mit 8 Bit Breite und kann so eigentlich nur Adressen im Bereich von $00 bis $FF erzeugen. Per definitionem fügt der 6502 für jede Stack-Operation den Wert $0100 zur Adresse hinzu. Damit ist der Stack grundsätzlich auf den Speicherbereich $100 bis $1FF festgelegt. Ein Programm wie Applesoft besteht übrigens aus einer Vielzahl ineinandergeschachtelter Unterprogramme.

Auch in diesem Speicherbereich ist Vorsicht geboten, das folgende Programm stürzt ebenfalls mit Sicherheit ab:

```
10 FOR X = 256 TO 511: POKE X,0: NEXT
```

Theoretisch kann man diesen Speicherbereich allerdings wie jeden anderen zur Speicherung von Programmen benutzen. Tatsächlich gibt es einige "kopiergeschützte" Programme, die wichtige Daten und Programmteile hier verstecken und so die Verfolgung ihres Ablaufs stark erschweren.

Durch die Verwendung des 6502 ist auch von vornherein festgelegt, daß die höchste Adresse des Systems $FFFF ist (mehr läßt sich über 16 Adreßleitungen nicht adressieren) und daß zumindest nach dem Einschalten der Stromversorgung der Bereich $FFFX einen ROM-Baustein adressiert, weil der Prozessor nach einem RESET' aus den Speicherstellen $FFFC/D die Startadresse des Programms erwartet.

Eine weitere offensichtliche Konsequenz ist die Organisation des gesamten Speichers mit 8 Bit Breite.

Da der 6502 keine speziellen Steuerleitungen für die Ein- und Ausgabe besitzt, müssen *alle* Datentransporte über die Adreßdekodierung stattfinden. Im Apple ist dafür ein spezieller Adreßbereich vorgesehen, über den Zusatzkarten und die auf der Hauptplatine eingebauten Ein- und Ausgabefunktionen angesprochen werden. Durch das Fehlen eines Signals, mit dem zwischen "Speicheroperation" und "Ein-/Ausgabe" unterschieden werden könnte, ist eine gleichzeitige Belegung dieses Bereichs mit RAM nicht möglich. Diese Art der Organisation von Ein- und Ausgabe wird als *memory mapped* (in den Speicher eingebettet) bezeichnet. In der Konstruktion des Apple erscheint es nur logisch, daß der entsprechende Bereich zwischen RAM und ROM angesiedelt ist. Damit verfügt der Apple

über drei durchgehende Blöcke: RAM ($0000 bis $BFFF), Eingabe/Ausgabe ($C000 bis $CFFF) und ROM ($D000 bis $FFFF).

Die Zeitsteuerung des 6502 im Apple //e

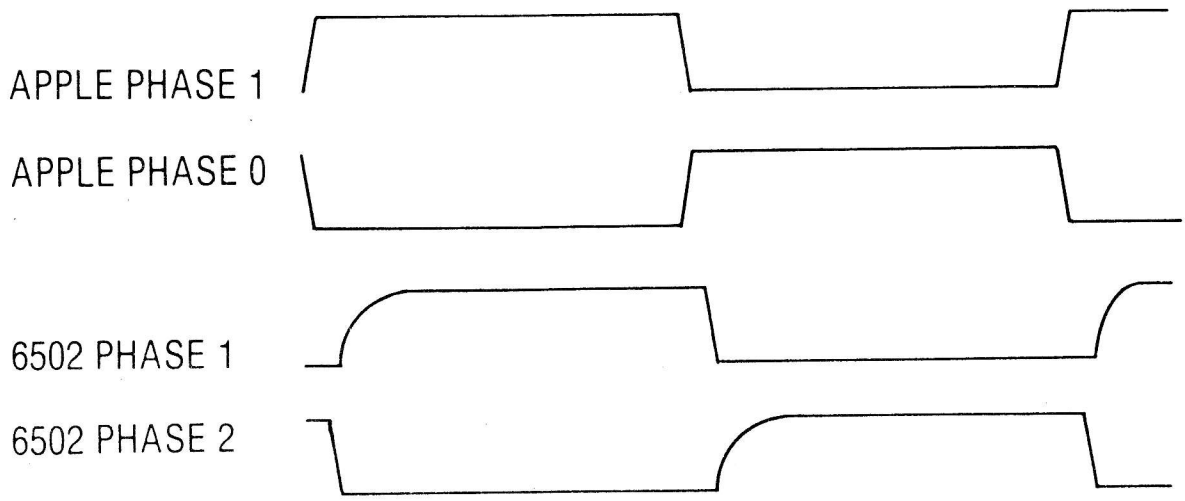
Der 6502 wurde so entworfen, daß er sich wie ein Mikroprozessor des (älteren) Typs MC6800 verhält, für den eine Reihe von Peripheriebausteinen existiert, der darüber hinaus aber einige verbesserte Eigenschaften besitzt. Die benötigten Taktimpulse und das resultierende Zeitverhalten sind dieselben wie beim MC6800: zwei einander abwechselnde Pulse. Für den 6800 müssen diese beiden Signale extern erzeugt und eingespeist werden, der 6502 erzeugt beide intern aus einem einzigen Taktsignal, das ihm über PHASE0 zugeführt wird. Das ist allerdings nur eine von mehreren Verbesserungen.

Die Beziehung zwischen dem zugeführten Zeittakt PHASE0 und den erzeugten Signalen PHASE1 und PHASE2 ist in Bild 4.3 gezeigt. PHASE1 und PHASE2 sind nicht vollständig symmetrisch, sondern "0" ist ein paar Nanosekunden länger als "1". Dadurch paßt der Zustand "1" eines Signals inklusive der verschliffenen Signalflanken genau in den Zustand "0" des anderen Signals und umgekehrt. Die Pegeländerungen von PHASE1 und PHASE2 werden durch PHASE0 ausgelöst: auf eine fallende Flanke von PHASE0 fällt PHASE2, danach steigt PHASE1; auf eine steigende Flanke von PHASE0 fällt PHASE1, danach steigt PHASE2. Andersherum gesagt: Eine fallende Flanke von PHASE0 beendet PHASE2 und beginnt PHASE1, eine steigende Flanke von PHASE0 beendet PHASE1 und beginnt PHASE2.

In einem *langen Zyklus* wird (nur) PHASE2 um rund 140 Nanosekunden gestreckt. Auswirkungen auf die Funktion des Prozessors hat das keine (abgesehen von der in den vorigen Kapiteln beschriebenen Erhöhung der Programmlaufzeit) - im Gegenteil: die Zeitanforderungen an durch den Prozessor angesprochene Bauelemente werden unkritischer. Die zeitlichen Spezifikationen des Prozessors (Anstiegs- und Fallzeiten von Signalen, Ablauflogik etc.) werden durch den langen Zyklus nicht verändert.

Die folgende Diskussion der Zeittakte ist deshalb sowohl für normale als auch für lange Zyklen gültig - in den Diagrammen sind allerdings grundsätzlich normale Zyklen gezeichnet.

Bild 4.3 Die Taktsignale des 6502 und ihre Beziehungen untereinander



Das im 6502 erzeugte Signal PHASE1 ist nicht mit dem Signal PHASE1 des Taktgenerators auf der Hauptplatine identisch, PHASE1 des Taktgenerators ist eine genaue Inversion von PHASE0 und wurde nur wegen seiner Verwandschaft mit dem Prozessorsignal so genannt (schließlich muß ja nicht jeder verstehen, wovon gerade die Rede ist...). Das Taktgenerator-Signal PHASE1 auf der Hauptplatine ist das Signal, das zu allen Steckplätzen geführt, mit der Adreßdekodierung und mit dem RAM verbunden ist. Das Prozessor-Signal PHASE1 wird nur innerhalb des 6502 verwendet und hat keine Verbindung zu anderen Teilen der Hauptplatine. Nur innerhalb dieses

Kapitels ist mit "PHASE1" das Prozessor-Signal gemeint, in allen anderen Fällen ist vom Taktgenerator-Signal die Rede.

Der Apple //e benutzt einen 6502A, der sich von den Ausführungen 6502B und dem 6502 (ohne folgenden Buchstaben) vor allem dadurch unterscheidet, daß er anstelle von 1 MHz Taktfrequenz 2 MHz "verträgt". Dadurch wird eine zusätzliche Sicherung gegen Fehler im Zeitverhalten geschaffen.[1]

Für eine Aufstellung der wichtigsten Zeitanforderungen des 6502 muß zuerst einmal ein Referenzzeitpunkt definiert werden: alle folgenden Zeitangaben beziehen sich auf die steigende bzw. fallende Flanke von PHASE2 (und hierbei auf den Zeitpunkt, an dem dieses Signal einen Pegel von 0.4 Volt hat). Die Zeitangaben sind den Spezifikationen des Herstellers Synertek entnommen, Angaben der Hersteller MOS Technology und Rockwell folgen in Klammern, falls sie unterschiedlich sind.

1. Die ausgegebene Adresse und der Zustand der Leitung R/W' ist spätestens 140 Nanosekunden (MOS: 150 ns) nach der fallenden Flanke von PHASE2 gültig und bleibt es bis mindestens 30 ns nach der nächsten fallenden Flanke von PHASE2, d.h.die Adresse wird im ersten Teil von PHASE1 gültig.
2. Vom 6502 ausgegebene Daten ("write data") sind spätestens 100 ns nach der steigenden Flanke von PHASE2 gültig. Sie bleiben bis mindestens 60 ns (MOS: 30 ns, Rockwell: 30 ns) nach der fallenden Flanke von PHASE2 gültig.
3. Daten für den 6502 ("read data") müssen spätestens 50 ns (Rockwell: 40 ns) vor der fallenden Flanke von PHASE2 gültig sein und es mindestens bis 10 ns nach der fallenden Flanke von PHASE2 bleiben. PHASE2 dient als Signal für den Datentransfer.
4. Die maximale Verzögerung zwischen der fallenden Flanke von PHASE0 und der dadurch ausgelösten fallenden Flanke von PHASE2 beträgt 65 ns, die maximale Verzögerung zwischen der steigenden Flanke von PHASE0 und der dadurch ausgelösten steigenden Flanke von PHASE2 beträgt 75 ns. Diese beiden Werte werden nur von Synertek angegeben und gelten auch nur für eine kapazitive Last von maximal 100 pF an PHASE2.

Diese Zeitangaben stellen "worst case"-Werte dar, d.h. Werte, die unter allen Umständen eingehalten werden. Sie gelten für eine Umgebungstemperatur im Bereich von 0 bis 70 Grad Celsius. Die entsprechenden Signalverläufe für den "worst case" finden Sie in Bild 4.4, wobei hier die Angaben von Synertek zur Grundlage gemacht wurden - Apple, Inc. scheint diesen Hersteller stark zu bevorzugen. Auch das *Technical Reference Manual* des //e enthält Angaben von Synertek[2], und dieser Hersteller gibt als einziger einen Wert für die (wichtigen!) Verzögerungszeiten zwischen PHASE0 und PHASE2 an. Von den angegebenen Verzögerungszeiten könnte man wahrscheinlich noch einmal rund 10 ns abziehen, da PHASE2 im Apple //e nicht weiter verbunden ist und so keine kapazitive Last zu tragen hat.

Es ist übrigens durchaus möglich, daß sich die 6502 aller drei Hersteller völlig gleich verhalten - manchmal liegen engere Spezifikationen eines Herstellers nur an verschärften Testmethoden.

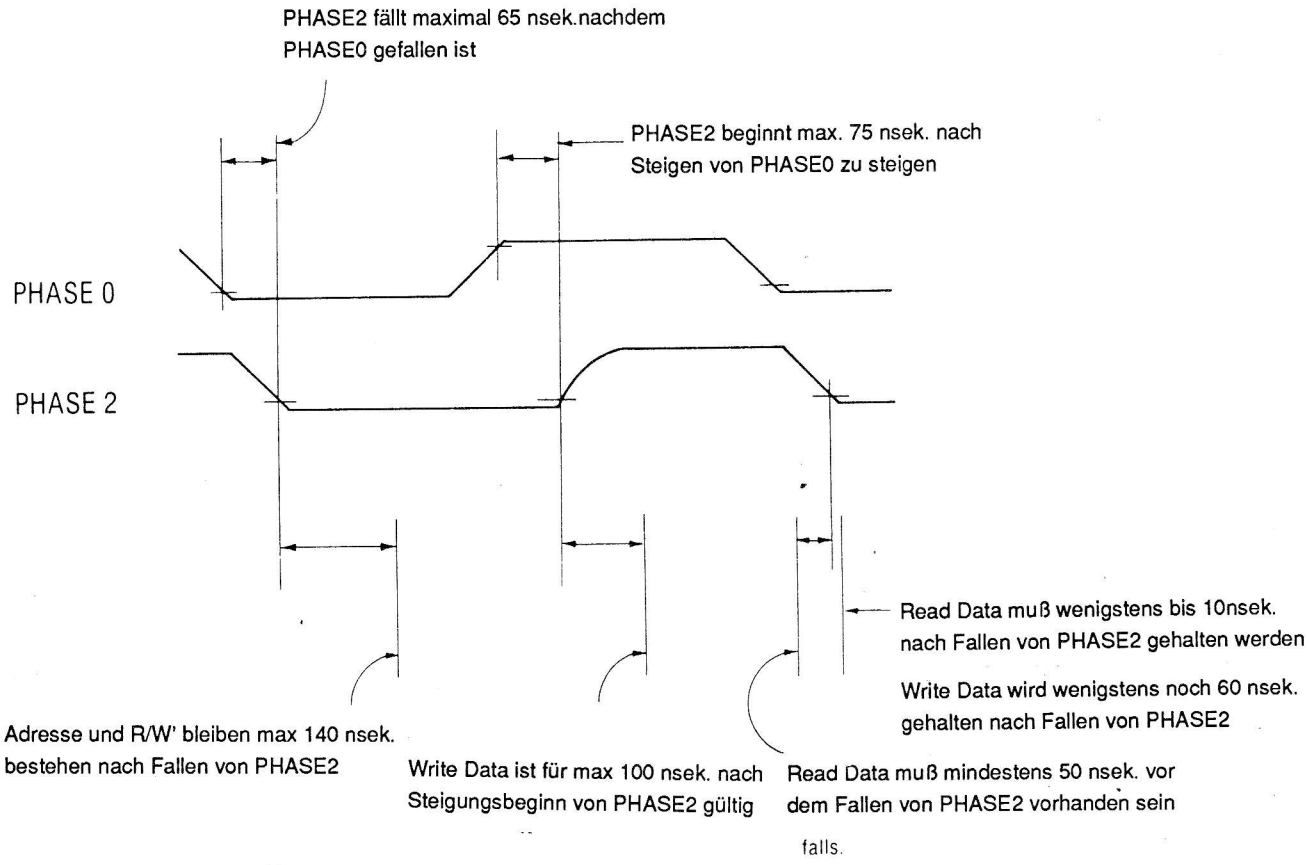
Genug der grauen Theorie. Das Diagramm von Bild 4.5 zeigt die Zeitverhältnisse der Signale, wie sie tatsächlich im Apple //e zu messen sind. Die Messungen wurden mit einem 6502 von Synertek mit den Kennzeichnungen 8307 (7. Kalenderwoche 1983) und S10891, 370-6502 (kein folgender Buchstabe??) ausgeführt und können als typisch für das Zeitverhalten eines 6502 im Apple //e angesehen werden - natürlich sind "typische" Meßergebnisse, die auf einem einzigen Gerät beruhen, mit Vorsicht zu genießen.

---

1 Das *Technical Reference Manual* für den //e behauptet, daß ein (1MHz-)6502B benutzt wird und gibt auch die entsprechenden Spezifikationen und Zeitdiagramme wieder. Auf einigen Ausgaben (selbst noch in der Rev. B) der Hauptplatine steht neben der Fassung des Prozessors ebenfalls noch die Aufschrift "6502B". Peter Baum von Apple, Inc. (USA) hat mir gesagt, daß das von vornherein ein Irrtum war und grundsätzlich nur 6502A's verwendet werden. Er hat mir auch den Grund genannt: Die zusätzliche Sicherheit durch Verwendung des Typs A kostet glatt 25 Cent pro Prozessor. In den neuesten Ausgaben des //e tut ein 65C02 seinen Dienst - der verträgt sogar 4 MHz und hat teilweise ein noch unkritischeres Zeitverhalten. Er wird allerdings nach wie vor mit 1 MHz betrieben, Abfolge und Logik der Signale sind unverändert.

2 Das trifft nur für die älteren Ausgaben des Manuals zu, die noch irrtümlicherweise die Zeitangaben für den 6502B enthalten und sich außerdem auf die fallende Flanke von PHASE0 anstelle der fallenden Flanke von PHASE2 beziehen. Die Ausgabe "Juli 1985" enthält in der englischen Version bereits die Angaben für den 65C02 von NCR für den verbesserten Apple //e.

Bild 4.4 Einige "worst-case" Spezifikationen



Die gemessenen Zeiten sind:

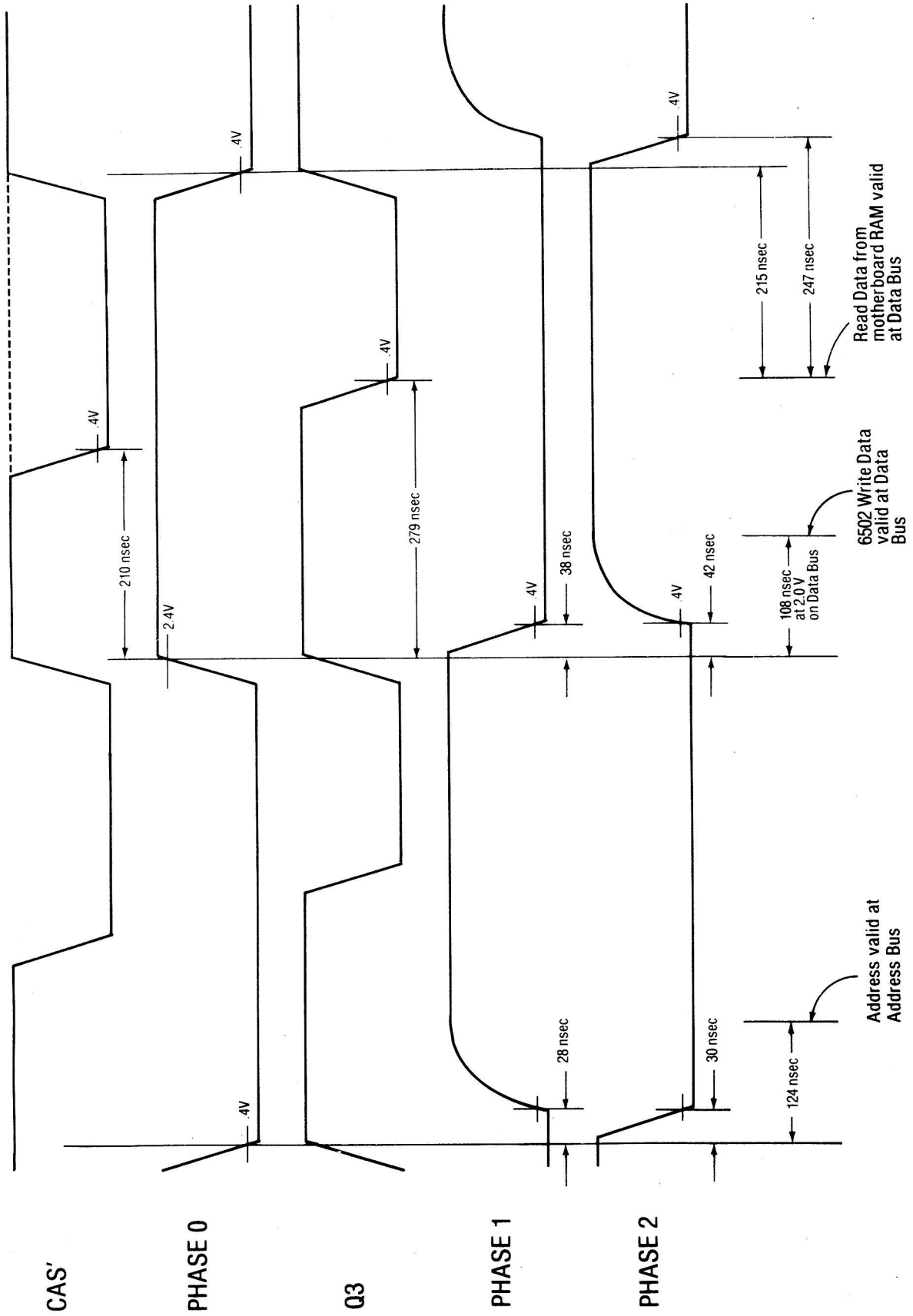
1. Die vom 6502 ausgegebenen Adressen werden 124 ns nach der fallenden Flanke von PHASE0 (gemessen an den Steckplätzen) gültig. Daraus folgt eine Verzögerungszeit von unter 100ns zwischen der fallenden Flanke von PHASE2 und dem Gültigwerden der Adressen. Auch wenn man den "worst case" des 6502A annimmt, werden die Adressen im //e immer gültig sein, bevor Q3 fällt.

2. Vom Prozessor ausgegebene Daten ("write data") werden 108 ns nach der steigenden Flanke von PHASE0 gültig. Unter Berücksichtigung von 42 ns Verzögerung zwischen PHASE0 und PHASE2 ergibt sich die Zeit von 66 ns zwischen der steigenden Flanke von PHASE2 und dem Gültigwerden der Daten (bei Zimmertemperatur). Vom 6502 ausgegebene Daten müssen gültig sein, bevor CAS' für den RAM der Hauptplatine aktiv wird. Der HAL erzeugt CAS' rund 210 ns nach der steigenden Flanke von PHASE0 - hier ist also mehr als genug Freiraum. Dieses Kriterium würde von einer 1MHz-Ausführung des 6502 *nicht* erfüllt - auch hier nur unter der Annahme des "worst case". Übrigens: Der im alten Apple II eingesetzte Prozessor übersteht das im "worst case" auch nur deshalb, weil ihm der Taktgenerator vor CAS' 20 ns mehr Zeit läßt.

3. Daten für den Prozessor ("read data") vom RAM werden 215 ns vor der fallenden Flanke von PHASE0 bzw. 247 ns vor der fallenden Flanke von PHASE2 gültig. Diese Zeiten hängen natürlich stark von den verwendeten RAM-Bausteinen ab, das gemessene Gerät war mit RAM-Chips (200 ns) von OKI bestückt. Für den 6502 alleine könnte man ohne Schwierigkeiten wesentlich langsamere RAMs einsetzen und die Zeit würde immer noch mehr als ausreichend sein. Daten vom AUX-RAM werden erst 80 ns später gültig, aber auch hier dauert es noch eine ganze Weile bis zu der fallenden Flanke von PHASE2, wo sie wirklich benötigt werden.[3]

3 Falls eine Zusatzkarte mit einem Z80-Prozessor verwendet werden soll, sieht das anders aus - durch das etwas merkwürdige Timing des Z80 liegen RAMs mit 200 ns gerade an der zulässigen Grenze, bei RAM-Bausteinen mit 350 ns dürfte es bereits Ärger geben.

Bild 4.5 Gemessene Signalfolgen des 6502A im Apple //e



4. Die Pegeländerungen von PHASE1 und PHASE2 finden rund 30 ns nach einer fallenden Flanke von PHASE0 statt. Die Verzögerungszeiten für Reaktionen auf steigende Flanken von PHASE0 sind etwas größer, weil auch die Anstiegszeit des 74S02 (über den PHASE0 erzeugt wird) etwas größer als die Abfallzeit ist.

Aus den gegebenen und den gemessenen Daten für den 6502 ergeben sich einige Richtlinien für das Design der Hauptplatine und die Zeitanforderungen für Zusatzkarten, von denen Sie einige in der folgenden Aufstellung finden. Weitergehende Details der Kommunikation des 6502 mit den anderen Funktionsgruppen der Hauptplatine finden Sie in den Kapiteln 5 bis 8 im jeweiligen Zusammenhang.

1. Um eine DMA-Operation noch mit dem momentanen Prozessorzyklus zu beginnen, kann eine Zusatzkarte die vom 6502 ausgegebene Adresse vor der fallenden Flanke von Q3 (während PHASE0) lesen und DMA' aktivieren.
2. Die fallende Flanke von PHASE0 zeigt an, daß vom 6502 ausgegebene Daten gültig sind. Eine Zusatzkarte kann PHASE0 damit als Indikator benutzen.
3. Von Zusatzkarten ausgegebene Daten müssen 50 ns vor der fallenden Flanke von PHASE2 gültig sein und es mindestens bis 10 ns nach dieser Flanke bleiben. Die angegebene minimale Verzögerungszeit zwischen PHASE0 und PHASE2 beträgt 5 ns, die maximale Verzögerung von Daten durch den LS245, der den peripheren Datenbus mit dem Datenbus der Hauptplatine verbindet, beträgt 18 ns. Daraus ergibt sich, daß von einer Zusatzkarte ausgegebene Daten mindestens 63 ns vor der fallenden Flanke von PHASE0 gültig sein müssen.

Die Anforderungen für Daten, die von Zusatzkarten in Richtung Prozessor ausgegeben werden, können einfacher als über PHASE0 erfüllt werden, nämlich mit dem Signal DEVICE SELECT'. Dieses Signal überlappt sich nicht mit PHASE2 - die Daten bleiben aber auf dem peripheren Datenbus und auf dem Datenbus der Hauptplatine nach PHASE2 sowieso gültig. Wenn einer der beiden Busse komplett in den hochohmigen Zustand übergeht (d.h. alle Treiber, die Daten auf diesen Bus legen können, inaktiv sind), dann bleibt das letzte Datum auf diesem Bus solange gültig, bis der Buszustand von einem aktiven Sender (= aktiver Treiber) verändert wird. Aus diesem Grund bleiben auf den peripheren Bus ausgegebene Daten auch dann gültig, wenn der periphere Bus kurz vor der fallenden Flanke von PHASE2 hochohmig geschaltet wird. Sie werden vom bidirektionalen Bustreiber verzögert und danach von 6502 korrekt gelesen.[4]

Etwas unkomplizierter gesagt: Die Steuerung ausgegebener Daten von Zusatzkarten über DEVICE SELECT' funktioniert korrekt, obwohl DEVICE SELECT' vor PHASE2 wieder inaktiv wird. Die Forderung, daß Daten bis 10 ns nach PHASE0 gültig sein müssen, ist trotzdem erfüllt.

Die Programmierung des Apple

Es gibt vier Arten, ein Programm für den Apple zu schreiben: direkt in 6502-Maschinensprache, mit einem 6502-Assembler, über eine Compiler-Hochsprache oder über eine Interpreter-Hochsprache. Diese Folge gibt auch gleichzeitig den Schwierigkeitsgrad in absteigender Reihenfolge wieder.

Auch wenn Maschinensprache normalerweise mit Assembler in einen Topf geworfen wird: Ein 6502-Maschinenprogramm besteht aus einer Folge von Bytes, die der Reihe nach ("sequentiell") gespeichert werden. Der 6502 liest dieses Programm durch kontinuierliches Erhöhen des Programmzählers und der damit verbundenen Ausgabe einer Adresse über den Datenbus und führt die einzelnen Befehle aus. Ein Befehl besteht aus einem, zwei oder drei Bytes, die in aufsteigender Reihenfolge gespeichert sein müssen. Das erste Byte jedes Befehls ist der *Opcode* (d.h. der eigentliche Befehl), danach folgt entweder nichts oder ein *Operand* (d.h. ein Zahlenwert oder eine Adresse) mit einem oder zwei Byte Länge. Die Ausführung eines 3-Byte-Befehls erfordert drei Zyklen zum Lesen des Befehls und einen weiteren für seine Ausführung.

Der 6502 verfügt über eine Reihe von internen Registern, deren Inhalt durch Programmbefehle verändert werden kann. Ein komplettes Programm besteht aus einer Manipulation von Daten im Speicher über die Register des Prozessors. Der Vollständigkeit halber folgt hier eine Liste dieser Register:

---

4 Alle tri-State-Busse des Apple //e halten daraufgelegte Daten für eine lange Zeit gültig, wenn sie in den hochohmigen Zustand übergehen (Stichwort: keine aktive Busterminierung). In einigen Fällen bestimmt diese Eigenschaft die Operationsweise (weil man sich darum herumtricksen muß), in anderen Fällen ist sie sogar notwendig, damit eine Baugruppe überhaupt funktioniert. Wir werden, soweit nötig, in den folgenden Kapiteln bei der Besprechung einzelner Baugruppen darauf zurückkommen.

Register	Funktion
Programmzähler	enthält die Adresse des Befehls, der gerade ausgeführt wird bzw. die Adresse des nächsten zu lesenden Befehls
Akkumulator	logische und arithmetische Manipulation von Daten
X-Register	Indexregister
Y-Register	Indexregister
Stack Pointer	zeigt auf die oberste unbesetzte Adresse des Stack-Bereichs
Statusregister	enthält einzelne Flaggenbits, die als Ergebnis von Rechenoperationen, Vergleichen etc. gesetzt bzw. zurückgesetzt werden. Über Tests dieser Flags trifft ein Programm Entscheidungen.

Generell gesagt konzentriert sich ein Programm auf Veränderungen des Akkumulators und einzelner Speicherstellen, während X- und Y-Register zur *Indexierung* von Speicheradressen benutzt werden. Die Werte des Programmzählers, des Stack Pointers und des Statusregisters werden normalerweise vom Prozessor automatisch gesetzt und bedürfen nur in Ausnahmefällen der direkten Kontrolle durch ein Programm. (Der Programmzähler wird natürlich durch jede Art von Sprung innerhalb eines Programms indirekt kontrolliert - gemeint ist, daß sich der Programmierer nicht darum kümmern muß, den Stand des Programmzählers abhängig von der Länge einzelner Befehle explizit zu erhöhen.)

Ein "Programm" mit 3 Befehlen, das den Inhalt der Speicherstelle $1D89 in die Speicherstelle $16 kopiert und danach mit einem BRK endet, sieht im Speicher so aus:

Befehl	Opcode	niederw. Adresse	höherw. Adresse
"Lade Akku"	AD	89	1D
"Speichere A"	85	16	
"BRK"	00		

Der Prozessor erkennt dabei aus dem Opcode, wieviele Bytes für den kompletten Befehl gelesen werden müssen, d.h. ob ein Operand folgt, und wenn ja, ob dieser Operand aus einem oder aus 2 Bytes besteht. Außerdem enthält der Opcode die notwendige Information, ob es sich bei dem Operanden um einen direkten Wert handelt oder um eine Adresse, auf der ein Wert zu finden ist. Dazu kommt die Information, was mit dem Operanden geschehen soll - ganz schön viel für ein einziges Byte.

Maschinensprache-Programme können über das Monitor-Programm des Apple //e eingegeben und ausgeführt werden, die Einzelheiten sind im *Technical Reference Manual* für den //e beschrieben.

Assembler-Sprache ist eine Möglichkeit, Maschinenprogramme mit Unterstützung des Computers zu schreiben. Es gibt einen Haufen Dinge und Arbeiten, die ein Computer besser beherrscht als ein Mensch. Dazu gehören das Auswendiglernen von Opcodes, Addition und Subtraktion von Adressen, das Merken von Unterprogramm-Startadressen und die Prüfung auf syntaktische Fehler. Alle diese Dinge werden von einem Assembler erledigt - der Programmierer formuliert den gewünschten Ablauf mit symbolischen dreibuchstabigen Abkürzungen und arbeitet mit symbolischen Namen anstelle absoluter Adressen. Die Übersetzung des Programmtextes in Maschinensprache und das Einsetzen tatsächlicher Adressen erledigt der Übersetzerteil des Assemblers nach einer Prüfung auf ungültige Befehle. Das oben in Hexadezimalschreibweise gezeigte Programm noch einmal, dieses Mal aber in verständlicherer Form:

```
LABEL       OPCODE  ADRESSE         KOMMENTAR

RESTORE     LDA     $1D89           ;erläutert den
            STA     $16             ;Zweck der Befehle
            BRK
```

Wie bereits gesagt: Dieses Programm enthält Buchstaben und Symbole anstelle von Zahlen - der Prozessor kann damit überhaupt nichts anfangen. Er kann aber mittels eines Übersetzerprogramms aus diesem Text ein Maschinenprogramm machen - und das kann er wesentlich zuverlässiger als ein Mensch.

Für den Apple sind eine ganze Reihe von Assemblern auf dem Markt, deren primäre Funktion in der Eingabe und der Übersetzung symbolischer Prozessoranweisungen liegt - manche enthalten noch zusätzliche Funktionen zur Fehlersuche in Programmen etc. Für die Erstellung eines längeren Programms in Maschinensprache ist ein Assembler eine unbedingte Voraussetzung.

Verglichen mit den meisten anderen Prozessoren hat der 6502 einen sehr einfachen Instruktionssatz. Diese Einfachheit wirkt sich auch auf die in der Assembler-Sprache benutzten Symbole aus: gerade 56 sind es, die ein angehender Programmierer lernen muß, und diese 56 Symbole sind auch noch logisch untereinander verknüpft.

Einfache und wenige Instruktionen haben natürlich Vor- und Nachteile, wie auch die in letzter Zeit in Gang gekommene Diskussion um RISC-Maschinen ("Reduced Instruction Set Computer" = Prozessor mit beschränktem Befehlssatz) zeigt: einem überschaubaren Satz einfacher Befehle, die auch schon beim 6502 recht schnell ausgeführt werden, steht ein Alleskönner gegenüber, der einen riesigen Befehlssatz enthält, aber dafür langsamer ist, als es nach dem momentanen Stand der Technik möglich wäre. Die meisten Prozessoren der letzten Jahre haben z.B. einen Befehl für Multiplikation und Division bereits eingebaut, beim 6502 muß man dafür ein ganzes Unterprogramm schreiben.

Das soll allerdings nicht heißen, daß der 6502 ein "dummer" Prozessor ist - er besitzt eine Reihe von Adressierungsarten, bei deren Simulation sich andere Prozessoren oft ganz schön schwer tun. Gemeint ist hier lediglich, daß die Industrie seit dem Ende der siebziger Jahre nicht geschlafen hat und daß es inzwischen schnellere, bessere und schönere Prozessoren auf dem Markt gibt - allerdings wenige, die so gründlich erforscht worden sind wie der 6502.

Ein weiterer Weg, Maschinencode zu erzeugen, führt über Compiler. Ein Programm kann in einer höheren Sprache wie BASIC, Pascal oder FORTRAN geschrieben und danach von einem Compiler in einzelne Maschinenbefehle übersetzt werden. In einer höheren Programmiersprache muß sich der Programmierer nicht mehr darum kümmern, *wie* z.B. der Prozessor einen Befehl wie PRINT letztendlich ausführt, und kann sich ganz auf den logischen Ablauf des Programms konzentrieren. Natürlich versteht der Prozessor einen Befehl wie PRINT erst recht nicht - der Compiler übersetzt einen solchen Befehl in einen ganzen Block von Maschinenbefehlen. Assembler und Compiler ähneln sich in dieser Beziehung etwas - beide erzeugen aus einem Text Befehle für den Prozessor. Dabei hat der Assemblerprogrammierer allerdings die direkte Kontrolle über alles und jedes, was der Prozessor tut: ein Assemblerbefehl erzeugt exakt einen Prozessorbefehl, nicht mehr und nicht weniger. Ein Befehl in einer Hochsprache kann unter Umständen mehrere hundert Prozessorbefehle zur Folge haben. Darin liegt auch der Grund, warum Compiler Assembler noch nicht komplett verdrängt haben: in Assembler geschriebene Programme sind wesentlich kompakter, und der verfügbare Speicherplatz ist bei den meisten weitverbreiteten Computern immer noch ein Problem - die wenigsten Computerbesitzer sind "Byte-Millionäre".

Gegenüber einem *Interpreter* ist die Erstellung eines Programms mit einem Compiler immer noch recht aufwendig: mit einem Textverarbeitungsprogramm wird der Programmtext eingegeben, danach mit dem Compilerprogramm übersetzt; erst dann kann man das Ergebnis ausprobieren. Falls dabei Fehler auftauchen, wiederholt sich der gesamte Prozeß.

Ein Interpreter analysiert eingegebene Hochsprachen-Befehle während des Programmlaufs und ruft Unterprogramme auf, die Bestandteil des Interpreterprogramms sind und den Befehl ausführen. Dadurch werden "Eingabe", "Übersetzen" und "Ausprobieren" praktisch in einem einzigen Schritt zusammengefaßt. Die beiden BASIC-Varianten Applesoft und INTEGER BASIC arbeiten nach diesem Prinzip.

Natürlich hat das auch seine Nachteile: zum einen ist es erforderlich, daß sich während der Programmlaufzeit der Interpreter immer zusätzlich im Speicher befindet, zum anderen werden auch Befehle, die innerhalb von Programmschleifen stehen, jedesmal wieder neu analysiert. Der Erfolg ist, daß ein Interpreterprogramm den größten Teil der Zeit mit der Analyse anstelle der Ausführung von Befehlen verbringt.

Eine (in "Benchmarks" gern verwendete und für tatsächliche Anwendungen völlig irrelevante) Schleife wie

```
FOR X = 0 TO 255: NEXT
```

wird in einem compilierten BASIC-Programm zwischen 10 und 30 mal schneller, in einem reinen Maschinenprogramm rund 1000 mal schneller als in der interpretierten Form ausgeführt.

Über die Frage, wer was in welcher Art von Programmiersprache schreiben sollte, ist schon soviel gesagt worden, daß wir uns hier auf das Allerwesentlichste beschränken wollen. Abgesehen von den Fähigkeiten und der Erfahrung des Programmierers lassen sich die folgenden Vor- und Nachteile der einzelnen Programmierebenen auflisten:

- Reine Maschinenprogramme sind am schnellsten und am kompaktesten. Die Erstellung ist aufwendig, die Fehlerbeseitigung sehr aufwendig, die Wartung eines Maschinenprogramms (d.h. spätere Veränderungen, Erweiterungen etc.) katastrophal.

- Compilerorientierte Sprachen sind bei größeren Programmprojekten sinnvoll. Die Erstellung des Programms ist etwas aufwendiger als bei Interpretern, die Arbeitsgeschwindigkeit ist ein guter Kompromiß zwischen Maschinen- und Interpreterprogrammen. Die Fehlerbeseitigung und Wartung ist in den meisten Fällen einfach.
- Interpretersprachen sind langsam und benötigen am meisten Speicherplatz (weil z.B. Kommentare auch zusammen mit dem Programmtext gespeichert werden). Die Programmerstellung ist speziell für Anfänger am einfachsten, weil man direkt ausprobieren kann, was man geschrieben hat. Die Fehlerbeseitigung ist sehr einfach, die Wartung längerer Programme aufgrund struktureller Probleme bei Interpretersprachen (es gibt löbliche Ausnahmen!) im Bereich von mittelmäßig schwierig bis unmöglich.

Diese Aufstellung ist mit Sicherheit unvollständig und angreifbar. Probleme wie Strukturierung, Portabilität etc. sind überhaupt nicht berücksichtigt - schließlich geht dieses Buch auch über die Innereien des Apple //e und nicht über einen akademischen Streit.

Ich will auf ein ganz anderes Ergebnis hinaus: Jedes Programm, das auf dem Apple ausgeführt wird, ist ein Maschinenprogramm - egal, ob es direkt in Maschinensprache, in Applesoft oder in UCSD-Pascal geschrieben ist. Etwas anderes als seine Maschinensprache versteht kein Prozessor der Welt. Applesoft, DOS 3.3, ProDOS, UCSD-Pascal, der Monitor, der Miniassembler und was sich sonst noch so im Apple finden läßt, sind alle in Maschinensprache geschrieben, sonst würden diese Programme nicht funktionieren.

Bleibt uns noch eine wichtige Fußnote zum Thema "Betriebssysteme und Programmiersprachen": Über die Logik von DMA' kann man den 6502 zeitweilig oder komplett abschalten und über eine Zusatzkarte durch einen anderen Prozessor ersetzen bzw. ihm einen "Zweitprozessor" anhängen. Es ist sogar direktes "Co-processing" mit einem zweiten Prozessor möglich, der sich im speziellen Steckplatz befindet und seinen eigenen RAM-Bereich hat. Um welche Prozessormarke es sich dabei handelt, ist ziemlich egal: für den Apple existieren Zusatzkarten für den 6800, 6809, 8086/88, 68008 ... und für den Z80, mit dem das Betriebssystem CP/M benutzt werden kann. Einige Leute behaupten, daß der Apple zusammen mit einer der Z80-Karten der am meisten verkaufte CP/M-Computer überhaupt ist. Damit wird der DMA-Prozeß so wichtig, daß sich diese Fußnote zu einem ganzen Abschnitt mausert.

DMA im Apple //e

Unter dem Begriff *DMA* ("Direct Memory Access" = direkter Speicherzugriff) versteht man im allgemeinen eine besonders schnelle Form des Zugriffs auf den Speicher, die hauptsächlich für Ein- und Ausgabe genutzt wird. Festplatten und andere Geräte, die in einer solchen Geschwindigkeit Daten liefern oder benötigen, daß ein programmgesteuerter Prozessor dabei nicht mehr mitkommt, arbeiten über DMA. In diesem Zustand wird der Prozessor vom Daten- und Adreßbus getrennt, meist übernimmt ein spezieller DMA-Baustein die Kontrolle, der eigentlich nicht sehr viel kann außer riesige Datenmengen transportieren - und das mit einer Geschwindigkeit, die beim zehn- bis zwanzigfachen des Prozessors liegt. Ein typischer DMA-Baustein transportiert pro Taktzyklus des Prozessors ein Byte; der Prozessor braucht für einen Durchlauf einer entsprechenden Programmschleife (mit Index setzen, Test auf Ende, Laden eines Bytes und Speichern an einer anderen Adresse) mindestens um die 15 bis 20 Zyklen.

Im Apple ist der Begriff DMA noch etwas weiter ausgelegt: hier wird der Prozessor nicht nur vom RAM getrennt, sondern von der gesamten restlichen Elektronik des Computers. Als Folge davon kann ein via DMA arbeitender Zusatzprozessor nicht nur Datenmengen verschieben etc., sondern sich sämtlicher Funktionen des Computers bedienen (Tastatureingabe, Softswitches, etc.).

Das Ansprechen des RAMs durch den Videoscanner ist ein Beispiel für *simultanen* DMA und ist nur deshalb möglich, weil der RAM des Apple //e die doppelte Zugriffsgeschwindigkeit des Prozessors verträgt. Wie bereits des öfteren gesagt, finden innerhalb eines Prozessorzyklus jeweils ein Zugriff des Videoscanners und (eventuell) ein Zugriff des Prozessors statt. Dieser simultane DMA ist für die CPU völlig "transparent", d.h. sie merkt nichts davon, Programmabläufe werden dadurch nicht beeinflußt.

Eine zweite Form von DMA wird als *cycle stealing* ("Zyklusklauen") bezeichnet. Bei dieser Form wird der Takt für die CPU einen Prozessorzyklus lang gestoppt - während die CPU "in der Luft hängt", findet der Speicherzugriff eines anderen Bausteins statt. Diese Art von DMA setzt natürlich die Ausführungsgeschwindigkeit eines Programms herunter.

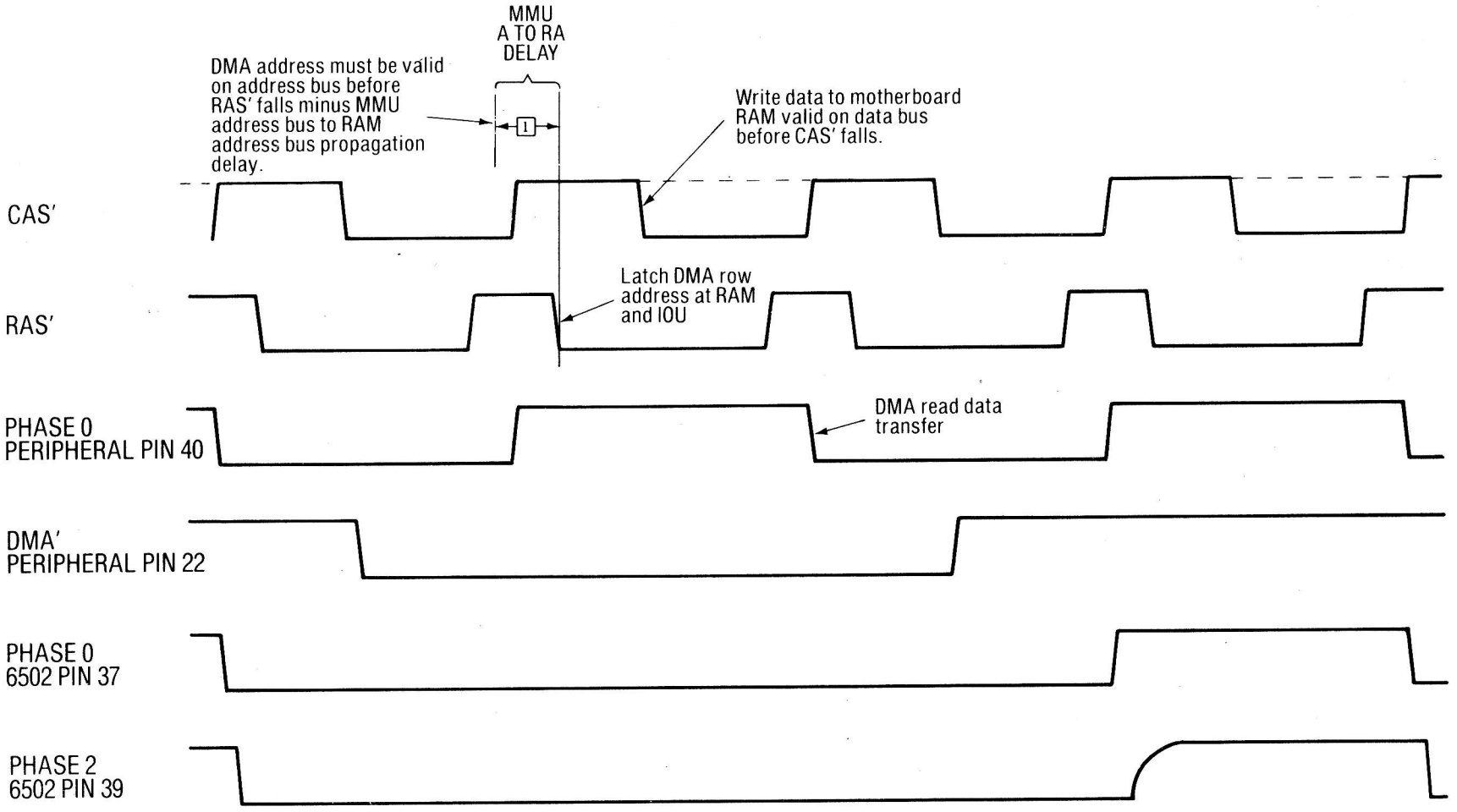


Bild 4.6 Ein "gestohlener" DMA-Zyklus

[1] Measured at 82 nsec in author's Apple IIe. Maximum MMU address bus to RAM address bus propagation delay not published by Apple.

Diese "cycle steal"-Methode wird im Apple //e zusammen mit dem simultanen DMA des Videoscanners benutzt. Die Leitung DMA' ist mit Kontakt 22 aller Steckplätze verbunden, jede Zusatzkarte kann sie auf "0" ziehen und damit den 6502 abschalten. Die zeitliche Voraussetzung: DMA' muß während PHASE0 auf "0" gebracht und (nach einer Anzahl von Zyklen) wieder während PHASE0 zurück auf "1" gesetzt werden. Solange sich DMA' im aktiven Zustand befindet, sind die Adreßbus-Treiber der CPU (+ R/W') in den hochohmigen Zustand geschaltet, und der Takteingang PHASE0 des 6502 (Pin 37) bleibt konstant im Zustand "1". PHASE0 wird nur direkt am Prozessor geschaltet, alle anderen Verteilungspunkte dieses Signals bleiben von DMA' unberührt. Der 6502 wartet dann mit PHASE1 auf "1" und PHASE2 auf "0" auf den nächsten Taktimpuls, die Datenein-/-ausgänge sind auf "Lesen" geschaltet und beeinflussen den Datenbus nicht. Damit ist der Prozessor komplett abgehängt, die Zusatzkarte kann die Kontrolle über den Computer übernehmen. Sie sollte natürlich beim Lesen und Schreiben in den RAM denselben Regeln folgen wie der 6502, d.h. Datenübergaben dürfen nur am Ende von PHASE0 stattfinden. Der Videoscanner, der während PHASE1 aktiv ist, wird durch den Pegel von DMA' in keiner Weise berührt, dasselbe gilt für die MMU und die restliche Adreßdekodierung.

Bild 4.6 zeigt die Zeitabläufe in einem "gestohlenen" Zyklus des 6502 im Apple //e. Die Art und Weise, wie eine DMA-Baugruppe den Speicher und andere Bausteine anspricht, muß in derselben Weise erfolgen wie beim 6502: der Adreßbus muß während PHASE1 vor der steigenden Flanke von RAS' eine gültige Adresse enthalten, damit die MMU genügend Zeit für die Kontrolle des gemultiplexten RAM-Adreßbusses und der Leitung CXXX hat, bevor RAS' fällt (und damit aktiv wird).[5]

Daten, die in den RAM geschrieben werden sollen, müssen vor der fallenden Flanke von CAS' gültig sein, vom RAM ausgegebene Daten sind während der fallenden Flanke von PHASE0 gültig.

Der 6502 ist nicht *statisch* aufgebaut und funktioniert intern ähnlich einem dynamischen RAM: Falls man ihm zu viele Zyklen auf einmal stiehlt (d.h. den Takt zu lange anhält), gehen die internen Registerinhalte verloren mit dem Ergebnis eines Programmabsturzes. Es ist dabei nicht klar, wie lange man ihm Taktimpulse vorenthalten darf: das Datenblatt von MOS Technology gibt z.B. eine maximale Breite von 520 Nanosekunden für PHASE0 an. Das stimmt mit Sicherheit nicht - schließlich hat jeder lange Zyklus im Apple bereits eine Impulsbreite von 625 ns, Probleme damit sind bis jetzt nicht bekannt geworden.

Das *4 & 8 Bit Microprocessor Handbook* (McGraw Hill 1981) von Adam Osborne und Gerry Kane gibt an, daß man PHASE2 für alle Prozessoren der Serie MCS65XX überhaupt nicht verlängern darf - die müssen wohl auch das Datenblatt von MOS gelesen haben. Synertek gibt eine maximale Zykluszeit von 40 Mikrosekunden für PHASE0 an, was bedeuten würde, daß man im Apple //e 40 DMA-Zyklen hintereinander ausführen könnte; Rockwell legt sich dagegen auf 10 Mikrosekunden fest.

Erfreulicherweise weiß Steve Wozniak, der Urvater des Apple, über dieses Thema mehr als die meisten Leute. Er hat mir erzählt, daß er in seinem allerersten Entwurf des Apple II eine andere Methode simultaner DMA für die Videoerzeugung benutzt hat als in der endgültigen Version. In der Zeit, als er die ersten Entwürfe machte, wurden RAM-Bausteine für 2 MHz gerade erst verfügbar, deshalb arbeitete der erste Entwurf mit RAM-Bausteinen und einem 6502 mit 1 MHz, wobei jeweils 40 von 65 Zyklen für die Videoausgabe gestohlen wurden. Das Ergebnis war in mehrfacher Hinsicht traurig: zum einen arbeitete der 6502 dadurch mit einer effektiven Taktfrequenz von 385 kHz (25/65 * 1 MHz), zum anderen mußte Woz feststellen, daß diese Methode nur mit unbenutzten 6502-Prozessoren funktionierte - je länger der Prozessor in Betrieb war, desto vergeßlicher wurde er, und es wurde nötig, immer einen Haufen frischer 6502-Prozessoren auf Lager zu haben. Wobei die ausgetauschten 6502s nicht kaputt waren - er vermutete lediglich, daß die Fähigkeit des Prozessors, interne Daten für 40 Mikrosekunden ohne Taktimpulse zu behalten, nach einer gewissen Betriebszeit aufgrund des Nachlassens innerer Bausteinkapazitäten abnahm.

Zum Glück war es nie nötig, durch lange Meßreihen zu bestimmen, wie lange ein "gebrauchter" 6502 seine Daten zuverlässig halten konnte: kurz danach waren RAM-Bausteine auf dem Markt verfügbar, die mit 2 MHz arbeiten konnten, und er änderte das Design des Apple entsprechend, bevor der Computer auf den allgemeinen Markt kam.

---

5 Für die MMU-Laufzeiten (hier speziell: Adresse -> gemultiplexter RAM-Adreßbus und CXXX) existieren keine offiziellen Angaben oder Datenblätter. Ich glaube, daß eine DMA-Baugruppe in jedem Fall funktioniert, wenn die ausgegebenen Adressen vor der steigenden Flanke von RAS' (während PHASE1) auf den Adreßbus stabil sind - garantieren kann ich das allerdings nicht. Kapitel 5 beschäftigt sich eingehender mit den diversen Signalen der MMU.

Nach einigen Experimenten einigte man sich auf 5 Mikrosekunden als sicheren Wert - auch die von Microsoft konstruierte Z80-Karte für den Apple II hält sich daran, und damit hat es bis jetzt keine Probleme gegeben.[6] Eine wirklich gründliche Untersuchung scheiterte damals schon daran, daß sie nicht nur neue und "gebrauchte", sondern auch "sehr gebrauchte" Prozessoren hätte einschließen müssen - der 6502 war damals gerade ein gutes Jahr alt, "sehr gebrauchte" Exemplare gab es aus diesem Grund noch gar nicht.

Es ist bereits gesagt worden, daß eine Zusatzkarte, die über DMA' arbeitet, sich nicht nur auf RAM-Zugriffe beschränken muß, sie kann sämtliche Möglichkeiten des Apple //e nutzen, auf Softswitches, Tastatur, Lautsprecher und sogar auf andere Zusatzkarten zugreifen, wie es z.B. die (normale) Z80-Karte mit einer Druckerschnittstelle tut. Für den Apple sind nicht nur Zusatzprozessoren wie Z80, 6809, 8086, 68008 oder 65C02 mit 3.5 MHz ("SpeedDemon") auf dem Markt, die via DMA' arbeiten und damit den Computer praktisch auf den Kopf stellen, sondern auch (tatsächlich!) Karten, die DMA' für den eigentlichen Zweck, nämlich für die Kommunikation mit schnellen Massenspeichern nutzen ("MEGACore", Corvus etc.).

Der Pegel von DMA' hat keinen Einfluß auf den Speicherzugriff des Videoscanners, weil dessen Adreßerzeugung vom (über DMA' erreichbaren) Adreßbus der Hauptplatine isoliert ist. Anders gesagt: Auch für eine DMA-Baugruppe ist die Arbeit des Videoscanners vollkommen transparent.

Die Entwickler des 6502 haben eigentlich den *READY-Eingang* des Prozessors für DMA-Zugriffe vorgesehen. Gedacht war, den Prozessor über READY in einem Lesezugriff zu stoppen und den Adreßbus über externe tri-State-Treiber während einer DMA-Operation vom Prozessor zu trennen. Die entsprechende Leitung READY im Apple //e hat keine Kontrolle über die tri-State-Treiber des Adreßbusses, sie ist nur mit den Steckplätzen verbunden, DMA kann also nur über die Leitung DMA' erfolgen. Man könnte eine Zusatzkarte so konstruieren, daß sie READY zusammen mit DMA' auf "0" bringt - aber durch das Unterbrechen der Taktimpulse wird der Prozessor nach kurzer Zeit vergeßlich. Aber auch daran wurde im Apple //e gedacht: Die Hauptplatine enthält zwei Lötverbindungen, die als X4 und X5 bezeichnet werden; wenn X4 verbunden und X5 durchtrennt wird, dann wird PHASE0 durch DMA' nicht mehr vom Prozessor getrennt, der Prozessor kann über DMA' zusammen mit READY in einen theoretisch endlosen Wartezustand gebracht werden, in dem er seine internen Daten nicht vergißt. Der Grund, warum diese beiden Verbindungen ab Werk andersherum gelegt sind, liegt natürlich wieder in der Kompatibilität mit bereits existierenden Zusatzkarten für den Apple II.

Der Apple hat ein *Prioritätssystem* für DMA-Zugriffe: wenn mehrere Zusatzkarten gleichzeitig eine DMA-Operation beginnen wollen, dann hat die Karte im Steckplatz mit der niedrigsten Nummer den Vorrang. Dieses System besteht aus einer Leitungskette von Ein- und Ausgängen, die sich von Steckplatz zu Steckplatz zieht.

Eine Karte, die kein DMA durchführen kann, hat zwischen Ein- und Ausgang eine schlichte Verbindung und leitet so das Signal weiter. Karten, die DMA durchführen können, müssen zuerst ihren Eingang prüfen: wenn dieser Eingang den Pegel "0" hat, bedeutet das, daß eine Karte höherer Priorität gerade DMA durchführt. Eine Karte, die DMA durchführt, setzt ihren Ausgang auf "0" und teilt so allen Karten niedrigerer Priorität mit, daß eine DMA-Operation im Gange ist. Kontakt 27 jedes Steckplatzes ist der Eingang, Kontakt 24 der Ausgang, Steckplatz 1 hat die höchste Vorrangstufe und deshalb keinen Eingang, Steckplatz 7 hat die niedrigste Stufe und deshalb keinen Ausgang, das komplette Schema finden Sie in Bild 7.6.

Besonders hochentwickelt ist die Systematik nicht: ein unbesetzter Steckplatz unterbricht die Kette, die Vorrangstufen sind durch die physikalische Lage der Karten unverrückbar festgelegt. Allerdings dürfte ein Apple mit mehreren konkurrierenden DMA-Karten auch ein recht schöner "Verhau" werden, der programmtechnisch nur noch unter größten Schwierigkeiten beherrschbar ist.

Diese Kette kann außer für DMA auch für andere Funktionen benutzt werden - Apple, Inc. hat das mit den Firmware-Karten für INTEGER BASIC und Applesoft bereits vorgemacht. Diese Karten ersetzen jeweils den ROM der Hauptplatine durch ihren eigenen und stellen so den jeweils fehlenden Basic-Dialekt zur Verfügung (als Alternative zum Laden des Dialekts von Diskette in die oberen 16 kByte RAM). Mehrere dieser Karten können in aufeinanderfolgende Steckplätze gesetzt werden; über die Prioritätskette wird sichergestellt, daß jeweils nur eine Karte aktiv ist.

---

6 Die Version der Z80-Karte für den //e (ebenfalls von Microsoft) benutzt überhaupt kein DMA, sondern besteht aus einem kompletten Mikrocomputer mit 64 k eigenem RAM, der vom 6502 wie der AUX-RAM angesprochen wird. Konsequenterweise ist die Karte für den speziellen Steckplatz ausgelegt und verkehrt mit dem 6502 über den (karteneigenen) AUX-RAM.

Die unvollkommene Organisation der Prioritätskette hat auch ihre Vorteile: Wenn mehrere Gruppen von Zusatzkarten für verschiedene Zwecke benutzt werden sollen (Beispiel: Applesoft-Sprachkarte zusammen mit Festplatten-Schnittstelle), dann kann man durch Leerlassen eines dazwischenliegenden Steckplatzes einfach zwei getrennte Ketten fabrizieren - eine, die gleichzeitiges Aktivieren von mehreren Firmware-Karten verhindert, und eine, die den tatsächlichen DMA-Betrieb regelt.

6502-Interrupts im Apple //e

Es gibt vier Möglichkeiten, den Prozessor bei der Abarbeitung eines Programms zu unterbrechen: RESET', NMI', IRQ' und BRK. Jede dieser Möglichkeiten hat ihre eigenen Eigenschaften (und meist ihren eigenen Zweck). Die ersten drei werden hardwaremäßig ausgelöst, die entsprechenden Leitungen sind zu den Steckplätzen durchverbunden. RESET' ist zusätzlich noch mit der RESET-Taste und dem Pin 15 der IOU verbunden. Der Befehl BRK dagegen ist ein Software-Interrupt. Die Reaktion des 6502 auf alle vier Unterbrechungsarten wird durch den Inhalt des ROMs $E0..FF bestimmt.

RESET'

RESET' stellt eine Ausnahme im Interrupt-Konzept dar: Allgemein wird durch einen Interrupt ein laufendes Programm zeitweilig unterbrochen, ein spezieller Programmteil ausgeführt, und danach geht es im eigentlichen Programm weiter. Bei RESET' entfällt der dritte Schritt: Der 6502 speichert bei einem RESET'-Impuls keine Registerinhalte, der momentane Stand des Programmzählers geht dabei in die ewigen Jagdgründe, und es gibt keine Möglichkeit, das Programm an der Unterbrechungsstelle fortzusetzen. RESET' ist deshalb auch nicht für eine Programmunterbrechung, sondern für einen definierten Programmstart gedacht. Der 6502 reagiert darauf mit den folgenden Schritten:

1. Lesen von drei Bytes vom Stack, die nicht interpretiert werden;
2. Lesen der Startadresse der RESET-Routine von den Speicherstellen $FFFC/$FFFD, das niederwertige Byte kommt zuerst;
3. Sperren von IRQ' durch Setzen des entsprechenden Bits im Statusregister, die restlichen Flags bleiben unverändert;
4. Sprung zum Start der RESET-Routine.

Der Grund für die drei Stack-Zugriffe liegt darin, daß RESET' prozessorintern wie eine leicht veränderte Form von NMI' und IRQ' behandelt wird, bei denen drei Bytes (PC und Status) auf den Stack geschrieben werden. Bei RESET' ist R/W' im Zustand "Lesen" anstelle von "Schreiben", der Stack Pointer wird aber wie bei einer Speicheraktion auf den Stack erniedrigt.

Der RESET'-Ablauf hinterläßt einen "Fingerabdruck" auf dem Adreßbus, der von der MMU (die nicht mit der RESET'-Leitung verbunden ist!) erkannt wird. Die MMU interpretiert jede Folge, die aus drei Zugriffen auf den Stack-Bereich besteht, als Systemreset und setzt daraufhin alle internen Softswitches zurück (ob da wohl ein Pin am Gehäuse zu wenig war?). Unerfreulicherweise macht die MMU keinen Unterschied zwischen auf- und absteigenden Zugriffen auf den Stack - ein Programm auf der Speicherseite 1 (also im Stack-Bereich), das auf $FFFC zugreift, setzt damit alle Softswitches zurück.[7] Man darf gespannt sein, wer damit als nächstes einen "Kopierschutz" realisiert.

Einer der Softswitches, die von der MMU bei einem RESET' zurückgesetzt werden, ist der Schalter für die oberen 16 kByte RAM/ROM. Aus diesem Grund wird der RESET'-Ablauf im Apple //e immer durch den Inhalt des ROMs $E0-FF bestimmt. Der Inhalt der Speicherstellen $FFFC/D ist $FA62, auf $FA62 beginnt das RESET-Programm des Monitors. Es führt unter anderem folgende Initialisierungen aus:

- Setzen der Tastatur als Eingabequelle;
- Setzen des Bildschirms als Ausgabegerät;
- Setzen des Videomodus TEXT mit 40 Zeichen und 24 Zeilen.

---

7 Die Tabellen am Ende dieses Kapitels enthalten eine detaillierte Darstellung der Befehlsabläufe des 6502.

Die RESET-Routine des Apple enthält einige Programmteile, die sie ziemlich einzigartig machen. Der wohl wichtigste Unterschied zu anderen Mikrocomputern liegt darin, daß sie über einen RAM-Vektor enden kann. Ein Programm kann also das letztendliche Ergebnis von RESET' kontrollieren, indem es diesen Vektor setzt.

Der erste Durchlauf der RESET-Routine wird durch das Einschalten der Stromversorgung ausgelöst. Da der RAM-Vektor in diesem Fall undefiniert ist, wird er nicht benutzt, die RESET-Routine setzt den RAM-Vektor auf den Warmstart von Applesoft und startet entweder Applesoft "kalt" oder setzt sich über das Startprogramm des Diskettencontrollers fort (wenn vorhanden). In diesem Fall wird der RAM-Vektor durch das von der Diskette geladene Betriebssystem noch einmal neu gesetzt.

Den automatischen Startprozeß (d.h. das Absuchen aller "normalen" Steckplätze nach einem Diskettencontroller und den darauffolgenden Sprung dorthin) gibt es im Modell Apple II noch nicht. Der entsprechende Programmteil wurde erst nach Fertigstellung der Diskettenlaufwerke für den Apple II+ hinzugefügt und das Monitorprogramm in "Autostart-Monitor" umbenannt. Das RESET'-Programm des Apple //e ist eine leicht erweiterte Version des Autostart-Monitors des II+ mit zusätzlichen Programmteilen für die Tasten "offener Apfel" und "geschlossener Apfel" sowie für den 80-Zeichen-Betrieb.

Der RAM-Vektor steht auf den Speicherstellen $03F2/3, zusätzlich existiert ein "Power-up Byte" ($3F4), das zusammen mit dem Vektor gesetzt werden muß. Wenn dieses Byte das Ergebnis der Verknüpfung "$3F3 EXOR #$A5" enthält, nimmt die RESET-Routine an, daß der Computer bereits eingeschaltet war und springt zu der Adresse, die auf den Speicherstellen $03F2/3 angegeben ist, ansonsten wird ein "Kaltstart" durchgeführt, an dessen Ende die Speicherstellen $03F2-4 gesetzt werden. Die Wahrscheinlichkeit, daß das "Power-up Byte" nach dem Einschalten der Stromversorgung zufällig der Bedingung "$3F3 EXOR #$A5" entspricht, ist reichlich gering.

Nachfolgende RESET'-Impulse bewirken einen Sprung über den im RAM gespeicherten Vektor anstelle eines "Kaltstarts". Selbstverständlich kann man die Speicherstelle $3F4 von einem Programm aus demolieren, woraufhin der Apple bei jedem Druck auf RESET' von neuem die Diskettenlaufwerke anwirft (wieder so ein "Kopierschutz").

Wenn zusammen mit RESET eine der Apfeltasten gedrückt wird, führt das RESET'-Programm spezielle Funktionen aus: "geschlossener Apfel" bewirkt einen Aufruf von Selbsttestroutinen. Nach diesem Selbsttest ist der gesamte RAM gelöscht, der nächste Druck auf RESET bewirkt einen "Kaltstart". Ein Druck auf "offener Apfel" füllt den Speicherbereich von $0000 bis $BFFF mit zwei einander abwechselnden Werten (und überschreibt so auf $3F4), bevor der Test dieses Bytes stattfindet. RESET zusammen mit "offener Apfel" setzt somit die Programmkontrolle des RESET'-Vektors außer Gefecht und startet den Computer zu jedem Zeitpunkt "kalt" - gegenüber dem beim Apple II+ benötigten Griff zum Netzschalter ein erfreulicher Fortschritt.

Es ist offensichtlich, daß für eine schlichte Veränderung der Speicherstelle $3F4 nicht sämtliche 49152 Byte der unteren 48 k verändert werden müssen. Genauso offensichtlich ist es, daß dadurch kommerzielle Programme vor unberechtigtem(?) Zugriff durch Apple-Benutzer geschützt werden sollen. Noch offensichtlicher ist, daß durch diesen Schutz bestenfalls "Hobbyhacker" davon abgehalten werden, in einem gekauften Programm herumzustochern - Apple, Inc. hat dieses "Feature" eingebaut, um den Softwareherstellern, die bis jetzt ihre Programme über den RESET'-Vektor "geschützt" haben, nicht den Boden unter den Füßen wegzuziehen.

NMI' und IRQ'

Beide Signale sind nur mit den "normalen" Steckplätzen verbunden und werden auf der Hauptplatine des Apple nicht genutzt. Über Ein- und Ausgabe ausgelöste Interrupts von Zusatzkarten arbeiten normalerweise über IRQ', weil dieses Signal durch Programmkontrolle gesperrt werden kann. Der nicht maskierbare Interrupt (NMI') ist für Fälle gedacht, in denen eine Unterbrechung so wichtig ist, daß sie sofort ausgeführt werden muß. Bei vielen Systemen (wenn auch bis jetzt noch nicht beim Apple) wird NMI' verwendet, um bei einem drohenden Stromausfall Diskettenoperationen zu beenden und Daten in batteriegepufferte Speicherbereiche zu retten. Die Stromversorgung enthält zu diesem Zweck einen Detektor, der auf Ausbleiben der Netzspannung mit einem NMI' reagiert - bis zu einem Zusammenbruch der Betriebsspannung des Systems bleiben dann aufgrund der Speicherkondensatoren des Netzteils noch einige Dutzend Millisekunden Zeit.

Die vom 6502 ausgeführten Schritte sind für NMI' und IRQ' gleich, abgesehen davon, daß IRQ' nur dann eine Reaktion auslöst, wenn das entsprechende Bit im Statusregister zurückgesetzt ist:

1. Der Prozessor führt zuerst den momentan laufenden Befehl zu Ende.
2. Der Stand des Programmzählers wird auf den Stack gebracht, das höherwertige Byte zuerst.
3. Das Statusregister wird auf den Stack gebracht.

4. Der Interrupt-Vektor (NMI': $FFFA/B, IRQ': $FFFE/F) wird gelesen, das höherwertige Byte kommt dabei zuerst.
5. Durch Setzen des IRQ'-Sperrbits im Statusregister wird IRQ' gesperrt.
6. Sprung zur Interrupt-Behandlungsroutine.

Hardwaremäßig besteht zwischen NMI' und IRQ' ein kleiner, aber sehr wichtiger Unterschied: der NMI'-Eingang des Prozessors reagiert auf negative (fallende) Flanken, der IRQ'-Eingang auf negative Pegel. Eine typische Signalfolge für einen NMI' sieht so aus:

1. Der Pegel von NMI' fällt von "1" auf "0".
2. Die entsprechende Behandlungsroutine wird vom Prozessor ausgeführt, der IRQ'-Eingang ist dabei gesperrt.
3. Die NMI'-Leitung wird als Reaktion auf die Behandlungsroutine (oder einfach durch ein Zeitglied) wieder auf "1" gesetzt.
4. Rücksprung zum normalen Programm, der Sperrmechanismus für den IRQ'-Eingang wird wieder auf den Stand gesetzt, den er vor dem Interrupt hatte.

Ein Gerät, das einen NMI' auslöst, kann verhindern, daß die Behandlungsroutine des Prozessors durch den NMI' eines anderen Gerätes unterbrochen wird, indem es die NMI'-Leitung einfach konstant auf "0" hält, bis der Prozessor signalisiert, daß die Behandlung beendet ist. Solange innerhalb der Behandlungsroutine nicht durch einen expliziten Befehl IRQ' entsperrt wird, ist die Routine durch Unterbrechungen via IRQ' ebenfalls geschützt.

Die typische Signalfolge für einen IRQ':

1. Der Pegel von IRQ' fällt von "1" auf "0".
2. Wenn der IRQ'-Eingang entsperrt ist, reagiert der Prozessor mit der Ausführung der IRQ'-Behandlungsroutine. Für die Dauer der Routine wird IRQ' automatisch gesperrt.
3. Der Prozessor teilt dem Baustein oder der Baugruppe mit, daß die Behandlung erfolgt ist ("Acknowledge"), als Reaktion darauf wird der Pegel der Leitung IRQ' wieder auf "1" gesetzt.
4. Nach Ende der Behandlungsroutine wird das normale Programm fortgesetzt, IRQ' ist dabei wieder zugelassen.

Der Eingang IRQ' wird während der Interrupt-Behandlung in derselben Weise wie bei einem NMI' gesperrt. Das verhindert, daß über den "0"-Pegel der Leitung sofort ein weiterer IRQ' ausgelöst wird. Während der IRQ'-Behandlung ist eine Unterbrechung durch NMI' jederzeit möglich. Der IRQ' kann auf vielfältige Weise eingesetzt werden, allen Abläufen ("Protokollen") gemeinsam ist, daß der Prozessor ein "Acknowledge" (Bestätigung) ausgibt, der mitteilt, daß die Unterbrechungsanforderung zur Kenntnis genommen wurde. Die IRQ'-auslösende Baugruppe reagiert damit, daß sie den Pegel von IRQ' wieder auf "1" setzt. In den diversen Anwendungen für Uhrenkarten, die Maus und andere Zusatzkarten im Apple //e findet der Acknowledge im Normalfall über das Ansprechen einer Kontrolladresse der jeweiligen Karte statt.

Für die meisten Anwendungen ist die Interrupt-Behandlung des 6502 eine recht einfache Angelegenheit: via NMI' oder IRQ' werden Programmzähler und Statusregister auf den Stack gebracht, danach wird über den jeweiligen Vektor zur Behandlungsroutine gesprungen. Das Statusregister wird vor dem Setzen des IRQ'-Sperrbits ("I-Flag") gespeichert. Die Behandlungsroutine endet normalerweise mit dem Prozessorbefehl RTI (ReTurn from Interrupt), durch den sowohl das Statusregister als auch der Programmzähler wieder vom Stack geholt werden. Zusammen mit dem Inhalt des Statusregisters wird auch der alte Zustand des IRQ'-Sperrbits wieder hergestellt.

Die restlichen Register des 6502 werden durch Interrupts nicht automatisch gespeichert. Das ist wieder einmal eine Philosophiefrage: Die in der Behandlungsroutine benutzten und veränderten Register müssen natürlich vorher gesichert werden - ein grundsätzliches Sichern *aller* Register kann dagegen unnötig Zeit kosten, wenn dabei auch Register gespeichert werden, die die Interrupt-Behandlungsroutine überhaupt nicht benutzt. (Das ist allerdings bei den paar Registern, die der 6502 zur Verfügung stellt, recht unwahrscheinlich.)

Interrupts innerhalb von Interrupts sind möglich - durch das "Last In First Out"-Prinzip des Stacks funktioniert das in derselben Weise wie Unterprogramme innerhalb von Unterprogrammen. Das geht natürlich nur solange gut, wie genügend Platz auf dem Stack ist: falls die Behandlungsroutine ständig mehr Zeit benötigt als zwischen einzelnen Interrupts bleibt, "kracht" es über kurz oder lang. (Das ist auch ein Thema, über das schon ganze Bibliotheken geschrieben worden sind.)

Ein Interrupt kann von jeder Zusatzkarte in einem der "normalen" Steckplätze ausgelöst werden. Falls mehrere Interruptquellen möglich sind, muß die Behandlungsroutine mit einer Abfrage beginnen, in der festgestellt wird, aus welcher Quelle der Interrupt stammt. Das bis jetzt einzige Betriebssystem, das mehrere Interrupts systematisch unterstützt, ist ProDOS, für alle anderen bekannten Betriebssysteme des Apple (DOS 3.3, UCSD und CP/M bis Version 2.23) sind mehr oder weniger große Verrenkungen nötig.

Die Steckplätze sind untereinander durch eine *Prioritätskette* für Interrupts verbunden, die exakt wie die Prioritätskette für DMA funktioniert. Wie auch dort hat Steckplatz 1 die höchste, Steckplatz 7 die niedrigste Priorität. Karten in einem Steckplatz mit niedrigerer Nummer können damit theoretisch "darüberliegende" Karten kontrollieren bzw. von einem Interrupt abhalten. Diese Möglichkeiten sind bis jetzt von den Designern zusätzlicher Karten nur in sehr beschränktem Rahmen ausgeschöpft worden. Außerdem hilft es dem Prozessor nicht weiter - er muß nach wie vor sämtliche in Frage kommenden Karten abfragen.

Es gibt eine Möglichkeit, die Quelle eines Interrupts ohne weitere Verzögerung zu bestimmen, nämlich dadurch, daß jede Karte ihren eigenen Interrupt-Vektor auf den Datenbus legt: jede Zusatzkarte kann den ROM der Hauptplatine via INHIBIT' abschalten und durch eigene Bausteine ersetzen. Für Interrupts reicht eine einfache Logik auf der Karte, die Adressen im Bereich $FFFX erkennt und dem 6502 nach einem Interrupt den karteneigenen Vektor unterschiebt. Die via NMI' arbeitenden "Programmklaukarten" arbeiten zum Beispiel nach diesem Schema. Hier kommt dann auch die Prioritätskette wieder ins Spiel: über sie kann verhindert werden, daß mehrere Karten gleichzeitig versuchen, dem Prozessor ihren eigenen Vektor unterzuschieben.

Die Routinen des $E0-FF-ROMs zur Behandlung von Interrupts sind sehr einfach ausgefallen: ein NMI' bewirkt über den $FFFA/B-Vektor einen direkten Sprung zur Speicherstelle $3FB, wo dann ein Sprung zur Startadresse der Behandlungsroutine stehen muß. Der IRQ' wird etwas anders behandelt, nämlich über eine kurze Routine, die mit $FA40 beginnt und zuerst einmal feststellt, ob der Interrupt durch IRQ' oder einen BRK-Befehl ausgelöst wurde. Diese Unterscheidung wird dadurch getroffen, daß das Statusregister wieder vom Stack geholt und das B-Flag getestet wird. Bei einem durch IRQ' ausgelösten Interrupt wird danach zu der Adresse gesprungen, die in den Speicherstellen $03FE/F verzeichnet ist, ansonsten geht es innerhalb des Monitorprogramms weiter.

Während dieses Unterscheidungsprozesses wird der Inhalt des Akkus auf der Speicherstelle $45 zwischengespeichert, danach wird das Statusregister vom Stack in den Akku geladen. Der Prozessor beginnt dadurch jede IRQ'-Behandlungsroutine mit einem falschen Akkuinhalt, der erst von der Speicherstelle $45 nachgeladen werden muß.

Programme, die $45 zur Speicherung eigener Werte benutzen, können folglicherweise nicht gefahrlos mit Zusatzkarten arbeiten, die IRQ' erzeugen (solange diese Karten nicht eine eigene Startadresse für $FFFE/F unterschieben). Demnach sollte man annehmen, daß die Speicherstelle $45 von Betriebssystemen und anderen wichtigen Programmen umgangen wird - leider ist dem nicht so: sowohl DOS 3.2 als auch DOS 3.3 benutzen diese Speicherstelle für kritische Daten, es ist mit einem IRQ' möglich, sie bei der Arbeit zu unterbrechen und danach z.B. Binärdateien mit Werten zu füllen, die aus einem ganz anderen Speicherbereich stammen als ursprünglich vorgesehen (nein - das ist kein neuer Programmiertrick, sondern eine Katastrophe).[8]

Auch das neue Betriebssystem ProDOS benutzt diese Speicherstelle für kritische Werte - hier war allerdings das Problem bereits erkannt und es wurde ein geradezu wahnwitziger Aufwand getrieben, um es zu beseitigen (was allerdings für den II+ und den //e in der Originalversion des Monitorprogramms nur teilweise gelungen ist).[9]

Das Problem mit der Speicherstelle $45 und den sich daraus ergebenden Konsequenzen kann umgangen werden, indem Zusatzkarten nur so konstruiert werden, daß sie dem Prozessor eine eigene IRQ'-Startadresse auf $FFFE/F via INHIBIT' unterschieben.

Für die neueren Ausführungen des Apple //e (ab Anfang 1986) ist es ohnehin nicht mehr vorhanden, sie enthalten eine weitere Modifikation des Autostart-Monitors, bei dem die Speicherstelle $45 nicht mehr für Interrupts benutzt wird. In Kapitel 6 ist diesem neuen ROM ein ganzer Abschnitt mit dem Titel "Der verbesserte Apple //e" gewidmet, Spezifisches zum Thema "Interrupts" im Zusammenhang mit dem neuen Monitor-ROM finden Sie im übernächsten Abschnitt dieses Kapitels.

---

8 Der Artikel "Go Ahead and Interrupt your Apple" von Dan Fischer und Morgan Caffrey (erschienen in März und April 1982 in SOFTALK) beschäftigt sich detailliert mit diesem Problem.

9 Eine komplette Analyse dieses Ablaufs und der notwendigen Vorsichtsmaßnahmen bei dem als "interruptfähig" verkauften ProDOS finden Sie in "Die ProDOS-Analyse" (Hüthig Verlag 1985).

Der Befehl BRK

Dieser Befehl erzeugt einen programmgesteuerten IRQ' und kann nicht über das IRQ'-Sperrbit im Statusregister abgeschaltet werden. Der Prozessor erzeugt diesen "Interrupt", wenn er den Opcode $00 erhält. Sein Zweck ist eigentlich auch Programmierprofis unklar - wozu sollte sich ein Programm selbst unterbrechen wollen?

Die einzige sinnvolle Möglichkeit für die Verwendung von BRK liegt im Austesten von Programmen. Die Entwickler des Apple haben dem auch insoweit Rechnung getragen, daß die BRK-Routine des Monitorprogramms auf die Kommandoebene des Monitors zurückkehrt und den Inhalt sämtlicher Prozessorregister ausgibt. Wenn man in der Entwicklungsphase eines Programms an einem bestimmten Punkt herausfinden möchte, welche Inhalte die einzelnen Register haben, gibt es im Prinzip zwei Möglichkeiten:

a) Man ersetzt drei Byte an dieser Stelle durch einen Sprung und schreibt eine Routine, die die Registerinhalte ausgibt.

b) Man setzt ein einziges Byte 00 und überläßt den Rest dem Monitorprogramm.

BRK bewirkt prozessorintern exakt dasselbe wie ein IRQ': der Stand des Programmzählers und das Statusregister werden auf den Stack gebracht, danach folgt ein Sprung zu der Adresse, die aus den Speicherstellen $FFFE/F gelesen wird. Es ist auch möglich, ein echtes Prüfprogramm zu schreiben, das ohne sichtbare Unterbrechung an verschiedenen Stellen via BRK die Registerinhalte liest, sie ausgibt und danach im zu prüfenden Programm weitermacht.

Ein weiterer Zweck von BRK ist die "weiche Landung" eines außer Rand und Band geratenen Programms und liegt somit auch nur in der Programmentwicklung. Wild gewordene Programme machen meistens an irgendeiner Stelle im Speicher weiter, die überhaupt nicht vorgesehen war, und stellen dabei allen möglichen Unsinn an. Die Wahrscheinlichkeit, daß der Prozessor dabei über kurz oder lang auf den Opcode 00 trifft und damit seinen Amoklauf beendet, ist erfreulich groß - der überwiegende Teil der RAM-Speicherzellen nimmt nach dem Einschalten der Stromversorgung konstruktionsbedingt den Wert 0 an.

Die Voraussetzung für einen erfolgreichen Stop des Prozessors ist allerdings, daß das Programm nicht in den oberen 16 k auf "RAM" geschaltet hat - ansonsten beginnen die Probleme erst richtig, weil der Prozessor einen undefinierten IRQ'-Vektor liest.

Die einzelnen Schritte des Prozessors nach einem BRK-Befehl sind:

1. Der Programmzähler wird um 2 erhöht und auf den Stack gebracht, das höherwertige Byte zuerst.
2. Im Statusregister wird das BRK-Flag gesetzt, danach landet es ebenfalls auf dem Stack.
3. Die Adresse der Behandlungsroutine für BRK/IRQ' wird von den Speicherstellen $FFFE/F gelesen, das niederwertige Byte zuerst.
4. IRQ' wird gesperrt und es folgt ein Sprung zu der von $FFFE/F gelesenen Adresse.

Es gibt hier zwei Unterschiede zu einem IRQ': zum einen wird der Stand des Programmzählers + 2 gespeichert. Das unterstützt die Vermutung, daß BRK hauptsächlich zu Testzwecken gedacht ist, der Programmzähler steht dadurch auf dem nächsten Befehl, wenn durch BRK ein Sprung ersetzt wurde (diese Erklärung ist zwar auch etwas wacklig, es gibt aber zumindest keinen anderen ersichtlichen Grund für diese Erhöhung).

Zum zweiten wird im Statusregister das BRK-Flag gesetzt. Dieses Flag unterscheidet sich von allen anderen Bits dieses Registers: es gibt weder Setz- noch Rücksetzbefehle dafür noch eine auf diesem Flag basierende bedingte Verzweigung. Auf BRK kann nur geprüft werden, nachdem das Statusregister auf den Stack gebracht wurde, das BRK-Flag wird nur durch einen ausgeführten BRK gesetzt. Letztendlich läßt sich somit nicht beurteilen, ob dieses Flag überhaupt zum Statusregister gehört: Das BRK-Flag scheint eine Charakteristik des Prozessors bei der Speicherung des Statusregisters auf dem Stack zu sein. Anders gesagt: Dieses Flag existiert nicht als Bit 4 des Statusregisters im Prozessor, sondern nur im RAM. Es wird bei einer Speicherung des Statusregisters folgendermaßen gesetzt:

1. Der Befehl PHP setzt immer Bit 4 im RAM.
2. Speichern des Statusregisters nach einem NMI' setzt immer Bit 4 im RAM.
3. Speichern des Statusregisters nach einem BRK setzt immer Bit 4 im RAM.
4. Speichern des Statusregisters nach einem IRQ' löscht Bit 4 im RAM.

Achtung: Diese Wertung ist einzig und allein das Ergebnis meiner Versuche und der dabei gemachten Beobachtung, daß es auch über Tricks nicht möglich ist, "Bit 4" des Statusregisters auf "0" zu setzen - selbst dann, wenn man in das Statusregister über den Stack den Wert 00 einlädt, findet sich nach einem direkt darauffolgenden PHP-Befehl der Wert $30 (d.h. Bit 4 gesetzt) auf dem Stack wieder.

Wie dem auch sei: Die einzige Möglichkeit, zwischen BRK und IRQ' zu unterscheiden, liegt im Herunterholen des Statusregisters vom Stack in den Akku und sieht so (oder zumindest funktionell gleich) aus:

```
PLA                 ;Status vom Stack
PHA                 ;wieder zurück
AND  #%0001 0000    ;Test von Bit 4
BNE  BRK.HANDLER
BEQ  IRQ.HANDLER
```

Im Monitorprogramm des Apple //e sind die ersten Schritte zur Behandlung von BRK und IRQ' gleich. Falls danach ein BRK erkannt wird, trennen sich die Wege. Die Register des Prozessors werden in den Speicherstellen $3A, $3B und $45-49 abgelegt und es geht mit der in den Speicherstellen $03F0/1 gespeicherten Adresse weiter.

Dieser BRK-Vektor wird beim Einschalten der Stromversorgung des Apple mit der Startadresse der Routine im Monitorprogramm gesetzt, das die Werte der Register des 6502 ausgibt. Außerdem disassembliert dieses Programm die bei PC+2 stehende Instruktion und setzt danach den Benutzer auf die Kommandoebene des Monitorprogramms. Diese BREAK-Routine ist zum "Entwanzen" von Maschinenprogrammen sehr nützlich, die zwischengespeicherten Registerinhalte können über den Befehl "**G**"O des Monitorprogramms wieder eingeladen werden, das unterbrochene Programm wird dadurch bei PC+2 fortgesetzt. Selbstverständlich ist es auch hier möglich, für ein Programm mit weiterentwickelten Fehlersuchfunktionen die Adresse der BRK-Behandlung im Vektor $03F0/1 zu verändern.

Die Behandlung von IRQ' und BRK im verbesserten Apple //e

Bis in letzter Zeit hat Apple, Inc. dem Thema "Interrupts" nicht allzuviel Aufmerksamkeit geschenkt. Das scheint sich zu ändern: Es sind große Anstrengungen unternommen worden, ProDOS trotz des $45/IRQ'-Problems voll interruptfähig zu programmieren, und der Apple //c wurde mit mehreren internen Interruptquellen und einem dementsprechend komplexen Interrupt-Handling ausgestattet. Dieser Handler ist seit kurzem auch als Teil einer "Upgrade"-Packung für den //e verfügbar. Das Programm verändert den Wert der Speicherstelle $45 nicht mehr und speichert den Akku zusammen mit den restlichen Registern auf dem Stack. Die folgenden Seiten enthalten eine Diskussion der entsprechenden Routinen.[10]

Für diese Besprechung ist es notwendig, daß Sie einen gewissen Überblick über die Möglichkeiten der Speicherkonfiguration via Softswitches haben - Details dazu finden Sie in Kapitel 5 im Abschnitt "Die Verwaltung des Speichers".

In der neuen Version[11] des $E0-FF-ROMs für den Apple //e zeigt der IRQ'/BRK-Vektor auf $FFFE/F nicht mehr auf $FA40, sondern auf $C3FA. Wenn wir davon ausgehen, daß die oberen 16 k auf "ROM" geschaltet sind, dann muß für einen kontrollierbaren Ablauf entweder INTCXROM gesetzt, SLOTC3ROM zurückgesetzt oder sich eine Zusatzkarte mit entsprechendem Code in Steckplatz 3 befinden. Daraus folgt: Interrupts funktionieren nur, wenn die Firmware der 80-Zeichen-Karte (d.h. der entsprechende ROM) aktiv ist. Die Belegung von Steckplatz 3 mit einer anderen Karte, die eigenen ROM enthält, ist nicht mehr möglich.

Für den Fall, daß die oberen 16k auf "RAM" (und nicht auf ROM) geschaltet sind, wird die entsprechende Startadresse aus dem RAM gelesen. In ProDOS, das die oberen 16k RAM benutzt, findet sich dafür eine kurze Routine, die den RAM ab- und den ROM anschaltet, bevor die Interrupt-Routine des $C0-FF-ROMs angesprungen wird. In den ProDOS-Versionen ab 1.1.1 sind die obersten rund 10 Byte des AUX-RAMs ebenfalls mit entsprechenden Vektoren belegt, dafür ist auf der RAM-Disk dann ein Block weniger verfügbar.

Die neuen Routinen zur Interrupt-Behandlung sind gegenüber der alten $FA40-Routine sehr stark erweitert: per Programm wird dem 6502 soweit auf die Sprünge geholfen, daß er bei einem IRQ' nicht nur die automatische Sicherung von Status und Programmzähler auf den Stack vornimmt, sondern auch noch alle anderen Register

---

10 Eine generelle Beschreibung aller Neuerungen finden Sie in Kapitel 6.

11 Jim Sather bezieht sich auf eine Vorversion vom 9.7.84. Mir liegt mittlerweile das "offizielle" ROM-Listing vom Juni 1985 vor, bei dem noch ein Fehler korrigiert worden ist (falsches Setzen der Schreib/Lesekontrolle der oberen 16 k RAM). Dadurch haben sich einige Speicheradressen und die Belegung des Maschinenstatusbytes verschoben. Die folgende Beschreibung entspricht der endgültigen(?) Version - (Anm. d. Übers.).

ebenfalls über den Stack sichert. Für einen RTI nach einem IRQ' gilt das dann umgekehrt, hier werden sämtliche Register wieder per Programm vom Stack geholt, die der 6502 nicht freiwillig wieder einsetzt. Damit nicht genug: Das Programm bringt die momentane Speicherkonfiguration und einen Wert für momentan den EXROM-Bereich benutzende Zusatzkarten ebenfalls auf den Stack und stellt nach einem RTI die Zustände wieder her, die vor dem IRQ' geherrscht haben. Anstelle eines IRQ'-Handlers sind damit zwei voneinander getrennte Routinen getreten: ein Programm, das bei einem IRQ' den momentanen Zustand sichert, und ein zweites, das nach Ende der *Interrupt-Behandlung* den vorherigen Zustand wiederherstellt - ein "RTI-Handler" sozusagen.

Die ersten Schritte sind auch hier für IRQ' und BRK dieselben: Akku, X- und Y-Register werden auf den Stack gebracht, es wird eine festgelegte Speicherkonfiguration gesetzt (INTCXROM an, Bildschirmseite 1, AUX-RAM aus, die oberen 16k auf "ROM"). Während dieses Setzens wird ein *Speicherstatusbyte* ermittelt, das in den Bits D7-D0 die folgenden Daten enthält: ALTZP, 80STORE*PAGE2, RAMRD, RAMWRT, HRAMRD, HRAMRD*BANK1, BANK2, INTCXROM. Dieses Byte wird für einen BRK in $44, für einen IRQ' auf dem Stack gespeichert. Der Status von HRAMWRT (d.h. ob die oberen 16k RAM schreibgeschützt sind oder nicht) wird dabei zwar nicht gespeichert, wird aber auch nicht benötigt - nach einem RTI werden die oberen 16k RAM (falls vorher aktiv) so trickreich wieder angeschaltet, daß der "write status" dabei unverändert bleibt. (Die Vorversion des IRQ-Handlers hatte hier noch einen Fehler und setzte den RAM immer auf "Schreiben erlaubt", wenn er auch für "Lesen" aktiviert war.) Innerhalb dieser ersten Schritte wird auch der gesamte AUX-RAM abgeschaltet, inklusive von ALTZP ($0-1FF und $D000-FFFF). Damit ergibt sich das Problem, daß eine Interrupt-Behandlung keinen Zugriff zum Stack des unterbrochenen Programms mehr hat, wenn das Programm den AUX-Stack benutzt. Um dieses Problem zu lösen, hat Apple das folgende Stack-Protokoll definiert:

1. Der Stack Pointer für den AUX-Stack wird nach der ersten Schaltung von ALTZP immer auf $FF gesetzt.
2. Der Stack Pointer für den RAM der Hauptplatine wird auf $100 von AUX gespeichert, nachdem ALTZP geschaltet wurde.
3. Der Stack Pointer für den AUX-Stack wird auf $101 des Hauptplatinen-RAMs gespeichert, nachdem ALTZP zurückgesetzt wurde.

Diese Vereinbarung ist deshalb zusammen mit der Verarbeitung von Interrupts kritisch, weil eine Interrupt-Behandlungsroutine ALTZP schalten muß, wenn sie auf Stack-Daten des unterbrochenen Programms zugreifen will und ALTZP während des IRQ' gesetzt war. Hier verbirgt sich ein Fallstrick: IRQ' muß während einer Umschaltung von ALTZP und der Speicherung der Stack Pointer auf $100 bzw. $101 gesperrt sein!

Nach diesen ersten Schritten geht die Bearbeitung von IRQ' und BRK unterschiedliche Wege. Bei einem IRQ' wird die Adresse des "RTI-Handlers" als Returnadresse für die IRQ'-Behandlungsroutine gesetzt und ein Markierung für eine eventuell aktive Zusatzkarte auf den Stack gespeichert, bevor es dann (endlich) nach Abschalten des internen C300-ROM via $03FE/F zur IRQ'-Behandlungsroutine geht. Folgende Werte sind in der gezeigten Reihenfolge auf dem Stack gespeichert worden:

Stack des unterbrochenen Programms (Hauptplatine oder AUX)	**Stackbereich auf der Hauptplatine**
PCH	Speicherstatus
PCL	Aktive Karte ($Cn)
Statusregister	$C3
Akkumulator	$F4
Akkumulator	
Akkumulator	
X-Register	
Y-Register	

Die "aktive Karte" wird von der Speicherstelle $7F8 gelesen. Diese Speicherstelle ist Teil einer weiteren Vereinbarung: Karten, die den EXROM-Bereich ($C800-CFFF) benutzen, müssen das höherwertige Byte ihrer Startadresse ($Cn, n = Steckplatznummer) in $7F8 speichern, damit die "RTI-Routine" diese Karte nach einer IRQ'-Behandlung wieder aktivieren kann. Wenn die 80-Zeichen-Karte selber aktiv ist, enthält $7F8 den Wert $C3.

Die Bytes $C3 und $F4 stehen an der Spitze des Stacks auf der Hauptplatine. Solange die IRQ'-Behandlung nicht den Stack durcheinanderbringt, führt ein RTI-Befehl am Ende der Routine damit zu $C3F4, dem "RTI-Handler". Diese Routine setzt den I/O STROBE' für eine eventuell vorher aktive und während der IRQ'-Behandlung abgeschaltete Zusatzkarte, stellt die Speicherkonfiguration wieder her und lädt die Registerinhalte vom Stack zurück. Danach wird ein "STA $C007" (setzt INTCXROM) oder "STA $C006" (schaltet INTCXROM ab), gefolgt

von einem RTI, auf den Stack gebracht(!). Dieser Code wird danach via RTS (von $C4C0) ausgeführt und setzt INTCX wieder auf den Stand vor dem IRQ', bevor das unterbrochene Programm fortgesetzt wird.

Wird ein *BRK* anstelle eines IRQ' erkannt, wird außer den 6502-Registern nichts auf den Stack gebracht. Der Speicherstatus wird auf $44 abgeladen, danach werden die Register wieder vom Stack geholt und in die Speicherstellen $3A, $3B und $45 bis $49 geschrieben. Damit wird derselbe Zustand hergestellt wie durch die alte BRK-Behandlung, es folgt ein Sprung über den Vektor $03F0/1, wo normalerweise die Adresse der unverändert übernommenen BRK-Behandlung des alten Monitor-ROMs steht ($FA59). Dieser Programmteil gibt die Registerinhalte des 6502 (von der Seite 0) auf den Bildschirm aus und setzt den Benutzer auf die Kommandoebene des Monitors.

Der große Unterschied zwischen alter und neuer Behandlung von BRK liegt darin, daß in der neuen Version INTCXROM zurückgesetzt und der AUX-RAM abgeschaltet wird, bevor via $03F0/1 zur eigentlichen Behandlungsroutine gesprungen wird. Im alten Monitor-ROM hätte ein BRK innerhalb des AUX-RAMs einen kompletten Absturz zur Folge gehabt. Die neue Routine hat nur einen Schönheitsfehler: wenn ALTZP während des BRK gesetzt ist, gibt die $FA59-Routine falsche Registerinhalte aus. Das kommt daher, weil die Registerwerte auf dem AUX-Stack gespeichert werden, danach wird ALTZP zurückgesetzt, und der Transport der Registerinhalte vom Stack auf die Seite 0 erfolgt vom Stack des Hauptplatinen-RAMs.

Zusammenfassend läßt sich sagen, daß es Apple, Inc. durch diese neue Unterstützung der Interrupts und durch die Festlegung eines verbindlichen Protokolls den Softwarehäusern erheblich leichter gemacht hat, interruptfähige Programme zu schreiben. Vielleicht wirkt das Ergebnis ein bißchen sehr massiv - man sollte aber nicht vergessen, daß bei der Unzahl von Autoren für Apple-Programme eine lückenlose Standardisierung dringend notwendig ist. Was dabei für mich etwas fragwürdig bleibt, ist die Entscheidung, die entsprechenden Routinen in den Bereich $C3XX zu legen und dadurch eine dauernde Aktivität der 80-Zeichen-Firmware vorauszusetzen sowie die Schaltung von I/O STROBE' als unumgänglicher Teil der Interrupt-Routine. Ein Handler im Bereich von $FXXX könnte zuerst den Status von INTCXROM speichern und danach INTCXROM vor einem Sprung in die Gegend von $C4XX aktivieren - dadurch würde eine während des IRQ' aktive Zusatzkarte automatisch nach Ende der Behandlungsroutine durch schlichtes Abschalten von INTCXROM wieder angeschaltet.

Ein zweiter Kritikpunkt betrifft das Zurücksetzen von ALTZP: dadurch wird es zwar einfach gemacht, einen IRQ'-Handler in den oberen 16k des RAMs der Hauptplatine (anstelle des ROMs) zu installieren, es ergibt sich aber eine Reihe von negativen Konsequenzen. Es wird ein Protokoll für die Sicherung der Stack Pointer benötigt, kritische Werte werden unter Umständen auf zwei verschiedene Stacks verteilt und der BRK-Handler funktioniert unzuverlässig. Um auch hier meinen Senf dazuzugeben: Solange ALTZP durch IRQ' und BRK zurückgesetzt wird, sollte zumindest die BRK-Routine so umgeschrieben werden, daß sie auch dann korrekt arbeitet, wenn der BRK mit gesetztem ALTZP stattgefunden hat.

Ungeachtet meiner kleineren Kritikpunkte stellt die neue IRQ'/BRK-Verarbeitung einen Fortschritt dar, sie funktioniert, zerstört die Speicherstelle $45 nicht mehr und ist ein wesentlicher Schritt in Richtung Standardisierung und vollständiger Kompatibilität mit dem Apple //c (und das ist der eigentliche Hintergrund dieser Neuerung).

Die Rangfolge der Interrupts

Es ist theoretisch möglich, daß Interrupts mehrerer Klassen zum selben Zeitpunkt auftreten. Damit dabei noch ein vorhersagbares Ergebnis herauskommt, wird eine klare Rangfolge benötigt, die so aussieht:

RESET'	(höchste Priorität)
NMI'	
BREA	
IRQ'	(niedrigste Priorität)

RESET' "überstimmt" alle anderen Unterbrechungen. Während RESET' aktiv ist, wird auch eine fallende Flanke von NMI' vom Prozessor vollständig ignoriert. Sobald der Prozessor mit der Abarbeitung einer RESET-Routine beginnt und sich damit in einem normalen Programm befindet, kann er natürlich durch NMI' oder BRK unterbrochen werden. Aus diesem Grund dürfte es für eine Zusatzkarte das Beste sein, NMI'-erzeugende Schaltkreise (soweit vorhanden) auf ein RESET'-Signal hin so lange abzuschalten, bis der Prozessor explizit signalisiert, daß die RESET'-Routine abgearbeitet ist. Die RESET' zugrundeliegende Idee ist ein komplettes Rücksetzen des gesamten Systems - nicht nur des Prozessors allein.

Alle Interrupts setzen das IRQ'-Sperrbit im Statusregister des Prozessors (d.h. RESET' tut es auch).

Wenn NMI' während eines durch IRQ' oder BRK ausgelösten Interrupt-Zyklus aktiv wird (also zu dem Zeitpunkt, wo der Prozessor bereits reagiert, aber noch nicht den entsprechenden Vektor gelesen hat - darauffolgende Befehle sind wieder "normales Programm"), dann wird der NMI'-Vektor anstelle des BRK/IRQ'-Vektors gelesen. Ein IRQ'-Zyklus wird auf diese Weise unterbrochen, der NMI' wird zuerst behandelt und (solange wir davon ausgehen, daß IRQ' mangels "Acknowledge" während der NMI'-Behandlung "0" bleibt) der IRQ' wird ausgeführt, nachdem am Ende der NMI'-Behandlung IRQ' wieder entsperrt wird. Mit BRK ist es etwas komplizierter: Wenn BRK innerhalb des Interrupt-Zyklus von NMI' unterbrochen wird, liest der Prozessor ebenfalls den NMI'-Vektor - und vergißt BRK danach vollständig. Noch schlimmer: Nach dem RTI von der NMI'-Routine versucht der Prozessor, das durch BRK unterbrochene Programm mit dem übernächsten Befehl (PC + 2) fortzusetzen! Das ist allerdings keine Apple-Besonderheit - dieser "Bug" ist den Entwicklern des 6502 beim Testen durch die Lappen gegangen, er ist erst im 65C02 korrigiert worden (der zusammen mit der Firmware der verbesserten Version geliefert wird). Wenn ein NMI' beim 65C02 während eines Interrupt-Zyklus für BRK auftritt, wird der Lesezyklus des BRK-Vektors erst beendet und danach der NMI' ausgeführt, der dann nach Ende der Behandlungsroutine via RTI zu BRK zurückkehrt.

Falls (mit zurückgesetztem IRQ'-Sperrbit) ein IRQ' während eines BRK auftritt, beenden sowohl 6502 als auch 65C02 erst den Interrupt-Zyklus für BRK, sperren danach IRQ' und beginnen die IRQ'/BRK-Routine mit dem gesetzten Bit 4 des Statusregisters, d.h. als BRK. Die Firmware des Apple (sowohl normal als auch verbessert) entleert den Stack nach der Erkennung eines BRK und stellt den vorherigen Zustand (d.h. IRQ'-Sperrbit gelöscht) wieder her. Daraufhin wird als nächstes der IRQ' durchgeführt - auch hier vorausgesetzt, daß IRQ' in der Zwischenzeit aktiv geblieben ist, weil der Prozessor dem IRQ'-erzeugenden Baustein bis jetzt keine Antwort ("Acknowledge") gegeben hat. Nach dem RTI der IRQ'-Routine wird daraufhin der Rest der BREAK-Routine korrekt durchgeführt.

Der Mikroprozessor 65C02

Dieser Baustein wird von NCR, Rockwell und einigen Zweitherstellern gefertigt und ist ein Erzeugnis der letzten Jahre. Der Fortschritt der Fertigungstechnik wird hier deutlich: der 65C02 wird in CMOS-Technologie (6502: NMOS) produziert, der Stromverbrauch liegt um einige Zehnerpotenzen niedriger, die mögliche Taktfrequenz wurde von 1 bzw. 2 MHz auf 4 MHz gesteigert. Der 65C02 ist fast vollständig "aufwärtskompatibel" zum 6502, d.h. er kennt sämtliche Befehle des 6502 und dazu noch ein paar neue, hat aber für einige Befehle leicht unterschiedliche Ausführungszeiten. Ein 65C02 kann alle Programme ausführen, die für den 6502 geschrieben worden sind, solange es dabei nicht auf ganz genau abgestimmte Verzögerungszeiten ankommt; ein 6502 kann dagegen mit Programmen nichts anfangen, die die neuen Befehle des 65C02 nutzen. (Falls Sie jetzt an die zeitkritischen Programmteile von DOS 3.3, ProDOS, UCSD-Pascal und dem CP/M-BIOS denken, können Sie beruhigt sein: die unterschiedlichen Ausführungszeiten gelten hauptsächlich für die Adressierung über Pointer auf der Seite 0, die in den zeitkritischen Teilen für den Diskettenbetrieb nicht benutzt werden. Alle bekannten Betriebssysteme laufen mit dem 65C02 problemlos).

Der 65C02 gehört im Apple //c bereits zum Standard, ab Anfang 1986 wird auch der Apple //e standardmäßig in der verbesserten Version mit den neuen ROMs und dem 65C02-Prozessor sowie einer leicht geänderten Tastaturbelegung ausgeliefert. Ebenfalls seit Januar 1986 ist ein "Upgrade" für die älteren Modelle des //e erhältlich, das die ROMs und den 65C02 enthält - eine neue Tastatur ist aus verständlichen Gründen nicht dabei. Auch hier ist das primäre Ziel die Kompatibilität mit dem //c, Details über die neuen ROMs finden Sie in Kapitel 6.

Da es so aussieht, als würde der Apple in nächster Zukunft ein 65C02-Computer, werden wir uns hier und in weiteren Teilen dieses Buchs zumindest ansatzweise mit diesem Prozessor beschäftigen.

Der 65C02 verfügt über eine Reihe von zusätzlichen Befehlen, die dem Programmierer das Leben leichter machen (Beispiel: PHX und PHY, also direktes Speichern dieser Register auf dem Stack ohne den beim 6502 notwendigen Umweg über den Akku). Programme können damit wieder einmal etwas schneller und kompakter geschrieben werden. Außerdem sind einige der altbekannten Designfehler des 6502 beseitigt worden, Details finden Sie im Anhang dieses Kapitels. Aus diesem Grund beschränken wir uns hier auf ein paar Punkte, die durch die Datenblätter nicht vollständig geklärt werden.

Die von Rockwell und NCR hergestellten 65C02 sind nicht identisch - der Rockwell-65C02 kennt noch ein paar Befehle mehr, die der NCR-65C02 nicht hat. Diese Befehle beziehen sich alle auf Speicherstellen der Seite 0 und haben die Namen RMBn (Reset Memory Bit n), SMBn (Set Memory Bit n), BBRn (Branch on Bit n Reset) und BBSn (Branch on Bit n Set). Die entsprechenden Opcodes ($X7 und $XF) werden von dem NCR-Chip als NOPs interpretiert. Es sieht so aus, als würde Apple, Inc. NCR-kompatible 65C02-Prozessoren einsetzen, der 65C02 von

Rockwell funktioniert in einem Apple //e aber auch ohne Probleme. Die zusätzlichen Rockwell-Befehle finden Sie in den Tabellen 4.3 und 4.4 zusammen mit den dazugehörigen Ausführungszeiten.

Der 6502 kann während eines Schreibzyklus nicht über READY angehalten werden - beim 65C02 funktioniert auch das. Dabei erhebt sich natürlich die Frage, was mit dem Datenbus des //e passiert, wenn READY während eines Schreibzyklus auf "0" gebracht und dort für mehrere Zyklen gehalten wird. Wenn der 65C02 versucht, den Datenbus durchgehend während dieses Wartezustands zu kontrollieren, müßte es eigentlich spätestens am Ende von PHASE1 Ärger geben, wenn der RAM versucht, Daten für den Videoscanner auszugeben. Eine Messung zeigt, daß die Entwickler an ähnliche Probleme gedacht haben: der 65C02 kontrolliert während eines Wartezustands den Datenbus auch nur in den Zeitabschnitten, in denen er auch beim Normalbetrieb darauf zugreift. Aus diesem Grund ist der Datenbus auch während eines Wartezustands die überwiegende Zeit im hochohmigen Zustand, und es findet keine Kollision mit dem RAM am Ende von PHASE1 statt.

Die Tatsache, daß die Ausführung eines BRK von Interrupts nicht abgebrochen wird, wird auf Seite 3 des Datenblatts für den 65C02 als Verbesserung des 65C02 hervorgehoben. Mit "Interrupt" ist in diesem Fall NMI' gemeint - die IRQ'-Verarbeitung funktioniert auch beim 6502 bereits problemlos zu jedem Zeitpunkt. Die Probleme mit der NMI'-Verarbeitung während eines Interrupt-Zyklus für BRK und ihre Lösung im Design des 65C02 finden Sie im vorhergehenden Abschnitt besprochen.

Das NCR-Datenblatt bezeichnet die neuen Befehle zur Erhöhung und Heruntersetzen des Akku-Inhalts mit den mnemonischen Kürzeln INA bzw. DEA und lehnt sich damit an die bereits bekannten Befehle INX/DEX bzw. INY/DEY an. Das kann man auch andersrum sehen und diese Befehle als neue Adressierungsart für INC und DEC auffassen. Rockwell tut das und bezeichnet die beiden Befehle mit INC A bzw. DEC A oder einfach als INC und DEC. Die letzteren beiden folgen derselben Konvention wie bereits ASL, LSR, ROL und ROR, die sich ohne weitere Angaben eines Arguments ebenfalls auf den Akkuinhalt beziehen. Welche Art von Kürzeln zukünftige Standard-Assembler für die neuen Befehle "schlucken" werden, scheint noch nicht ganz klar zu sein - der von Apple, Inc. gelieferte EDASM bzw. die ProDOS-Version EDASM.SYSTEM erwarten auch bei den direkten Adressierungen den Zusatz "A" - statt LSR muß es LSR A heißen. Probleme mit der Speicherstelle $000A gibt es dabei keine - EDASM unterscheidet zwischen LSR A und LSR $A. Der Miniassembler dagegen, der im alten ROM des Apple II bereits vorhanden war und im ROM (in der verbesserten Version) des //e wieder auferstanden ist, erwartet grundsätzlich Hexzahlen: hier erzeugt die Eingabe von LSR A den Befehl zur Rotation der Speicherstelle $000A, für eine direkte Adressierung des Akkus muß es schlicht LSR heißen...

Eine weitere bemerkenswerte Eigenschaft des 65C02, die aus den "AC Characteristics" hervorgeht, ist, daß als maximale Zykluszeit 5000 Mikrosekunden angegeben werden. Eine vorsichtigere Interpretation dieser Angabe lautet, daß man den Takt während PHASE0 = "1" für 5000 Mikrosekunden anhalten kann, ohne daß der Prozessor dabei vergeßlich wird - nicht aber bei PHASE0 = "0". Das Datenblatt von Rockwell macht einen deutlicheren Unterschied zwischen den beiden Zuständen von PHASE0 und gibt an: "Der Takteingang kann im Zustand "1" für eine beliebig lange Zeit angehalten werden; wird er im Zustand "0" für länger als 5 Mikrosekunden angehalten, besteht die Gefahr des internen Datenverlusts". Dieser Punkt ist nur deshalb so wichtig, weil das Anhalten der Taktimpulse des Prozessors im Apple //e via DMA' den Takteingang auf den Zustand "0" bringt und dort festhält. Infolgedessen sind bei einem längeren DMA-Zugriff die Probleme dieselben sind wie mit dem 6502. In beiden Fällen kann ein längerer DMA-Zugriff nur dann stattfinden, wenn der Prozessor via READY in den Wartezustand gebracht worden ist und außerdem die Lötverbindungen X4 und X5 andersherum verbunden worden sind, so daß DMA' die Taktimpulse für den Prozessor nicht mehr unterbricht.

Eine weitere Besonderheit des 65C02 wird aus den Datenblättern nicht ersichtlich: Der Befehl "BIT #" (immediate) erzeugt andere Ergebnisse als BIT mit anderen Adressierungsarten. BIT (direkt) setzt nur das Zeroflag (Null-Flagge) entsprechend, alle anderen BIT-Befehle setzen die Flags Zero, Negative und Overflow über die Operation "Akku * Operand" sowie Bit 7 und 6 des Operanden.

Zum Schluß dieses Kapitels noch eine mehr spekulative Aussage über den 65C02: Dieser Prozessor ist (abgesehen von dem im Apple sowieso nicht verbundenen Ausgang ML') pin-kompatibel mit dem 6502. Der Austausch ist reichlich einfach - man nimmt den 6502 aus der Fassung auf der Hauptplatine heraus und steckt einen 65C02 hinein. Mit dem Apple //e funktioniert das großartig - mit dem Apple II funktioniert es nicht zuverlässig. Wieso?

Meine Vermutung geht dahin, daß der 65C02 (zumindest der von NCR) die Daten etwas eher haben möchte als der 6502 bei gleicher Taktfrequenz. Vom RAM ausgegebene Daten werden im Apple II erheblich später für den Prozessor verfügbar als im Apple //e, nämlich erst 60 ns vor der fallenden Flanke von PHASE 2 (//e: 250 ns). Der 6502 hat damit offensichtlich keine Probleme, dem 65C02 scheint diese Zeit manchmal etwas zu kurz zu sein.

Ich habe nur einige kleine Experimente mit einem Apple II unternommen und dabei mit zwei verschiedenen Exemplaren eines 65C02A (2 MHz?) sowie einem NCR-kompatiblen GTE G65SC02P-2 (2 MHz) verschiedentlich

Programmabstürze erlebt, die umso häufiger wurden, je mehr der periphere Datenbus durch Zusatzkarten belastet wurde. Ein Versuch mit einem R65C02P1 (1 MHz) von Rockwell zeigte ein überraschendes Ergebnis: hier waren keine Fehlfunktionen festzustellen.

Die Abstürze der NCR-Prozessoren passierten nur nach bestimmten Bytefolgen auf dem Datenbus: Wenn ein RTS-Befehl direkt nach einem NOP oder SBC kommt und die Videodaten des Apple II vor dem Einlesen des Opcodes für RTS einen Wert von $A0, $A2 oder $A9 haben, dann setzt der Prozessor das Carry-Flag während einer ansonsten normalen Ausführung des RTS(!). Dieses Verhalten war bei allen drei ausprobierten NCR-Chips nachzuweisen, einer der drei reagierte außerdem noch auf das Videodatum $89, ein anderer auf die Opcodes $89 und $E9 vor dem RTS. Ich habe das hier nur deshalb so detailliert aufgeführt, weil alle diese Bytes ($89, $A0, $A2, $A9 und $E9) Opcodes des 65C02 für direkte Adressierung sind.

In diesen Experimenten konnte ich nicht mit absoluter Sicherheit feststellen, daß der Grund für die Fehlfunktion in der zu kurzen Zeit zwischen "RAM-Daten gültig" und der fallenden Flanke von PHASE2 liegt - es handelt sich also um eine (allerdings gut untermauerte) Vermutung, auf die ich gerne eine Flasche Schampus verwetten möchte.[12] Ein weiterer Beweis dafür ist, daß Daten aus der 16k-Speichererweiterung problemlos gelesen werden - und die werden bereits nach der fallenden Flanke von Q3 (also in der Mitte von PHASE2) gültig. Aus diesen Gründen bin ich der Angabe von NCR gegenüber sehr skeptisch, daß der 65C02 mit 50 ns Trdst auskommt.

Befehlsübersicht und Unterschiede zwischen 6502 und 65C02

Der Zustand des Adreßbusses und des Datenbusses während der Ausführung eines Prozessorbefehls ist für den Programmierer normalerweise uninteressant. Eine genauere Betrachtung zeigt jedoch, daß der 6502 einige nicht so offensichtliche Eigenschaften besitzt, die bei der Programmierung von Ein- und Ausgabe berücksichtigt werden müssen, weil diese beiden Funktionen mangels eines "I/O-Signals" ebenfalls über die Adreßdekodierung laufen müssen. Speziell für Assemblerprogrammierer sind die folgenden Ausführungen deshalb unter Umständen wichtig - sogar die Ausführung von BASIC-Befehlen wird durch diese Ungewöhnlichkeiten beeinflußt.

Tabelle 4.1 stellt alle Möglichkeiten des 6502-Befehlsablaufs anhand willkürlich ausgewählter Beispiele dar und zeigt dabei den Zustand von Adreß- und Datenbus für jeden Zyklus innerhalb der Befehlsausführung. Wie zu sehen, ist der Ablauf der Befehle DEX und ASL A, PLA und PHA etc. gleich. Tabelle 4.2 ist der Schlüssel für alle anderen Befehle des 6502. Wenn Sie den Ablauf eines Befehls herausfinden wollen, müssen Sie sich dazu in 4.2 die entsprechende Befehlsgruppe heraussuchen , Sie finden darüber den Befehl in 4.1, dessen Ablauf mit dem des gesuchten Befehls identisch ist.

In Tabelle 4.1 wird durchgehend angenommen, daß der Opcode des Befehls von der Speicherstelle $1000 gelesen wird sowie das X- und Y-Register jeweils den Wert $20 enthalten. Befehle mit indexierter Adressierung über das Y-Register werden durch Beispiele mit indexierter Adressierung über X dargestellt, solange die beiden sich in der Ausführung nicht unterscheiden. Soweit möglich, werden Schreiboperationen (STA, STX, STY) durch LDA vertreten, in den entsprechenden Beispielen steht ein "w" hinter der Adresse. Zyklen, in denen immer geschrieben wird, sind durch ein großes "W" gekennzeichnet. Die Buchstaben "PX" stehen für Seitenüberschreitung ("Page Crossing"). Einige Beispiele zeigen zusätzlich noch den ersten Zyklus des nächsten Befehls, der mit "NEXT OP" bezeichnet wird.

Bei einigen Gelegenheiten adressiert der 6502 Speicherstellen, die überhaupt nichts mit der eigentlichen Operation zu tun haben. Das passiert dann, wenn der Prozessor eine interne Operation ausführt und im entsprechenden Zyklus überhaupt keinen Speicherzugriff durchführt. Indexierung und relative Sprünge über eine Seitengrenze hinweg erzeugen grundsätzlich eine überflüssige Adressierung der falschen Speicherseite.[13] Wenn der Prozessor bei einer indexierten Adressierung die Adresse über eine Seitengrenze hinweg berechnen muß, wird für die Erhöhung des höherwertigen Bytes der Adresse ein zusätzlicher Zyklus benötigt. Der Befehl "LDA $5472,X" benötigt z.B. vier Zyklen, wenn der Inhalt von X im Bereich von $0-8D liegt und sich so eine Adresse im Bereich von $5472-54FF ergibt, für größere Werte von X und daraus folgende Adressen $5500-5571 dauert der Befehl fünf Zyklen, für STA-Befehle mit indexierten Adressen ergibt sich dasselbe. Das *Synertek Programming Manual* (Ausgabe Mai 1978) gibt dazu den Hinweis, daß dieser zusätzliche Zyklus gebraucht wird, um das Beschreiben einer falschen Adresse ($5400 anstelle von $5500 für STA $5472,X mit X = $8E) zu verhindern.

---

12 Den Sekt hat er gewonnen: In der Ausgabe März 1985 hat NCR die Minimalzeit für Trdst auf 100 ns heraufkorrigiert - also klar zu lang für einen Apple II.

13 Wenn ein Sternchen (*) im Text auftaucht, gilt der Sachverhalt nicht mehr für den 65C02.

Es gibt noch andere interessante dunkle Flecken auf der Weste des 6502: "Read-Modify-Write"-Zyklen, wie ASL, LSR, ROL, ROR, INC und DEC, die eigentlich ein Byte lesen, es verändern und danach zurückschreiben sollten, führen eine doppelte Schreiboperation aus, beim ersten Mal wird das gerade gelesene Byte unverändert wieder in die Speicherstelle hineingeschrieben, erst beim zweiten Mal wird der veränderte Wert zurückgespeichert*. Das Herunterholen eines Bytes vom Stack (PLA, PLP, RTS, RTI) erzeugt erst einmal ein überflüssiges Ansprechen einer falschen Adresse auf der Seite 1, bevor der tatsächliche Wert gelesen wird. Alle überflüssigen Adressierungen finden als Lesezyklen statt, die dabei gelesenen Daten werden vom 6502 ignoriert.

Tabelle 4.1 6502-Instruktionen

	1	2	3	4	5	6	7
1. DEX	$1000 $CA	$1001 IGNORE					
2. ASL A	$1000 $0A	$1001 IGNORE					
3. PHA	$1000 $48	$1001 IGNORE	SPNT W DATA				
4. PLA	$1000 $68	$1001 IGNORE	SPNT IGNORE	SPNT+1 DATA			
5. RTS	$1000 $60	$1001 IGNORE	SPNT IGNORE	SPNT+1 PCL	SPNT+2 PCH	PCH,PCL IGNORE	PCH,PCL+1 NEXT OP
6. RTI	$1000 $40	$1001 IGNORE	SPNT IGNORE	SPNT+1 STATUS	SPNT+2 PCL	SPNT+3 PCH	PCH,PCL NEXT OP
7. BRK	$1000 $00	$1001 INGORE	SPNT W $10	SPNT-1 W $02	SPNT-2 W STATUS	$FFFE IRQLO	$FFFF IRQHI
8. BEQ $10 (Z=0)	$1000 $F0	$1001 $10	$1002 NEXT OP				
9. BEQ $10 (Z=1)	$1000 $F0	$1001 $10	$1002 IGNORE	$1012 NEXT OP			
10. BEQ $F3 (Z=1) (PX)	$1000 $F0	$1001 $F3	$1002 IGNORE	$10F5 IGNORE	$FF5 NEXT OP		
11. LDA #$AA	$1000 $A9	$1001 $AA					
12. LDA $70 STA $70	$1000 $A5	$1001 $70	$0070 w DATA				
13. ASL $70	$1000 $06	$1001 $70	$0070 OLD DATA	$0070 W OLD DATA	$0070 W NEW DATA		
14. LDA $70,X STA $70,X	$1000 $B5	$1001 $70	$0070 IGNORE	$0090 w DATA			
15. ASL $70,X	$1000 $16	$1001 $70	$0070 IGNORE	$0090 OLD DATA	$0090 W OLD DATA	$0090 W NEW DATA	
16. LDA $5772 STA $5772	$1000 $AD	$1001 $72	$1002 $57	$5772 w DATA			
17. ASL $5772	$1000 $0E	$1001 $72	$1002 $57	$5772 OLD DATA	$5772 W OLD DATA	$5772 W NEW DATA	
18. JMP $5772	$1000 $4C	$1001 $72	$1002 $57	$5772 NEXT OP			
19. JSR $5772	$1000 $20	$1001 $72	SPNT IGNORE	SPNT W $10	SPNT-1 W $02	$1002 $57	$5772 NEXT OP
20. LDA $5772,X (NO PX)	$1000 $BD	$1001 $72	$1002 $57	$5792 DATA			
21. LDA $57F2,X STA $57F2,X (PX)	$1000 $BD	$1001 $F2	$1002 $57	$5712 IGNORE	$5812 w DATA		
22. STA $5772,X (NO PX)	$1000 $9D	$1001 $72	$1002 $57	$5792 IGNORE	$5792 W DATA		
23. ASL $5772,X (NO PX)	$1000 $1E	$1001 $72	$1002 $57	$5792 OLD DATA	$5792 OLD DATA	$5792 W OLD DATA	$5792 W NEW DATA
24. ASL $57F2,X (PX)	$1000 $1E	$1001 $F2	$1002 $57	$5712 IGNORE	$5812 OLD DATA	$5812 W OLD DATA	$5812 W NEW DATA
25. LDA ($70,X) STA ($70,X)	$1000 $A1	$1001 $70	$0070 IGNORE	$0090 ADL	$0091 ADH	ADH,ADL w DATA	
26. LDA ($70),Y (NO PX)	$1000 $B1	$1001 $70	$0070 $72	$0071 $57	$5792 DATA		
27. LDA ($70),Y STA ($70),Y (PX)	$1000 $B1	$1001 $70	$0070 $F2	$0071 $57	$5712 IGNORE	$5812 w DATA	
28. STA ($70),Y (NO PX)	$1000 $91	$1001 $70	$0070 $72	$0071 $57	$5792 IGNORE	$5792 W DATA	
29. JMP ($5772) (NO PX)	$1000 $6C	$1001 $72	$1002 $57	$5772 PCL	$5773 PCH	PCH,PCL NEXT OP	
30. JMP ($57FF) (PX)	$1000 $6C	$1001 $FF	$1002 $57	$57FF PCL	$5700 PCH	PCH,PCL NEXT OP	

ADDR BUS
DATA BUS

W - WRITE CYCLE
w - WRITE CYCLE IF STORING INSTRUCTION
PX - PAGE CROSSING
NEXT OP - OP CODE NEXT INSTRUCTION
X-REG = $20, Y-REG = $20
$70/$71 CONTAIN $5772 OR $57F2 AS NEEDED FOR ILLUSTRATION

Beispiel 30 der Tabelle 4.1 zeigt einen wirklich obskuren Fehler des 6502: Der Befehl JMP (indirekt) ist nicht in der Lage, den neuen Inhalt des Programmzählers über eine Seitengrenze hinweg zu lesen. Ein "JMP" ($XXFF) liest das niederwertige Byte des Programmzählers von der Speicherstelle $XXFF - das höherwertige von $XX00 und nicht etwa von $(XX+1)00, wie man erwarten sollte*. Damit kann man natürlich herrlich Verwirrung stiften und andere Leute von der Analyse eines Programms abhalten (in den Locksmith-Versionen 4.2 bis 4.6 wird dieser Trick z.B. verwendet). Speziell auf dem Apple //e kann das aber ganz gewaltig Ärger geben - der 65C02 hat diesen Fehler nicht mehr.

Im Apple //e werden direkt über die Adressierung des 6502 drei Baugruppen betrieben: die seriellen Ausgänge, die oberen 16k RAM und die Ein-/Ausgabe zu den Zusatzkarten. Die Ausgänge zum Kassettenrecorder und zum Lautsprecher wechseln jedesmal ihren Zustand, wenn ihre Adresse angesprochen wird, wobei die Ansprechfrequenz im hörbaren Bereich liegen sollte. Wenn man z.B. versucht, den Lautsprecher über einen Befehl mit mehrfacher Adressierung zu betreiben, sollte man sich nicht wundern, wenn dabei nichts Brauchbares herauskommt: ein Befehl wie "STA $C030,X" mit X = 00 versucht, den Lautsprecher innerhalb von zwei Mikrosekunden hin- und herzubewegen. Für das Signal C040STROBE', das jeweils nur eine halbe Mikrosekunde auf "0" bleibt, könnte man mit dem 6502 sogar Impulsraten zustandebringen, von denen andere Prozessoren nur träumen können:

STA $C040	ein Strobe
STA $C040,X	zwei Strobes für X=0
ASL $C040	drei Strobes
ASL $C040,X	vier Strobes für X=0*

Bei Speicherzugriffen mit PEEK und POKE von BASIC aus ist es auch ganz nützlich, wenn man weiß, wie der eigentliche Zugriff stattfindet. Die folgende Tabelle zeigt die Adressierungsarten und die entsprechende Speicherstelle, von der dieser Befehl ausgeführt wird:

Applesoft PEEK ($E76F):	**LDA ($50),Y**
Applesoft POKE ($E781):	**STA ($50),Y**
Integer PEEK ($EEF9):	**LDA ($CE),Y**
Integer POKE ($EF0D):	**STA ($CE),Y**

In allen Fällen hat Y den Wert 0. Der Vergleich von PEEK und POKE mit den Beispielen 26 und 28 der Tabelle 4.1 zeigt, daß POKE einen doppelten Zugriff auf die angegebene Adresse, PEEK dagegen nur einen einfachen Zugriff zur Folge hat. Damit erklärt sich auch die Behauptung des *Applesoft BASIC Reference Manual*, daß Lautsprecher und Kassettenrecorder-Ausgang nur über PEEK, nicht aber über POKE betrieben werden können.

Durch die Art und Weise, wie die oberen 16k RAM gesteuert werden, eröffnen sich damit ungeahnte Möglichkeiten, völlig undurchschaubare Programme zu schreiben. Wir werden uns in Kapitel 5 detailliert mit diesem Speicherbereich und seiner Steuerung beschäftigen, deshalb hier nur ein kleiner Hinweis: Dieser Bereich läßt sich durch einen Lesevorgang der Softswitches $C081, $C083, $C089 oder $C08B auf "Lesen vom ROM, Schreiben zum RAM" und durch doppeltes Ansprechen dieser Adressen auf "Lesen und Schreiben vom/zum RAM" schalten (s. Tabelle 5.5). Normalerweise wird das durch eine Befehlsfolge wie "LDA $C081/LDA $C081" erreicht - wie oben demonstriert, läßt sich das auch durch "ASL $C081,X" oder "STA $C081,X" (mit X=0) erreichen. Im ersten Fall ergeben sich vier, im zweiten Fall zwei Zugriffe auf $C081 - RMW-Zyklen, die mit absolut indexierter Adressierung und ohne Seitenüberschreitung arbeiten, erzeugen zwei Lesezugriffe und zwei Schreibzugriffe (aber nur einen Schreibzugriff beim 65C02), wobei jedesmal die korrekte Adresse angesprochen wird. Selbstverständlich sollten solche Scherze in einem vernünftig geschriebenen Programm unterbleiben - es ist aber vorstellbar, daß man diese "Features" des 6502 einmal braucht.

Die "Write-Sector"-Routine von DOS 3.3 (Beginn: $B82A) zeigt ein derartiges Beispiel: Diskettenoperationen finden über die "state machine" des Diskettencontrollers statt, die so ähnlich aufgebaut ist wie der HAL. Für das Schreiben von Daten auf die Diskette ist eine Synchronisation des Schreibzyklus der "state machine" mit dem Zeitablauf des kontrollierenden (Prozessor-)Programms erforderlich. Der folgende Programmausschnitt prüft, ob die eingelegte Diskette schreibgeschützt ist, und setzt die "state machine" in den Wartezustand:

```
LDA  $C08D,X      ;mit X=$60 für Steckplatz 6
LDA  $C08E,X
BMI  WPROT        ;Abbruch, wenn geschützt
```

Wenn die Diskette nicht schreibgeschützt ist, geht es mit dem nächsten Befehl weiter. Die "state machine" befindet sich jetzt in einem Wartezustand und kann mit dem Befehl

```
STA  $C08F,X
```

synchron zum laufenden Programm aktiviert werden: der erste durch diesen Befehl erzeugte Zugriff auf $C0EF aktiviert die "state machine", der zweite Zugriff speichert die tatsächlich zu schreibenden Daten in das Schreib/Leseregister der Controllerkarte. Der Controller akzeptiert Daten nur dann, wenn er sie mit dem nächsten Taktimpuls nach der Aktivierung der "state machine" erhält sowie danach jeweils in Abständen, die ein ganzzahliges Vielfaches von 4 betragen. Deshalb bestehen die Schreibroutinen der RWTS aus Schleifen, deren Laufzeit exakt 32 Mikrosekunden (= 4 * 8) beträgt.

Falls Sie eine eigene RWTS schreiben, die grundsätzlich von Steckplatz 6 ausgeht, sollten Sie deshalb trotzdem den Befehl "STA $C08F,X" nicht durch "STA $C0EF" ersetzen - dadurch wird die "state machine" eine Mikrosekunde "daneben" gestartet und der Controller schreibt Unsinn auf die Diskette. "STA $C0EF,X" mit X=0 funktioniert - genauso wie "STA ($xx),Y", wobei der Pointer xx auf $C0EF zeigt und das Y-Register den Wert 0 hat. Bei einer Adressierung über einen Pointer auf der Seite 0 darf allerdings keine Seitengrenze überschritten werden, sonst erzeugt 6502 wie bekannt für den ersten Zugriff eine falsche Adresse. Der Diskettencontroller des Apple ist zweifellos so konstruiert worden, daß er über "STA $C080,X" mit X = Steckplatznummer * 16 angesprochen werden kann, also unabhängig von der Nummer des tatsächlichen Steckplatzes. Durch die eingesetzte Hardware ergibt sich aber auch, daß ein Programmierer tatsächlich bis in die letzten Details der 6502-Adressierung einsteigen muß, wenn dabei etwas Vernünftiges herauskommen soll.

Die Tabellen 4.3 und 4.4 haben dasselbe Format wie 4.1 und 4.2 und zeigen die Ausführungsabläufe des 65C02. Sie sind fast identisch, was die Befehle betrifft - der Ablauf einzelner Befehle unterscheidet sich dagegen manchmal recht stark, die entsprechenden Zyklen sind fett dargestellt.

Einige der Eigenheiten des 6502 haben sich auch (teilweise absichtlich) in den 65C02 hinübergerettet - andere dagegen nicht. Im einzelnen sind die folgenden Punkte zu beachten:

1. Indexierung oder relative Sprünge ergeben ebenfalls einen überflüssigen Lesezugriff - aber nicht auf $XX00 anstelle von $(XX+1)00, sondern auf die Adresse des Programmzählers (Beispiele 10, 21..).
2. "Read-Modify-Write"-Zyklen erzeugen nur einen Schreibzugriff auf den adressierten Operanden und maximal zwei Lesezugriffe. Ein Vierfach-Zugriff mit einem einzigen Befehl ist also nicht mehr möglich (Beispiele 13,15...). Das vorher gegebene Beispiel "ASL $C040,X" erzeugt somit zwei Lesezugriffe und einen Schreibzugriff auf $C040.
3. Der Befehl "JMP ($XXFF)" wird korrekt ausgeführt (Beispiel 30).

Tabelle 4.2 Cross-Referenz der 6502-Befehle

	IMP	REL	IMM	ACC	ØPG	ØPG X	ØPG Y	ABS	ABS X	ABS Y	IND	IND X	IND Y
ADC AND CMP EOR LDA ORA SBC			11		12	14		16	2Ø 21	2Ø 21		25	26 27
ASL LSR ROL ROR				2	13	15		17	23 24				
BCC BCS BEQ BMI BNE BPL BVC BVS		8,9 1Ø											
CLC CLD CLI CLV DEX DEY INX INY NOP SEC SED SEI TAX TAY TSX TXA TXS TYA	1												
BIT					12			16					
BRK	7												
CPX CPY			11		12			16					
DEC INC					13	15		17	23 24				
JMP								18			29 3Ø		
JSR								19					
LDX			11		12		14	16		2Ø 21			
LDY			11		12	14		16	2Ø 21				
PHA PHP	3												
PLA PLP	4												
RTI	6												
RTS	5												
STA					12	14		16	21 22	21 22		25	27 28
STX					12		14	16					
STY					12	14		16					

Tabelle 4.3 Befehlsabläufe des 65C02

	1	2	3	4	5	6	7	8
1. DEX	$1000 $CA	$1001 IGNORE						
2. ASL A	$1000 $0A	$1001 IGNORE						
3. PHA	$1000 $48	$1001 IGNORE	SPNT W DATA					
4. PLA	$1000 $68	$1001 IGNORE	SPNT IGNORE	SPNT+1 DATA				
5. RTS	$1000 $60	$1001 IGNORE	SPNT IGNORE	SPNT+1 PCL	SPNT+2 PCH	PCH,PCL IGNORE	PCH,PCL+1 NEXT OP	
6. RTI	$1000 $40	$1001 IGNORE	SPNT IGNORE	SPNT+1 STATUS	SPNT+2 PCL	SPNT+3 PCH	PCH,PCL NEXT OP	
7. BRK	$1000 $00	$1001 INGORE	SPNT W $1Ø	SPNT-1 W $02	SPNT-2 W STATUS	$FFFE IRQLO	$FFFF IRQHI	
8. BEQ $10 (Z=0)	$1000 $F0	$1001 $10	$1002 NEXT OP					
9. BEQ $10 (Z=1)	$1000 $F0	$1001 $10	$1002 IGNORE	$1012 NEXT OP				
10. BEQ $F3 (Z=1) (PX)	$1000 $F0	$1001 $F3	$1002 IGNORE	**$1002 IGNORE**	$0FF5 NEXT OP			
11. LDA #$AA	$1000 $A9	$1001 $AA	**$1002 D IGNORE**					
12. LDA $70 STA $70	$1000 $A5	$1001 $70	$0070 w DATA	**$1002 D IGNORE**				
13. ASL $70	$1000 $06	$1001 $70	$0070 OLD DATA	**$0070 OLD DATA**	$0070 W NEW DATA			
14. LDA $70,X STA $70,X	$1000 $B5	$1001 $70	**$1001 $70**	$0090 w DATA	**$1002 D IGNORE**			
15. ASL $70,X	$1000 $16	$1001 $70	**$1001 $70**	$0090 OLD DATA	**$0090 OLD DATA**	$0090 W NEW DATA		
16. LDA $5772 STA $5772	$1000 $AD	$1001 $72	$1002 $57	$5772 w DATA	**$1003 D IGNORE**			
17. ASL $5772	$1000 $0E	$1001 $72	$1002 $57	$5772 OLD DATA	**$5772 OLD DATA**	$5772 W NEW DATA		
18. JMP $5772	$1000 $4C	$1001 $72	$1002 $57	$5772 NEXT OP				
19. JSR $5772	$1000 $20	$1001 $72	SPNT IGNORE	SPNT W $1Ø	SPNT-1 W $02	$1002 $57	$5772 NEXT OP	
20. LDA $5772,X (NO PX)	$1000 $BD	$1001 $72	$1002 $57	$5792 DATA	**$1003 D IGNORE**			
21. LDA $57F2,X STA $57F2,X (PX)	$1000 $BD	$1001 $F2	$1002 $57	**$1002 $57**	$5812 w DATA	**$1003 D IGNORE**		
22. STA $5772,X (NO PX)	$1000 $9D	$1001 $72	$1002 $57	$5792 IGNORE	$5792 W DATA			
23. ASL $5772,X (NO PX)	$1000 $1E	$1001 $72	$1002 $57	$5792 OLD DATA	$5792 OLD DATA	**$5792 W NEW DATA**		
24. ASL $57F2,X (PX)	$1000 $1E	$1001 $F2	$1002 $57	**$1002 $57**	$5812 OLD DATA	**$5812 OLD DATA**	$5812 W NEW DATA	
25. LDA ($70,X) STA ($70,X)	$1000 $A1	$1001 $70	**$1001 $70**	$0090 ADL	$0091 ADH	ADH,ADL w DATA	**$1002 D IGNORE**	
26. LDA ($70),Y (NO PX)	$1000 $B1	$1001 $70	$0070 $72	$0071 $57	$5792 DATA	**$1002 D IGNORE**		
27. LDA ($70),Y STA ($70),Y (PX)	$1000 $B1	$1001 $70	$0070 $F2	$0071 $57	**$0071 $57**	$5812 w DATA	**$1002 D IGNORE**	
28. STA ($70),Y (NO PX)	$1000 $91	$1001 $70	$0070 $72	$0071 $57	**$0071 $57**	$5792 W DATA		
29. JMP ($5772) (NO PX)	$1000 $6C	$1001 $72	$1002 $57	**$1002 $57**	**$5772 PCL**	**$5773 PCH**	**PCH,PCL NEXT OP**	
30. JMP ($57FF) (PX)	$1000 $6C	$1001 $FF	$1002 $57	**$1002 $57**	**$57FF PCL**	**$5800 PCH**	**PCH,PCL NEXT OP**	
31. LDA ($70) STA ($70)	**$1000 $B2**	**$1001 $70**	**$0070 $72**	**$0071 $57**	**$5772 DATA**	**$1002 D IGNORE**		
32. JMP ($5772,X)	**$1000 $7C**	**$1001 $72**	**$1002 $57**	**$1002 $57**	**$5792 PCL**	**$5793 PCH**	**PCH,PCL NEXT OP**	
33. BBS0 $70,$10*	**$1000 $8F**	**$1001 $70**	**$0070 $72**	**$0070 $72**	**$1002 $10**	**$1003 NEXT OP**		
34. BBS1 $70,$10* (NO PX)	**$1000 $9F**	**$1001 $70**	**$0070 $72**	**$0070 $72**	**$1002 $10**	**$1003 IGNORE**	**$1013 NEXT OP**	
35. BBS1 $70,$F3* (PX)	**$1000 $9F**	**$1001 $70**	**$0070 $72**	**$0070 $72**	**$1002 $F3**	**$1003 IGNORE**	**$1003 IGNORE**	**$0FF6 NEXT OP**
36. $X3, $XB ** $X7, $XF ***	**$1000 $03**							
37. $5C $5772 **	**$1000 $5C**	**$1001 $72**	**$1002 $57**	**$FF72 IGNORE**	**$FFFF IGNORE**	**$FFFF IGNORE**	**$FFFF IGNORE**	**$FFFF IGNORE**

ADDR BUS
DATA BUS

W - WRITE CYCLE
w - WRITE CYCLE IF STORING INSTRUCTION
D - ONE CYCLE EXTENSION OF ADC OR SBC IF DECIMAL MODE
P - PAGE CROSSING

NEXT OP - OP CODE NEXT INSTRUCTION
* - AVAILABLE IN ROCKWELL BUT NOT NCR 65C02
** - UNUSED OP CODES (NOPs) IN ALL 65C02
*** - UNUSED OP CODES (NOPs) IN NCR 65C02 ONLY

X-REG = $20, Y-REG = $20.
$70/$71 contain $5772 or $57F2 as needed for illustration.
Boldfaced type is used where 65C02 is different from 6502.

Tabelle 4.4 Cross-Referenz der 65C02-Befehle

	IMP	REL	IMM	ACC	ØPG	ØPG X	ØPG Y	ABS	ABS X	ABS Y	ABS IND	ØPG IND X	ØPG IND Y	ØPG IND	ABS IND X	ØPG REL	*
ADC AND CMP EOR LDA ORA SBC			11		12	14		16	20 21	20 21		25	26 27	31			
ASL LSR ROL ROR DEC INC				2	13	15		17	23 24								
BBRn BBSn **																**33 34 35**	
BCC BCS BEQ BMI BNE BPL BVC BVS		8,9 10															
BRA		**9 10**															
CLC CLD CLI CLV DEX DEY INX INY NOP SEC SED SEI TAX TAY TSX TXA TXS TYA	1																
BIT			**11**		12	**14**		16	**20 21**								
BRK	7																
CPX CPY			11		12			16									
JMP								18			29 30				32		
JSR								19									
LDX			11		12		14	16		20 21							
LDY			11		12	14		16	20 21								
PHA PHP **PHX PHY**	3																
PLA PLP **PLX PLY**	4																
RMBn SMBn **					13												
RTI	6																
RTS	5																
STA					12	14		16	21 22	21 22		25	27 28	31			
STX					12		14	16									
STY					12	14		16									
STZ					**12**	**14**		**16**	**21 22**								
TRB TSB					13			17									
02 22 42 62 82 C2 E2 ***			11														
X3 XB * X7 XF******																	36
44 ***					12												
54 D4 F4 ***						14											
5C ***																	37
DC FC ***								16									

* unused op codes $X3, $XB, $5C (NCR and Rockwell) and $X7, $XF (NCR) generate abnormal addressing modes.
** BBRn, BBSn, RMBn, and SMBn are found on Rockwell 65C02 but not NCR 65C02.
*** unused op codes for NCR and Rockwell 65C02.
**** unused op codes for NCR 65C02 but not Rockwell 65C02.

Kapitel 5

RAM und die Verwaltung des Speichers

Man sollte eigentlich meinen, daß die Besprechung des RAMs und der dazugehörigen Elektronik recht kurz und schmerzlos ausfällt. Der Prozessor schreibt entweder Daten in den RAM hinein oder liest welche - was sonst?

Tatsächlich macht das der Prozessor auch so. Allerdings haben wir noch den Videoscanner, der ebenfalls Daten aus dem RAM liest. Außerdem arbeiten sowohl CPU als auch der Videoscanner mit dem AUX-RAM[1]. Dann haben wir *dynamische* RAMs mit 64 kBit und gemultiplexten Adressen, dazu eine sehr verzwickte Bankumschaltung... Das reicht für ein eigenes Kapitel.

Wenn der RAM vom Prozessor angesprochen wird, muß eine entsprechende Elektronik auf der Hauptplatine Signale aktivieren, die den RAM-Chips mitteilen, ob Daten gelesen oder geschrieben werden sollen. Diese Aufgabe übernimmt im Apple //e die MMU. Ein Programm legt über das Setzen von MMU-Softswitches die Speicherkonfiguration fest, die MMU reagiert aufgrund der Stellung der Softswitches mit der Aktivierung einzelner Bau- und Funktionsgruppen auf bestimmte Adressen bzw. Adreßbereiche. Die Aktivierung von Bausteinen kann dabei sowohl durch ein direktes Kontrollsignal der MMU als auch auf indirektem Wege erfolgen. Apple, Inc. bezeichnet diese Kontrollfunktion der MMU als *Memory Management* (= Speicherverwaltung). Wie in den folgenden Abschnitten gezeigt wird, besteht der größte Teil der Speicherverwaltung aus der des RAMs.

Wir werden uns in diesem Kapitel mit den Bedürfnissen und Nöten von dynamischen 64 kBit RAM-Chips und den konstruktiven Details ihres Aufbaus im Apple //e beschäftigen. Außerdem wird gezeigt, wie die MMU den Adreßbus überwacht, um die einzelnen Speicherbereiche zu verwalten.

Die dynamischen RAMs

Die im Apple //e verwendeten RAM-Bausteine haben jeweils 65536 Speicherzellen mit einer Breite von einem *Bit* ("65536 * 1 - Organisation"). Wie bereits in Bild 2.7 gezeigt, werden acht dieser Bausteine für die 64k Byte RAM auf der Hauptplatine benötigt.

64k-RAMs sind der momentane Industriestandard (in den letzten Jahren waren es 16k, in den nächsten werden es 256k und 1 Megabit sein), sie werden von mehreren Herstellern gefertigt und sind für verschiedene Geschwindigkeiten erhältlich. Die Rate von 2MHz, mit der der Apple auf den RAM zugreift, liegt mehr oder weniger an der unteren Grenze - mittlerweile sind 64k-Bausteine verfügbar, die bis zu 16 Millionen Zugriffe pro Sekunde erlauben.

Ein 64 kBit RAM-Baustein ist in einem Plastikgehäuse mit 16 Pins verpackt und benötigt (im Gegensatz zu den älteren 16k-RAMs) nur noch jeweils eine Zuführung von +5 Volt und Masse. Bild 5.2 zeigt für den mit F6 bezeichneten Chip die detaillierte Anschlußbelegung. Neben den Zuführungen für die Stromversorgung existiert ein Anschluß zur Eingabe von Daten, einer mit tri-State-Charakteristik zur Ausgabe von Daten und ein Eingang für R/W' Control.

Um eine aus 65536 Speicherzellen zu adressieren, werden 16 Adreßleitungen benötigt - der RAM-Chip hat aber nur acht Adreßeingänge. Aus diesem Grund muß eine Adresse gemultiplext werden: zuerst kommt eine ROW-Adresse mit 8 Bit, danach eine COLUMN-Adresse mit den zweiten 8 Bit. Diese an sich komplizierte Methode wird etwas verständlicher, wenn man sich einen RAM als Matrix aus 256 * 256 Speicherzellen vorstellt.[2] Die erste 8-Bit-Adresse gibt an, in welcher Reihe die gewünschte Zelle liegt, die zweite 8-Bit-Adresse wählt danach eine Zelle die-

---

1 Solange nicht anders angegeben, gehen wir in diesem Buch immer davon aus, daß im speziellen Steckplatz 64 kByte AUX-RAM installiert sind.

2 Wie bereits in den vorigen Kapiteln dieses Buchs unterscheiden wir zwischen *Speicherzellen* und Speicherstellen. Mit dem ersten Begriff ist eine einzelne Zelle eines Bausteins gemeint, die eine von der Baustein-Organisation abhängige Bit-Breite hat, mit dem zweiten grundsätzlich eine Speicheradresse mit 8 Bit Breite.

ser Reihe. Die ROW-Adresse wird mit der fallenden Flanke von RAS' vom Adreßbus übernommen, die COLUMN-Adresse mit der fallenden Flanke von CAS'. Zusammen mit der Übernahme der zweiten 8-Bit-Adresse wird die Lese- oder Schreibaktion gestartet.

Bild 5.1 zeigt die Signale des Taktgenerators, über die RAM-Zugriffe im Apple //e kontrolliert werden. Die Art der Signale und ihr Zusammenhang wird von den folgenden Punkten bestimmt:

- der Charakteristik der dynamischen 64 kBit-RAMs;
- den Zeitanforderungen des mit 1 MHz laufenden 6502-Prozessors;
- dem abwechselnden Zugriff von Videoscanner und Prozessor in jedem Zyklus.

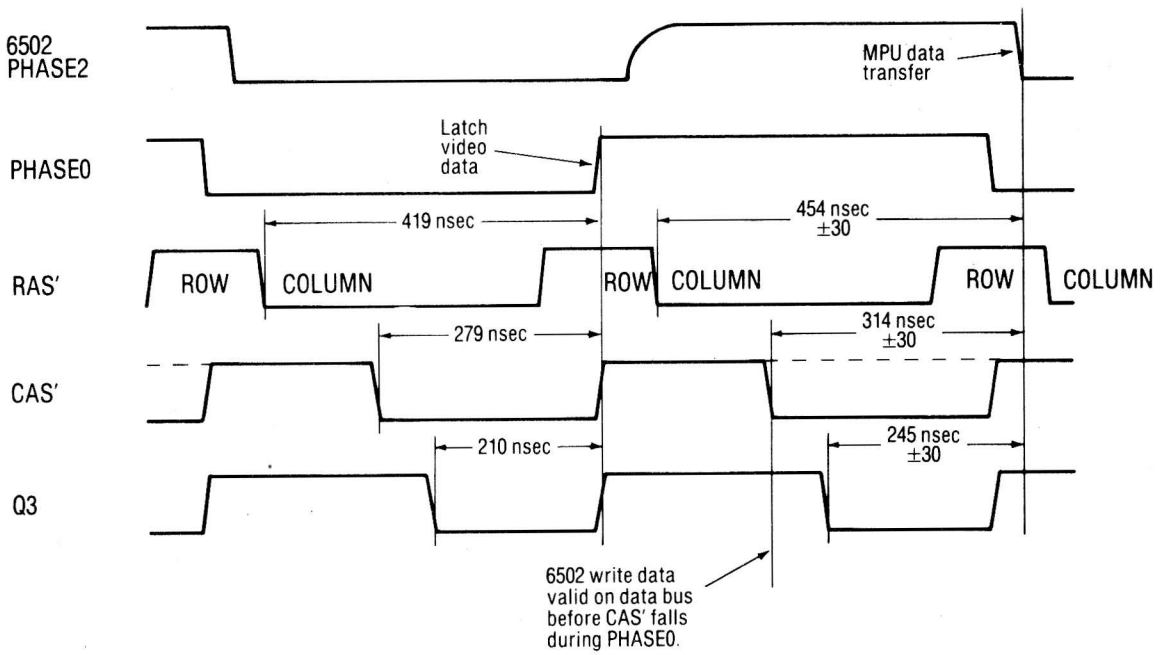
PHASE0 und RAS' liefern den Bezugspunkt für die RAM-Zugriffe von Videoscanner und CPU (die sich jeweils in ROW und COLUMN unterteilen), PHASE0 bestimmt, wer von beiden auf den RAM zugreift.

RAS' und CAS' sind direkt mit den entsprechenden Eingängen der RAM-Bausteine auf der Hauptplatine verbunden - während PHASE0 aktiv ist, wird CAS' nur dann aktiv, wenn der Prozessor auch wirklich auf den RAM der Hauptplatine zugreift. Die Bausteine des AUX-RAMs sind ebenfalls mit RAS' direkt verbunden, anstelle von CAS' erhalten sie das Signal Q3, das auch während PHASE0 immer eine fallende Flanke hat.

Auf die fallende Flanke von RAS' wird die ROW-Adresse von den RAMs übernommen. Das bedeutet natürlich, daß zu diesem Zeitpunkt eine gültige ROW-Adresse an den entsprechenden Anschlüssen anliegen muß. Analoges gilt für CAS' und die COLUMN-Adresse. Die *Adreßmultiplexer* innerhalb der IOU und der MMU sorgen dafür, daß das der Fall ist.

CAS' signalisiert den Beginn des Datentransfers durch seine fallende Flanke, wenn RAS' sich zu diesem Zeitpunkt bereits auf dem Pegel "0" befindet. Der RAM der Hauptplatine muß spätestens 374 Nanosekunden nach der fallenden Flanke von RAS' bzw. 234 ns nach CAS den Inhalt der adressierten Speicherzelle ausgegeben haben (Lesezugriff), die Bausteine des AUX-RAMs haben dafür 256 ns nach RAS' bzw. 147 ns nach der fallenden Flanke von Q3 Zeit. Diese Bedingungen werden von RAM-Bausteinen mit einer Zugriffszeit von 200 ns (oder weniger) erfüllt, d.h. eigentlich von allen RAM-Bausteinen, die auf dem Markt erhältlich sind.

Bild 5.1 Die Zeitsignale der RAM-Adressierung



Dynamische RAM-Bausteine benötigen einen konstanten *Refresh* ("Auffrischung"), damit sie nicht vergeßlich werden. Die Bedingung lautet, daß alle der 256 ROW-Adressen periodisch innerhalb von maximal 2 Millisekunden angesprochen werden müssen).[3] Dieses Ansprechen kann entweder durch RAS'/CAS'-Zyklen geschehen oder auch durch Zyklen, in denen die RAM-Bausteine nur eine ROW-Adresse ("RAS' only") erhalten. Im Apple //e wird der Refresh durch RAS'/CAS'-Zyklen während PHASE1 sichergestellt, nämlich über das zyklische Durchlaufen eines Speicherbereichs durch den Videoscanner.

Die Verbindungen des RAMs im Apple //e

Die generellen Flußwege für RAM-Daten sind bereits in Kapitel 2 besprochen worden. Bitte blättern Sie noch einmal zurück zu Bild 2.7 ("Die komplette Busstruktur des Apple //e") - es vermittelt einen guten Überblick der Gesamtzusammenhänge. Fassen wir einmal zusammen, was bereits über den RAM gesagt wurde:

1. Der RAM der Hauptplatine und der AUX-RAM im speziellen Steckplatz bestehen jeweils aus acht Bausteinen mit einer Kapazität von 64 kBit und einer internen Organisation von 65536 * 1 Bit.
2. Die Dateneingänge und die Datenausgänge des RAMs der Hauptplatine sind direkt mit der CPU ("Lesen" und "Schreiben") sowie mit dem Videolatch der Hauptplatine (nur "Lesen") verbunden.
3. Die Dateneingänge und die Datenausgänge des AUX-RAMs sind direkt mit dem AUX-Videolatch sowie über einen bidirektionalen Treiber mit dem Datenbus der Hauptplatine verbunden. Der bidirektionale Treiber trennt den AUX-RAM vom Datenbus, solange die CPU nicht auf den AUX-RAM zugreift.
4. Die Eingänge der beiden Videolatches sind mit den Datenausgängen der RAMs verbunden, die Ausgänge der Videolatches mit dem Videogenerator.
5. Die Adressierung der RAMs geschieht in gemultiplexter Form. Während eines Prozessorzyklus enthält der RAM-Adreßbus nacheinander die ROW-Adresse und die COLUMN-Adresse des Videoscanners aus der IOU, gefolgt von der ROW-Adresse und der COLUMN-Adresse des Prozessors aus der MMU.

Der in Bild 5.2 gezeigte vollständige Schaltplan der RAMs enthält sämtliche Verbindungen der RAM-Bausteine zu den anderen Baugruppen und den Datenbussen. Die RAMs sind untereinander in einer Art und Weise verbunden, die an die Steckplätze erinnert: die Mehrzahl der Verbindungen zieht sich einfach von Baustein zu Baustein. Darunter fallen die Adreßleitungen (RA0-RA7), RAS' und die Stromversorgung (+5 Volt und Masse). Auf der Hauptplatine sind CAS' und RAM R/W' durchverbunden, auf der Zusatzkarte Q3 und R/W'80.

Durch diese Verbindungen sind sowohl Schreib- und Lesezugriffe der CPU auf den RAM der Hauptplatine oder auf den AUX-RAM als auch der gleichzeitige Zugriff des Videoscanners auf beide RAM-Gruppen möglich. Während PHASE1 aktiviert der Videoscanner beide RAM-Gruppen gleichzeitig. Der AUX-RAM ist dabei vom Datenbus der Hauptplatine durch den bidirektionalen Treiber abgetrennt, vom RAM der Hauptplatine ausgegebene Daten werden vom Prozessor ignoriert. Die von den RAM-Gruppen ausgegebenen Videodaten werden auf die steigende Flanke von PHASE0 von den dazugehörigen Videolatches übernommen. Während PHASE0 tauscht die CPU Daten mit dem RAM der Hauptplatine, dem AUX-RAM oder einer anderen Baugruppe aus. Die Übernahme von Daten findet kurz nach der fallenden Flanke von PHASE0 statt, nämlich auf die fallende Flanke des (nicht weiter verbundenen) Prozessorausgangs PHASE2.

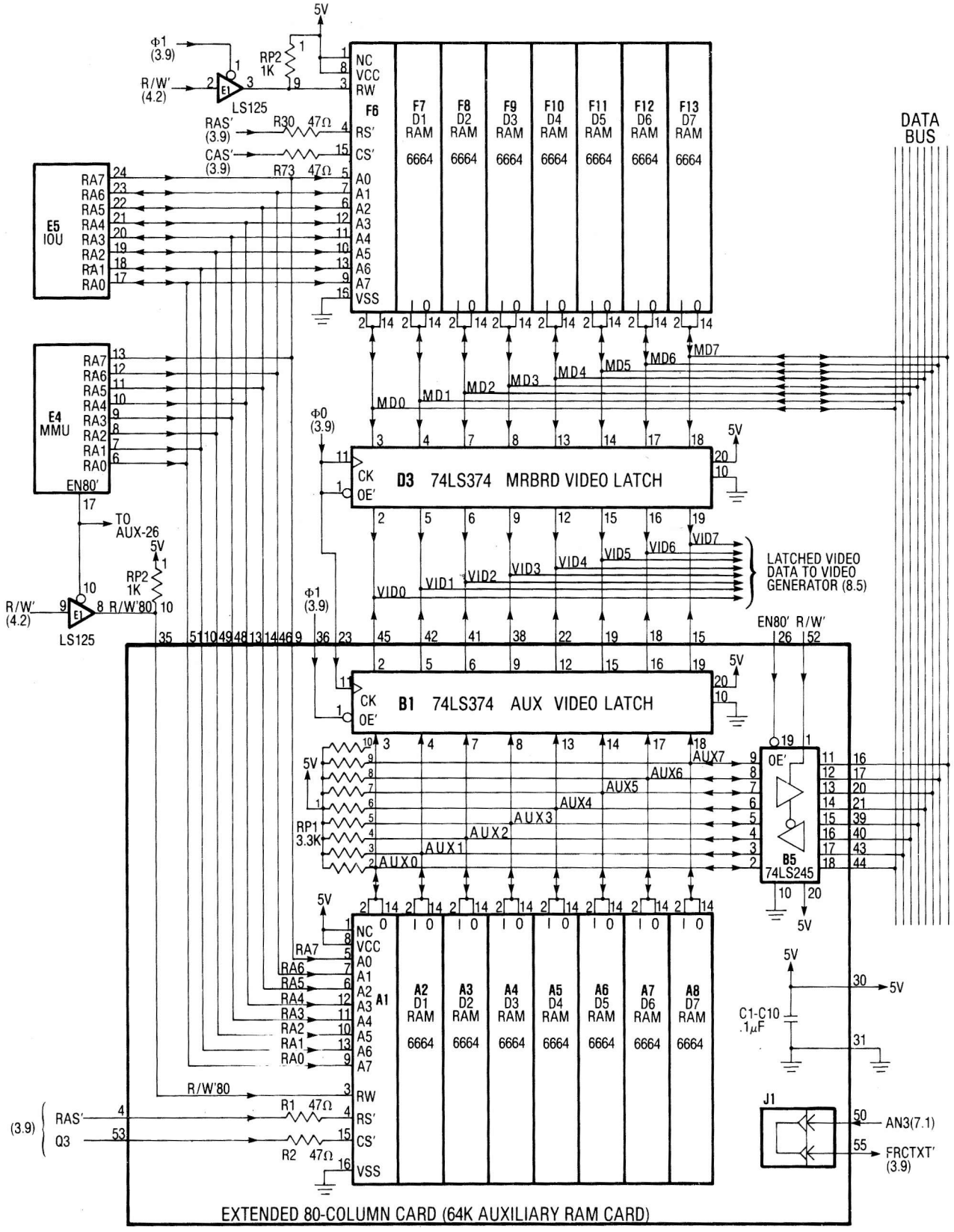
Der Ablauf eines RAM-Zugriffs ist komplex, wird aber im Apple //e durch wenige Bauteile realisiert. Die kritischen Kontrollelemente sind die Signale CASEN' und EN80' von der MMU, die Schreib/Lesekontrolle und der CAS'-Eingang der RAM-Bausteine sowie die Steuerleitung der tri-State-Treiber des AUX-RAM-Datenbusses.

Das Signal RAM R/W' auf der Hauptplatine ist nicht mit R/W' Control des 6502 identisch - dazwischen sitzt ein tri-State-Treiber, dessen Ausgang nur dann aktiv wird, wenn PHASE1 den Zustand "0" hat. Das Signal R/W' Control der CPU wird damit nur während PHASE0 zu den RAMs durchgeschaltet. Der Grund dafür: Der 6502 setzt R/W' bei Schreibzyklen manchmal auch während PHASE1 auf "0" und würde damit die Videoausgabe stören, wenn diese Leitung direkt durchverbunden wäre. Durch den Treiber hat der Videoscanner während PHASE1 eine komplett von der CPU unabhängige Kontrolle des RAMs.

---

3 Einige Hersteller produzieren RAM-Bausteine mit 64 kBit, die nur 128 verschiedene ROW-Adressen für den Refresh benötigen. Im Apple //e können beide Typen eingesetzt werden.

Bild 5.2 Schematische Darstellung der Verbindungen des RAM's im Apple //e



Das Signal CAS' wird vom HAL erzeugt und ist direkt mit den CAS'-Eingängen der RAM-Chips (Pin 15) auf der Hauptplatine verbunden. Die RAMs reagieren auf eine fallende Flanke von CAS' nach RAS' mit einem Datentransfer in der durch den Pegel von RAM R/W' angegebenen Richtung. Die CAS'-Erzeugung im HAL wird über CASEN' von der MMU gesteuert. Dieses Signal wird von der MMU aktiv gesetzt, wenn der Prozessor eine Adresse anspricht, die über die MMU-internen Softswitches dem RAM der Hauptplatine zugeordnet ist. Nur wenn die MMU CASEN' aktiv setzt und so eine fallende Flanke von CAS' während PHASE0 erlaubt, findet ein Datenaustausch zwischen CPU und RAM statt. Während PHASE1 setzt die MMU CASEN' immer aktiv - der Videoscanner hat während dieser Zeit immer Zugriff auf den RAM der Hauptplatine.

Der Datenfluß zwischen CPU und AUX-RAM wird auf eine andere Weise kontrolliert: Q3 ist mit den CAS'-Eingängen der AUX-RAM-Bausteine direkt verbunden und liefert nicht nur während PHASE1, sondern auch während PHASE0 immer eine fallende Flanke. Der AUX-RAM hat damit pro Prozessorzyklus immer zwei vollständige Zugriffe. Der AUX-Datenbus ist aber nur mit dem Datenbus der Hauptplatine verbunden, wenn die CPU wirklich auf den AUX-RAM zugreift - ansonsten bleibt EN80' inaktiv und der bidirektionale Treiber (LS245) bleibt genauso wie während des Videoscanner-Zugriffs im hochohmigen Zustand. Bei diesem Treiber ist nicht nur die Datenrichtung umschaltbar, er verfügt zusätzlich noch über eine Steuerleitung, über die seine Ausgänge abgeschaltet werden können. EN80' ist während PHASE1 immer auf "1" und geht während PHASE0 nur dann auf "0", wenn der Prozessor auf den AUX-RAM zugreift.

Die Datenrichtung des LS245 wird direkt über R/W' des Prozessors geschaltet, die R/W'-Eingänge der AUX-RAM-Bausteine dagegen über R/W'80. R/W'80 entspricht R/W' Control, wird aber zusätzlich noch über EN80' geschaltet und geht nur auf "Schreiben", wenn EN80' aktiv ist. Da EN80' nur während PHASE0 aktiv sein kann, führt der (während PHASE1 aktive) Videoscanner automatisch immer nur Lesezugriffe auf den AUX-RAM durch.

Die Behandlung der vom RAM ausgegebenen Daten enthält noch einige subtile, aber wichtige Kleinigkeiten: Der Datenbus ist am Ende von PHASE1 immer verfügbar, die CPU kontrolliert ihn zu diesem Zeitpunkt auch bei einem Schreibzyklus (noch) nicht, sondern nur während der zweiten Hälfte von PHASE0 und (bei Schreibzyklen) etwas darüber hinaus. Auch während eines Schreibzyklus der CPU kann den Videoscanner nichts und niemand daran hindern, während PHASE1 auf den RAM zuzugreifen - als Ergebnis haben wir ein völlig flimmerfreies Bild auch dann, wenn die CPU gerade darin "herumschreibt".

Ein weiterer interessanter Punkt liegt darin, daß die Videodaten des RAMs der Hauptplatine nicht nur für das Videolatch, sondern auch auf dem Datenbus verfügbar sind, wenn PHASE0 wieder von "0" auf "1" wechselt. Diese Daten werden über den bidirektionalen Treiber auf den peripheren Datenbus sofort weitergegeben, wenn R/W' "1" ist, oder mit einer sehr kurzen Verzögerung, falls R/W' "0" ist. Daraus folgt, daß die Videodaten des Hauptplatinen-RAMs von einer einfach konstruierbaren Zusatzkarte gelesen und eventuell weiterverarbeitet werden könnten.

Sowohl in den 40er als auch in den 80er Modi sind die vom AUX-Videolatch gespeicherten Daten während PHASE0, die Daten vom Videolatch der Hauptplatine während PHASE1 auf dem Videodatenbus verfügbar. Der Unterschied zwischen beiden Modi liegt *nicht* im RAM, sondern in der Takterzeugung: in den 40er Modi wird kein LDPS'-Puls während PHASE0 erzeugt - dadurch werden die anliegenden Daten des AUX-RAMs ignoriert und der Videogenerator verarbeitet nur die Videodaten des Hauptplatinen-RAMs, die über einen LDPS'-Puls während PHASE1 eingeladen werden. In den 80er Modi wird LDPS' sowohl einmal während PHASE1 als auch einmal während PHASE0 erzeugt. Der Videogenerator wird dadurch mit der doppelten Datenmenge pro Zyklus versorgt, entsprechend wird VID7M auf die doppelte Frequenz geschaltet, und ein 7-Bit-Datum braucht nur noch anstelle eines halben einen ganzen Prozessorzyklus, bis es aus dem Schieberegister des Videogenerators herausgeschoben wird.

Im Modus LoRes40 wird allerdings auch mit der doppelten Taktrate gearbeitet, wir werden noch in Kapitel 8 darauf zurückkommen.

Noch eine Anmerkung zum in Bild 5.2 gezeigten Schaltplan: Sie werden sich vielleicht bereits gewundert haben, wieso RA0 zum Anschluß A7, RA1 zum Anschluß A6 usw. geht - sollte da etwas durcheinander gekommen sein? Tatsächlich sieht es so aus, daß es überhaupt keine Rolle spielt, welche Adreßleitung mit welchem Anschluß verbunden ist: ob über eine Adresse wie 11110000 tatsächlich RAM-intern die Reihe 11110000 beschrieben wird oder 00001111, ist völlig egal, solange diese "verkehrte" Zuordnung auch beim Lesen benutzt wird.[4]

Auch die vom Hersteller vorgegebene Bezeichnung ist so gesehen völlig willkürlich - allerdings hält man sich in den meisten Fällen daran, um unnötige Verwirrung zu vermeiden. (Ob sich wohl bei Apple, Inc. das Layout-Programm verschluckt hat?)

---

4 Eine Ausnahme stellen RAM-Chips mit 128 Zyklen in Systemen dar, bei denen nur die Adressleitungen A0 bis A6 für den Refresh angesprochen werden. Der Apple //e durchläuft immer 256 Zyklen und benutzt dafür A0 bis A7, dadurch gibt es auch hier keine Probleme.

Die RAM-Adreßmultiplexer

Sowohl die IOU als auch die MMU enthalten einen Multiplexer für die RAM-Adressen. Auf dem RAM-Adreßbus werden damit pro Prozessorzyklus insgesamt 4 Teiladressen gemultiplext: die MMU liefert ROW- und COLUMN-Adressen für CPU-Zugriffe, die IOU erzeugt ROW- und COLUMN-Adressen aus dem Stand des Videoscanners, die vom momentan aktiven Bildmodus abhängig sind.

Sämtliche Multiplexvorgänge sind in Bild 5.3 zusammengefaßt. Sowohl bei der Adressierung durch die MMU als auch durch die IOU wird über RAS' = "1" und die danach fallende Flanke von RAS' die ROW-Adresse ausgegeben, danach bleibt RAS' auf "0" und die COLUMN-Adresse wird ausgegeben, gefolgt von einer fallenden Flanke von CAS'. Die Multiplexadresse der MMU wird während der letzten 14M-Periode von PHASE1 und den vier ersten 14M-Perioden von PHASE0 auf den RAM-Adreßbus gelegt, die der IOU während der letzten 14M-Periode von PHASE0 und den ersten vier 14M-Perioden von PHASE1. Diese um eine 14M-Periode vorgezogene Durchschaltung stellt sicher, daß die Adresse trotz der unvermeidlichen Laufzeiten an den RAM-Bausteinen gültig ist, wenn RAS', CAS' und Q3 während PHASE0 (für die MMU) bzw. während PHASE1 (für den Videoscanner) fallen.

Die Multiplexfunktion der MMU ist eine direkte Teilung der 16-Bit-Adresse der CPU in die ROW- und die COLUMN-Adresse mit jeweils acht Bit. Die einzige Komplikation besteht darin, daß das Adreßbit A12 (auf dem RAM-Adreßbus RA4 der COLUMN-Adresse) auf "0" gesetzt werden muß, wenn der Adreßbus eine Adresse im Bereich $DXXX enthält und der Softswitch BANK1 gesetzt ist. Durch Setzen von A12 auf "0" wird $DXXX zu $CXXX - damit haben wir den CPU-Zugriff zur *Bank1* der oberen 16k RAM.

Die Multiplexfunktionen der IOU sind weitaus komplexer: Die RAM-Adresse für die Videodaten ist nicht einfach eine gemultiplexte Form des Videoscanner-Zählstandes, sondern muß über den momentan gesetzten Bildmodus umgerechnet werden. Dazu kommt noch eine spezielle Logik, die mit den "unnatürlichen" 40 Byte pro Zeile fertigwerden muß.

Die Softswitches für den Bildschirmmodus können als Erweiterung des Videoscanners in puncto RAM-Adressierung betrachtet werden, generell stellen sie die höherwertigen Bits der RAM-Adresse.

Das Signal HIRES TIME, das für die RAM-Adressierung benutzt wird, ist ein Produkt des Videoscanner-Zählstandes und der Softswitches für den Bildschirmmodus. HIRES TIME hat durchgehend den Pegel "1", wenn der Apple sich im Modus HiRes/GRAPHICS/NOMIX befindet. Im Modus HiRes/GRAPHICS/MIXED hat HIRES TIME den Pegel "1", wenn V4 * V2 "0" ist. Über V4 * V2 = "1" wird die vom Videoscanner erzeugte Adresse so umgeschaltet, daß auf dem untersten Teil des Bildschirms vier Zeilen TEXT dargestellt werden. Wie man sich denken kann, muß die Umschaltung zwischen GRAPHICS und TEXT natürlich synchronisiert mit dem Videoscanner erfolgen.

Die in Bild 5.3 gezeichneten Eingänge PHASE2 und STORE80 sind Softswitches der IOU. Wenn PAGE2 gesetzt ist, wird die Seite 2 des jeweiligen RAM-Bereichs adressiert - es sei denn, STORE80 ist gesetzt. Dieser Softswitch blockiert PAGE2, die Bilddarstellung bleibt auf Seite 1.

Die MMU enthält einen zweiten Satz der Softswitches 80STORE, PAGE2 und HIRES und benutzt sie, um zwischen dem RAM der Hauptplatine und dem AUX-RAM hin- und herzuschalten.

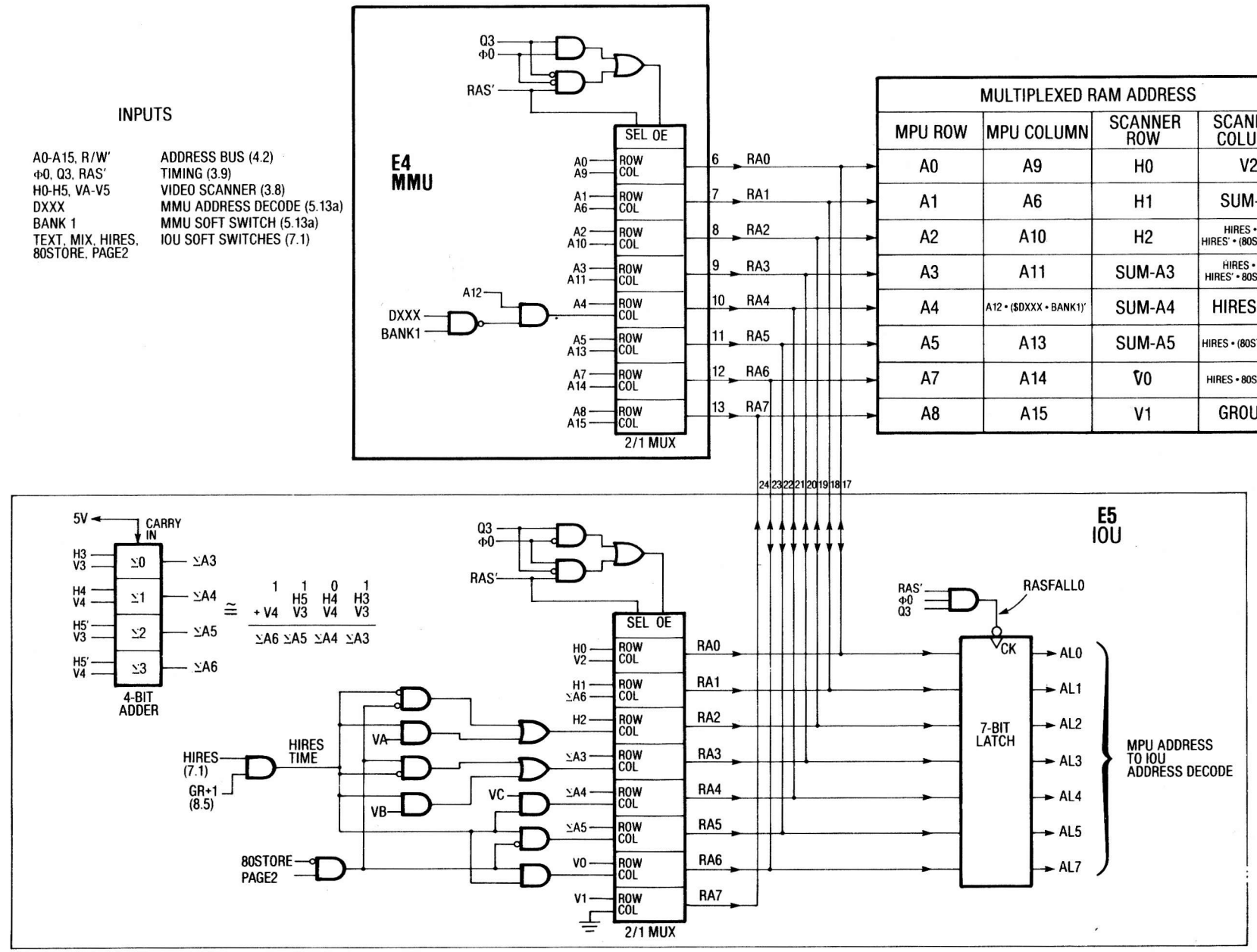
An dieser Stelle sollte noch gesagt werden, daß das *Technical Reference Manual* für den //e behauptet, die RAM-Adressierung würde über 80VID und nicht über 80STORE geschaltet. Dabei geht es hier sowieso etwas durcheinander mit den Bezeichnungen: die Termini 80VID, 80STORE und 80COL werden für *zwei* verschiedene Softswitches benutzt (W$C000/1 und W$C00C/D). Korrekterweise muß im gesamten Handbuch 80VID durch 80COL ersetzt werden. Auf den Schaltplänen muß 80VID' (IOU Pin 6 und AUX-Verbinder Kontakt 25) durch 80COL' ersetzt werden.[5] Der Softswitch, mit dem PAGE2 unwirksam gemacht wird, ist 80STORE (W$C000/1) und nicht 80COL (W$C00C/D).

Der Unterschied ist gewaltig: Wenn der RAM über 80COL adressiert werden würde, hätten wir nur eine mögliche Seite für HiRes80. Tatsächlich kann man aber durch Zurücksetzen von 80STORE und Setzen von PAGE2 an die Seite 2 heran, sowohl für TEXT bzw. LoRes-Grafik ($800-$BFF) als auch für HiRes ($4000-$5FFF) in *doppelter Auflösung*. Die Adressierung geschieht über 80STORE - wenn dieser Schalter gesetzt ist, kommt der Prozessor in den 80er Modi an die Seite 2 nicht heran.

---

5 In der Ausgabe "Juli 1985" ist das im laufenden Text geschehen, in den Schaltplänen findet sich allerdings noch VID80 - (Anm. d. Übers.).

Bild 5.3 Funktionsschema der RAM-Adreßerzeugung im Apple //e



MULTIPLEXED RAM ADDRESS			
MPU ROW	MPU COLUMN	SCANNER ROW	SCANNER COLUMN
A0	A9	H0	V2
A1	A6	H1	SUM-A6
A2	A10	H2	HIRES • VA + HIRES' • (80STR' • PG2)'
A3	A11	SUM-A3	HIRES • VB + HIRES' • 80STR' • PG2
A4	A12 • ($DXXX • BANK1)'	SUM-A4	HIRES • VC
A5	A13	SUM-A5	HIRES • (80STR' • PG2)'
A7	A14	V0	HIRES • 80STR' • PG2
A8	A15	V1	GROUND

Die Tabelle MULTIPLEXED RAM ADDRESS BUS in Bild 5.3 faßt alle möglichen Zustände des RAM-Adreßbusses zusammen und zeigt, welche RAM-Adreßeingänge mit den einzelnen Leitungen des Busses verbunden sind. Dabei fallen einige Punkte auf:

1. Die niederwertigen Bits des Videoscanners liefern die ROW-Adressen der RAM-Bausteine. Der Refresh der RAMs geschieht über den Videoscanner.
2. Die Umsetzung der CPU-Adressen über die MMU entspricht der Umsetzung der Videoscanner-Adresse über die IOU: A0 der CPU bestimmt RA0 der ROW-Adresse während eines CPU-Zugriffs, H0 des Videoscanners kontrolliert dieselbe Leitung während eines IOU-Zugriffs auf den RAM. Dasselbe gilt für A1 und H1, A2 und H2 etc.
3. Die niederwertigen Bits des Adreßbusses sind äquivalent mit der von der MMU ausgegebenen ROW-Adresse. Auf diese Weise erhält die IOU über die MMUSSS den Stand der Adreßleitungen A0..A5 und A7, wenn die CPU einen IOU-Softswitch adressiert. Die MMU liefert bei einem Zugriff der CPU auf $C0XX das Signal CXXX an den peripheren Adreßdekoder, der das Signal C0XX' erzeugt. Das einzige Adreßbit, was der IOU jetzt noch für eine vollständige Adreßdekodierung fehlt, ist A6 - und das erhält sie direkt vom Adreßbus.
4. Die RAM-Adressierung ist in den 80er Modi dieselbe wie in den 40er Modi.

Tabelle 5.1 zeigt den Zusammenhang zwischen den Leitungen des Adreßbusses und den Bits des Videoscanners. Über diese Tabelle kann jeder Bildmodus zusammen mit jedem Videoscanner-Zählerstand in eine äquivalente CPU-Adresse verwandelt werden. Der Zusammenhang funktioniert auch in umgekehrter Richtung: Ein an der errechneten CPU-Adresse gespeichertes Byte wird beim entsprechenden Stand des Videoscanners ausgelesen und erscheint auf dem Bildschirm.

Tabelle 5.1 Äquivalente Adreßbits von CPU und Videoscanner

MPU	VIDEO SCANNER	
A0	H0	
A1	H1	
A2	H2	
A3	SUM-A3	
A4	SUM-A4	
A5	SUM-A5	
A6	SUM-A6	
A7	V0	
A8	V1	
A9	V2	
A10	HIRES • VA	TEXT/LORES • (PAGE2 • 80STORE')'
A11	HIRES • VB	TEXT/LORES • PAGE2 • 80STORE'
A12	HIRES • VC	—
A13	HIRES • (PAGE2 • 80STORE')'	—
A14	HIRES • PAGE2 • 80STORE'	—
A15	—	

Die Adreßlogik des Videoscanners

Wenn der Apple nicht auf andere Weise berühmt geworden wäre, dann hätte er es wenigstens für seine Adressierungsart des Video-RAMs werden müssen - eines der größten Kreuze für Programmierer ist die Berechnung von anscheinend völlig unsystematisch angeordneten Bildschirmadressen. Zu verdanken haben wir das dem Apple II, weil der //e aus Kompatibilitätsgründen immer noch dasselbe Adressierschema benutzt.

Auch wenn die Speicherstellen des Bildschirms völlig durcheinandergewürfelt erscheinen, *gibt* es ein logisches Prinzip - dem liegt allerdings binäre Bitmanipulation zugrunde. Um das zu verstehen, müssen wir uns in Zeit um

1975 zurückversetzen und den Apple II vom Standpunkt des Designers betrachten. Wie bekommen wir den Apple II dazu, LoRes- und HiRes-Grafik sowie TEXT mit 40 Zeichen pro Zeile zu produzieren?

Moment! - *40* Zeichen? Hat Woz noch nie etwas von Computern und von Zweierpotenzen gehört? Wie wäre es mit 32, 64 oder 128 Zeichen pro Zeile?

Aber nein - vierzig Zeichen müssen es sein. Das ist genauso unsinnig wie Manuskripte mit 55 Anschlägen pro Zeile, 80-Zeichen-Bildschirme und Menschen mit ihrem Dezimalsystem.

Die Probleme fangen damit an, daß man normalerweise alle Speicherstellen der Reihe nach für die Videoausgabe adressiert. Wenn der Apple z.B. mit 32 Zeichen pro Zeile arbeiten würde, könnte man einfach H0-H5 und V0-V4 direkt mit einem 4:1-Adreßmultiplexer verbinden und den Bildspeicher damit schön ordentlich in 32 * 24 Byte aufteilen.

Mit 40 Zeichen pro Zeile funktioniert das so nicht. Auch hier könnte man zwar H0-H5 direkt als Adresse benutzen - dafür ergeben sich aber für jede Zeile 24 (= 64-40) ungenutzte Bytes. Für TEXT und LoRes summiert sich das auf 576 Byte, für HiRes auf 4608 Byte. Aber was sollen 4608 Byte, die in nicht zusammenhängenden 192 Stückchen mit je 24 Byte im Speicher verteilt sind?

Speicherverschwendung war zwar nicht zu umgehen, jedoch bei der Konstruktion des Apple durch eine geringe Hardware-Komplizierung zu mindern: anstelle der Verschwendung von 24 Byte pro Zeile werden 120 von 128 Byte genutzt. Auf drei TEXT-Zeilen bzw. drei HiRes-Fernsehzeilen werden damit 8 Byte nicht genutzt, für TEXT ergeben sich insgesamt 64, für HiRes 512 ungenutzte Bytes.

Sehen wir uns diese Hardware-Implementation einmal an: Der Bildschirm ist in Segmente mit jeweils *128 Byte* unterteilt, wie in Bild 5.4 gezeigt. Jedes Segment wird in *FIRST 40, SECOND 40, THIRD 40* und *UNUSED 8* unterteilt (dieser letzte Teil ist "Niemandsland"). Dadurch haben wir wieder eine vernünftige Einteilung des Bildschirms durch V3 und V4 des Videoscanners:

V4' V3'	-	oberes Drittel
V4' V3	-	mittleres Drittel
V4 V3'	-	unteres Drittel
V4 V3	-	VBL

Das letzte Viertel (VBL) wird nicht dargestellt, hier finden sich der obere und der untere Bildschirmrand sowie der vertikale Rücklauf des Elektronenstrahls.

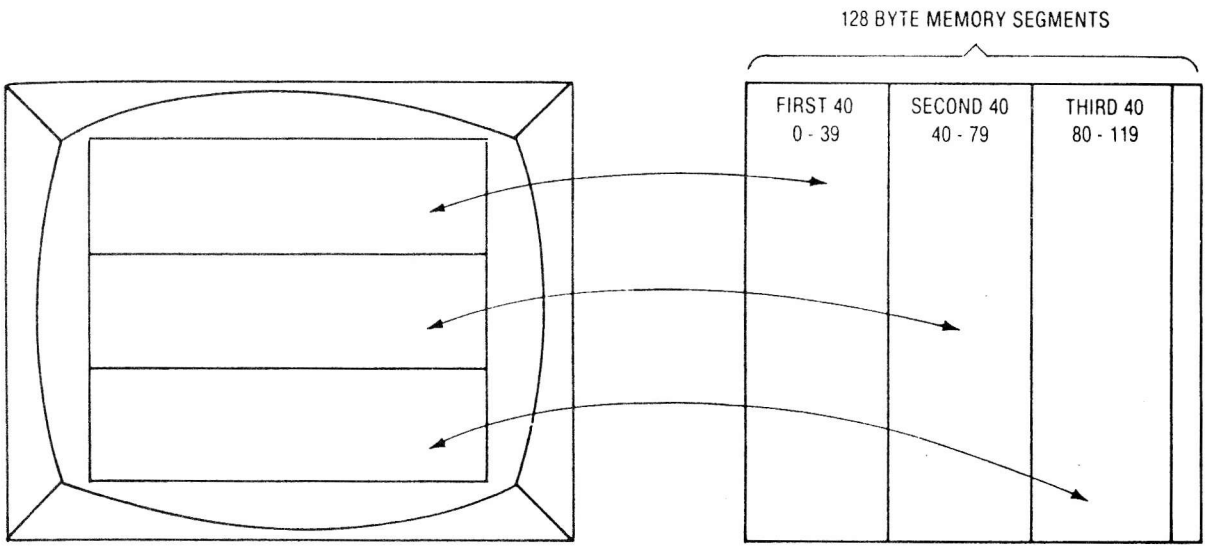
Nachdem sich das so schön aufteilt, kann man die drei (dargestellten) Zustände direkt auf drei Abschnitte mit 40 Byte in jedem 128-Byte-Segment abbilden:

Position auf dem Bildschirm	**LSB der Adresse**
Oben	0000000 - 0100111 (ersten 40)
Mitte	0101000 - 1001111 (zweiten 40)
Unten	1010000 - 1110111 (dritten 40)

Wie aus dieser Tabelle hervorgeht, beginnen alle drei 40-Byte-Abschnitte mit 000 und enden mit 111 der niederwertigsten drei Bits. Infolgedessen kann die Adressierung dieser drei Bits in FIRST 40, SECOND 40 und THIRD 40 in gleicher Weise geschehen. Jedes Byte auf einem Zeilenanfang hat die Adresse xxx x000, deshalb sind die Bits H0, H1 und H2 des Videoscanners direkt zum Adreßmultiplexer der IOU und von da aus zu den Adreßleitungen RA0, RA1 und RA2 weitergeführt.

Die nächsten vier Adreßbits unterscheiden die einzelnen 40-Byte-Abschnitte voneinander: 0000 bis 0100 entspricht FIRST 40, 0101 bis 1001 entspricht SECOND 40 und 1010 bis 1110 THIRD 40. Diese vier Bits werden durch H5-H4-H3 plus einem *Offset* erzeugt. Der Offset wird über den Zustand von V3 und V4 erzeugt und über einen 4-Bit-Addierer innerhalb der IOU zu H5-H4-H3 dazuaddiert. Das Ergebnis dieser Addition ergibt die Adreßbits A3, A4, A5 und A6, die einzelnen Bits werden im folgenden als SUM-A3, SUM-A4, SUM-A5 und SUM-A6 bezeichnet.

Bild 5.4 128-Byte-Segment aus drei 40-Byte-Abschnitten, jeder Abschnitt in einem unterschiedlichen Bildschirmteil



Mit H5-H4-H3 lassen sich insgesamt 8 verschiedene Zustände erzeugen, von denen aber nur fünf tatsächlich auf dem Bildschirm erscheinen. Die Zustände 000 bis 010 werden während des rechten Bildschirmrandes, des horizontalen Rücklaufs und des linken Bildschirmrandes durchlaufen. Die RAM-Adressierung beginnt bei 011 - hier muß das erste 40-Byte-Segment adressiert werden. Ein erster Vorschlag zur Summenbildung könnte so aussehen:

```
          H5      H4      H3
+ V4      V3      V4      V3
---------------------------
SUM-A6 SUM-A5 SUM-A4 SUM-A3
```

Daraus ergeben sich bereits die benötigten 40-Byte-Offsets 0000, 0101 und 1010. Funktionieren würde es sogar auch in dieser Form, allerdings wäre die resultierende Adressierung für den Programmierer noch komplexer als sie in der tatsächlichen Ausführung ist. Um sie wenigstens etwas einfacher zu machen, berücksichtigen wir die Tatsache, daß die Bildausgabe mit H5-H4-H3 = 011 beginnt und ziehen diesen Wert ab, damit FIRST 40 auf einer "natürlichen" Adresse (d.h. $XX00 oder $XX80) beginnt. Die dazu benötigten Offsets sind 1101, 0010 und 0111 (dezimal: -3, 2 und 7). Wir müssen also A6-A5-A4-A3 über die folgenden Werte setzen:

- H5-H4-H3 minus 0011 für FIRST 40;
- H5-H4-H3 plus 0010 für SECOND 40;
- H5-H4-H3 plus 0111 für THIRD 40.

Das wird im Apple mit dem folgenden Additionsschema elegant gelöst:

```
                          1
  H5'     H5'     H4      H3
+ V4      V3      V4      V3
---------------------------
SUM-A6  SUM-A5  SUM-A4  SUM-A3
```

In der benutzten 4-Bit-Arithmetik mit Vorzeichen entspricht H5'-H5'-H4-H3 dem Wert von H5-H4-H3 minus 0100. 0001 minus 0100 ergibt -0011 - damit haben wir den gewünschten Offset. Der Wert "1" für die Zweierkomplementbildung wird einfach über einen Carry-Eingang des 4-Bit-Addierers dazugenommen. Eine äquivalente Rechnung ohne Vorzeichen wäre:

```
   1      1      0      1
         H5     H4     H3
+ V4     V3     V4     V3
-------------------------
SUM-A6 SUM-A5 SUM-A4 SUM-A3
```

Der Scan-Prozeß für TEXT und LoRes

Die Zusammensetzung von A0-A6 haben wir im letzten Abschnitt gezeigt, sie ist für alle Videomodi dieselbe. Die Adreßbits A7, A8 und A9 entsprechen V0, V1 und V2 des Videoscanners und legen fest, welches 128-Byte-Segment ausgelesen wird.

Eine Bildschirmseite für TEXT oder LoRes besteht aus acht aufeinanderfolgenden 128-Byte-Segmenten, die über V0..V2 bzw. A7..A9 bestimmt werden.

Um eine TEXT-Zeile oder zwei LoRes-Zeilen auf einem Fernseher abzubilden, sind acht horizontale Durchläufe des Elektronenstrahls nötig - ein 40-Byte-Segment wird deshalb achtmal hintereinander durchlaufen.

Bild 5.5 Speicheradressen von TEXT und LoRes

	BASE	TOP SCREEN/ FIRST 40		MIDDLE SCREEN/ SECOND 40		BOTTOM SCREEN/ THIRD 40		UNUSED 8
	ADDRESS	LIN#	RANGE	LIN#	RANGE	LIN#	RANGE	RANGE
	$400	00	$400-$427	08	$428-$44F	16	$450-$477	$478-$47F
	1024		1024-1063		1064-1103		1104-1143	1144-1151
	$480	01	$480-$4A7	09	$4A8-$4CF	17	$4D0-$4F7	$4F8-$4FF
	1152		1152-1191		1192-1231		1232-1271	1272-1279
	$500	02	$500-$527	10	$528-$54F	18	$550-$577	$578-$57F
	1280		1280-1319		1320-1359		1360-1399	1400-1407
	$580	03	$580-$5A7	11	$5A8-$5CF	19	$5D0-$5F7	$5F8-$5FF
PAGE 1	1408		1408-1447		1448-1487		1488-1527	1528-1535
	$600	04	$600-$627	12	$628-$64F	20	$650-$677	$678-$67F
	1536		1536-1575		1576-1615		1616-1655	1656-1663
	$680	05	$680-$6A7	13	$6A8-$6CF	21	$6D0-$6F7	$6F8-$6FF
	1664		1664-1703		1704-1743		1744-1783	1784-1791
	$700	06	$700-$727	14	$728-$74F	22	$750-$777	$778-$77F
	1792		1792-1831		1832-1871		1872-1911	1912-1919
	$780	07	$780-$7A7	15	$7A8-$7CF	23	$7D0-$7F7	$7F8-$7FF
	1920		1920-1959		1960-1999		2000-2039	2040-2047
	$800	00	$800-$827	08	$828-$84F	16	$850-$877	$878-$87F
	2048		2048-2087		2088-2127		2128-2167	2168-2175
	$880	01	$880-$8A7	09	$8A8-$8CF	17	$8D0-$8F7	$8F8-$8FF
	2176		2176-2215		2216-2255		2256-2295	2296-2303
	$900	02	$900-$927	10	$928-$94F	18	$950-$977	$978-$97F
	2304		2304-2343		2344-2383		2384-2423	2424-2431
	$980	03	$980-$9A7	11	$9A8-$9CF	19	$9D0-$9F7	$9F8-$9FF
PAGE 2	2432		2432-2471		2472-2511		2512-2551	2552-2559
	$A00	04	$A00-$A27	12	$A28-$A4F	20	$A50-$A77	$A78-$A7F
	2560		2560-2599		2600-2639		2640-2679	2680-2687
	$A80	05	$A80-$AA7	13	$AA8-$ACF	21	$AD0-$AF7	$AF8-$AFF
	2688		2688-2727		2728-2767		2768-2807	2808-2815
	$B00	06	$B00-$B27	14	$B28-$B4F	22	$B50-$B77	$B78-$B7F
	2816		2816-2855		2856-2895		2896-2935	2936-2943
	$B80	07	$B80-$BA7	15	$BA8-$BCF	23	$BD0-$BF7	$BF8-$BFF
	2944		2944-2983		2984-3023		3024-3063	3064-3071

Die Ausgänge VA, VB und VC des Videoscanners durchlaufen dabei die Zustände von 000 bis 111. Der Videogenerator bekommt über VA-VC mitgeteilt, welche der acht Zeilen einer Buchstabenmatrix bzw. der vier Zeilen eines LoRes-Blocks ausgegeben werden muß - in der RAM-Adressierung spielen sie keine Rolle.

In beiden Modi haben die Äquivalente der Adreßbits A15..A12 den Wert "0". A10 ergibt sich aus (80STORE' * PAGE2)', A11 entspricht 80STROBE' * PAGE2. Die sich daraus ergebenden Adreßbereiche finden Sie am Fuß dieser Seite - wie zu erwarten, entsprechen sie den für TEXT und LoRes reservierten Speicherbereichen.

Bild 5.5 zeigt den Zusammenhang zwischen Bildschirmzeilen und entsprechenden 40-Byte-Segmenten im Detail. Letztendlich enthält dieses Bild dieselbe Information wie die entsprechende Darstellung im *Technical Reference Manual* für den //e - allerdings wurde hier eine andere Anordnung mit Schwerpunkt auf der Abfolge der einzelnen Segmente gewählt. Das Reference Manual gibt nur die Zusammenhänge zwischen TEXT-Zeichen und LoRes-Blocks wieder und geht in keiner Weise auf den logischen Aufbau der Bildschirmadressen ein - durch Bild 5.5 sollte der im vorherigen Abschnitt besprochene Stoff etwas klarer werden.

Einen Punkt haben wir bis jetzt ausgelassen, nämlich die Frage, auf welche Speicherstellen der Videoscanner zugreift, während nichts auf dem Bildschirm dargestellt wird. Diese Zeit wird durch die Sicherheitszonen zu allen Seiten des Bildschirms festgelegt, die entsprechenden Steuersignale werden innerhalb der IOU über Zustände des Videoscanners erzeugt:

- HBL ("Horizontal BLanking" = horizontale Schwarzschulter) hat für den linken und rechten Bildschirmrand sowie für die Zeit des horizontalen Strahlrücklaufs den Pegel "1", ansonsten den Pegel "0".
- VBL ("Vertical BLanking" = vertikale Schwarzschulter) hat für den oberen und unteren Bildschirmrand sowie für die Zeit des vertikalen Strahlrücklaufs den Wert "1", ansonsten den Wert "0".

Es ist über das Lesen von Videodaten möglich, von einem Programm aus zu bestimmen, wann eines der beiden Signale aktiv ist. Nützlich ist eine solche Möglichkeit bei den meisten Anwendungen, die in irgendeiner Form mit Video zu tun haben, wie Lightpens, Digitizer etc.

SCREEN MODE	BINARY	HEXADECIMAL
PAGE 1	ØØØØ Ø1XX XXXX XXXX	$Ø4ØØ-$Ø7FF
PAGE 2, 8ØSTORE	ØØØØ Ø1XX XXXX XXXX	$Ø4ØØ-$Ø7FF
PAGE 2, 8ØSTORE'	ØØØØ 1ØXX XXXX XXXX	$Ø8ØØ-$ØBFF

Die Videodaten der Hauptplatine können über eine Adresse gelesen werden, die keine Daten zurückliefert. Um den Beginn einer bestimmten Zeile herauszufinden, kann man z.B. folgendermaßen vorgehen:

1. Sämtliche Bytes eines Bildspeicherbereichs auf den Wert 00 setzen.
2. Den unbenutzten Bereich vor der gesuchten Zeile auf 00 setzen. Dieser RAM-Bereich wird vom Videoscanner adressiert, die Daten werden aus dem RAM ausgelesen und befinden sich danach auf dem Datenbus, sie werden vom Videogenerator ignoriert.
3. Die folgende Programmschleife erkennt danach den zeitlichen Anfang der gesuchten Zeile:

```
WAIT:     LDA   $C054          ; Softswitch PAGE1
          BPL   WAIT
```

Bild 5.6 zeigt die kompletten Speicherbereiche von TEXT und LoRes und unterscheidet dabei zwischen auf dem Bildschirm dargestellten und nicht dargestellten Bytes, die Anordnung ist in diesem Fall dieselbe wie in dem entsprechenden Bild im *Technical Reference Manual*. Der vor der Bildausgabe vom Videoscanner adressierte Speicherbereich steht jeweils am Beginn der entsprechenden Zeile, der unterste Teil des Bildes zeigt den unteren Bildschirmrand und die Zeit des vertikalen Strahlrücklaufs zusammen mit dem oberen Bildschirmrand. Dabei lassen sich die folgenden Gesetzmäßigkeiten feststellen:

1. Während HBL werden $18 Speicherstellen adressiert, bevor die Darstellung der eigentlichen Zeile beginnt. Die Speicheradresse, mit der die HBL-Adressierung beginnt, läßt sich über die Adresse des ersten dargestellten Bytes ("BASE") folgendermaßen berechnen:

```
10 HBL = BASE-24
20 IF INT (HBL/128) <> INT (BASE/128) THEN HBL = HBL+128
```

Der Schritt in Zeile 20 ist deshalb notwendig, weil die Adresse über eine 7-Bit-Rechnung ermittelt wird - ein Ergebnis von 128 erzeugt den Wert 0, 129 erzeugt 1 etc.

2. Die erste Adresse während HBL wird immer zweimal hintereinander angesprochen, weil H0-H5 für die ersten beiden Zustände des Videoscanners innerhalb einer Fernsehzeile auf dem Wert 00000 bleiben (Sprung von 0000000 auf 1000000, s. Kapitel 3).

3. Während VBL haben V3 und V4 beide den Wert "1". SUM-A3 bis SUM-A6 ergeben sich deshalb aus H5-H4-H3 minus 0100. Das unterscheidet sich nicht wesentlich von der Adressierung des oberen Bildschirmdrittels (H5-H4-H3 minus 0011) - die Startadressen während VBL entsprechen den HBL-Startadressen während FIRST 40 minus 8, allerdings wird die Subtraktion Modulo 128 ausgeführt ($400 minus 8 ergibt $478, nicht $3F8).

4. Die Folge horizontaler Adressen läuft ebenfalls Modulo 128, vor der Speicherstelle $400 wird nicht $3FF, sondern $47F angesprochen.

Bild 5.6 Sämtliche durch den Scan-Prozeß für TEXT und LoRes angesprochenen Speicheradressen

		HORIZONTAL BLANKING (HBL)			HORIZONTAL DISPLAY ENABLE		
	LIN NUM	PAGE 1	PAGE 2	11111111 ØØ123456789ABCDEFØ1234567	PAGE 1	PAGE 2	1111111111111111122222222 Ø123456789ABCDEFØ123456789ABCDEFØ1234567
SCREEN TOP	Ø	$468 1128	$868 2152	++++++++++####++++++++++	$4ØØ 1Ø24	$8ØØ 2Ø48	++++++++++++++++++++++++++++++++
	1	$4E8 1256	$8E8 2280	++++++++++####++++++++++	$48Ø 1152	$88Ø 2176	++++++++++++++++++++++++++++++++
	2	$568 1384	$968 2408	++++++++++####++++++++++	$5ØØ 128Ø	$9ØØ 2304	++++++++++++++++++++++++++++++++
	3	$5E8 1512	$9E8 2536	++++++++++####++++++++++	$58Ø 14Ø8	$98Ø 2432	++++++++++++++++++++++++++++++++
	4	$668 1640	$A68 2664	++++++++++####++++++++++	$6ØØ 1536	$AØØ 256Ø	++++++++++++++++++++++++++++++++
	5	$6E8 1768	$AE8 2792	++++++++++####++++++++++	$68Ø 1664	$A8Ø 2688	++++++++++++++++++++++++++++++++
	6	$768 1896	$B68 2920	++++++++++####++++++++++	$7ØØ 1792	$BØØ 2816	++++++++++++++++++++++++++++++++
	7	$7E8 2024	$BE8 3048	++++++++++####++++++++++	$78Ø 192Ø	$B8Ø 2944	++++++++++++++++++++++++++++++++
SCREEN MIDDLE	8	$41Ø 1Ø4Ø	$81Ø 2Ø64	++++++++++####++++++++++	$428 1Ø64	$828 2Ø88	++++++++++++++++++++++++++++++++
	9	$49Ø 1168	$89Ø 2192	++++++++++####++++++++++	$4A8 1192	$8A8 2216	++++++++++++++++++++++++++++++++
	1Ø	$51Ø 1296	$91Ø 232Ø	++++++++++####++++++++++	$528 132Ø	$928 2344	++++++++++++++++++++++++++++++++
	11	$59Ø 1424	$99Ø 2448	++++++++++####++++++++++	$5A8 1448	$9A8 2472	++++++++++++++++++++++++++++++++
	12	$61Ø 1552	$A1Ø 2576	++++++++++####++++++++++	$628 1576	$A28 26ØØ	++++++++++++++++++++++++++++++++
	13	$69Ø 168Ø	$A9Ø 27Ø4	++++++++++####++++++++++	$6A8 17Ø4	$AA8 2728	++++++++++++++++++++++++++++++++
	14	$71Ø 18Ø8	$B1Ø 2832	++++++++++####++++++++++	$728 1832	$B28 2856	++++++++++++++++++++++++++++++++
	15	$79Ø 1936	$B9Ø 296Ø	++++++++++####++++++++++	$7A8 196Ø	$BA8 2984	++++++++++++++++++++++++++++++++
SCREEN BOTTOM	16	$438 1Ø8Ø	$838 2Ø4	++++++++++####++++++++++	$45Ø 11Ø4	$85Ø 2128	++++++++++++++++++++++++++++++++
	17	$4B8 12Ø8	$8B8 2232	++++++++++####++++++++++	$4DØ 1232	$8DØ 2256	++++++++++++++++++++++++++++++++
	18	$538 1336	$938 236Ø	++++++++++####++++++++++	$55Ø 136Ø	$95Ø 2384	++++++++++++++++++++++++++++++++
	19	$5B8 1464	$9B8 2488	++++++++++####++++++++++	$5DØ 1488	$9DØ 2512	++++++++++++++++++++++++++++++++
	2Ø	$638 1592	$A38 2616	++++++++++####++++++++++	$65Ø 1616	$A5Ø 264Ø	++++++++++++++++++++++++++++++++
	21	$6B8 172Ø	$AB8 2744	++++++++++####++++++++++	$6DØ 1744	$ADØ 2768	++++++++++++++++++++++++++++++++
	22	$738 1848	$B38 2872	++++++++++####++++++++++	$75Ø 1872	$B5Ø 2896	++++++++++++++++++++++++++++++++
	23	$7B8 1976	$BB8 3ØØØ	++++++++++####++++++++++	$7DØ 2ØØØ	$BDØ 3Ø24	++++++++++++++++++++++++++++++++
VERTICAL BLANKING	24	$46Ø 112Ø	$86Ø 2144	++++++++++####++++++++++	$478 1144	$878 2168	++++++++++++++++++++++++++++++++
	25	$4EØ 1248	$8EØ 2272	++++++++++####++++++++++	$4F8 1272	$8F8 2296	++++++++++++++++++++++++++++++++
	26	$56Ø 1376	$96Ø 24ØØ	++++++++++####++++++++++	$578 14ØØ	$978 2424	++++++++++++++++++++++++++++++++
	27	$5EØ 15Ø4	$9EØ 2528	++++++++++####++++++++++	$5F8 1528	$9F8 2552	++++++++++++++++++++++++++++++++
	28	$66Ø 1632	$A6Ø 2656	++++++++++################	$678 1656	$A78 268Ø	################################
	29	$6EØ 176Ø	$AEØ 2784	++++++++++####++++++++++	$6F8 1784	$AF8 28Ø8	++++++++++++++++++++++++++++++++
	3Ø	$76Ø 1888	$B6Ø 2912	++++++++++####++++++++++	$778 1912	$B78 2936	++++++++++++++++++++++++++++++++
	31	$7EØ 2Ø16	$BEØ 3Ø4Ø	++++++++++####++++++++++	$7F8 2Ø4Ø	$BF8 3Ø64	++++++++++++++++++++++++++++++++

The last row of memory is scanned 14 consecutive times. All other rows are scanned 8 consecutive times.

HORIZONTAL SYNC

VERTICAL SYNC

Der Scan-Prozeß für HiRes

Wie aus Bild 5.1 bereits hervorgeht, ist die Erzeugung der Adreßbits A0-A9 und A15 dieselbe wie in den Modi TEXT und LoRes. Die unterschiedliche Erzeugung der Adreßbits A10-A14 reflektiert die Tatsache, daß der Bildspeicherbereich für HiRes die achtfache Größe hat und außerdem mit anderen Adressen beginnt.

A13 entspricht (PAGE2 * 80STORE')', A14 entspricht PAGE2 * 80STORE'. Dadurch ergibt sich als Basis für die HiRes-Seite 1 die Adresse $2000, für die Seite 2 die Startadresse $4000. 80STORE hat denselben Effekt wie in den Modi TEXT und LoRes - es sperrt die Umschaltung auf Seite 2 durch PAGE2.

VA, VB und VC erzeugen A10, A11 und A12. Hier liegt der große Unterschied zu TEXT und LoRes: ein 40-Byte-Segment in HiRes enthält nicht mehr die Informationsmenge für acht aufeinanderfolgende Fernsehzeilen, sondern nur noch für eine einzige. Daraus folgt, daß für jede Fernsehzeile ein anderes 40-Byte-Segment adressiert werden

muß. Anstatt acht aufeinanderfolgende ROM-Adressen anzusprechen, die die Punktmatrix für ein Zeichen enthalten, erzeugen VA, VB und VC deshalb im Modus HiRes jeweils eine neue Segmentadresse im Video-RAM.

Eine Bildschirmseite für TEXT/LoRes umfaßt $400 Byte, mit PAGE2 werden deshalb die Adreßbits A10 und A11 umgeschaltet, die sich ergebenden Basisadressen liegen damit $400 Byte auseinander.

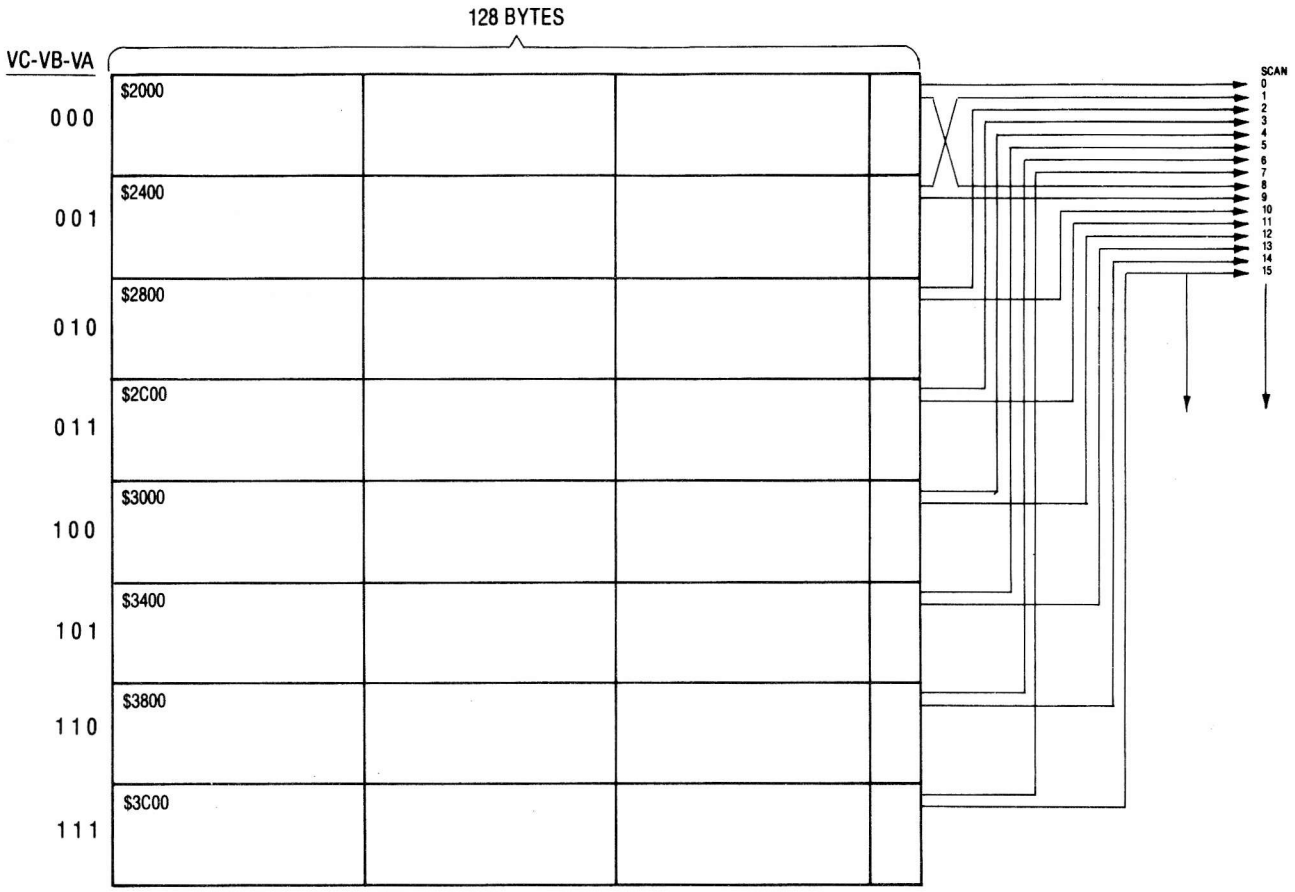
Eine Bildschirmseite für HiRes umfaßt dagegen $2000 Byte - hier müssen über PAGE2 die Adreßbits A13 und A14 umgeschaltet werden, damit die Startadressen der beiden Seiten $2000 Byte auseinanderliegen.

Die Designer haben sich leider einen zusätzlichen Baustein gespart: V0, V1 und V2 werden unverändert für die Adreßbits A5-A7 verwendet. Daraus ergibt sich die Merkwürdigkeit, daß VA-VC höherwertige Adreßbits setzen als die eigentlich für die Zeilennummern vorgesehenen Bits V0-V2: auf dem Bildschirm direkt untereinanderliegende Zeilen liegen nicht 128 Byte (d.h. über A7-A8-A9 bestimmt), sondern 1024 Byte (über A10-A11-A12 bestimmt) auseinander.

Auch das haben wir dem Apple II zu verdanken, bei dem ein weiterer Chip zur Umschaltung der Wertigkeit von V0-V2 nicht mehr auf die Platine gepaßt hat.

Um sich den Aufbau einer HiRes-Seite zu verdeutlichen, kann man sich diese Seite als acht hintereinanderliegende TEXT-Seiten vorstellen (s. Bild 5.7). VA-VC bestimmen dabei, welcher der acht Teile adressiert wird. Acht untereinanderliegende Bildschirmzeilen (also eine "TEXT-Zeile") ergeben sich daraus, daß aus jedem der acht Bereiche 64 Byte ausgelesen und 40 davon auf dem Schirm dargestellt werden. Genauso wie in TEXT/LoRes setzen sich oberes, mittleres und unteres Drittel des Schirms aus FIRST 40, SECOND 40 und THIRD 40 einer Gruppe aus 128 aufeinanderfolgenden Bytes zusammen.

Bild 5.7 Die Speicherbereiche des HiRes-Bildschirms (Seite 1)



Die Anordnung der einzelnen Speicherstellen und die Abfolge der Zeilen auf dem HiRes-Bildschirm verdeutlicht das folgende Programm, das Sie einmal ausprobieren sollten:

```
10 HGR: POKE -16302,0: POKE -16372,0: REM HiRes40, NOMIX
20 FOR A = 8192 TO 16383
30 POKE A,255
40 FOR B = 1 TO 100: NEXT: REM Verzögerung
50 NEXT A: GOTO 10
```

Dieses Programm füllt den Video-RAM des HiRes-Bildschirms in aufsteigender Reihenfolge der Adressen und langsam genug, daß man ihm bei der Arbeit zusehen kann.

Bild 5.8 gibt die Speicherbelegung durch eine HiRes-Seite wieder und betont dabei den Aufbau der 128-Byte-Segmente aus den Teilen FIRST, SECOND und THIRD sowie den jeweils acht unbenutzten Bytes. Es ist in derselben Weise wie Bild 5.5 aufgebaut und verdeutlicht so die Gleichheit der Adreßfolgen zwischen TEXT/LoRes und dem HiRes-Bildschirm. Bild 5.9 gibt den vollständigen Aufbau des Video-RAMs inklusive nicht ausgegebener Bytes wieder und entspricht in seinem Aufbau Bild 5.6. Durch die vier aufeinanderfolgenden "#" in jeder Zeile wird das horizontale SYNC-Signal angezeigt, die langen Ketten von "#" in den Zeilen 224 bis 227 stehen für das vertikale SYNC-Signal, nach dem ein neues Bild beginnt.[6] Die Bilderzeugung des Apple //e ist vollkommen identisch[7] mit der RFI-Version des Apple II, die Bilder 5.4 bis 5.9 gelten damit für beide Computer.

Die während der Schwarzperioden erzeugten Adressen des Videoscanners sind denen des Modus TEXT/LoRes sehr ähnlich. Während HBL werden $24 Byte angesprochen, die erste angesprochene Adresse liegt $24 Byte unterhalb der Adresse des ersten tatsächlich dargestellten Bytes der Zeile, wobei wieder Modulo 128 gerechnet wird. Der während VBL adressierte Bereich entspricht dem oberen Drittel des Bildschirms minus 8 Byte. Die Adressierung der Zeilen 256 bis 261 ist identisch mit der Adressierung der Zeilen 250 bis 255, d.h. diese sechs Bereiche mit jeweils 64 Byte werden pro Bilddurchlauf zweimal angesprochen. Tabelle 5.2 faßt die angesprochenen Speicherbereiche zusammen.

Der Scan-Prozeß im Modus MIXED

HiRes-Grafik gemischt mit TEXT stellt einen Spezialfall für die RAM-Adressierung des Videoscanners dar, weil hierbei zwischen der Adressierung des TEXT/LoRes-Bereichs und des HiRes-Bereichs umgeschaltet werden muß, um die unteren vier TEXT-Zeilen des Bildschirms darzustellen. Dieses Problem ergibt sich bei der Mischung von LoRes mit TEXT nicht, weil hier die verwendeten Speicherbereiche und ihre Adressierung identisch sind.

Der zur Umschaltung von HiRes auf TEXT benutzte Term HIRES TIME wird nicht direkt durch den Softswitch $C056/7 (LORES/HIRES) gesetzt, sondern von der IOU für den Zeitraum erzeugt, in dem tatsächlich HiRes-Grafik ausgegeben wird.

Wenn V4 * V2 "1" wird, dann setzt die IOU HIRES TIME einen Videoscanner-Zyklus (d.h. auf die steigende Flanke von RAS' während PHASE1) danach auf "0". V2 * V4 = "1" identifiziert die letzten vier Zeilen des TEXT-Bildschirms bzw. die letzten 32 aktiven Zeilen des HiRes-Bildschirms. Wenn V2 * V4 den Wert "0" ergibt, wird HIRES TIME einen Videoscanner-Zyklus danach wieder auf "1" gesetzt. Das ist nach Zeile 24 bzw. nach Zeile 192 der Fall, die entsprechenden Signalverläufe finden Sie in den Bildern 8.5 und 8.17.

Die Umschaltung der Adressen des Video-ROMs ist im Abschnitt "Umschaltungen im Modus MIXED" in Kapitel 8 besprochen.

Um es genau zu nehmen, ist V2 * V4 nicht nur für die Zeilen 160 bis 191, sondern auch für die Zeilen 224 bis 261 "wahr", im zweiten Fall wird HIRES TIME ebenfalls umgeschaltet. Diese Umschaltung bewirkt, daß im Modus MIXED der erste Teil der vertikalen Schwarzperiode als "HiRes" und der zweite Teil als "TEXT" ausgeführt wird, bevor mit Zeile 0 V2 * V4 wieder "0" ergibt. Da aber während der ganzen Zeit VBL aktiv ist, bekommt man davon auf dem Bildschirm nichts zu sehen - wichtig ist diese Information nur, wenn es auf die während der Schwarzperioden adressierten Speicherstellen ankommt. Für den Modus MIXED zusammen mit HiRes ergibt sich folgendes:

---

6 Bild 5.9 gilt für die amerikanische Version mit 262 Fernsehzeilen. Für die PAL-Version kommen vor und nach dem vertikalen SYNC-Impuls noch jeweils 25 Leerzeilen hinzu, wobei sich die während des vertikalen SYNC angesprochenen Speicheradressen entsprechend verschieben.

7 Das trifft nur für die amerikanische RFI-Version zu - mir ist es jedenfalls nie gelungen, auf einem PAL-Apple II Farben zu erzeugen.

Bild 5.8 Dargestellte 128-Byte-Gruppen in HiRes

		TOP SCREEN/ FIRST 40		MIDDLE SCREEN/ SECOND 40		BOTTOM SCREEN/ THIRD 40		
PAGE 1	PAGE 2	LIN#	PAGE 1 RANGE	LIN#	PAGE 1 RANGE	LIN#	PAGE 1 RANGE	UNUSED 8 PAGE 1 RANGE
$2000 8192	$4000 16384	000	$2000-$2027	064	$2028-$204F	128	$2050-$2077	$2078-$207F
$2400 9216	$4400 17408	001	$2400-$2427	065	$2428-$244F	129	$2450-$2477	$2478-$247F
$2800 10240	$4800 18432	002	$2800-$2827	066	$2828-$284F	130	$2850-$2877	$2878-$287F
$2C00 11264	$4C00 19456	003	$2C00-$2C27	067	$2C28-$2C4F	131	$2C50-$2C77	$2C78-$2C7F
$3000 12288	$5000 20480	004	$3000-$3027	068	$3028-$304F	132	$3050-$3077	$3078-$307F
$3400 13312	$5400 21504	005	$3400-$3427	069	$3428-$344F	133	$3450-$3477	$3478-$347F
$3800 14336	$5800 22528	006	$3800-$3827	070	$3828-$384F	134	$3850-$3877	$3878-$387F
$3C00 15360	$5C00 23552	007	$3C00-$3C27	071	$3C28-$3C4F	135	$3C50-$3C77	$3C78-$3C7F
$2080 8320	$4080 16512	008	$2080-$20A7	072	$20A8-$20CF	136	$20D0-$20F7	$20F8-$20FF
$2480 9344	$4480 17536	009	$2480-$24A7	073	$24A8-$24CF	137	$24D0-$24F7	$24F8-$24FF
$2880 10368	$4880 18560	010	$2880-$28A7	074	$28A8-$28CF	138	$28D0-$28F7	$28F8-$28FF
$2C80 11392	$4C80 19584	011	$2C80-$2CA7	075	$2CA8-$2CCF	139	$2CD0-$2CF7	$2CF8-$2CFF
$3080 12416	$5080 20608	012	$3080-$30A7	076	$30A8-$30CF	140	$30D0-$30F7	$30F8-$30FF
$3480 13440	$5480 21632	013	$3480-$34A7	077	$34A8-$34CF	141	$34D0-$34F7	$34F8-$34FF
$3880 14464	$5880 22656	014	$3880-$38A7	078	$38A8-$38CF	142	$38D0-$38F7	$38F8-$38FF
$3C80 15488	$5C80 23680	015	$3C80-$3CA7	079	$3CA8-$3CCF	143	$3CD0-$3CF7	$3CF8-$3CFF
$2100 8448	$4100 16640	016	$2100-$2127	080	$2128-$214F	144	$2150-$2177	$2178-$217F
$2500 9472	$4500 17664	017	$2500-$2527	081	$2528-$254F	145	$2550-$2577	$2578-$257F
$2900 10496	$4900 18688	018	$2900-$2927	082	$2928-$294F	146	$2950-$2977	$2978-$297F
$2D00 11520	$4D00 19712	019	$2D00-$2D27	083	$2D28-$2D4F	147	$2D50-$2D77	$2D78-$2D7F
$3100 12544	$5100 20736	020	$3100-$3127	084	$3128-$314F	148	$3150-$3177	$3178-$317F
$3500 13568	$5500 21760	021	$3500-$3527	085	$3528-$354F	149	$3550-$3577	$3578-$357F
$3900 14592	$5900 22784	022	$3900-$3927	086	$3928-$394F	150	$3950-$3977	$3978-$397F
$3D00 15616	$5D00 23808	023	$3D00-$3D27	087	$3D28-$3D4F	151	$3D50-$3D77	$3D78-$3D7F
$2180 8576	$4180 16768	024	$2180-$21A7	088	$21A8-$21CF	152	$21D0-$21F7	$21F8-$21FF
$2580 9600	$4580 17792	025	$2580-$25A7	089	$25A8-$25CF	153	$25D0-$25F7	$25F8-$25FF
$2980 10624	$4980 18816	026	$2980-$29A7	090	$29A8-$29CF	154	$29D0-$29F7	$29F8-$29FF
$2D80 11648	$4D80 19840	027	$2D80-$2DA7	091	$2DA8-$2DCF	155	$2DD0-$2DF7	$2DF8-$2DFF
$3180 12672	$5180 20864	028	$3180-$31A7	092	$31A8-$31CF	156	$31D0-$31F7	$31F8-$31FF
$3580 13696	$5580 21888	029	$3580-$35A7	093	$35A8-$35CF	157	$35D0-$35F7	$35F8-$35FF
$3980 14720	$5980 22912	030	$3980-$39A7	094	$39A8-$39CF	158	$39D0-$39F7	$39F8-$39FF
$3D80 15744	$5D80 23936	031	$3D80-$3DA7	095	$3DA8-$3DCF	159	$3DD0-$3DF7	$3DF8-$3DFF
$2200 8704	$4200 16896	032	$2200-$2227	096	$2228-$224F	160	$2250-$2277	$2278-$227F
$2600 9728	$4600 17920	033	$2600-$2627	097	$2628-$264F	161	$2650-$2677	$2678-$267F
$2A00 10752	$4A00 18944	034	$2A00-$2A27	098	$2A28-$2A4F	162	$2A50-$2A77	$2A78-$2A7F
$2E00 11776	$4E00 19968	035	$2E00-$2E27	099	$2E28-$2E4F	163	$2E50-$2E77	$2E78-$2E7F
$3200 12800	$5200 20992	036	$3200-$3227	100	$3228-$324F	164	$3250-$3277	$3278-$327F
$3600 13824	$5600 22016	037	$3600-$3627	101	$3628-$364F	165	$3650-$3677	$3678-$367F
$3A00 14848	$5A00 23040	038	$3A00-$3A27	102	$3A28-$3A4F	166	$3A50-$3A77	$3A78-$3A7F
$3E00 15872	$5E00 24064	039	$3E00-$3E27	103	$3E28-$3E4F	167	$3E50-$3E77	$3E78-$3E7F
$2280 8832	$4280 17024	040	$2280-$22A7	104	$22A8-$22CF	168	$22D0-$22F7	$22F8-$22FF
$2680 9856	$4680 18048	041	$2680-$26A7	105	$26A8-$26CF	169	$26D0-$26F7	$26F8-$26FF
$2A80 10880	$4A80 19072	042	$2A80-$2AA7	106	$2AA8-$2ACF	170	$2AD0-$2AF7	$2AF8-$2AFF
$2E80 11904	$4E80 20096	043	$2E80-$2EA7	107	$2EA8-$2ECF	171	$2ED0-$2EF7	$2EF8-$2EFF
$3280 12928	$5280 21120	044	$3280-$32A7	108	$32A8-$32CF	172	$32D0-$32F7	$32F8-$32FF
$3680 13952	$5680 22144	045	$3680-$36A7	109	$36A8-$36CF	173	$36D0-$36F7	$36F8-$36FF
$3A80 14976	$5A80 23168	046	$3A80-$3AA7	110	$3AA8-$3ACF	174	$3AD0-$3AF7	$3AF8-$3AFF
$3E80 16000	$5E80 24192	047	$3E80-$3EA7	111	$3EA8-$3ECF	175	$3ED0-$3EF7	$3EF8-$3EFF
$2300 8960	$4300 17152	048	$2300-$2327	112	$2328-$234F	176	$2350-$2377	$2378-$237F
$2700 9984	$4700 18176	049	$2700-$2727	113	$2728-$274F	177	$2750-$2777	$2778-$277F
$2B00 11008	$4B00 19200	050	$2B00-$2B27	114	$2B28-$2B4F	178	$2B50-$2B77	$2B78-$2B7F
$2F00 12032	$4F00 20224	051	$2F00-$2F27	115	$2F28-$2F4F	179	$2F50-$2F77	$2F78-$2F7F
$3300 13056	$5300 21248	052	$3300-$3327	116	$3328-$334F	180	$3350-$3377	$3378-$337F
$3700 14080	$5700 22272	053	$3700-$3727	117	$3728-$374F	181	$3750-$3777	$3778-$377F
$3B00 15104	$5B00 23296	054	$3B00-$3B27	118	$3B28-$3B4F	182	$3B50-$3B77	$3B78-$3B7F
$3F00 16128	$5F00 24320	055	$3F00-$3F27	119	$3F28-$3F4F	183	$3F50-$3F77	$3F78-$3F7F
$2380 9088	$4380 17280	056	$2380-$23A7	120	$23A8-$23CF	184	$23D0-$23F7	$23F8-$23FF
$2780 10112	$4780 18304	057	$2780-$27A7	121	$27A8-$27CF	185	$27D0-$27F7	$27F8-$27FF
$2B80 11136	$4B80 19328	058	$2B80-$2BA7	122	$2BA8-$2BCF	186	$2BD0-$2BF7	$2BF8-$2BFF
$2F80 12160	$4F80 20352	059	$2F80-$2FA7	123	$2FA8-$2FCF	187	$2FD0-$2FF7	$2FF8-$2FFF
$3380 13184	$5380 21376	060	$3380-$33A7	124	$33A8-$33CF	188	$33D0-$33F7	$33F8-$33FF
$3780 14208	$5780 22400	061	$3780-$37A7	125	$37A8-$37CF	189	$37D0-$37F7	$37F8-$37FF
$3B80 15232	$5B80 23424	062	$3B80-$3BA7	126	$3BA8-$3BCF	190	$3BD0-$3BF7	$3BF8-$3BFF
$3F80 16256	$5F80 24448	063	$3F80-$3FA7	127	$3FA8-$3FCF	191	$3FD0-$3FF7	$3FF8-$3FFF

Bild 5.9a Sämtliche durch den Scan-Prozeß von HiRes angesprochenen Speicheradressen (NTSC)

	SCREEN TOP					
	HORIZONTAL BLANKING (HBL)			HORIZONTAL DISPLAY ENABLE		
LINE NUM	PAGE 1	PAGE 2	11111111 Ø0123456789ABCDEFØ1234567	PAGE 1	PAGE 2	1111111111111111122222222 Ø123456789ABCDEFØ123456789ABCDEFØ1234567
Ø	$2Ø68 8296	$4Ø68 16488	++++++++####++++++++++	$2ØØØ 8192	$4ØØØ 16384	++++++++++++++++++++++++++++++++
1	$2468 9320	$4468 17512	++++++++####++++++++++	$24ØØ 9216	$44ØØ 17408	++++++++++++++++++++++++++++++++
2	$2868 1Ø344	$4868 18536	++++++++####++++++++++	$28ØØ 1Ø24Ø	$48ØØ 18432	++++++++++++++++++++++++++++++++
3	$2C68 11368	$4C68 19560	++++++++####++++++++++	$2CØØ 11264	$4CØØ 19456	++++++++++++++++++++++++++++++++
4	$3Ø68 12392	$5Ø68 2Ø584	++++++++####++++++++++	$3ØØØ 12288	$5ØØØ 2Ø48Ø	++++++++++++++++++++++++++++++++
5	$3468 13416	$5468 21608	++++++++####++++++++++	$34ØØ 13312	$54ØØ 21504	++++++++++++++++++++++++++++++++
6	$3868 14440	$5868 22632	++++++++####++++++++++	$38ØØ 14336	$58ØØ 22528	++++++++++++++++++++++++++++++++
7	$3C68 15464	$5C68 23656	++++++++####++++++++++	$3CØØ 15360	$5CØØ 23552	++++++++++++++++++++++++++++++++
8	$2ØE8 8424	$4ØE8 16616	++++++++####++++++++++	$2Ø8Ø 8320	$4Ø8Ø 16512	++++++++++++++++++++++++++++++++
9	$24E8 9448	$44E8 17640	++++++++####++++++++++	$248Ø 9344	$448Ø 17536	++++++++++++++++++++++++++++++++
1Ø	$28E8 1Ø472	$48E8 18664	++++++++####++++++++++	$288Ø 1Ø368	$488Ø 18560	++++++++++++++++++++++++++++++++
11	$2CE8 11496	$4CE8 19688	++++++++####++++++++++	$2C8Ø 11392	$4C8Ø 19584	++++++++++++++++++++++++++++++++
12	$3ØE8 12520	$5ØE8 2Ø712	++++++++####++++++++++	$3Ø8Ø 12416	$5Ø8Ø 2Ø608	++++++++++++++++++++++++++++++++
13	$34E8 13544	$54E8 21736	++++++++####++++++++++	$348Ø 13440	$548Ø 21632	++++++++++++++++++++++++++++++++
14	$38E8 14568	$58E8 22760	++++++++####++++++++++	$388Ø 14464	$588Ø 22656	++++++++++++++++++++++++++++++++
15	$3CE8 15592	$5CE8 23784	++++++++####++++++++++	$3C8Ø 15488	$5C8Ø 23680	++++++++++++++++++++++++++++++++
16	$2168 8552	$4168 16744	++++++++####++++++++++	$21ØØ 8448	$41ØØ 16640	++++++++++++++++++++++++++++++++
17	$2568 9576	$4568 17768	++++++++####++++++++++	$25ØØ 9472	$45ØØ 17664	++++++++++++++++++++++++++++++++
18	$2968 1Ø6ØØ	$4968 18792	++++++++####++++++++++	$29ØØ 1Ø496	$49ØØ 18688	++++++++++++++++++++++++++++++++
19	$2D68 11624	$4D68 19816	++++++++####++++++++++	$2DØØ 11520	$4DØØ 19712	++++++++++++++++++++++++++++++++
2Ø	$3168 12648	$5168 2Ø84Ø	++++++++####++++++++++	$31ØØ 12544	$51ØØ 2Ø736	++++++++++++++++++++++++++++++++
21	$3568 13672	$5568 21864	++++++++####++++++++++	$35ØØ 13568	$55ØØ 21760	++++++++++++++++++++++++++++++++
22	$3968 14696	$5968 22888	++++++++####++++++++++	$39ØØ 14592	$59ØØ 22784	++++++++++++++++++++++++++++++++
23	$3D68 1572Ø	$5D68 23912	++++++++####++++++++++	$3DØØ 15616	$5DØØ 23808	++++++++++++++++++++++++++++++++
24	$21E8 8680	$41E8 16872	++++++++####++++++++++	$2180 8576	$4180 16768	++++++++++++++++++++++++++++++++
25	$25E8 9704	$45E8 17896	++++++++####++++++++++	$2580 9600	$4580 17792	++++++++++++++++++++++++++++++++
26	$29E8 1Ø728	$49E8 18920	++++++++####++++++++++	$2980 1Ø624	$4980 18816	++++++++++++++++++++++++++++++++
27	$2DE8 11752	$4DE8 19944	++++++++####++++++++++	$2D80 11648	$4D80 19840	++++++++++++++++++++++++++++++++
28	$31E8 12776	$51E8 2Ø968	++++++++####++++++++++	$3180 12672	$5180 2Ø864	++++++++++++++++++++++++++++++++
29	$35E8 13800	$55E8 21992	++++++++####++++++++++	$3580 13696	$5580 21888	++++++++++++++++++++++++++++++++
3Ø	$39E8 14824	$59E8 23016	++++++++####++++++++++	$3980 14720	$5980 22912	++++++++++++++++++++++++++++++++
31	$3DE8 15848	$5DE8 24040	++++++++####++++++++++	$3D80 15744	$5D80 23936	++++++++++++++++++++++++++++++++
32	$2268 8808	$4268 17ØØØ	++++++++####++++++++++	$22ØØ 8704	$42ØØ 16896	++++++++++++++++++++++++++++++++
33	$2668 9832	$4668 18Ø24	++++++++####++++++++++	$26ØØ 9728	$46ØØ 17Ø2Ø	++++++++++++++++++++++++++++++++
34	$2A68 1Ø856	$4A68 19Ø48	++++++++####++++++++++	$2AØØ 1Ø752	$4AØØ 18944	++++++++++++++++++++++++++++++++
35	$2E68 11880	$4E68 2ØØ72	++++++++####++++++++++	$2EØØ 11776	$4EØØ 19968	++++++++++++++++++++++++++++++++
36	$3268 12904	$5268 21Ø96	++++++++####++++++++++	$32ØØ 12800	$52ØØ 2Ø992	++++++++++++++++++++++++++++++++
37	$3668 13928	$5668 22120	++++++++####++++++++++	$36ØØ 13824	$56ØØ 22Ø16	++++++++++++++++++++++++++++++++
38	$3A68 14952	$5A68 23144	++++++++####++++++++++	$3AØØ 14848	$5AØØ 23Ø4Ø	++++++++++++++++++++++++++++++++
39	$3E68 15976	$5E68 24168	++++++++####++++++++++	$3EØØ 15872	$5EØØ 24Ø64	++++++++++++++++++++++++++++++++
4Ø	$22E8 8936	$42E8 17128	++++++++####++++++++++	$2280 8832	$4280 17024	++++++++++++++++++++++++++++++++
41	$26E8 9960	$46E8 18152	++++++++####++++++++++	$2680 9856	$4680 18048	++++++++++++++++++++++++++++++++
42	$2AE8 1Ø984	$4AE8 19176	++++++++####++++++++++	$2A8Ø 1Ø88Ø	$4A8Ø 19072	++++++++++++++++++++++++++++++++
43	$2EE8 12ØØ8	$4EE8 2Ø2ØØ	++++++++####++++++++++	$2E8Ø 11904	$4E8Ø 2ØØ96	++++++++++++++++++++++++++++++++
44	$32E8 13032	$52E8 21224	++++++++####++++++++++	$3280 12928	$5280 21120	++++++++++++++++++++++++++++++++
45	$36E8 14056	$56E8 22248	++++++++####++++++++++	$3680 13952	$5680 22144	++++++++++++++++++++++++++++++++
46	$3AE8 15Ø8Ø	$5AE8 23272	++++++++####++++++++++	$3A8Ø 14976	$5A8Ø 23168	++++++++++++++++++++++++++++++++
47	$3EE8 161Ø4	$5EE8 24296	++++++++####++++++++++	$3E8Ø 16ØØØ	$5E8Ø 24192	++++++++++++++++++++++++++++++++
48	$2368 9Ø64	$4368 17256	++++++++####++++++++++	$23ØØ 896Ø	$43ØØ 17152	++++++++++++++++++++++++++++++++
49	$2768 1ØØ88	$4768 18280	++++++++####++++++++++	$27ØØ 9984	$47ØØ 18176	++++++++++++++++++++++++++++++++
5Ø	$2B68 11112	$4B68 19304	++++++++####++++++++++	$2BØØ 11ØØ8	$4BØØ 19200	++++++++++++++++++++++++++++++++
51	$2F68 12136	$4F68 2Ø328	++++++++####++++++++++	$2FØØ 12Ø32	$4FØØ 2Ø224	++++++++++++++++++++++++++++++++
52	$3368 131ØØ	$5368 21352	++++++++####++++++++++	$33ØØ 13Ø56	$53ØØ 21248	++++++++++++++++++++++++++++++++
53	$3768 14184	$5768 22376	++++++++####++++++++++	$37ØØ 14Ø8Ø	$57ØØ 22272	++++++++++++++++++++++++++++++++
54	$3B68 152Ø8	$5B68 234ØØ	++++++++####++++++++++	$3BØØ 151Ø4	$5BØØ 23296	++++++++++++++++++++++++++++++++
55	$3F68 16232	$5F68 24424	++++++++####++++++++++	$3FØØ 16128	$5FØØ 24320	++++++++++++++++++++++++++++++++
56	$23E8 9192	$43E8 17384	++++++++####++++++++++	$2380 9Ø88	$4380 17280	++++++++++++++++++++++++++++++++
57	$27E8 1Ø216	$47E8 18408	++++++++####++++++++++	$2780 1Ø112	$4780 18304	++++++++++++++++++++++++++++++++
58	$2BE8 11240	$4BE8 19432	++++++++####++++++++++	$2B8Ø 11136	$4B8Ø 19328	++++++++++++++++++++++++++++++++
59	$2FE8 12264	$4FE8 2Ø456	++++++++####++++++++++	$2F8Ø 12160	$4F8Ø 2Ø352	++++++++++++++++++++++++++++++++
6Ø	$33E8 13288	$53E8 21480	++++++++####++++++++++	$338Ø 13184	$538Ø 21376	++++++++++++++++++++++++++++++++
61	$37E8 14312	$57E8 22504	++++++++####++++++++++	$378Ø 142Ø8	$578Ø 224ØØ	++++++++++++++++++++++++++++++++
62	$3BE8 15336	$5BE8 23528	++++++++####++++++++++	$3B8Ø 15232	$5B8Ø 23424	++++++++++++++++++++++++++++++++
63	$3FE8 16360	$5FE8 24552	++++++++####++++++++++	$3F8Ø 16256	$5F8Ø 24448	++++++++++++++++++++++++++++++++

Bild 5.9b Sämtliche durch den Scan-Prozeß von HiRes angesprochenen Speicheradressen (NTSC)

	SCREEN MIDDLE					
	HORIZONTAL BLANKING (HBL)			HORIZONTAL DISPLAY ENABLE		
LINE NUM	PAGE 1	PAGE 2	11111111 ØØ123456789ABCDEFØ1234567	PAGE 1	PAGE 2	1111111111111111122222222 Ø123456789ABCDEFØ123456789ABCDEFØ1234567
64	$2010 8208	$4010 16400	++++++++####++++++++++	$2028 8232	$4028 16424	++++++++++++++++++++++++++++++++
65	$2410 9232	$4410 17424	++++++++####++++++++++	$2428 9256	$4428 17448	++++++++++++++++++++++++++++++++
66	$2810 10256	$4810 18448	++++++++####++++++++++	$2828 10280	$4828 18472	++++++++++++++++++++++++++++++++
67	$2C10 11280	$4C10 19472	++++++++####++++++++++	$2C28 11304	$4C28 19496	++++++++++++++++++++++++++++++++
68	$3010 12304	$5010 20496	++++++++####++++++++++	$3028 12328	$5028 20520	++++++++++++++++++++++++++++++++
69	$3410 13328	$5410 21520	++++++++####++++++++++	$3428 13352	$5428 21544	++++++++++++++++++++++++++++++++
70	$3810 14352	$5810 22544	++++++++####++++++++++	$3828 14376	$5828 22568	++++++++++++++++++++++++++++++++
71	$3C10 15376	$5C10 23568	++++++++####++++++++++	$3C28 15400	$5C28 23592	++++++++++++++++++++++++++++++++
72	$2090 8336	$4090 16528	++++++++####++++++++++	$20A8 8360	$40A8 16552	++++++++++++++++++++++++++++++++
73	$2490 9360	$4490 17552	++++++++####++++++++++	$24A8 9384	$44A8 17576	++++++++++++++++++++++++++++++++
74	$2890 10384	$4890 18576	++++++++####++++++++++	$28A8 10408	$48A8 18600	++++++++++++++++++++++++++++++++
75	$2C90 11408	$4C90 19600	++++++++####++++++++++	$2CA8 11432	$4CA8 19624	++++++++++++++++++++++++++++++++
76	$3090 12432	$5090 20624	++++++++####++++++++++	$30A8 12456	$50A8 20648	++++++++++++++++++++++++++++++++
77	$3490 13456	$5490 21648	++++++++####++++++++++	$34A8 13480	$54A8 21672	++++++++++++++++++++++++++++++++
78	$3890 14480	$5890 22672	++++++++####++++++++++	$38A8 14504	$58A8 22696	++++++++++++++++++++++++++++++++
79	$3C90 15504	$5C90 23696	++++++++####++++++++++	$3CA8 15528	$5CA8 23720	++++++++++++++++++++++++++++++++
80	$2110 8464	$4110 16656	++++++++####++++++++++	$2128 8488	$4128 16680	++++++++++++++++++++++++++++++++
81	$2510 9488	$4510 17680	++++++++####++++++++++	$2528 9512	$4528 17704	++++++++++++++++++++++++++++++++
82	$2910 10512	$4910 18704	++++++++####++++++++++	$2928 10536	$4928 18728	++++++++++++++++++++++++++++++++
83	$2D10 11536	$4D10 19728	++++++++####++++++++++	$2D28 11560	$4D28 19752	++++++++++++++++++++++++++++++++
84	$3110 12560	$5110 20752	++++++++####++++++++++	$3128 12584	$5128 20776	++++++++++++++++++++++++++++++++
85	$3510 13584	$5510 21776	++++++++####++++++++++	$3528 13608	$5528 21800	++++++++++++++++++++++++++++++++
86	$3910 14608	$5910 22800	++++++++####++++++++++	$3928 14632	$5928 22824	++++++++++++++++++++++++++++++++
87	$3D10 15632	$5D10 23824	++++++++####++++++++++	$3D28 15656	$5D28 23848	++++++++++++++++++++++++++++++++
88	$2190 8592	$4190 16784	++++++++####++++++++++	$21A8 8616	$41A8 16808	++++++++++++++++++++++++++++++++
89	$2590 9616	$4590 17808	++++++++####++++++++++	$25A8 9640	$45A8 17832	++++++++++++++++++++++++++++++++
90	$2990 10640	$4990 18832	++++++++####++++++++++	$29A8 10664	$49A8 18856	++++++++++++++++++++++++++++++++
91	$2D90 11664	$4D90 19856	++++++++####++++++++++	$2DA8 11688	$4DA8 19880	++++++++++++++++++++++++++++++++
92	$3190 12688	$5190 20880	++++++++####++++++++++	$31A8 12712	$51A8 20904	++++++++++++++++++++++++++++++++
93	$3590 13712	$5590 21904	++++++++####++++++++++	$35A8 13736	$55A8 21928	++++++++++++++++++++++++++++++++
94	$3990 14736	$5990 22928	++++++++####++++++++++	$39A8 14760	$59A8 22952	++++++++++++++++++++++++++++++++
95	$3D90 15760	$5D90 23952	++++++++####++++++++++	$3DA8 15784	$5DA8 23976	++++++++++++++++++++++++++++++++
96	$2210 8720	$4210 16912	++++++++####++++++++++	$2228 8744	$4228 16936	++++++++++++++++++++++++++++++++
97	$2610 9744	$4610 17936	++++++++####++++++++++	$2628 9768	$4628 17960	++++++++++++++++++++++++++++++++
98	$2A10 10768	$4A10 18960	++++++++####++++++++++	$2A28 10792	$4A28 18984	++++++++++++++++++++++++++++++++
99	$2E10 11792	$4E10 19984	++++++++####++++++++++	$2E28 11816	$4E28 20008	++++++++++++++++++++++++++++++++
100	$3210 12816	$5210 21008	++++++++####++++++++++	$3228 12840	$5228 21032	++++++++++++++++++++++++++++++++
101	$3610 13840	$5610 22032	++++++++####++++++++++	$3628 13864	$5628 22056	++++++++++++++++++++++++++++++++
102	$3A10 14864	$5A10 23056	++++++++####++++++++++	$3A28 14888	$5A28 23080	++++++++++++++++++++++++++++++++
103	$3E10 15888	$5E10 24080	++++++++####++++++++++	$3E28 15912	$5E28 24104	++++++++++++++++++++++++++++++++
104	$2290 8848	$4290 17040	++++++++####++++++++++	$22A8 8872	$42A8 17064	++++++++++++++++++++++++++++++++
105	$2690 9872	$4690 18064	++++++++####++++++++++	$26A8 9896	$46A8 18088	++++++++++++++++++++++++++++++++
106	$2A90 10896	$4A90 19088	++++++++####++++++++++	$2AA8 10920	$4AA8 19112	++++++++++++++++++++++++++++++++
107	$2E90 11920	$4E90 20112	++++++++####++++++++++	$2EA8 11944	$4EA8 20136	++++++++++++++++++++++++++++++++
108	$3290 12944	$5290 21136	++++++++####++++++++++	$32A8 12968	$52A8 21160	++++++++++++++++++++++++++++++++
109	$3690 13968	$5690 22160	++++++++####++++++++++	$36A8 13992	$56A8 22184	++++++++++++++++++++++++++++++++
110	$3A90 14992	$5A90 23184	++++++++####++++++++++	$3AA8 15016	$5AA8 23208	++++++++++++++++++++++++++++++++
111	$3E90 16016	$5E90 24208	++++++++####++++++++++	$3EA8 16040	$5EA8 24232	++++++++++++++++++++++++++++++++
112	$2310 8976	$4310 17168	++++++++####++++++++++	$2328 9000	$4328 17192	++++++++++++++++++++++++++++++++
113	$2710 10000	$4710 18192	++++++++####++++++++++	$2728 10024	$4728 18216	++++++++++++++++++++++++++++++++
114	$2B10 11024	$4B10 19216	++++++++####++++++++++	$2B28 11048	$4B28 19240	++++++++++++++++++++++++++++++++
115	$2F10 12048	$4F10 20240	++++++++####++++++++++	$2F28 12072	$4F28 20264	++++++++++++++++++++++++++++++++
116	$3310 13072	$5310 21264	++++++++####++++++++++	$3328 13096	$5328 21288	++++++++++++++++++++++++++++++++
117	$3710 14096	$5710 22288	++++++++####++++++++++	$3728 14120	$5728 22312	++++++++++++++++++++++++++++++++
118	$3B10 15120	$5B10 23312	++++++++####++++++++++	$3B28 15144	$5B28 23336	++++++++++++++++++++++++++++++++
119	$3F10 16144	$5F10 24336	++++++++####++++++++++	$3F28 16168	$5F28 24360	++++++++++++++++++++++++++++++++
120	$2390 9104	$4390 17296	++++++++####++++++++++	$23A8 9128	$43A8 17320	++++++++++++++++++++++++++++++++
121	$2790 10128	$4790 18320	++++++++####++++++++++	$27A8 10152	$47A8 18344	++++++++++++++++++++++++++++++++
122	$2B90 11152	$4B90 19344	++++++++####++++++++++	$2BA8 11176	$4BA8 19368	++++++++++++++++++++++++++++++++
123	$2F90 12176	$4F90 20368	++++++++####++++++++++	$2FA8 12200	$4FA8 20392	++++++++++++++++++++++++++++++++
124	$3390 13200	$5390 21392	++++++++####++++++++++	$33A8 13224	$53A8 21416	++++++++++++++++++++++++++++++++
125	$3790 14224	$5790 22416	++++++++####++++++++++	$37A8 14248	$57A8 22440	++++++++++++++++++++++++++++++++
126	$3B90 15248	$5B90 23440	++++++++####++++++++++	$3BA8 15272	$5BA8 23464	++++++++++++++++++++++++++++++++
127	$3F90 16272	$5F90 24464	++++++++####++++++++++	$3FA8 16296	$5FA8 24488	++++++++++++++++++++++++++++++++

Bild 5.9c Sämtliche durch den Scan-Prozeß von HiRes angesprochenen Speicheradressen (NTSC)

	SCREEN BOTTOM					
	HORIZONTAL BLANKING (HBL)			HORIZONTAL DISPLAY ENABLE		
LINE NUM	PAGE 1	PAGE 2	11111111 Ø0123456789ABCDEFØ1234567	PAGE 1	PAGE 2	1111111111111111222222222 0123456789ABCDEFØ123456789ABCDEF01234567
128	$2038 8248	$4038 16440	++++++++####++++++++++	$2050 8272	$4050 16464	++
129	$2438 9272	$4438 17464	++++++++####++++++++++	$2450 9296	$4450 17488	++
130	$2838 10296	$4838 18488	++++++++####++++++++++	$2850 10320	$4850 18512	++
131	$2C38 11320	$4C38 19512	++++++++####++++++++++	$2C50 11344	$4C50 19536	++
132	$3038 12344	$5038 20536	++++++++####++++++++++	$3050 12368	$5050 20560	++
133	$3438 13368	$5438 21560	++++++++####++++++++++	$3450 13392	$5450 21584	++
134	$3838 14392	$5838 22584	++++++++####++++++++++	$3850 14416	$5850 22608	++
135	$3C38 15416	$5C38 23608	++++++++####++++++++++	$3C50 15440	$5C50 23632	++
136	$20B8 8376	$40B8 16568	++++++++####++++++++++	$20D0 8400	$40D0 16592	++
137	$24B8 9400	$44B8 17592	++++++++####++++++++++	$24D0 9424	$44D0 17616	++
138	$28B8 10424	$48B8 18616	++++++++####++++++++++	$28D0 10448	$48D0 18640	++
139	$2CB8 11448	$4CB8 19640	++++++++####++++++++++	$2CD0 11472	$4CD0 19664	++
140	$30B8 12472	$50B8 20664	++++++++####++++++++++	$30D0 12496	$50D0 20688	++
141	$34B8 13496	$54B8 21688	++++++++####++++++++++	$34D0 13520	$54D0 21712	++
142	$38B8 14520	$58B8 22712	++++++++####++++++++++	$38D0 14544	$58D0 22736	++
143	$3CB8 15544	$5CB8 23736	++++++++####++++++++++	$3CD0 15568	$5CD0 23760	++
144	$2138 8504	$4138 16696	++++++++####++++++++++	$2150 8528	$4150 16720	++
145	$2538 9528	$4538 17720	++++++++####++++++++++	$2550 9552	$4550 17744	++
146	$2938 10552	$4938 18744	++++++++####++++++++++	$2950 10576	$4950 18768	++
147	$2D38 11576	$4D38 19768	++++++++####++++++++++	$2D50 11600	$4D50 19792	++
148	$3138 12600	$5138 20792	++++++++####++++++++++	$3150 12624	$5150 20816	++
149	$3538 13624	$5538 21816	++++++++####++++++++++	$3550 13648	$5550 21840	++
150	$3938 14648	$5938 22840	++++++++####++++++++++	$3950 14672	$5950 22864	++
151	$3D38 15672	$5D38 23864	++++++++####++++++++++	$3D50 15696	$5D50 23888	++
152	$21B8 8632	$41B8 16824	++++++++####++++++++++	$21D0 8656	$41D0 16848	++
153	$25B8 9656	$45B8 17848	++++++++####++++++++++	$25D0 9680	$45D0 17872	++
154	$29B8 10680	$49B8 18872	++++++++####++++++++++	$29D0 10704	$49D0 18896	++
155	$2DB8 11704	$4DB8 19896	++++++++####++++++++++	$2DD0 11728	$4DD0 19920	++
156	$31B8 12728	$51B8 20920	++++++++####++++++++++	$31D0 12752	$51D0 20944	++
157	$35B8 13752	$55B8 21944	++++++++####++++++++++	$35D0 13776	$55D0 21968	++
158	$39B8 14776	$59B8 22968	++++++++####++++++++++	$39D0 14800	$59D0 22992	++
159	$3DB8 15800	$5DB8 23992	++++++++####++++++++++	$3DD0 15824	$5DD0 24016	++
160	$2238 8760	$4238 16952	++++++++####++++++++++	$2250 8784	$4250 16976	++
161	$2638 9784	$4638 17976	++++++++####++++++++++	$2650 9808	$4650 18000	++
162	$2A38 10808	$4A38 19000	++++++++####++++++++++	$2A50 10832	$4A50 19024	++
163	$2E38 11832	$4E38 20024	++++++++####++++++++++	$2E50 11856	$4E50 20048	++
164	$3238 12856	$5238 21048	++++++++####++++++++++	$3250 12880	$5250 21072	++
165	$3638 13880	$5638 22072	++++++++####++++++++++	$3650 13904	$5650 22096	++
166	$3A38 14904	$5A38 23096	++++++++####++++++++++	$3A50 14928	$5A50 23120	++
167	$3E38 15928	$5E38 24120	++++++++####++++++++++	$3E50 15952	$5E50 24144	++
168	$22B8 8888	$42B8 17080	++++++++####++++++++++	$22D0 8912	$42D0 17104	++
169	$26B8 9912	$46B8 18104	++++++++####++++++++++	$26D0 9936	$46D0 18128	++
170	$2AB8 10936	$4AB8 19128	++++++++####++++++++++	$2AD0 10960	$4AD0 19152	++
171	$2EB8 11960	$4EB8 20152	++++++++####++++++++++	$2ED0 11984	$4ED0 20176	++
172	$32B8 12984	$52B8 21176	++++++++####++++++++++	$32D0 13008	$52D0 21200	++
173	$36B8 14008	$56B8 22200	++++++++####++++++++++	$36D0 14032	$56D0 22224	++
174	$3AB8 15032	$5AB8 23224	++++++++####++++++++++	$3AD0 15056	$5AD0 23248	++
175	$3EB8 16056	$5EB8 24248	++++++++####++++++++++	$3ED0 16080	$5ED0 24272	++
176	$2338 9016	$4338 17208	++++++++####++++++++++	$2350 9040	$4350 17232	++
177	$2738 10040	$4738 18232	++++++++####++++++++++	$2750 10064	$4750 18256	++
178	$2B38 11064	$4B38 19256	++++++++####++++++++++	$2B50 11088	$4B50 19280	++
179	$2F38 12088	$4F38 20280	++++++++####++++++++++	$2F50 12112	$4F50 20304	++
180	$3338 13112	$5338 21304	++++++++####++++++++++	$3350 13136	$5350 21328	++
181	$3738 14136	$5738 22328	++++++++####++++++++++	$3750 14160	$5750 22352	++
182	$3B38 15160	$5B38 23352	++++++++####++++++++++	$3B50 15184	$5B50 23376	++
183	$3F38 16184	$5F38 24376	++++++++####++++++++++	$3F50 16208	$5F50 24400	++
184	$23B8 9144	$43B8 17336	++++++++####++++++++++	$23D0 9168	$43D0 17360	++
185	$27B8 10168	$47B8 18360	++++++++####++++++++++	$27D0 10192	$47D0 18384	++
186	$2BB8 11192	$4BB8 19384	++++++++####++++++++++	$2BD0 11216	$4BD0 19408	++
187	$2FB8 12216	$4FB8 20408	++++++++####++++++++++	$2FD0 12240	$4FD0 20432	++
188	$33B8 13240	$53B8 21432	++++++++####++++++++++	$33D0 13264	$53D0 21456	++
189	$37B8 14264	$57B8 22456	++++++++####++++++++++	$37D0 14288	$57D0 22480	++
190	$3BB8 15288	$5BB8 23480	++++++++####++++++++++	$3BD0 15312	$5BD0 23504	++
191	$3FB8 16312	$5FB8 24504	++++++++####++++++++++	$3FD0 16336	$5FD0 24528	++

Bild 5.9d Sämtliche durch den Scan-Prozeß von HiRes angesprochenen Speicheradressen (NTSC)

	VERTICAL BLANKING PERIOD (VBL)					
	HORIZONTAL BLANKING (HBL)			HORIZONTAL DISPLAY ENABLE		
LINE NUM	PAGE 1	PAGE 2	11111111 Ø0123456789ABCDEFØ1234567	PAGE 1	PAGE 2	1111111111111111122222222 0123456789ABCDEFØ123456789ABCDEFØ1234567
192	$2060 8288	$4060 16480	++++++++####++++++++++++	$2078 8312	$4078 16504	++++++++++++++++++++++++++++++++
193	$2460 9312	$4460 17504	++++++++####++++++++++++	$2478 9336	$4478 17528	++++++++++++++++++++++++++++++++
194	$2860 10336	$4860 18528	++++++++####++++++++++++	$2878 10360	$4878 18552	++++++++++++++++++++++++++++++++
195	$2C60 11360	$4C60 19552	++++++++####++++++++++++	$2C78 11384	$4C78 19576	++++++++++++++++++++++++++++++++
196	$3060 12384	$5060 20576	++++++++####++++++++++++	$3078 12408	$5078 20600	++++++++++++++++++++++++++++++++
197	$3460 13408	$5460 21600	++++++++####++++++++++++	$3478 13432	$5478 21624	++++++++++++++++++++++++++++++++
198	$3860 14432	$5860 22624	++++++++####++++++++++++	$3878 14456	$5878 22648	++++++++++++++++++++++++++++++++
199	$3C60 15456	$5C60 23648	++++++++####++++++++++++	$3C78 15480	$5C78 23672	++++++++++++++++++++++++++++++++
200	$20E0 8416	$40E0 16608	++++++++####++++++++++++	$20F8 8440	$40F8 16632	++++++++++++++++++++++++++++++++
201	$24E0 9440	$44E0 17632	++++++++####++++++++++++	$24F8 9464	$44F8 17656	++++++++++++++++++++++++++++++++
202	$28E0 10464	$48E0 18656	++++++++####++++++++++++	$28F8 10488	$48F8 18680	++++++++++++++++++++++++++++++++
203	$2CE0 11488	$4CE0 19680	++++++++####++++++++++++	$2CF8 11512	$4CF8 19704	++++++++++++++++++++++++++++++++
204	$30E0 12512	$50E0 20704	++++++++####++++++++++++	$30F8 12536	$50F8 20728	++++++++++++++++++++++++++++++++
205	$34E0 13536	$54E0 21728	++++++++####++++++++++++	$34F8 13560	$54F8 21752	++++++++++++++++++++++++++++++++
206	$38E0 14560	$58E0 22752	++++++++####++++++++++++	$38F8 14584	$58F8 22776	++++++++++++++++++++++++++++++++
207	$3CE0 15584	$5CE0 23776	++++++++####++++++++++++	$3CF8 15608	$5CF8 23800	++++++++++++++++++++++++++++++++
208	$2160 8544	$4160 16736	++++++++####++++++++++++	$2178 8568	$4178 16760	++++++++++++++++++++++++++++++++
209	$2560 9568	$4560 17760	++++++++####++++++++++++	$2578 9592	$4578 17784	++++++++++++++++++++++++++++++++
210	$2960 10592	$4960 18784	++++++++####++++++++++++	$2978 10616	$4978 18808	++++++++++++++++++++++++++++++++
211	$2D60 11616	$4D60 19808	++++++++####++++++++++++	$2D78 11640	$4D78 19832	++++++++++++++++++++++++++++++++
212	$3160 12640	$5160 20832	++++++++####++++++++++++	$3178 12664	$5178 20856	++++++++++++++++++++++++++++++++
213	$3560 13664	$5560 21856	++++++++####++++++++++++	$3578 13688	$5578 21880	++++++++++++++++++++++++++++++++
214	$3960 14688	$5960 22880	++++++++####++++++++++++	$3978 14712	$5978 22904	++++++++++++++++++++++++++++++++
215	$3D60 15712	$5D60 23904	++++++++####++++++++++++	$3D78 15736	$5D78 23928	++++++++++++++++++++++++++++++++
216	$21E0 8672	$41E0 16864	++++++++####++++++++++++	$21F8 8696	$41F8 16888	++++++++++++++++++++++++++++++++
217	$25E0 9696	$45E0 17888	++++++++####++++++++++++	$25F8 9720	$45F8 17912	++++++++++++++++++++++++++++++++
218	$29E0 10720	$49E0 18912	++++++++####++++++++++++	$29F8 10744	$49F8 18936	++++++++++++++++++++++++++++++++
219	$2DE0 11744	$4DE0 19936	++++++++####++++++++++++	$2DF8 11768	$4DF8 19960	++++++++++++++++++++++++++++++++
220	$31E0 12768	$51E0 20960	++++++++####++++++++++++	$31F8 12792	$51F8 20984	++++++++++++++++++++++++++++++++
221	$35E0 13792	$55E0 21984	++++++++####++++++++++++	$35F8 13816	$55F8 22008	++++++++++++++++++++++++++++++++
222	$39E0 14816	$59E0 23008	++++++++####++++++++++++	$39F8 14840	$59F8 23032	++++++++++++++++++++++++++++++++
223	$3DE0 15840	$5DE0 24032	++++++++####++++++++++++	$3DF8 15864	$5DF8 24056	++++++++++++++++++++++++++++++++
224	$2260 8800	$4260 16992	++++++++################	$2278 8824	$4278 17016	################################
225	$2660 9824	$4660 18016	++++++++################	$2678 9848	$4678 18040	################################
226	$2A60 10848	$4A60 19040	++++++++################	$2A78 10872	$4A78 19064	################################
227	$2E60 11872	$4E60 20064	++++++++################	$2E78 11896	$4E78 20088	################################
228	$3260 12896	$5260 21088	++++++++####++++++++++++	$3278 12920	$5278 21112	++++++++++++++++++++++++++++++++
229	$3660 13920	$5660 22112	++++++++####++++++++++++	$3678 13944	$5678 22136	++++++++++++++++++++++++++++++++
230	$3A60 14944	$5A60 23136	++++++++####++++++++++++	$3A78 14968	$5A78 23160	++++++++++++++++++++++++++++++++
231	$3E60 15968	$5E60 24160	++++++++####++++++++++++	$3E78 15992	$5E78 24184	++++++++++++++++++++++++++++++++
232	$22E0 8928	$42E0 17120	++++++++####++++++++++++	$22F8 8952	$42F8 17144	++++++++++++++++++++++++++++++++
233	$26E0 9952	$46E0 18144	++++++++####++++++++++++	$26F8 9976	$46F8 18168	++++++++++++++++++++++++++++++++
234	$2AE0 10976	$4AE0 19168	++++++++####++++++++++++	$2AF8 11000	$4AF8 19192	++++++++++++++++++++++++++++++++
235	$2EE0 12000	$4EE0 20192	++++++++####++++++++++++	$2EF8 12024	$4EF8 20216	++++++++++++++++++++++++++++++++
236	$32E0 13024	$52E0 21216	++++++++####++++++++++++	$32F8 13048	$52F8 21240	++++++++++++++++++++++++++++++++
237	$36E0 14048	$56E0 22240	++++++++####++++++++++++	$36F8 14072	$56F8 22264	++++++++++++++++++++++++++++++++
238	$3AE0 15072	$5AE0 23264	++++++++####++++++++++++	$3AF8 15096	$5AF8 23288	++++++++++++++++++++++++++++++++
239	$3EE0 16096	$5EE0 24288	++++++++####++++++++++++	$3EF8 16120	$5EF8 24312	++++++++++++++++++++++++++++++++
240	$2360 9056	$4360 17248	++++++++####++++++++++++	$2378 9080	$4378 17272	++++++++++++++++++++++++++++++++
241	$2760 10080	$4760 18272	++++++++####++++++++++++	$2778 10104	$4778 18296	++++++++++++++++++++++++++++++++
242	$2B60 11104	$4B60 19296	++++++++####++++++++++++	$2B78 11128	$4B78 19320	++++++++++++++++++++++++++++++++
243	$2F60 12128	$4F60 20320	++++++++####++++++++++++	$2F78 12152	$4F78 20344	++++++++++++++++++++++++++++++++
244	$3360 13152	$5360 21344	++++++++####++++++++++++	$3378 13176	$5378 21368	++++++++++++++++++++++++++++++++
245	$3760 14176	$5760 22368	++++++++####++++++++++++	$3778 14200	$5778 22392	++++++++++++++++++++++++++++++++
246	$3B60 15200	$5B60 23392	++++++++####++++++++++++	$3B78 15224	$5B78 23416	++++++++++++++++++++++++++++++++
247	$3F60 16224	$5F60 24416	++++++++####++++++++++++	$3F78 16248	$5F78 24440	++++++++++++++++++++++++++++++++
248	$23E0 9184	$43E0 17376	++++++++####++++++++++++	$23F8 9208	$43F8 17400	++++++++++++++++++++++++++++++++
249	$27E0 10208	$47E0 18400	++++++++####++++++++++++	$27F8 10232	$47F8 18424	++++++++++++++++++++++++++++++++
250	$2BE0 11232	$4BE0 19424	++++++++####++++++++++++	$2BF8 11256	$4BF8 19448	++++++++++++++++++++++++++++++++
251	$2FE0 12256	$4FE0 20448	++++++++####++++++++++++	$2FF8 12280	$4FF8 20472	++++++++++++++++++++++++++++++++
252	$33E0 13280	$53E0 21472	++++++++####++++++++++++	$33F8 13304	$53F8 21496	++++++++++++++++++++++++++++++++
253	$37E0 14304	$57E0 22496	++++++++####++++++++++++	$37F8 14328	$57F8 22520	++++++++++++++++++++++++++++++++
254	$3BE0 15328	$5BE0 23520	++++++++####++++++++++++	$3BF8 15352	$5BF8 23544	++++++++++++++++++++++++++++++++
255	$3FE0 16352	$5FE0 24544	++++++++####++++++++++++	$3FF8 16376	$5FF8 24568	++++++++++++++++++++++++++++++++
256	$2BE0 11232	$4BE0 19424	++++++++####++++++++++++	$2BF8 11256	$4BF8 19448	++++++++++++++++++++++++++++++++
257	$2FE0 12256	$4FE0 20448	++++++++####++++++++++++	$2FF8 12280	$4FF8 20472	++++++++++++++++++++++++++++++++
258	$33E0 13280	$53E0 21472	++++++++####++++++++++++	$33F8 13304	$53F8 21496	++++++++++++++++++++++++++++++++
259	$37E0 14304	$57E0 22496	++++++++####++++++++++++	$37F8 14328	$57F8 22520	++++++++++++++++++++++++++++++++
260	$3BE0 15328	$5BE0 23520	++++++++####++++++++++++	$3BF8 15352	$5BF8 23544	++++++++++++++++++++++++++++++++
261	$3FE0 16352	$5FE0 24544	++++++++####++++++++++++	$3FF8 16376	$5FF8 24568	++++++++++++++++++++++++++++++++

Zeile 0, HPE'+1 bis Zeile 160, HPE'	- HiRes (5.9)
Zeile 160, HPE'+1 bis Zeile 192, HPE'	- TEXT (5.6)
Zeile 192, HPE'+1 bis Zeile 224, HPE'	- HiRes (5.9)
Zeile 222, HPE'+1 bis Zeile 0, HPE'	- TEXT (5.6)

HPE' wird während des ersten Zustands des Videoscanner-Horizontalzählers aktiv, also innerhalb von HBL. Auf dem Bildschirm findet die Umschaltung von HiRes nach TEXT und umgekehrt jeweils am Ende der Fernsehzeile (d.h. rechts davon) statt.

Tabelle 5.2 Zusammenfassung des Bildschirm-Scan-Prozesses

LOCATION	HBL	HBL'
SCREEN TOP	Last 16 of THIRD 40 and UNUSED 8	FIRST 40
SCREEN MIDDLE	Last 24 of FIRST 40	SECOND 40
SCREEN BOTTOM	Last 24 of SECOND 40	THIRD 40
VBL	Last 24 of THIRD 40	UNUSED 8 and first 32 of FIRST 40

Der RAM-Refresh im Apple //e

Um die dynamischen Bausteine sicher "aufzufrischen", muß jede ROW-Adresse innerhalb von 2 Millisekunden mindestens einmal angesprochen werden. Damit der Videoscanner diese Funktion sozusagen automatisch erfüllt, sind die ROW-Adressen der RAM-Bausteine in den diversen Videomodi sehr sorgfältig gewählt worden. Nur sie sind in der folgenden Diskussion von Bedeutung, ihre Zuordnungen finden Sie in Tabelle 5.3.

Tabelle 5.3 ROW-Adressen der RAMs und dazugehörige Ausgänge des Videoscanners

RAM Address	Scanner Input
RA0	H0
RA1	H1
RA2	H2
RA3	SUM-A3
RA4	SUM-A4
RA5	SUM-A5
RA6	V0
RA7	V1

Die ersten Bits in dieser Zuordnung sind leicht erklärt: Es handelt sich um die niederwertigen Bits des Videoscanners, die folglicherweise auch am schnellsten ihren Wert wechseln, nämlich um H0, H1, H2, SUM-A3, SUM-A4 und SUM-A5. In jeder Fernsehzeile kommen alle Zustände von 000000 bis 111111 vor.

Der nächsthöhere Ausgang des Videoscanners ist SUM-A6 - dieses Bit ist allerdings für den Refresh unbrauchbar. Über SUM-A6 und die sechs niederwertigen Bits wird ein 7-Bit-Wort (= 128 mögliche Zustände) gebildet, von denen in einer Fernsehzeile aber nur 64 (40 aktiv und 24 während HBL) durchlaufen werden. SUM-A6 behält seinen Wert jeweils für ein Viertel des gesamten Bildaufbaus - weit länger als die geforderten zwei Millisekunden.

Tatsächlich sind die beiden höchstwertigen ROW-Adressen mit V0 und V1 verbunden. Diese beiden Signale werden durch Umschaltungen des Bildmodus nicht beeinflußt und durchlaufen ihre vier möglichen Zustände schnell genug, um die geforderten Bedingungen zu erfüllen. VA, VB und VC sind für diesen Zweck ebenfalls unbrauchbar, sie durchlaufen ihre Zustände zwar schnell genug, haben aber in den Modi TEXT und LoRes keinen Einfluß auf die RAM-Adresse.

Das höchstwertige für den Refresh benutzte Bit ist V1, das seinen Zustand alle 32 Fernsehzeilen wechselt. Die Wertigkeit der Videoscannerbits ist H0-SUM-A5, VA, VB, VC, V0, V1 - daraus folgt, daß für jede Zustandsänderung von V0/V1 alle Zustände von H0-SUM-A5 *achtmal* durchlaufen werden. Ein Überschlag zeigt, daß die Maximalzeit für ein komplettes Durchlaufen aller 256 Refresh-Adressen nicht länger als 25 (d.h. 32 - 7) Fernsehzeilen dauert. Die dafür benötigte Zeit liegt bei 1.59 Millisekunden. Allerdings haben wir am Ende von VBL sechs zusätzliche Zeilen, in denen V0 und V1 beide den Zustand "1" haben - der entsprechende Refresh-Zyklus dauert hier 31 (d.h. 25 + 6) Fernsehzeilen: knapp 1.97 Millisekunden.

Damit ist die Bedingung erfüllt, daß ein Refresh-Zyklus maximal 2 Millisekunden dauern darf (wenn auch recht knapp).

Wie bereits erwähnt, gibt es auch RAM-Bausteine, die mit 128 anstelle von 256 Refresh-Zyklen auskommen. Diese 128 Zyklen werden über A0-A6 adressiert; auf dem gemultiplexten RAM-Adreßbus des Apple //e entspricht das den Leitungen RA1-RA7. Da diese Leitungen zusammen mit RA0 in der geforderten Zeit sämtliche Zustände annehmen, können im Apple //e beide RAM-Typen eingesetzt werden.

Die Speicherverwaltung

Welcher Baustein reagiert, wenn die CPU die Adresse $D000 anspricht? Leichter gefragt als beantwortet: Denn abhängig vom Stand der Softswitches ist es einer der folgenden:

- der $C1-DF-ROM der Hauptplatine;
- Bank 1 der oberen 16k RAM der Hauptplatine;
- Bank 2 der oberen 16k RAM der Hauptplatine;
- Bank 1 der oberen 16k des AUX-RAMs;
- Bank 2 der oberen 16k des AUX-RAMs.

Diese Aufstellung ist immer noch nicht vollständig - schließlich haben wir noch die Leitung INHIBIT', über die der ROM der Hauptplatine abgeschaltet und durch Speicherbausteine einer beliebigen Zusatzkarte in einem der sieben Steckplätze ersetzt werden kann.

Die Technik, mit der diese erstaunliche Vielfalt überhaupt erst möglich wird, trägt den Namen *bank switching* (Bankumschaltung), mehrere Bausteine, die abwechselnd in ein und denselben Speicherbereich geschaltet werden können, werden als *bank switched* ("bankgeschaltet" oder einfach "geschaltet") bezeichnet.

Wenn ein Computer mehr adressierbare Speicherstellen enthält als die CPU in einem durchgehenden Bereich adressieren kann, müssen einzelne Bereiche oder Bausteine geschaltet werden. Im Apple //e haben wir 128 kByte RAM, 16 kByte ROM und einen I/O-Bereich von weiteren 4 kByte, zusammen also 148 kByte Adressen - und eine CPU, die gerade 64 kByte auf einmal adressieren kann.

Allerdings hätte die Bankumschaltung im Apple //e nicht halb so kompliziert ausfallen müssen, wie sie ist - wenn hier nicht wieder einmal die Kompatibilität zum Apple II im Vordergrund gestanden hätte.

Blicken wir einmal ein knappes Jahrzehnt zurück und klären die "historischen" Hintergründe: Der erste Apple II wurde zu einer Zeit entworfen, als RAM-Bausteine mit 16 kBit und ROMs mit 2kByte das maximal Machbare waren. Die RAMs mit 16 kBit waren nur für eine Unmenge Geld zu haben, weitere Entwicklungen waren vorläufig nicht in Sicht. Deshalb enthält der Apple II drei Reihen mit jeweils acht RAM-Bausteinen a 16 kBit, aus denen sich die unteren 48 kByte RAM zusammensetzen. Direkt darüber kommen 4 kByte I/O-Bereich, danach die 12 kByte ROM, deren höchste Adresse $FFFF ist - eine schöne und saubere Einteilung. Der nächste Entwicklungsschritt brachte eine Zusatzkarte mit 16 kByte RAM für den Steckplatz 0 und die erste Komplikation: da diese Zusatzkarte auch für bereits existierende Geräte passen mußte, bei denen der I/O-Bereich auf die Adressen $C000-$CFFF festgelegt war, konnte die Zusatzkarte diesen Bereich nicht mit RAM belegen. Um trotzdem nicht nur 12 kByte der Karte zu nutzen, wurde die erste Bankumschaltung eingebaut, der Adreßbereich $D000-$DFFF existiert in zweifacher Ausführung und ist schaltbar. Bei der Einführung des Apple //e mit weiteren 64 kByte RAM wurde die Bankumschaltung für den Bereich $D000-$DFFF beibehalten - schließlich muß ein Programm, das innerhalb des AUX-RAMs arbeitet, auch an den I/O-Bereich herankommen.

Für alte Apple-Hasen ist an dieser komplexen Systematik nichts besonderes, Neulingen dagegen, die während der Evolutionsschritte des Apple nicht anwesend waren, dürfte es etwas verwirrend vorkommen.

Die Softswitches der MMU

Bankumschaltungen und die Aktivierung von Speicherbereichen und/oder -bausteinen im Apple //e sind die Aufgabe der MMU. Sie enthält 13 Softswitches, über die ein 6502-Programm die Konfiguration des Speichers bestimmen kann, ihr Stand legt fest, welcher Baustein auf einen bestimmten Adreßbereich reagiert. Die dreizehn Softswitches lassen sich in drei Gruppen einteilen:

- sechs Switches sind für die Schaltung zwischen dem RAM der Hauptplatine und dem AUX-RAM zuständig;
- vier Switches schalten zwischen RAM und ROM innnerhalb der oberen 16 kByte und Bank1/Bank2 des RAMs;
- die restlichen drei bestimmen, ob bei einem Zugriff auf den I/O-Bereich der ROM einer Zusatzkarte oder der $C1-$DF-ROM der Hauptplatine angesprochen wird.

Tabelle 5.4 Kontrolladressen der MMU-Softswitches

SOFT SWITCH	OFF ADDRESS	ON ADDRESS	READ ADDRESS	CONDITION AFTER RESET' (OFF)
80STORE	W$C000	W$C001	R$C018*	PAGE2 does not bank switch RAM
RAMRD	W$C002	W$C003	R$C013	Read from motherboard RAM
RAMWRT	W$C004	W$C005	R$C014	Write to motherboard RAM
INTCXROM	W$C006	W$C007	R$C015	Slot response to $C100—$CFFF
ALTZP	W$C008	W$C009	R$C016	Motherboard RAM read/write
SLOTC3ROM	W$C00A	W$C00B	R$C017	Motherboard ROM response to $C3XX
PAGE2	$C054	$C055	R$C01C*	Motherboard RAM read/write
HIRES	$C056	$C057	R$C01D*	PAGE2 does not switch $2000—$3FFF
BANK1**	A3'	A3	R$C011***	High RAM bank 2 response to $DXXX
HRAMRD**	(A1+A2) • (A1 • A2)'	A1 • A2+A1' • A2'	R$C012	$D000—$FFFF read from ROM
PRE-WRITE**	A0'+(R/W')'	A0 • R/W'	None	Reset
HRAMWRT' **	PRE-WRITE • R/W' • A0	A0'	None	$D000—$FFFF write to high RAM
INTC8ROM	$C3XX • SLOTC3ROM'	$CFFF	None	Slot response to $C800—$CFFF

R vor einer Adresse bedeutet, daß von dieser Adresse nur gelesen werden kann.

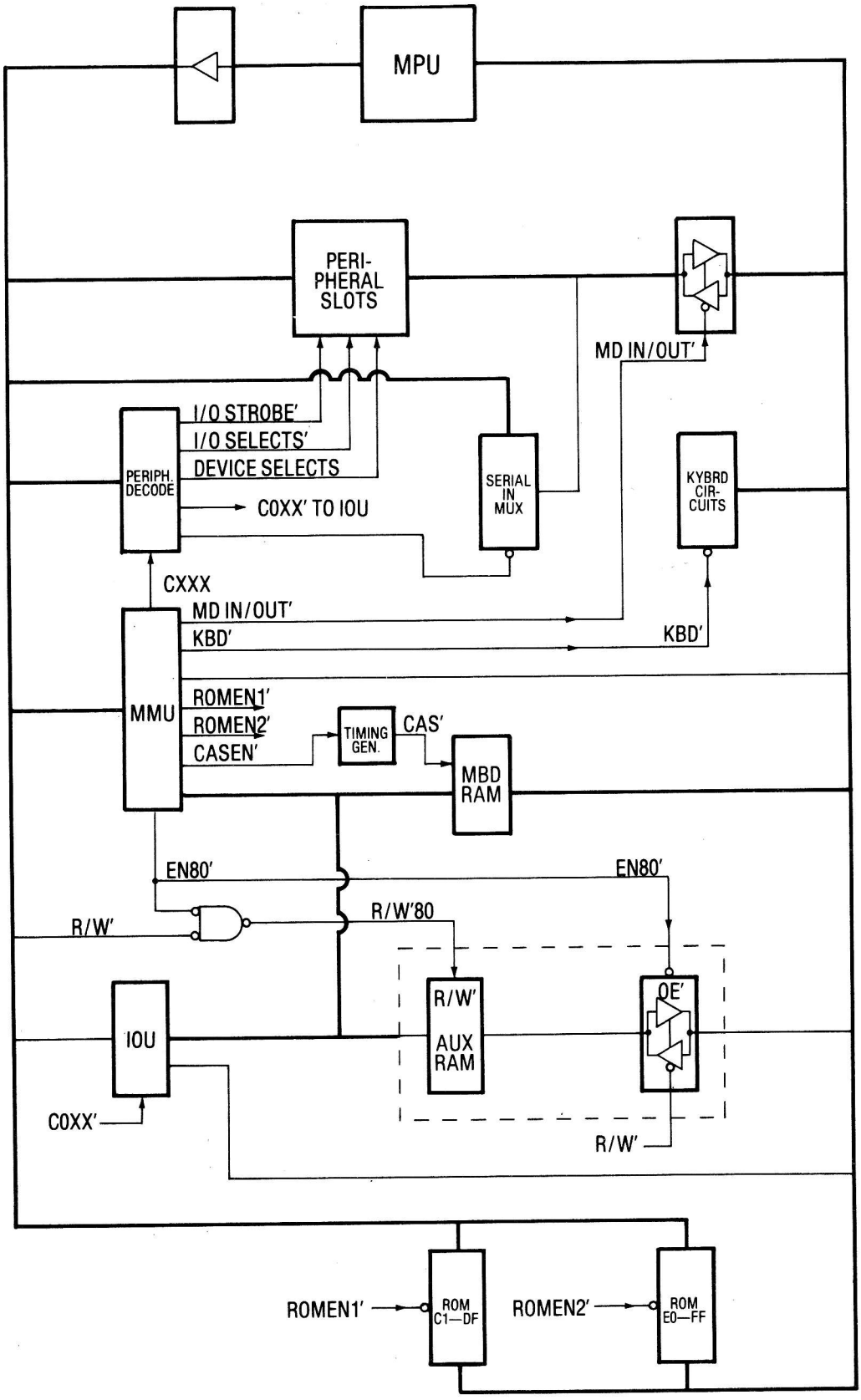
W vor einer Adresse bedeutet, daß auf diese Adresse nur geschrieben werden kann.

* 80STORE, PAGE2 und HIRES existieren elektronisch sowohl in der MMU als auch in der IOU. Bei einem Lesezugriff auf $C018 gibt die MMU den Stand von 80STORE auf MD7 des Datenbusses aus, bei einem Lesezugriff auf $C01C oder $C01D übernimmt die IOU die Ausgabe des Standes von PAGE2 bzw. HIRES.

** Die Kontrolladressen für den hohen RAM sind im Bereich $C08X.

*** Die MMU gibt bei einer Leseaktion von $C011 den Stand von BANK1 invertiert aus (d.h. "0" = "an", "1" = "aus").

Bild 5.10 MMU-Datenbusverwaltung, Verteilung der Signale



Alle Softswitches werden über die Adreßdekodierung gesteuert. Einige haben lediglich eine "An-Adresse" und eine "Aus-Adresse", die Kontrolle von INTC8ROM und den oberen 16k RAM verläuft dagegen etwas komplizierter (s. nächster Abschnitt).

Nach dem Einschalten der Stromversorgung oder nach der Erkennung von RESET' setzt die MMU sämtliche internen Softswitches zurück, d.h. schaltet sie in den Zustand "aus". Tabelle 5.4 listet die Steueradressen der MMU-Softswitches und erklärt die Wirkung der einzelnen Schalter nach einem RESET'.

Einige Spezialisten werden sich wahrscheinlich darüber wundern, daß der Stand der Softswitches BANK1 und HRAMRD über die Adressen $C011 und $C012 gelesen werden kann. Diese beiden Adressen wurden in der ersten Ausgabe des *Technical Reference Manual* vergessen und finden sich erst in der Ausgabe "Juli 1985". Was aber auch in dieser überarbeiteten Ausgabe nicht dokumentiert wird, ist der Softswitch INTC8ROM, über den zwischen I/O STROBE' für Zusatzkarten und ROM der Hauptplatine im Bereich $C3XX und $C800..$CFFF geschaltet wird.

Die Schaltung des hohen RAMs ($D000-$FFFF)

Im Normalfall (d.h. mit zurückgesetzten Softswitches) werden durch Adressen im Bereich von $D000-$FFFF insgesamt 12 kByte ROM auf der Hauptplatine angesprochen. Man kann diesem Bereich aber auch 16 kByte RAM zuordnen. Die Fähigkeit, die oberen 16k zwischen RAM und ROM hin- und herzuschalten, verleiht dem Apple so etwas wie eine gespaltene Persönlichkeit: einerseits verhält er sich wie ein echter Homecomputer mit eingebautem BASIC und einem Kassettenrecorderanschluß, zum anderen wie ein diskettenorientiertes System, das nur ein Startprogramm enthält und erst einmal ein Betriebssystem einlesen muß.

Wir bezeichnen den RAM, den man in den Bereich $D000-$FFFF schalten kann, im folgenden als hohen RAM.

Im Apple //e ist der hohe RAM so ausgelegt, daß er sich wie eine RAM-Zusatzkarte mit 16 kByte im Steckplatz 0 des Apple II verhält. Das Konzept und die Regeln für die Hin- und Herschaltung sind so vollständig und direkt von dieser Zusatzkarte übernommen, daß der hohe RAM auch im //e wie eine Zusatzkarte wirkt, die den ROM-Bereich von $D000-$FFFF überdeckt und sich an- und abschalten läßt.

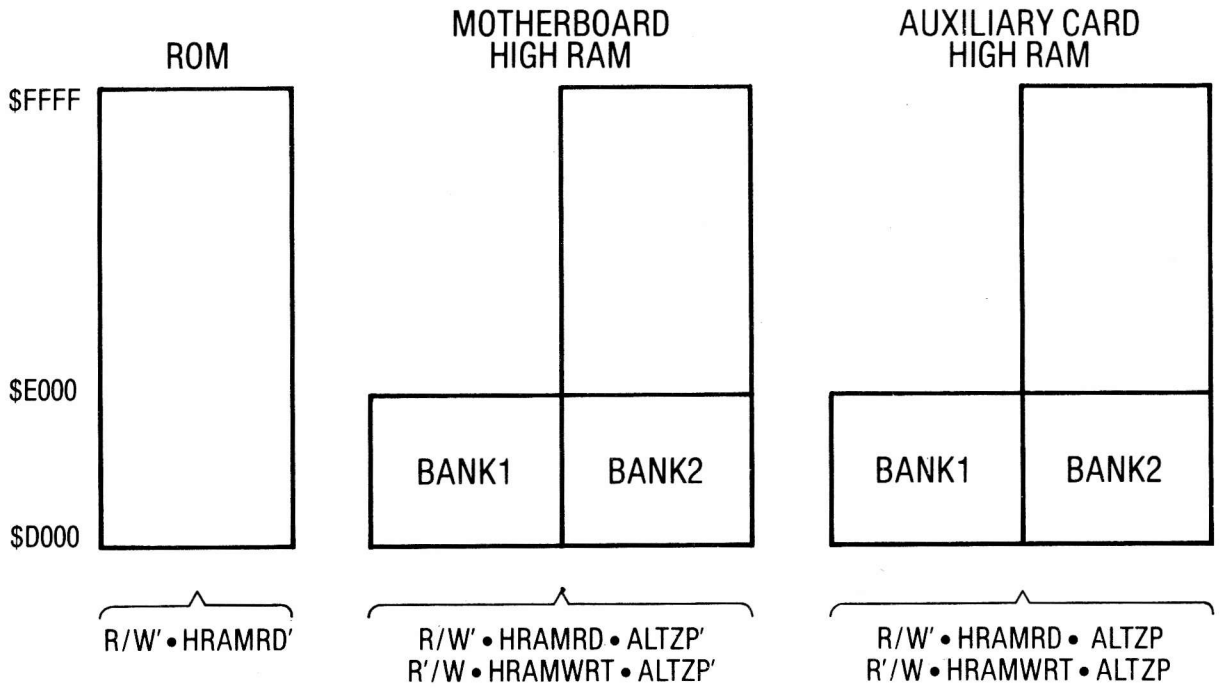
Über $D000-$FFFF lassen sich nur 12 der 16 kByte des hohen RAMs adressieren, 4 kByte bleiben dabei übrig. Aus diesem Grund ist der Bereich $DXXX noch einmal schaltbar (s. Bild 5.11). Eine Adresse im Bereich von $E000-$FFFF spricht immer dieselbe Speicherstelle im RAM an (solange der RAM aktiviert ist), eine Adresse im Bereich $DXXX dagegen wird MMU-intern abhängig vom Stand von BANK1 entweder unverändert weitergegeben oder auf eine RAM-Adresse im Bereich $CXXX übersetzt. Bank2 ist dabei die "primäre" Bank, weil der Softswitch BANK1 durch RESET' zurückgesetzt wird, und wird von den meisten Programmen benutzt, die keine weiteren Bankumschaltungen vornehmen.

Für die Aktivierung und die Umschaltungen innerhalb des hohen RAMs sind vier MMU-Softswitches zuständig, deren Verhalten mit dem der vier Flipflops auf der 16k-Zusatzkarte für den Apple II identisch ist. Alle Kontrolladressen dieser Switches liegen im Bereich $C08X, über den im Apple II ein DEVICE SELECT' für Steckplatz 0 ausgelöst wird.

Die Softswitches zur Steuerung des hohen RAMs sind mit dem Adreßbus auf folgende Weise verknüpft:

1. Über A3 wird die Bankumschaltung innerhalb des hohen RAMs kontrolliert. Das Ansprechen von Adressen im Bereich $C080-$C087 setzt den Switch BANK1 zurück und selektiert damit Bank2; über $C088-$C08F wird BANK1 gesetzt und damit Bank1 selektiert.
2. Über A0 und A1 wird der Softswitch HRAMRD ("High RAM Read" = Lesen des hohen RAMs) kontrolliert. Das Ansprechen der Adressen $C080, $C083, $C084 oder $C087 setzt HRAMRD und gleichzeitig Bank2, das von $C088, $C08B, $C08C oder $C08F setzt HRAMRD und gleichzeitig BANK1. Alle anderen Adressen im Bereich $C08X setzen HRAMRD zurück und schalten so den hohen RAM ab.

Bild 5.11 Die Bankumschaltung in den oberen 16k



3. Speicherstellen des hohen RAMs können nur beschrieben werden, wenn der Softswitch HRAMWRT' ("High RAM WRiTe") zurückgesetzt ist. Um diesen Switch zurückzusetzen, muß erst ein weiterer Softswitch mit dem Namen PRE-WRITE gesetzt werden. Das geschieht durch Ansprechen einer beliebigen ungeraden Adresse im Bereich $C08X. PRE-WRITE wird durch jede gerade Adresse in diesem Bereich zurückgesetzt. HRAMWRT' wird nach Setzen von PRE-WRITE durch eine ungerade Adresse im Bereich $C08X zurückgesetzt und durch gerade Adressen im selben Bereich gesetzt. Wenn PRE-WRITE nicht vorher gesetzt worden ist, bleibt der Stand von HRAMWRT' unverändert.

4. Wenn die MMU ein RESET erkennt, setzt sie alle internen Softswitches zurück. Der hohe RAM wird dadurch für "Lesen" gesperrt (HRAMRD aus, der Prozessor liest vom ROM der Hauptplatine) und für "Schreiben" entsperrt (HRAMWRT' aus). PRE-WRITE wird zusammen mit BANK1 zurückgesetzt, Schreibaktionen des Prozessors im Bereich $DXXX verändern Speicherstellen der Bank2. Da auch beim Einschalten der Stromversorgung ein RESET stattfindet, ist der hohe RAM bei einem Start des Computers immer für "Lesen" gesperrt und für "Schreiben" entsperrt.

5. Wenn der hohe RAM aktiviert ist, spricht eine Adresse im Bereich $D000-$FFFF entweder den hohen RAM der Hauptplatine oder den entsprechenden Bereich im AUX-RAM an. Welcher von beiden angesprochen wird, ist vom Stand des Softswitches ALTZP abhängig. Die Schaltung zwischen dem RAM der Hauptplatine und dem AUX-RAM ist im nächsten Abschnitt besprochen.

Die Softswitches PRE-WRITE und HRAMWRT' arbeiten im Prinzip wie ein Zähler für Lesezugriffe auf ungerade Adressen im Bereich $C08X. Der Zähler wird durch einen Lesezugriff auf eine gerade Adresse und durch Schreibzugriffe auf diesen Bereich zurückgesetzt. Wenn er durch hintereinanderfolgende Lesezugriffe den Wert 2 erreicht hat, ist Schreiben in den hohen RAM hinein erlaubt. Ab diesem Punkt bleibt HRAMWRT' solange aktiv (d.h."0"),

bis ein Zugriff auf eine gerade Adresse im Bereich $C08X erfolgt. Damit ergibt sich eine Kontrollmöglichkeit, die nirgendwo dokumentiert ist: ein Schreibzugriff auf eine ungerade Adresse im Bereich von $C08X setzt HRAMRD zurück, ohne dabei HRAMWRT' zu verändern.[8]

Tabelle 5.5 faßt die Kontrollfunktionen und -adressen für den hohen RAM zusammen. Jede Kontrollfunktion kann über zwei verschiedene Adressen angesprochen werden. Es hat sich aber eingebürgert, die Adressen $C080-83 und $C088-8B zu verwenden.

Der einzige Adreßbereich, in dem der Prozessor nie RAM im Apple //e vorfindet, ist der Bereich $CXXX (vorausgesetzt natürlich, daß keine entsprechende Zusatzkarte eingesteckt ist). Wenn der hohe RAM aktiviert und der Softswitch BANK1 gesetzt ist, übersetzt die MMU eine $DXXX-Adresse auf dem Adreßbus in eine $CXXX-Adresse auf dem gemultiplexten RAM-Adreßbus. Für diese Übersetzung wird schlicht das Adreßbit A12 von "1" auf "0" gesetzt ($D = 1110, $C = 1100). A12 wird über RA4 als COLUMN-Adresse an die RAM-Bausteine weitergegeben, die Zustandsgleichung innerhalb der MMU für die Übersetzung der Adresse lautet:

A4COLUMN = A12 * (BANK1 * DXXX)'

A12 wird also nur auf "0" gesetzt, wenn BANK1 gesetzt ist und die Adresse im Bereich $DXXX liegt.

Wie bereits gesagt, verhält sich der hohe RAM im //e wie eine 16k-Zusatzkarte in Steckplatz 0 des Apple II+, was die Softswitches und ihre Programmierung betrifft. Zusätzlich sind allerdings noch zwei wesentliche Verbesserungen im //e angebracht worden:

1. Die 16k-RAM-Karte des II+ reagiert nicht auf RESET' - mit dem Erfolg, daß ein innerhalb dieser Karte "abgestürztes" Programm nur noch durch den Griff zum Netzschalter zu bändigen ist. Sie wird lediglich durch Einschalten der Stromversorgung in den Zustand "Lesen gesperrt", "Schreiben erlaubt" und "Bank2" gesetzt und ist von da an nur noch über die Softswitches kontrollierbar.
   Im //e setzt jeder RESET' die Softswitches für den hohen RAM in ihren Anfangszustand, der Griff zum Netzschalter wird damit überflüssig.

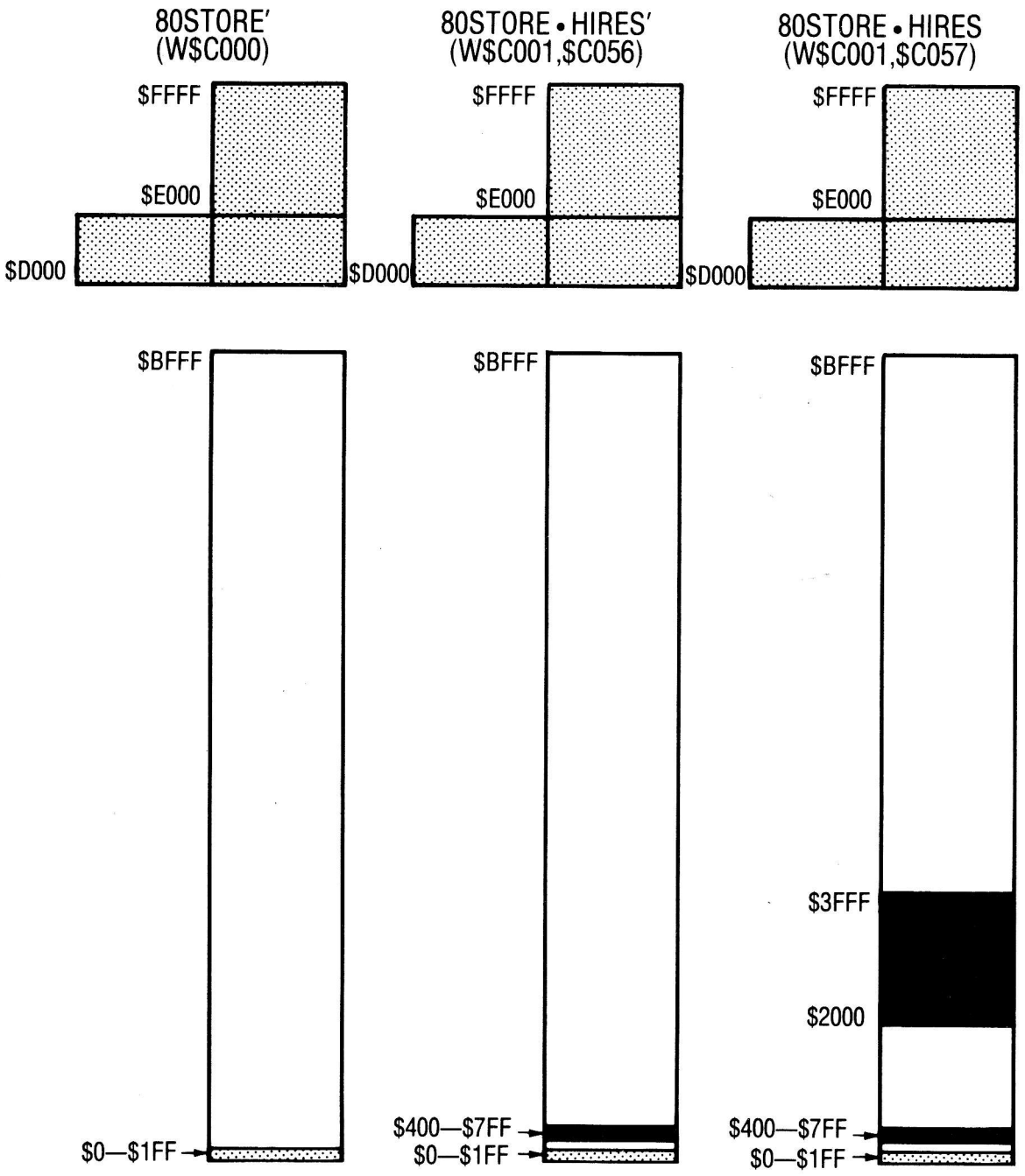
Tabelle 5.5 Die Steuerung des hohen RAMs

BANK 2	BANK 1	ACTION	
$C080 $C084	**$C088** $C08C	WRTCOUNT = 0*, WRITE DISABLE	READ ENABLE
R$C081 R$C085	**R$C089** R$C08D	WRTCOUNT = WRTCOUNT + 1*	READ DISABLE
W$C081 W$C085	**W$C089** W$C08D	WRTCOUNT = 0*	READ DISABLE
$C082 $C086	**$C08A** $C08E	WRTCOUNT = 0*, WRITE DISABLE	READ DISABLE
R$C083 R$C087	**R$C08B** R$C08F	WRTCOUNT = WRTCOUNT + 1*	READ ENABLE
W$C083 W$C087	**W$C08B** W$C08F	WRTCOUNT = 0*	READ ENABLE

** Wenn WRTCOUNT den Stand 2 erreicht, ist Schreiben in den RAM möglich.*

8 Dieser Trick ist zwar auch in der Ausgabe "Juli 1985" des *Technical Manual* nicht in den Tabellen enthalten, macht aber die IRQ'-Behandlung im "enhanced" //e überhaupt erst möglich: Da im Maschinenstatusbyte kein Bit mehr für HRAMWRT' übrig ist, wird der hohe RAM über STA $C081 abgeschaltet und im RTI-Handler mit STA $C081 bzw. STA $C08B wieder angeschaltet - (Anm. d. Übers.).

Bild 5.12 Geschaltete Adreßbereiche auf der Hauptplatine / dem AUX-RAM



2 Die 16k-RAM-Karte des II+ reagiert nicht auf INHIBIT', wenn sie aktiviert ist. Noch ärgerlicher: Der Bereich $F800-$FFFF ist überhaupt nicht freizubekommen - falls der RAM ausgeschaltet ist, wird dieser Bereich von einem ROM auf der Karte übernommen.[9]
Im Apple //e kann über INHIBIT' alles abgeschaltet werden, was im Bereich $D000-$FFFF liegt: der ROM der Hauptplatine, der hohe RAM der Hauptplatine und die oberen 16k des AUX-RAMs. Die Aktivierung von RAM oder ROM findet über die MMU-Steuersignale CASEN', EN80', ROMEN1' und ROMEN2' statt - alle diese Signale werden MMU-intern über INHIBIT' geschaltet. Dadurch kann jede Zusatzkarte unabhängig von der momentanen Konfiguration der oberen 16k diesen Bereich für sich beanspruchen, indem sie INHIBIT' auf den Pegel "0" bringt.

Die Schaltung zwischen dem RAM der Hauptplatine und dem AUX-RAM

Ein Programm kann die Speicherbank, in der es sich selber befindet, nur unter größeren Schwierigkeiten umschalten. Das Problem besteht darin, daß die CPU nach dem Umschaltvorgang beginnt, weitere Programmbefehle aus der neu aktivierten Bank zu lesen - das Programm, das sie eigentlich ausführen sollte, steht in der gerade abgeschalteten Bank. Eine Lösungsmöglichkeit besteht darin, die neue Bank vor der Umschaltung so vorzubereiten, daß am Schaltpunkt ein Programm steht, mit dem Daten und Befehle von der vorher aktiven Bank gelesen werden können. Unmöglich ist so etwas nicht - allerdings ist es eine ärgerliche und unproduktive Arbeit für den Programmierer. Eine bessere Lösung ist ein anderer Aufbau des Computers, nämlich so, daß das umschaltende Programm immer außerhalb des geschalteten Bereichs stehen kann.

Apple, Inc. hat große Schritte in diese Richtung unternommen: der Apple //e bietet dem Programmierer einiges an Komfort für die Bankumschaltung zwischen dem RAM der Hauptplatine und dem AUX-RAM.

Es gibt eine ROM-residente Transferroutine, über die ein Programm zwischen beiden RAM-Bänken hin- und herspringen kann, beide Bänke sind unterteilt und einzeln schaltbar. Eine Beschreibung der Transferroutine finden Sie im Handbuch der 80-Zeichen-Karte sowie im *Technical Reference Manual* für den //e - wir wollen uns in den nächsten Abschnitte hauptsächlich mit den Softswitches beschäftigen.

Anstelle eines einzigen Schalters, über den einfach der gesamte RAM von "Hauptplatine" auf "AUX" umgeschaltet wird, gibt es immerhin sechs davon, die sich allesamt innerhalb der MMU befinden.

Die Schalter sind so aufgebaut, daß ein Schalter im gesetzten Zustand den AUX-RAM aktiviert und im zurückgesetzten Zustand ("aus") den RAM der Hauptplatine. Eine grafische Darstellung der Funktion dieser sechs Schalter finden Sie in Bild 5.12, ihre Kontrolladressen in Tabelle 5.4. Im folgenden eine kurze Beschreibung der einzelnen Switches und ihrer Kombinationsmöglichkeiten:

RAMWRT und **RAMRD** schalten den Bereich von $0200-$BFFF und lassen sich unabhängig voneinander setzen. Wenn RAMRD gesetzt ist, liest der Prozessor bei Adressen im Bereich $0200-$BFFF aus dem AUX-RAM; wenn RAMWRT gesetzt ist, finden Schreibaktionen innerhalb dieses Adreßbereichs in den AUX-RAM hinein statt.

Wenn 80STORE gesetzt ist, haben RAMRD und RAMWRT für den Bereich $0400-$07FF keine Bedeutung; wenn 80STORE und HIRES gesetzt sind, gilt dasselbe auch noch für den Bereich $2000-$3FFF.

80STORE, PAGE2 und **HIRES** schalten die Seite 1 der Bildspeicherbereiche ($0400-$07FF und $2000-$3FFF) zwischen dem RAM der Hauptplatine und dem AUX-RAM.

Wenn 80STORE gesetzt ist, dann schaltet PAGE2 abhängig von HIRES entweder den Speicherbereich $400-$7FF (HIRES "aus") oder $2000-$3FFF (Hires "an") zwischen "Hauptplatine" (PAGE2 "aus") und "AUX" um (PAGE2 "an").

Wenn 80STORE zurückgesetzt ist, dann schalten RAMRD und RAMWRT diese beiden Bereiche in derselben Weise wie den Rest des Bereichs $0200-$BFFF.

80STORE, PAGE2 und HIRES existieren sowohl in der MMU als auch in der IOU. In der MMU sind sie für die Umschaltung der Bänke, in der IOU für die Umschaltung der Bildmodi zuständig. Wie schon erwähnt, macht 80STORE im gesetzten Zustand PAGE2 in der IOU unwirksam, was den Bildmodus betrifft.

---

9 Das gilt nur für die Originalkarten von Apple, auf denen sich in diesem Bereich der "Autostart"-ROM befindet, sowie für exakte Nachbauten. Die meisten 16k-Karten von anderen Herstellern haben keinen ROM und belegen deshalb den Bereich $F800-$FFFF nur dann, wenn sie auch tatsächlich aktiviert sind.

ALTZP schaltet den Bereich $0-$1FF und (falls aktiviert) den hohen RAM ($D000-$FFFF). Im zurückgesetzten Zustand von ALTZP ist der RAM der Hauptplatine, ansonsten der AUX-RAM aktiv. Eine getrennte Kontrolle für Lesen und Schreiben gibt es hier nicht.

Dem Gesamtaufbau dieser Bankumschaltung liegt ein logisches und durchdachtes Konzept zugrunde. Der hohe RAM wird separat vom Hauptteil des RAM geschaltet und stellt einen von ihm völlig getrennten Bereich dar. Die Speicherseiten 0 und 1 werden ebenfalls separat geschaltet, weil sie spezielle Funktionen für den 6502 haben. Schlußendlich existiert eine separate Schaltungsmöglichkeit für den Bildspeicherbereich, die eine einfachere Handhabung der 80er Videomodi erlaubt. Anhand von einigen Beispielen wollen wir uns die Vor- und Nachteile eines derartigen Aufbaus einmal ansehen.

Beispiel 1: Ein Maschinensprache-Programm befindet sich in den oberen 16k des RAMs, des ROMs oder auf einer Zusatzkarte und benutzt die unteren 48k beider RAM-Bänke für die Datenspeicherung.

Dieses Programm kann sehr einfach mit RAMRD und RAMWRT zwischen dem RAM der Hauptplatine und dem AUX-RAM schalten. Die Pointer des Programms auf Seite 0 und der Stack auf Seite 1 werden nicht beeinflußt, weil durch RAMRD und RAMWRT der Bereich $0-$1FF nicht geschaltet wird. Auch bei ROM-Programmen enthält dieser Bereich normalerweise kritische Daten und muß bei Bankumschaltungen mit äußerster Sorgfalt behandelt werden.

Auch das Hin- und Herkopieren von Daten zwischen beiden Bänken ist sehr einfach durchführbar:

```
ORG  $D000          ; in den oberen 16k
STA  $C003          ; setzt RAMRD
STA  $C004          ; RAMWRT zurück
LDA  $8000          ; liest AUX
STA  $8000          ; schreibt MAIN
```

Beispiel 2: Ein Maschinensprache-Programm befindet sich in den unteren 48k des Hauptplatinen-RAMs und benutzt die oberen 16k von Hauptplatine und AUX-RAM zur Datenspeicherung.

Hier wird es mit dem Design des Apple //e problematisch. Das Programm kann die oberen 16k RAM über HRAMRD und HRAMWRT' problemlos aktivieren - wenn es aber über ALTZP diesen Bereich auf "AUX" umschaltet, ist damit ebenfalls die Seite 0 und der Stack des Programms verschwunden. Der Programmierer muß immer darauf achten, ALTZP vor dem nächsten Zugriff auf Seite 0 oder den Stack des Programms wieder zurückzusetzen. Eine andere Möglichkeit wäre, den Inhalt der Seite 0 und 1 vor dem Setzen von ALTZP in den AUX-RAM zu kopieren bzw. von da aus zum RAM der Hauptplatine, bevor ALTZP wieder zurückgesetzt wird. In allen Fällen dürfte die Arbeitsgeschwindigkeit des Programms stark herabgesetzt werden. Ein Beispiel dafür, wie es nicht funktioniert:

```
ORG  $1000          ; in den unteren 48k
STA  $C009          ; setzt ALTZP
LDX  $BA            ; funktioniert nicht
LDA  ($44),Y        ; auch nicht
RTS                 ; erst recht nicht
```

Beispiel 3: Ein Maschinensprache-Programm befindet sich in den unteren 48k und enthält eigene Routinen für die 80-Zeichen-Karte.

In den 80er Modi besteht der Bildspeicher für TEXT aus den beiden Bereichen $0400-$07FF im RAM der Hauptplatine und im AUX-RAM, die 80-Zeichen-Routinen müssen auf beide Bereiche gleich oft zugreifen.

Die Umschaltung funktioniert sehr einfach durch Setzen von 80STORE und Zurücksetzen von HIRES. Danach kann der Bereich $0400-$07FF mit PAGE2 unabhängig vom Rest des Speichers hin- und hergeschaltet werden. Dasselbe gilt für LoRes80 und (nach Setzen von HIRES) für HiRes80 und die Umschaltung des Bereichs von $2000-$3FFF.

Problematisch wird es dagegen, wenn ein Programm die "Seite 2" in einem der 80er Modi benutzen will und deshalb 80STORE zurücksetzt und via PAGE2 zwischen "Seite 1" und "Seite 2" schaltet. Hier kann die CPU auf die entsprechenden Bereiche des AUX-RAMs ($800..$BFF und $4000-$5FFF) nur über RAMWRT bzw. RAMRD zugreifen.

Man kann zu diesem Zweck das kontrollierende Programm einfach doppelt speichern (einmal im RAM der Hauptplatine und zum anderen im AUX-RAM) - es ist aber erheblich einfacher, wenn man das gesamte Programm in die oberen 16k verlegt. Ein Beispiel für die Ausgabe eines Zeichens auf "Seite 1" durch ein Programm in den unteren 48k:

```
ORG $1000  ; in den unteren 48k
STA $C001  ; setzt 80STORE
STA $C055  ; PAGE2 setzt AUX
STA (BASL),Y ; schreibt Zeichen
```

Beispiel 4: Ein Programmierer möchte den AUX-RAM für die Speicherung von Daten eines Applesoft- oder Integer BASIC-Programms benutzen.

Abgesehen von der (immer vorhandenen) Möglichkeit, einen neuen Applesoft-Interpreter zu schreiben, sieht es da schlecht aus. Durch eine Umschaltung des Bereichs $200-$BFFF bleiben zwar Seite 0, Stack und der Applesoft-Interpreter selber unberührt - nur ist leider der BASIC-Programmtext danach in der falschen Bank. Von einem BASIC-Programm aus kann man 80STORE, HIRES und PAGE2 setzen und auf diese Weise auf den AUX-RAM in den Bereichen $400..$BFF und $2000-$3FFF mit PEEK, POKE, HPLOT, PLOT, VLIN, HLIN und PRINT zugreifen. Außerhalb der via 80STORE erreichbaren Bereiche ist nur noch mit Maschinensprache etwas zu machen. Ein Beispiel für einen funktionierenden AUX-Zugriff von BASIC aus:

```
10 POKE -16383,0: REM setzt 80STORE
20 POKE -16299,0: REM PAGE2 setzt AUX
30 POKE 1024,193: REM "A" in der oberen linken Ecke
```

Die MMU schaltet über die Signale **CASEN'** und **EN80'** zwischen dem RAM der Hauptplatine und dem AUX-RAM. Wenn der Prozessor den RAM der Hauptplatine anspricht, setzt die MMU CASEN' aktiv und sperrt EN80'; bei einem Zugriff auf den AUX-RAM wird CASEN' gesperrt und EN80' aktiviert. Beide Signale werden MMU-intern über INHIBIT' geschaltet. Eine Zusatzkarte, die einen Adreßbereich für sich beansprucht, braucht sich deshalb nicht darum zu kümmern, ob der RAM der Hauptplatine oder der AUX-RAM gerade aktiv ist.

Der Aufbau des I/O-Bereichs ($C000-$CFFF)

Der Bereich von $C000-$CFFF ist im Apple für die Ein- und Ausgabe reserviert. Im Apple II war dieser Bereich ausschließlich der Adressierung von Zusatzkarten vorbehalten, im Apple //e ist es möglich, den Bereich $C100-$CFFF mit internem ROM zu belegen. Ob der interne ROM oder eine Zusatzkarte auf eine Adresse im Bereich $C100-$CFFF reagiert, läßt sich mit drei Softswitches der MMU bestimmen: **INTCXROM**,[10] **SLOTC3ROM** und **INTC8ROM**.

Auf diesen Bereich reagiert auch die Erzeugung der Signale I/O STROBE' und I/O SELECT', mit denen normalerweise der ROM einer Zusatzkarte im entsprechenden Steckplatz aktiviert wird. Die drei Softswitches schalten zwischen der Aktivierung einer Zusatzkarte und dem Ansprechen des $C0-DF-ROMs auf der Hauptplatine um. Der Bereich $C0XX wird dabei nicht beeinflußt. Es gibt keine Möglichkeit auf dem Apple //e, den Bereich $C0XX zu kontrollieren.

Ein merkwürdiger Nebeneffekt dieser Tatsache liegt darin, daß die Erzeugung des Signals DEVICE SELECT' für Zusatzkarten deshalb nicht durch die drei Softswitches unterdrückt werden kann. Eine Zusatzkarte, die nur auf DEVICE SELECT', nicht aber auf I/O STROBE' und I/O SELECT' reagiert, kann dann sogar in Steckplatz 3 betrieben werden, wenn eine 80-Zeichen-Karte im speziellen Steckplatz installiert ist.

INTCXROM schaltet den Bereich $C100-$CFFF zwischen "ROM der Zusatzkarten" und "ROM der Hauptplatine" um, SLOTC3ROM führt dieselbe Umschaltung für den Bereich $C3XX durch. Dadurch beeinflussen *beide* Switches den Adreßbereich $C3XX und bestimmen, welcher Baustein auf eine $C3XX-Adresse reagiert. Wenn nur einer der beiden Switches auf "intern" gesetzt ist, aktiviert ein Zugriff auf $C3XX den ROM der Hauptplatine. Die folgende Wahrheitstabelle zeigt die vier Kombinationsmöglichkeiten und ihre Wirkung:

---

10 Apple, Inc. bezeichnet INTCXROM als SLOTCXROM - allerdings wird der ROM der Steckplätze abgeschaltet, wenn dieser Softswitch gesetzt ist. Die Bezeichnung SLOTCXROM' wäre demzufolge korrekt, ich halte INTCXROM aber für deskriptiver. Überhaupt scheint es da ein Mißverständnis bei Apple über die Funktion dieses Switches gegeben zu haben - die Beschreibungen auf den Seiten 133 und 214 des *Technical Manual* sind nachweislich unvollständig.

Die entsprechende Tabelle (5.5) findet sich auch in der Ausgabe "Juli 1985" des *Technical Manual* unverändert wieder, der Zusammenhang zwischen den beiden Switches SLOTC3ROM und SLOTCXROM wird nicht gezeigt - (Anm. d. Übers.).

INTCXROM	SLOTC3ROM	$C100-$C2FF $C400-$CFFF	$C300-$C3FF
reset	reset	slot	internal
reset	set	slot	slot
set	reset	internal	internal
set	set	internal	internal

Was ich auch in der Dokumentation von Apple, Inc. vergeblich gesucht habe, ist der Softswitch INTC8ROM. Gefunden habe ich allerdings die Möglichkeit, den ROM der Hauptplatine im Bereich $C800-$CFFF zu aktivieren, ohne vorher INTCXROM setzen zu müssen. In diesem Bereich ist der ROM immer aktiv, nachdem SLOTC3ROM zurückgesetzt und eine $C3XX-Adresse angesprochen worden ist. Der Bereich $C800-$CFFF bleibt dabei solange dem ROM der Hauptplatine zugeordnet, bis entweder die Adresse $CFFF angesprochen wird oder die MMU ein RESET' erkennt.

INTC8ROM hat keine direkt zugeordneten Setz- und Rücksetzadressen. Durch Ansprechen einer $C3XX-Adresse, während INTC3ROM zurückgesetzt ist, wird der Softswitch gesetzt; durch Ansprechen der Adresse $CFFF oder ein RESET' wird er zurückgesetzt.

Genauso wie im Bereich $C3XX, der entweder über INTCXROM oder über INTC3ROM auf "intern" geschaltet werden kann, gibt es eine ODER-Bedingung für den Bereich $C800-$CFFF. Wenn INTC3ROM oder INTCXROM gesetzt sind, erzeugt ein Zugriff auf $C3XX ein aktives ROMEN1'-Signal und ein inaktives I/O SELECT' für eine eventuelle Zusatzkarte in Steckplatz 3. Wenn INTCXROM oder INTC8ROM gesetzt sind, erzeugt ein Zugriff auf den Bereich $C800-$CFFF ein aktives ROMEN1'-Signal, I/O STROBE' bleibt inaktiv.

Die Ausführung von INTC8ROM folgt dem festgelegten Protokoll für eine Zusatzkarte in Steckplatz 3, die auf I/O SELECT' und I/O STROBE' reagiert. Über dieses Protokoll setzt sich die Simulation einer 80-Zeichen-Karte in Steckplatz 3 im //e fort (mit I/O STROBE werden wir uns in Kapitel 7 noch ausführlich beschäftigen).

Über INTCXROM, SLOTC3ROM und INTC8ROM hat die CPU Zugriff auf die untere Hälfte des $C1-$DF-ROMs. Über INTCXROM sind die Erweiterungen des Monitorprogramms und die eingebauten Selbsttestfunktionen erreichbar, über INTC3ROM und INTC8ROM die Routinen zur Steuerung der 80-Zeichen-Karte.

Wie auch alle anderen Softswitches der MMU werden INTCXROM, INTC3ROM und INTC8ROM bei einem RESET' zurückgesetzt. Dadurch wird die "normale" I/O-Dekodierung aktiv, die sämtliche Signale außer I/O SELECT' für Steckplatz 3 liefern kann. Die RESET-Routine prüft, ob eine RAM-Karte im speziellen Steckplatz installiert wurde. Wenn nicht, wird SLOTC3ROM gesetzt, ansonsten bleibt SLOTC3ROM zurückgesetzt.

Die MMU bestimmt die Umschaltung zwischen "Karten-ROM" und "internem ROM" mit zwei Signalen, nämlich **$CXXX** und **ROMEN1'**. ROMEN1' wird MMU-intern über INHIBIT' geschaltet - womit auch dieser Bereich von einer Zusatzkarte beansprucht werden kann, indem sie zuerst auf "internen ROM" schaltet und danach INHIBIT' auf aktiven Pegel bringt.

KBD' und MD IN/OUT'

Die MMU steuert zwei Buskontrollsignale, die nichts mit dem RAM zu tun haben: KBD', über das die Tastatur aktiviert wird, und **MD IN/OUT'** zur Steuerung der Richtung des peripheren Datenbusses.

Die MMU setzt KBD' während PHASE0 auf "0", wenn die CPU auf eine Adresse im Bereich von $C000-$C01F zugreift. KBD' hat keine weitere Steuerung (d.h. ist nicht abschaltbar) und aktiviert die Datenausgänge des Tastatur-ROMs, dessen Datum danach von der CPU gelesen wird. Im Normalfall wird zur Tastaturabfrage nur die Adresse $C000 benutzt, mit der momentan verwendeten MMU ist theoretisch auch eine Tastaturabfrage im Bereich $C01X möglich. Natürlich kommt man mit einem entsprechend geschriebenen Programm auf einem Apple II in Schwierigkeiten - deshalb möchte es sein, daß Apple, Inc. diese zusätzliche Möglichkeit für den //e nirgends dokumentiert hat.

MD IN/OUT' hat normalerweise den Pegel "0", d.h. die Richtung des Datenflusses geht vom Datenbus der Hauptplatine zu den Zusatzkarten. Unter den folgenden Bedingungen geht MD IN/OUT' während PHASE0 auf "1":

1. Bei einer Leseaktion im Bereich von $C020-$C0FF. Dadurch kann die CPU oder ein DMA-Baustein die seriellen Eingänge auf $C06X, Eingänge von Zusatzkarten, die über DEVICE SELECT' geschaltet werden, sowie Daten von "Exoten" lesen, die auf diesen Adreßbereich mit Datenausgaben reagieren.

2. Bei einer Leseaktion im Bereich von $C100-$CFFF, wenn über die entsprechende Stellung von INTCXROM, SLOTC3ROM und INTC8ROM eine physikalisch existierende Zusatzkarte angesprochen wird. Darüber geschieht das Lesen von Daten von Zusatzkarten, die über I/O SELECT' oder I/O STROBE geschaltet werden.
3. Bei jeder Leseaktion des Prozessors, wenn INHIBIT' aktiv ist. Dadurch kann der Prozessor oder ein DMA-Baustein Daten von einer Zusatzkarte geliefert bekommen, die vorher via INHIBIT' ROM und RAM der Hauptplatine abgeschaltet hat.
4. Bei jeder Schreibaktion, wenn DMA' aktiv ist. Dadurch kann ein DMA-Baustein auf einer Zusatzkarte Bausteine der Hauptplatine mit Daten beschicken.

Die Kontrolle von MD IN/OUT' sieht recht komplex aus - wenn man sich die Steuerung noch einmal genau ansieht, wird man allerdings merken, daß im Betrieb des Apple //e sämtliche formulierten Bedingungen erfüllt werden müssen. Der einzige Punkt, bei dem es ein Mißverständnis gegeben haben muß, ist der, daß MD IN/OUT' bei einem Lesezugriff auf den Bereich $C000-1F ebenfalls umgeschaltet wird, wenn INHIBIT' aktiv ist - obwohl der Tastatur-ROM durch INHIBIT' nicht abgeschaltet werden kann. Das Ergebnis ist eine Kollision zwischen dem Treiber des peripheren Datenbusses und den Ausgangstreibern des Tastatur-ROMs bzw. der IOU für MD7. Größere Schäden entstehen dadurch im allgemeinen nicht - im Dauerbetrieb dürfte sich allerdings der (stärkere) Peripherietreiber "durchsetzen". Für die Entwickler von Peripheriekarten ist das auch kein größeres Problem: Wenn die Karte mit INHIBIT' arbeitet, muß sie diese Leitung eben bei Lesezugriffen des Prozessors im Bereich $C000-1F grundsätzlich auf den Pegel "1" bringen.

Der funktionale Aufbau der MMU

Die Bilder 5.13a und 5.13b geben den funktionalen Aufbau der MMU wieder. Bild 5.13a zeigt die Adreßdekodierung, die Erkennung von RESET', den Adressmultiplexer und die Softswitches der MMU. Bild 5.13b zeigt den logischen Aufbau der Erzeugung von Datenbuskontrollsignalen aus der Adreßdekodierung, den Zuständen der Softswitches und den Eingängen R/W', DMA' und INHIBIT'. Beide Zeichnungen sind nur als Beschreibung der MMU-Funktionen, nicht aber als Wiedergabe der tatsächlichen Ausführung zu verstehen. Ich habe diese Bilder gezeichnet, um die Arbeitsweise der MMU zu illustrieren, und kann nichts darüber sagen, wie einzelne Gatter tatsächlich in der MMU aufgebaut und miteinander verbunden sind.

Diese Zeichnungen sind mehr oder weniger eine Zusammenfassung aller über die MMU in diesem Buch gemachten Aussagen und sollten als Referenz für Leser dienen, die (wie ich) lieber Schaltpläne vor der Nase haben. Anhand dieser Schaltpläne noch ein paar Erklärungen:

1. Die MMU hat keinen Eingang für RESET' - hier sind den Entwicklern die Pins am Gehäuse ausgegangen. Die MMU erkennt ein RESET des Systems an drei hintereinanderfolgenden Zugriffen auf Seite 1, gefolgt von einem Zugriff auf $FFFC. Diese Erkennung funktioniert für den Start des Computers und für jeden Druck auf RESET - außerdem "funktioniert" sie noch für jedes Programm auf Seite 1, das die Speicherstelle $FFFC liest.
2. Es scheint ein zeitliches Fenster zu geben, in dem der Adreßbus auf Steueradressen für Softswitches überprüft wird. Wenn sich eine Softswitch-Steueradresse länger als 40 ns während dieses Fensters auf dem Adreßbus befindet, reagiert der entsprechende Switch darauf. Soweit ich das bestimmen konnte, besteht dieses Fenster aus PHASE0 * (Q3 + RAS')', d.h. aus der letzten 14M-Periode von PHASE0. Während dieses Zeitraums ist eine vom 6502 ausgegebene Adresse immer gültig.
3. Wenn der Prozessor den Stand eines MMU-Softswitches liest, gibt die MMU den entsprechenden Wert auf MD7 aus. Der entsprechende Zeitpunkt scheint PHASE0 + PHASE0 * Q3 * RAS' zu sein, d.h. die MMU belegt MD7 während der gesamten PHASE0 und dem ersten 14M-Zyklus der folgenden PHASE1.
4. Die vier Softswitches zur Konfiguration des hohen RAMs verhalten sich wie die vier Flipflops der entsprechenden Zusatzkarte des II+.
5. Nach einem RESET der MMU liest der Prozessor sämtliche Softswitches als "0" - außer BANK1. Bild 5.13a zeigt ein schaltungstechnisches Äquivalent für BANK1, dessen invertierter Zustand von der CPU gelesen werden kann. Denkbar wäre ebenfalls ein Softswitch BANK2, der durch ein RESET gesetzt wird und dessen nicht-invertierter Zustand von der CPU gelesen werden kann. Das in Bild 5.13a gezeigte Schaltbild lehnt sich aber enger an die 16k-RAM-Karte an und folgt der generellen Regel, daß die MMU alle Softswitches bei einem RESET' zurücksetzt.

6. RAS' schaltet im Adressmultiplexer der MMU zwischen ROW und COLUMN um. Das zeitliche Fenster, in dem die MMU die gemultiplexte RAM-Adresse ausgibt, scheint PHASE0' * Q3' * RAS' + PHASE0 * Q3 zu sein, d.h. die letzte 14M-Periode von PHASE1 und die ersten vier 14M-Perioden von PHASE0.
7. RA4 für die COLUMN-Adresse besitzt eine spezielle Logik für die Schaltung zwischen Bank1 und Bank2 des hohen RAMs.
8. ROMEN1', ROMEN2', EN80', KBD' und MD IN/OUT' werden alle über PHASE0 freigegeben. CASEN' wird ebenfalls über PHASE0 gesteuert - allerdings nicht in der MMU, sondern im HAL.
9. Alle Signale, mit denen Speicherbausteine aktiviert werden, d.h. ROMEN1', ROMEN2', EN80' und CASEN', sind über INHIBIT' steuerbar und können von einer Zusatzkarte abgeschaltet werden.
10. CASEN' und EN80' schließen sich gegenseitig aus. Bei einem RAM-Zugriff wird immer nur eines dieser beiden Signale aktiv.
11. KBD' wird über RC000-RC01F aktiviert - und nicht über RC000-RC00F, wie zu erwarten gewesen wäre.

Bild 5.13a Die MMU

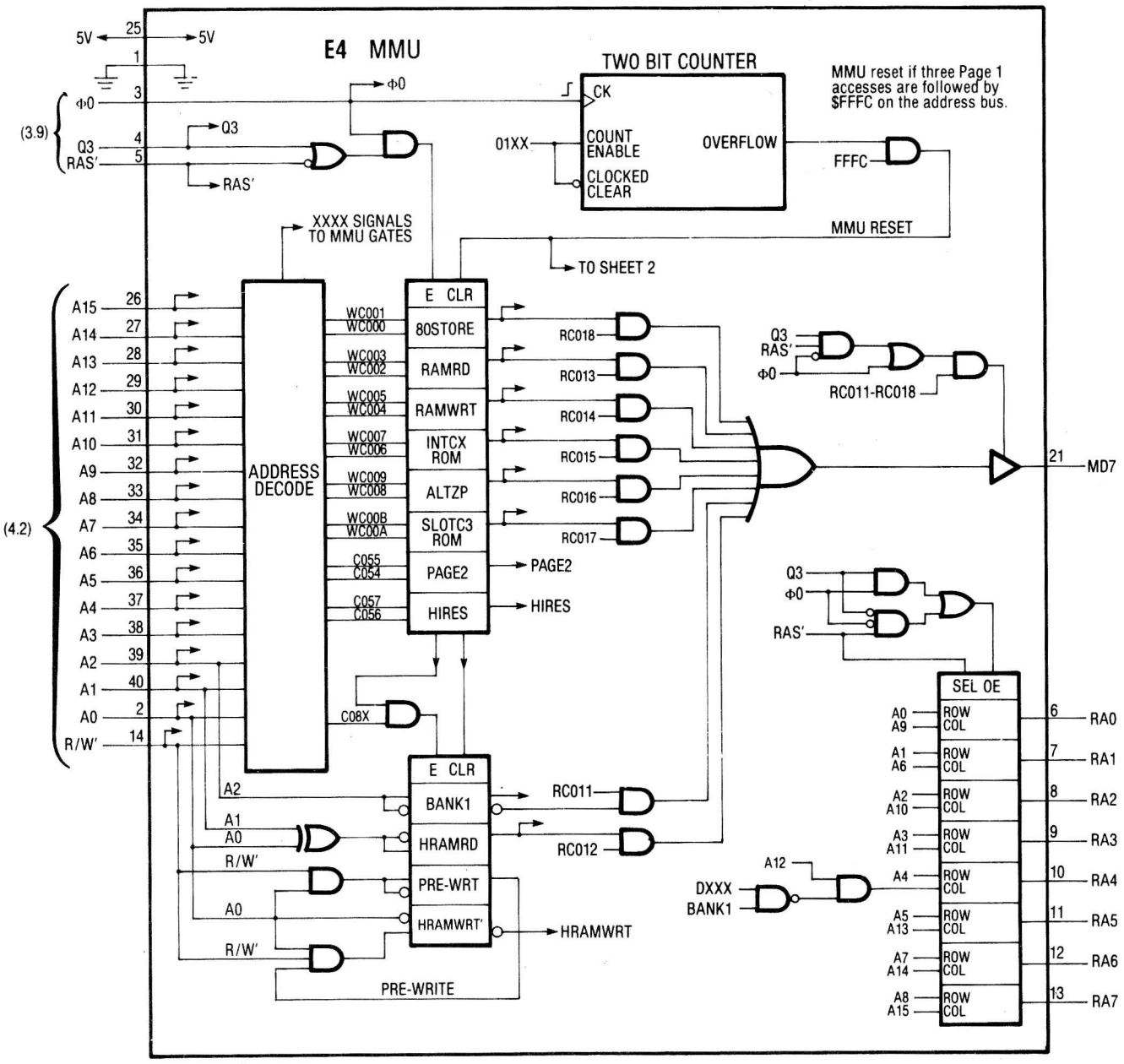
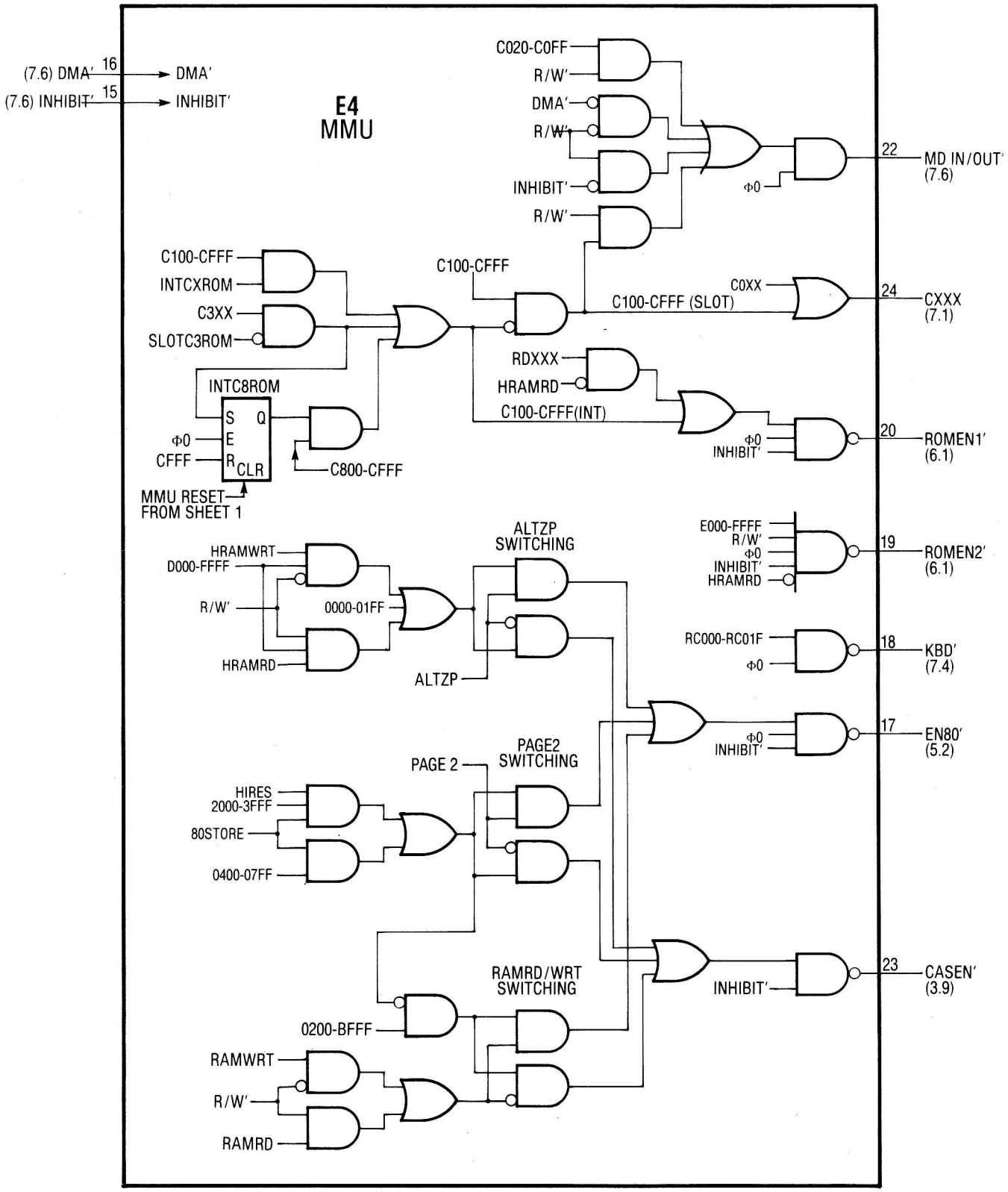


Bild 5.13b Die MMU



Hinweis:
(1) Die Adreßbereichssignale sind über den Inhalt des Adreßbusses erzeugt.

Tabelle 5.6 Datenbus-Steuersignale der MMU

ADDRESS RANGE	R/W'	RAMRD	RAMWRT	RHAMRD	HRAMWRT'	80STORE	HIRES	PAGE2	ALTZP	INTCXRM	ISLTC3M	INTC8M	PHASE	INHIBIT'	CAS EN'	ROM EN0'	ROM EN1'	ROM EN2'	CXXX'	KBD'
										INPUTS					**DATA BUS GATING SIGNALS**					
$0000-$01FF	.	.	.	.	.	.	.	.	0	.	.	.	.	H	**L**	H	H	H	L	H
	.	.	.	.	.	.	.	.	1	.	.	.	H	H	H	**L**	H	H	L	H
TABLE A	R	0	.	.	.	.	.	.	.	.	.	.	.	H	**L**	H	H	H	L	H
$0200-$03FF,	R	1	.	.	.	.	.	.	.	.	.	.	H	H	H	**L**	H	H	L	H
$0800-$1FFF,	W	.	0	.	.	.	.	.	.	.	.	.	.	H	**L**	H	H	H	L	H
$4000-$BFFF	W	.	1	.	.	.	.	.	.	.	.	.	H	H	H	**L**	H	H	L	H
$0400-$07FF	.	.	.	.	.	0	.	.	.	.	.	.	.	.	SEE TABLE A					
	.	.	.	.	.	1	.	0	.	.	.	.	.	H	**L**	H	H	H	L	H
	.	.	.	.	.	1	.	1	.	.	.	.	H	H	H	**L**	H	H	L	H
$2000-$3FFF	.	.	.	.	.	0	.	.	.	.	.	.	.	.	SEE TABLE A					
	.	.	.	.	.	.	0	.	.	.	.	.	.	.	SEE TABLE A					
	.	.	.	.	.	1	1	0	.	.	.	.	.	H	**L**	H	H	H	L	H
	.	.	.	.	.	1	1	1	.	.	.	.	H	H	H	**L**	H	H	L	H
$C000-$C01F	R	.	.	.	.	.	.	.	.	.	.	.	*	.	H	H	H	H	**H**	**L**
	W	.	.	.	.	.	.	.	.	.	.	.	.	.	H	H	H	H	**H**	H
$C020-$C0FF	.	.	.	.	.	.	.	.	.	.	.	.	.	.	H	H	H	H	**H**	H
$C100-$C2FF,	.	.	.	.	.	.	.	.	.	0	.	.	.	.	H	H	H	H	**H**	H
$C400-$C7FF	.	.	.	.	.	.	.	.	.	1	.	.	H	H	H	H	**L**	H	L	H
$C300-$C3FF	.	.	.	.	.	.	.	.	.	0	1	.	.	.	H	H	H	H	**H**	H
	.	.	.	.	.	.	.	.	.	1	.	.	H	H	H	H	**L**	H	L	H
	.	.	.	.	.	.	.	.	.	.	0	.	H	H	H	H	**L**	H	L	H
$C800-$CFFF	.	.	.	.	.	.	.	.	.	0	.	0	.	.	H	H	H	H	**H**	H
	.	.	.	.	.	.	.	.	.	1	.	.	H	H	H	H	**L**	H	L	H
	.	.	.	.	.	.	.	.	.	.	.	1	H	H	H	H	**L**	H	L	H
$D000-$DFFF	R	.	.	0	.	.	.	.	.	.	.	.	H	H	H	H	**L**	H	L	H
$E000-$FFFF	R	.	.	0	.	.	.	.	.	.	.	.	H	H	H	H	H	**L**	L	H
$D000-$FFFF	R	.	.	1	.	.	.	.	0	.	.	.	.	H	**L**	H	H	H	L	H
	R	.	.	1	.	.	.	.	1	.	.	.	H	H	H	**L**	H	H	L	H
	W	.	.	.	0	.	.	.	0	.	.	.	.	H	**L**	H	H	H	L	H
	W	.	.	.	0	.	.	.	1	.	.	.	H	H	H	**L**	H	H	L	H
**	.	.	.	.	.	.	.	.	.	.	.	.	.	.	H	H	H	H	L	H

Legende:
R - Lesen
W - Schreiben
1 - Softswitch gesetzt
0 - Softswitch zurückgesetzt
H - Signal "1"
L - Signal "0"

Anmerkungen:
* KBD' wird über PHASE0 gesteuert, CXXX nicht.
** Alle nicht gezeigten Kombinationen haben das Ergebnis "inaktiv". Fett gedruckte Signale sind aktiv.

12. MD IN/OUT' wird bei jedem Lesevorgang "1", während INHIBIT' aktiv ist, der Bereich $C000-$C01F ist nicht ausgeschlossen.

13. Wenn der Bereich $C100-$CFFF auf "intern" geschaltet ist, wird ROMEN1' für Adressen in diesem Bereich bei Schreibzugriffen aktiv. Das Ergebnis ist eine Kollision zwischen den Datenausgängen der CPU und denen der ROMs - wer dabei auf Dauer den Sieg davonträgt, ist ungewiß. Diese Konstellation ist die einzige, bei der einer der ROMs durch einen Schreibzugriff aktiviert werden kann.

Der logische Aufbau der Kontrollsignale für den Datenbus entspricht den in diesem Kapitel beschriebenen Funktionen zur Verwaltung des Speichers. Bild 5.13b gibt diese Logik in Form eines Schaltplans wieder, die Tabellen 5.6 und 5.7 haben denselben Inhalt und sind für Leser gedacht, denen diese Darstellungsart mehr zusagt.

Tabelle 5.7 Die Steuerung von MD IN/OUT'

ADDRESS RANGE	R/W'	DMA'	INH'	INT CXRM	SLOT C3RM	INT C8RM	PHS 0	MD IN/OUT'
$0000—$FFFF	W	L	-	-	-	-	H	H
$0000—$FFFF	R	-	L	-	-	-	H	H
$C020—$C0FF	R	-	-	-	-	-	H	H
$C100—$C2FF, $C400—$C7FF	R	-	-	0	-	-	H	H
$C300—$C3FF	R	-	-	0	1	-	H	H
$C800—$CFFF	R	-	-	0	-	0	H	H

Legende:
R - Lesen
W - Schreiben
1 - Softswitch gesetzt
0 - Softswitch zurückgesetzt
H - Signal "1"
L - Signal "0"

Hinweis:
Alle nicht gezeigten Kombinationen haben den Pegel "0", d.h. den Datenfluß von der CPU zu den Steckplätzen zur Folge. Wenn MD IN/OUT' den Pegel "1" hat, empfängt die CPU Daten von den Steckplätzen.

Laufzeiten in der MMU

Die MMU und die IOU sind wie auch der 6502, der RAM und der ROM Chips, die in MOS-Technik hergestellt werden. Mit diesem Verfahren sind hohe Integrationsdichten möglich, dafür sind die Schaltungen im Vergleich zu bipolaren ICs wie den restlichen TTL-Bausteinen und dem HAL reichlich langsam. Wo bei einem LSTTL-Gatter mit einer Verzögerung im Bereich von 5 bis 25 Nanosekunden zu rechnen ist, muß bei einem MOS-Chip mit Zeiten zwischen 30 und 125 ns gerechnet werden. Diese Schätzungen sind reichlich vage - ich möchte Ihnen damit auch nur das Gefühl vermitteln, daß die Geschwindigkeitsgrenzen des Apple fast ausschließlich von den MOS-Chips gesetzt werden.

Apple, Inc. hat die Spezifikationen der MMU nicht zur Veröffentlichung freigegeben. Wenn wir sie sehen könnten, fänden wir wahrscheinlich minimale und maximale Zeiten, die auf die steigenden und fallenden Flanken von PHASE0, Q3, RAS' und des Adreßbusses bezogen wären. Ich habe mir die MMU-Signale meines Apple //e mit einem Oszilloskop etwas genauer angesehen und dabei durchschnittliche Laufzeiten zwischen 40 und 100 Nanosekunden gefunden. Über den Daumen gepeilt, liegt die Laufzeit eines Signals in der MMU in der Größenordnung einer 14M-Periode.

Eine MMU-Spezifikation, die ich gerne von Apple, Inc. veröffentlicht gesehen hätte, ist die Maximalzeit zwischen gültiger Adresse auf dem Adreßbus und Ausgabe der gemultiplexten RAM-Adresse. Diese Angabe bestimmt, in welcher Zeit eine mit DMA arbeitende Zusatzkarte eine Adresse ausgeben muß, damit sie bei der fallenden Flanke von RAS' korrekt als ROW-Adresse zu den RAMs als auch an der IOU(!) ankommt. Die entsprechende Zeit liegt in meinem Apple //e bei 82 Nanosekunden. Ich schätze, daß ein DMA-Zugriff korrekt ausgeführt wird, wenn eine ausgegebene Adresse vor oder während der steigenden Flanke von RAS' innerhalb von PHASE1 stabil ist. Allerdings habe ich gehört, daß Apple, Inc. die MMU nur für den "worst case" des 6502A prüft. Die Folge wäre, daß ein DMA-Zugriff im Apple //e nur dann mit Sicherheit funktioniert, wenn eine ausgegebene Adresse spätestens 210 Ns nach der fallenden Flanke von PHASE0 stabil ist (5 ns für den LS02 plus 65 ns zwischen PHASE1 und PHASE2

des 6502 plus 140 ns, die der 6502 selber für das Setzen einer Adresse braucht). Das wäre allerdings äußerst konservativ und unrealistisch - die dabei veranschlagten 349 ns Laufzeit hat die MMU garantiert nicht.

Das "Timing" des RAMs im Apple //e

Die meisten Aspekte dieses Themas sind bereits im Rahmen anderer Diskussionen abgehandelt worden. Die Absicht dieses Abschnitts ist eine Zusammenfassung und die Erklärung der fehlenden Details.

Bild 5.14 gibt die zeitlichen Abläufe der RAM-Signale der Hauptplatine wieder. Es zeigt einen Lese- und einen Schreibzugriff des 6502 sowie den dazwischenliegenden Zugriff des Videoscanners. Alle RAM-Zugriffe werden über RAS' und CAS' gesteuert - darüber ist die Zugriffsrate von 2 MHz festgelegt. Auch wenn die 200ns-RAM-Bausteine des Apple mit wesentlich höheren Zugriffsraten betrieben werden könnten, kann auch kein Baustein via DMA' schneller darauf zugreifen, es sein denn, er ersetzt die Signale CAS' und RAS' ebenfalls.[11] RAS' zeigt die Gültigkeit der ROW-Adresse an, CAS' die Gültigkeit der COLUMN-Adresse und den Beginn des Datentransfers.

Der größte Teil der Abläufe ist recht geradlinig. Vom 6502 ausgegebene Daten sind mit Sicherheit auf dem Bus stabil, wenn CAS' während PHASE0 fällt; vom RAM ausgegebene Daten sind mit Sicherheit stabil, wenn PHASE0 steigt und die Videolatches sie festhalten bzw. wenn PHASE2 innerhalb des 6502 steigt und der Prozessor sie übernimmt.

Was nicht so geradlinig funktioniert, ist ein Detail in der Datenausgabe durch den RAM: jeder RAM-Baustein muß ausgegebene Daten stabil halten, wenn CAS' wieder steigt und die CPU sie liest. Es ist sehr stark anzunehmen, daß die RAM-Bausteine ausgegebene Daten bestenfalls bis zur fallenden Flanke von PHASE2 halten - die restliche Zeit wird der Datenbus bei einer Leseaktion des Prozessors von keinem einzigen aktiven Sender versorgt. Hier spielt die fehlende Terminierung wieder eine Rolle: der Bus geht in den hochohmigen Zustand über und hält seine Daten mit Sicherheit lange genug, um einen stabile Übermittlung zum Prozessor zu gewährleisten.

Während eines Schreibzyklus des Prozessors greift der Videoscanner in PHASE1 genauso wie sonst auch auf den RAM zu - allerdings gelangen die vom RAM ausgegebenen Videodaten nicht zu den Steckplätzen wie sonst üblich.

Die Reihenfolge der wichtigen Ereignisse und Signale sieht folgendermaßen aus:

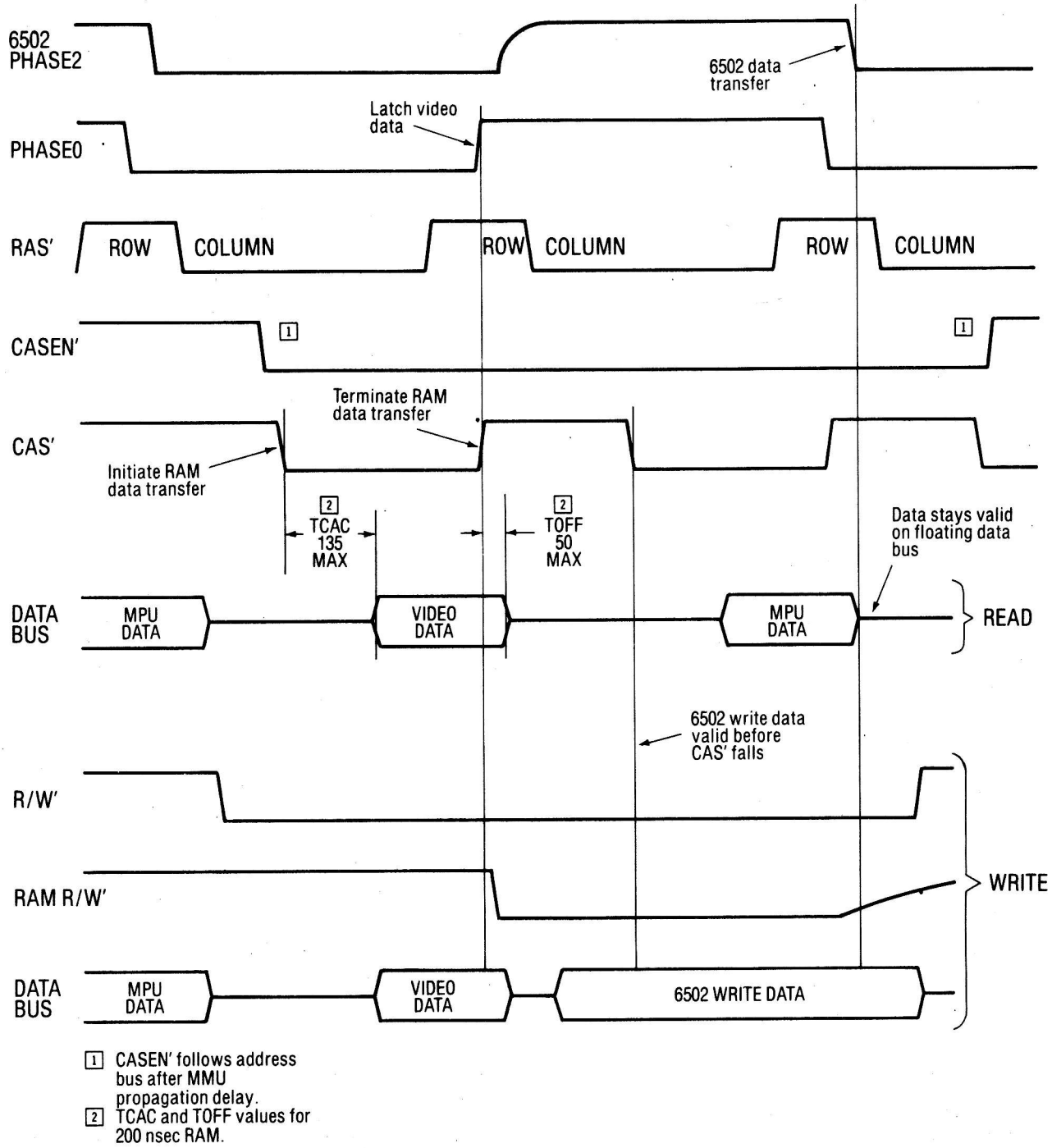
1. RAS' steigt kurz vor Ende von PHASE0 wieder. Nach einer durch die Laufzeiten der IOU bedingten Verzögerung wird damit die ROW-Adresse des Videoscanners auf den gemultiplexten RAM-Adreßbus geschaltet. Die Verzögerung ist IOU-spezifisch, die ROW-Adresse muß aber gültig sein, bevor RAS' wieder fällt.
2. PHASE2 fällt, der Prozessor übernimmt bei Lesezyklen die auf dem Datenbus anliegenden Pegel.
3. RAS' fällt, die RAM-Bausteine übernehmen die anliegende ROW-Adresse des Videoscanners. Nach einer durch die Laufzeiten der IOU bedingten Verzögerung wird damit die COLUMN-Adresse des Videoscanners auf den gemultiplexten RAM-Adreßbus geschaltet. Die Verzögerung muß kurz genug sein, daß die COLUMN-Adresse vor dem Fallen von CAS' gültig ist.
4. Bereits zu Beginn von PHASE1 ist die nächste vom 6502 ausgegebene Adresse gültig. Die MMU übernimmt diese Adresse, wartet aber während PHASE1 auf die steigende Flanke von RAS'. Der Pegel der Leitung R/W' Control wird dagegen erst zu Beginn der nächsten PHASE0 zu den RAM-Bausteinen durchgeschaltet.
5. CAS' fällt und die RAM-Bausteine übernehmen die COLUMN-Adresse des Videoscanners. Diese fallende Flanke ist auch gleichzeitig das Startsignal für die Datenausgabe der RAMs.
6. Die vom RAM ausgegebenen Daten werden gültig. Die Zeit zwischen Punkt 5 und Punkt 6 hängt vom Typ der RAM-Bausteine ab. Die normalerweise verwendeten 200 ns-RAMs benötigen maximal 135 ns von der fallenden Flanke von CAS' bis zur Gültigkeit der ausgegebenen Daten. Die Daten sind in jedem Fall vor der steigenden Flanke von PHASE0 gültig.
7. RAS' steigt wieder und schaltet die ROW-Adresse der MMU (nach der Laufzeit der MMU) auf den gemultiplexten RAM-Adreßbus. Der noch nicht abgeschlossene Videolesezyklus wird dadurch nicht gestört, weil die RAM-Bausteine anliegende Adressen erst mit der fallenden Flanke von RAS' übernehmen.

---

11 Schneller kann man nicht darauf zugreifen, aber bei gleicher Taktfrequenz "ungünstiger" arbeiten, d.h. mit einer kürzeren Zeit zwischen "Adresse gültig" und "Daten einlesen", wie es z.B. der Z80 tut.

8. PHASE0 und CAS' steigen im selben Moment. Auf die steigende Flanke von PHASE0 übernehmen die Videolatches die Videodaten, über PHASE0 wird danach R/W' zu den RAMs durchgeschaltet. Die steigende Flanke von CAS' bewirkt nach einer Verzögerung den Übergang der RAM-Ausgänge in den hochohmigen Zustand. Bei RAMs mit 200ns liegt diese Verzögerung bei maximal 50 ns.
9. Während PHASE0 adressiert die CPU über die MMU den RAM in derselben Weise, wie es der Videoscanner während PHASE1 getan hat: der Pegel von RAS' selektiert ROW/COLUMN, auf die fallenden Flanken von RAS' bzw. CAS' werden die Adressen von den RAM-Bausteinen übernommen. Der wirklich einzige Unterschied bis zu diesem Punkt liegt darin, daß nicht die IOU, sondern die MMU die Adressen ausgibt. Die vom RAM ausgegebenen Daten sind auf jeden Fall vor der steigenden Flanke von PHASE1 gültig.

Bild 5.14 Zeitdiagramm der Zugriffe auf den RAM der Hauptplatine



1 CASEN' follows address bus after MMU propagation delay.
2 TCAC and TOFF values for 200 nsec RAM.

10. In Schreibzyklen müssen vom 6502 ausgegebene Daten gültig sein, bevor CAS' während PHASE0 fällt. Die Daten bleiben über die fallende Flanke von PHASE2 hinaus gültig. Dieser Ablauf wird in der Literatur als "early write cycle" (= früher oder vorgezogener Schreibzyklus) bezeichnet. R/W' fällt vor CAS', zu schreibende Daten müssen gültig sein, bevor CAS' fällt.

11. PHASE0 fällt im selben Moment, in dem CAS' steigt. Wenn es sich bei dem Zyklus um einen Schreibzyklus gehandelt hat, steigt der Pegel von R/W' ebenfalls wieder - er braucht aber verhältnismäßig lange dazu, weil er lediglich über einen Widerstand mit 1 kOhm auf "1" gezogen wird.
Wenn der Zyklus ein Lesezyklus war, dann schaltet die steigende Flanke von CAS' die Ausgänge der RAMs nach einer Verzögerung von rund 50ns in den hochohmigen Zustand. Der Datenbus kann damit vor der fallenden Flanke von PHASE2 komplett im hochohmigen Zustand sein, wegen der fehlenden Terminierung liest der Prozessor die Daten trotzdem korrekt ein.

12. Mit der fallenden Flanke von PHASE2 übernimmt der Prozessor den Inhalt des Datenbusses, wenn es sich bei diesem Zyklus um einen Lesezyklus gehandelt hat.

Bild 5.15 zeigt denselben Ablauf für den AUX-RAM unter der Voraussetzung, daß der AUX-RAM ebenfalls aus 64 kByte dynamischem RAM besteht. Die durch RAS' und CAS' ausgelösten Reaktionen eines RAM-Bausteins im AUX-RAM sind natürlich identisch mit denen des RAMs der Hauptplatine, die Details des Zeitablaufs sind aber verschieden, weil anstelle von CAS' das Signal Q3 für die COLUMN-Adresse benutzt wird. Außerdem enthält die Zusatzkarte für den speziellen Steckplatz einen bidirektionalen Treiber für den Datenbus.

Die Erzeugung der gemultiplexten RAM-Adressen verläuft für den RAM der Hauptplatine und den AUX-RAM gleich - der gemultiplexte RAM-Adreßbus ist durchverbunden. Der einzige Unterschied besteht in der Übernahme der COLUMN-Adresse durch die fallende Flanke von Q3.

Wenn eine RAM-Karte in Steckplatz 3 installiert ist, wird sie automatisch mit durch den Videoscanner adressiert. Die vom AUX-RAM als Reaktion auf die Videoscanner-Adresse ausgegebenen Daten werden mit der steigenden Flanke von PHASE0 im AUX-Videolatch gespeichert. Der Videoscanner gibt während jeder PHASE1 eine Adresse aus - egal, ob die CPU während dieses Zyklus schreibt, liest oder den RAM überhaupt nicht benutzt.

Die Reihenfolge wichtiger Signale und Ereignisse in Bild 5.15 läßt sich folgendermaßen darstellen:

1. PHASE2 fällt und beendet damit den Datentransfer des vorherigen Zyklus.

2. Während PHASE1 gibt die IOU die ROW- und COLUMN-Adresse des Videoscanners aus wie bei Bild 5.14 beschrieben. Die ROW-Adresse wird auf die fallende Flanke von RAS', die COLUMN-Adresse auf die fallende Flanke von Q3 übernommen.

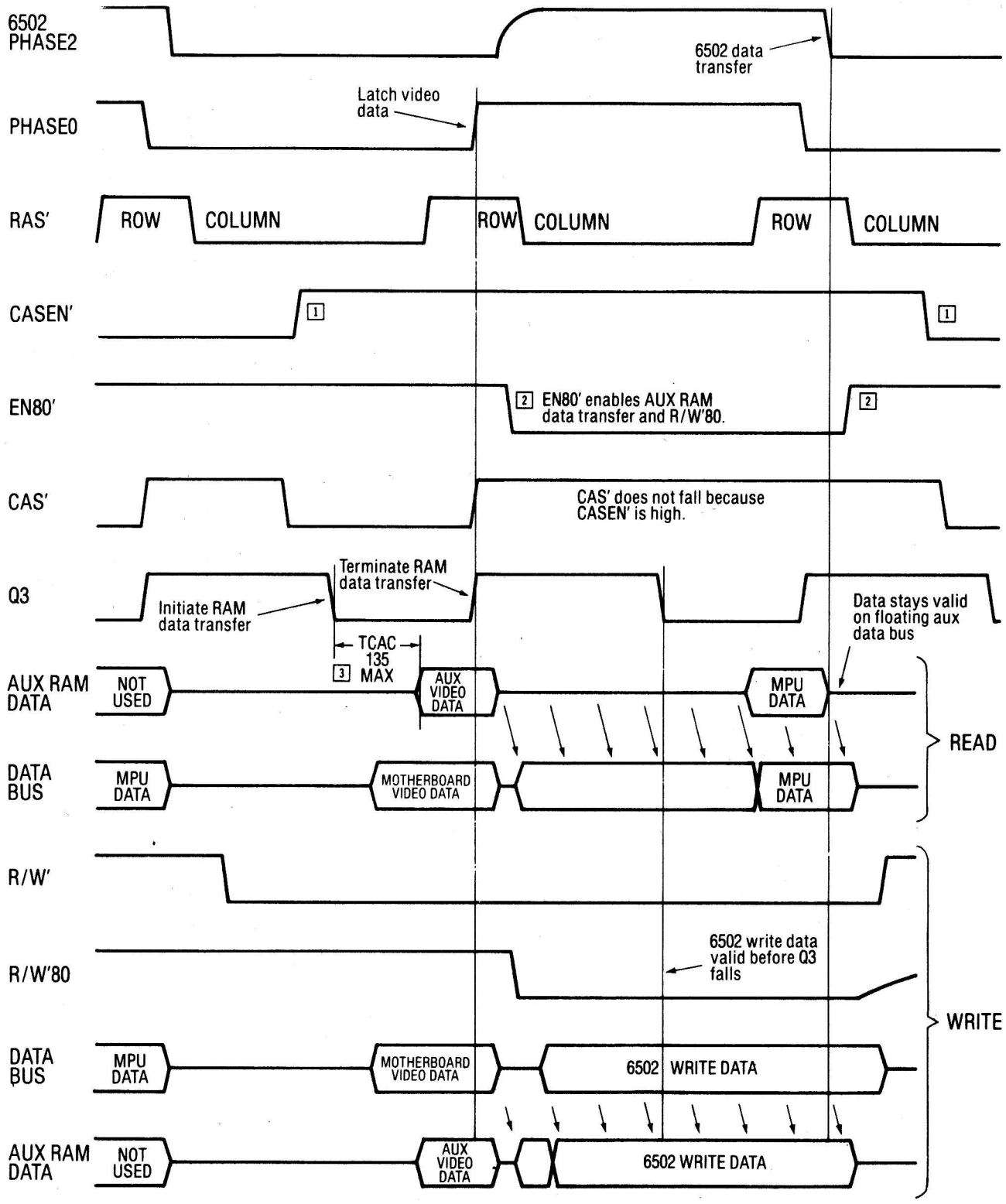
3. Bereits zu Beginn von PHASE1 ist die vom 6502 ausgegebene Adresse auf dem Adreßbus zusammen mit R/W' gültig. Der AUX-RAM wird zu diesem Zeitpunkt nicht davon beeinflußt, R/W' wird über EN80' geschaltet und als Signal R/W'80 weitergeführt. EN80' wird über PHASE0 gesteuert und ist deshalb während PHASE1 immer inaktiv.
Über R/W'80 wird auch die Richtung der bidirektionalen Bustreiber des AUX-RAMs gesteuert. Diese Treiber liegen momentan als "Empfänger" am Datenbus der Hauptplatine, ihre Ausgänge zum AUX-Datenbus sind im hochohmigen Zustand.

4. Maximal 135 ns nach der fallenden Flanke von Q3 (mit der die COLUMN-Adresse von den RAMs übernommen wurde), sind die Videodaten auf dem AUX-Datenbus gültig. Q3 für den AUX-RAM kommt rund 70 ns nach CAS' für den RAM der Hauptplatine, die Daten sind aber immer noch auf jeden Fall gültig, bevor PHASE0 steigt. Die Daten landen nicht auf dem Datenbus der Hauptplatine, weil EN80' nach wie vor den Pegel "1" hat und die Ausgänge des bidirektionalen Bustreibers im hochohmigen Zustand hält.

5. PHASE0 und Q3 steigen zusammen. Auf die steigende Flanke von PHASE0 übernehmen die Videolatches auf der Hauptplatine und auf der Zusatzkarte die Videodaten. Die Bausteine des AUX-RAMs bringen ihre Datenausgänge maximal 50 ns danach wieder in den hochohmigen Zustand (200 ns-Typen).

6. Über den Pegel "1" von PHASE0 schaltet die MMU nach entsprechender Laufzeit EN80' auf "0". EN80' wiederum schaltet R/W' Control als R/W'80 zum AUX-RAM durch, wo hierüber auch noch die Richtung des bidirektionalen Treibers bestimmt wird. EN80' schaltet die Ausgänge des Treibers in den aktiven Zustand. Je nach Richtung des geplanten Datentransfers erhält entweder der AUX-Datenbus erst einmal den Inhalt des (hochohmigen) Datenbusses der Hauptplatine oder umgekehrt.

Bild 5.15 Zeitdiagramm der Zugriffe auf den 64k AUX-RAM



1 CASEN' follows address bus after MMU propagation delay.
2 EN80' follows PHASE0 after MMU propagation delay.
3 TCAC is 135 nsec MAX with 200 nsec RAM chips.

7. Während PHASE0 wird die MMU die von der CPU ausgegebene Adresse auf den gemultiplexten RAM-Adreßbus legen. Es spielt dabei keine Rolle, welchen Baustein die CPU eigentlich adressiert - die RAMs bekommen immer eine Adresse. Auch der AUX-RAM bleibt davon nicht verschont. Falls die CPU überhaupt nichts vom AUX-RAM will, dann bleibt EN80' auf dem Pegel "1". Damit bleibt R/W'80 ebenfalls auf "1" und die bidirektionalen Treiber hochohmig. Der Überflüssige Zugriff auf den AUX-RAM wird damit zu einem Lesezugriff hinter verschlossenen Türen, die CPU bekommt von den ausgegebenen Daten schließlich auch nichts zu sehen.
8. Wenn die CPU einen Schreibzugriff auf den AUX-RAM ausführt, dann werden die vom 6502 ausgegebenen Daten am Anfang von PHASE0 gültig. Der bidirektionale Treiber verzögert die Daten um rund 25 ns, sie müssen an den RAMs gültig anliegen, bevor Q3 fällt. Das ist allerdings erst 70 ns nach CAS' der Fall und damit noch unkritischer als beim RAM der Hauptplatine.
9. Wenn die CPU einen Lesezugriff auf den AUX-RAM ausführt, dann werden die Daten auf dem AUX-Datenbus maximal 135 ns nach der fallenden Flanke von Q3 gültig und über den bidirektionalen Bustreiber um weitere 25 ns verzögert. Sie sind aber auf dem Datenbus der Hauptplatine mit Sicherheit vor der fallenden Flanke von PHASE2 verfügbar.
10. PHASE2 fällt im selben Moment, in dem Q3 steigt. Die Bausteine des AUX-RAMs bringen ihre Datenausgänge innerhalb der nächsten 50 ns in den hochohmigen Zustand. Der gesamte AUX-Datenbus ist damit im hochohmigen Zustand und wird die RAM-Daten über eine verhältnismäßig lange Zeit halten. Der bidirektionale Treiber liefert diese Daten auch dann noch korrekt zum Datenbus der Hauptplatine, wenn die Datenausgänge der RAMs bereits im hochohmigen Zustand sind.
11. Der 6502 übernimmt bei einem Lesezyklus die Daten vom AUX-RAM mit der fallenden Flanke von PHASE2. Bei einem Schreibzyklus kontrolliert der Prozessor den Datenbus ein kurzes Stück über die fallende Flanke von PHASE2 hinaus.
12. Der Pegel "0" von PHASE0 bewirkt nach einer Laufzeit innerhalb der MMU, daß EN80' in jedem Fall inaktiv gesetzt wird. Dadurch wird der bidirektionale Treiber abgeschaltet, bei einem Schreibzyklus wird zusätzlich die Leitung R/W'80 damit wieder auf den Pegel "1" gebracht. Diese Leitung bewegt sich verhältnismäßig langsam von "0" auf "1", weil sie lediglich von einem Widerstand mit 1 kOhm auf "1" gezogen wird.

Die 80-Zeichen-Karte mit 1 kByte RAM

Ein guter Teil der Schaltkreise in der MMU ist mit der Verwaltung der insgesamt möglichen 128 kByte RAM beschäftigt. Die 80-Zeichen-Ausgabe ist ebenfalls in den Zeitabläufen der Hauptplatine und in der Firmware bereits vorgesehen - so komplett, daß eine 80-Zeichen-Karte aus den acht RAM-Bausteinen, dem Latch und einem Treiber besteht.

Apple, Inc. hat noch eine weitere 80-Zeichen-Karte entwickelt, die nur 1 kByte statischen RAM enthält. Dieser RAM liegt für den Prozessor im (AUX-)Bereich von $400-$7FF und verhält sich für Prozessor und Programm in derselben Weise wie der entsprechende RAM-Bereich der "erweiterten" 80-Zeichen-Karte mit 64 kByte RAM. Bild 5.16 gibt den Schaltplan dieser Karte wieder. Die Zeichnung ist von mir durch Verfolgen der Leiterbahnen der Karte erstellt worden - ein "offizieller" Schaltplan war nicht aufzutreiben.

Diese Karte ist ihrem "großen Bruder" auch sonst recht ähnlich, für die Verbindung zum Datenbus der Hauptplatine und das Videolatch wurden sogar dieselben Bausteine verwendet. Auch in der Kontrolle gibt es keine Unterschiede, die entsprechenden Signale sind:

Kontrollsignal	**Funktion**
fallende Flanke von RAS'	Übernahme der ROW-Adresse
Q3 auf "0"	Übernahme der COLUMN-Adresse und Beginn des Datentransfers
R/W'80	Schreib/Lesekontrolle
steigende Flanke PHASE0	Speichern der Videodaten
PHASE1 = "0"	Abgabe der Videodaten zum Videobus
R/W'	Richtungskontrolle des LS245
EN80'	Ausgangskontrolle des LS245

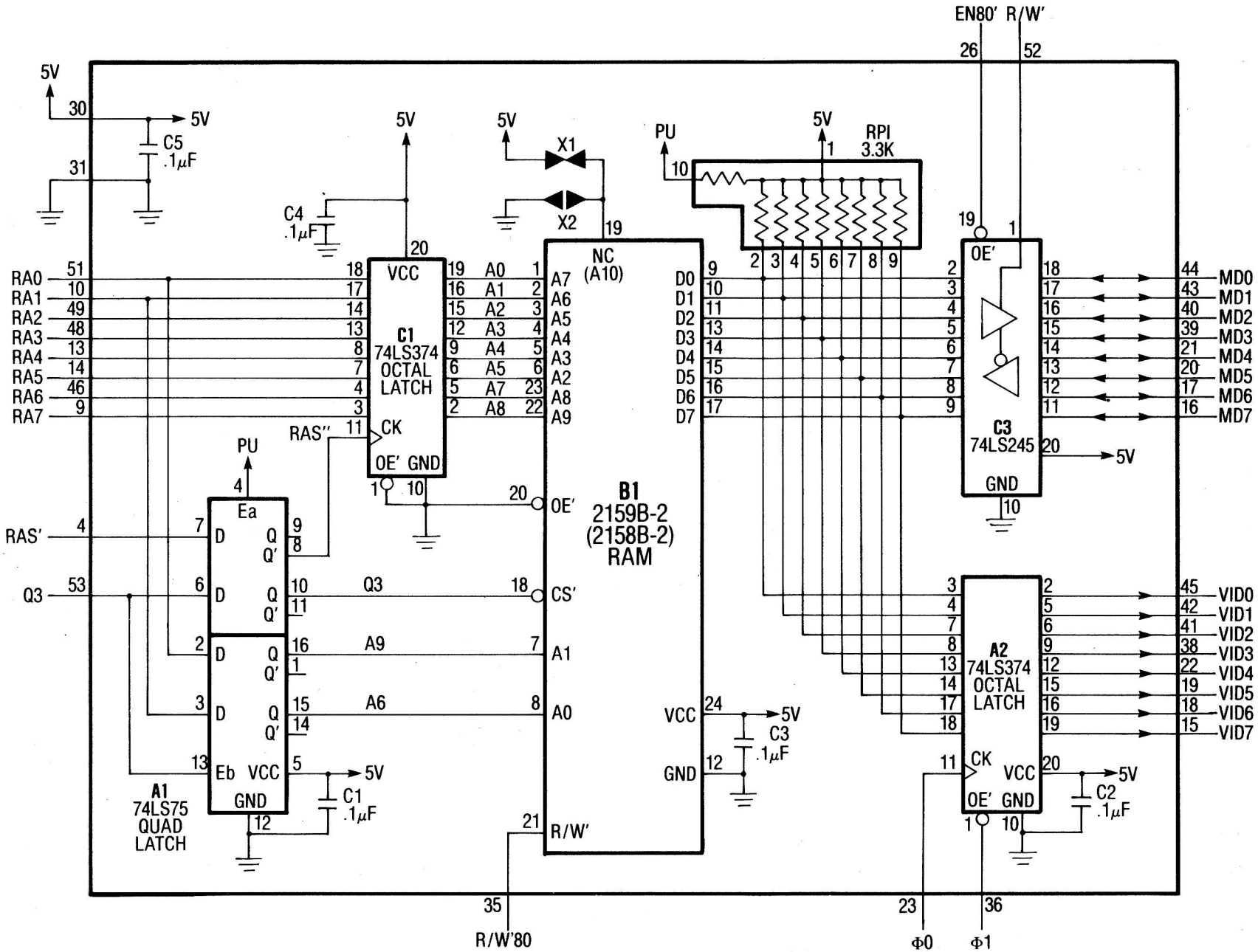


Bild 5.16 Schematische Darstellung des 1k-AUX-RAM

Diese Kontrollfunktionen ergeben sich aus den zeitlichen Abläufen des RAMs der Hauptplatine, des Videoscanner-Zugriffs während PHASE1 etc.

Ein sehr interessantes "Feature" der 80-Zeichen-Karte mit 1 kByte RAM ist, daß sie auf EN80' reagiert, ohne Rücksicht darauf, ob die Adresse im Bereich $0400-$07FF liegt. Damit kann man z.B. RAMWRT setzen, einen Wert auf der Speicherstelle $8213 abspeichern und ihn von der Speicherstelle $613 wieder lesen. Im Prinzip ist das kein Problem, weil alle Programme erst eine Prüfung vornehmen (sollten!), bevor sie den AUX-RAM außerhalb des Bereichs $0400-$07FF benutzen. Es macht diesen Test nur etwas schwieriger, weil bei beiden Karten eine Speicherung auf eine beliebige Adresse und anschließendes Zurücklesen möglich ist. Der Unterschied liegt darin, daß eine 1k-Karte denselben Wert mehrfach zurückliefert, und zwar immer in Abständen von jeweils $400. Das folgende Programmbeispiel prüft, ob der Computer über eine 64k-Karte im speziellen Steckplatz verfügt:

```
TEST64K:    STA  $C001        ; setzt 80STORE
            LDA  $C057        ; setzt HIRES
            LDA  $C055        ; setzt PAGE2
            LDA  #$00
            STA  $0400
            LDA  #$88
            STA  $2000
            CMP  $0400        ; $400 verändert?
            BEQ  NOT64
            CMP  $2000        ; Wert behalten?
            BNE  NOT64
            CMP  $2000        ; 12
            BEQ  GOT64        ; o.k. 64 k AUX
```

12 Wenn versucht wird, den AUX-RAM zu lesen, ohne daß eine entsprechende Karte im speziellen Steckplatz installiert ist, wird der Prozessor das zuletzt verarbeitete Videodatum des RAMs der Hauptplatine lesen. $88 (d.h. "Pfeil links") wird deshalb als Test verwendet, weil dieser Wert als Videodatum für den TEXT-Bildschirm sehr unwahrscheinlich ist. Ein doppelter Test auf diesen Wert stellt sicher, daß nicht beim ersten Versuch anstelle des AUX-RAMs ein zufällig in den UNUSED 8 gespeicherter Wert gelesen wurde. Alle Testprogramme auf Speicher etc. müssen berücksichtigen, daß der Prozessor bei Nichtvorhandensein des Bausteins in den meisten Fällen Videodaten zurückliefert.

Kapitel 6

ROM im Apple //e

Die Vielfalt der als Festwertspeicher verwendeten Bauteile in der Geschichte der Computer ist - wie diese selbst - recht beeindruckend, sie reicht von Vakuumröhren und Dioden über Chips mit niedriger Integrationsdichte bis zu den heute verfügbaren ROMs, die bis zu 512 kBit auf einem einzigen Chip unterbringen. Je größer die möglichen Kapazitäten, desto mehr Anwendungen werden in ROMs hineingepackt - es gibt mittlerweile Computer, bei denen sowohl das Betriebssystem als auch Anwenderprogramme wie Textverarbeitung und/oder Compiler in ROMs gespeichert sind.

Der Apple //e ist für 16128 Byte ROM auf der Hauptplatine ausgelegt, denen der Adreßbereich von $C100-$FFFF zugeordnet ist. Zusätzliche Schaltungen ermöglichen die Kontrolle des Computers durch Programme, die in ROMs diverser Zusatzkarten gespeichert sind. Dazu gehören die Ansprechmöglichkeit für peripheren ROM über eine steckplatzabhängige Adresse (via $C0nX DEVICE SELECT' und/oder $CnXX I/O SELECT'), die Adressierung im Bereich $C800-$CFFF via I/O STROBE' und schließlich die Möglichkeit, die Bereiche $0000-$BFFF/$C100-$FFFF für die Bausteine der Hauptplatine via INHIBIT' zu sperren. Zusammengenommen kann man die Möglichkeiten der ROM-gesteuerten Kontrolle des Apple nur als "kaum noch zu übertreffen" bezeichnen.

Im Gegensatz zu RAM-Bausteinen und anderen komplizierten Dingen sind ROMs dank der fortgeschrittenen Herstellungstechnik sehr einfach in ein Mikrocomputersystem zu integrieren. Trotzdem bleibt die tatsächliche Ausführung im Apple //e raffiniert genug, um ein eigenes Kapitel zu verdienen. Der größte Teil der folgenden Diskussion beschäftigt sich mit den beiden ROM-Bausteinen $C1-$DF und $E0-$FF sowie möglichen ROMs auf Zusatzkarten, soweit sie ebenfalls Maschinencode für den 6502 enthalten. Die Besprechung von Tastatur- und Video-ROM finden Sie in Kapitel 7 bzw. Kapitel 8.

ROM-Hardware

Die Firmware des Apple //e ist in zwei ROM-Bausteine des Typs 2365A einprogrammiert. Der 2365A ist ein in NMOS-Technik hergestellter Baustein mit 8192 Byte Speicherkapazität und 28 Anschlüssen. Er verfügt über zwei programmierbare "Select"-Eingänge sowie über einen CE'- und einen OE'-Eingang und ist in Zugriffszeiten zwischen 450 und 200 ns erhältlich. (NMOS steht für "Negative channel Metal Oxide Semiconductor" - falls Sie einen äquivalenten Chip in Listen anderer Hersteller suchen, müssen Sie in der Sparte "NMOS-ROMs, 8 * 8192" nachsehen. CE' und OE' stehen für "Chip Enable" und "Output Enable" und sind Eingänge, über die der Chip bzw. dessen Ausgänge aktiviert werden können.)

Es gibt eine ganze Reihe Hersteller von ROMs, die im Apple //e funktionieren würden. 2365A ist die Kennziffer von Synertek und wird hier und im weiteren benutzt, weil sie auch in den von Apple, Inc. publizierten Schaltplänen verwendet wird. Genaue Angaben über die Zugriffsgeschwindigkeit liegen nicht vor, es ist aber anzunehmen, daß ein 450 ns-Typ verwendet wird - schließlich ist das mehr als schnell genug für jede Operation des 6502 im Apple //e.

Beide Bausteine sind ROMs im eigentlichen Sinne, sie werden bereits vom Chip-Hersteller in der programmierten Form geliefert ("Maskenprogrammierung"). Apple, Inc. hat dabei nicht nur die Daten angegeben, die die ROMs enthalten sollen, sondern auch noch spezifiziert, auf welche Pegel die beiden "Select"-Eingänge reagieren sollen (deshalb werden diese beiden Eingänge als "programmierbar" bezeichnet). Diese Eingänge sind bei allen im Apple //e verwendeten Hauptplatinen-ROMs ($C1-$DF, $E0-$FF, Tastatur und Video) so programmiert, daß an ihrer Stelle der Industriestandard von INTEL (2716, 2732, 2764) eingesetzt werden kann.

Bild 6.1 zeigt die Verbindungen der ROMs $C1-$DF und $E0-$FF im Apple //e. Wie auch nicht anders zu erwarten, sind die Anschlüsse beider Bausteine größtenteils einfach durchverbunden, die Leitungen ziehen sich vom jeweiligen Bus über einen Chip zum anderen. Um 8192 verschiedene Adressen darzustellen, werden 13 Adreßleitungen benötigt - die dreizehn Adreßeingänge der ROMs sind direkt mit den Leitungen A0-A12 des

Adreßbusses verbunden. Die acht Datenausgänge haben tri-State-Charakteristik und sind direkt mit dem Datenbus verbunden. Verglichen mit dem Anschlußschema der RAM-Bausteine kann man sich eigentlich nichts einfacheres mehr vorstellen.

Es bleiben nur noch die Stromversorgungseingänge (+5 Volt und Masse) sowie die Aktivierungseingänge (CE', OE' und die beiden "Selects"). Die letzteren haben eine Gemeinsamkeit: Wenn nur einer dieser Eingänge den "falschen" Pegel erhält, bleiben die Ausgänge des ROM-Bausteins im hochohmigen Zustand. Um etwas genauer zu werden: Bevor der $C1-$DF-ROM oder der $E0-$FF-ROM die Kontrolle des Datenbusses übernimmt, müssen die Eingänge CE' und OE' auf den Pegel "0", die beiden "Select"-Eingänge dagegen auf den Pegel "1" gebracht werden. Die "Select"-Eingänge beider ROMs sind im Apple //e direkt mit den +5 Volt der Stromversorgung verbunden - womit ihre Besprechung auch bereits beendet wäre. OE' und CE' verdienen dagegen eine etwas genauere Betrachtung.

CE' unterscheidet sich von den anderen Aktivierungseingängen in einem wesentlichen Punkt - er ist "echt". Durch ihn wird nicht nur ein Ausgang gesperrt, sondern tatsächlich der gesamte Chip in einen Wartezustand ("Stand by") gebracht, in dem nur ein Bruchteil der sonst notwendigen Versorgungsleistung gebraucht wird. Leider benötigt die Aktivierung des gesamten Chips aus diesem Zustand heraus sehr viel Zeit, nämlich 450 ns. Die Aktivierung der Ausgänge allein über OE' dauert dagegen nur 180 ns. Aus diesem Grund werden die ROMs im Apple primär über OE' geschaltet. In der Rev. A der Hauptplatine wurde CE' noch über (INHIBIT' * ENFIRM)' geschaltet, in der Rev. B ist dieser Eingang schlicht mit Masse verbunden und dauernd aktiv.

Auf Hauptplatinen der Rev. A kann eine Zusatzkarte im speziellen Steckplatz den ROM abschalten, indem sie ENFIRM auf den Pegel "0" setzt. Mir ist nicht ganz klar, was Apple, Inc. mit dieser Möglichkeit vorhatte - jedenfalls wurde der Steckkontakt ENFIRM zu FRCTXT', das zur Schaltung von CE' benutzte NAND-Gatter schaltet die Signalfolge des HAL auf TEXT-Videoausgabe (d.h. setzt GATED GR+2' inaktiv), wenn FRCTXT' den Pegel "0" hat (s. Bild 3.9). Daraus ergibt sich die doppelt hochauflösende Grafik, wenn 80COL gesetzt und TEXT zurückgesetzt ist. Der Abschaltmechanismus für die ROMs der Hauptplatine via INHIBIT' ist komplett in die MMU hineinverlegt worden. *Alle Aktivierungs- und Abschaltvorgänge der ROMs $C1-$DF und $E0-$FF finden auf Hauptplatinen der Rev. B über die MMU-Signale ROMEN1' und ROMEN2', also über die ROM-Eingänge OE' statt.*

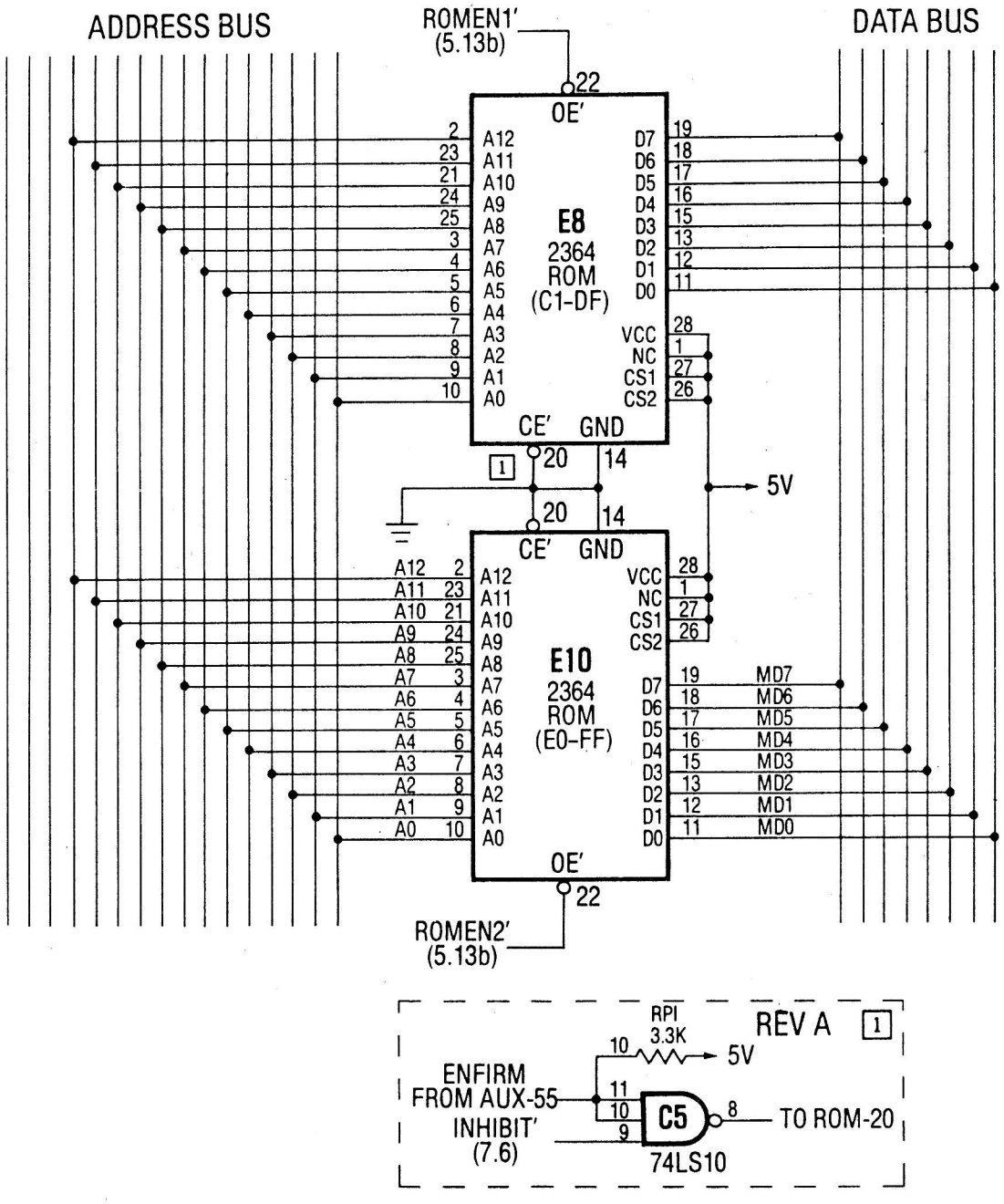
ROMEN1' und ROMEN2'

Über diese beiden Signalleitungen steuert die MMU die Aktivierung der ROMs $C1-$DF bzw. $E0-$FF. Die Erzeugung dieser Signale innerhalb der MMU und die Integration der ROMs in den Adressraum des Apple //e ist in Kapitel 5 beschrieben, wir wiederholen an dieser Stelle nur noch einmal die wichtigsten Punkte:

Der ROM der Hauptplatine wird über den Bereich $C000-$FFFF adressiert, wobei der Adressraum $C100-$CFFF zwischen ROM und Zusatzkarten der Steckplätze, der Adressraum $D000-$FFFF zwischen ROM und dem hohen RAM umgeschaltet werden kann. $D000-$FFFF ist primär für den ROM vorgesehen, jeder RESET'-Impuls schaltet diesen Bereich auf "Lesen vom ROM, Schreiben zum RAM". Durch den Stand des Softswitches HRAMRD wird bestimmt, ob innerhalb der oberen 16k von ROM oder RAM gelesen wird. Wenn HRAMRD zurückgesetzt und INHIBIT' inaktiv ist, reagiert die MMU auf einen Lesezugriff im Bereich $D000-$FFFF damit, daß sie während PHASE0 entweder ROMEN1' ($D000-$DFFF) oder ROMEN2' ($E000-$FFFF) auf den Pegel "0" bringt. Zwischen der steigenden Flanke von PHASE0 und der Aktivierung von ROMEN1' bzw. ROMEN2' liegen wie üblich die Laufzeiten der MMU, dasselbe gilt für den Abschaltvorgang am Ende von PHASE0.

$C100-$CFFF ist primär für die ROMs eingesteckter Zusatzkarten vorgesehen - über das Setzen der Softswitches INTCXROM, SLOTC3ROM und INTC8ROM kann aber eine komplette oder teilweise Umschaltung auf den ROM der Hauptplatine erfolgen. Das ist für einige Funktionen sogar der Regelfall: die Firmware für den 80-Zeichen-Betrieb, die Selbsttests und eine Anzahl von Routinen des Monitors befinden sich im Bereich $C100-$CFFF im ROM. Wenn INHIBIT' inaktiv ist und ein Lesezugriff auf eine Adresse stattfindet, die über die Softswitches als "Hauptplatinen-ROM" konfiguriert ist, dann reagiert die MMU mit einer Aktivierung von ROMEN1' während PHASE0.

Bild 6.1 Schaltplan der ROM-Anschlüsse im Apple //e



(1) Auf Hauptplatinen der Rev. A kann CE' über INHIBIT' und ENFIRM geschaltet werden. Auf Hauptplatinen der Rev. B sind die CE'-Eingänge mit Masse verbunden, die mögliche Abschaltung der ROMs über INHIBIT' wird rein MMU-intern verarbeitet und findet über die Steuerung von ROMEN1' bzw. ROMEN2' statt. ENFIRM wurde durch FRCTXT' ersetzt und hat nichts mehr mit der ROM-Aktivierung zu tun.

ROMEN1' und ROMEN2' sind beide MMU-intern über INHIBIT' steuerbar.[1] Wenn diese Leitung durch eine Zusatzkarte aktiviert wird, können ROMEN1' und ROMEN2' nicht aktiv werden - der ROM der Hauptplatine ist damit abgeschaltet. Die Funktion des ROM besteht darin, als Reaktion auf eine Adresse Daten auf den Bus zu legen - eine Zusatzkarte, die via INHIBIT' die ROMs der Hauptplatine abgeschaltet hat, kann den entsprechenden Adreßraum entweder auf dieselbe oder auch auf eine völlig andere Art und Weise benutzen. Falls INTCXROM und INTC8ROM entsprechend gesetzt sind, kann eine Zusatzkarte damit sogar den Adreßraum aller anderen Zusatzkarten belegen(!).

Die Aktivierung von ROMEN1' und ROMEN2' als Reaktion auf Adressen im Bereich $D000-$FFFF ist MMU-intern über R/W' gesteuert, bei Adressen im Bereich $C100-$CFFF wird ROMEN1' dagegen nach entsprechendem Setzen von INTCXROM auch dann aktiviert, wenn der Prozessor eine Schreibaktion ausführen will. Das Ergebnis ist eine Kollision auf dem Datenbus zwischen der CPU und dem $C1-$DF-ROM.Ich bin mir nicht so ganz im klaren darüber, warum Apple, Inc. für diesen Bereich auf eine Steuerung von ROMEN1' über R/W' verzichtet hat. Eine Spekulation: Damit sollte eine "Kompatibilität" zwischen ROMEN1' und den Signalen I/O STROBE' und I/O SELECT' hergestellt werden - beide reagieren im Bereich $C100-$CFFF sowohl auf Lese- als auch auf Schreibaktionen des Prozessors. Was man damit anfangen kann? Nun, genausoviel wie mit der Möglichkeit, den ROM der Hauptplatine zu "beschreiben" - nämlich nichts.

ROM in den Steckplätzen

Außer den ROMs der Hauptplatine enthält ein Apple //e im Normalfall eine oder mehrere Zusatzkarten, die ROM-Bausteine mit 6502-Programmen enthalten. Im "Kartenbereich" $CnXX adressierbare ROMs werden über I/O SELECT' der jeweiligen Karte aktiviert, ROMs für den Bereich $C800-$CFFF über I/O STROBE'. Für eine Karte, die INHIBIT' aktiviert, existiert keine direkte Zuordnung dieser Art: sie kann im Bereich $0-$BFFF und $C100-$FFFF überall und nirgends ROM enthalten. ROMs, die auf I/O SELECT' und/oder auf I/O STROBE' reagieren, enthalten normalerweise Programme zur Steuerung der jeweiligen Karte. Über INHIBIT' sollte nur dann gearbeitet werden, wenn entweder die ROMs anderer Karten und/oder die ROMs der Hauptplatine überdeckt werden sollen.

Ein über I/O SELECT' aktivierter ROM enthält ein Programm von 256 Byte Länge, mit dem die Ein-/Ausgabe einer Zusatzkarte nach einem PR#- oder IN#-Befehl zum entsprechenden Steckplatz initialisiert wird. Ein Beispiel dafür ist der ROM des Diskettencontrollers für die DISK II. Das Programm in diesem ROM wird über $C6XX adressiert (Voraussetzung: die Karte befindet sich in Steckplatz 6) und stellt den ersten Schritt des Ladevorgangs eines Betriebssystems von der Diskette in den Speicher des Computers dar. 256 Byte sind für diese Aufgabe nicht einmal annähernd genug - dieser Platz reicht aber zum Laden der nächsten "Boot-Stufe" von der Diskette.

Für die komplette Kontrolle einer Ein-/Ausgabe über eine Zusatzkarte sind 256 Byte fast immer zuwenig. Aus diesem Grund wird ein über I/O SELECT' aktivierter ROM meistens durch einen über I/O STROBE' gesteuerten ROM ersetzt. Dieses Signal wird an allen "normalen" Steckplätzen aktiv, sobald der Prozessor eine Adresse im Bereich $C800-$CFFF ausgibt (Voraussetzung: INTCXROM und INTC8ROM sind zurückgesetzt). Dieser Bereich kann vom ROM einer Zusatzkarte belegt werden - dafür müssen aber bestimmte Konstruktionsregeln eingehalten werden, weil natürlich nur jeweils eine einzige Karte reagieren darf. Details dazu finden Sie in Kapitel 7. Im Prinzip kann über I/O STROBE' ein beliebiges Peripheriegerät aktiviert werden, die meisten Anwendungen schalten hier aber einen ROM, der irgendeine Art von I/O-Treiber enthält.

Die dritte Schaltungsmöglichkeit geht über die Leitung INHIBIT', die im Gegensatz zu I/O STROBE' und I/O SELECT' kein Eingang, sondern ein Ausgang einer Karte ist. Die beiden ersteren ermöglichen den Zugriff des Prozessors auf den ROM einer Karte, der INHIBIT'-Ausgang liefert dagegen ein Abschaltsignal, über das eine Zusatzkarte ROM und/oder RAM der Hauptplatine (und des speziellen Steckplatzes) durch eigene Speicherbausteine ersetzen kann. Eine mit INHIBIT' arbeitende Karte muß natürlich nicht unbedingt ROM enthalten - das ist nur eine von vielen Möglichkeiten (wenn auch neben RAM-Speichererweiterungen die wahrscheinlichste). Ein Beispiel für eine derartige Konstruktion sind die Firmware-Karten von Apple, Inc. Sie enthalten jeweils 12 kByte ROM und ermöglichen den alternativen Einsatz des jeweils fehlenden BASIC-Dialektes. Auch auf einem Apple //e kann man damit eine ROM-Version von Integer BASIC nachrüsten.

---

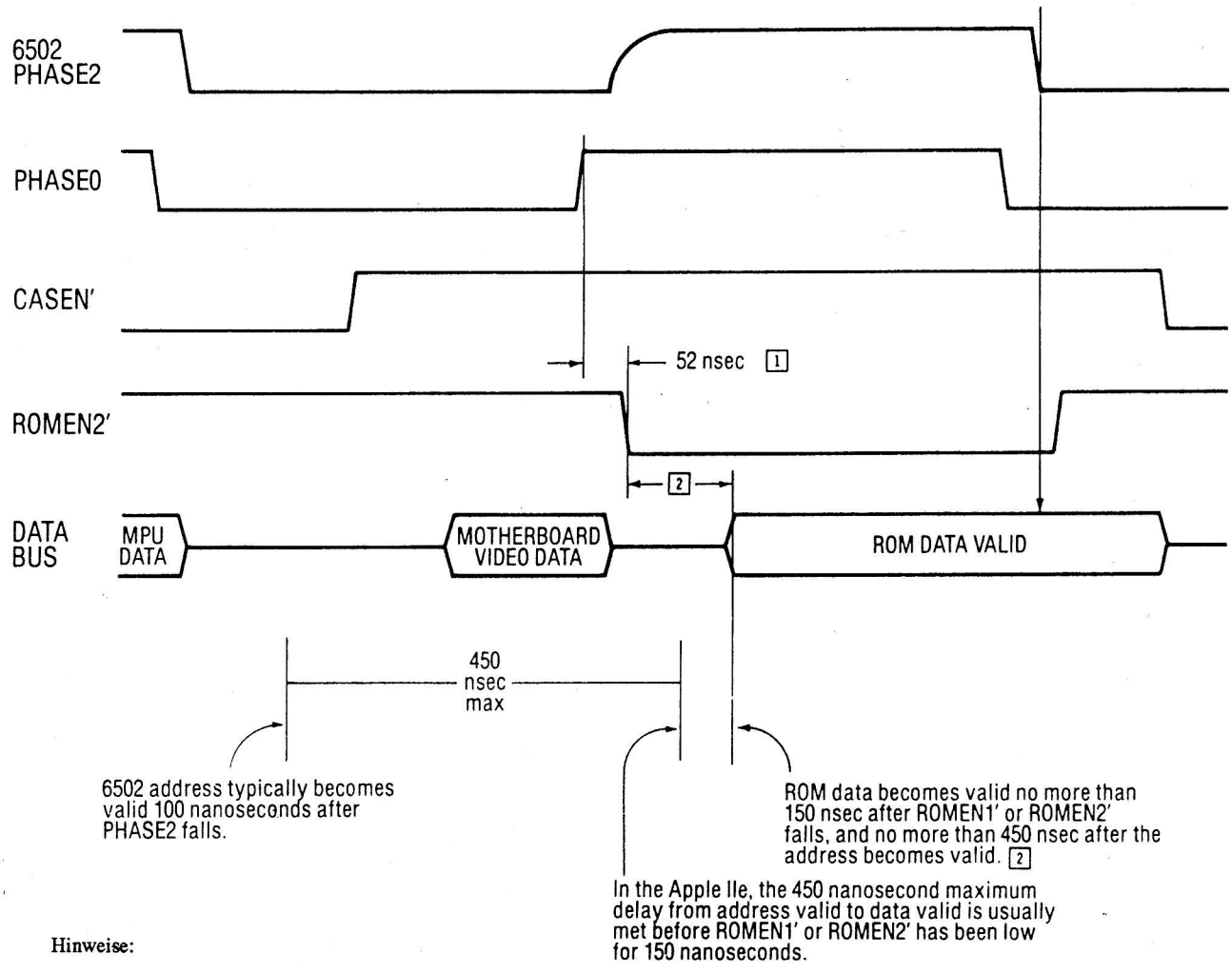
1 Wie in Kapitel 5 beschrieben, sind CASEN' und EN80' ebenfalls über INHIBIT schaltbar, der Bereich $D000-$FFFF läßt sich damit auch dann abschalten, wenn anstelle des ROMs der hohe RAM aktiviert ist.

Das Zeitverhalten der ROMs

Die Zeitabläufe eines Lesezugriffs sind sehr einfach und für alle ROMs der Hauptplatine, die direkt vom Prozessor gelesen werden können (d.h. nicht für den Video-ROM), dieselben. Die Eckwerte werden durch das Zeitverhalten des ROM-Bausteins selber, die An- und Abschaltzeiten der RAMs via CASEN' (bzw. EN80' für den AUX-RAM) und die Laufzeit von ROMEN1' bzw. ROMEN2' in der MMU vorgegeben. Wenn wir davon ausgehen, daß Apple, Inc. ROMs mit 450 ns benutzt, ergeben sich die folgenden Charakteristiken:

1. Vom ROM ausgegebene Daten werden allerspätestens 450 ns nach Anlegen der Adresse gültig.
2. Vom ROM ausgegebene Daten werden spätestens 150 ns nach der fallenden Flanke am Eingang OE' gültig.
3. Die Datenausgänge des ROMs sind spätestens 150 ns nach der steigenden Flanke am Eingang OE' wieder im hochohmigen Zustand.[2]

Bild 6.2 Zeitverhalten: Lesezugriff auf die E0-FF-ROMs



Hinweise:

(1) Die Werte sind durch Messungen des Autors im eigenen Apple //e entstanden.

(2) Die maximale OE-Zugriffszeit (TAOE) ist herstellerabhängig: Sie beträgt für den General Instruments RO9864AB 75 nsek., für den Synertek SY2365A 150 nsek.

2 Die hier gegebenen Werte gelten für den ROM SY2365 von Synertek. Verschiedene andere Hersteller geben für 450 ns-ROMs kürzere Zugriffszeiten nach OE' an.

Die Signalfolgen bei einem Lesezugriff des Prozessors auf einen der ROMs finden Sie in Bild 6.2. Die Reihenfolge der wichtigsten Ereignisse ist:

1. Während PHASE1 führt der Videoscanner einen normalen RAM-Zugriff aus.
2. Rund 100 ns nach der fallenden Flanke von PHASE2 wird die vom Prozessor ausgegebene Adresse auf dem Datenbus gültig. Spätestens 450 ns nach diesem Punkt werden vom ROM ausgegebene Daten gültig, wenn der OE'-Eingang des entsprechenden ROM-Bausteins lange genug aktiv gehalten worden ist.
3. Wenn der Prozessor im vorherigen Zyklus auf den RAM der Hauptplatine zugegriffen hat, reagiert die MMU auf den Moment, in dem die ROM-Adresse gültig wird, und setzt (nach interner Laufzeitverzögerung) CASEN' auf "1". Damit wird die Verbindung der CPU zum RAM während PHASE0 unterbrochen.
4. PHASE0 steigt: eines der Signale ROMEN1', ROMEN2' oder KBD' wird von der MMU nach interner Laufzeitverzögerung aktiviert. Falls EN80' noch aktiv ist (d.h. der letzte Prozessorzugriff auf den AUX-RAM stattgefunden hat), wird dieses Signal auf "1" gesetzt. CAS' und Q3 steigen zum selben Zeitpunkt und beenden so den RAM-Zugriff des Videoscanners. Der Datenbus geht in den hochohmigen Zustand über und behält so die Videodaten des Hauptplatinen-RAMs bis zu dem Zeitpunkt, in dem der ROM die Kontrolle übernimmt.
5. Maximal 150 ns nach der fallenden Flanke am Eingang OE' werden vom ROM ausgegebene Daten gültig - vorausgesetzt, daß der Adreßbus früh genug stabil geworden ist. Wenn man von den typischen Werten eines 6502A und eines 2365 ausgeht, werden die Adressen schnell genug gültig und der Zeitpunkt der Datenausgabe wird rein durch die Verzögerung zwischen "OE' aktiv" und "Daten gültig" bestimmt. Auf jeden Fall sind ausgegebene ROM-Daten eine geraume Weile vor der fallenden Flanke von PHASE2 gültig.
6. PHASE0 fällt, gefolgt von der fallenden Flanke von PHASE2 des Prozessors (Datenübernahme). Wenn der nächste Zugriff des Prozessors nicht auf denselben ROM-Baustein erfolgt, steigt im selben Zeitraum das vorher aktiv gewesene ROM-Aktivierungssignal (ROMEN1', ROMEN2' oder KBD'). Falls der nächste Zugriff auf den AUX-RAM stattfindet, wird gleichzeitig EN80' aktiv. Die Laufzeiten der MMU sind in meinem Apple //e höher als die Verzögerung der CPU zwischen PHASE0 und PHASE2 (das dürfte wahrscheinlich sogar bei allen //e der Fall sein). Der aktivierte ROM kontrolliert den Datenbus maximal noch 150 ns, nachdem sein OE'-Eingang wieder inaktiv geworden ist.
7. Wenn der Prozeor im nächsten Zyklus auf den RAM der Hauptplatine zugreift, wird CASEN' kurz nach dem Moment wieder aktiv, in dem die nächste Prozessoradresse gültig wird (d.h. nach der entsprechenden Laufzeit in der MMU).

Die Firmware des Apple

Der hardwaremäßige Aufbau des Apple bestimmt letztendlich seine Möglichkeiten und Grenzen. Bekanntlich ist ein Computer (egal wie kompliziert er aufgebaut sein mag) immer nur so gut wie sein Programm. Die Programme, die im ROM gespeichert sind, dürften die wahrscheinlich wichtigsten sein, die je für den Apple geschrieben wurden. Diese Programme erwecken den Computer zum "Leben" und werden so oft und mit einer solchen Selbstverständlichkeit benutzt, daß man manchmal sogar vergißt, daß es sich auch nur um (änderbare!) Befehlsfolgen für den Prozessor handelt.

Sinn und Zweck, Möglichkeiten und Fehler der eingebauten Firmware des Apple sind Gegenstand einer solchen Unzahl von Büchern und Artikeln, daß wir hier erst gar nicht den Versuch unternehmen werden, dieser Publikationsflut etwas entgegenzusetzen. Wir begnügen uns statt dessen mit einem knappen Überblick, der eine gewisse Einsicht in die Wichtigkeit dieser Programme vermitteln soll. Der Anspruch ist dabei mehr "historischer" Natur und beginnt mit den ersten Ausgaben des Apple II.

Die ersten Firmware-Programme bestanden aus Integer BASIC, dem Monitor und einigen Utilities für den 6502. Sie wurden hauptsächlich von Woz selber geschrieben. Das war wieder einmal in den schlechten alten Zeiten, als Diskettenlaufwerke preislich noch jenseits von Gut und Böse waren, und bevor Microsoft, Inc. begann, die Hardware-Hersteller mit hochwertigen BASIC-Interpretern zu versorgen. Integer BASIC belegt rund 5 kByte Speicherplatz und hat einige sehr eng gesetzte Grenzen wie z.B. keine Fließkommarechnung, keine Befehle für die hochauflösende Grafik und keine Stringarrays, um nur drei zu nennen. Damit soll nicht die Leistung von Steve Wozniak geschmälert werden - schließlich war Integer BASIC der erste Dialekt, der jemals auf einem Mikrocomputer lief - es heißt nur, daß dieser Interpreter wie alle Vorläufer nach wenigen Jahren überholt war.

Die zusammen mit Integer BASIC gelieferten Utilities bestanden aus einigen Fließkomma-Rechenroutinen, dem 16-Bit-Interpreter SWEET 16 und dem Mini-Assembler. SWEET 16 ist ein kleines, aber sehr raffiniert aufgebautes Programm, über das der Programmierer Daten in Form von Worten mit 16 Bit manipulieren kann. Dazu werden die Speicherstellen $0 bis $1F als 16-Bit-Register benutzt, der normale Befehlssatz eines Prozessors wie "Addiere", "Subtrahiere", "Vergleiche" etc. ist vorhanden - SWEET 16 ist also ein Interpreter für erweiterte Maschinenbefehle des 6502. In dieser Sprache geschriebene Programme benötigen meist weniger Speicherplatz als Programme, die eigene 16-Bit-Routinen enthalten, dafür ist die Ausführung von SWEET 16 natürlich auch etwas langsamer.

Der Mini-Assembler ist der eigentliche "Renner" des Programmpakets. Unzählige Apple-Besitzer haben mit diesem Programm die Maschinensprache und die Programmierung des 6502 gelernt. Das *Technical Reference Manual* für den //e enthält eine ausführliche Beschreibung der Benutzung des Mini-Assemblers.

Über SWEET 16 und die Fließkomma-Routinen findet sich keine einzige Beschreibung in den offiziell von Apple, Inc. herausgegebenen Handbüchern. Eine komplette Beschreibung findet sich statt dessen in der Novemberausgabe 1977 von BYTE ("Sweet 16: The Dream Machine"). Der Autor ist Steve Wozniak höchstpersönlich.

Das Monitorprogramm

In der Terminologie der Mikrocomputer ist ein "Monitor" ein Programm im ROM, das die Primitivfunktionen des Computers wie "Lesen eines Zeichens von der Tastatur" und "Ausgabe eines Zeichens auf den Bildschirm" und Möglichkeiten zum Verändern und zur Ausgabe einer oder mehrerer Speicherstellen enthält. Vor der Erfindung von BASIC im ROM war der Monitor die wichtigste "Schnittstelle" von Mensch zu Maschine - in den meisten Fällen auch die einzige, die überhaupt vorhanden war.

Zusammen mit der Evolution des Apple fand der Übergang von "Monitor" zu "BASIC" als primäre Kommunikationsform statt. Im Apple II findet sich der Benutzer nach Einschalten der Stromversorgung noch im Monitor und muß BASIC erst durch einen Befehl starten - diese Prozedur wurde später durch den "Autostart" ersetzt.

Der Monitor des Apple besteht aus folgenden Programmteilen:

- Einsprünge und Vektoren zu BASIC;
- Tastatur-Eingabe;
- Video-Ausgabe von Text und LoRes-Grafik;
- Kassettenein- und -ausgabe;
- Zuordnung der Steckplätze zu den I/O-Funktionen mit Möglichkeit der Umleitung;
- Speicherstellenausgabe und -veränderung;
- 6502-Disassembler;
- 6502-TRACE;
- 6502-Einzelschrittmodus;
- Behandlungsroutinen für RESET', NMI', IRQ' und BRK';
- verschiedene Routinen zur Multiplikation und Division von 16-Bit-Werten.

Die Kontrolle des Monitors findet über einen sehr übersichtlich aufgebauten Interpreter statt, dessen Benutzung vollständig und gut in den jeweiligen Handbüchern dokumentiert ist.

Damit hätten wir alle Einzelteile für den Apple II zusammen: Ein BASIC für arme Leute, zwei leere ROM-Sockel mit jeweils 2 kByte für spätere Erweiterungen und einen Monitor, von dem manche Computerbesitzer heutzutage noch träumen. Einen sehr wichtigen Punkt haben wir noch vergessen: Apple, Inc. hat sich entschieden, den Quelltext des Monitorprogramms zu veröffentlichen. Das war ein sehr riskanter Schritt, der sich aber fast sofort auszahlte: damit war der Weg für alle Neugierigen offen, sich wirklich bis in die letzte Kleinigkeit durch den Apple hindurchzuwühlen. Das Ergebnis ist bekannt: eine große Zahl von Anwendungen und Zusatzkarten für Zwecke, an die auch nicht einmal die verrückteste Werbeabteilung je gedacht hätte, und eine fast unglaubliche Zahl von "Freaks". Mit diesem Schritt zum "offenen System" hat es Apple, Inc. geschafft, Bastler, Edelbastler und Geschäftsleute für ein und dieselbe Maschine zu interessieren.

Das Monitorprogramm legt einige Charakteristiken des Apple fest: wie sich der Computer nach einem RESET verhält, die Behandlung auszugebender Zeichen, die Funktionsweise von Cursorbewegungen und die Zuordnung von Steckplätzen für die Ein-/Ausgabe. Die vom Monitor belegten Speicherstellen der Seite 0 müssen vom Programmierer genauso berücksichtigt werden wie die Tatsache, daß der Monitor die Seite 2 als Tastaturpuffer benutzt. Zusatzkarten, die als primäre Ein-/Ausgabegeräte arbeiten können, müssen zumindest eine Einsprungadresse auf $Cn00 haben, denn der Monitor führt bei einer Aktivierung zuerst einen Sprung zu dieser Adresse aus ($C100 für Steckplatz 1, $C200 für Steckplatz 2 etc.). Alle diese Dinge sind nicht für die Ewigkeit festgelegt: Das Monitorprogramm kann durch Einladen eines neuen Betriebssystems in den hohen RAM oder durch Ersetzen eines ROMs geändert werden. Den letzteren Weg ist Apple, Inc. für die Erweiterung des II auf den II+ gegangen.

Der Apple II+

Gegen Ende der siebziger Jahre vollführte der Apple II in rascher Folge einige Evolutionsschritte als Folge neuer Entwicklungen der Elektronikindustrie und des Softwaremarktes. Die einzelnen Schritte in chronologischer Ordnung:

1. Applesoft BASIC wurde als Kassettenversion verfügbar.
2. Die DISK II wurde zusammen mit den ersten Versionen von DOS vorgestellt.
3. Applesoft wurde auf Diskette verfügbar.
4. Applesoft wurde als Firmware-Karte verfügbar.
5. Der Apple II+ erschien mit Applesoft im ROM und dem "Autostart"-Monitor.
6. Das Sprachsystem mit der 16k-RAM-Karte, Pascal und Disketten mit 16 Sektoren erschien.
7. Die "endgültige" Version von DOS (3.3) kam zusammen mit Disketten mit 16 Sektoren auf den Markt.

Eine Konfigurationsmöglichkeit wurde in dieser Zeit (1980) besonders populär: Ein Dialekt im ROM der Hauptplatine, der zweite auf einer Firmware-Karte und eine automatische Umschaltung von DOS 3.3 zwischen beiden beim Laden eines BASIC-Programmes von Diskette. Wie dieses Beispiel zeigt, stehen einem mit bankgeschalteter Firmware fast alle Möglichkeiten offen.

Die Verfügbarkeit von Applesoft BASIC hatte einen ganz entscheidenden Einfluß auf die weitere Verbreitung des Apple: mit Applesoft wurden hochauflösende Farbgrafik, Fließkommarechnung und große Textdateien auf der Diskette auch für "normale" Programmierer möglich. Unerfreulicherweise ist Applesoft weder in der Kommandostruktur noch in der Speicherbelegung kompatibel zu Integer BASIC, die Befehle AUTO (automatische Zeilennumerierung) und DSP (TRACE von Variablenwerten) verschwanden zusammen mit dem Mini-Assembler von der Bildfläche. SWEET 16 und die zu Integer BASIC gehörenden Fließkomma-Routinen sind mit Applesoft ebenfalls nicht mehr verfügbar, werden aber auch eigentlich nicht mehr gebraucht.

Die im neuen "Autostart"-Monitor vorgenommenen Veränderungen kennzeichnen die Wandlung des "typischen" Computerbesitzers in diesen Jahren. Ein Apple mit diesem Monitor startet nach Einschalten der Stromversorgung selbständig ein angeschlossenes Diskettenlaufwerk, anstatt den Benutzer in den Monitor zu setzen; die ESCape-Sequenzen zur Bewegung des Cursor wurden stark erweitert. Dafür entfielen die Fehlersuchhilfen STEP und TRACE für 6502-Maschinenprogramme sowie die 16-Bit-Routinen im Monitor - es wurden also Programmierhilfen für "Freaks" zugunsten der Benutzerfreundlichkeit geopfert. Der Inhaber eines kleineren Geschäftsbetriebes legt seine Diskette ein und startet den Computer durch Einschalten der Stromversorgung, ohne sich um weitere Details zu kümmern - der Hacker kümmert sich schleunigst um eine 16k-Sprachkarte, in die er bei Bedarf den alten Monitor hineinlädt, um den Mini-Assembler, STEP und TRACE wieder zur Verfügung zu haben.

Der Einfluß der 16k-Sprachkarte

Die Entwicklung dieser Karte markiert einen weiteren "Meilenstein" der Evolution des Apple. Da mittlerweile auch Diskettenlaufwerke verfügbar waren, entfiel damit die Notwendigkeit weiterer Betriebssysteme in Firmware-Karten: das gesamte Integer BASIC kann z.B. innerhalb weniger Sekunden in den Bereich $E000-$F7FF von Diskette geladen werden (mit einer Kassette hätte das wohl eine Viertelstunde gedauert!). Ein in dieser Karte gespeicherter Monitor wurde für den Benutzer ohne weiteren Aufwand änderbar, die Anzahl der möglichen Betriebssysteme für den Bereich $D000-$FFFF nur noch durch die Fantasie begrenzt.

Die Nachteile dieser Karte liegen in der Möglichkeit, ihren Inhalt versehentlich zu überschreiben, einer gewissen Wartezeit für das Laden von der Diskette und der nun vorhandenen Möglichkeit, mit veränderten RESET-Vektoren die übelsten Dinge anzustellen. Diese Möglichkeit ist von "Kopierschutz"-Programmierern weidlich genutzt worden - es soll sogar Programme geben, die bei einem RESET eingelegte Disketten neu formatieren...

Der für Designer gewichtigste Nachteil der (originalen) 16k-Sprachkarte liegt allerdings in einem anderen Punkt: Nach Installation dieser Karte ist der Bereich $F800-$FFFF permanent belegt- auch über INHIBIT' ist hier für andere Karten nichts zu holen.

Der Apple //e

Anfang 1983 brachte Apple, Inc. den //e auf den Markt. Der //e enthält gegenüber dem II(+) einige starke Verbesserungen: Groß- und Kleinschreibung, 128 kByte RAM, 80-Zeichen-Darstellung und doppelt hochauflösende Grafik. Trotzdem verhält er sich für ein "normales" Programm wie ein II+ mit 48 kByte RAM, einer 16k-Sprachkarte in Steckplatz 0 und einer 80-Zeichen-Karte in Steckplatz 3. Die Simulation der 16k-Sprachkarte ist eine reine Hardware-Mechanisierung mit 64 kByte RAM auf der Hauptplatine, über $C08X kontrollierte MMU-Softswitches und der Steuerung des Datenbusses über die MMU-Ausgänge ROMEN1', ROMEN2', CASEN' und EN80'. Die einzigen Unterschiede zur originalen 16k-Sprachkarte liegen darin, daß dieser RAM-Bereich sowohl über INHIBIT' als auch über RESET' abgeschaltet werden kann.

Die 80-Zeichen-Karte in Steckplatz 3 existiert ebenfalls nicht (bzw. nur zu geringen Teilen) in physikalischer Form, sondern hauptsächlich im Taktgenerator, der Videogenerator und der MMU. Die zur Steuerung der Zeichenausgabe notwendige Firmware befindet sich ebenfalls auf der Hauptplatine, nämlich im $C1-$DF-ROM und ist praktisch "co-resident" mit den Routinen für den 40-Zeichen-Betrieb des alten Apple II+. Die 40-Zeichen-Routinen werden beim Einschalten der Stromversorgung oder durch RESET aktiviert, die 80-Zeichen-Routinen dagegen erst durch den Befehl PR#3 oder IN#3.

Die 80-Zeichen-Routinen sind nicht nur für die notwendige Umschaltung zwischen AUX-RAM und dem RAM der Hauptplatine zur Speicherung von 80 Zeichen pro Zeile zuständig, sie umfassen auch noch die Interpretation einiger Steuerzeichen und ESCape-Modi, diverse "Housekeeping"-Routinen für Pascal und den Programmteilen XFER (Übergabe der Programmkontrolle zwischen Hauptplatinen- und AUX-RAM) sowie AUXMOVE (Kopieren zwischen Hauptplatinen- und AUX-RAM).

Alle diese Routinen (oder zumindest ihre Einsprungvektoren) befinden sich im Bereich $C3XX und werden zur 80-Zeichen-Firmware gerechnet. Da eine derartige Menge natürlich nicht in 256 Byte unterzubringen ist, wurde der ROM-Bereich des //e von den 12 kByte des II(+) auf 16 kByte erweitert. Diese Erweiterung hätte durch eine Bankumschaltung des ROMs im Bereich $D000-$DFFF stattfinden können, also analog zur Schaltung des hohen RAMs - stattdessen wurde beschlossen, im Bereich $C100-$CFFF zwischen Ein-/Ausgabe und ROM zu schalten. Damit bleibt auch in diesem Punkt die Kompatibilität zu einer "echten" 80-Zeichen-Karte erhalten: die Firmware liegt in beiden Fällen im Bereich $C3XX.

Abgesehen von den 80-Zeichen-Routinen weist der Monitor des //e keine großen Unterschiede zum II+ auf. Es gibt einige kleine Erweiterungen wie eine neue KEYIN-Routine, die Interpretation der vier Pfeiltasten im Escape-Modus und die Abfrage der Apfeltasten bei einem RESET. Dadurch ist der gesamte Monitor etwas länger geraten, einige Teile wurden deshalb in den Bereich $C100-$C2FF verschoben und müssen via GOTOCX mit einem Index zur gewünschten Routine im Y-Register aufgerufen werden.

Eine zusätzliche Erweiterung stellen die Firmware-Routinen zum Selbsttest des Computers dar. Sie befinden sich im Bereich $C400-$C7FF, der Quelltext wurde erstmals in der Ausgabe "Juli 1985" des *Technical Reference Manual* für den //e veröffentlicht. Hier ist ein Test des gesamten Hauptplatinen-RAMs (nicht aber des AUX-RAMs) sowie die Prüfung verschiedener Softswitches von IOU und MMU enthalten. Außerdem wird eine Prüfsumme der beiden ROMs $C1-$DF und $E0-$FF berechnet und verglichen.

Die untersten 256 Byte des $C1-$DF-ROMs können vom Prozessor nicht gelesen werden, weil die MMU im Bereich $C0XX ROMEN1' nicht schaltet. Selbstverständlich war ich neugierig genug, um mir diesen Bereich über einen EPROM-Programmierer anzusehen. Was drin steht? Lauter Nuller.

Der verbesserte Apple //e

Bei der Entwicklung des //e ergab sich natürlicherweise ein ständiger Konflikt zwischen den sich manchmal widersprechenden Zielen "maximale Kompatibilität zum II+" und "der beste aller möglichen Computer". Einige der sich daraus ergebenden Kompromisse waren nicht der Weisheit letzter Schluß. Meine persönliche Kritik betrifft die Behandlung der 80-Zeichen-Firmware als Zusatzkarte, die nicht vorhandene Erweiterung von Applesoft auf doppelt hochauflösende Grafik und die Tatsache, daß sich Applesoft bei einem in Kleinbuchstaben gegebenen Befehl mit einem schlichten SYNTAX ERROR beschwert.

Mittlerweile war der //c in der Entwicklung. Einige kleinere Fehler der Firmware wurden dabei beseitigt und einige Änderungen vorgenommen wie die eingebaute Maus-Firmware, die 65C02-CPU und die //c-typische Struktur der Ein-/Ausgabe. Durch die neuerliche Arbeit am Monitorprogramm der Apple-Familie ist schließlich dabei ein "Upgrade" für den Apple //e abgefallen, das seit Anfang 1986 verfügbar ist und in zwei Formen existiert - einmal als Austauschsatz von drei ROMs ($C1-$DF, $E0-$FF, Video) und einem 65C02-Prozessor für bereits ausgelieferte Geräte, zum anderen als "verbesserter" //e mit vier neuen ROMs, dem 65C02-Prozessor und einer leicht veränderten Tastatur nebst dazugehörigem veränderten Tastatur-ROM.[3]

Unter einer Ikone versteht man ein kleines Bildchen, mit dem ein Ding oder eine Aktion dargestellt wird - und Apple, Inc. ist zur Zeit Weltmeister in der Herstellung von Benutzeroberflächen, die mit Ikonen arbeiten, auf die der Benutzer dann nur noch mit einem maus-gesteuerten Pfeil zeigen muß, um die entsprechende Aktion auszulösen. Der verbesserte // e ersetzt die INVERSEn Großbuchstaben im Video-ROM auf den (ROM-)Adressen $200-$2FF durch den bereits im //c vorhandenen "Maus-Zeichensatz" (s. Bild 8.8). Es existiert zwar noch ein zweiter INVERSE-Zeichensatz auf den ROM-Adressen $000-$0FF im neuen Video-ROM, die 40- und 80-Zeichen-Routinen arbeiten nach dem Befehl INVERSE aber mit ASCII-Werten im Bereich von $40-$5F - nach einer Schaltung von ALTCHARSET erscheinen deshalb anstelle inverser Großbuchstaben Maus-Zeichen auf dem Bildschirm.

Zusätzlich zum "Maus-Zeichensatz" im Video-ROM und den in Kapitel 4 beschriebenen Erweiterungen der Behandlung von Interrupts enthalten die ROMs des verbesserten //e wieder einen Mini-Assembler (allerdings nicht mit den neuen Befehlen des 65C02), die Möglichkeit der Eingabe von ASCII-Zeichen (anstelle von Hexzahlen) im Monitor, einen Monitor-Befehl SEARCH ("Suchen"), Verbesserungen der 80-Zeichen-Routinen hinsichtlich der Scroll-Geschwindigkeit sowie die korrekte Interpretation von Kommandos für Applesoft und Pascal auch dann, wenn sie mit Kleinbuchstaben eingegeben werden. Um für diese Erweiterungen Platz zu schaffen, wurden diverse Programmteile in den Bereichen $CXXX und $FF75-$FFFF optimiert, gekürzt oder ganz weggelassen. Die bemerkenswerteste Veränderung betrifft die Selbsttestroutinen, die um fast zwei Speicherseiten geschrumpft sind.

Das Ergebnis kann man nur als gelungen bezeichnen. Die neuen ROMs enthalten einige wichtige Verbesserungen und eigentlich keine Nachteile, wenn man einmal von dem Problem mit INVERSE absieht. Ich schätze, daß auch eine große Zahl der Besitzer älterer Ausgaben des //e sich dieses "Upgrade" in nächster Zukunft leisten wird - der neue Standard der Apple II-Familie dürfte damit für längere Zeit gesetzt sein.

---

3 Eine detaillierte Beschreibung der neuen ROMs - soweit es IRQ' und NMI' betrifft - finden Sie am Ende von Kapitel 4.

Kapitel 7

Ein-/Ausgabe im Apple //e

Um es noch einmal zu wiederholen: Die Übergabe von Befehlen der CPU zu allen anderen Bausteinen des Systems geschieht über die Dekodierung des Adreßbusses. Jeder, der Programme für den Apple schreibt, wird sich dessen früher oder später bewußt - selbst Benutzer ersparen sich mit dieser Sicht der Dinge eine Anzahl möglicher Probleme. Mögliche Formen der Datenübergabe zwischen der CPU und einem anderen Baustein bestehen eben nicht nur aus der relativ leicht zu verstehenden Übergabe von Daten an eine adressierte Speicherstelle, sondern teilweise auch aus dem einfachen Ansprechen einer Adresse, wobei der Inhalt des Datenbusses völlig irrelevant ist.

Der größte Teil des I/O ("Input/Output" = Eingabe/Ausgabe) findet unter direkter Kontrolle der CPU statt. Das schließt alle I/O-Baugruppen auf der Hauptplatine außer dem Videoausgang sowie den allergrößten Teil des Datenverkehrs mit Zusatzkarten ein. Die CPU kontrolliert diese Bausteine und -gruppen durch direkte Adressierung, also in derselben Weise, als ob einzelne Speicherstellen beschrieben oder gelesen würden. Genauso wie jede Speicherstelle eine eigene Adresse haben muß, hat auch jede I/O-Gruppe, die von der CPU direkt angesprochen werden kann, ihre eigene Adresse bzw. ihren eigenen Adreßbereich.

Der 6502 kennt keinen Befehl im Sinne von "Kontrolliere Adreßbus" - er gibt schlicht eine Adresse aus und kontrolliert entweder gleichzeitig den Datenbus oder liest ihn. Was macht also ein Programmierer, der ein "Klick" im eingebauten Lautsprecher des Apple erzeugen möchte? Er benutzt "LDA $C030" oder "CPX $C030" oder "IRGENDWAS $C030" und ignoriert den Inhalt des Datenbusses. Hier liegt der Grund, wieso man den Lautsprecher mit einem Befehl wie "SOUND = PEEK (-16336)" zum Leben erwecken kann. Der Sinn dieses Befehls liegt nicht im Lesen eines Wertes, sondern im schlichten Ansprechen der Speicherstelle $C030 (-16336). Bitte nicht vergessen: Was für ein Programm der Prozessor gerade ausführt, spielt überhaupt keine Rolle - aber jede einzelne von ihm ausgegebene Adresse wird von der restlichen Elektronik des Systems analysiert und dekodiert.

Dieses Kapitel enthält eine Beschreibung der diversen eingebauten I/O-Möglichkeiten des Apple //e und zeigt, wie die von der CPU ausgegebenen Adressen dekodiert und zur Erzeugung von Signalen verwendet werden, auf die andere Baugruppen des Systems reagieren. Zum eingebauten I/O gehören ebenfalls die Steckplätze und die dazugehörige Firmware. Die Erzeugung der Videosignale werden Sie in diesem Kapitel allerdings nicht finden - sie ist so komplex, daß sich mit ihr das gesamte nächste Kapitel beschäftigt.

Die Dekodierung der Peripherieadressen

Wir haben den Begriff Adreßdekodierung bereits des öfteren verwendet, ohne ihn exakt zu definieren: unter diesem Begriff wird im Apple //e der Prozeß der Auswahl eines von 65536 Plätzen durch Analyse einer 16-Bit-Adresse verstanden. Der gröbste Schritt besteht dabei in der Unterteilung des Speichers in die Hauptkategorien RAM, ROM und I/O ; er wird innerhalb der MMU ausgeführt. Die MMU erzeugt aus einer 16-Bit-Adresse weitere Signale, die *innerhalb* einer der drei Kategorien zur weiteren Unterscheidung benutzt werden. Alle diese Signale finden Sie ausführlich in Kapitel 5 beschrieben, wir werden uns hier nur mit denen befassen, die für den I/O-Bereich Verwendung finden: CXXX (I/O ENABLE = Aktivierung des I/O), KBD' und MD IN/OUT'. KBD' aktiviert die Ausgänge des Tastatur-ROMs bei einem Lesezugriff des Prozessors auf den Bereich $C000-$C01F (s. Bild 7.4), MD IN/OUT' bestimmt die Flußrichtung des peripheren Datenbustreibers (S. Bild 7.6). CXXX ist das Signal, mit dem ein weiterer Dekodierungsprozeß in Gang gesetzt wird (s. Bild 7.1).

$CXXX wird aktiv, sobald sich eine Adresse im Bereich $CXXX auf dem Adreßbus befindet. Dieses Signal wird nicht wie die meisten anderen Prozessoraktivitäten und Kontrollsignale durch PHASE0 geschaltet - die Reaktion erfolgt direkt nach einer Verzögerung durch die MMU-internen Laufzeiten. Falls eine der folgenden Bedingungen zutrifft, bleibt CXXX inaktiv:

- INTCXROM ist gesetzt und der Prozessor greift auf den Bereich $C100-$CFFF zu;
- SLOTC3ROM ist gesetzt und der Prozessor greift auf $C3XX zu;
- INTC8ROM ist gesetzt und der Prozessor greift auf den Bereich $C800-$CFFF zu.

Über das Signal CXXX wird der periphere Adreßdekoder aktiviert, der eine weitere Unterteilung vornimmt. Alle von diesem Dekoder erzeugten Signale werden wieder über PHASE0 geschaltet, sie sind im Zustand "0" aktiv. Die Bausteine des Dekoders sind in LSTTL-Technik aufgebaut, die Signallaufzeiten fallen deshalb erheblich kürzer aus als innerhalb der in MOS-Technik hergestellten MMU. Die wichtigsten vom peripheren Adreßdekoder erzeugten Signale sind:

(C800-CFFF)'	I/O STROBE'
C1XX'-C7XX	I/O SELECT's
C09X'-C0FX'	DEVICE SELECT's
C07X'	TIMER TRIGGER'
C06X'	SERIAL INPUT ENABLE'
C04X'	C040 STROBE'
C0XX'	IOU DECODE ENABLE'

Die I/O SELECT'-, DEVICE SELECT'-Signale und I/O STROBE' sind mit den Steckplätzen verbunden und kontrollieren eingesteckte Zusatzkarten. Theoretisch wäre es auch denkbar, daß Zusatzkarten eine eigene Adreßdekodierung enthalten, die Erzeugung der SELECT-Signale auf der Hauptplatine macht aber nicht nur die Verwendbarkeit einer Karte in einem beliebigen Steckplatz möglich, sondern auch mehrere (im Maximalfall: 7) getrennte Dekodierungen unnötig.

$C07X' startet die vier Timer für die Auswertung der Paddles, wenn der Prozessor eine Adresse im Bereich $C07X anspricht; C06X' startet eine weitere Dekodierung, die den Zustand von einem der acht seriellen Eingänge auf MD7 des Datenbusses legt (s. Bild 7.2). C04X' ist direkt mit Pin 5 des GAME I/O verbunden und stellt das Signal C040 STROBE'.

Das Signal C0XX' ist mit Pin 30 der IOU verbunden und startet eine weitere (interne) Dekodierung. Hier wird auch ein Unterschied zwischen IOU und MMU deutlich: die MMU überwacht den gesamten Adreßbus und erzeugt die Signale zur Schaltung des Datenbusses, mit denen die Speicherkonfiguration des Apple festgelegt wird. Die IOU dagegen überwacht lediglich Teile des Bereichs $C0XX und kontrolliert darüber begrenzte Teile des I/O und der Videoausgabe.

Die Hardware des peripheren Adreßdekoders ist sehr einfach, sie besteht aus einem NAND-Gatter, einem 3 zu 8 Dekoder (74LS183) und einem 4 zu 16 Dekoder (74LS154); den vollständigen Schaltplan finden Sie in Bild 7.1. Das NAND-Gatter bringt I/O STROBE' während PHASE0 auf den Pegel "0", wenn CXXX und A11 den Zustand "1" haben. I/O STROBE' ist somit aktiv, wenn die CPU auf Adressen im Bereich $C800-$CFFF zugreift und INTCXROM sowie INTC8ROM zurückgesetzt sind.

Wenn A11 während PHASE0 "0" ist, aktiviert der 3 zu 8 Dekoder eines der acht CnXX'-Signale. Diese acht Signale sind die sieben I/O-SELECT'-Leitungen und C0XX'. C0XX' aktiviert eine weitere Dekodierung im 4 zu 16 Dekoder und innerhalb der IOU. Der 4 zu 16 Dekoder erzeugt die sieben DEVICE SELECT's ($C0nX, n von $9 bis $F) sowie C04X', C06X' und C07X'. Innerhalb des Bereichs $C0XX wird also schlicht in 16 einzelne Bereiche unterteilt ($C00X, $C01X, $C02X..), einige Bereiche bzw. die vom 4 zu 16 Dekoder erzeugten Signale dazu werden nicht genutzt - deren Dekodierung findet innerhalb der IOU statt.

In der Verbindung des peripheren Adreßdekoders gibt es einige interessante Kleinigkeiten zu entdecken:

1. CXXX muß aktiv sein, bevor eine weitere Dekodierung stattfinden kann. Damit ist es der MMU möglich, sämtliche in Bild 7.1 gezeigten Signale zu sperren, indem sie CXXX nicht aktiviert. Die einzigen Peripheriesignale, die von der MMU zeitweilig gesperrt werden, sind aber I/O STROBE' und die I/O SELECT's (als Folge des Standes von INTCXROM, SLOTC3ROM und INTC8ROM).
2. PHASE1 ist mit einem Aktivierungseingang des 3 zu 8 Dekoders verbunden, der auf den Pegel "0" reagiert. Bevor eine Dekodierung für den Bereich $C000-$C7FF stattfinden kann, muß also PHASE1 "0" bzw. PHASE0 "1" sein. Wieso wird der 4 zu 16 Dekoder, der ja über den 3 zu 8 Dekoder erst geschaltet wird, noch einmal über PHASE1 aktiviert bzw. gesperrt? Die Antwort: Diese direkte Verbindung hat eine schnellere Sperrung der vom 4 zu 16 Dekoder erzeugten Signale zur Folge, wenn PHASE1 wieder auf den Pegel "1" geht. Der sich ergebende Signalverlauf ist damit fast identisch zu dem des Apple II: die DEVICE SELECT's und C06X' enden exakt zusammen mit PHASE0.[1]

---

1 Die Kompatibilität zwischen //e und II(+) ist erfreulich - ich hätte es allerdings für besser gehalten, wenn in beiden Maschinen die DEVICE SELECT's etwas über PHASE0 hinausgehen würden. Da von diesen Signalen der Hauptanteil der Datenbuszuordnung innerhalb des Peripheriebusses abhängig ist, hätte man mit ihnen eigentlich die fallende Flanke von PHASE2 des 6502 etwas verlängern können.

3. Der periphere Adreßdekoder ist nicht über R/W' geschaltet. Aus diesem Grund kann ein Zugriff auf eine I/O-Adresse sowohl mit einer Lese- als auch mit einer Schreibaktion des Prozessors erfolgen. Vermeiden sollte man lediglich Schreibaktionen im Bereich C06X, weil hier eine Kollision zwischen dem Multiplexer der seriellen Eingänge und dem bidirektionalen peripheren Datenbustreiber auf D7 stattfindet.

Die Softswitches der IOU

Wenn man die Erzeugung des Videosignals einmal außer acht läßt, hat die IOU nicht übermäßig viel mit dem eingebauten I/O des Apple //e zu tun. Man könnte die IOU ohne Untertreibung als Apple-spezifischen Videocontroller bezeichnen, in den beim Entwurf noch ein paar heimatlose I/O-Funktionen hineingerutscht sind. Dazu gehören zwei Umschalter für Kassettenausgang und Lautsprecher, die vier Announciator-Softswitches, der Tastatur-Softswitch und etwas Elektronik zum Lesen des Softswitches AKD (Any Key Down). Eine Darstellung dieser Funktionen zusammen mit den Softswitches für die Videomodi finden Sie in Bild 7.1, das eine vollständige Darstellung sämtlicher Softswitches und Adreßdekodierungen der IOU ist.

Tabelle 7.1 gibt die Kontrolladressen der IOU-Softswitches wieder. Die meisten der Switches haben eine Setz- und eine Rücksetzadresse, ausgenommen **SPKR** und **CSSTOUT**, die durch Ansprechen des Bereichs $C03X bzw. $C02X jeweils umgeschaltet werden, sowie **KEYSTROBE**, der durch einen Tastendruck oder die Autorepeat-Funktion gesetzt und durch Schreibzugriffe auf den Bereich $C01X oder durch einen Lesezugriff auf $C010 zurückgesetzt wird. AKD und VBL' sind keine programmierbaren Softswitches, können aber auf dieselbe Weise wie Softswitches vom Prozessor gelesen werden, ihr Stand wird bei Lesezugriffen auf $C010 bzw. $C019 von der IOU auf MD7 des Datenbusses gelegt.

Mit den Softswitches 80STORE, 80COL, ALTCHRSET, TEXT, MIXED, PAGE2 und HIRES wird die Erzeugung des Videosignals kontrolliert. Ihr Stand beeinflußt den Speicherbereich, der vom Videoscanner abgefragt wird (s. Bild 5.3) und/oder die Erzeugung der Taktsignale (Bild 3.9) sowie die Adreßlage des Video-ROMs (Bild 8.5). Wir werden uns in Kapitel 8 noch ausführlich mit diesen Softswitches beschäftigen.

Die Announciators, CSSTOUT und SPKR sind Ausgangssignale der IOU. CSSTOUT und SPKR werden noch elektronisch umgeformt bzw. verstärkt, die vier Announciators sind direkt von der IOU aus mit den entsprechenden Pins des GAME I/O verbunden.

Der Softswitch KEYSTROBE hat nur eine einzige Funktion, nämlich die Anzeige für die CPU, daß eine Taste gedrückt wurde (oder die Autorepeat-Funktion rasch aufeinanderfolgende Tastendrücke simuliert (s. Bild 3.8)). Anders gesagt: KEYSTROBE hat keinerlei Kontrollfunktion innerhalb der IOU und existiert nur als Flag für die CPU. In derselben Weise ist das Signal AKD mehr oder weniger durch die IOU "hindurchgeführt", damit es von der CPU gelesen werden kann.

Bei einem Lesezugriff auf den Bereich $C000-$C01F setzt die MMU KBD' aktiv und der Tastatur-ROM legt den ASCII-Wert für die zuletzt gedrückte Taste auf MD0-$MD6 des Datenbusses. MD7 wird entweder von KEYSTROBE ($C00X), AKD ($C010) oder von einem der zusammen mit dem ASCII-Wert lesbaren Softswitches im Bereich $C011-$C01F gestellt. Für die CPU ergibt sich damit immer ein 8-Bit-Wort, das in den sieben niederwertigen Bits den ASCII-Wert der Tastatur und im höchstwertigen Bit den Stand eines Flags der IOU oder der MMU enthält.

Wie auch alle anderen Schaltpläne dieses Buches, in denen Einzelheiten des Innenlebens von MMU oder IOU dargestellt werden, ist auch Bild 7.1 eine funktionale Repräsentation der logischen Zusammenhänge, die sich hauptsächlich auf meine eigenen Messungen gründet. Es bleibt zu hoffen, daß die dargestellten Schaltkreise den tatsächlich verwendeten äquivalent sind, die tatsächliche Realisierung und Anordnung einzelner Gatter, Flipflops etc. dürfte mit Sicherheit eine andere sein. Im folgenden einige Erklärungen zu funktionellen Details der IOU:

1. Alle Softswitches außer KEYSTROBE, TEXT und MIXED werden direkt durch einen aktiven Pegel von RESET' zurückgesetzt. So wie es scheint, werden sämtliche Softswitches durch das Einschalten der Stromversorgung zurückgesetzt.
2. AL0-AL5 und AL7 sind die Werte der Adreßleitungen A0-A5 und A7. Sie werden von der IOU über den RAM-Adreßbus während PHASE0 gelesen, die Übernahme erfolgt auf die fallende Flanke von RAS' (s. Bild 5.3).
3. Ähnlich wie die MMU scheint auch die IOU ein zeitliches Fenster zu haben, in dem die Signale C0XX', A6, R/W', AL0-AL5 und AL7 daraufhin überprüft werden, ob mit dem laufenden Zyklus ein Softswitch gesetzt oder gelesen werden soll. Wenn eine derartige Konstellation während des Fensters mehr als 40 ns

stabil gehalten wird, reagiert der dadurch adressierte Softswitch. Soweit ich das bestimmen konnte, wird das Fenster durch PHASE0 * RAS'' * Q3 bestimmt. Während dieser Zeit sind sowohl vom 6502 ausgegebene Adressen als auch ein darauffolgendes C0XX' durchgehend gültig.

4. AKD wird um zwei CPU-Zyklen verzögert, bevor dieses Signal über MD7 erreichbar ist. KSTRB wird ebenfalls um zwei CPU-Zyklen verzögert, danach wird die Länge dieses Pulses genau auf einen CPU-Zyklus gebracht, bevor damit KEYSTROBE gesetzt wird. Soweit ich das abschätzen kann, hat diese Verzögerung keine sinnvolle Funktion: zwischen dem Setzen von AKD und der Gültigkeit des vom Tastatur-ROM ausgegebenen ASCII-Zeichens liegen rund 10 Millisekunden, zwischen dem KSTRB-Puls und dem gültigen ASCII sind es 20 Millisekunden. Die Begrenzung der Länge von KSTRB auf einen CPU-Zyklus verhindert, daß KEYSTROBE durch einen KSTRB-Puls mehr als einmal gesetzt wird. Wie dem auch sei - die Verzögerungsfunktion wird nach meinen Messungen durch die fallende Flanke von RAS' während PHASE0 ausgelöst.

5. Die Umschaltung des Pegels von CSSTOUT und SPKR scheint dann ausgelöst zu werden, wenn C02X bzw. C03X zum Zeitpunkt der steigenden Flanke von RAS' während PHASE0 an der IOU gültig ist. Die eigentliche Umschaltung findet allerdings nicht unmittelbar danach, sondern erst mit der fallenden Flanke von RAS' während PHASE1 statt.

6. Wenn der Prozessor den Stand eines IOU-Softswitches liest, legt die IOU diesen Stand offensichtlich im Zeitraum von PHASE0 * Q3' + PHASE0' * Q3 * RAS' auf MD7 des Datenbusses. Dieser Term ist während der letzten drei 14M-Perioden von PHASE0 und während der ersten 14M-Periode der folgenden PHASE1 gültig.

Tabelle 7.1 Kontrolladressen der IOU-Softswitches

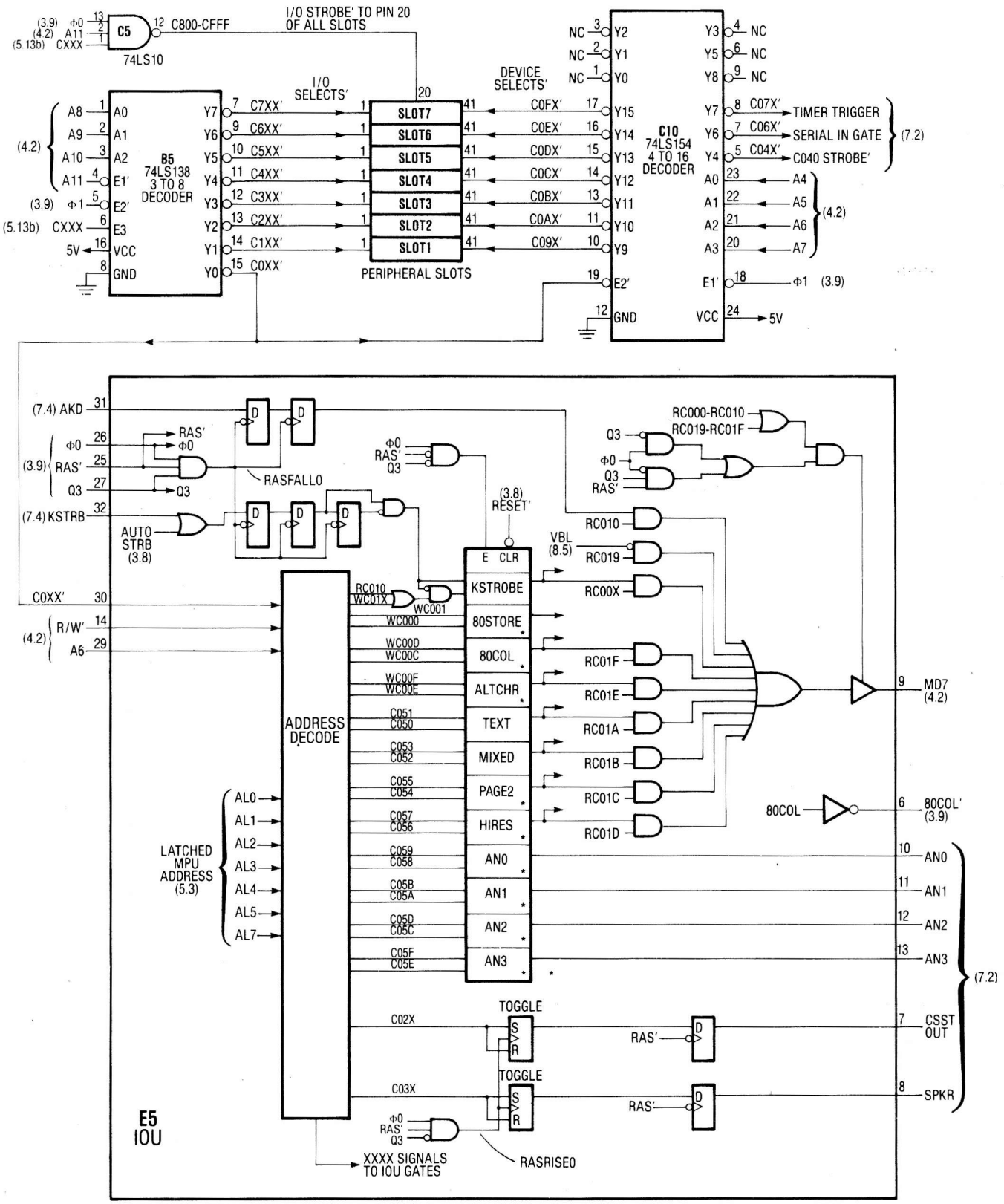
SOFT SWITCH	OFF ADDRESS	ON ADDRESS	READ ADDRESS	CONDITION AFTER RESET'
KEYSTROBE	$C010/ W$C01X	KSTRB/ AUTOSTRB	R$C00X	NA (NOT APPLICABLE)
80STORE*	W$C000	W$C001	R$C018	PAGE2 SWITCHES DISPLAY AREA
80COL	W$C00C	W$C00D	R$C01F	SINGLE-RES DISPLAY
ALTCHRSET	W$C00E	W$C00F	R$C01E	FLASHING TEXT ACTIVE
TEXT	$C050	$C051	R$C01A	NA
MIXED	$C052	$C053	R$C01B	NA
PAGE2*	$C054	$C055	R$C01C	PAGE1
HIRES*	$C056	$C057	R$C01D	LORES
AN0	$C058	$C059	NA	OFF
AN1	$C05A	$C05B	NA	OFF
AN2	$C05C	$C05D	NA	OFF
AN3	$C05E	$C05F	NA	OFF
CSSTOUT**	$C02X	$C02X	NA	NA
SPKR**	$C03X	$C03X	NA	NA
AKD	NA	NA	R$C010	NA
VBL'	NA	NA	R$C019	NA

Anmerkungen:

* PAGE2, HIRES und 80STORE existieren innerhalb der MMU ein zweites Mal. Der Stand von 80STORE wird von der MMU bei einer Leseoperation von C018 auf MD7 ausgegeben, der Stand von PAGE2 und HIRES dagegen von der IOU.

** CSSTOUT und SPKR wechseln die Polarität, wenn ihre Kontrolladresse angesprochen wird.

Bild 7.1 Schaltplan der IOU-Softswitches und des peripheren Adreßdekoders



* kennzeichnet Softswitches, die automatisch zurückgesetzt werden, wenn RESET' den Pegel "0" hat. Durch Einschalten der Stromversorgung werden alle Softswitches zurückgesetzt.

Der Aufbau des seriellen I/O

Bild 7.2 zeigt den Schaltplan der seriellen I/O-Baugruppen. Die tatsächliche Verbindung zur Außenwelt findet bei den meisten Ein- und Ausgängen über den GAME I/O (J15) bzw. den Steckverbinder für die Spielsteuerungen auf der Rückseite des Gehäuses (J8) statt. Über ihre Steckkontakte lassen sich externe Geräte wie Joysticks mit Druckknöpfen anschließen. Der GAME I/O enthält zusätzlich noch die vier Announciator-Ausgänge und das Signal C040 STROBE'. Diese fünf Signale kommen direkt von den Ausgängen der IOU.

Die seriellen Eingänge des //e sind über den seriellen Eingangsmultiplexer (74LS251) mit D7 des peripheren Datenbusses verbunden. Der LS251 ist ein 8 zu 1 Multiplexer, dessen tri-State-Ausgang durch den Pegel "0" der Leitung C06X' aktiviert wird. Die Adreßbits A0, A1 und A2 werden als Multiplexadresse verwendet, womit sich acht aufeinanderfolgende Adressen im Bereich C060-$C067 ergeben, die jeweils einen anderen Eingang abfragen. Bei einem Lesezugriff der CPU auf den Bereich $C06X setzt die MMU das Signal MD IN/OUT' auf "1", und der adressierte serielle Eingang wird über den peripheren Datenbus auf MD7 gebracht. Da die oberen acht Adressen des Bereichs $C06X nicht noch einmal extra dekodiert sind, kann man jeden der seriellen Eingänge über zwei verschiedene Adressen lesen ($C060/C068, $C061/C069 etc.); es ist aber allgemein üblich, nur die unteren Adressen (also $C060-$C067) zu benutzen. Die acht Eingänge und ihre dazugehörigen Adressen sind:

Kassetteneingang	$C060/$C068
Druckknopf 1	$C061/$C069
Druckknopf 2	$C062/$C06A
Druckknopf 3	$C063/$C06B
Paddle 0	$C064/$C06C
Paddle 1	$C065/$C06D
Paddle 2	$C066/$C06E
Paddle 3	$C067/$C06F

Bild 7.2 enthält zusätzlich den Schaltplan eines angeschlossenen Joysticks. Der Anschluß erfolgt hier über den GAME I/O - die restlichen seriellen Ein- und Ausgänge werden dadurch unzugänglich, solange man einen marktgängigen Joystick mit einem 16-poligen Anschlußstecker verwendet. Seit es den //e gibt, sind auch Joysticks auf dem Markt, die den Steckverbinder auf der Gehäuserückseite benutzen und dabei nicht alle anderen Anschlüsse des GAME I/O blockieren - allerdings folgen auch diese Geräte einem ungeschriebenen Gesetz der Apple-Welt, immer nur PDL0, PDL1, PB0 und PB1 zu benutzen. Die anderen Anschlüsse sind von kommerziellen Programmen mit ganz wenigen Ausnahmen nie genutzt worden - warum, weiß eigentlich niemand.

Die drei "Action-Tasten"-Eingänge (PB0, PB1 und PB2) sind tatsächlich einfache TTL-Eingänge des seriellen Eingangsmultiplexers. Im Prinzip könnte man sie für alle möglichen Arten serieller TTL-Signale benutzen. PB0 ist im Apple //e zusätzlich mit der offenen Apfeltaste, PB1 mit der geschlossenen Apfeltaste verbunden. Beide Eingänge werden an der Tastatur über Widerstände mit jeweils 470 Ohm auf den Pegel "0" gezogen (s. Bild 7.4). Wenn eine der Apfeltasten gedrückt wird, verbindet sie die entsprechende Leitung mit +5 Volt, setzt sie also auf "1".

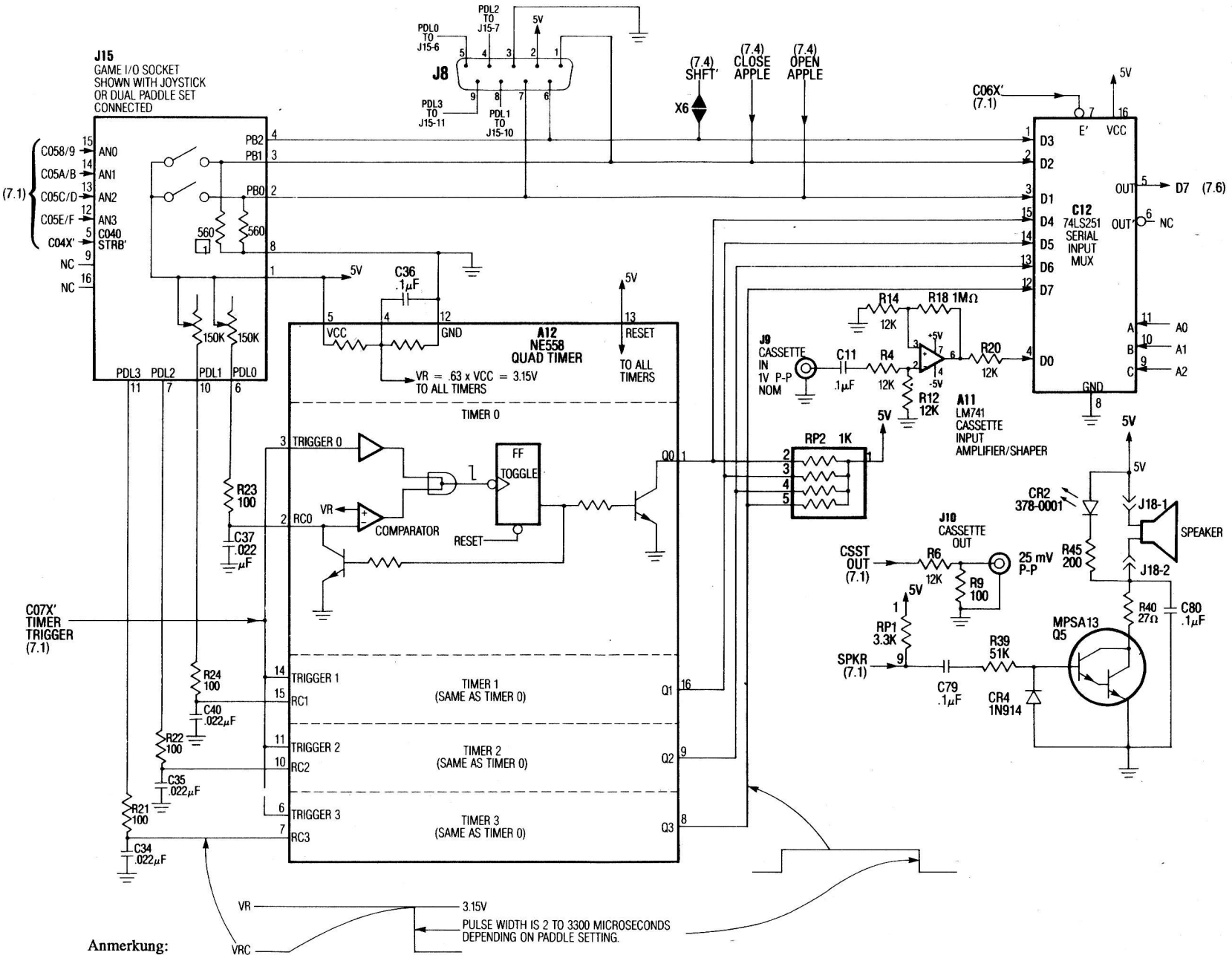
Zusätzlich zu den "pull-down"-Widerständen an der Tastatur haben die meisten Joysticks an ihren "Action-Tasten" noch einmal Widerstände für PB0 und PB1. Sie sind für den Betrieb am Apple II(+) auch notwendig - die Vorläufer des //e haben keine internen Widerstände an diesen Leitungen. Ihr Wert liegt je nach Hersteller zwischen 200 und 1000 Ohm.

Damit ergeben sich zwei Merkwürdigkeiten für den Apple //e:

1. Der Anschluß einer externen Tastatur ohne die Apfeltasten ist ohne Modifikationen nur dann möglich, wenn gleichzeitig ein Joystick mit "pull-down"-Widerständen angeschlossen ist. Ansonsten befinden sich die Leitungen PB0 und PB1 ständig auf dem Pegel "1". Folge: Der Apple führt nach dem Einschalten und nach jedem RESET einen Selbsttest durch, bis zum Absuchen der Steckplätze nach einem Diskettencontroller kommt er nie.

2. Wenn ein Joystick mit eingebauten "pull-down"-Widerständen angeschlossen ist, liegen diese Widerstände parallel zu den in der Tastatur bereits eingebauten. Im schlimmsten Fall (d.h. mit 200-Ohm-Widerständen im Joystick) ergibt sich damit bei gleichzeitigem Druck auf beide Apfeltasten eine zusätzliche Belastung des Netzteils von mehr als 0.1 Ampere - die dadurch hergestellte Verbindung aller "pull-down"-Widerstände von +5 Volt nach Masse hat gerade noch einen Gesamtwert von 45 Ohm. Für die meisten Netzteile stellt das kein Problem dar - falls Sie aber einen mit Zusatzkarten vollgestopften Apple besitzen, der bei einem Druck auf die Action-Tasten oder die Apfeltasten manchmal "abstürzt", dann ist der Grund hier zu suchen.

Bild 7.2 Schaltplan der seriellen I/O-Baugruppe



Anmerkung:

(1) Je nach Ausführung des Joysticks kann der Wert dieser Widerstände unterschiedlich sein, manchmal fehlen sie ganz.

Der serielle Eingangsmultiplexer ist ein gutes Beispiel, wie der Aufbau eines digitalen Computers in Achtergruppen (oder anderen Zweierpotenzen) vor sich geht. Drei Adreßleitungen können zusammen acht verschiedene Zustände annehmen - folglicherweise stehen dem Designer Multiplexer zur Verfügung, die nach dem Schema "8 zu 1" (und nicht 9 zu 1 oder 10 zu 1 etc.) aufgebaut sind. Dabei kommt dann ein Computer heraus, der "natürlicherweise" acht Eingänge hat, ungeachtet der Tatsache, daß dabei vier Paddle-Eingänge, aber nur drei Druckknöpfe angeschlossen werden können - irgendwo mußte noch der Kassetteneingang hin.

Es ist möglich, die Shift Key-Modifikation über die Lötverbindung X6 der Hauptplatine herzustellen. Diese Modifikation besteht aus einer Verbindung der SHIFT-Tasten mit PB2 des GAME I/O und rührt vom Apple II(+) her, dessen Tastatur nur Großbuchstaben erzeugen kann. Ein Programm kann über diese Modifikation durch Lesen von PB2 bestimmen, ob die SHIFT-Taste zusammen mit einem Buchstaben gedrückt wurde, und darüber zwischen eingegebenen Groß- und Kleinbuchstaben unterscheiden. Falls Sie im Besitz von Programmen sind, die sich um "echte" Groß- und Kleinbuchstaben nicht kümmern und statt dessen diese Modifikation benötigen, können Sie sie nach Verbindung der Lötbrücke X6 auch auf dem //e benutzen. Falls Sie PB2 überhaupt nicht benutzen, können Sie die Modifikation trotzdem durchführen: Sie können danach alle drei "Action-Tasten" über die Tastatur auslösen.

Die vier Paddle-Eingänge (PDL0-PDL3) des GAME I/O sind mit einem Vierfach-Timer verbunden. Diese Eingänge reagieren im Gegensatz zu den "Action-Tasten" nicht auf die Pegel "0" und "1" - sie haben zwar eine digitale Verbindung zum seriellen Eingangsmultiplexer, sind aber auf der Eingangsseite nicht digital.

Bevor wir uns um die elektronischen Einzelheiten kümmern, erst einmal etwas zur Bedienung und den Möglichkeiten dieser Eingänge: vor einer Auswertung werden alle vier Zeitkreise durch einen Zugriff auf $C07X gestartet. Alle vier Ausgänge zum seriellen Eingangsmultiplexer gehen daraufhin auf "1" - wann ein Zeitkreis-Ausgang wieder auf "0" zurückgeht, hängt vom Stand des angeschlossenen Joysticks ab. Ein Programm kann diese Zeit messen und darüber den Stand eines Paddles oder Joysticks ermitteln.

Etwas technischer: Ein Paddle besteht aus einem Potentiometer (einem veränderlichen Widerstand), ein Joystick besteht aus zwei kreuzförmig zueinander angeordneten Potentiometern. Jeder der Timer enthält einen Kondensator von 0.22 Mikrofarad, der durch den Startimpuls schlagartig aufgeladen und über das angeschlossene Potentiometer langsam wieder entladen wird ("R/C-Glied"). Je höher der Widerstand des Potentiometers ist, desto länger dauert der Entladevorgang. Der Maximalwert für Paddles und Joysticks ist 150000 Ohm, der Minimalwert wird über einen Widerstand des Timers auf 100 Ohm festgelegt. Die damit erreichbaren Zeiten liegen zwischen 2 (100 * 0.22) und 3300 (150100 * 0.22) Mikrosekunden.

Der NE558/SE558, der die vier Zeitkreise enthält, hat einen RESET-Eingang, der alle vier Ausgänge zum Eingangsmultiplexer auf den Pegel "0" bringt und einen erneuten Start verhindert. Dieser Eingang wird im Apple nicht genutzt.[2] Jeder Zeitkreis des Bausteins hat einen eigenen Start-Eingang ("trigger"), im Apple sind alle vier Eingänge mit dem Signal C07X' verbunden, das Starten eines einzelnen Zeitkreises ist also nicht möglich. Um RESET und die Möglichkeit von Einzelstarts zu nutzen, hätte man eine weitere Adreßdekodierung durchführen müssen, die dafür benötigten Bausteine hatten auf der Hauptplatine des Apple II(+) einfach keinen Platz mehr. So wie die Zeitkreise nun einmal geschaltet sind, kann man dafür mehrere Paddle-Eingänge auf einmal auswerten, der fehlende RESET-Eingang erzwingt eine Warteschleife vor der Auslese, um sicherzustellen, daß vor einem erneuten Start auch wirklich alle Zeitkreise "abgelaufen" und nicht noch von einer vorherigen Abfrage her aktiv sind. Leider fanden beide technischen Gegebenheiten in der Routine PREAD keine Berücksichtigung.

Der Aufbau mit einem Vierfach-Timer ist ein fast genialer Weg, einen billigen Analog-/Digitalwandler mit vier Eingängen zu konstruieren. An diesen Wandler kann man veränderliche Widerstände anhängen, die auf Licht, Temperatur, lineare oder kreisförmige Bewegungen, chemische Zusammensetzungen und wahrscheinlich noch alles mögliche andere reagieren. Anders gesagt: Alle diese Prozesse ließen sich mit einem Apple über die vier Timer-Eingänge verfolgen. Zugegeben, es dauert einen Moment, die Eingänge zu lesen (nämlich 22 Mikrosekunden pro 1000 Ohm) - aber um wieviel ändert sich eine Temperatur in 1000 oder 5000 Mikrosekunden?

Der Zweck, für den diese Eingänge wirklich überhaupt nicht geeignet sind, ist derjenige, zu dem sie in den allermeisten Fällen verwendet werden: als Spielkontrollen in einem "Arcade"-Spiel, bei dem es auf Reaktionszeiten ankommt. Ein gutes Arcade-Spiel mit hochauflösender Grafik auf dem Apple kann nur mit allen Haken und Ösen programmiert werden, die einem erstklassigen Programmierer zur Verfügung stehen - der 6502 braucht für einen Befehl im Schnitt um die 3.5 Mikrosekunden, das Auslesen der Paddles dauert rund fünfhundertmal länger. Damit ist die Geschwindigkeit eines Arcade-Spiels zu einem guten Teil indirekt proportional zur Zeit, die während der

---

2 Eigentlich hätte ein schlichter Kondensator zwischen Triggereingang und RESET genügt. Aus der negativen Flanke des C70X'-Triggerimpulses würde der ein (positives) RESET herausdifferenzieren - damit wäre sichergestellt, daß alle vier Timer vor einer Messung zurückgesetzt sind -(Anm. d. Übers.).

Paddle-Auslese verschenkt wird. Wenn der Apple //e ohne Rücksicht auf Kompatibilität zum II(+) entworfen worden wäre, dann hätte er mit Sicherheit anstelle der Timer einen schnellen, mehrkanaligen A/D-Wandler spendiert bekommen.

Der achte Anschluß des seriellen Eingangsmultiplexers ist der Kassetteneingang. Der Stecker auf der Gehäuserückseite führt zu einem Operationsverstärker, der das relativ kleine Signal vom Kopfhörerausgang des Kassettenrecorders nicht nur erheblich verstärkt, um daraus die Pegel "0" und "1" zurückzugewinnen, sondern auch gleichzeitig noch eine "Kurvenformung" vornimmt. Man kann zwar einen angeschlossenen Kassettenrecorder immer noch zu laut oder zu leise stellen und so die Übertragung unsicher machen, der Eingangsverstärker macht diese Einstellung aber wesentlich unkritischer.

Die verwendete Konstruktion ist etwas ungewöhnlich - der Operationsverstärker LM741 arbeitet nicht mit der üblichen negativen Gegenkoppplung (was bei dem verwendeten Verstärkungsfaktor auch nicht praktikabel wäre), er ist stattdessen *mitgekoppelt* und befindet sich ständig im positiven oder negativen Sättigungsbereich. (An dieser Stelle eine leise Entschuldigung für das verwendete Elektronikerchinesisch, aber schließlich `ist` das ein elektronischer Schaltkreis.) Er funktioniert folgendermaßen:

1. Der vor dem Verstärker liegende Kondensator C10 läßt nur den Wechselspannungsanteil des Eingangssignals durch. Die am Eingang des 741 anliegenden Spannungswerte liegen damit im Bereich unterhalb und oberhalb von 0 Volt.
2. In der gegebenen Schaltung arbeitet der LM741 wie ein Schwellwert-Detektor. Wenn die Spannung an Pin 2 über 0,15 Volt geht, erzeugt er die maximal mögliche negative Spannung (ca. -4.3 Volt), die mit seiner Versorgungsspannung von -5 Volt möglich ist; wenn die Spannung an Pin 2 unter -0,15 Volt fällt, erzeugt er ca. + 4,3 Volt am Ausgang. Solange die Spannung vom Kassettenrecorder sich nicht allzulange im Bereich von -0,15 bis +0,15 Volt befindet, verstärkt der LM741 damit das vom Kassettenreorder gelieferte Signal und formt es gleichzeitig in eine rechteckig verlaufende Schwingung um - egal, welche Kurvenform das ursprüngliche Eingangssignal hat.
3. Der Ausgang des LM741 ist mit dem letzten Eingang des seriellen Eingangsmultiplexers über einen Widerstand mit 12000 Ohm verbunden. Wenn der 741 +4,3 Volt liefert, registriert der Eingangsmultiplexer eine "1", bei -4,3 Volt schließt die integrierte Schutzdiode am Eingang des Multiplexers die negative Spannung über den Widerstand kurz (dabei fließen rund 0,36 Milliampere, d.h. fast nichts) und es wird eine "0" gelesen.

Das Schöne an diesem Kassetteneingang ist die Tatsache, daß er auch aus einem völlig verzerrten Sinussignal eines billigen Recorders wieder ein sauberes Rechteck mit den ursprünglichen Pulsbreiten erzeugt. Bild 7.3 zeigt, wie ein Recorder der Preisklasse "unter 100 Mark" ein Rechtecksignal aufzeichnet und wiedergibt. Der LM741 schaltet aber nach wie vor an Punkten, deren zeitlicher Abstand proportional zu den vorher aufgezeichneten Nullen und Einsen ist und reproduziert so ein sauberes Signal. Natürlich gibt es auch hier Grenzen - aber einen Klirrfaktor von 30% übersteht das Verfahren schadlos.

Das *Technical Reference Manual* für den //e gibt an, daß der Kassetteneingang des Computers mit einer Spannung von 1 Volt Spitze-Spitze betrieben werden sollte, also ziemlich genau mit dem, was so ein Kopfhörerausgang meistens liefert.

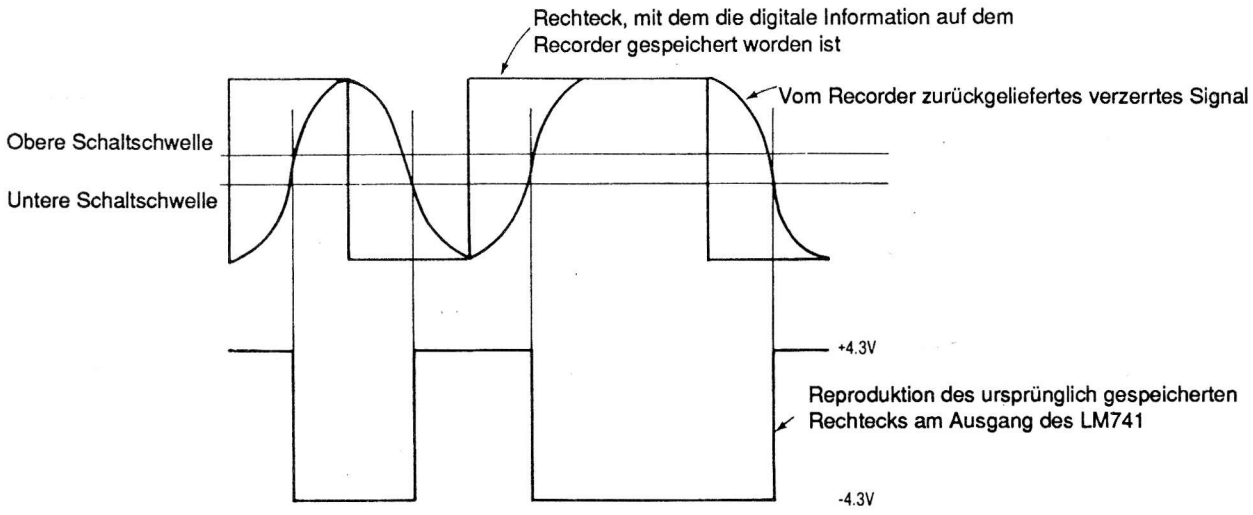
Der Kassettenausgang des Apple liefert dagegen eine wesentlich geringere Spannung, um den Eingangsverstärker eines angeschlossenen Recorders nicht zu überfordern: das von der IOU gelieferte Signal CSSTOUT wird durch den Faktor 121 verkleinert. Es ändert seine Polarität bei jedem Ansprechen von $C02X. Wenn man davon ausgeht, daß die IOU ungefähr einen Spannungsunterschied von 3 Volt zwischen "0" und "1" liefert, ergeben sich damit am Kassettenrecorder-Ausgang Spannungsänderungen um 0,025 Volt.

Das Signal SPKR von Pin 8 der IOU wechselt in derselben Weise wie CSSTOUT bei jedem Ansprechen seiner Kontrolladresse die Polarität. Es ist mit dem Eingang eines einfachen Audioverstärkers verbunden, der die nötige "Power" für den Lautsprecher des Apple liefert - die IOU wäre mit der Erzeugung der benötigten Ströme etwas überfordert. Der Stromfluß durch den Lautsprecher findet nur in einer Richtung statt (der Verstärker liefert nur keine oder positive Spannung und hat keinen Ausgangskoppelkondensator), die Membran wird also zwischen "gespannt" und "entspannt" geschaltet und nicht zwischen "Auslenkung vorwärts" und "Auslenkung rückwärts". Aufeinanderfolgendes Ansprechen von $C03X bewegt so die Membran vorwärts und wieder zurück - ein Programm kann allerdings nicht feststellen, in welchem Zustand sich der Lautsprecher gerade befindet. Damit sind die Möglichkeiten der Erzeugung komplexerer Töne leider unnötig eingeschränkt.

Egal, wie der Lautsprecher zu einem gegebenen Zeitpunkt steht - die Erzeugung einer einzigen Schwingung erfordert einen zweimaligen Zugriff auf $C03X, sie besteht aus einer "Anspannungshälfte" und einer "Entspannungshälfte". Um z.B. einen Ton mit 1000 Hz zu erzeugen, sind 2000 Zugriffe auf $C03X pro Sekunde erforderlich.

Parallel zum Lautsprecher des Apple //e ist eine LED ("Light Emitting Diode" = Leuchtdiode) geschaltet, die normalerweise nicht leuchtet, weil der Spannungsabfall über den Lautsprecher zu klein ist. Wenn man dagegen die Verbindung zum Lautsprecher unterbricht, ist der Spannungsabfall groß genug und die LED leuchtet, wenn der Apple sonst einen Piepton produzieren würde. Sie können das einmal ausprobieren, wenn Sie den Lautsprecherstecker abziehen und danach die Apfeltasten zusammen mit Ctrl und RESET drücken. Dadurch wird ein Teil der Selbsttest-Routinen zur Ausführung gebracht, die auch den Lautsprecher prüft - die LED bietet eine Möglichkeit, die Funktion der Hauptplatine ohne angeschlossenen Lautsprecher zu testen.

Bild 7.3 Signalformung am Kassettenrecorder-Eingang



Die Tastatur des Apple //e

Die meisten Leser werden sich wohl darüber im klaren sein, wie groß der Sprung von der alten Tastatur des II(+) zu der des //e ist. Das liegt allerdings mehr an den Unzulänglichkeiten des II(+) als an irgendwelchen aufregenden "Features" der neuen Tastatur, die schlicht vollständig ist. Im Gegensatz zur alten verfügt sie über Kleinbuchstaben und Umlaute; was ihr fehlt, sind ein numerischer Ziffernblock und definierbare Funktionstasten - die Apple-Fans scheinen auch ohne diese Dinge gut leben zu können.

Auch wenn die neue Tastatur äußerlich keine Besonderheiten zu bieten hat, verstecken sich einige unbekannte Möglichkeiten innerhalb der dazugehörigen Schaltkreise der Hauptplatine. Damit ist hauptsächlich die Tatsache gemeint, daß die Zuordnung zwischen Tasten und dem erzeugten ASCII-Zeichen in einem ganz normalen ROM mit 2 kByte steht - jeder, der ein 2716-EPROM programmieren kann, ist damit in der Lage, die Tastaturbelegung nach seinen Wünschen zu ändern. Außerdem ist standardmäßig eine alternative Belegung vorgesehen - die amerikanischen Tastaturen erhalten nach Umlegen des Schalters ein Dvorak-Layout (eine Anordnung der Tasten, die eine wesentlich schnellere Eingabe erlaubt), die internationalen Apples schalten zwischen dem amerikanischen und dem jeweiligen nationalen Zeichensatz um. Falls Sie ihr eigenes EPROM schießen, sind die Möglichkeiten alternativer Belegungen praktisch unbegrenzt.[3]

Bild 7.4 gibt den Schaltplan der Tastatur und der dazugehörigen Elektronik wieder. Diese Zeichnung basiert hauptsächlich auf dem im *Technical Reference Manual* für den //e veröffentlichten Schaltplan - bis auf die eigentliche Tastatur, die ich nirgendwo gefunden und deshalb aus der Tastatur meines //e "herausgeholt" habe.

Innerhalb der Tastatur befindet sich reichlich wenig Elektronik - sie besteht aus einem Haufen normalerweise offener Kontakte mit dazugehörigen Federn. Der weitaus größte Teil dieser Kontakte ist über eine XY-Matrix ange-

---

3 *Fast* unbegrenzt: Der Umschalter für die Tastaturbelegung scheint nur in den internationalen Apples Standard zu sein (in den amerikanischen Ausgaben muß man ihn erst noch einbauen). Die internationalen Apples schalten damit die Leitung ALTCHR und darüber noch die Adreßlage des Video-ROMs zwischen ASCII und dem nationalen Zeichensatz auf dem Bildschirm.

schlossen, die aus acht X- und zehn Y-Leitungen besteht. Diese Leitungen werden von einem Spezial-IC, dem Tastaturencoder, gesteuert. Die X-Leitungen funktionieren dabei als Sender, die Y-Leitungen als Empfänger. Der Encoder steuert alle X-Leitungen zyklisch der Reihe nach an und fragt gleichzeitig die Y-Leitungen ab, um darüber festzustellen, ob eine Taste gedrückt wurde. Wenn Sie z.B. die Taste 61 (im Feld oben links) drücken, erkennt der Encoder das daran, daß die Leitung Y9 durch eine Ansteuerung von X7 aktiv wird. Wenn eine Matrixverbindung als geschlossen erkannt wird, erzeugt der Encoder ein Datenwort, das über den Tastatur-ROM in ein ASCII-Zeichen übersetzt wird.

56 der 63 Tasten der Tastatur des Apple //e sind über die XY-Matrix angeschlossen, die restlichen 7 haben spezielle Funktionen: Ein Druck auf eine der Tasten SHIFT oder CTRL setzt die Leitung SHIFT' bzw. CTRL' auf aktiven Pegel. Die Taste CAPS LOCK hat eine ähnliche Funktion, nur daß sie zusätzlich noch über einen Mechanismus verfügt, der sie entweder in offener oder in geschlossener Position festhält. Wenn CAPS LOCK in der gedrückten Position festgehalten wird, dann ist die Leitung CAPLOCK' auf aktivem Pegel und der Tastatur-ROM erzeugt den ASCII-Code für Großbuchstaben. Die Apfeltasten sind völlig von der restlichen Tastaturelektronik getrennt und haben keine Verbindung zum Encoder. PB0 und PB1 haben "pull-down"-Widerstände von 470 Ohm, die Leitungen SHIFT', CTRL' und CAPSLOCK' haben "pull-up"-Widerstände von 1000 Ohm.

Die Leitung RESET' wird nur dann aktiv, wenn die Tasten RESET und CTRL gleichzeitig gedrückt werden. Durch Umsetzen der Lötbrücke auf der Tastaturplatine läßt sich das ändern. Die beiden Lötverbinder finden Sie auf der Unterseite der Tastaturplatine, sie sind entweder nach einem Ausbau der Tastatur oder nach Entfernung des Gehäusebodens (zusammen mit der Hauptplatine) zugänglich. *Durchtrennen Sie die normalerweise geschlossene Verbindung und verbinden Sie die normalerweise offene Lötbrücke. Danach funktioniert die RESET-Taste ohne gleichzeitigen Druck auf CTRL.*

Gegen diese Änderung ist wirklich nichts einzuwenden - die RESET-Taste des //e ist nicht nur von der restlichen Tastatur abgesetzt, sondern auch noch im Gehäuse eingesenkt, ein versehentliches Drücken ist damit so gut wie ausgeschlossen.

Die Leitungen SHIFT' und CTRL' sind mit Kontrolleingängen des Encoders verbunden und beeinflussen den durch die Matrix erzeugten Code. CAPLOCK ist dagegen direkt mit dem Tastatur-ROM verbunden, der damit in zwei Sektionen unterteilt ist: "Großbuchstaben" und "Groß/Klein gemischt". Die Leitung RESET' zieht sich quer durch die gesamte Hauptplatine, sie ist mit der IOU, der CPU und den "normalen" Steckplätzen verbunden. Die Tastatur ist zwar ein guter Platz für einen RESET-Taster (darum befindet er sich wohl auch hier) - mit der Elektronik der Tastatur hat die entsprechende Leitung allerdings überhaupt keine Verbindung, genausowenig wie mit den beiden Apfeltasten, deren Leitungen (PB0 und PB1) mit dem seriellen Eingangsmultiplexer verbunden und über die Kontrolladressen $C061 bzw. $C062 abfragbar sind (s. Bild 7.2).

Die Matrixknotenpunkte Y0-Y5 im Bereich X4-X7 sind auf der Tastatur nicht belegt, sie sind für den Anschluß einer externen Zusatztastatur reserviert, der über J16 mit der Hauptplatine verbunden werden kann. Diese Tastatur könnte dieselben Knotenpunkte benutzen wie die normale Tastatur - die im Apple verwendete Konstruktion ist jedoch wesentlich besser. Da ihr eigene Knotenpunkte zur Verfügung stehen, können damit unabhängig von der restlichen Tastatur beliebige Codes definiert werden.

Bild 7.4 zeigt die Standardanwendung für eine Zusatztastatur, nämlich einen numerischen Zehnerblock. Die innerhalb des Blocks gezeigten Zeichen entsprechen den Codes, die sich für diese Knotenpunkte im Tastatur-ROM des //e finden.[4] Für die externe Tastatur sind 4 * 6, also insgesamt 24 Knotenpunkte vorhanden, die über einen selbstprogrammierten Tastatur-ROM beliebig definiert werden können.

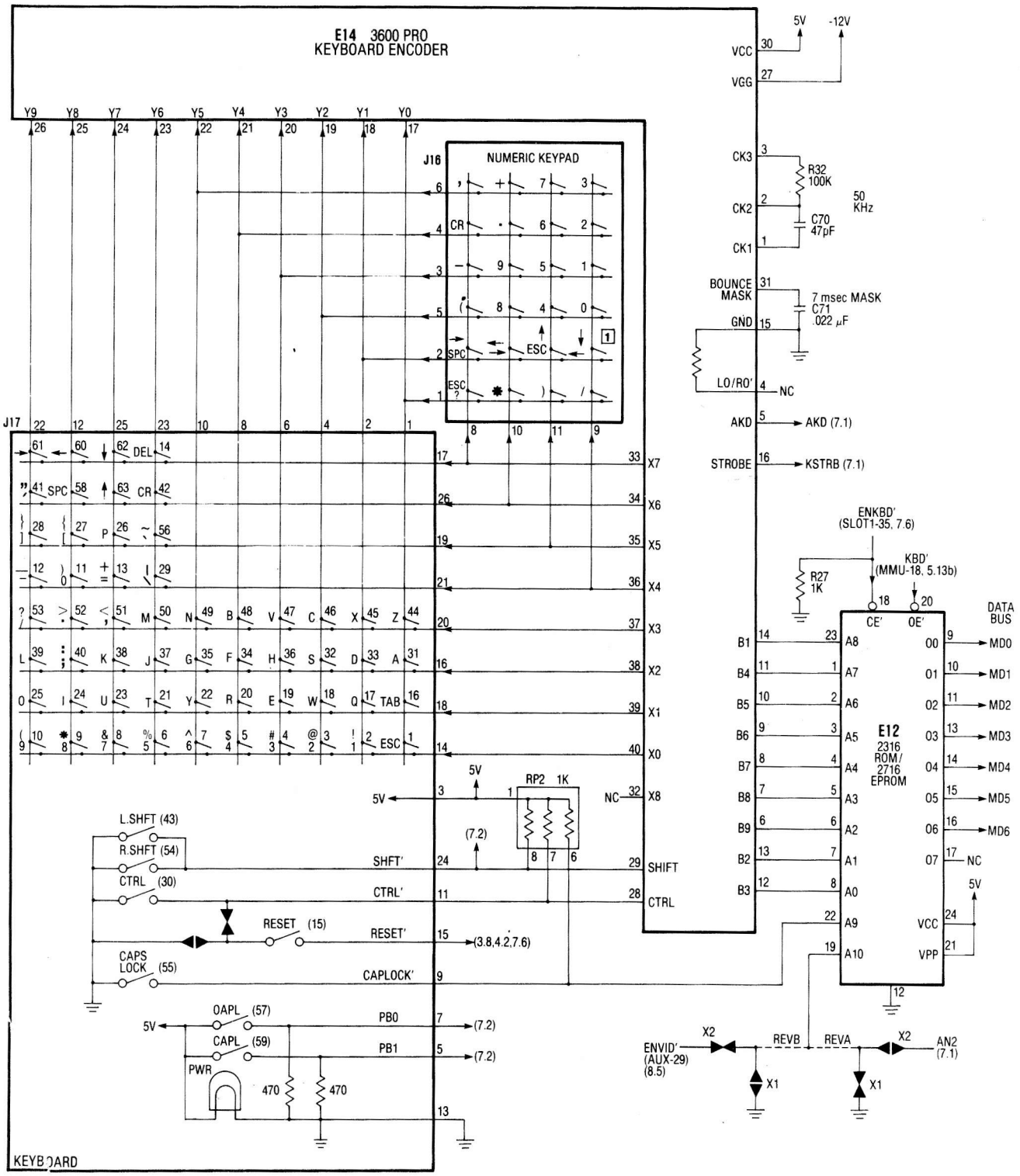
Der verwendete Tastatur-Encoder trägt die Bezeichnung 3600 PRO und ist eine spezielle Version des normalen 3600. Normalerweise kann der Kunde beim Hersteller die Tastenbelegung spezifizieren und die Funktionen der Pins 1 bis 5 (interne/externe Takterzeugung, CE', lockout/rollover' etc.) bestimmen, der 3600 wird dann entsprechend maskenprogrammiert.

Die von Apple, Inc. verwendete Version ist ein fest programmierter 3600, dessen erzeugte Codes über einen ROM in die gewünschten ASCII-Codes umgewandelt werden. Auch die Funktionen der Pins 1 bis 5 sind festgelegt: der Baustein besitzt einen internen Taktgenerator, der über die Pins 1, 2 und 3 kontrolliert wird, eine Wahlmöglichkeit zwischen "Lockout" und "Rollover" an Pin 4 mit einem internen "pull-down"-Widerstand, nicht komplementierte Ausgänge und einen AKD ("Any Key Down")-Ausgang an Pin 5.

---

4 Die Zeichnung zeigt die Zuordnung für die amerikanische Apple-Version.

Bild 7.4 Schaltplan der Apple //e-Tastatur



Anmerkungen:

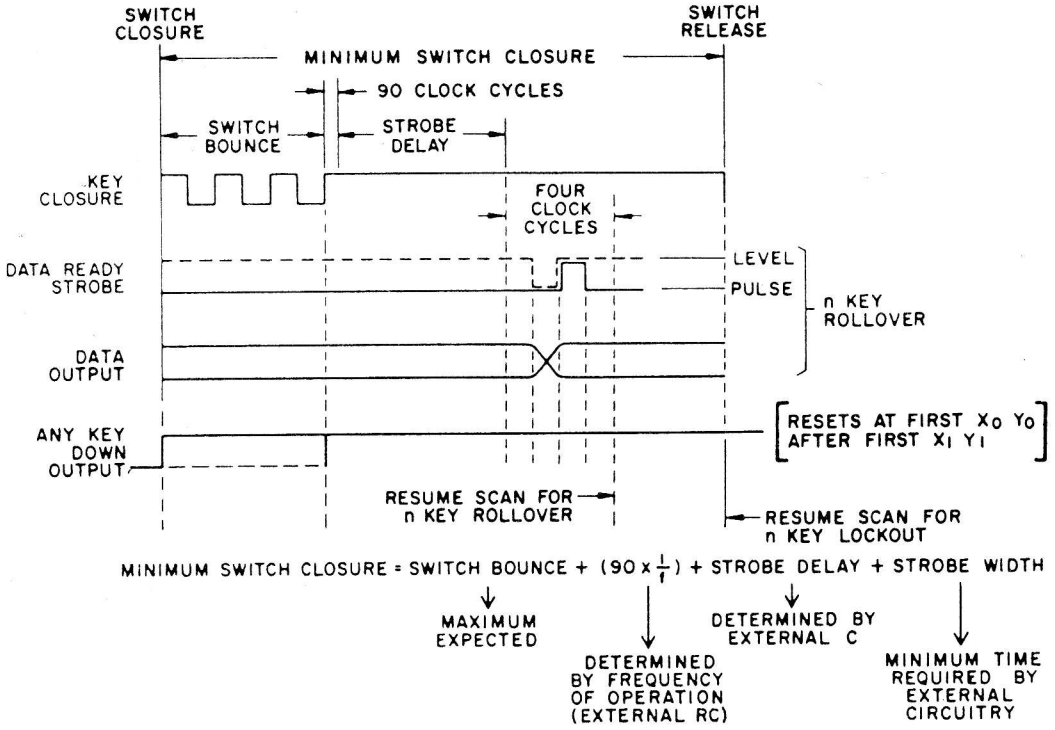
1) Die gezeigten Belegungen sind für die amerikanische Ausgabe des //e und dürften nicht 100%ig mit der deutschen Version übereinstimmen.

2) Die internationale Ausführung hat anstelle des Lötverbinders X1 den Umschalter für nationalen/internationalen Zeichensatz. A10 ist nicht mit ENVID' verbunden, vom Schalter aus führt die Leitung ALTCHR zum Video-ROM.

Die Frequenz des Tastaturencoder-Oszillators wird im Apple //e durch die Bauteile R32 und C70 auf rund 50 kHz festgelegt. Die Zeit zur Entprellung eines Schalters ist ebenfalls durch ein diskretes Bauteil bestimmbar und liegt nach den Unterlagen von General Instruments (ROM 3, AY-5-3600) durch den im //e verwendeten Wert bei rund 7 Millisekunden. Bei dieser "Entprellung" handelt es sich um eine Verzögerung zwischen einer vom Encoder festgestellten Verbindung zwischen zwei Matrixpunkten und der Erzeugung von AKD und KSTRB. Der Grund dafür liegt in der Tatsache, daß ein mechanischer Schalter immer "prellt", d.h. bei einem Schaltvorgang wippt die Schaltzunge ein paarmal hin- und her. Die Verzögerung verhindert, daß ein Tastendruck fälschlicherweise mehrfach interpretiert wird. Eigene Messungen ergaben eine Verzögerungszeit von rund 13 Millisekunden (theoretisch sollten es, wie gesagt, sieben sein) - man kann also von ungefähr 10 Millisekunden Entprellzeit ausgehen.

Bild 7.5 ist das vom Hersteller General Instruments herausgegebene Zeitdiagramm des AY-5-3600. Wie aus diesem Diagramm ersichtlich, gibt der Chip sein Signal "Data ready strobe" nur verzögert aus, während der Verzögerungszeit hat das Prellen eines Schaltkontakts keinen nachteiligen Einfluß auf die Funktion. Der "Strobe" ist ein Puls mit 20 Mikrosekunden Länge (eine Taktperiode des 3600), der für jeden Tastendruck einmal ausgegeben wird. Im Apple //e ist dieser Ausgang mit dem Eingang KSTRB der IOU verbunden, ein "Strobe" schaltet das Flag KEYSTROBE (Bild 7.1) und setzt den Zähler für die Autorepeat-Funktion zurück (Bild 3.8).

Bild 7.5 Zeitdiagramm des Tastaturencoders AY-5-3600



(mit freundlicher Genehmigung der Firma General Instruments)

Bild 7.5 zeigt, daß der Datenausgang des Encoders manchmal bereits im Taktzyklus vor dem "Strobe" gesetzt wird. Das ist reichlich irreführend - die Daten am Ausgang des 3600 sind grundsätzlich schon kurz nach Anfang des Taktzyklus gültig, auf den der "Strobe" folgt. Mit der im Apple //e verwendeten Taktfrequenz sind die Daten rund 20 Mikrosekunden gültig, bevor der "Strobe" aktiv wird. Innerhalb dieses Zeitraums hat der Tastatur-ROM (450 ns) wirklich genug Zeit zur Reaktion. Umso verwunderlicher ist es, daß Apple, Inc. keine Mühen gescheut hat, KSTRB um einige CPU-Zyklen innerhalb der IOU zu verzögern. Vielleicht hat sich der zuständige Designer von dem in Bild 7.5 gezeigten Zeitdiagramm ins Bockshorn jagen lassen und auf diese Weise sicherzustellen versucht, daß die ROM-Daten vor KEYSTROBE gültig sind.

Der Ausgang AKD des Encoders wird aktiv ("1"), sobald eine beliebige Taste der Matrix gedrückt wird. Er bleibt in diesem Zustand, solange die Taste gedrückt ist. Im Apple //e ist die entsprechende Leitung mit der IOU verbunden

und entsperrt dort die Autorepeat-Funktion (Bild 3.8), außerdem legt die IOU den Stand von AKD auf MD7 des Datenbusses, wenn der Prozessor die Adresse $C010 liest (Bild 7.1). Bitte beachten Sie, daß AKD unmittelbar nach einem Tastendruck aktiv wird, ohne die Entprellzeit abzuwarten. Damit existiert ein Zeitraum von rund 10 Millisekunden nach einem Tastendruck, in dem ein Programm AKD als aktiv lesen kann, während der ASCII-Wert der gedrückten Taste noch nicht gültig ist. Aus diesem Grund sollte ein Programm warten, bis KEYSTROBE gesetzt ist, bevor es Daten von der Tastatur liest.

Der Eingang lockout/rollover' des Encoders ist im Apple //e offen gelassen. Da dieser Eingang chip-intern mit einem "pull-down"-Widerstand auf "0" gezogen wird, ist deshalb die Funktion "rollover" aktiv: wenn bereits eine Taste gedrückt ist und der Benutzer eine weitere Taste drückt, dann wird der Code für die neu gedrückte Taste ausgegeben. Im Modus "lockout" werden neue Tastendrücke solange gesperrt, bis die zuerst gedrückte Taste losgelassen wird. Wenn Ihnen dieses Verhalten der Tastatur lieber ist, können Sie einfach Pin 4 des Encoders mit +5 Volt verbinden.

Die folgende Beschreibung der Operation "N-Key-Rollover" ist ein (übersetztes) Zitat aus dem Datenblatt des AY-5-3600: "Wenn eingeschlossener Kontakt entdeckt wird und die entsprechende Taste noch nicht kodiert wurde, wird das Entprell-Verzögerungsnetzwerk aktiviert. Wenn die Taste nach Ende der gewählten Verzögerungszeit immer noch gedrückt ist, wird der dazugehörige Code über den Datenausgang ausgegeben, das Signal "Data ready" erscheint, eine "1" wird im Speicher für die bereits kodierten Tasten an der entsprechenden Stelle geschrieben und die Absuche der Matrix fortgesetzt. Wenn ein geschlossener Schalter an einer anderen Stelle gefunden wird, wiederholt sich die beschriebene Folge. Falls die entsprechende Taste bereits kodiert war, erfolgt keine Reaktion. Der Code der zuletzt gedrückten Taste bleibt im Datenausgangspuffer."

Das vom Encoder ausgegebene Datenwort ist *gespeicherte Information, die nicht von der CPU verändert werden kann.* Wenn eine Taste gedrückt wird und die Entprellzeit abgelaufen ist, gibt der 3600 den entsprechenden Code aus. Dieser Code adressiert den Tastatur-ROM, dessen (ASCII-)Datenwort spätestens 450 ns danach von der CPU gelesen werden kann. Dieser Wert kann danach zu einem beliebigen Zeitpunkt und beliebig oft gelesen werden, er bleibt solange stabil, bis eine andere Taste gedrückt wird. Bei einem Druck auf eine neue Taste wird der alte Wert nach der Entprell-Verzögerung (+ rund 450 ns ROM-Zugriffszeit) durch den neuen ASCII-Code ersetzt.

Eine Zusatzkarte in Steckplatz 1 kann den Bereich $C000-$C01F durch eigene Adressen substituieren, indem sie ENKBD' ("ENable KeyBoarD" = Tastatur-Entsperrung) auf den Pegel "1" setzt. Damit ist es möglich, die eingebaute Tastatur des Apple //e durch ein via Steckplatz 1 verbundenes Zusatzgerät zu ersetzen. Die marktgängigen Zusatztastaturen für die Apple-Computer arbeiten allerdings samt und sonders nicht über eine Steckkarte, sondern sind für einen direkten Anschluß an den Steckverbinder des II(+) vorgesehen - die erst im //e eingebaute Möglichkeit der Schaltung von ENKBD' ist möglicherweise für einen Testvorgang während der Herstellung vorgesehen.

Die Ausgänge des AY-5-3600-PRO bilden das Datenwort B1/B4/B5/B6/B7/B8/B9-B2/B3, B2/B3 repräsentiert SHIFT und/oder CONTROL, die restlichen 6 Bit geben den Code für die gedrückte Taste wieder. B2/B3 werden auch erst zusammen mit einer gedrückten Taste gültig und können die folgenden Kombinationen bilden: 11 für "normal", 10 für CONTROL, 01 für SHIFT und 00 für SHIFT und CONTROL zusammen mit einer Taste. Diese Werte sind exakt das Komplement der Angaben im Datenblatt des 3600: hier wird von der Annahme ausgegangen, daß SHIFT oder CONTROL gedrückt wurde, wenn die entsprechende Leitung den Pegel "1" hat - im Apple //e setzt ein Druck auf eine dieser Tasten die dazugehörige Leitung auf "0", "1" ist der Normalzustand.[5]

Zum selben Zeitpunkt wie B2/B3 erscheinen auch die anderen sieben Bit des Datenworts am Ausgang des Encoders. Diese 7 Bit sind direkt aus der räumlichen Orientierung der Matrixpunkte abgeleitet: wenn z.B. die Verbindung X0-Y0 geschlossen wird, erzeugt das das Datenwort 0000000.

In Dezimalschreibweise kann man die Verbindung X0-Y0 mit dem Wert 00, die Verbindung X5-Y4 mit 54, die Verbindung X7-Y8 mit 87 usw. kennzeichnen und so für jeden Matrixknoten eine Kennzahl erhalten.

Der 3600 macht das, kodiert das Ergebnis allerdings binär: das Schließen der Matrixverbindung X5-Y4 erzeugt die Binärzahl 0110110 (dezimal 54).

---

5 Der 3600 verfügt über interne "pull-down"-Widerstände für diese Leitungen, deshalb werden sie im offenen Zustand mit "0" gelesen. Die Werte dieser Widerstände sind allerdings sehr hoch, die im //e angeschlossenen "pull-up"-Widerstände mit 1 kOhm ziehen die Leitungen SHIFT' und CTRL' deshalb auf einen Pegel, der sehr nahe an +5 Volt liegt, solange weder SHIFT noch CONTROL gedrückt wird.

Der einzig denkbare Grund, warum Apple, Inc. diese beiden Leitungen "verkehrt" herum angelegt hat, liegt in der RESET'-Leitung, die damit ohne zusätzliche Elektronik über CTRL geschaltet werden kann. Allerdings hätte es da ein einziger Transistor auch getan - (Anm. d. Übers.).

Die Ausgänge B1/B4/B5/B6/B7/B8/B9-B2/B3 sind mit den Adreßleitungen A8/A7/A6/A5/A4/A3/A2-A1/A0 des Tastatur-ROMs verbunden. Da mit B1/B4/B5/B6/B7/B8/B9 das binäre Äquivalent von XY erzeugt wird und für jede Taste vier Kombinationen möglich sind, entspricht die erzeugte ROM-Adresse dem Ergebnis von XY*4. Beispiel: Die Taste "J" liegt auf dem Knotenpunkt X2-Y6, die erzeugte ROM-Adresse ist 26 * 4 = 108 = $68. Der Tastatur-ROM enthält den ASCII-Code für SHIFT-CONTROL-J auf der Speicherstelle $68, den Code für CONTROL-J auf $69, den für SHIFT-J auf $6A und den für J alleine auf $6B.

Die beiden höchstwertigen Adressen des Tastatur-ROMs (A9 und A10) sind mit CAPLOCK' und ALTCHR (Rev A: AN2, amerikanische Rev. B: ENVID') verbunden. Der ROM enthält also vier komplette ASCII-Sätze: amerikanischen ASCII zusammen mit CAPS LOCK auf den Adressen $00-$$13F, denselben Code ohne CAPS LOCK auf $200-$33F, den deutschen ASCII-Satz mit CAPS LOCK auf $400-$53F und den deutschen ASCII-Satz ohne CAPS LOCK auf $600-$73F. Die Taste J aus dem vorigen Beispiel adressiert damit je nach Stand von CAPLOCK' und ALTCHR die Speicherstellen $068-$6B, $268-$26B, $468-$46B und $668-$66B innerhalb des Tastatur-ROMs.

Ein "normaler" Satz und sein CAPS LOCK-Gegenstück sind so gut wie identisch - lediglich die 29 Codes für das Alphabet und die Umlaute sind im zweiten Fall grundsätzlich Großbuchstaben. Auf diese Weise bewirkt CAPS LOCK eine Kapitalisierung aller Buchstaben, ohne dabei Ziffern, Sonderzeichen und die CONTROL-Funktionen zu beeinflussen.

Die internationalen Versionen des Apple //e haben als ersten ASCII-Satz den amerikanischen, der alternative Satz entspricht zusammen mit der Tastaturbelegung und den entsprechenden Matrizen im Video-ROM dem jeweiligen nationalen Standard. ENVID' hat keine Verbindung zu den Lötverbindern X1 oder X2 bzw. mit dem Tastatur-ROM, die Umschaltung erfolgt über ALTCHR und damit rein mechanisch über den Schalter an der Unterseite des Gehäuses, der auch gleichzeitig den Video-ROM mit umschaltet (s. Bild 8.5).

Tabelle 7.2 gibt den Inhalt des amerikanischen Tastatur-ROMs wieder, der zwischen Standard-ASCII und "Dvorak" umschaltet - letzteres ist eine veränderte Belegung der Tastatur bei gleichbleibendem Zeichensatz, die schnellere Eingaben erlaubt und in den USA langsam populär wird. Die Reihenfolge der Tabelleneinträge orientiert sich an der XY-Matrix der Tastatur, die bei sämtlichen Modellen des //e dieselbe ist. Das Programmieren eines eigenen Tastatur-ROMs sollte deshalb anhand dieser Tabelle eine einfache Angelegenheit sein.

Die ersten amerikanischen //e wurden noch mit einem anderen Tastatur-ROM (Nummer 341(2)-0132B) ausgeliefert, der sich von der endgültigen Version 0132C in einigen kleineren Details unterscheidet, die entsprechenden Stellen sind in der Tabelle mit B und C markiert. Die Lage von Anführungszeichen, Doppelpunkt und Fragezeichen entsprach im ROM 1032B nicht dem Original-Layout von Dvorak und nachdem Apple, Inc. der erste Computerhersteller ist, der dieses Layout unterstützt, wurde sehr schnell eine Anpassung an den vorgeschlagenen Standard vorgenommen. Die deutschen Ausführungen haben bereits von Anfang an die Typnummer 341-015X auf der Hauptplatine - nachdem hier beide ASCII-Sätze für "amerikanisch" und "national" benötigt werden und somit kein Platz für andere Layouts blieb, sind damit verbundene Probleme auch nicht aufgetreten.

Ein schärferer Blick auf Tabelle 7.2 enthüllt einige andere interessante Details wie z.B. die Tatsache, daß sämtliche ASCII-Codes von $00 bis $7F erzeugt werden können. CONTROL-2 und CONTROL-6 mit den Codes $00 und $1E dürften speziell deshalb eigens definiert worden sein. Was die Matrix-Tasten mit spezieller Funktion angeht, so sind nur DEL und ESC allein vertreten: alle anderen wie TAB, RETURN und die Pfeiltasten lassen sich auch über entsprechende Kombinationen mit CONTROL erzeugen.

Zusammenfassung der funktionellen Eigenschaften der Tastatur

Weitere zur Tastatur gehörige Schaltkreise sind in anderen Kapiteln und Abschnitten dieses Buchs beschrieben. Dazu gehören die Autorepeat-Funktion (Bild 3.8), die Erzeugung von KBD' (Bild 5.13b) sowie die in diesem Kapitel behandelten Softswitches bzw. Signale KEYSTROBE und AKD nebst der dazugehörigen Ausleseelektronik (s. Bild 7.1). Damit der Leser nicht das gesamte Buch nach Teilbeschreibungen funktionaler Aspekte der Tastatur durchsuchen muß, folgt an dieser Stelle eine Zusammenfassung:

1. Zwischen einem Tastendruck und der Verfügbarkeit des entsprechenden ASCII-Codes vergehen rund 10 Millisekunden.
2. Die Tastatur des //e ist mit einem "Rollover" ausgerüstet und liefert immer den Code der zuletzt gedrückten Taste - auch dann, wenn während dieses Tastendrucks andere Tasten gedrückt gehalten werden.

3. Wenn die MMU einen Lesezugriff auf den Bereich $C000-$C01F erkennt, aktiviert sie KBD' während PHASE0, der Tastatur-ROM legt den ASCII-Wert des letzten Tastendrucks auf MD0-MD6 des Datenbusses.
4. Das Signal AKD wird aktiv, sobald eine beliebige ("Matrix"-)Taste gedrückt wird. AKD ist vom Encoder zur IOU geführt, die dieses Signal um 2 bis 3 Prozessorzyklen verzögert und es danach bei einem Lesezugriff der CPU auf $C010 auf MD7 des Datenbusses legt. Da AKD innerhalb des Encoders nicht um die Entprellzeit verzögert wird, ergibt sich ein Zeitraum von rund 10 Millisekunden, in dem die CPU nach einem positiv ausgefallenen Test von AKD das ASCII-Datum des vorherigen Tastendrucks zu lesen bekommt.
5. Das Signal KSTRB wird für jeden Tastendruck rund 20 Mikrosekunden aktiv, zwischen dem Tastendruck und KSTRB liegt die Entprellzeit von rund 10 Millisekunden. KSTRB findet rund 20 Mikrosekunden nach dem Gültigwerden des ASCII-Wertes (am Ausgang des Tastatur-$ROMs) statt. Es ist vom Encoder zur IOU geführt und setzt dort KEYSTROBE nach 2 bis 3 Mikrosekunden Verzögerungszeit.
6. Wenn eine Taste für einen Zeitraum von 647 bis 971 Millisekunden gedrückt gehalten wird (die Verzögerungszeit ist abhängig vom Stand des "Flash"-Zählerbits F3), dann beginnt die IOU mit der internen Erzeugung von Autorepeat-Strobes mit einer Frequenz von rund 12 Hz. Diese Strobes setzen das Flag KEYSTROBE in derselben Weise wie ein erneuter Tastendruck des Benutzers. Die Verzögerungszeit zwischen Tastendruck und Beginn der Wiederholung sowie die Wiederholfrequenz sind von der vertikalen Durchlaufzeit eines Fernsehbildes abgeleitet (s. Bild 3.8).
7. Der Verzögerungszähler für die Autorepeat-Funktion wird sowohl von AKD als auch von KSTRB zurückgesetzt, die automatische Wiederholung einer Taste wird damit durch jeden anderen Tastendruck (außer durch SHIFT, CTRL, RESET oder die Apfeltasten) abgebrochen.
8. KEYSTROBE wird entweder durch KSTRB oder durch die Autorepeat-$Funktion gesetzt. Durch Einschalten der Stromversorgung, einen Schreibzugriff auf $C01X oder eine Lesezugriff auf $C010 wird dieses Flag zurückgesetzt.
9. Die programmierbaren Funktionen der Tastatur sind:

Adresse	**Funktion**
R$C00X	Lesen von KSTRB und ASCII
R$C010	Lesen von AKD und ASCII, setzt KEYSTROBE zurück
R$C011-$C01F	Lesen eines Softswitches der IOU oder der MMU zusammen mit ASCII
W$C01X	Zurücksetzen von KEYSTROBE

Die Verbindungen zu den peripheren Steckplätzen

Bei diesem Thema weiß man wirklich nicht, wo man anfangen soll, so vielschichtig ist es. Das Steckplatz-Konzept eröffnet mit seiner vollen Verbindung zum Adress- und Datenbus dieselben Möglichkeiten wie die Busplatinen modernster Computer. Es ist, als ob jemand einen wirklich schönen Computer entworfen hätte, dessen Schaltplan sieben leere Kästen aufweist, in denen "bitte einfüllen" oder ähnliches steht. So weiß man jedenfalls nie, was unter dem Deckel eines an sich harmlos aussehenden Apple alles lauert.

Die Möglichkeiten der Steckplätze sollten durch eine Einteilung der Signale in funktionelle Gruppen etwas offensichtlicher und klarer werden. Bild 7.6 zeigt eine derartige Einteilung, aus der z.B. mit einem einzigen Blick hervorgeht, daß die an den Steckplätzen verfügbaren Stromversorgungsleitungen dieselben sind wie auf dem Rest der Hauptplatine.

Neben der Stromversorgung sind die wichtigsten Leitungen der Adreßbus zusammen mit R/W' Control, der Datenbus und die Zeittakte. Überlegen wir einmal kurz, was über die (via Adreßbus) prozessorgesteuerte Kontrolle des Apple während PHASE0 bekannt ist: Sämtliche Kontrollen des I/O finden über den Adreßbus statt. Der Rückschluß ist ebenfalls korrekt: jede Aktion, die auf der Hauptplatine stattfindet, kann genauso auf einer Zusatzkarte stattfinden. Diese Feststellung sollte eine erste Ahnung davon vermitteln, welche Möglichkeiten in diesem Konzept stecken. (Manche Leute stecken allerdings Sachen in ihren Apple hinein, von denen Sie bis jetzt noch nicht einmal geträumt haben.)

Der periphere Datenbus ist mit der Hauptplatine über den Treiberbaustein B2 verbunden. Der primäre Zweck dieses Treibers liegt in der Stromverstärkung und in einer Trennung beider Busse. Aufgrund dieses Bausteins ändern sich die Verhältnisse auf der Hauptplatine bei einer Änderung der Steckplatzbelegung nicht (oder nur unwesentlich), und der periphere Datenbus verträgt ein erhebliches Mehr an Belastung.

Die MMU ist für die Richtungskontrolle des peripheren Datenbustreibers zuständig und übt sie über das Signal MD IN/OUT' so aus, daß das Ergebnis für den Prozessor völlig transparent ist. Für den Prozessor existiert dieser Baustein praktisch nicht: wenn Daten von den Steckplätzen oder vom seriellen Eingangsmultiplexer gelesen werden, dann setzt die MMU MD IN/OUT' während PHASE0 auf "1" und schaltet den Treiber damit auf die Richtung *IN* zum Datenbus bzw. Prozessor. In allen anderen Fällen bleibt MD IN/OUT' auf dem Pegel "0", d.h. in Richtung *OUT*.[6]

Der Eingang OE' des Treibers ist immer auf dem Pegel "0" - außer während PHASE1 in Schreibzyklen. Tatsächlich würde der Apple //egenauso funktionieren, wenn OE' einfach mit der Masse des Systems verbunden wäre. Die einzige Auswirkung dieser Abschaltung während PHASE1 besteht darin, daß Videodaten des RAMs der Hauptplatine nicht zusammen mit einem Schreibbefehl an den Steckplätzen anliegen, wenn PHASE0 am Ende von PHASE1 wieder steigt. Ich habe so den Verdacht, daß das aus Kompatibilitätsgründen zum II(+) geschehen ist, es kann aber auch sein, daß sich diese Schaltung als Notwendigkeit während der Testphase herausstellte und ich etwas übersehen habe.

Eine ganze Reihe von Taktsignalen steht an den peripheren Steckplätzen zur Verfügung - man kann nie genug davon haben, um zusätzliche Elektronik mit Vorgängen auf der Hauptplatine zu synchronisieren. PHASE1, Q3, 7M, COLOR REFERENCE (Steckplatz 7), 6502 SYNC und Video SYNC' (Steckplatz 7) sind vorhanden. Schmerzhaft vermißt werden 14M, RAS', CAS' und eine Prioritätskette für INHIBIT'. Diese Signale schreien förmlich danach, verbunden zu werden - umso erstaunlicher ist es, daß die Kontake 19 und 35 an den Steckplätzen von 2 bis 6 nicht verbunden sind.

Sämtliche Kontrolleingänge des 6502 sind mit den Steckplätzen verbunden. Die wichtigsten sind IRQ', NMI', RESET' und RDY. Sie sind als verbundene ODER-Schaltung verdrahtet, jede Leitung hat einen "pull-up"-Widerstand von 3300 Ohm. Damit kann jede Karte Interrupts erzeugen, RESET auslösen oder den 6502 via RDY anhalten. Die Leitung RESET' ist nicht nur mit den Steckplätzen und dem Prozessor, sondern auch mit der Tastatur und der IOU verbunden, die nicht nur auf RESET' reagiert, sondern auch selber einen RESET'-Impuls beim Einschalten der Stromversorgung erzeugt.

Weitere als verbundene ODER-Schaltung ausgeführte Kontrolleitungen sind DMA' und INHIBIT. DMA' ermöglicht einer beliebigen Zusatzkarte, den 6502 vom Daten- und Adreßbus der Hauptplatine zu trennen und die Kontrolle des Apple für schnelle I/O-Operationen oder andere Zwecke zu übernehmen. INHIBIT' sperrt die Speicherbausteine der Hauptplatine und ermöglicht den Verkehr der CPU mit Bausteinen in den Adreßbereichen $0000-$BFFF und $D000-$FFFF auf einer Zusatzkarte, der auf beliebige Art stattfinden kann. INHIBIT' erschöpft sich nicht mit diesem Bereich: auch der $C1-$DF-ROM wird komplett abgeschaltet, eine Zusatzkarte kann damit auch noch diesen Bereich benutzen. Um es genau zu nehmen: Via INHIBIT' und zusammen mit INTCXROM, SLOTC3ROM und ENKBD' von Steckplatz 1 ist die Übernahme des Adreßbereichs $0000-$C01F und $C090-$FFFF möglich.

Im //e werden für alle verbundenen ODER-Schaltungen "pull-up"-Widerstände mit 3300 Ohm verwendet (im II(+) hatten sie noch einen Wert von 1000 Ohm). Damit ist man den Spezifikationen für den 6502, in denen 3000 Ohm für RDY etc. vorgeschlagen werden, zwar wesentlich näher gekommen - das Resultat sind aber so große Schaltverzögerungen, daß es bei manchen Zusatzkarten notwendig sein kann, einen parallelen "pull-up"-Widerstand anzubringen, weil die entsprechende Leitung nicht mehr schnell genug auf den Pegel "1" zurückgeht.

Der Steckplatz 1 verfügt über zwei Verbindungen, die sonst nirgendwo vorhanden sind, nämlich ENKBD' und CLKEN'. Beide sind sie eigentlich wohl nur zu Testzwecken gedacht - erfindungsreiche Kartendesigner können aber sicher etwas damit anfangen. Der Tastatur-ROM reagiert nur dann auf Lesezugriffe im Bereich $C000-$C01F, wenn ENKBD' aktiv ist. Damit ist es möglich, dem Prozessor für diesen Bereich nach Setzen von ENKBD' auf "1" eine andere Datenquelle unterzuschieben, die MD0 bis MD6 des Datenbusses bei entsprechenden Abfragen setzt. Eine "1" auf der Leitung CLKEN' sperrt 14M auf der Hauptplatine und ermöglicht es einer Karte im speziellen Steckplatz, alle Zeitsignale selbst zu generieren. Sowohl CLKEN' als auch ENKBD' sind bei normalen Karten für den Steckplatz 1 nicht verbunden und werden mit "pull-down"-Widerständen auf dem Pegel "0" gehalten.

---

6 Eine komplette Beschreibung von MD IN/OUT' und der dazugehörigen Steuerung finden Sie in Kapitel 5.

Tabelle 7.2a Der Inhalt des Tastatur-ROMs

XY	SW #	ROM* ADDR	QWERTY	CAPS LOCK BO	CT	SH	AL	NO CAP LOCK BO	CT	SH	AL	DVORAK	CAPS LOCK BO	CT	SH	AL	NO CAP LOCK BO	CT	SH	AL
00	01	$000	ESCAPE	1B	1B	1B	1B	1B	1B	1B	1B	ESCAPE	1B	1B	1B	1B	1B	1B	1B	1B
01	02	$004	1 !	21	31	21	31	21	31	21	31	1 !	21	31	21	31	21	31	21	31
02	03	$008	2 @	00	00	40	32	00	00	40	32	2 @	00	00	40	32	00	00	40	32
03	04	$00C	3 #	23	33	23	33	23	33	23	33	3 #	23	33	23	33	23	33	23	33
04	05	$010	4 $	24	34	24	34	24	34	24	34	4 $	24	34	24	34	24	34	24	34
05	07	$014	6 ^	1E	1E	5E	36	1E	1E	5E	36	6 ^	1E	1E	5E	36	1E	1E	5E	36
06	06	$018	5 %	25	35	25	35	25	35	25	35	5 %	25	35	25	35	25	35	25	35
07	08	$01C	7 &	26	37	26	37	26	37	26	37	7 &	26	37	26	37	26	37	26	37
08	09	$020	8 *	2A	38	2A	38	2A	38	2A	38	8 *	2A	38	2A	38	2A	38	2A	38
09	10	$024	9 (	28	39	28	39	28	39	28	39	9 (	28	39	28	39	28	39	28	39
10	16	$028	TAB	09	09	09	09	09	09	09	09	TAB	09	09	09	09	09	09	09	09
11	17	$02C	Q	11	11	51	51	11	11	51	71	/ ? (B)	3F	2F	3F	2F	3F	2F	3F	2F
11	17	$02C	Q	11	11	51	51	11	11	51	71	' " (C)	22	27	22	27	22	27	22	27
12	18	$030	W	17	17	57	57	17	17	57	77	, <	3C	2C	3C	2C	3C	2C	3C	2C
13	19	$034	E	05	05	45	45	05	05	45	65	. >	3E	2E	3E	2E	3E	2E	3E	2E
14	20	$038	R	12	12	52	52	12	12	52	72	P	10	10	50	50	10	10	50	70
15	22	$03C	Y	19	19	59	59	19	19	59	79	F	06	06	46	46	06	06	46	66
16	21	$040	T	14	14	54	54	14	14	54	74	Y	19	19	59	59	19	19	59	79
17	23	$044	U	15	15	55	55	15	15	55	75	G	07	07	47	47	07	07	47	67
18	24	$048	I	09	09	49	49	09	09	49	69	C	03	03	43	43	03	03	43	63
19	25	$04C	O	0F	0F	4F	4F	0F	0F	4F	6F	R	12	12	52	52	12	12	52	72
20	31	$050	A	01	01	41	41	01	10	41	61	A	01	01	41	41	01	01	41	61
21	33	$054	D	04	04	44	44	04	04	44	64	E	05	05	45	45	05	05	45	65
22	32	$058	S	13	13	53	53	13	13	53	73	O	0F	0F	4F	4F	0F	0F	4F	6F
23	36	$05C	H	08	08	48	48	08	08	48	68	D	04	04	44	44	04	04	44	64
24	34	$060	F	06	06	46	46	06	06	46	66	U	15	15	55	55	15	15	55	75
25	35	$064	G	07	07	47	47	07	07	47	67	I	09	09	49	49	09	09	49	69
26	37	$068	J	0A	0A	4A	4A	0A	0A	4A	6A	H	08	08	48	48	08	08	48	68
27	38	$06C	K	0B	0B	4B	4B	0B	0B	4B	6B	T	14	14	54	54	14	14	54	74
28	40	$070	; :	3A	3B	3A	3B	3A	3B	3A	3B	S	13	13	53	53	13	13	53	73
29	39	$074	L	0C	0C	4C	4C	0C	0C	4C	6C	N	0E	0E	4E	4E	0E	0E	4E	6E
30	44	$078	Z	1A	1A	5A	5A	1A	1A	5A	7A	' " (B)	22	27	22	27	22	27	22	27
30	44	$078	Z	1A	1A	5A	5A	1A	1A	5A	7A	; : (C)	3A	3B	3A	3B	3A	3B	3A	3B
31	45	$07C	X	18	18	58	58	18	18	58	78	Q	11	11	51	51	11	11	51	71
32	46	$080	C	03	03	43	43	03	03	43	63	J	0A	0A	4A	4A	0A	0A	4A	6A
33	47	$084	V	16	16	56	56	16	16	56	76	K	0B	0B	4B	4B	0B	0B	4B	6B
34	48	$088	B	02	02	42	42	02	02	42	62	X	18	18	58	58	18	18	58	78
35	49	$08C	N	0E	0E	4E	4E	0E	0E	4E	6E	B	02	02	42	42	02	02	42	62
36	50	$090	M	0D	0D	4D	4D	0D	0D	4D	6D	M	0D	0D	4D	4D	0D	0D	4D	6D
37	51	$094	, <	3C	2C	3C	2C	3C	2C	3C	2C	W	17	17	57	57	17	17	57	77
38	52	$098	. >	3E	2E	3E	2E	3E	2E	3E	2E	V	16	16	56	56	16	16	56	76
39	53	$09C	/ ?	3F	2F	3F	2F	3F	2F	3F	2F	Z	1A	1A	5A	5A	1A	1A	5A	7A
40	KP	$0A0	/	2F	2F	2F	2F	2F	2F	2F	2F	/	2F	2F	2F	2F	2F	2F	2F	2F
41	KP	$0A4	left(B)	08	08	08	08	08	08	08	08	left(B)	08	08	08	08	08	08	08	08
41	KP	$0A4	down(C)	0A	0A	0A	0A	0A	0A	0A	0A	down(C)	0A	0A	0A	0A	0A	0A	0A	0A
42	KP	$0A8	0	30	30	30	30	30	30	30	30	0	30	30	30	30	30	30	30	30
43	KP	$0AC	1	31	31	31	31	31	31	31	31	1	31	31	31	31	31	31	31	31
44	KP	$0B0	2	32	32	32	32	32	32	32	32	2	32	32	32	32	32	32	32	32
45	KP	$0B4	3	33	33	33	33	33	33	33	33	3	33	33	33	33	33	33	33	33
46	29	$0B8	\ |	1C	1C	7C	5C	1C	1C	7C	5C	\ |	1C	1C	7C	5C	1C	1C	7C	5C

Legende:

AL - Taste alleine
CT - CONTROL
KP - zusätzliche Tastatur ("Keypad")
(C) = ROM 0132-C
SH - SHIFT
BO - SHIFT-CONTROL + Taste
(B) = ROM 0132-B

Hinweise:

* addieren Sie
$00n für CAPS LOCK, Standard
$20n für Normalbetrieb, Standard
$40n für CAPS LOCK, Dvorak-Layout
$60n für Normalbetrieb, Dvorak

n =
0 (BOth)
1 (ConTrol)
2 (SHift)
3 (ALone)

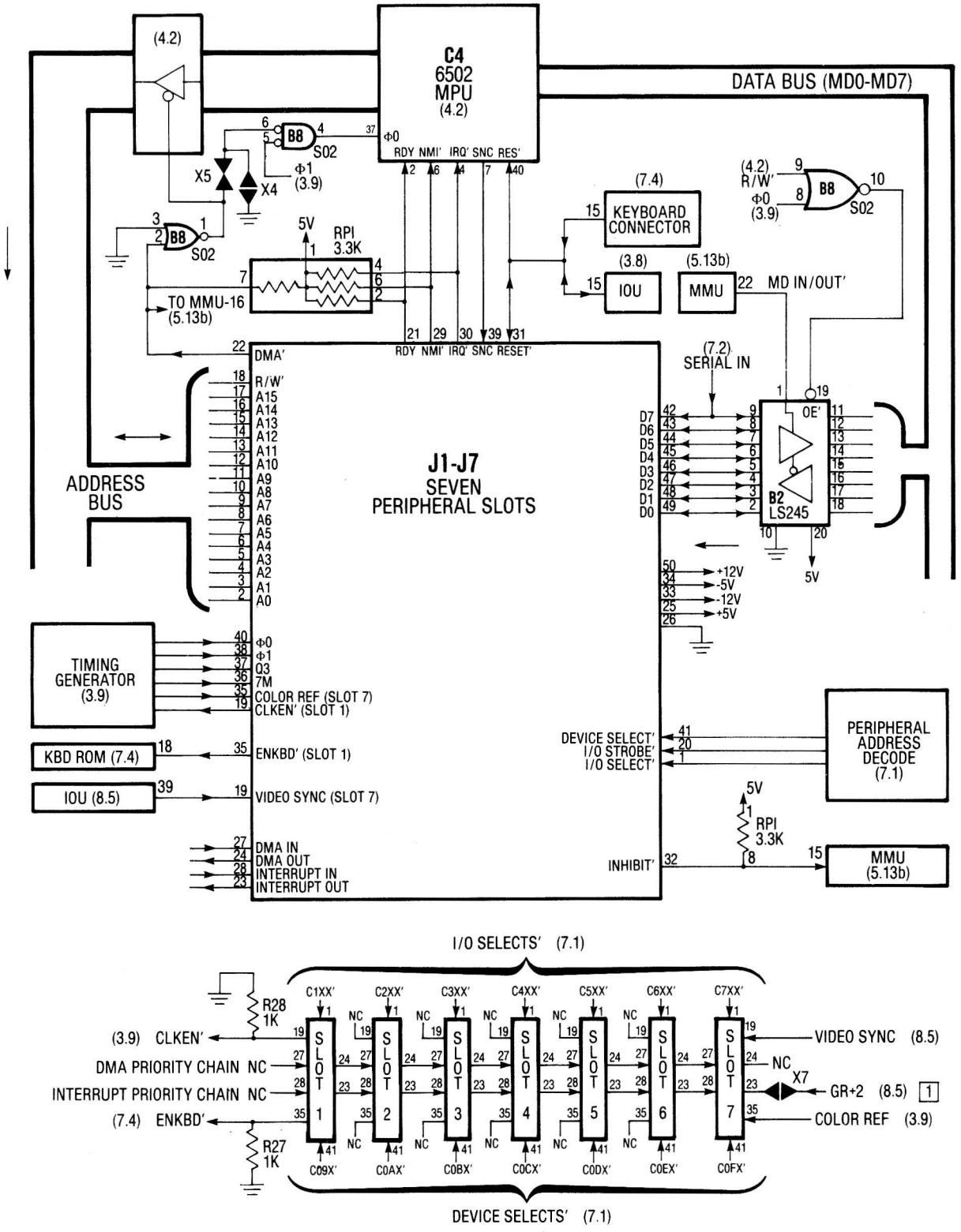
Tabelle 7.2b Der Inhalt des Tastatur-ROMs

XY	SW #	ROM* ADDR	QWERTY	CAPS LOCK BO CT SH AL	NO CAP LOCK BO CT SH AL	DVORAK	CAPS LOCK BO CT SH AL	NO CAP LOCK BO CT SH AL
47	13	$ØBC	= +	2B 3D 2B 3D	2B 3D 2B 3D	] }	1D 1D 7D 5D	1D 1D 7D 5D
48	11	$ØCØ	Ø)	29 3Ø 29 3Ø	29 3Ø 29 3Ø	Ø)	29 3Ø 29 3Ø	29 3Ø 29 3Ø
49	12	$ØC4	- _	1F 1F 5F 2D	1F 1F 5F 2D	[{	1B 1B 7B 5B	1B 1B 7B 5B
5Ø	KP	$ØC8	)	29 29 29 29	29 29 29 29	)	29 29 29 29	29 29 29 29
51	KP	$ØCC	ESC (B)	1B 1B 1B 1B	1B 1B 1B 1B	ESC (B)	1B 1B 1B 1B	1B 1B 1B 1B
51	KP	$ØCC	up (C)	ØB ØB ØB ØB	ØB ØB ØB ØB	up (C)	ØB ØB ØB ØB	ØB ØB ØB ØB
52	KP	$ØDØ	4	34 34 34 34	34 34 34 34	4	34 34 34 34	34 34 34 34
53	KP	$ØD4	5	35 35 35 35	35 35 35 35	5	35 35 35 35	35 35 35 35
54	KP	$ØD8	6	36 36 36 36	36 36 36 36	6	36 36 36 36	36 36 36 36
55	KP	$ØDC	7	37 37 37 37	37 37 37 37	7	37 37 37 37	37 37 37 37
56	56	$ØEØ	` ~	7E 6Ø 7E 6Ø	7E 6Ø 7E 6Ø	` ~	7E 6Ø 7E 6Ø	7E 6Ø 7E 6Ø
57	26	$ØE4	P	1Ø 1Ø 5Ø 5Ø	1Ø 1Ø 5Ø 7Ø	L	ØC ØC 4C 4C	ØC ØC 4C 6C
58	27	$ØE8	[{	1B 1B 7B 5B	1B 1B 7B 5B	; : (B)	3A 3B 3A 3B	3A 3B 3A 3B
58	27	$ØE8	[{	1B 1B 7B 5B	1B 1B 7B 5B	/ ? (C)	3F 2F 3F 2F	3F 2F 3F 2F
59	28	$ØEC	] }	1D 1D 7D 5D	1D 1D 7D 5D	= +	2B 3D 2B 3D	2B 3D 2B 3D
6Ø	KP	$ØFØ	*	2A 2A 2A 2A	2A 2A 2A 2A	*	2A 2A 2A 2A	2A 2A 2A 2A
61	KP	$ØF4	rght(B)	15 15 15 15	15 15 15 15	rght(B)	15 15 15 15	15 15 15 15
61	KP	$ØF4	left(C)	Ø8 Ø8 Ø8 Ø8	Ø8 Ø8 Ø8 Ø8	left(C)	Ø8 Ø8 Ø8 Ø8	Ø8 Ø8 Ø8 Ø8
62	KP	$ØF8	8	38 38 38 38	38 38 38 38	8	38 38 38 38	38 38 38 38
63	KP	$ØFC	9	39 39 39 39	39 39 39 39	9	39 39 39 39	39 39 39 39
64	KP	$1ØØ	.	2E 2E 2E 2E	2E 2E 2E 2E	.	2E 2E 2E 2E	2E 2E 2E 2E
65	KP	$1Ø4	+	2B 2B 2B 2B	2B 2B 2B 2B	+	2B 2B 2B 2B	2B 2B 2B 2B
66	42	$1Ø8	RETURN	ØD ØD ØD ØD	ØD ØD ØD ØD	RETURN	ØD ØD ØD ØD	ØD ØD ØD ØD
67	63	$1ØC	up	ØB ØB ØB ØB	ØB ØB ØB ØB	up	ØB ØB ØB ØB	ØB ØB ØB ØB
68	58	$111	SPACE	2Ø 2Ø 2Ø 2Ø	2Ø 2Ø 2Ø 2Ø	SPACE	2Ø 2Ø 2Ø 2Ø	2Ø 2Ø 2Ø 2Ø
69	41	$114	' "	22 27 22 27	22 27 22 27	- _	1F 1F 5F 2D	1F 1F 5F 2D
7Ø	KP	$118	? (B)	3F 3F 3F 3F	3F 3F 3F 3F	? (B)	3F 3F 3F 3F	3F 3F 3F 3F
7Ø	KP	$118	ESC (C)	1B 1B 1B 1B	1B 1B 1B 1B	ESC (C)	1B 1B 1B 1B	1B 1B 1B 1B
71	KP	$11C	SPCE(B)	2Ø 2Ø 2Ø 2Ø	2Ø 2Ø 2Ø 2Ø	SPCE(B)	2Ø 2Ø 2Ø 2Ø	2Ø 2Ø 2Ø 2Ø
71	KP	$11C	rght(C)	15 15 15 15	15 15 15 15	rght(C)	15 15 15 15	15 15 15 15
72	KP	$12Ø	(	28 28 28 28	28 28 28 28	(	28 28 28 28	28 28 28 28
73	KP	$124	-	2D 2D 2D 2D	2D 2D 2D 2D	-	2D 2D 2D 2D	2D 2D 2D 2D
74	KP	$128	RETURN	ØD ØD ØD ØD	ØD ØD ØD ØD	RETURN	ØD ØD ØD DØ	ØD ØD ØD ØD
75	KP	$12C	,	2C 2C 2C 2C	2C 2C 2C 2C	,	2C 2C 2C 2C	2C 2C 2C 2C
76	14	$13Ø	DELETE	7F 7F 7F 7F	7F 7F 7F 7F	DELETE	7F 7F 7F 7F	7F 7F 7F 7F
77	62	$134	down	ØA ØA ØA ØA	ØA ØA ØA ØA	down	ØA ØA ØA ØA	ØA ØA ØA ØA
78	6Ø	$138	left	Ø8 Ø8 Ø8 Ø8	Ø8 Ø8 Ø8 Ø8	left	Ø8 Ø8 Ø8 Ø8	Ø8 Ø8 Ø8 Ø8
79	61	$13C	right	15 15 15 15	15 15 15 15	right	15 15 15 15	15 15 15 15
8Ø	**	$14Ø	J	2Ø 2Ø 2Ø 4A	2Ø 2Ø 2Ø 4A	J	2Ø 2Ø 2Ø 4A	2Ø 2Ø 2Ø 4A
81	**	$144	O	2Ø 2Ø 4F 2Ø	2Ø 2Ø 4F 2Ø	O	2Ø 2Ø 4F 2Ø	2Ø 2Ø 4F 2Ø
82	**	$148	H	2Ø 48 2Ø 2Ø	2Ø 48 2Ø 2Ø	H	2Ø 48 2Ø 2Ø	2Ø 48 2Ø 2Ø
83	**	$14C	N	4E 2Ø 2Ø 2Ø	4E 2Ø 2Ø 2Ø	N	4E 2Ø 2Ø 2Ø	4E 2Ø 2Ø 2Ø
84	**	$15Ø		2Ø 2Ø 2Ø 2Ø	2Ø 2Ø 2Ø 2Ø		2Ø 2Ø 2Ø 2Ø	2Ø 2Ø 2Ø 2Ø
85	**	$154		2Ø 2Ø 2Ø 2Ø	2Ø 2Ø 2Ø 2Ø		2Ø 2Ø 2Ø 2Ø	2Ø 2Ø 2Ø 2Ø
86	**	$158	M	2Ø 2Ø 2Ø 4D	2Ø 2Ø 2Ø 4D	M	2Ø 2Ø 2Ø 4D	2Ø 2Ø 2Ø 4D
87	**	$15C	A	2Ø 2Ø 41 2Ø	2Ø 2Ø 41 2Ø	A	2Ø 2Ø 41 2Ø	2Ø 2Ø 41 2Ø
88	**	$16Ø	C	2Ø 43 2Ø 2Ø	2Ø 43 2Ø 2Ø	C	2Ø 43 2Ø 2Ø	2Ø 43 2Ø 2Ø
89	**	$164	D	44 2Ø 2Ø 2Ø	44 2Ø 2Ø 2Ø	D	44 2Ø 2Ø 2Ø	44 2Ø 2Ø 2Ø

** XY = 80-89 werden im Apple //e nicht benutzt, X8 des Encoders hat keine Verbindung. In beiden Versionen des amerikanischen Tastatur-ROMs steht hier "JOHN MACD", das Kürzel von John MacDougall.

*** Die Adressen $168-$1FF, $369-$3FF, $568-$5FF und $768-$7FF sind unbenutzt, weil der Encoder insgesamt 90 (und nicht 128) Matrixknoten verarbeiten kann. In beiden Versionen der amerikanischen Tastatur-ROMs sind diese Bereiche mit $A0 aufgefüllt, außer dem obersten ($768-$7FF), der den Text "314-0132B COPYRIGHT APPLE COMPUTER 1982" oder einen ähnlichen Text enthält.

Bild 7.6 Schaltplan der Verbindungen der peripheren Steckplätze



Anmerkungen:

(1) In Rev. A ist Kontakt 23 von Steckplatz 7 nicht verbunden, X7 existiert erst seit Rev. B.

Die Leitung USER1' des Apple II(+) ist im //e nicht mehr vorhanden. Über diese Leitung konnte eine Zusatzkarte im II(+) den gesamten Bereich $C000-$CFFF für sich beanspruchen. Im //e ist der Bereich $C020-$C08F (und damit DEVICE SELECT') nicht abschaltbar, für den Bereich $C100-$CFFF müssen die Softswitches INTCXROM, SLOTC3ROM und INTC8ROM entsprechend gesetzt sein.

Der Kontakt 39, der im II(+) für USER1' benutzt wurde, führt im //e das Signal 6502 SYNC und ermöglicht auf diese Weise hardwaregestützte Einzelschrittbearbeitung und/oder die Identifikation einzelner Opcodes auf dem Datenbus - eine entsprechende Zusatzkarte natürlich vorausgesetzt. Es ist mit diesem Signal möglich, Zusatzkarten zu entwerfen, die direkt auf Opcodes des 6502 reagieren. Beispiel: Der Befehl RTI kann von einer Karte daran erkannt werden, daß SYNC den Pegel "1" hat und $40 auf dem Datenbus anliegt, während PHASE0 fällt.

Die mehrfach vorhandenen Auswahlsignale DEVICE SELECT' und I/O SELECT' sowie das Signal I/O STROBE sind Reaktionen der MMU auf vom Prozessor angesprochene Adreßbereiche, die den Steckplätzen zugeordnet sind. Die Zuordnung "Steckplatz - Adreßbereich(e)" hätte auch als reine Vereinbarung stattfinden können, eine in "Steckplatz 1" sitzende Karte könnte genausogut durch eigene Elektronik anstelle des entsprechenden DEVICE SELECT'-Signals den Adreßbereich $C09X erkennen. Wie durchdacht das von Apple verwendete Konzept ist, wird einem erst klar, wenn man sich überlegt, was man mit einer derartigen Karte anfangen kann, wenn Steckplatz 1 bereits belegt ist - für jede Karte würde mindestens ein "Mäuseklavier" benötigt, damit sie für einen Steckplatz bestimmter Nummer eingestellt werden könnte. Abgesehen davon: Solange DEVICE SELECT' auf der Hauptplatine dekodiert und erzeugt wird, kommt wenigstens niemand auf die Idee, sich mit einer neuen Karte im SELECT-Bereich anderer Karten breitzumachen. Bei der Unzahl von teilweise wirklich obskuren Quellen für Apple-Zusatzkarten ist das vielleicht wichtiger als man denken mag. Schließlich: Durch die auf der Hauptplatine konzentrierte Dekodierung aller Karten-Adreßbereiche erspart man sich eine ganze Menge sonst doppelt und dreifach (d.h. einmal pro Karte) vorhandener Chips.

Das Auswahlsignal DEVICE SELECT' jedes Steckplatzes kennzeichnet einen Bereich von 16 aufeinanderfolgenden Adressen, der spezifisch für jeweils einen Steckplatz vorgesehen ist. Dieser Bereich wird normalerweise für den Datenverkehr des Prozessors mit einer Karte oder die Übergabe bestimmter Befehle benutzt. Ein Karten-Design kann nur eine einzige Datenadresse enthalten wie z.B. ein Sprachsynthesizer, der ein Phonem von sich gibt, wenn er ein Datenwort erhält. Diese Art von Karten kann DEVICE SELECT' ohne weitere Dekodierung benutzen. RAM-Zusatzkarten dagegen benutzen diesen Bereich meist für Bankumschaltungen. Hier kann DEVICE SELECT' die momentan gesetzte Konfiguration entsperren, das übergebene Datenwort wird aus den unteren Bits der Adresse und R/W' herausdekodiert. Eine Karte kann allein durch Dekodierung der Adreßbits A0-A3 und R/W' zwischen 32 unterschiedlichen Befehlen unterscheiden; falls auch noch der Inhalt des Datenbusses mit einbezogen wird, ergeben sich immerhin 8192 Möglichkeiten.

Das Auswahlsignal DEVICE SELECT' kennzeichnet einen Bereich von 256 Byte, der spezifisch für jeweils einen Steckplatz ist. Dieser Bereich wird normalerweise von einem ROM oder einem PROM belegt, die ersten Adressen ($Cn00-) müssen von einem Programm für den 6502 belegt werden, wenn die Karte über IN# und/oder PR# angesteuert werden können soll.[7]

Das Signal I/O STROBE' wird bei einem Zugriff auf den Bereich $C800-$CFFF aktiv, solange INTC8ROM nicht gesetzt ist, und für alle sieben Steckplätze zu den gleichen Bedingungen. Im Gegensatz zu den durch die SELECT's festgelegten I/O-Bereiche kann im Bereich von I/O STROBE' immer nur eine einzige Karte auf einmal aktiv sein. Diesem Konzept liegt die Idee zugrunde, daß in den meisten Fällen immer nur eine Karte auf einmal aktiv sein muß, die ein großes Steuerprogramm (> 256 Byte) benötigt. Wie im nächsten Abschnitt beschrieben, müssen alle Karten, die diesen Bereich benutzen, einem genau festgelegten "Abschaltprotokoll" folgen, damit verhindert wird, daß mehrere ROMs gleichzeitig auf eine vom Prozessor ausgegebene Adresse reagieren. Über dieses Konzept wird es möglich, in einem einzigen Apple solche Zusatzgeräte wie Druckerschnittstellen mit eingebautem Grafikausdruck, "intelligente" 80-Zeichen-Karten und EPROM-Programmierer mit entsprechenden Treiberprogrammen in den verschiedenen Steckplätzen zu haben, ohne dabei Probleme mit dem beschränkten Adressraum zu bekommen.

Nicht vergessen werden darf allerdings, daß jedes Programm durch entsprechendes Setzen der Softswitches INTCXROM, SLOTC3ROM und INTC8ROM die Erzeugung von I/O STROBE' und den I/O SELECT's komplett abschalten kann. Programme, die innerhalb des Bereichs $C100-$CFFF ungestört auf den ROM der Hauptplatine zugreifen wollen, werden das im allgemeinen während der Initialisierungsphase tun. Auch das Monitorprogramm

---

7 Ein PR# oder IN#-Befehl bewirkt normalerweise einen Sprung des Prozessors "auf" die entsprechende Steckkarte, PR#1 bewirkt einen Sprung nach $C100, PR#2 nach $C200 etc. Die Steckplatznummern 0 und 3 werden in der Firmware des Apple //e gesondert behandelt: PR#0 und IN#0 bewirken die Rückleitung der Ein- bzw. Ausgabe auf die 40-Zeichen-Routinen (und nicht einen Sprung zum "Steckplatz 0"). PR#3 und IN#3 bewirken die Aktivierung der 80-Zeichen-Routinen, die zwar ebenfalls mit der Adresse $C300 beginnen, aber im ROM der Hauptplatine gespeichert sind.

nimmt bei verschiedenen Gelegenheiten eine Abschaltung vor - wobei hier allerdings das Konzept "Steckplätze aus, Aufruf einer CXXX-Routine des Hauptplatinen-ROMs, Steckplätze wieder an" dahintersteht. Ein letzter Grund zur Abschaltung von I/O STROBE und SELECT's wäre eine Karte, die via INHIBIT' den Bereich $C100-$CFFF belegt.

Die letzten noch nicht erwähnten Signale der peripheren Steckplätze sind DMA IN, DMA OUT, INT IN und INT OUT. Diese vier Signale bilden die DMA- bzw. die Interrupt-Prioritätskette, die Sie sehr ausführlich in Kapitel 4 beschrieben finden. Beide Ketten verhindern, daß ein und dieselbe Operation von mehr als einer Karte zum selben Zeitpunkt ausgeführt bzw. angefordert wird. Die Steckplätze mit der niedrigeren Nummer haben jeweils die höhere Priorität.

Eine weitere Prioritätskette, die nicht existiert, aber unbedingt notwendig wäre, ist die für INHIBIT'. Es ist nicht so, daß Apple, Inc. das nicht gemerkt hätte - als die erste Firmware-Karte für den Apple II entworfen wurde, war es allerdings zu spät und man ist zwangsläufig auf die DMA-Kette ausgewichen. Mit dieser Zweckentfremdung ist es überhaupt möglich, zwei via INHIBIT' arbeitende Karten in einem Computer zu haben. Die Voraussetzung dafür ist, daß alle Karten in nebeneinanderliegenden Steckplätzen installiert sind. Bei einem Versuch, mehrere Karten auf einmal zu aktivieren, reagiert automatisch die Karte im Steckplatz mit der höchsten Priorität.

Das Ausweichmanöver der Apple-Designer illustriert auch noch einen anderen Aspekt der beiden existierenden Prioritätsketten: ihre Zuordnung zu DMA und Interrupt ist reine Konvention; elektronisch gesehen lassen sich die Verbindungen für Prioritätsketten mit beliebigem Zweck nutzen.

DMA IN und INT IN sind für den Steckplatz 1 nicht notwendig - schließlich hat eine hier installierte Karte die höchste Priorität. Aus dem entgegengesetzten Grund sind die Anschlüsse DMA OUT und INT OUT an Steckplatz 7 nicht notwendig - es gibt nichts, was damit abgeschaltet werden könnte. Das Ergebnis sind ein paar weiße Flecken auf der Landkarte des Apple //e: die Kontakte 27 und 28 von Steckplatz 1 und der Kontakt 24 von Steckplatz 7 sind frei und stehen für weitere Signale zur Verfügung - falls Apple, Inc. sie in zukünftigen Generationen verwenden will. Kontakt 23 von Steckplatz 7 kann über den Lötverbinder X7 mit dem Signal GR+2 von Pin 2 der IOU verbunden werden. Wenn Sie eine Karte für Steckplatz 7 besitzen (Light Pen o.ä.), die mit dem Signal GRAPHICS arbeitet, enthält die dazugehörige Installationsanleitung wahrscheinlich die Anweisung, X7 zu verbinden.

Das I/O STROBE'-Protokoll

Dieses Signal ist zu Kontakt 20 aller peripheren Steckplätze geführt. Seine Funktion ist die Aktivierung von ROM im Bereich $C800-$CFFF auf der Karte, die zum entsprechenden Zeitpunkt aktiv ist. Auf der Hauptplatine existiert keine Möglichkeit, Zusatzkarten mitzuteilen, wann sie auf I/O STROBE' reagieren müssen und wann nicht. Statt dessen existiert die folgende, von Apple, Inc. festgelegte Vereinbarung, der alle Karten mit ROM für den Bereich $C800-$CFFF folgen müssen:

1. Wenn Kontakt 1 (I/O SELECT') einer Karte aktiv wird, kann diese Karte im folgenden auf I/O STROBE' reagieren.
2. Wenn der Adreßbus während PHASE0 den Wert $CFFF hat, müssen alle Zusatzkarten eine eventuell gesetzte Reaktion auf I/O STROBE' abschalten.

Ein Beispiel dazu: Nehmen wir an, wir haben zwei Karten in den Steckplätzen 1 und 2, die beide ROM für den Bereich $C800-$CFFF besitzen. Von BASIC aus wird der Befehl PR#1 ausgeführt, als Resultat dieses Befehls folgt ein Sprung des 6502 zur Adresse $C100. Die Karte in Steckplatz 1 reagiert auf $C1XX-Adressen (d.h. auf I/O SELECT') mit der Aktivierung eines ROMs und legt so Befehle für den 6502 auf den Datenbus. Nehmen wir einmal an, das Programm beginne mit:

```
C100:BIT $CFFF

C103:JMP $C800
```

Im letzten Prozessorzyklus des ersten Befehls enthält der Adreßbus den Wert $CFFF während PHASE0. Alle Zusatzkarten stellen sich daraufhin auf folgende I/O STROBE'-Signale "taub" - inklusive der Karte in Steckplatz 1. Mit dem nächsten Zyklus gibt der 6502 allerdings die Adresse $C103 zum Lesen des nächsten Befehls aus und löst damit einen weiteren I/O SELECT' für Steckplatz 1 aus. Die Karte reagiert darauf mit einer internen Umschaltung und wird im folgenden auf I/O SELECT' reagieren - solange, bis wieder die Adresse $CFFF angesprochen wird. Für ein Programm in Steckplatz 2 funktioniert der Ablauf auf dieselbe Weise.

Es ist möglich, die Programme für die Bereiche $CnXX und $C800-$CFFF in einem einzigen ROM mit 2 kByte zu speichern. Der ROM muß dabei so geschaltet sein, daß er sowohl durch I/O SELECT' als auch durch I/O STROBE' aktivierbar ist. In diesem Fall liest der Prozessor einen 256-Byte-Bereich des ROMs doppelt: die Daten, die er über die $CnXX-Adresse erhält, erscheinen auf $C9XX ein zweites Mal, wobei auf die 2048 Byte im Bereich $C800-$CFFF nur zugegriffen werden darf, nachdem die Reaktion der Karte auf I/O STROBE' durch Ansprechen einer $CnXX-Adresse sichergestellt worden ist.

Aus Kapitel 5 ist bereits bekannt, daß über INTC8ROM die Erzeugung von I/O STROBE' abgeschaltet und statt dessen der ROM der Hauptplatine für Adressen im Bereich $C800-$CFFF aktiviert werden kann. INTC8ROM wird durch einen Zugriff auf $C3XX gesetzt, wenn SLOTC3ROM zurückgesetzt ist, und wird durch einen Zugriff auf $CFFF zurückgesetzt. Der Apple //e emuliert eine 80-Zeichen-Karte in Steckplatz 3 auch in diesem Punkt fast vollständig - mit einer kleinen Ausnahme: Die Firmware der Hauptplatine hat immer Vorrang vor den ROMs eventueller Zusatzkarten. Nach dem Setzen von INTC8ROM ist das Ansprechen von $CFFF nicht mehr nötig, um "echte" Zusatzkarten von einer Reaktion auf I/O STROBE' abzuhalten - schließlich wird I/O STROBE' durch INTC8ROM komplett abgeschaltet.

Diese Schaltmöglichkeit von I/O STROBE' darf nicht mit der Substitution des Bereichs $C100-$CFFF via INHIBIT' verwechselt werden: I/O STROBE' ordnet den Bereich $C800-$CFFF einer Karte zu, wenn INTC8ROM und INTCXROM zurückgesetzt sind, über INHIBIT' kann eine Zusatzkarte sämtliche Reaktionen der Hauptplatine auf die Bereiche $0000-$BFFF und $C100-$CFFF unterdrücken und durch eigene Speicherplätze ersetzen.

Die Struktur der Ein-/Ausgabe: KSW und CSW

Die Möglichkeiten und Grenzen der peripheren Steckplätze werden letztendlich durch die verfügbaren Signale gesetzt - in der Wahrnehmung des Programmierers sind ihre Charakteristika allerdings durch das verwendete Betriebssystem wesentlich stärker bestimmt. Die Eckbedingungen für die Ein-/Ausgabe des Apple wurden durch den (ganz) alten Monitor-ROM gesetzt, d.h. durch die Firmware im Bereich $F800-$FFFF im alten Apple II. Der wichtigste Punkt hier ist die Tatsache, daß jeder I/O *vektorisiert* stattfindet: die Speicherstellen $36 und $37 enthalten immer die Startadresse der (primären) Ausgaberoutine und werden mit **CSW** ("Character output SWitch" = Zeichenausgabeschalter) bezeichnet, die Speicherstellen $38 und $39 enthalten die Startadresse der (primären) Zeicheneingaberoutine und haben den Namen **KSW** ("Keyboard input SWitch" = Tastatureingabeschalter).

I/O und der Monitor des Apple

Das Monitorprogramm des Apple hat sich über die Jahre in derselben Weise weiterentwickelt wie der Apple II zum //e - der weitaus größte Teil ist aber nach wie vor identisch mit dem Programm, das von einem Hobbyisten Anfang der siebziger Jahre geschrieben wurde. Spezifischer: Auch im //e haben wir nach wie vor ein System, bei dem auf jede Eingabe über die Tastatur sofort eine Ausgabe auf den Bildschirm folgt, Ein- und Ausgaberoutine werden durch KSW bzw. CSW bestimmt. Um etwas weiter in diese Systematik einzusteigen, verfolgen wir einmal den Ablauf nach Einschalten der Stromversorgung und nehmen dabei an, daß kein Diskettencontroller eingesteckt ist.

Der automatisch ausgelöste RESET-Impuls und die dadurch aufgerufene RESET-Routine setzen KSW auf eine Routine des Monitors mit dem Namen KEYIN, die einen blinkenden Cursor ausgibt und auf die Eingabe eines Zeichens wartet; CSW wird auf die Routine COUT1 gesetzt, die den Inhalt des Akkus in den TEXT-Bildspeicher schreibt und die Cursorposition danach um eine Position nach rechts versetzt. Beide Routinen arbeiten im 40-Zeichen-Modus. Nach dem Setzen von CSW und KSW werden die Steckplätze nach einem Diskettencontroller abgesucht; bleibt die Suche erfolglos, wird Applesoft BASIC gestartet. Dieses Programm unterhält sich (wie auch Integer BASIC, die Kommandoebene des Monitors und viele andere) mit dem Benutzer über die Routine GETLN ("GET LiNe" = Lies Zeile).

GETLN liest solange Zeichen vom primären Eingabegerät, bis ein RETURN eingegeben wird. Für jedes einzelne Zeichen wird dazu die "momentan aktive" Eingaberoutine aufgerufen, d.h. ein indirekter Sprung zu der in KSW gespeicherten Adresse ausgeführt. Gelesene Zeichen werden im Eingabepuffer ($200-$2FF) gespeichert und zur Ausgaberoutine gesendet, d.h. auf die Speicherung folgt ein indirekter Sprung zu der in CSW gespeicherten Adresse. Wenn das eingegebene (und jetzt auf dem Bildschirm ausgedruckte) Zeichen kein RETURN war, wird danach eine weitere Zeicheneingabe angefordert usw. GETLN kehrt mit einem gefüllten Eingabepuffer zu dem Programm zurück, vom dem diese Routine aufgerufen wurde. Das Programm kann danach mit einer Auswertung der eingegebe-

nen Zeichen beginnen. Die Art und Weise, wie sich GETLN verhält, bestimmt einen großen Teil des "äußeren Eindrucks" des Apple.

Die Umleitung des I/O zu anderen Geräten

KSW und CSW sind I/O-Vektoren. Im Prinzip kann man jedem Programm, das seine Ein-/Ausgabe über KSW/CSW abwickelt (und das ist meistens der Fall) auf ein beliebiges Ein-/Ausgabegerät mit dazugehörigem Treiberprogramm umleiten, indem man KSW und/oder CSW entsprechend setzt. Der Betrieb eines an sich sehr komplexen Programms wie Applesoft über eine mit den Announciator-Ausgängen aufgebaute serielle Schnittstelle ist ohne weiteres möglich, wenn Sie ein Treiberprogramm schreiben, es irgendwo im RAM ablegen und danach KSW und CSW so verändern, daß sie auf dieses Treiberprogramm zeigen. Von diesem Zeitpunkt an wird Applesoft sämtliche Ein- und Ausgaben über Ihr Programm abwickeln (Ausnahmen: Grafikausgabe und Test auf Control-C).

Jede beliebige Zusatzkarte kann über die BASIC-Befehle PR#n und IN#n oder n CONTROL-P bzw. n CONTROL-K vom Monitor aus als primäres Ein- und/oder Ausgabegerät gesetzt werden. Bei der Ausführung eines Befehls wie PR#1 passiert folgendes:

1. Der Monitor setzt die Speicherstellen $36/36 (CSW) auf den Wert 00/$C1, also auf die Adresse $C100. Damit ist der Befehl PR# beendet.

Moment mal, werden Sie sagen - und jetzt? Ganz einfach: Der zweite Schritt wird erst dann ausgeführt, wenn das nächste Zeichen tatsächlich ausgegeben werden soll. Wenn Sie den Befehl PR#1 im direkten Modus geben, dann reagiert Applesoft danach mit einem erneuten Prompt (Ü bzw.]) - da haben wir bereits die nächste Ausgabe.

2. Der Aufruf der Routine "Ausgabe eines Zeichens" (d.h. COUT) führt einen indirekten Sprung zu dem Programm aus, dessen Startadresse in CSW steht. Wenn wir bei unserem Beispiel bleiben, dann ist das C100. Hier muß nun ein Programm stehen, das die folgenden Schritte ausführt:
   a. Initialisierung der angeschlossenen Hardware, soweit nötig.
   b. *Veränderung* des KSW-Vektors für zukünftige Zeichenausgaben: den nächsten Ausgaben muß keine Initialisierung mehr vorausgehen. Im Normalfall wird KSW von $Cn00 auf $Cn03 oder $Cn07 gesetzt - hier steht dann ein direkter Sprung zur Ausgaberoutine der Karte.
   c. Ausgabe des Zeichens über die mittlerweile initialisierte Hardware.
   d. Rücksprung zu dem Programm, das die Zeichenausgabe angefordert hat.

Dieser Ablauf offenbart eine interessante Kleinigkeit: Ein in Steckplatz 6 befindlicher Diskettenkontroller wird *nicht* durch den Befehl PR#6 gestartet, sondern erst durch die darauffolgende Ausgabe. Ein Programm wie das folgende ist also durchaus lauffähig:

```
10 PRINT "HALLO.."
20 PR# 6
30 FOR T = 1 TO 100 : NEXT T
40 PR# 0
50 PRINT "HÄHÄ"
```

Zusammen mit einem Betriebssystem wie DOS 3.3 oder ProDOS ist der Ablauf etwas komplizierter: beide setzen KSW und CSW so, daß bei jeder Ein- oder Ausgabe zuerst ein Sprung in das DOS hinein erfolgt (DOS 3.3: CSW $9FBD, KSW $9E81; ProDOS: $B84B und $B84E für die BASIC.SYSTEM-Versionen bis 1.1). An diesen Stellen steht erst einmal eine Routine, die eine Prüfung auf "DOS-Befehl" vornimmt. Erst wenn erkannt wurde, daß kein DOS-Befehl vorliegt, wird zur tatsächlichen Ein- bzw. Ausgaberoutine gesprungen. Hier liegt der Grund verborgen, warum DOS-Befehle mit einem bestimmten Zeichen (Control-D) eingeleitet und via PRINT ausgegeben werden müssen - an Control-D wird ein DOS-Befehl erkannt.

Ein Befehl wie PR#1 und die dazugehörige Zeichenausgabe umfaßt damit einige weitere Schritte, wenn zusätzlich ein DOS geladen wurde:

1. Die im DOS (und nicht auf CSW/KSW!) gespeicherten Startadressen der Ein-/Ausgaberoutine werden durch den PR#- (oder IN#-)Befehl auf $Cn00 gesetzt. Rücksprung zum aufrufenden Programm, CSW und KSW zeigen nach wie vor auf DOS-Adressen.
2. Ausgabe eines Zeichens: Wenn der Test auf "DOS-Befehl" negativ ausfällt, wird das Zeichen auf die folgende Weise weitergegeben:

a. Einsetzen der "richtigen" Startadressen in CSW/KSW und indirekter Sprung über diese Vektoren zur entsprechenden Karte.

b. Nach dem Rücksprung von der Karte (Initialisierung hat stattgefunden, KSW/CSW wurden dabei eventuell verändert): Speicherung der jetzigen Werte von CSW/KSW auf interne Adressen des DOS, Einsetzen der DOS-Adressen in CSW/KSW und Rücksprung zum Programm, das die Ein-/Ausgabe angefordert hat.

PR#- und IN#-Befehle müssen entweder im direkten Modus oder von einem laufenden Programm mit Control-D als DOS-Befehl gegeben werden, ansonsten werden sie entweder ignoriert oder führen zur kompletten Abtrennung des DOS. Ein Befehl wie

```
10 PR# 1
```

ergibt Probleme mit einem DOS: Wenn direkt auf diesen Befehl eine Ausgabe folgt, dann ist CSW entsprechend gesetzt, falls eine Eingabe folgt, dann wird über KSW zu DOS gesprungen - und DOS setzt erst einmal KSW und CSW mit den intern gespeicherten Adressen. Eine Befehlsfolge wie

```
10 PR# 0: IN# 0
```

trennt jedes DOS komplett ab - hier werden sowohl CSW als auch KSW neu gesetzt, im weiteren Verlauf des Programms hat DOS keine Möglichkeit mehr zum Eingreifen.

Das Setzen von Maschinensprache-Routinen innerhalb des RAMs als Ein- und/oder Ausgaberoutine ist genauso möglich. ProDOS BASIC enthält dazu eine Erweiterung der Befehle PR# und IN#, mit

```
10 PRINT CHR$(4); "PR# A$300"
```

werden alle folgenden Zeichenausgaben zu einer Routine geleitet, die mit der Speicherstelle $300 beginnt. Bevor ProDOS die entsprechenden Vektoren setzt, findet eine Art Prüfung statt, ob auf der angegebenen Adresse auch wirklich ein Programm steht - der erste Befehl der Routine muß der Befehl "CLD" sein.

Unter DOS 3.3 ist die Umleitung der Ein-/Ausgabe auf eigene Routinen etwas schwieriger - entweder kann man eine Startadresse direkt in die innerhalb von DOS gespeicherten Werte schreiben (CSW: $AA53/54, KSW: $AA55/56) oder (eleganter): man setzt die gewünschten Adressen in CSW/KSW und führt danach einen DOS-Kaltstart via "JSR $3EA" durch. In diesem Fall akzeptiert DOS die auf $36-$39 gespeicherten Adressen als die "richtigen" und setzt sie bei jeder Ein-/Ausgabe neu ein. Ein dritter und letzter Weg ist natürlich das komplette Abhängen von DOS mit den bereits gezeigten Befehlen PR#0 und IN#0.

Periphere Zusatzkarten und primäre I/O-Geräte

Zusatzkarten lassen sich unter dem Gesichtspunkt "I/O" in drei Klassen aufteilen. Die erste Kategorie hat eigene Firmware im Bereich $CnXX wie "normale" 80-Zeichen-Karten, Druckerschnittstellen oder der Diskettencontroller, die zweite kann als primäres I/O-Gerät arbeiten, obwohl sie keinen eigenen ROM im Bereich $CnXX hat, und die letzte hat meistens überhaupt nichts mit Zeichenein- und Ausgabe zu tun.

Zusatzkarten mit eigenem ROM im Bereich $CnXX funktionieren im Apple //e nur dann im Steckplatz 3, wenn der spezielle Steckplatz unbelegt ist. Das liegt daran, daß die Firmware des //e auf den Befehl PR#3 mit der Aktivierung der 80-Zeichen-Routinen im ROM der Hauptplatine reagiert, wenn eine RAM-Karte im speziellen Steckplatz installiert ist. Das Problem liegt im Prinzip: Die 80-Zeichen-Routinen sind im Bereich $C3XX und $C800-$CFFF gespeichert, also in den I/O SELECT'- und I/O STROBE'-Bereichen für Steckplatz 3. Durch eine Aktivierung der Firmware wird SLOTC3ROM zurückgesetzt, INTC8ROM wird gesetzt und damit die Erzeugung von I/O SELECT' für Steckplatz 3 sowie von I/O STROBE' für alle Steckplätze unterdrückt. Karten, die nicht über I/O SELECT' gesteuert sind, geraten dagegen nicht in Konflikt mit den 80-Zeichen-Routinen und können auch ohne Einschränkung in Steckplatz 3 benutzt werden.

Wie bereits gesagt, simuliert die Firmware zusammen mit SLOTC3ROM und INTC8ROM eine "echte" 80-Zeichen-Karte in Steckplatz 3, die auch auf I/O SELECT' und I/O STROBE' reagiert. Meiner Ansicht nach ist das ein echter Kritikpunkt - der 80-Zeichen-Modus sollte automatisch sein und nicht mögliche Konflikte mit anderen I/O-Geräten heraufbeschwören. In dem von Apple, Inc. gewählten Design muß der 80-Zeichen-Modus durch ein explizites "PR# 3" angeschaltet werden, und er wird durch RESET wieder abgeschaltet. Die Konzeption als Zusatzkarte macht ein gleichzeitiges Aktivieren anderer Karten via PR# und IN# unmöglich. Und wo wir gerade dabei sind: Es wäre auch schöner, wenn man nicht noch extra eine Zusatzkarte kaufen müßte, um 80 Zeichen pro Zeile zu erhalten.

Die zweite Kategorie von Zusatzkarten entspricht im Prinzip der ersten - nur daß der Benutzer entsprechende Treiberprogramme erst von der Diskette laden muß, bevor die Karte aktiviert werden kann. In den meisten Fällen wird sich dieses Programm unterhalb von HIMEM "einlagern" und CSW/KSW entsprechend setzen. Abgesehen von einem kleinen Vorteil wegen der einfachen Veränderbarkeit derartiger Programme weist dieses Konzept ansonsten nur Nachteile auf: Eine Karte mit eigenem Programm im ROM kann nicht nur bequem via PR# und IN# angesteuert werden, sie sind auch über die meisten kommerziellen Programme aktivierbar - Textverarbeitungen, Spreadsheets und andere Programme, die zusammen mit RAM-Treibern arbeiten, sind so gut wie nicht auf dem allgemeinen Markt erhältlich.

Die dritte Kategorie von Zusatzkarten wird normalerweise nicht via CSW/KSW angesteuert - eine 16k-Karte ist z.B. in den allermeisten Fällen eine schlichte Speichererweiterung. Eine RAM-Karte mit 128 kByte oder mehr kann dagegen bereits wieder zusammen mit einem entsprechenden Treiber geliefert werden, der zuerst in die Karte hineingeladen und dann in das Betriebssystem eingebunden wird. Unter DOS 3.3 ist so etwas nur via "Patch" möglich, mit ProDOS geht es völlig problemlos.

Eine Karte mit einem anderen Prozessor darauf wird wohl in den seltensten Fällen via KSW/CSW "eingebaut" werden, eine Karte mit einem Sprachsynthesizer vielleicht im Rahmen eines größeren Steuerprogramms.

Das Verständnis, in welche dieser Kategorien im Apple installierte Karten fallen, ist ein großer Schritt in Richtung des vollständigen Begreifens der gesamten I/O-Struktur - die so genial und facettenreich aufgebaut ist, daß die "Seele" des Apple unter der Kontrolle einer beliebigen Zusatzkarte oder ihres dazugehörigen Steuerprogramms sein kann. (Was ist ein Apple mit einer Z80-Karte? Ein CP/M-Rechner mit einem 6502 als I/O-Prozessor? Ein 6502-Rechner mit CP/M-Fähigkeit? Ein Bussystem, dessen Master meistens ein Z80 ist?). Falls diese Kontrolle einmal zusammenbricht und nichts mehr so funktioniert, wie es soll, dann bleibt dem Benutzer nur seine eigene Gewitztheit, um die Fehlerursache zu erkennen und die entsprechende Karte hinauszuwerfen.

Die Zeitabläufe des I/O

Das "Timing" aller I/O-Operationen ist eine Funktion der aus dem Adreßbus herausdekodierten Steuersignale, d.h. es ist vom Adreßbus abhängig. Der gesamte I/O auf der Hauptplatine sowie der größte Teil des peripheren I/O wird über Signale gesteuert, die innerhalb der MMU, der IOU oder dem peripheren Adreßdekoder erzeugt werden. An einem Zugriff auf eine DEVICE SELECT'-Adresse sind so gut wie alle Zeitabläufe beteiligt bzw. darstellbar - deshalb wird ein solcher Zugriff im folgenden Beispiel verwendet.

Bild 7.7 zeigt die Zeitabläufe für Schreib- und Lesezugriffe auf die Adresse $C090. Die Steuersignale für den Datenbus sind bei Schreibzugriffen auf $C090 dieselben wie bei einem Schreibzugriff auf den RAM - in allen Schreibzyklen außer bei DMA sind der Datenbus der Hauptplatine und der periphere Datenbus Empfänger der vom 6502 ausgegebenen Daten. I/O-Lesezugriffe unterscheiden sich dagegen vom RAM-Lesezugriffen in dem Punkt, daß MD IN/OUT' bei I/O-Zugriffen während PHASE0 auf "1" geht und den peripheren Datenbustreiber in Richtung auf die CPU schaltet, damit von I/O Geräten ausgegebene Daten von der CPU gelesen werden können.[8]

Die Reihenfolge der wichtigsten Ereignisse und Signalfolgen ist:

1. CASEN' und CXXX gehen auf den Pegel "1", nachdem eine Adresse im Bereich $CXXX auf dem Adreßbus gültig geworden ist. Diese beiden Signale sind innerhalb der MMU zwar nicht durch PHASE0 gesteuert, die durch CXXX aktivierbaren Bausteine werden aber über PHASE0 (oder den Pegel "0" von PHASE1) geschaltet. Der Pegel "1" von CASEN' sperrt den Datenverkehr der CPU mit dem RAM der Hauptplatine, derselbe Pegel für CXXX entsperrt die Aktivierung der I/O-Signale während PHASE0.
2. Das Signal C0XX' wird nach der steigenden Flanke von PHASE0 aktiv und nach Ende von PHASE0 wieder inaktiv, wenn der Adreßbus während dieser Zeit eine Adresse im Bereich $C0XX enthält. Über $C0XX' werden der 4 zu 16 Dekoder (C10) und weitere Dekodierungen innerhalb der IOU aktiviert. Das Zeitverhalten von I/O SELECT' ist identisch mit dem von C0XX'.
3. C09X' (DEVICE SELECT' für Steckplatz 1) wird nach der fallenden Flanke von C0XX' aktiv und erst durch die fallende Flanke von PHASE0 wieder abgeschaltet. Das Zeitverhalten der Signale C04X', C06X', C07X' und der anderen DEVICE SELECT's ist dasselbe wie das von C09X'.

---

8 Das Lesen von der Tastatur stellt dabei eine Ausnahme dar - die Steuersignale für den Datenbus werden auf dieselbe Weise wie bei einem Lesevorgang des Hauptplatinen-ROMs geschaltet, der einzige Unterschied besteht darin, daß die MMU anstelle von ROMEN1' oder ROMEN2' das Signal KBD' aktiviert.

4. Der periphere Datenbustreiber ist bei Schreibzyklen während PHASE1 komplett abgeschaltet. Videodaten des Hauptplatinen-RAMs sind deshalb bei Schreibzyklen der CPU an den Steckplätzen während PHASE1 nicht verfügbar. Während PHASE0 wird der periphere Datenbustreiber auch bei Schreibzyklen der CPU wieder aktiviert, der periphere Datenbus erhält am Anfang dieser PHASE zuerst noch Videodaten des Hauptplatinen-RAMs, danach das vom Prozessor ausgegebene Datum.

5. Bei einem Lesezugriff des Prozessors auf $C090 geht MD IN/OUT' als Reaktion der MMU auf die steigende Flanke von PHASE0 auf "1", die Laufzeit innerhalb der MMU liegt bei rund 60 nsec. Damit wird der periphere Datenbustreiber so umgeschaltet, daß der Datenbus der Hauptplatine Daten vom peripheren Datenbus empfängt. Die angesprochene Karte in Steckplatz 1 kann nun als Reaktion auf das entsprechende DEVICE SELECT' Daten auf den peripheren Datenbus legen. Wenn die Karte nicht reagiert (oder überhaupt keine Karte im Steckplatz installiert ist), speichert der periphere Datenbus die Videodaten der Hauptplatine, die während der steigenden Flanke von MD IN/OUT' auf diesen Bus gelangt sind. Diese Daten werden über den peripheren Datenbustreiber zurück zum Datenbus der Hauptplatine geliefert und von der CPU gelesen.

6. MD IN/OUT' fällt bei Lesezugriffen der CPU auf $C090 am Ende von PHASE 0 als Reaktion der MMU auf die fallende Flanke von PHASE0 wieder auf "0", die Laufzeit innerhalb der MMU liegt bei rund 50 ns. Der periphere Datenbustreiber wird dadurch wieder so geschaltet, daß er den Datenbus der Hauptplatine liest und diese Daten an den peripheren Datenbus weitergibt. Eine durch den vorherigen Zugriff angesprochene Karte muß spätestens zu diesem Zeitpunkt die Kontrolle des peripheren Datenbusses wieder abgeben, sonst kommt es zum Kurzschluß mit dem Treiber. MD IN/OUT' fällt in den meisten (wahrscheinlich in allen) Apple //e nach der fallenden Flanke von PHASE2 des 6502.

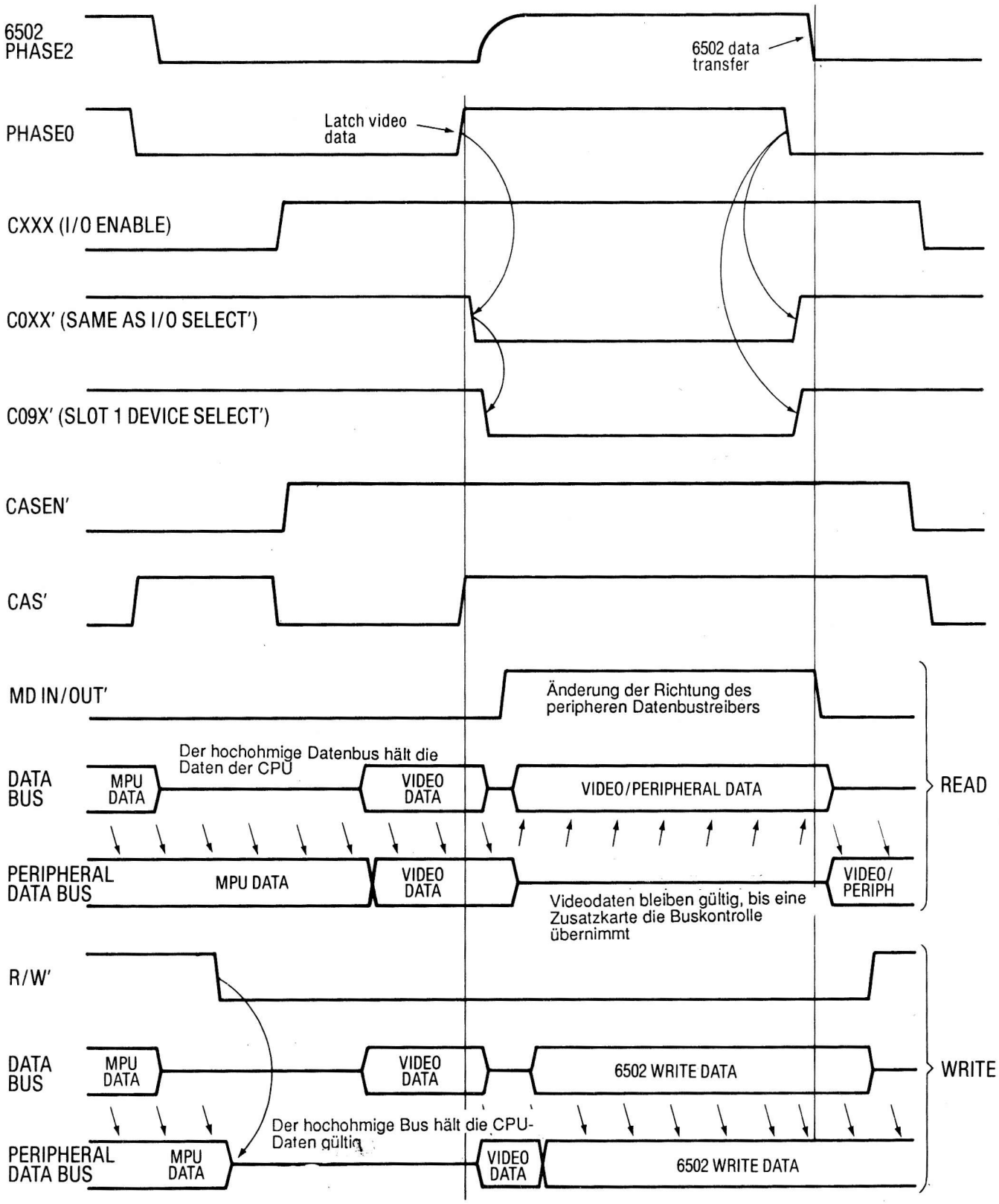
Die Signale DEVICE SELECT' und I/O SELECT' werden von den meisten Zusatzkarten benutzt, um bei Lesezugriffen der CPU dem Datenbus Daten zu übergeben. Wie Bild 7.7 zeigt, ist das Zeitverhalten beider Signale für diesen Zweck zwar nicht gerade das Gelbe vom Ei - es funktioniert aber trotzdem. Beide Signale haben dasselbe Zeitverhalten wie C0XX', d.h. sie sind zu früh gültig, nämlich schneller als MD IN/OUT' gesetzt werden kann, und sie werden auch zu früh wieder ungültig, nämlich vor der fallenden Flanke von PHASE2 des 6502.

Tatsächlich ergeben sich durch dieses Zeitverhalten nur dann Probleme, wenn ein relativ schneller Baustein (wie z.B. das Datenregister einer Disketten-Controllerkarte) sofort mit Datenausgaben reagiert. Bei langsameren Bausteinen wie NMOS-ROMs ist die Verzögerung zwischen Aktivierung und tatsächlicher Datenausgabe groß genug, und die Daten erscheinen zu den richtigen Zeiten. Im praktischen Betrieb funktionieren DEVICE SELECT' und I/O SELECT' auch mit schnellen (bipolaren) Bausteinen - der kurze Kampf um die Vorherrschaft auf dem Datenbus findet in einem Zyklusabschnitt statt, in dem keine kritischen Datentransfers gestört werden können, und so wie es aussieht, wird dadurch auch der allgemeine Störpegel des Systems nicht auf einen kritischen Wert erhöht. Die zu frühe Abschaltung beider Kontrollsignale ist ebenfalls kein Problem, weil die ausgegebenen Daten wie üblich längere Zeit auf dem nicht terminierten peripheren Datenbus gültig bleiben. Kurz und gut: Der Apple kommt trotz dieses recht eigenwilligen Designs ungeschoren davon - Konstrukteure von Zusatzkarten sollten sich aber über dieses Zeitverhalten im klaren sein.

Das Timing des Signals C06X', mit dem der serielle Eingangsmultiplexer aktiviert wird, ist dasselbe wie von DEVICE SELECT': bei einem Lesezugriff der CPU auf $C06X ist C06X' aktiv, bevor MD IN/OUT' umgeschaltet werden kann, der serielle Eingangsmultiplexer und der periphere Datenbustreiber führen deshalb einen kurzen Kampf um die Vorherrschaft auf D7 des peripheren Datenbusses aus. C06X' ist während der fallenden Flanke von PHASE2 bereits wieder inaktiv, aber auch in diesem Fall speichert der hochohmige periphere Datenbus D7 lange genug, um einen korrekten Datentransfer zur CPU zu gewährleisten.

Bild 7.7 zeigt, daß die CPU Videodaten der vorhergegangenen PHASE1 liest, wenn eine angesprochene Zusatzkarte auf DEVICE SELECT' nicht mit der Übernahme der Kontrolle des Datenbusses reagiert. Der Grund dafür ist etwas kompliziert: Am Anfang des Zyklus enthält der Datenbus der Hauptplatine die Videodaten, der periphere Datenbustreiber ist lange genug auf "Lesen des Hauptplatinen-Datenbusses" geschaltet, um diese Daten auf den peripheren Datenbus zu kopieren. Danach folgt als Reaktion auf den Lesezugriff der CPU die Umschaltung von MD IN/OUT': wenn kein aktiver Sender existiert, wird der Bus hochohmig, der Treiber kopiert die Daten auf den Datenbus der Hauptplatine zurück, wo sie von der CPU gelesen werden. Am Ende des Zyklus wird die Richtung des peripheren Datenbustreibers erneut umgeschaltet, die mittlerweile wieder auf dem Datenbus der Hauptplatine liegenden Daten werden dadurch ein zweites Mal auf den peripheren Datenbus kopiert. Derselbe Ablauf gilt für sämtliche Lesezugriffe der CPU auf den Bereich $C020-$CFFF, wenn daraufhin aktive Reaktionen von Zusatzkarten ausbleiben.

Bild 7.7 Zeitdiagramm eines CPU-Zugriffs auf $C090



Der spezielle Steckplatz

Zwischen dem speziellen Steckplatz und den peripheren Steckplätzen bestehen einige Unterschiede grundsätzlicher Natur. So ist der spezielle Steckplatz nicht mit den Kontrollsignalen des 6502 oder direkt mit dem Adreßbus verbunden, er hat keine Aktivierungssignale wie DEVICE SELECT', keinen zugeordneten Adreßbereich innerhalb des Apple und außerdem nur sehr beschränkte Möglichkeiten für DMA. Alles in allem kann man ihn schwerlich als Erweiterungsmöglichkeit der Ein-/Ausgabe bezeichnen - er ist vielmehr so verdrahtet, daß eine installierte Karte in eine bereits vorhandene Funktionsgruppe des Apple //e integriert wird und sie erweitert. Bild 7.8 zeigt die mit dem speziellen Steckplatz verbundenen Signale, aufgeteilt in funktionelle Gruppen. Wie zu sehen ist, gibt es drei Hauptgruppen: "RAM", "Timing" und "Video".

Die RAM-Signalgruppe besteht aus dem gemultiplexten RAM-Adreßbus, dem Videodatenbus, dem Datenbus, R/W', R/W'80, CASEN' und EN80'. Wie in Kapitel 5 gezeigt, sind diese Signalleitungen (zusammen mit einigen Zeittakten) ausreichend für eine Zusatzkarte mit 64 kByte RAM, der während PHASE1 Videodaten liefert und während PHASE0 der CPU zur Verfügung steht. Das ist auch die Funktion, die normalerweise mit diesem Steckplatz assoziiert wird - einfach deshalb, weil es die Hauptfunktion kommerziell vertriebener Karten für den speziellen Steckplatz ist.

Das Design des Apple //e ist für einen kompletten Zugriff auf 64 kByte RAM im speziellen Steckplatz ausgelegt - es wäre allerdings möglich, den Bereich $C07X' weiter zu dekodieren und darüber mehrere 64k-Bänke innerhalb dieses Steckplatzes umzuschalten. Aus irgendwelchen Gründen muß sich schließlich das Signal C07X' an den Kontakt 6 des speziellen Steckplatzes verirrt haben..

In diesem Kontext kann man C07X' als eine Art DEVICE SELECT' dieses Steckplatzes bezeichnen - C07X' kann für diesen Zweck allerdings nur solange benutzt werden, wie ein Programm durch das Zurücksetzen der Timer für die Spielsteuerungen nicht in Konflikt mit einer Bankumschaltung kommt.

Eine weitere bereits praktisch genutzte Verwendungsmöglichkeit für den speziellen Steckplatz ist der Einbau eines anderen Prozessors zusammen mit einer oder mehreren RAM-Bänken. Das bedeutet, daß hier praktisch ein unabhängiger Mikrocomputer installiert werden kann, der zusammen mit dem 6502 ein echtes Co-processing ausführt, der Datenaustausch zwischen beiden findet über den AUX-RAM statt, auf den ja auch der 6502 zugreifen kann. Sämtlicher I/O findet dabei über den 6502 auf der Hauptplatine statt, inklusive dem Laden von Programmen und deren Verschiebung in den AUX-RAM, wo sie dann von der anderen CPU ausgeführt werden können.

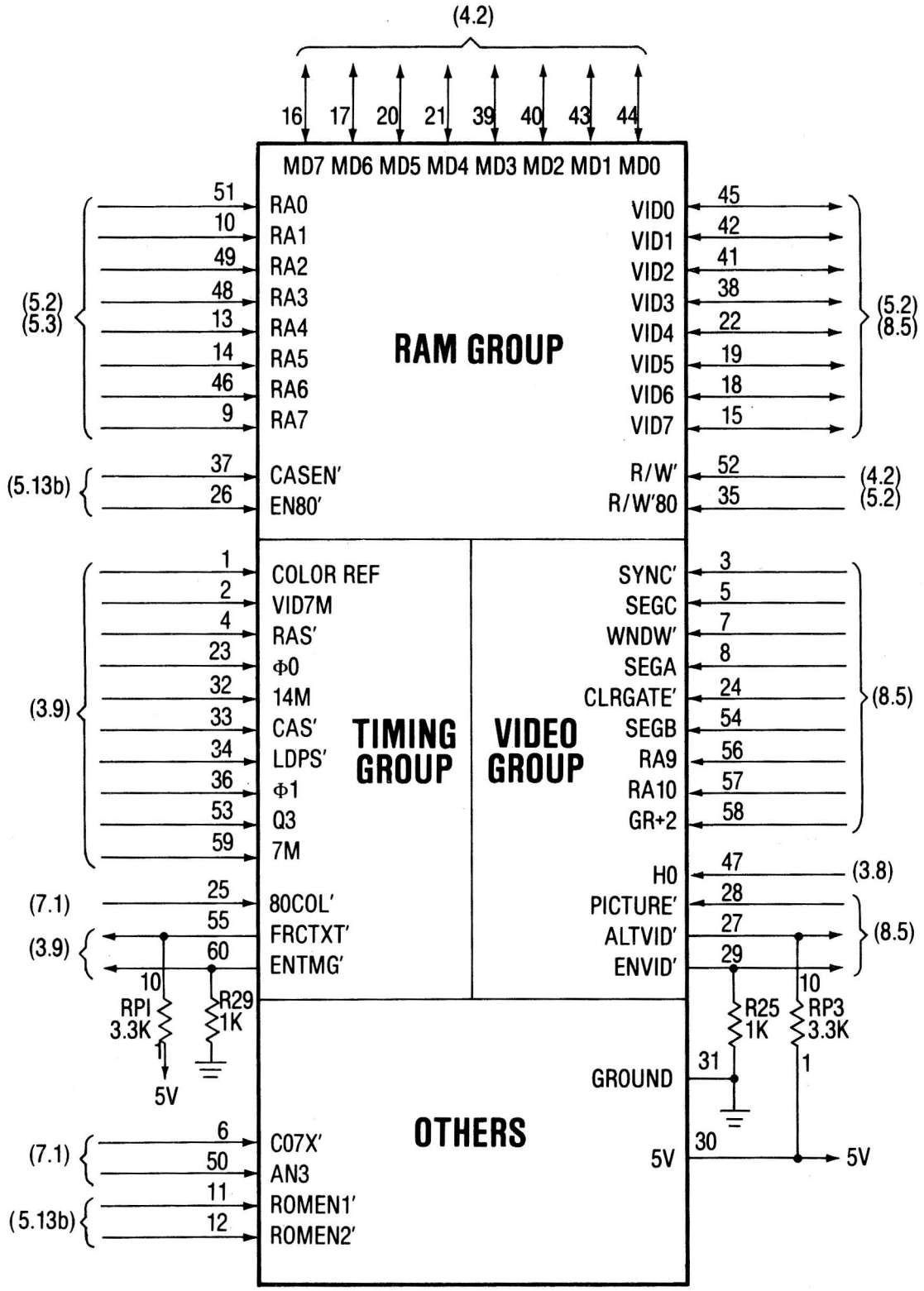
Die Gruppe der Zeitsignale besteht aus einem kompletten Satz von Verbindungen zu allen Ausgangssignalen des Taktgenerators zusammen mit der Leitung ENTMG'. Eine Diagnose-Karte im speziellen Steckplatz kann somit sämtliche Zeittakte des Systems überwachen und prüfen, ob Verhalten und Abfolge korrekt sind. Wenn eine derartige Karte ENTMG' auf den Pegel "1" bringt, wird darüber der HAL abgeschaltet: danach können eigene Zeittakte der Karte in das System eingespeist werden (vgl. Bild 3.9). Falls noch eine dazugehörige Karte in Steckplatz 1 CLKEN' ebenfalls auf "1" setzt, kann die Karte im speziellen Steckplatz auch noch ein eigenes 14M-Signal erzeugen und die Hauptplatine damit versorgen.

Die Videosignalgruppe besteht aus Adreß- und "Enable"-Eingängen zum Video-ROM sowie den Signalleitungen PICTURE' , SYNC', CLRGATE' und ALTVID'.[9] Eine Karte im speziellen Steckplatz kann sämtliche dieser Signale überwachen und damit die Operation des Computers überprüfen bzw. Fehlerstellen ausfindig machen. Weiterhin ist es möglich, den Video-ROM (und damit PICTURE') zu sperren und ein eigenes Videosignal über ALTVID' einzuspeisen, wenn vorher ENVID' auf den Pegel "1" gebracht wird. Über diese Signalleitungen könnte man auch verschiedene Bereiche des Bildschirms unabhängig vom gesetzten Videomodus zu Testzwecken auf Weißpegel bringen.

---

9 Eine andere Möglichkeit der logischen Zusammenfassung für die Gruppe ist "alle Video-Ausgangssignale der IOU plus PICTURE', ENVID' und dem Videodatenbus". Der Videodatenbus kann interessanterweise sowohl als Teil der Videogruppe als auch als Teil der RAM-Gruppe betrachtet werden.

Bild 7.8 Schaltplan der Verbindungen des speziellen Steckplatzes



Sowohl die Zeitsignale als auch die Videosignalgruppe sind wohl hauptsächlich für Diagnosezwecke dermaßen komplett mit dem speziellen Steckplatz verbunden. Sie unterstützen bzw. ermöglichen automatisierte Überprüfungen, Fehlersuche und Tests der Apple //e-Hauptplatinen am Ende des Herstellungsprozesses mit Hilfe spezieller Testkarten. Gut - es ist sicher möglich, die eine oder andere Karte zu konstruieren, die Signale dieser Gruppen auch im praktischen Betrieb nutzt, primär dürften sie aber für Testzwecke verwendet werden. Genauso wahrscheinlich ist es übrigens, daß so gut wie alle Karten auch 64 oder 128 kByte RAM enthalten werden - die Idee mit (mindestens) 128 kByte in einem Apple hat sich inzwischen weit genug verbreitet.

Einige Signale des speziellen Steckplatzes passen nicht so ganz in die drei erwähnten Gruppen hinein (die auch nicht als absolute Unterteilungen, sondern nur als Versuch einer anschaulichen Darstellung gewertet werden dürfen.) ROMEN1', ROMEN2', EN80' und CASEN' könnte man in einer "MMU-Gruppe" zusammenfassen, es ist auch möglich, daß Diagnose-Karten ebenfalls MMU-Funktionen prüfen. Auf Hauptplatinen der Rev. A könnte man ROMEN1', ROMEN2' und ENFIRM als ROM-Gruppe betrachten - durch Setzen von ENFIRM auf den Pegel "0" könnte eine Diagnose-Karte den ROM der Hauptplatine abschalten (s. Bild 6.1). Was auch immer mit ENFIRM beabsichtigt war - Apple, Inc. hat dieses Signal in der Rev. B hinausgeworfen und durch FRCTXT' ersetzt (s. Bild 3.9). Dieser Wechsel ist auch die größte Änderung gegenüber der Rev. A: über FRCTXT' ("FoRCe TeXT" = erzwinge TEXT) ist der "80-Zeichen-Betrieb", d.h. die Aktivität der 80-Zeichen-Karte auch in den Grafikmodi möglich und liefert so die doppelt hohe Auflösung.

Kapitel 8

Der Videogenerator

Die Möglichkeit der Datenausgabe über einen Bildschirm ist einer der Meilensteine in der Entwicklung der Computer. Zusammen mit der Tastatur ermöglicht diese Ausgabeform eine direkte Kommunikation zwischen Mensch und Maschine: Versuchen Sie sich demgegenüber einmal vorzustellen, daß Sie sich mit Ihrem Apple über eine Art Fernschreiber unterhalten müßten - die Arbeit mit "wichtigen" Programmen wie Textverarbeitung, Spreadsheets und Donkey Kong wäre weitaus unerfreulicher.

Natürlich verfügt der Apple über die Möglichkeit der Bildschirmausgabe von Daten, und ein großer Teil der Hauptplatinen-Elektronik ist mit der Erzeugung eines Videosignals beschäftigt. Wie in den vorhergehenden Kapiteln mehr als einmal gezeigt, werden große Teile der Busstruktur, der Taktsignale und der RAM-Adressierung im Apple //e durch die Tatsache bestimmt, daß die eigentlich entgegengesetzten Aufgaben "Programmausführung" und "Bilderzeugung" simultan ausgeführt werden. Speziell zwei Funktionsgruppen des Apple //e sind einzig und allein für die Bilderzeugung zuständig: der *Videoscanner* (innerhalb der IOU) sowie der *Videogenerator*, der sich teilweise innerhalb und teilweise außerhalb der IOU befindet. Beide sind Teil eines komplexen Aufbaus, mit dessen Hilfe im Apple ablaufende Programme die Videoausgabe kontrollieren können.

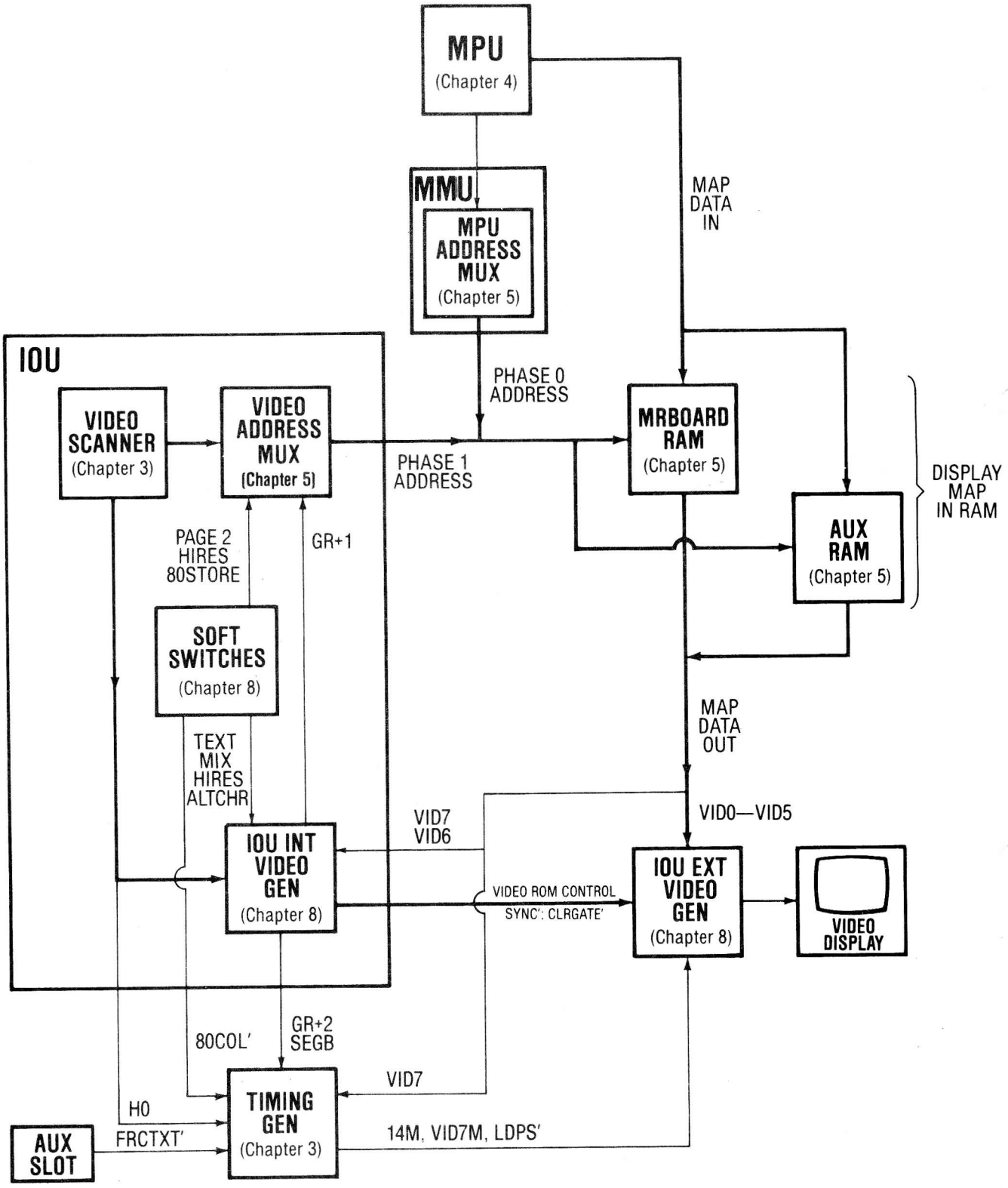
Bild 8.1 ist ein vereinfachtes Schema der Bilderzeugung und ihrer Kontrolle im Apple //e. Wie in diesem Bild zu sehen ist, kontrolliert die CPU die Erzeugung des Videosignals auf sehr indirekte Weise: über ein Programm setzt sie einen Videomodus, berechnet bestimmte Adressen im für diesen Modus reservierten Bildspeicherbereich und beschreibt die entsprechenden Speicherstellen mit Daten. Die Arbeit der CPU (und damit des Programmierers) erschöpft sich in diesen Aufgaben - mit der tatsächlichen Bilderzeugung hat sie nichts zu tun. Diese Arbeit wird von den beiden speziell dafür vorgesehenen Funktionsgruppen übernommen: der Videoscanner erzeugt fortlaufend Adressen und bringt Videodaten zur Auslese, der Videogenerator verarbeitet diese Daten zum Signal für den Monitor/Fernseher weiter. Selbst dann, wenn die CPU via DMA' oder RDY gestoppt wird, fährt der Apple mit der Erzeugung eines Videosignals fort und zeigt in diesem Fall einen "eingefrorenen" Bildschirm. Vergleichen Sie diese indirekte Beteiligung einmal mit einer Druckerschnittstelle, bei der die CPU tatsächlich die einzelnen Datenworte Byte für Byte zu einer speziellen Adresse schreibt!

Der RAM-Zugriff des Videoscanners ist eine Form von *simultanem DMA* und für Programme vollkommen transparent - die CPU hat nicht die geringste Ahnung, was der RAM während PHASE1 so alles treibt.

Der Videogenerator muß die vom Scanner "ausgetriebenen" Daten je nach Videomodus als TEXT, LoRes oder HiRes interpretieren und daraus ein Signal bilden, mit dem ein Monitor oder Fernseher dazu veranlaßt wird, das gewünschte Bild zu zeigen. Dieses Signal wird als **VIDEO** bezeichnet und ist das Hauptthema dieses Kapitels - es ist eines der komplexeren im Apple //e. In den folgenden Abschnitten finden Sie seine Zusammensetzung und seine Erzeugung im Videogenerator besprochen. Die Integration des Prozesses "Bilderzeugung" in die restliche Elektronik des //e finden Sie detailliert im jeweiligen Zusammenhang erklärt: Kapitel 1 enthält eine allgemeine Beschreibung der Videomodi, Kapitel 2 bespricht den Videoscanner zusammen mit der Busstruktur, Kapitel 3 den Videoscanner selber und Kapitel 5 die Adressierung des RAMs.

Bevor wir uns in die Details des Videogenerators stürzen, folgt hier erst einmal eine Beschreibung, wie ein Video-Signal im allgemeinen und das VIDEO-Signal des //e im speziellen aufgebaut ist.

Bild 8.1 Blockstruktur der Bilderzeugung im Apple //e



Das VIDEO-Signal des Apple //e

Schalten Sie einmal für eine oder zwei Minuten Ihren Fernseher ein: Das Bild, was Sie dabei zu sehen bekommen, entstammt einer Kamera, die ein *composite Videosignal* (zusammengesetztes Videosignal, BildAustastSynchronsignal oder kurz **BAS**-Signal) liefert. Wie der deutsche Name bereits sagt, besteht dieses Signalgemisch aus der Bildinformation selber, den Austastlücken und den SYNC-Signalen. Das BAS-Signal wird zu einem Wandler geführt, der damit (zusammen mit einer zusätzlichen Toninformation) ein *Hochfrequenz-Signal* moduliert. Dieses Signal wird über Sender und deren Relaisstationen im Sendebereich ausgestrahlt und von der Antenne Ihres Fernsehers empfangen. Der Fernseher demoduliert das Hochfrequenzsignal und holt so das BAS-Signal und die Toninformation wieder heraus. Aus dem BAS-Signal werden Steuerinformationen für den Elektronenstrahl gewonnen und damit das Fernsehbild erzeugt.

Der vorangegangene Absatz könnte das Fernsehnetz eines beliebigen Landes beschreiben - weil man sich aber auch hier nicht auf einen allgemeinen Standard einigen konnte, unterscheiden sich die Systeme in verschiedenen Ländern in einigen Details.

Als der erste Apple II in den USA auf den Markt kommen sollte, hatte die FCC[1] schwere Bedenken angemeldet - wenn ein Computer Hochfrequenzsignale erzeugt, gelangt immer ein kleiner Teil davon über die Antennenwirkung der Verbindungsleitung ins Freie und kann den Fernsehempfang in der Nachbarschaft beeinträchtigen. Um diese Schwierigkeiten von vornherein zu umgehen, wurde der Apple so konstruiert, daß er nicht modulierte Hochfrequenz, sondern nur ein BAS-Signal erzeugt. Das vom Apple //e erzeugte VIDEO-Signal ist so gut wie identisch mit dem des alten II(+).[2]

Wenn Sie ein Hochfrequenzsignal mit dem VIDEO-Signal des Apple modulieren, ist darüber der Betrieb eines ganz normalen Fernsehers möglich. Natürlich liefert der entsprechende HF-Modulator für 30 Mark ein Vielfaches an Störstrahlungen - dafür wird er aber von Fremdherstellern und nicht von Apple, Inc. selber angeboten, und die Tatsache, daß beide Produkte (Computer und Modulator) auf dem Markt erhältlich sind, kann kein Ausschuß rückgängig machen. Innerhalb des Fernsehers wird der Modulationsprozeß rückgängig gemacht und das BAS-Signal wieder herausgeholt, die restlichen Schritte der Verarbeitung sind dieselben wie in einem Computermonitor.

Bild 8.2 zeigt die Charakteristika des VIDEO-Signals und gilt gleichermaßen für das amerikanische NTSC-System als auch für die europäische PAL-Norm. Das Signal besteht aus den drei Komponenten **PICTURE** (der eigentlichen Bildinformation), **SYNC** und **COLOR REFERENCE BURST** (einem "Wellenpaket" von COLOR REFERENCE zu Anfang jeder Fernsehzeile). Diese drei Anteile werden so miteinander gemischt, daß ein Monitor sie wieder auseinanderdividieren kann. PICTURE kann dadurch von SYNC unterschieden werden, daß das Videosignal während SYNC-Impulsen auf einem niedrigeren Pegel als zu jeder anderen Zeit ist. Der COLOR REFERENCE BURST wird dagegen nicht am Pegel, sondern am Zeitpunkt erkannt - er kommt nach jedem horizontalen SYNC-Impuls.

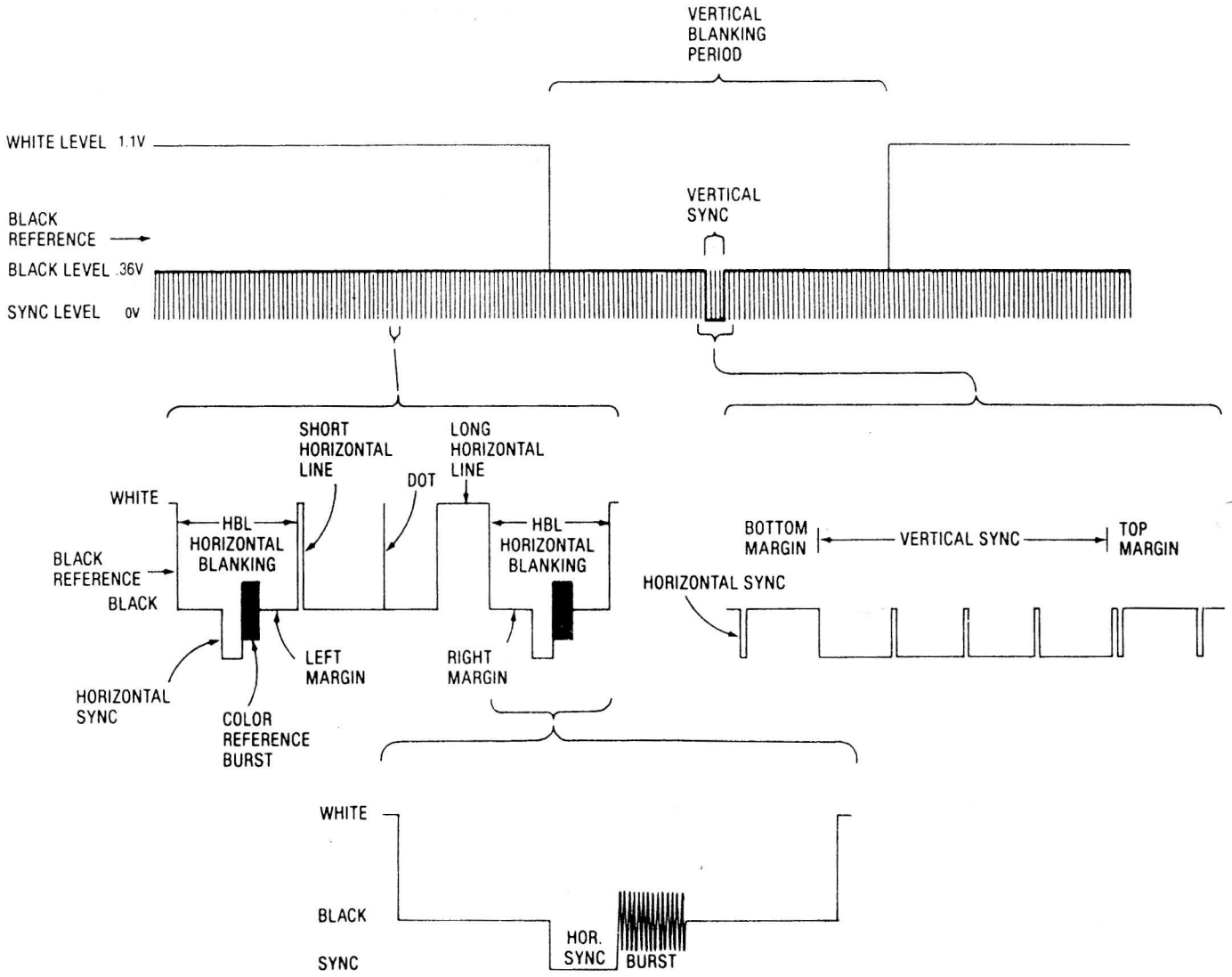
Eine bestimmte Spannung des VIDEO-Signals (0,36 Volt) entspricht nach einer vorher getroffenen Vereinbarung dem *Schwarzpegel*. Über diesem Pegel liegende Spannungen verstärken die Intensität des Elektronenstrahls soweit, daß die entsprechende Stelle im Bild hell dargestellt wird. Das VIDEO-Signal ist nur dann oberhalb des Schwarzpegels, wenn tatsächlich Bildinformation ausgegeben wird - die gesamte restliche Zeit ist die Spannung darunter. Die SYNC-Impulse sind "schwärzer als schwarz", liegen in ihrem Pegel also soweit unterhalb des Schwarzpegels, daß sie von der Elektronik des Monitors als Synchronisationsimpulse erkannt werden. Damit haben wir drei festgelegte Pegelhöhen: den Weißpegel (volle Helligkeit), den Schwarzpegel und den SYNC-Pegel. Das vom Apple erzeugte VIDEO-Signal ist erheblich einfacher gestrickt als ein normales Fernsehsignal, soweit es diese Pegel betrifft: auch in einem Schwarzweiß-Fernseher kann das PICTURE-Signal beliebige Werte zwischen Schwarz- und Weißpegel annehmen und erzeugt so die Farben weiß, schwarz und eine fast unendliche Menge von Graustufen dazwischen - im Apple dagegen gibt es in nicht-farbiger Darstellung nur ein echtes Schwarzweiß-Bild ohne Zwischenwerte.

---

1 ("Federal Communications Commission" = Bundeskommission für Kommunikation), ähnlich dem hierzulande zuständigen Ausschuß der Bundespost, allerdings bei weitem nicht so kleinkariert - (Anm. d. Übers.).

2 Das trifft nur für die amerikanische Version zu, soweit ich das beurteilen kann. Die deutsche PAL-Version des II(+) hatte noch diverse Probleme mit der Farberzeugung, der //e hat sie nicht mehr - (Anm. d. Übers.).

Bild 8.2 Das VIDEO-Signal des Apple //e



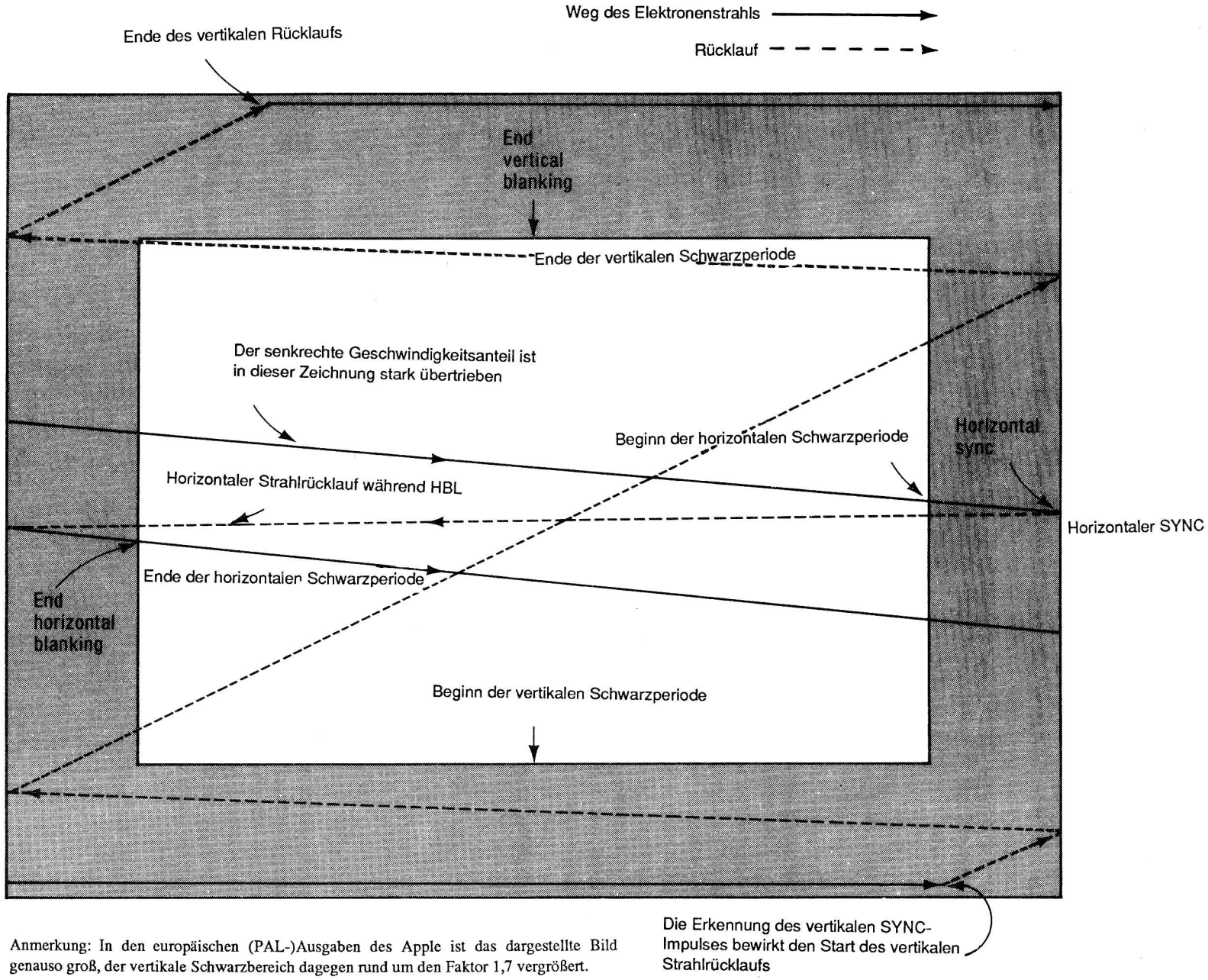
Die horizontalen und vertikalen SYNC-Pulse liegen beide unterhalb des Schwarzpegels. Jede steile Flanke unterhalb des Schwarzpegels wird von der Elektronik als horizontaler SYNC-Impuls, jeder lang anhaltende SYNC-Pegel als vertikaler SYNC-Impuls interpretiert. Der Apple erzeugt einen vertikalen SYNC-Impuls, der dieselbe Länge hat wie vier komplette Fernsehzeilen. Wie von der Norm vorgeschrieben, erhält der Monitor auch während dieser Zeit horizontale SYNC-Impulse.[3]

Auf jeweils 312 horizontale SYNC-Impulse kommt in der PAL-Version des Apple //e ein vertikaler SYNC-Impuls (NTSC: 262 zu 1). Dieser Impuls hat eine Länge von vier Fernsehzeilen und findet in der Mitte von HBL ("Horizontal BLanking" = horizontale Schwarzperiode) statt, die 25 Prozessor-Zyklen dauert. Während dieser Zeit wird das VIDEO-Signal unterhalb des Schwarzpegels gehalten und erzeugt so die linken und die rechten Ränder auf dem Bildschirm (s. Bild 8.3). Das Signal PICTURE wird während der restlichen Zeit erzeugt und füllt damit quasi die Zeit zwischen zwei HBL-Perioden. Der vertikale Sync-Impuls findet in der Mitte von VBL ("Vertical BLanking" = vertikale Schwarzperiode) statt, die 7800 Prozessor-Zyklen, also 120 Fernsehzeilen (NTSC: 4550 Zyklen, 70 Zeilen) dauert. Auch während dieser Zeit wird das VIDEO-Signal unterhalb des Schwarzpegels gehalten und erzeugt so die vertikalen Ränder auf dem Bildschirm.

---

3 Daß sich das VIDEO-Signal zu diesem Zeitpunkt bereits auf dem SYNC-Pegel befindet, spielt keine Rolle: Ein horizontaler SYNC-Impuls ist als "fallende Flanke bis hinunter auf den SYNC-Pegel" definiert - und das läßt sich auch während des vertikalen SYNC leicht erzeugen (s. Bild 8.2, Mitte rechts) - (Anm. d. Übers.)

Bild 8.3 Die Schwarzperioden im VIDEO-Signal erzeugen einen Rahmen im Fernsehbild



Anmerkung: In den europäischen (PAL-)Ausgaben des Apple ist das dargestellte Bild genauso groß, der vertikale Schwarzbereich dagegen rund um den Faktor 1,7 vergrößert.

Nehmen Sie einmal an, daß sich der Elektronenstrahl in der oberen linken Ecke des Bildschirms befindet und der Bildschirm rund 30 Zentimeter breit ist. Innerhalb von 40 Mikrosekunden bewegt sich der Elektronenstrahl von der linken Seite des Bildschirms auf die rechte - dabei hat er eine Geschwindigkeit von rund 27000 Stundenkilometern. Da der Elektronenstrahl in der obersten Zeile des Schirms steht, ist der vertikale SYNC-Impuls gerade beendet worden, wir befinden uns in der zweiten Hälfte von VBL. Der Strahl bewegt sich nun von links nach rechts auf dem Bildschirm - zu sehen ist allerdings nichts, weil VBL das VIDEO-Signal auf dem Schwarzpegel hält. Am Ende der Zeile folgt ein horizontaler SYNC-Impuls, der den Strahlrücklauf zur linken Seite auslöst. Dieser Rücklauf geschieht sehr schnell (d.h. nicht mit schlappen 27000 Sachen, sondern *wirklich* schnell, nämlich rund in einem vierzigstel der Zeit), und darauf folgt die nächste (langsame) Bewegung von links nach rechts. Weil alle Fernsehzeilen ein ganz kleines bißchen "schief" verlaufen, stehen die Zeilen untereinander im Bildschirm. Dieser Kreislauf wird solange unverändert wiederholt, bis sich der Strahl rund zweieinhalb Zentimeter unterhalb der oberen Bildschirmkante befindet.

In dieser letzten nicht dargestellten Zeile endet VBL rund zwei Zentimeter vor der rechten Kante des Bildschirms; da HBL allerdings zur selben Zeit beginnt, bleibt auch diese Zeile komplett schwarz. Wenn der Strahl die rechte Bildschirmkante erreicht, folgt wie üblich ein horizontaler SYNC, der Strahl läuft zurück, und die erste dargestellte Zeile beginnt. Auch in dieser Zeile ist HBL für die ersten rund zwei Zentimeter Weglänge noch aktiv und unterdrückt jede Bildausgabe - erst nach dem Ende von HBL beginnt das VIDEO-Signal, zwischen Schwarz- und Weißpegel hin- und herzuschalten und so dunkle Stellen und helle Punkte innerhalb der Zeile zu erzeugen.

Rund zwei Zentimeter vor der rechten Kante des Bildschirms wird HBL wieder aktiv und setzt das VIDEO-Signal erneut auf den Schwarzpegel, wobei dieser Pegel bis kurz nach Anfang der nächsten Zeile anhält und zwischendurch nur durch den horizontalen SYNC-Impuls noch weiter abgesenkt wird. Der Kreislauf wiederholt sich für insgesamt 192 Fernsehzeilen. Am Ende der Bildperiode der letzten dargestellten Zeile werden VBL und HBL gleichzeitig aktiv: die vertikale Schwarzperiode beginnt. Der Elektronenstrahl durchläuft die restlichen 60 (d.h. (312-192 / 2)) Fernsehzeilen wieder auf dem Schwarzpegel, danach erzwingt der vertikale SYNC-Impuls einen relativ schnellen Rücklauf zur obersten Zeile, bevor sich der gesamte Ablauf wiederholt.

Sowohl NTSC als auch die PAL-Norm benutzen ein als *Zeilensprung* ("Interlace") bekanntes Verfahren: Zur Zeit der Erfindung des Fernsehens war es unter vertretbarem Aufwand nicht möglich, 50 (NTSC: 60) komplette Fernsehbilder pro Sekunde zu erzeugen und darzustellen. Damit ein Fernsehbild nicht noch mehr flimmert, als es das sowieso schon tut, wird ein genialer Trick benutzt: jedes Fernsehbild besteht aus zwei *Halbbildern*, das erste Halbbild füllt die Zeilen 1,3,5,7,9.., das zweite Halbbild die Zeilen 2,4,6,8,10... Obwohl damit nur 25 komplette Bilder pro Sekunde dargestellt werden, flimmert das Bild trotzdem mit einer Frequenz von 50 Hertz (also knapp an der Wahrnehmungsgrenze) - alle fünfzigstel Sekunde wird ein Halbbild aufgebaut und der Elektronenstrahl durchläuft den Bildschirm von oben nach unten, durch die zeilenweise Verschachtelung der beiden Halbbilder fällt es nicht auf, daß dabei eigentlich nur ein halbes Bild neu gezeichnet wird. Ein Fernsehbild nach der PAL-Norm besteht damit aus zwei Halbbildern mit jeweils 312.5 Zeilen, das ergibt zusammen eine vertikale Auflösung von 625 Zeilen (NTSC: ein Halbbild mit 262.5, das gesamte Bild mit 525 Zeilen). Die zeitliche Folge der SYNC-Impulse des Apple ist nicht so ausgelegt, daß dadurch ein Zeilensprung im Monitor ausgelöst wird - beide erzeugten Halbbilder sind exakt gleich und werden von Computermonitoren auch jeweils auf denselben Zeilen dargestellt.

Farbsignale

Der COLOR REFERENCE BURST ist ein "Paket" aus 14 Perioden des Signals COLOR REFERENCE, das vom Taktgenerator des Apple erzeugt wird. Farbfernseher bzw. farbfähige Monitoren enthalten eine zusätzliche Schaltung, die nach jedem horizontalen SYNC-Impuls ein 3.56 MHz-Signal (NTSC- 3.58 MHz) erwartet. Mit diesem Wellenpaket ist der Monitor in der Lage, für die Dauer einer Fernsehzeile einen eigenen Generator exakt zu synchronisieren.[4] Der COLOR BURST kommt direkt hinter dem horizontalen SYNC-Signal, also während des zweiten Teils von HBL, und wird somit zu einem anderen Zeitpunkt übertragen als die Bildinformation. Damit können sich beide Signale nicht gegenseitig stören.

Der COLOR BURST wird vom Apple nicht erzeugt, wenn der TEXT-Softswitch gesetzt ist. Damit werden farbige Ränder um die Textzeichen herum vermieden: Farbfähige Geräte haben einen "Color Killer" als Zusatzschaltung,

---

4 Selbstverständlich hätte man auch eine Fernsehnorm definieren können, mit der die Farbinformation auf andere Weise übertragen wird - nur wäre dann der Empfang von farbigen Sendungen auf Schwarzweiß-Geräten nicht mehr möglich. Als die Schwarzweiß-Fernsehnormen festgelegt wurden, haben nicht einmal als Spinner bekannte Physiker an farbige Übertragungen gedacht - schließlich gab es 1941 gerade die ersten Farbfilme. Die spätere Erfindung einer Farbfernsehnorm bestand hauptsächlich daraus, in die bereits existierende Norm zusätzliche Informationen so hineinzutricksen, daß Schwarzweiß-Geräte dadurch nicht gestört wurden - (Anm. d. Übers)

der auf das Ausbleiben des COLOR BURST mit der internen Abschaltung der Farbdarstellung reagiert.[5] In den Modi "TEXT und Grafik gemischt" wird der COLOR BURST für die Textzeilen nicht unterdrückt - als Folge davon erscheinen in diesen Zeilen ausgegebene Zeichen mit grünen, violetten und weißen Rändern (vorausgesetzt, es wird in einfacher Auflösung gearbeitet).

In der Art der Farbcodierung unterscheiden sich die Fernsehsysteme PAL und NTSC ähnlich stark wie in der Zeilenzahl. Leider ist es in diesem Fall mit einer relativ einfachen Änderung wie einem anderen Startwert für den Vertikal-Zähler in der IOU nicht getan: Um nicht den gesamten Bildgenerator (und damit einen größeren Teil der Hauptplatine) auf europäische Verhältnisse "umstricken" zu müssen, hat Apple, Inc. den //e wie einen II+ mit einer "Eurocolor"-Karte konstruiert - das VIDEO-Signal wird erst einmal der NTSC-Norm gemäß erzeugt und dann auf die PAL-Norm gewandelt. Bevor wir uns detaillierter um den Wandler kümmern, gehen wir deshalb erst einmal auf die Erzeugung der NTSC-Signale ein.

Die Bildinformation innerhalb des NTSC-Systems setzt sich aus zwei Komponenten zusammen - einer Information über die Farbe (*"Chroma-Signal"*) und einer Information über die Helligkeit (*"Luminanz-Signal"*). Beide Informationen werden im Empfängerteil eines Fernsehers zusammen behandelt. Erst nach der Demodulation des BAS-Signals aus der Hochfrequenz (also bei Computermonitoren, die nur mit BAS arbeiten, praktisch innerhalb der ersten Stufe) werden sie voneinander getrennt und einzeln weiterverarbeitet.

Das Chrominanzsignal ist eine sehr komplexe Kombination aus zwei Signalen mit 3.58 MHz, die zueinander um 90 Grad phasenverschoben sind. Über eine mehr als erstaunliche mathematische/elektronische Manipulation des Gesamtsignals wird aus diesem Signal die Farbe für jeden Punkt des Bildschirms herausgeholt und gleichzeitig aus dem Luminanzsignal die Helligkeit. Ein Teil dieser Mysterien liegt darin, daß beide Signalkomponenten aus einander überlappenden Frequenzbereichen bestehen können und trotzdem keine Interferenzen produzieren. Wir machen es uns an dieser Stelle etwas einfacher und stellen schlicht fest, daß ein Computermonitor die beiden Signalkomponenten voneinander trennt und aus dem Chroma-Signal die Einzelinformationen zur Ansteuerung der roten, der blauen und dergelben Farbkanone durch Vergleich der Phasenlage zu COLOR REFERENCE bzw. zum COLOR BURST herausholt.

Im Videogenerator des Apple ist es noch ein bißchen einfacher: Da wir keine Helligkeitsabstufungen haben, sondern immer nur "an" oder "aus", gibt es auch keine Zwischenstufen des Luminanzsignals, der entsprechende Anteil im Signal PICTURE ist damit entweder "vorhanden" oder er ist es nicht.

Wenn Sie einen HF-Modulator verwenden, dann demoduliert die Eingangsstufe des Fernsehers das BAS-Signal aus der Hochfrequenz wieder heraus. Das Ergebnis ist im Prinzip wieder das VIDEO-Signal des Apple - mit einem Unterschied: alle rechteckigen Spannungsverläufe mit einer Frequenz, die größer als rund 1.4 MHz ist, sind zu Sinuswellen geworden. Davon betroffen sind der COLOR REFERENCE BURST und alle PICTURE-Signalanteile höherer Frequenz - speziell die farberzeugenden Signalfolgen. Das so veränderte Signal liegt innerhalb des Fernsehers an den Eingängen des Chroma- und des Luminanzverstärkers sowie an einigen anderen Stellen an: das Ergebnis sind verwaschene Umrisse und ein insgesamt undeutlicheres Bild als mit einem Computermonitor.

Je nach Qualität eines (eventuell farbfähigen) Monitors läuft hier derselbe Prozeß ab: Je geringer die Bandbreite der Verstärker, desto stärker werden die Signale verformt und desto verschwommener werden die Übergänge zwischen einzelnen Bildpunkten. Monochrome Monitoren mit hoher Bandbreite haben keinen Chroma-Verstärker, hier wird das VIDEO-Signal des Apple fast unverformt weitergegeben - außer den bei LoRes und HiRes80 erzeugten Signalen, die auch in einem breitbandigen Monitor verformt werden, solange dessen Frequenzbereich nicht über 21 MHz liegt.

Zurück zur Weiterverarbeitung farbiger Bilder: Nach dem Durchlauf des Luminanzverstärkers gelangt der entsprechende Signalanteil zur Steuerung der Elektronenkanonen, wo er die Helligkeit der einzelnen Bildpunkte steuert. Was die Bildinformation betrifft, kommen wir nach dem Chroma-Verstärker zum wichtigsten Punkt der Verarbeitung: Wenn das PICTURE-Signal mit einer Frequenz von 3.58 MHz schwingt, schaltet es der Chroma-Verstärker weiter zu einem "Synchron-Demodulator", wo seine Phasenlage mit dem aus dem COLOR BURST rekonstruierten Signal COLOR REFERENCE verglichen wird. Über diese Phasenunterschiede wird die Ansteuerung der einzelnen Farbkanonen (rot, blau und gelb) bestimmt. Um eine Farbe zu erzeugen, muß das PICTURE-Signal des Apple also mit 3.58 MHz (oder der doppelten Frequenz) schwingen.

Das wohl stärkste Charakteristikum der Apple-Grafik ist ihre Art, die Farbinformation zusammen mit dem eigentlichen Bild zu speichern - im Gegensatz zu fast allen anderen Computern existiert kein separater Farbspeicher,

---

5 Die internationalen Versionen enthalten zusätzlich in der Rev. B. noch einen Umschalter auf der Hauptplatine, mit dem die Erzeugung des COLOR BURST auch in den Grafik-Modi unterdrückt werden kann, um starkes Flimmern auf monochromen Monitoren (4.44 PAL-Subcarrier zu 3.58 MHz PICTURE) zu verhindern (s. Bild 8.6).

die Farbart wird statt dessen durch die *Lage* eines Punktes innerhalb der Zeile bestimmt. Diese Methode erbringt fantastische Einsparungen in Bezug auf die Menge des benötigten Speichers - die Speicherung einer HiRes-Seite mit sechs Farben auf konventionelle Art würde 24 kByte anstelle von 8 kByte benötigen! Die Kehrseite der Medaille liegt darin, daß die Programmierung farbiger Grafik entsprechend verkompliziert wird.

Vom Apple erzeugte Farben lassen sich in vier Klassen unterteilen, wobei die ersten drei Klassen eigentlich gar keine Farben sind:

1. **Schwarz** wird schlicht durch die Absenz des Luminanz-Signals dargestellt.
2. **Weiß** wird durch alle Signalfolgen von PICTURE erzeugt, die mit niedrigeren Frequenzen als 3.58 MHz schwingen - der Monitor kann dann keine Phasenbeziehung zu COLOR REFERENCE herstellen und behandelt das Signal als "keine Farbe". Zwei aufeinanderfolgende Punkte im Modus HiRes40 halten das VIDEO-Signal für zwei Perioden von COLOR REFERENCE auf demselben Pegel und erscheinen deshalb weiß - vier nebeneinanderliegende Punkte im Modus HiRes80 haben dasselbe Ergebnis. Weitere Möglichkeiten zur Erzeugung der Farbe "Weiß" ist ein LoRes-Block mit COLOR = 15 (sieben aufeinanderfolgende Punkte) sowie TEXT-Zeichen, weil dabei kein COLOR BURST erzeugt wird.
3. **Grau** ist die Farbe, die durch Frequenzen von 7 MHz (genauer: 7.16 MHz) im PICTURE-Signal erzeugt wird. Das ist in LoRes-Blocks der Farben 5 und 10 der Fall sowie in HiRes80, wenn innerhalb einer Zeile jeweils jeder zweite Punkt gesetzt ist. Die LoRes-Farben 5 und 10 sind im Erscheinungsbild identisch und etwas dunkler als weiß, das Signal wechselt ständig zwischen "weiß" und "schwarz" hin und her.
4. **Farbe**: Alle Farbsignale ändern ihren Pegel mit einer Frequenz von 3.58 MHz - der gewählte Grafikmodus spielt dabei keine Rolle. Es sind vier Farben dieser Art in HiRes40 möglich, in den Modi LoRes und Hires80 sind es zwölf. Von diesen zwölf sind vier mit den HiRes40-Farben identisch: Grün, Violett, Orange und Blau. Die restlichen acht Farben sind in jeweils zwei zueinander gehörige Farbschattierungen unterteilbar: Hell- und Dunkelblau, helles und dunkles Magentarot, helles und dunkles Blaugrün, Hell- und Dunkelbraun. Sechs dieser acht Farben können im Modus HiRes40 als Interferenzerscheinung zwischen aufeinanderfolgenden Bytes produziert werden, wenn in einem Byte Bit 7 gesetzt und dieses Bit im darauffolgenden Byte zurückgesetzt ist.

Diese Erklärung der Farberzeugung ist das Ergebnis einer Analyse der Zeitabläufe und der Hardware des Videogenerators. Die Details sind teilweise reichlich kompliziert, aber, oh Leser, so du in meine Fußstapfen trittst, wirst du schon keine kalten Zehen bekommen.

Die Repräsentation der Bildinformation im Speicher

Wie die Bildinformation gespeichert ist (oder andersherum: was für ein Bild von einem gespeicherten Byte erzeugt wird), hängt natürlich vom Videomodus ab. TEXT-Zeichen werden durch ihre ASCII-Werte (mit kleineren Variationen für NORMAL, INVERSE und FLASH) gespeichert, alle Daten für die Grafikmodi dagegen Punkt für Punkt - eine "1" im Byte steht für einen Weißpegel, eine "0" für einen Schwarzpegel in PICTURE. Etwas formaler und detaillierter finden Sie diese Zusammenhänge in Bild 8.4 wiedergegeben, auf das sich auch die folgende Besprechung bezieht.

ASCII-Daten aus dem TEXT-Bildspeicher werden abhängig vom Stand des Softswitches ALTCHRSET auf zwei verschiedene Arten interpretiert. Wenn ALTCHRSET zurückgesetzt ist, stellen die Werte $00-$3F die 64 in INVERSE möglichen Zeichen, $40-$7F den FLASH-Zeichensatz und $80-$FF den normalen ASCII-Satz dar. Wenn ALTCHRSET gesetzt ist, ist die Unterteilung etwas einfacher: $00-$7F ergibt den kompletten ASCII-Zeichensatz INVERSE, $80-$FF den ASCII-Zeichensatz in NORMAL.

Über ALTCHRSET läßt sich also lediglich die Interpretation von ASCII-Werten für INVERSE und FLASH verändern, der NORMAL-Zeichensatz bleibt davon unberührt. Ein kontrollierendes Programm, das direkt auf den Bildspeicher zugreift, muß je nach gewünschtem Erscheinungsbild ALTCHRSET sowie B6 und B7 der darzustellenden Zeichen ändern.

Der Name ALTCHRSET ist eigentlich ein wenig irreführend - manche Leute glauben, damit sei die Auswahl eines anderen Erscheinungsbildes (d.h. anderer Punktmatrizen für die Zeichen) möglich. Auch im Video-ROM ist der NORMAL-Zeichensatz nicht doppelt gespeichert, ALTCHRSET schaltet lediglich den unteren Bereich zwischen FLASH/INVERSE und einem kompletten INVERSE-Zeichensatz um. Wenn Sie mit ALTCHRSET wirklich ein alternatives Punktmuster der einzelnen Zeichen erhalten wollen, bleibt Ihnen natürlich auch hier die Möglichkeit offen, den Video-ROM durch ein selbstprogrammiertes EPROM zu ersetzen.

Im Videomodus TEXT wird jede 40-Byte-Zeile des RAMs vom Videoscanner achtmal hintereinander ausgelesen. Wenn Sie z.B. ein normales "A" auf HTAB 1 geschrieben haben, enthält der Videodatenbus jeweils am Anfang von acht aufeinanderfolgenden Zeilendurchläufen den entsprechenden Code ($C1). Die Elektronik des Videogenerators bekommt über die Ausgänge VA, VB und VC des Videogenerators mitgeteilt, welches der acht Bitmuster adressiert werden muß, aus denen sich ein "A" auf dem Bildschirm zusammensetzt. Anders gesagt: Die 40 Byte einer TEXT-Zeile enthalten die Information für acht aufeinanderfolgende Fernsehzeilen.

Die Arbeitsweise des Videoscanners ist im Modus LoRes dieselbe wie bei TEXT, jede 40-Byte-Zeile wird achtmal hintereinander ausgelesen. Die einzelnen Bytes enthalten allerdings keine ASCII-Codes, sondern die tatsächlichen Bitfolgen für PICTURE, und sind in zwei Hälften unterteilt: für die ersten vier Durchläufe verarbeitet der Videogenerator die Bits 0-3 und erzeugt so die oberen LoRes-Blocks, in den zweiten vier Durchläufen wird mit den Bits 4-7 gearbeitet, aus denen die unteren Blocks einer Zeile erzeugt werden. Zwischen den "ersten vier" und "zweiten vier" wird über VC unterschieden.

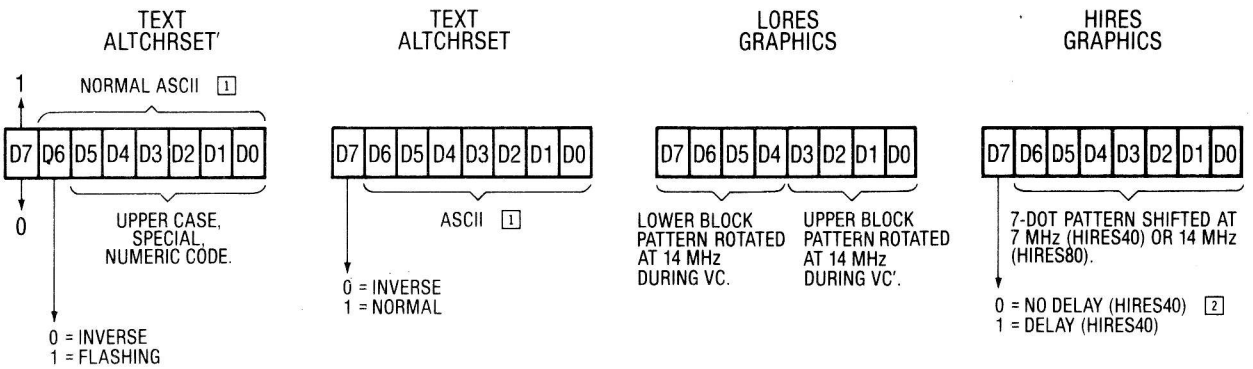
Der Videogenerator schiebt die LoRes-Bitfolgen mit 14 MHz in das PICTURE-Signal hinein, also doppelt so schnell wie bei TEXT40 und HiRes40. Die Frage, wieso der Videoelektronik dabei nicht die Daten ausgehen (4 anstelle von jeweils 8 Bit, und das mit der doppelten Geschwindigkeit), ist leicht beantwortet: jede LoRes-Bitfolge wird vom Videogenerator viermal wiederholt. Die durch diese Operation erzeugten Pegelwechsel im PICTURE-Signal haben eine so hohe Frequenz, daß sie auf einem Farbfernseher nicht mehr klar dargestellt werden können und deshalb zu farbigen Blocks verschwimmen. Wenn Sie einen monochromen Monitor mit hoher Bandbreite verwenden, sind die Punktfolgen dagegen deutlich sichtbar.

Die Bitfolgen für HiRes sind ebenfalls direkt gespeichert, die Bits 0-6 jedes Bytes in den entsprechenden Speicherbereichen ergeben jeweils ein Punktmuster mit 7 Bit. Die Verarbeitung dieser Folgen besteht hauptsächlich aus einem Hineinschieben in PICTURE mit entweder 7 Bit pro Mikrosekunde (HiRes 40) oder 7 Bit pro halbe Mikrosekunde (HiRes 80). Die Bitfolgen werden mit dem niederwertigsten Bit zuerst hinausgeschoben, dieses Bit stellt also den jeweils linkesten Punkt eines Bytes im Bildschirm. Das zwingt manchmal dazu, "verkehrtherum" zu denken - schließlich werden Bits bei Zahlenwerten mit dem höchstwertigen Bit auf der linkesten Position notiert.

Bit 7 jedes "HiRes-Bytes" geht nicht direkt in das PICTURE-Signal ein: wenn dieses Bit im Modus HiRes40 gesetzt ist, werden die darauffolgenden 7 Bit um eine Periode von 14M (d.h. eine viertel Periode von COLOR REFERENCE) verzögert ausgegeben. Damit ergibt sich eine Phasenverschiebung um 90 Grad zu COLOR REFERENCE und konsequenterweise eine Änderung der darstellbaren Farben: Ohne Verzögerung können in HiRes40 Schwarz, Weiß, Grün und Violett dargestellt werden, mit Verzögerung ergeben diese Bitmuster die Farben Schwarz, Weiß, Orange und Blau.

Im Modus HiRes80 wird keine Verzögerung benötigt - durch die höhere Auflösung können nicht nur die "Verzögerungsfarben", sondern auch noch acht weitere Bitmuster dargestellt werden. Das höchstwertige Bit der "Grafik-Bytes" wird in diesem Modus vom Videogenerator ignoriert.

Bild 8.4 Formate der Videodaten in den Bildspeicherbereichen



Anmerkungen:

(1) Für CONTROL ASCII ausgegebene Zeichen entsprechen den normalen Großbuchstaben, d.h. $81 erzeugt ein "A", $82 ein "B" etc.

(2) Im Modus HiRes80 wird B7 ignoriert.

(3) In den Modi TEXT und LoRes wird jedes Byte achtmal für acht aufeinanderfolgende Fernsehzeilen gelesen, in HiRes "hält" jedes Byte nur eine Zeile.

Die Hardware des Videogenerators

Der Videogenerator besteht aus einigen Schaltkreisen, die den Videoscanner, die Softswitches für die Videomodi und die über den Videoscanner aus dem RAM herausgeholten Daten überwachen bzw. verarbeiten. Um daraus das Signal VIDEO zu erzeugen, werden ein beträchtlicher Teil der IOU sowie der Video-ROM, das Schieberegister für PICTURE und ein Summenverstärker benötigt, die zusammen in den Bildern 8.5 und 8.6 gezeigt sind.[6]

Ein Charakteristikum der Bilderzeugung im Apple //e liegt darin, daß ein Zähler existiert, der synchron zur Bewegung des Elektronenstrahls auf dem Bildschirm RAM-Adressen liefert und damit die entsprechenden Videodaten aus dem jeweiligen Bildspeicherbereich "heraustreibt". In Kapitel 5 wurde bereits gezeigt, wie der RAM über die Ausgänge des *Videoscanners* während PHASE1 adressiert wird. Dieses Kapitel geht einen Schritt weiter und zeigt, daß sämtliche mit der Bilderzeugung verbundenen Abläufe in Synchronisation mit dem Videoscanner sind - und damit auch mit dem Elektronenstrahl eines angeschlossenen Monitors. Erreicht wird diese Synchronisation dadurch, daß bereits innerhalb der IOU über den Stand des Videoscanners Signale gebildet werden, die den Videogenerator steuern. Das Erzwingen eines Schwarzpegels, die Zuschaltung von COLOR BURST, die Erzeugung der SYNC-Signale und einige andere Takte werden direkt aus dem Videoscanner abgeleitet.

Ein weiterer Schlüsselpunkt liegt in der Verwendung eines *Video-ROMs*, mit dem Videodaten des Bildspeicherbereichs in Punktmuster umgesetzt werden, die danach als PICTURE-Signal bitweise geschoben werden. Nach jedem halben Prozessorzyklus (d.h. bei jeder Flanke von PHASE0) liegen neue Videodaten an den Adreßeingängen dieses ROMs an. Je nach Bildmodus übersetzt der Video-ROM den Inhalt des Videodatenbusses entweder in Teile einer Zeichenmatrix (TEXT) oder in Punktfolgen für die Darstellung von Grafik. Das Ergebnis der Übersetzung ist an den Datenausgängen des ROMs verfügbar (und wird dort in das Schieberegister geladen). In den Modi mit einfacher Auflösung werden die Videodaten des AUX-RAMs ignoriert: Das Ergebnis sind sieben HiRes-Punkte, eine Zeile eines LoRes-Blocks oder eine Zeile eines TEXT-Zeichens pro Mikrosekunde. In den Modi mit doppelter Auflösung werden in einer Hälfte jedes Zyklus Videodaten des Hauptplatinen-RAMs, in der anderen Hälfte die des AUX-RAMs angenommen. Die Schiebefrequenz beträgt 14 MHz, innerhalb einer Mikrosekunde werden damit 14 HiRes-Punkte, eine Zeile von zwei LoRes-Blocks oder eine Zeile von zwei TEXT-Zeichen ausgegeben.

VID0-VID5 sind direkt mit den Adreßeingängen A3-A8 des Video-ROMs verbunden. VID6 und VID7 werden dagegen zuerst innerhalb der IOU verarbeitet - diese Adreßbits werden für FLASH zyklisch zwischen den Speicheradressen der Punktmuster für NORMAL und INVERSE hin- und hergeschaltet, außerdem spielt hierbei der Stand von ALTCHRSET eine Rolle. Die entsprechenden Ausgänge der IOU haben die Namen RA9 und RA10 (ROM-Adressen 9 und 10), sie sind - wie könnte es auch anders sein - mit den Adreßeingängen 9 und 10 des Video-ROMs verbunden. SEGA, SEGB und SEGC bilden die drei niederwertigsten Adreßbits (A0-A2), GR+2 schaltet A11, über WNDW' von der IOU und ENVID' vom speziellen Steckplatz kann der Video-ROM abgeschaltet werden. Die amerikanische Version der Hauptplatine enthält einen ROM des Typs 2732, die internationalen Versionen dagegen einen mit der doppelten Kapazität: hier kann mit dem an der Unterseite des Gehäuses angebrachten Schalter (ALTCHR') über A12 zwischen dem ASCII-Zeichensatz und dem jeweiligen nationalen Zeichensatz umgeschaltet werden.

Die von der IOU kontrollierten Leitungen zum Video-ROM werden IOU-intern über bestimmte Zustände des Videoscanners geschaltet und um einen oder zwei Taktimpulse des Scanners (steigende Flanke von RAS' während PHASE1) verzögert, um diverse Laufzeiten auszugleichen. SEGC ist z.B. nichts anderes als der Ausgang VC des Videoscanners, aber um einen Scanner-Taktimpuls verzögert. Der Grund für diese Verzögerung liegt hauptsächlich darin, daß zwischen der Adressausgabe des Scanners und dem Gültigwerden der Videodaten die Zugriffszeit der RAMs liegt - durch die Verzögerung kommen Kontrollsignale und von den RAMs ausgegebene Videodaten zur selben Zeit am Video-ROM "an". Wichtig ist diese Koinzidenz speziell bei der LoRes-Grafik: Die während H0 vom ROM ausgegebenen Bitfolgen unterscheiden sich von den während H0' erzeugten.[7]

---

6 In der folgenden Besprechung klammern wir das Thema "PAL-Wandlung" vorläufig aus, die Materie ist kompliziert genug. Details, wie es nach der Erzeugung von PICTURE weitergeht, finden Sie im Abschnitt "Der PAL-Wandler" weiter hinten in diesem Kapitel - (Anm. d. Übers)

7 Im Modus LoRes wird H0 zum Adreßbit SEGA des Video-ROMs, darüber findet die notwendige Korrektur der Phasenlage zu COLOR REFERENCE für ungerade Bildspeicheradressen statt (s. Bilder 3.2 und 8.5). Details dazu im nächsten Abschnitt - (Anm. d. Übers)

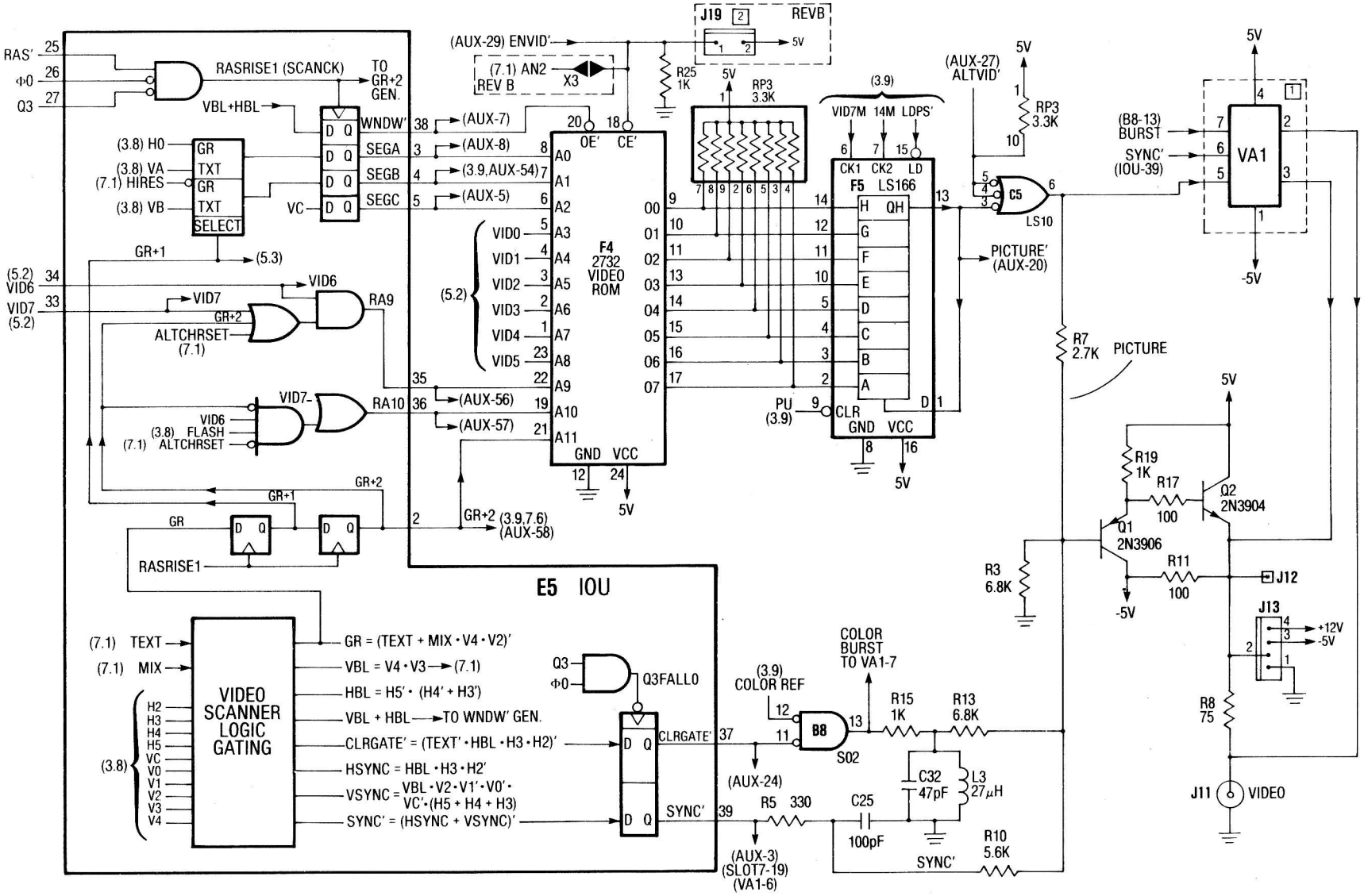


Bild 8.5 Schaltplan des Videogenerators

Anmerkungen:

(1) Verbindungsstecker für einen alternativen Videoverstärker (nur NTSC)

(2) ENVID'-Schaltmöglichkeiten auf Hauptplatinen der NTSC-Rev. B: Zusätzlicher Schalter über J19 oder Lötverbinder X3 und Schaltung unter Programmkontrolle

(3) PAL-Version: 2764 anstelle von 2732, A12 ist über ALTCHR' schaltbar

Die Umschaltungen zwischen GRAPHICS und TEXT in den MIXED-Modi werden um zwei Taktimpulse des Scanners verzögert: damit wird die letzte GRAPHICS-Bitfolge in der unteren rechten Ecke des Grafikbereichs auf dem Bildschirm vollständig aus dem Schieberegister hinausgeschoben, bevor der HAL auf "TEXT-Timing" umgeschaltet wird. Mit dem Modus MIXED werden wir uns weiter hinten in diesem Kapitel noch ausführlicher beschäftigen.

Man könnte sich eigentlich gut vorstellen, daß jeder "Ein-Zeilen-Zyklus" mit dem horizontalen SYNC-Impuls beginnt - im Apple sind die Abläufe aber so, daß der Beginn mit dem ersten HBL-Zyklus zusammenfällt. Er wird dadurch festgelegt, daß der Horizontalzähler des Videoscanners in diesem Moment auf seinen Startwert gesetzt und der Vertikalzähler um eine Position erhöht wird. Signale, die über den Stand des Videoscanners geschaltet werden, ändern ihren Zustand deshalb meistens direkt nach Ende der Ausgabe des sichtbaren Teils einer Fernsehzeile (d.h. am Anfang des rechten Schwarzbereichs auf dem Schirm). Um Verwechslungen mit einer Fernsehzeile auszuschließen, wird dieser Kreislauf aus 65 Zyklen des Prozessors, der mit HBL beginnt, im folgenden *Horizontalperiode* genannt.

Die Eingänge des Video-ROMs

Die Leitung **ENVID'** ist mit Kontakt 29 des speziellen Steckplatzes verbunden und wird normalerweise durch einen "pull-down"-Widerstand auf dem Pegel "0" gehalten. Die Datenausgänge des Video-ROM werden über "pull-up"-Widerstände auf den Pegel "1" gezogen, wenn der Video-ROM inaktiv ist; als Folge bleibt PICTURE' durchgehend auf dem Pegel "1" und das erzeugte Bild ist schwarz. Eine entsprechend konstruierte Karte im speziellen Steckplatz kann damit nicht nur den Video-ROM via ENVID' abschalten, sondern auch das PICTURE'-Signal durch ALTVID' (in invertierter Form) ersetzen.

Die Tatsache, daß der Pegel "1" auf der Leitung PICTURE' für einen schwarzen Bildschirm steht und nicht für weiß, ist schlicht eine hardwaremäßige Vereinfachung. Damit läßt sich nicht nur über einfache "pull-up"-Widerstände die Bildausgabe unterdrücken, auch eine eventuelle ODER-Mischung mit ALTVID' im folgenden Inverter funktioniert problemlos. Als Nebeneffekt ergibt sich, daß die GRAPHICS-Bitfolgen im ROM die Komplemente ihrer Adressen bilden - eine HiRes-Bitfolge von z.B. 1011 adressiert im Video-ROM eine Speicherstelle, deren Inhalt 0100 ist.

Auf Hauptplatinen der NTSC-Rev. B ist ENVID' ebenfalls über den Lötverbinder X2 zum Tastatur-ROM geführt, das Abschalten des Video-ROMs führt damit zur Anwahl der alternativen Tastaturbelegung bzw. umgekehrt (s. Bild 7.4). Weitere Möglichkeiten (ebenfalls nur NTSC-Rev. B) sind die Installation eines Schalters über den Steckverbinder J19 oder die Verbindung von X3, um ENVID' programmgesteuert zu kontrollieren.

WNDW' ("WiNDoW" = Fenster) ist das Signal, mit dem der Schwarzpegel im Videosignal erzwungen wird. Es geht nach einer Verzögerung von einem Scanner-Taktzyklus auf "1", wenn HBL und/oder VBL aktiv werden, und ist über Pin 38 der IOU zum zweiten Aktivierungseingang des Video-ROMs geführt: solange nicht WNDW' und ENVID' beide den Pegel "1" haben, ist der Video-ROM inaktiv und das PICTURE-Signal auf dem Pegel "Schwarz".

Die Signale HBL und VBL existieren nur innerhalb der IOU. Genauer: Ihre Existenz ist ein Postulat meinerseits. Der logische Weg, WNDW' zu erzeugen, führt über eine Kombination aus HBL und VBL, d.h. eine Erzeugung über verschiedene Zustände der Horizontal- und Vertikalsektionen des Videoscanners. Außerdem stimmt das Verhalten von WNDW' exakt mit dem des Signals HBL + VBL BLANKING des alten Apple II+ überein (abgesehen von der Verzögerung um einen Videoscanner-Zyklus). Auf einem Oszillografen wird das sehr deutlich: WNDW' hat 192 Mal denselben Verlauf - 40 Scanner-Zyklen negativ, danach 25 Zyklen positiv. Darauf folgt ein sehr langer Zeitraum, in dem WNDW' durchgehend den Pegel "1" hat, nämlich 7800 Zyklen. Die kurzen Perioden während der aktiven Zeilen entsprechen HBL, die lange Zeit der vertikalen Schwarzperiode.

VID0-VID5 sind, wie bereits gesagt, direkt mit Adreßeingängen des Video-ROMs verbunden. Zusammen mit VID6 und VID7, die erst innerhalb der IOU verändert werden und als RA9 und RA10 wieder erscheinen, werden sie über den ROM in eine Bitfolge für das PICTURE-Signal übersetzt. Generell gesagt besteht die Erzeugung von PICTURE aus der Übersetzung von VID0-VID7 des jeweiligen Bildspeicherbytes. *Wie* diese Übersetzung stattfindet, wird durch die restlichen Eingänge des Video-ROMs (GR+2, SEGA, SEGB und SEGC) bestimmt.

Das in Bild 8.5 gezeigte Signal **GR** hat nur indirekt mit dem Softswitch TEXT/GRAPHICS zu tun: es steht für GRAPHICS TIME und ist in den "ungemischten" Grafikmodi durchgehend aktiv, im Modus MIXED ist es während V4 * V2 zurückgesetzt und schaltet so die "unteren vier Zeilen des Bildes" auf TEXT. V4 * V2 ist insgesamt etwas länger gültig als nur während der letzten 32 Fernsehzeilen vor VBL; derselbe Zustand gilt auch noch während der

letzten 38 Fernsehzeilen von VBL am oberen Rand des Bildschirms. Im Modus MIXED schaltet der Apple also während VBL erst wieder auf GRAPHICS, dann auf TEXT und schließlich am Anfang der obersten dargestellten Fernsehzeile wieder auf GRAPHICS. Da die restlichen Umschaltungen während der vertikalen Schwarzperiode stattfinden, spielt es aber keine allzugroße Rolle.

GR+1 und **GR+2** entsprechen exakt dem Signal GR, nur um eine bzw. um zwei Takte des Videoscanners verzögert. GR+1 wird zur Umschaltung der Adressierung des Videoscanners auf HIRES verwendet und läuft nur IOU-intern (s. Bild 5.3), GR+2 ist zum Video-ROM herausgeführt und mit A11 verbunden. Dieses Adreßbit teilt den Video-ROM in zwei Hälften: TEXT- und GRAPHICS-Bitfolgen.

Auf Hauptplatinen der Rev. A wird GR+2 auch noch in direkter Form als Steuersignal des HALs verwendet und sperrt dort die Takterzeugung für doppelte Auflösung, während GRAPHICS aktiv ist. Seit der Rev. B wird GR+2 invertiert und über FRCTXT' gesteuert, bevor es zum HAL gelangt. Solange FRCTXT' den Pegel "0" hat (was bei zurückgesetztem AN3 und einer erweiterten 80-Zeichen-Karte im speziellen Steckplatz immer der Fall ist), wird der Eingang GR+2' des HAL auf den Pegel "1" gesetzt (d.h. gesperrt). Damit bleibt der HAL auch während GRAPHICS im Modus "doppelte Auflösung", wenn 80COL gesetzt ist.

SEGA, SEGB und **SEGC** sind die niederwertigsten Adreßbits des Video-ROMs. Abhängig davon, ob TEXT oder GRAPHICS gesetzt ist (über GR+1 bestimmt), ist die Erzeugung dieser Signale unterschiedlich (s. Tabelle 8.1). In beiden Fällen werden sie IOU-intern um einen Scanner-Zyklus verzögert, bevor sie zum Video-ROM gelangen.

Bei der Ausgabe von TEXT werden SEGA, SEGB und SEGC direkt aus den Zählbits VA, VB und VC des Videoscanners abgeleitet und bestimmen, welcher Teil der 7 * 8-Matrix eines TEXT-Zeichens adressiert wird. Im Modus TEXT/LoRes werden VA-VC nicht zur RAM-Adressierung verwendet, statt dessen wird jede Zeile achtmal hintereinander ausgelesen, VA-VC adressieren jedesmal einen anderen Teil der Zeichenmatrix im Video-ROM. Da SEGA, SEGB und SEGC mit den niederwertigsten Adreßbits des Video-ROMs verbunden sind, finden sich die acht Teilstücke einer Zeichenmatrix auf aufeinanderfolgenden Adressen, wenn man den ROM über einen EPROM-Programmierer oder ein ähnliches Gerät ausliest.[8]

Während GRAPHICS ist eine "Unter-Adressierung" via VA-VC nicht notwendig - im Modus LoRes kommen wir allerdings ohne weitere Informationen auch hier nicht aus. VC wird nach wie vor gebraucht, um zu unterscheiden, ob der obere oder der untere Block einer Buchstabenposition ausgegeben werden soll. Der über LORES und VC' adressierbare Bereich (d.h. alle über "A2 = 0" erreichbaren Adressen) des Video-ROMs enthält deshalb die Bitfolgen für VID0-VID3, die Adressen für VC ("A2 = 1") enthalten die Bitfolgen für VID4-VID7.

Ein Beispiel: Wenn das Videodatum den Wert 10111000 hat (oberer Block mit COLOR = 12, unterer mit COLOR = 8), dann liefert der Video-ROM bei LORES, VC' und H0' den Wert 01110111, also die Inversion der Bitfolge von VID0-VID3 zweimal hintereinander. Während LORES, VC und H0' (also nach dem Wechsel von VC) wird dagegen der Wert 01000100 geliefert (die Inversion von VID4-VID7). Im weiteren Verlauf dieses Kapitels wird gezeigt, wie das Schieberegister diese Werte mit 14 MHz rotiert[9] und so eine LoRes-Bitfolge insgesamt dreieinhalb Mal hintereinander während eines LoRes40-Zyklus erzeugt.

1. **TEXT-Scanning mit TEXT-Ausgabe** (VA-VC nicht in der RAM-Adressierung, aber für die Video-ROM-Adressierung verwendet, GR+1 inaktiv);
2. **TEXT-Scanning mit LoRES-Ausgabe** (VA-VC nicht in der RAM-Adressierung verwendet, VC als SEGC zur Unterscheidung "oberer/unterer Block", VA und VB nicht benötigt, weil dieselbe Bitfolge jeweils viermal wiederholt wird, GR+1 ist aktiv, HIRES' inaktiv);
3. **HIRES-Scanning und -Ausgabe** (VA-VC für die RAM-Adressierung verwendet, VC im Video-ROM ignoriert, die Bitfolgen existieren doppelt, VA und VB nicht benötigt, GR+1 aktiv, HIRES' aktiv) - (Anm. d. Übers.)

---

8 Bitte nicht verwirren lassen: Für die Arbeitsweisen von Videoscanner und Videogenerator sind drei Kombinationen möglich.

9 Der Terminus "rotieren" steht hier für denselben Ablauf, wie er innerhalb einer Speicherstelle durch einen Rotationsbefehl des 6502 hervorgerufen wird - die Bits wandern im Kreis.

Tabelle 8.1 Umschaltungsmöglichkeiten von SEGA-SEGC[10]

	SEGA	SEGB	SEGC
GR+1' GR+1	VA H0	VB HIRES'	VC VC

Für LoRes wird H0 als Adreßeingang des Video-ROMs benötigt. Wie Sie sich von Kapitel 3 her erinnern, ändert sich die Phasenlage von COLOR REFERENCE mit Beginn jedes CPU- bzw. Videoscanner-Zyklus, und diese Beziehung kann über H0 definiert werden (Bild 3.2). Damit ist die erzeugte Farbe davon abhängig, ob sich die gerade gelesene (Bild-)Speicherstelle auf einer geraden oder auf einer ungeraden Adresse innerhalb der momentanen Zeile befindet, falls keine Korrektur erfolgt.

Im Modus HIRES ist das tatsächlich der Fall - die durch eine Bitfolge erzeugte Farbe hängt tatsächlich von der Position innerhalb der Zeile ab. Die Korrektur über H0 im Modus LoRes sieht eigentlich recht einfach aus (nachdem man erst einmal daraufgekommen ist): Im über "H0 = "1"" adressierbaren Bereich des Video-ROMs befinden sich die Bitfolgen für LoRes - aber um zwei Bit verschoben. Um unser bereits benutztes Beispiel fortzusetzen: Das Videodatum 10111000 erzeugt während VC' und H0' die Bitfolge 11101110 (VID0-VID3 invertiert, zweimal hintereinander), während H0 dagegen die Folge 11011101 (VID0-VID3 invertiert, zweimal hintereinander und um zwei Bit nach rechts rotiert). Dieser "Offset" von 2 Bit versetzt die Folge exakt um einen halben Zyklus von COLOR REFERENCE.

Das dritte während GRAPHICS umgeschaltete Adreßbit des Video-ROMs ist HIRES', die Inversion des Softswitches HIRES. Über dieses Bit wird innerhalb des ROMs zwischen LoRes und HiRes unterschieden. Für HiRes-Bitfolgen haben H0 und VC keinen Einfluß: Der über "HIRES' = "0"" adressierbare Bereich des ROMs enthält viermal dieselben Bitfolgen, alle Bereiche, die sich über die möglichen Kombinationen von VC und H0 zusammen mit HIRES' adressieren lassen, enthalten dieselben Daten.

Der Ausgang SEGB der IOU ist zusätzlich noch mit einem Eingang des HAL verbunden, er wird zur Unterscheidung zwischen HiRes und LoRes benutzt, wenn GR+2' aktiv ist ("GRAPHICS Time"). Das ist deshalb notwendig, weil der HAL(!) den Zeitablauf der Bilderzeugung um eine Periode von 14M verzögert, wenn im Modus HIRES40 das Bit 7 eines Videodatums gesetzt ist.

Die für HiRes im Video-ROM gespeicherten Bitfolgen sind eine einfache Inversion ihrer Adressen, d.h. der Bitfolge, die sich aus VID0-VID6 zusammensetzt. Der Zustand von VID7 hat keinen Einfluß auf die Bitfolge, dieses Bit bestimmt in keinem der HiRes-Modi den Zustand eines Bildpunkts, die über VID7 und VID7' adressierbaren Bitfolgen des ROMs sind identisch. Außerdem wird in den HiRes-Modi das höchstwertige vom Video-ROM ausgegebene Bit nicht benutzt, es ist in den von Apple, Inc. gelieferten ROMs durchgehend auf "1" gesetzt.

Die verbleibenden Adreßbits des Video-ROMs sind **RA9** und **RA10**. In ähnlicher Weise wie SEGA-SEGC wird ihre Funktion dadurch bestimmt, ob TEXT oder GRAPHICS ausgegeben werden soll. Wenn GR+2 aktiv ist und so GRAPHICS anzeigt oder wenn TEXT ausgegeben werden soll und ALTCHRSET gesetzt ist, erhalten RA9 und RA10 (nach der IOU-internen Laufzeit) dieselben Werte wie VID6 und VID7. Während GRAPHICS gehen die Videodaten damit "durch" zum ROM und werden dort invertiert, wie in den vorhergehenden Abschnitten beschrieben.

Tabelle 8.2 Zuordnungsmöglichkeiten für RA9 und RA10

GR+2	ALTCHRSET	RA10	RA9
1	X	VID7	VID6
0	0	VID7 + VID6 • FLASH	VID6 • VID7
0	1	VID7	VID6

Tabelle 8.2 zeigt die Zuordnungen von RA9 und RA10 innerhalb der IOU. Wie zu sehen ist, sind auch im Modus TEXT (d.h. bei GR+2 = "0") RA9 und RA10 mit Vid6 und Vid7 identisch - es sein denn, ALTCHRSET ist gesetzt. Die spezielle Logik für diesen Fall ist für FLASH und die veränderte Aufteilung der "unteren" 128 ASCII-Codes

10 SEGA-SEGC werden IOU-intern vor der Ausgabe um einen Scanner-Taktzyklus verzögert.

notwendig. Tabelle 8.3 zeigt sämtliche Kombinationsmöglichkeiten von ALTCHRSET, VID6 und VID7 und die sich daraus ergebenden Zustände von RA9 und RA10 im TEXT-Modus.

Tabelle 8.3 Mögliche Zustände von RA9 und RA10 im TEXT-Modus

ALTCHRSET	VID7	VID6	RA10	RA9	CHARACTERS
0	0	0	0	0	INVERSE control/special
0	0	1	FLASH	0	Flash INVERSE/NORMAL*
0	1	0	1	0	NORMAL control/special
0	1	1	1	1	NORMAL upper/lower
1	0	0	0	0	INVERSE control/special
1	0	1	0	1	INVERSE upper/lower
1	1	0	1	0	NORMAL control/special
1	1	1	1	1	NORMAL upper/lower

** Alterniert zwischen NORMAL Control/Special und INVERSE Control/Special*

Die Logik der Erzeugung für RA9 und RA10 bei gesetztem ALTCHRSET ist vorhanden, wenn auch nicht gerade offensichtlich. Bei einer Erklärung fangen wir am besten ganz "vorne" an: Jedes TEXT-Zeichen ist im Bildspeicher des Apple mit einem Byte gespeichert, theoretisch sind also 256 verschiedene Zeichen (und damit 256 Kombinationen von VID0-VID7) möglich. Der ASCII-Zeichensatz kommt aber mit 128 Zeichen aus, um die Groß- und Kleinbuchstaben des Alphabets, Ziffern, Sonder- und Steuerzeichen zu definieren. Der TEXT-Bildspeicher verfügt damit über die Möglichkeit der Darstellung zweier kompletter und voneinander verschiedener Zeichensätze. Zusätzlich erhält der Programmierer über den Softswitch ALTCHRSET die Wahl zwischen zwei verschiedenen Interpretationen der gespeicherten Codes.

Der alte Apple II(+) war (und ist, solange es sich nicht um einen der "Klau-Kompatiblen" mit diversen Erweiterungen handelt) nur zur Darstellung von 64 Zeichen in der Lage, für Kleinbuchstaben war kein Platz mehr: die beiden höchstwertigen Bits wurden vom Videogenerator zur Unterscheidung zwischen NORMAL, INVERSE und FLASH benutzt. Die TEXT-Ausgabe des Apple //e ist kompatibel mit der des Apple II+, solange ALTCHRSET zurückgesetzt ist - abgesehen davon, daß nunmehr der komplette ASCII-Zeichensatz inklusive Kleinbuchstaben dargestellt werden kann.

Wenn ALTCHRSET dagegen gesetzt ist, werden die TEXT-Videodaten als 128 NORMALe und 128 INVERSE Zeichen interpretiert (und nicht mehr als 128 Zeichen NORMAL, 64 INVERSE und 64 FLASH). Der im Bildspeicher stehende Code für NORMAL und INVERSE unterscheidet sich in diesem Fall nur durch ein gesetztes bzw. zurückgesetztes höchstwertiges Bit (VID7).

Tabelle 8.4 zeigt die Zusammenhänge zwischen den im ROM gespeicherten Matrizen und dem ASCII-Wert des Bildspeichers. Die Anordnung für ALTCHRSET sieht erheblich logischer und vernünftiger aus als die für FLASH/INVERSE. Tatsächlich ist der TEXT-Bereich des Video-ROMs exakt so angeordnet wie der ALTCHRSET-Teil in der Tabelle - RA9, RA10 und VID5 unterteilen diesen Bereich folgendermaßen:

RA10	RA09	VID5	CHARACTER SET
0	0	0	INVERSE control (upper)
0	0	1	INVERSE special
0	1	0	INVERSE upper
0	1	1	INVERSE lower
1	0	0	NORMAL control (upper)
1	0	1	NORMAL special
1	1	0	NORMAL upper
1	1	1	NORMAL lower

Im Video-ROM sind keine speziellen Bitfolgen für FLASH gespeichert. Für Videodaten mit VID7-VID6 = "01" schaltet die IOU alle 16 vertikalen Bilddurchläufe die Erzeugung von RA9 und RA10 zwischen "10" und "00" hin und her. Damit wird abwechselnd "NORMAL Control/Special" und "INVERSE Control/Special" im Video-ROM adressiert, wenn ALTCHRSET zurückgesetzt ist. Das zur Umschaltung benutzte Signal FLASH ändert seinen Zustand 3.125 Mal pro Sekunde, FLASH-Zeichen blinken also mit rund 0.3 Hz.

Bei zurückgesetztem ALTCHRSET müssen die (Teil-)Zeichensätze INVERSE und FLASH anders codiert sein als NORMAL. Bedingt dadurch, daß sich hier drei Zeichensätze innerhalb von 256 Möglichkeiten drängeln, ist es mit dem einfachen Umschalten eines einzigen Bits nicht mehr getan: VID5 ist bei INVERSE oder FLASH für Großbuchstaben "1" und für Sonderzeichen "0", bei NORMAL ist es genau umgekehrt. Programme, die mit FLASH und/oder INVERSE arbeiten wollen, tun gut daran, vor Beginn ALTCHRSET auf einen definierten Stand zu bringen. Die 40-Zeichen-Firmware des Apple //e erzeugt leider Unsinn, wenn Sie versuchen, mit gesetztem ALTCHRSET INVERSE Kleinbuchstaben zu schreiben, sie ist nämlich vom II+ übernommen und auch im verbesserten Apple in diesem Punkt nicht korrigiert bzw. um einen Test von ALTCHRSET erweitert worden. Um diese Zeichen auch ohne einen direkten Zugriff auf den Bildspeicher korrekt ausgeben zu können, müssen Sie die 80-Zeichen-Routinen benutzen. Falls Sie dagegen FLASH benutzen wollen, müssen Sie ALTCHRSET zurücksetzen und die 40-Zeichen-Routinen benutzen.

Laden und Schieben von Bitmustern

Während jeder Mikrosekunde, die zwischen zwei steigenden Flanken von PHASE0 vergeht, finden zwei Zugriffe (jeweils 500 ns) auf den Video-ROM statt. Kurz nach der steigenden Flanke von PHASE0 werden die Videodaten vom AUX-RAM gültig und bleiben es bis kurz nach der fallenden Flanke von PHASE0 - zu diesem Zeitpunkt werden sie durch die Videodaten des Hauptplatinen-RAMs ersetzt. Das als Reaktion auf diese Daten vom Video-ROM ausgegebene Bitmuster benötigt maximal 500 ns, um an den Ausgängen gültig zu werden (wenn wir davon ausgehen, daß Apple, Inc. auch hier einen ROM mit 450ns Zugriffszeit verwendet). Damit ist das durch Videodaten vom AUX-RAM am Ausgang des Video-ROMs erzeugte Bitmuster gegen Ende von PHASE0 gültig, Bitmuster für den Hauptplatinen-RAM sind es gegen Ende von PHASE1.

Die Ausgänge des Video-ROMs sind mit den Paralleleingängen eines Lade-/Schieberegisters (LS166) verbunden, das dieses Bitmuster lädt und Bit für Bit hinausschiebt. Geladen werden die Muster, wenn LDPS' gegen Ende von PHASE1 auf "0" geht ("Hauptplatinen-Video"), bei doppelter Auflösung wird LDPS' auch noch gegen Ende von PHASE0 aktiv und lädt durch Daten des AUX-RAMs erzeugte Muster.

Wenn der LS166 nicht gerade mit einem Ladevorgang beschäftigt ist, dann schiebt er die geladenen Daten. Im Apple //e ist er so geschaltet, daß die Daten bitweise im Kreis herumlaufen, d.h. jedes hinausgeschobene Bit wird nicht nur von der folgenden Elektronik als PICTURE' interpretiert, sondern auch gleichzeitig wieder auf die vorderste Bitposition des Registers eingeladen. Diese Rotation ist allerdings nur für den Modus LoRes40 wichtig - in allen anderen Fällen wird das Schieberegister neu geladen, bevor dieselben Bits ein zweites Mal am Ausgang erscheinen.

Außer LDPS' wird das Schieberegister noch von zwei weiteren Signalen kontrolliert, nämlich VID7M und 14M. 14M liefert die Taktimpulse für Laden und Schieben und wird seinerseits über VID7M (leicht verzögert) geschaltet. Wenn VID7M den Pegel "0" hat, dann führt jede steigende Flanke von 14M entweder zu einem Lade- oder zu einem Schiebevorgang. In den 80er-Modi sowie im Modus LoRes40 setzt der Taktgenerator VID7M durchgehend auf "0", die Verarbeitung von Bitmustern geschieht dadurch mit 14 MHz. In den Modi TEXT40 und HiRes40 ist VID7M nur bei jeder zweiten steigenden Flanke von 14M auf "0", die Bitmuster werden deshalb mit 7 MHz geschoben.

Eine Rückmeldung über "einfache Auflösung" oder etwas ähnliches gibt es nicht: Auch in den 40er Modi erscheinen an den Eingängen des Schieberegisters jeweils 500 ns nach den Bitmustern der Hauptplatine die des AUX-RAMs (vorausgesetzt, daß eine 80-Zeichen-Karte installiert ist, ansonsten erscheinen die Hauptplatinen-Bitmuster ein zweites Mal). Der Video-ROM liefert also auch hier zwei Bitmuster pro Mikrosekunde, allerdings aktiviert der Taktgenerator LDPS' nur am Ende von PHASE1, und die AUX-Bitmuster werden ignoriert. In den folgenden Abschnitten werden wir uns noch detaillierter mit den Taktsignalen und Zeitabläufen der Bilderzeugung auseinandersetzen.

Tabelle 8.4 Dargestellte TEXT-Zeichen

ALTCHRSET'

	INVERSE				FLASH				NORMAL							
	UPPER		SPECIAL		UPPER		SPECIAL		CONTROL		SPECIAL		UPPER		LOWER	
ASCII	000 $00	016 $10	032 $20	048 $30	064 $40	080 $50	096 $60	112 $70	128 $80	144 $90	160 $A0	176 $B0	192 $C0	208 $D0	224 $E0	240 $F0
0 $0	@	P		0	@	P		0	@	P		0	@	P	`	p
1 $1	A	Q	!	1	A	Q	!	1	A	Q	!	1	A	Q	a	q
2 $2	B	R	"	2	B	R	"	2	B	R	"	2	B	R	b	r
3 $3	C	S	#	3	C	S	#	3	C	S	#	3	C	S	c	s
4 $4	D	T	$	4	D	T	$	4	D	T	$	4	D	T	d	t
5 $5	E	U	%	5	E	U	%	5	E	U	%	5	E	U	e	u
6 $6	F	V	&	6	F	V	&	6	F	V	&	6	F	V	f	v
7 $7	G	W	'	7	G	W	'	7	G	W	'	7	G	W	g	w
8 $8	H	X	(	8	H	X	(	8	H	X	(	8	H	X	h	x
9 $9	I	Y	)	9	I	Y	)	9	I	Y	)	9	I	Y	i	y
10 $A	J	Z	*	:	J	Z	*	:	J	Z	*	:	J	Z	j	z
11 $B	K	[	+	;	K	[	+	;	K	[	+	;	K	[	k	{
12 $C	L	\	,	<	L	\	,	<	L	\	,	<	L	\	l	|
13 $D	M	]	-	=	M	]	-	=	M	]	-	=	M	]	m	}
14 $E	N	^	.	>	N	^	.	>	N	^	.	>	N	^	n	~
15 $F	O	_	/	?	O	_	/	?	O	_	/	?	O	_	o	▒

ALTCHRSET

	INVERSE								NORMAL							
	UPPER		SPECIAL		UPPER		LOWER		CONTROL		SPECIAL		UPPER		LOWER	
ASCII	000 $00	016 $10	032 $20	048 $30	064 $40	080 $50	096 $60	112 $70	128 $80	144 $90	160 $A0	176 $B0	192 $C0	208 $D0	224 $E0	240 $F0
0 $0	@	P		0	@	P	`	p	@	P		0	@	P	`	p
1 $1	A	Q	!	1	A	Q	a	q	A	Q	!	1	A	Q	a	q
2 $2	B	R	"	2	B	R	b	r	B	R	"	2	B	R	b	r
3 $3	C	S	#	3	C	S	c	s	C	S	#	3	C	S	c	s
4 $4	D	T	$	4	D	T	d	t	D	T	$	4	D	T	d	t
5 $5	E	U	%	5	E	U	e	u	E	U	%	5	E	U	e	u
6 $6	F	V	&	6	F	V	f	v	F	V	&	6	F	V	f	v
7 $7	G	W	'	7	G	W	g	w	G	W	'	7	G	W	g	w
8 $8	H	X	(	8	H	X	h	x	H	X	(	8	H	X	h	x
9 $9	I	Y	)	9	I	Y	i	y	I	Y	)	9	I	Y	i	y
10 $A	J	Z	*	:	J	Z	j	z	J	Z	*	:	J	Z	j	z
11 $B	K	[	+	;	K	[	k	{	K	[	+	;	K	[	k	{
12 $C	L	\	,	<	L	\	l	|	L	\	,	<	L	\	l	|
13 $D	M	]	-	=	M	]	m	}	M	]	-	=	M	]	m	}
14 $E	N	^	.	>	N	^	n	~	N	^	.	>	N	^	n	~
15 $F	O	_	/	?	O	_	o	▒	O	_	/	?	O	_	o	▒

Hinweis: Der Video-ROM des "verbesserten" Apple //e enthält für den Bereich $40-$5F "Mauszeichen" (s. Kapitel 6 und Bild 8.8)

Der Ausgang QH des Schieberegisters ist mit der Leitung PICTURE' verbunden, d.h. PICTURE' steigt und fällt in Abhängigkeit des rotierten Bitmusters. Wenn QH den Pegel "0" hat oder ALTVID' vom speziellen Steckplatz auf "0" gesetzt wird, geht das Signal PICTURE auf "1", die entsprechende Stelle auf dem Bildschirm wird hell. Das niederwertigste vom Video-ROM gelieferte Bit ist mit dem niederwertigsten Eingang des Schieberegisters verbunden und wird nach der Ladeaktion als erstes hinausgeschoben. Anders gesagt: *Der Video-ROM enthält invertierte Bitmuster, deren niederwertigstes Bit vor den anderen auf dem Bildschirm erscheint.*

In der NTSC-Version ist es von diesem Punkt bis zur Video-Ausgangsbuchse nicht mehr weit: PICTURE' führt über einen Inverter und einen Widerstand zum Video-Summenverstärker. Die drei Eingänge dieses Verstärkers sind "gewichtet", d.h. die Eingangswiderstände sind so bemessen, daß sich darüber das korrekte Spannungsverhältnis zwischen PICTURE', SYNC' und COLOR BURST' ergibt. Alle drei Signale werden in der ersten Stufe des Summenverstärkers miteinander gemischt. Der Ausgang des Verstärkers führt zu J12, J13, und über einen Widerstand (Kurzschluß-Schutz) schließlich zu J11, der Videobuchse auf der Gehäuserückseite.

Bevor wir im nächsten Abschnitt die Details der PAL-Version besprechen, noch ein paar weitere Gemeinsamkeiten beider Hauptplatinen:

SYNC' ist ein direkter Ausgang der IOU und vom Zählstand des Videoscanners abgeleitet. Dieses Signal ist eine Kombination aus den horizontalen SYNC-Impulsen (vier Scanner-Taktzyklen in der Mitte von HBL) und dem vertikalen SYNC (ein Impuls von vier Fernsehzeilen Länge in der Mitte von VBL). Für die NTSC-Version gibt Bild 8.2 den horizontalen und den vertikalen SYNC detailgetreu wieder, Bild 5.9 zeigt, welche Speicherstellen während SYNC angesprochen werden. Für beide Versionen gilt die in Bild 8.5 gegebene Zustandsgleichung.

COLOR BURST' ist ein 14 Zyklen langer Ausschnitt von COLOR REFERENCE und wird durch **CLRGATE'** geschaltet. Dieses Signal kommt von der IOU und wird direkt nach jedem horizontalen SYNC-Impuls aktiv, wenn der Softswitch TEXT zurückgesetzt ist (s. Bild 8.2). Im Modus TEXT wird CLRGATE niemals aktiv, als Folge davon enthält das VIDEO-Signal keinen COLOR BURST, und die TEXT-Zeichen haben keine unerwünschten farbigen Ränder. Bei der Mischung von Grafik und TEXT (MIXED gesetzt) ist das allerdings nicht der Fall - hier wird CLRGATE auch während der vier TEXT-Zeilen aktiv. Das Ergebnis ist schön bunt, aber schwer zu lesen.

Die Pegelwechsel von SYNC' und CLRGATE' finden nicht direkt nach einem Taktimpuls des Videoscanners statt - diese beiden Signale scheinen IOU-intern über die fallende Flanke von Q3 während PHASE0 geschaltet zu sein. Ich schätze, daß diese Verzögerung zur Kompensation von produktionsabhängigen Streuungen (Laufzeiten etc.) und zur Unterdrückung von Störspitzen durch Schaltflanken eingebaut worden ist. Diese Spekulation hat eine solide Grundlage - schließlich erzeugt in den ersten Ausgaben des Apple II eine Schaltspitze auf der Leitung für den vertikalen SYNC eine sichtbare schwarze Linie links im Bildschirm - in späteren Ausgaben ist dieses Problem durch eine Verzögerung von SYNC gelöst.

Die Videoausgabe im europäischen Apple //e[11]

Die Unterschiede zwischen dem amerikanischen Apple und seiner Exportversion liegen hauptsächlich im Videoscanner und in der Erzeugung des Signals VIDEO. Wenn die Fernsehsysteme einzelner Länder nicht inkompatibel zueinander wären, bestände eine Anpassung schlicht aus dem Einbau eines anderen Netzteils und dem Austausch von Tastatur- und Video-ROM.

Die traurige Wahrheit sieht natürlich etwas anders aus: Es gibt verschiedene Fernsehnormen in der Welt, die miteinander zu mehr oder weniger großen Teilen kompatibel sind. In den meisten Fällen beginnt die Unverträglichkeit mit der Bildfrequenz: die NTSC-Norm legt einen Bildschirm mit 525 Zeilen fest, der in zwei Halbbilder zu jeweils einer sechzigstel Sekunde aufgeteilt ist. Um einen Apple mit einem PAL-Fernseher zu betreiben, muß deshalb die 60 Hz-IOU gegen eine 50 Hz-Ausführung getauscht werden.

In dieser Version der IOU wird der Vertikalzähler des Videoscanners nicht von 011111010, sondern von 011001000 hochgezählt. Damit ergeben sich 50 zusätzliche Fernsehzeilen pro (Halb-)Bild, insgesamt sind es dann 312. Sämtliche zusätzlichen Zeilen finden während VBL statt, VBL ist damit 120 Fernsehzeilen lang (und nicht 70). Außerdem ist die Lage des vertikalen SYNC innerhalb von VBL so verändert, daß die oberen und unteren Schwarzbereiche des Fernsehbildes exakt gleich groß sind. Der horizontale Scan-Prozeß und die Erzeugung des horizontalen SYNC laufen in beiden IOU-Versionen identisch ab.

---

11 Die meisten Details der PAL-Version (IOU-Preset, Zeilenzahl, Bildfrequenz etc.) wurden von mir in den laufenden Text eingearbeitet. Der hier folgende Abschnitt wurde aus Gründen der Vollständigkeit unverändert gelassen, auch wenn er damit teilweise eine Wiederholung darstellt - (Anm. d. Übers.).

Die Zustandsgleichung für den vertikalen SYNC in der PAL-Version ist VBL * V5' * V2' * V0' * VC' * (H5 + H4 + H3), im Gegensatz zur amerikanischen Ausführung, die mit VBL * V2 * V1' * V0' * VC' * (H5 + H4 + H3) arbeitet. Der vertikale SYNC selber besteht aus einem Impuls von vier Fernsehzeilen Länge und ist damit in beiden Versionen derselbe. Wo wir gerade beim Vergleichen sind: 60 Hz-IOUs haben 32 Fernsehzeilen innerhalb von VBL, dann folgt der SYNC (4 Zeilen) und weitere 34 VBL-Zeilen am oberen Bildschirmrand; europäische Apples haben 72 Zeilen innerhalb von VBL, danach den SYNC und weitere 44 Zeilen Schwarzperiode.

Tabelle 8.5 Unterschiede zwischen 60 Hz- und 50 Hz-Versionen der IOU

	START VBL V543210CBA	VERTICAL SYNC V543210CBA	PRESET ON OVERFLOW V543210CBA	VERTICAL SYNC V543210CBA	END VBL V543210CBA
American	111000000	1111000XX	011111010	---	100000000
European	111000000	---	011001000	011010 0XX	100000000

Tabelle 8.5 faßt die Unterschiede in den kritischen Umschaltpunkten von VA-V5 beider Versionen zusammen. Alte Apple-Freaks werden dabei feststellen, daß der Scan-Prozeß des //e für 60 Hz identisch mit der amerikanischen RFI-Version des Apple II ist. Für die 50 Hz-Version des //e und die europäische Version ("Europlus") des II+ gilt dasselbe - schließlich ist der "Europlus" nichts weiter als die amerikanische RFI-Version mit anders gesetzten Lötverbindern.

Die Feinabstimmung der europäischen Apples wird durch einen Austausch des Quartzes erreicht: 14M hat nicht mehr eine Frequenz von 14.31818, sondern entweder von 14.250450 (diskreter Oszillator) oder von 14.25 MHz (hybrider Oszillator und 50 Hz-//c). Damit kommt der Computer der geforderten Länge einer Fernsehzeile (64 Mikrosekunden) recht nahe.

Die Änderung der Frequenz von 14M zieht auch eine leichte Veränderung der Arbeitsgeschwindigkeit des Prozessors nach sich, er arbeitet in Europa mit 1.016 anstelle von 1.205 MHz durchschnittlicher Taktfrequenz. Diese Reduzierung scheint den Betrieb der Diskettenlaufwerke nicht unsicherer zu machen - selbst zeitkritisch "kopiergeschützte" Disketten, die in Amerika hergestellt worden sind, lassen sich in Europa problemlos lesen.

COLOR REFERENCE ist natürlich von dieser Herabsetzung auch betroffen und geht von 3.58 MHz auf 3.56 MHz herunter. Das spielt nur deshalb keine Rolle, weil dieses Signal im PAL-System sowieso nicht als Farbträger verwendet wird.

Der zweite Hauptpunkt, in dem sich die Fernsehnormen nicht miteinander vertragen, liegt in der Übergabe von Farbinformationen. Es gibt hier drei unterschiedliche Prinzipien): NTSC, PAL ("Phase Alternating Lines" = Phasenumkehr von Zeile zu Zeile) und SECAM ("Sequentiell a Memoire" = zeitlich aufeinanderfolgend mit Speicherung). Es gibt keine spezielle Ausführung des Apple //e für SECAM, der Computer muß in den entsprechenden Ländern über RGB-Adapter und entsprechende Monitoren benutzt werden. Speziell Frankreich ist davon betroffen - inzwischen haben sich französische Computerläden deshalb PAL-Monitoren zugelegt...

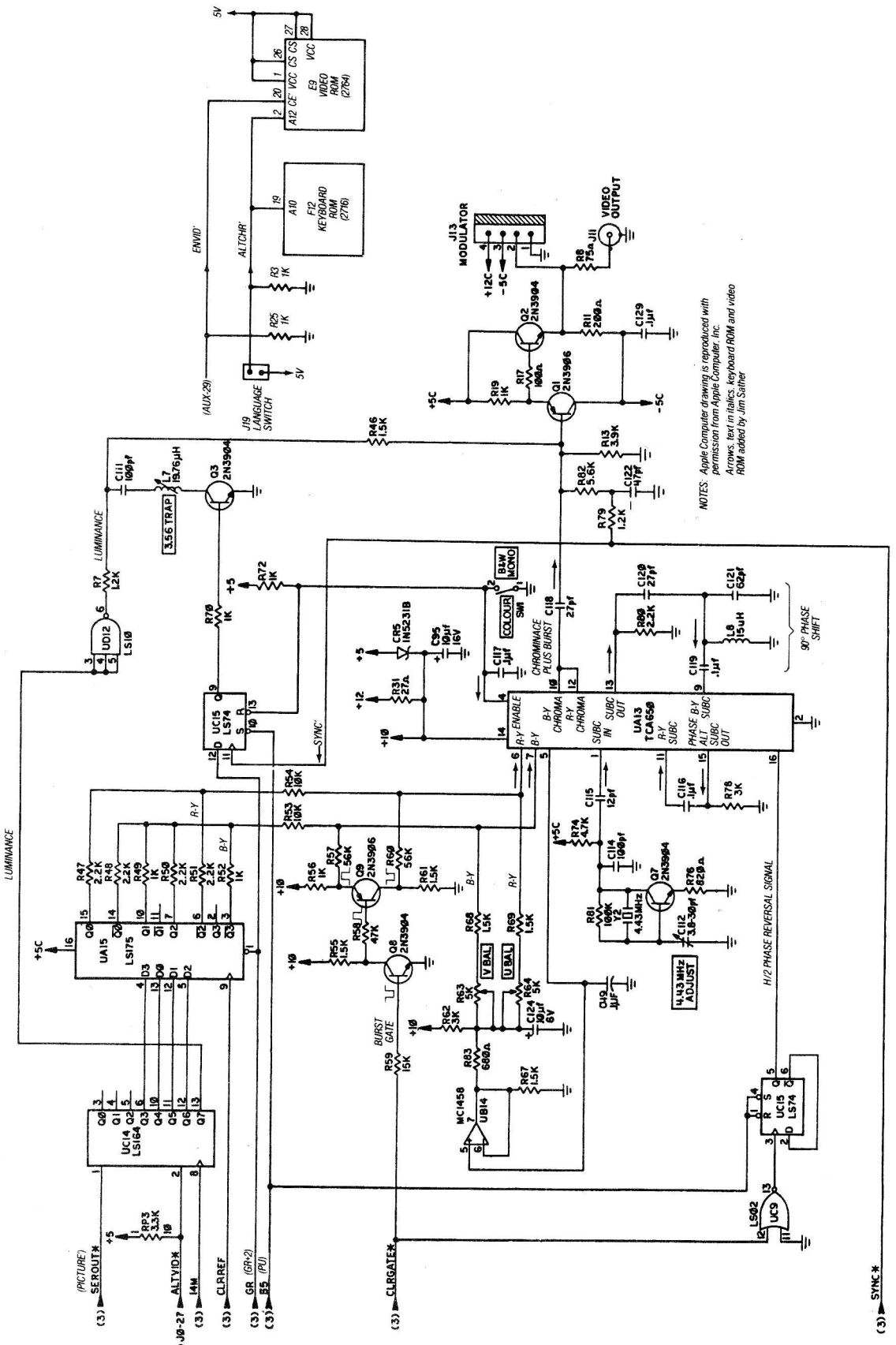
Bild 8.6 gibt den Schaltplan des PAL-Wandlers wieder, der sich als zusätzliche Baugruppe auf den Hauptplatinen der euröpäischen Apples befindet. Dieser Schaltplan ist von Apple, Inc. übernommen und mit deren freundlicher Genehmigung hier abgebildet. Die Anmerkungen in Schrägschrift sowie die Zeichnung von Tastatur- und Video-ROM stammen von mir.

Die Schaltkreise sind fast identisch mit der "Eurocolor"-Zusatzkarte für den II(+), die PAL-Version des Apple //e besteht also eigentlich aus einer NTSC-Hauptplatine mit einem 14.25 MHz-Oszillator, einer 50 Hz-IOU, entsprechend angepaßten ROMs für Video und Tastatur sowie einer integrierten Eurocolor-Karte.

Die PAL-Norm verwendet einen Farbträger von 4.43619 MHz (anstelle der 3.579545 MHz von NTSC), und der COLOR BURST von PAL ändert seine Phasenlage bei jeder Zeile um 90 Grad.[12] Fernseher und Monitoren, die für PAL eingerichtet sind, können deshalb aus einer mit 3.56 MHz alternierenden Bitfolge, die mit COLOR BURSTs von 3.56 MHz durchsetzt ist, nicht automatisch ein farbiges Bild erzeugen. Damit das funktioniert, holt der PAL-Wandler sich die Farbinformation aus dem PICTURE-Signal wieder heraus, indem er jeweils Stückchen von vier Bit "einsammelt" und ihre Phasenlage mit COLOR REFERENCE vergleicht.

12 Wenn man sich das so recht überlegt, dann wäre die Konstruktion des Apple für ein PAL-System erheblich einfacher: der 65. ("lange") Zyklus könnte mitsamt seiner aufwendigen Logik komplett entfallen - er ist schließlich nur da, weil sich sonst die Phasenlage von COLOR BURST mit jeder Fernsehzeile um 180 Grad ändern würde. Aber wie das Leben und die Wirtschaft so spielen, ist es offensichtlich billiger, erst ein NTSC-Signal zusammenzusetzen und es hinterher auf PAL "umzustricken" - (Anm. d. Übers.).

Bild 8.6 Schaltplan des PAL-Wandlers (europäische Version der Hauptplatine)



Aus diesen Informationen wird ein PAL-Signal erzeugt, das aus den Komponenten Luminanzsignal, modulierter 4.44 MHz-Farbträger, alternierender COLOR BURST mit 4.44 MHz und den SYNC-Signalen besteht. Die Bitmuster von PICTURE' (in Bild 8.6: SEROUT') werden mit 14 MHz in ein Gegenstück zum LS166 hineingeschoben: der Baustein LS144 ist ein Schieberegister mit seriellem Eingang und einem parallelen Ausgang. Dort werden sie als Folgen von jeweils 4 Bit gesammelt - soviele Bits werden maximal während einer Periode von COLOR REFERENCE (80er Modi und LoRes) ausgegeben. Mit jeder steigenden Flanke von COLOR REFERENCE wird ein neues 4-Bit-Stück im darauffolgenden Vierfach-Latch (LS175) festgehalten. Da der Haltezeitpunkt durch COLOR REFERENCE bestimmt wird, entsprechen die festgehaltenen Bits der momentanen Phasenlage von PICTURE zu COLOR REFERENCE, sprich der durch dieses Signalstück erzeugten Farbe.

Dieses "4-Bit-Farbwort" wird über Summierwiderstände in Gleichspannungen umgewandelt, die den "gewichteten" Signalen R-Y (Rot minus Luminanz) und B-Y (Blau minus Luminanz) entsprechen. Aus diesen beiden Signalen besteht die Farbträgermodulation des PAL-Standards. Sie werden mit den Spannungen für die BURST-Durchschaltung und die Balance summiert und den R-Y- und B-Y-Eingängen eines TCA650 zugeführt.

Der TCA650 ist ein synchroner *Demodulator* für PAL und SECAM - im Apple //e wird er für die umgekehrte Funktion benutzt und moduliert einen Farbträger konstanter Phase mit B-Y sowie R-Y, bevor er aus beiden modulierten Signalen ein Chroma-Signal bildet. Das ist alles andere als der vom Hersteller vorgesehene Zweck dieses Bausteins - es ist statt dessen eine wirklich erstaunliche Leistung von Gary Baker, der die Eurocolor-Karte entworfen hat. Bevor ich jetzt Verwirrung stifte, indem ich darauf bestehe, daß der TCA650 ein Demodulator ist, bezeichne ich ihn im folgenden auch als Modulator, dessen Modulationseingänge die Pins 6 und 7 sind, die Trägerfrequenz wird über 11 und 9 zugeführt.

Die Frequenz für den Farbträger (4.43619 MHz) wird von einem eigenen Oszillator erzeugt und dem TCA über Pin 1 eingespeist. Dieses Signal erscheint mit gleicher Amplitude an den Ausgängen 13 und 15. Das (phasenkonstante) Ausgangssignal von Pin 13 wird über ein Netzwerk mit einer Phasenverschiebung von exakt 90 Grad zu Pin 9 zurückgeführt und dort als zu modulierender Farbträger für B-Y wieder eingespeist. Das Ausgangssignal von Pin 15 ist dagegen nicht phasenkonstant, seine Lage wird durch den Schalteingang von Pin 16 kontrolliert. Dieser Eingang ist mit dem Ausgang eines Flipflops verbunden, das auf jede fallende Flanke von CLRGATE' mit einer Umschaltung reagiert - das Ausgangssignal von Pin 15 ändert also seine Phasenlage mit jeder neuen Fernsehzeile. Es ist zu Pin 11 weitergeführt, dem zu modulierenden Farbträger für R-Y.

Der um 90 Grad phasenverschobene und über B-Y modulierte und der phasenalternierende und über R-Y modulierte Farbträger werden an den Pins 10 und 12 des TCA 650 miteinander zum Chroma-/BURST-Signal kombiniert. Der BURST wird einfach dadurch erzeugt, daß die Spannungswerte von B-Y und R-Y durch ein aktives CLRGATE' auf entgegengesetzte Polarität geschaltet werden. Die Modulationsbalance des TCA650 ist so eingestellt, daß der Baustein kein Chroma-Signal erzeugt, wenn CLRGATE' den Pegel "1" hat und das 4-Bit-Farbwort den Wert 1111. Wenn CLRGATE' auf "0" fällt, erscheint für 4 Mikrosekunden ein Ausschnitt ("Wellenpaket") des 4.44 MHz-Farbträgers an den Pins 10 und 12 des TCA650 als Folge davon, daß der Pegel von B-Y gestiegen und der von R-Y zum selben Zeitpunkt gefallen ist. Die Kombination aus dem um 180 Grad phasengeschalteten und dem um 90 Grad verschobenen Farbträger mit gleicher Amplitude erzeugt einen COLOR BURST, dessen Phase für jede neue Fernsehzeile um 90 Grad verschoben ist.

Die serielle Bitfolge von PICTURE' wird über das Schieberegister LS164 verzögert (Ausgang Q7) und danach invertiert. Sie wird mit SYNC' und dem Chroma-/BURST-Signal zum Ausgangssignal PAL VIDEO summiert, wobei auch hier innerhalb des Ausgangsverstärkers die Mischung und eine weitere Invertierung stattfinden. Der Signalzweig für das Luminanz-Signal enthält eine geschaltete Falle, mit der während GRAPHICS 3.56 MHz-Anteile abgefangen werden, wenn der Schalter COLOUR/MONO aus COLOUR steht. Notwendig ist das deshalb, weil 3.56 MHz noch in die Bandbreite des Chroma-Verstärkers eines PAL-Monitors fallen und damit Farbinterferenzen erzeugen.

Durch die Entfernung der 3.56 MHz-Anteile aus dem Signal verschwimmen die entsprechenden Bitfolgen auf dem Bildschirm: größere Farbflächen erscheinen dadurch zwar dichter und einheitlicher, die hochauflösende grafische Darstellung auf einem monochromen Monitor ist aber fast nicht mehr möglich, weil das Bild zu stark flimmert. Deshalb sollte dieser Schalter immer auf MONO gesetzt werden, wenn der Apple mit einem monochromen Monitor betrieben wird. Dadurch wird nicht nur die 3.56 MHz-Falle abgeschaltet, sondern auch das Chroma-/BURST-Signal des TCA650.

Abgesehen von der integrierten PAL-Karte enthält ein europäischer Apple noch einige andere Kleinigkeiten, auf die die amerikanische Ausführung und ihre Besitzer mit Recht neidisch sein können. Der Video-ROM hat 28 Pins anstelle von 24 und die doppelte Kapazität (8 anstelle von 4 kByte), zwischen den beiden Hälften von 4 kByte läßt

sich mit einem simplen Schalter (ALTCHR) an der Gehäuseunterseite umschalten. Über ALTCHR wird auch der Tastatur-ROM und damit die Tastaturbelegung vom amerikanischen auf den jeweiligen nationalen Standard umgeschaltet.[13]

Es gibt einige Länder wie z.B. Kanada, die das NTSC-System verwenden und trotzdem mehrere Zeichensätze benötigen. Um diesem Bedarf nachzukommen, hat Apple, Inc. einen kleinen Adapter entwickelt, mit dem ein Video-ROM von 8 kByte in die Fassung einer NTSC-Hauptplatine gesteckt werden kann. Im Normalfall landet ein 2764 EPROM in dieser Fassung - wenn größere Stückzahlen zu erwarten sind, wird ein eigener maskenprogrammierter ROM verwendet.

Dieses Prinzip ließe sich natürlich fast endlos erweitern - wie wäre es mit zwei 2764 in einem Adapter mit 28 Pins und vier wählbaren Zeichensätzen (ASCII, deutsch, englisch und französisch?)

Die Softswitches für den Videomodus

Die Form der Bilderzeugung des Apple //e wird durch den Stand von programmierbaren Softswitches innerhalb der IOU bestimmt. Die Funktionen dieser Schalter finden Sie in Tabelle 8.6 zusammengefaßt, einen (zumindest logisch äquivalenten) Schaltplan ihrer hardwaremäßigen Ausführung in Bild 7.1.

Die drei grundlegenden Videomodi TEXT, LoRes und HiRes werden durch die beiden Softswitches **TEXT** (GRAPHICS') und **HIRES** (LORES') bestimmt. Zusätzlich sind über den Softswitch **MIXED** Mischformen von Grafik und TEXT möglich. Wenn TEXT zurückgesetzt und MIXED gesetzt ist, wird auf dem Bildschirm Grafik mit vier Zeilen TEXT dargestellt.

PAGE2 schaltet zwischen der "Seite 1" ($400-$7FF bzw. $2000-$3FFF) und der "Seite 2" ($800-$BFF bzw. $4000-$5FFF) um, wenn **80STORE** zurückgesetzt ist. Wenn 80STORE gesetzt ist, kann über PAGE2 in den Bildspeicherbereichen zwischen dem RAM der Hauptplatine und dem AUX-RAM geschaltet werden, wie in Kapitel 5 beschrieben.

Tabelle 8.6 Funktionen der Softswitches für den Bildmodus

Softswitch	Rücksetz.	Setzen	Lesen	Funktion
80STORE*	W$C000	W$C001	R$C018	Bild-Bereichsumsch. PAGE2 aus
80COL	W$C00C	W$C00D	R$C01F	DOUBLE-RES Timing bei TEXT oder AN3'
ALTCHRSET	W$C00E	W$C00F	R$C01E	kompletter INVERSE-Zeichensatz
TEXT***	$C050	$C051	R$C01A	TEXT-Modus
MIXED***	$C052	$C053	R$C01B	bei TEXT': Grafik/TEXT gemischt
PAGE2*	$C054	$C055	R$C01C	Bildausgabe von "Seite 2" bei 80STORE'
HIRES*	$C056	$C057	R$C01D	HIRES während GRAPHICS
AN3*	$C05E	$C05F	N/A	DOUBLE-RES GRAPHICS Timing (AN3')

Anmerkungen:

* PAGE2, HIRES und 80STORE existieren sowohl innerhalb der IOU $als auch in der MMU und werden zusammen gesetzt/rückgesetzt. $Die MMU gibt den Stand von 80STORE aus, wenn $C018 gelesen $wird, die IOU gibt den Stand von PAGE2 oder HIRES aus, wenn C01C bzw. $C01D gelesen wird.

** AN3 ist über eine Steckbrücke auf "erweiterten" 80-Zeichen-Karten (64 k RAM) mit FRCTXT' (Rev. A: ENFIRM') verbunden. Wenn AN3 auf "0" gesetzt ist, wird dadurch GATED GR+2' am Eingang des HAL auf "1" gesetzt, der HAL bleibt dann auch während GRAPHICS im Modus "doppelte Auflösung", wenn 80COL' aktiv ist.

*** Alle in dieser Tabelle aufgeführten Softswitches außer TEXT und MIXED werden durch ein RESET automatisch zurückgesetzt.

13 Das kann allerdings nicht nur Neid erregen: Über ALTCHR wird ein guter Teil der Sonderzeichen ebenfalls umgeschaltet, die Ergebnisse sind mehr als katastrophal (Komma, Doppelpunkt, Schrägstrich, Fragezeichen, Ä\Ü, Doppelkreuz und Circumflex sind plötzlich woanders). Apple, Inc. hat daraus die Konsequenzen gezogen und bietet den "enhanced" //e mit einer neuen Tastaturbelegung an, bei der durch ALTCHR "nur noch" 6 Zeichen umgeschaltet werden - (Anm. d. Übers.).

Der Softswitch **80COL** hat nur eine einzige Funktion: er wird innerhalb der IOU invertiert, als 80COL' ausgegeben und schaltet das Timing des HAL auf "doppelte Auflösung", wenn GR+2 den Pegel "0" hat (Rev. A) bzw. GATED GR+2' inaktiv ist (Rev. B). Anders gesagt: "doppelte Auflösung" ist dann gewählt, wenn 80COL' entweder während der TEXT-Ausgabe oder erzwungenermaßen (durch FRCTXT') den Pegel "0" hat. Auf Hauptplatinen der Rev. B. mit einer "erweiterten 80-Zeichen-Karte", bei der die Verbindung "DOUBLE-RES" gesteckt ist, wird **FRCTEXT'** durch das Zurücksetzen von **AN3** auf den Pegel "0" gebracht und setzt so GR+2' auch während GRAPHICS inaktiv. Auf Hauptplatinen der Rev. A existiert die Leitung FRCTXT' nicht, doppelt hoch auflösende Grafik ist deshalb nicht möglich.

Nachdem wir in den vorhergehenden Abschnitten bereits die Hardware des Videogenerators besprochen haben, können wir nun einmal vollständig auflisten, was die Softswitches für die Videomodi alles kontrollieren:

1. Die RAM-Adressierung - darüber wird bestimmt, welchen Speicherbereich der Videoscanner während PHASE1 anspricht.
2. Die Adressierung des Video-ROMs - damit wird festgelegt, in welche Bitfolgen für PICTURE die Videodaten des momentan aktiven Bildspeicherbereichs übersetzt werden.
3. Die Takterzeugung bzw. den HAL - hierüber wird einfache (7 MHz Schiebefrequenz) oder doppelte Auflösung (14 MHz Schiebefrequenz) gesetzt.

Tabelle 8.7 gibt die Wirkungen der entsprechenden Softswitches noch einmal im Detail wieder. Damit Sie nicht vollständig in dieser grauen Theorieflut untergehen, folgen hier ein paar Programmbeispiele für das Setzen der einzelnen Videomodi des Apple //e:

```
a) TEXT 40, Seite 1
     LDA  $C051          TEXT
     LDA  $C054          PAGE2 aus
     STA  $C00C          80COL aus (-> einf. Auflösung)

b) TEXT80, Seite 2
     LDA  $C051          TEXT
     LDA  $C055          PAGE2
     STA  $C00D          80COL (-> doppelte Auflösung)
     STA  $C000          80STORE aus (-> PAGE2 setzt "Seite 2")

c) HIRES40 mit TEXT40 gemischt, Seite 1
     LDA  $C050          TEXT aus (GRAPHICS an)
     LDA  $C053          MIXED
     LDA  $C054          PAGE2 aus
     LDA  $C057          HIRES
     LDA  $C05F          AN3 an (HiRes verzögert)
     STA  $C00C          80COL aus (-> einf. Auflösung)

d) HIRES80, Seite 1, NOMIX
     LDA  $C050          TEXT aus (GRAPHICS an)
     LDA  $C052          MIXED aus
     LDA  $C057          HIRES
     LDA  $C05E          AN3 aus (-> doppelte Aufl. bei GRAPHICS)
     STA  $C001          80STORE (PAGE2 setzt MAIN/AUX)
     STA  $C00D          80COL (-> doppelte Aufl. an)

e) LORES40, gemischt mit TEXT80, Seite 2
     LDA  $C050          TEXT aus (GRAPHICS an)
     LDA  $C053          MIXED
     LDA  $C055          PAGE2
     LDA  $C056          HIRES aus (LORES an)
     LDA  $C05F          AN3 an (-> einf. Aufl. bei GRAPHICS)
     STA  $C000          80STORE aus (PAGE2 setzt "Seite 2")
     STA  $C00D          80COL (-> doppelte Aufl. an)
```

Bild 8.9 Erzeugung und Ausgabe von LoRes

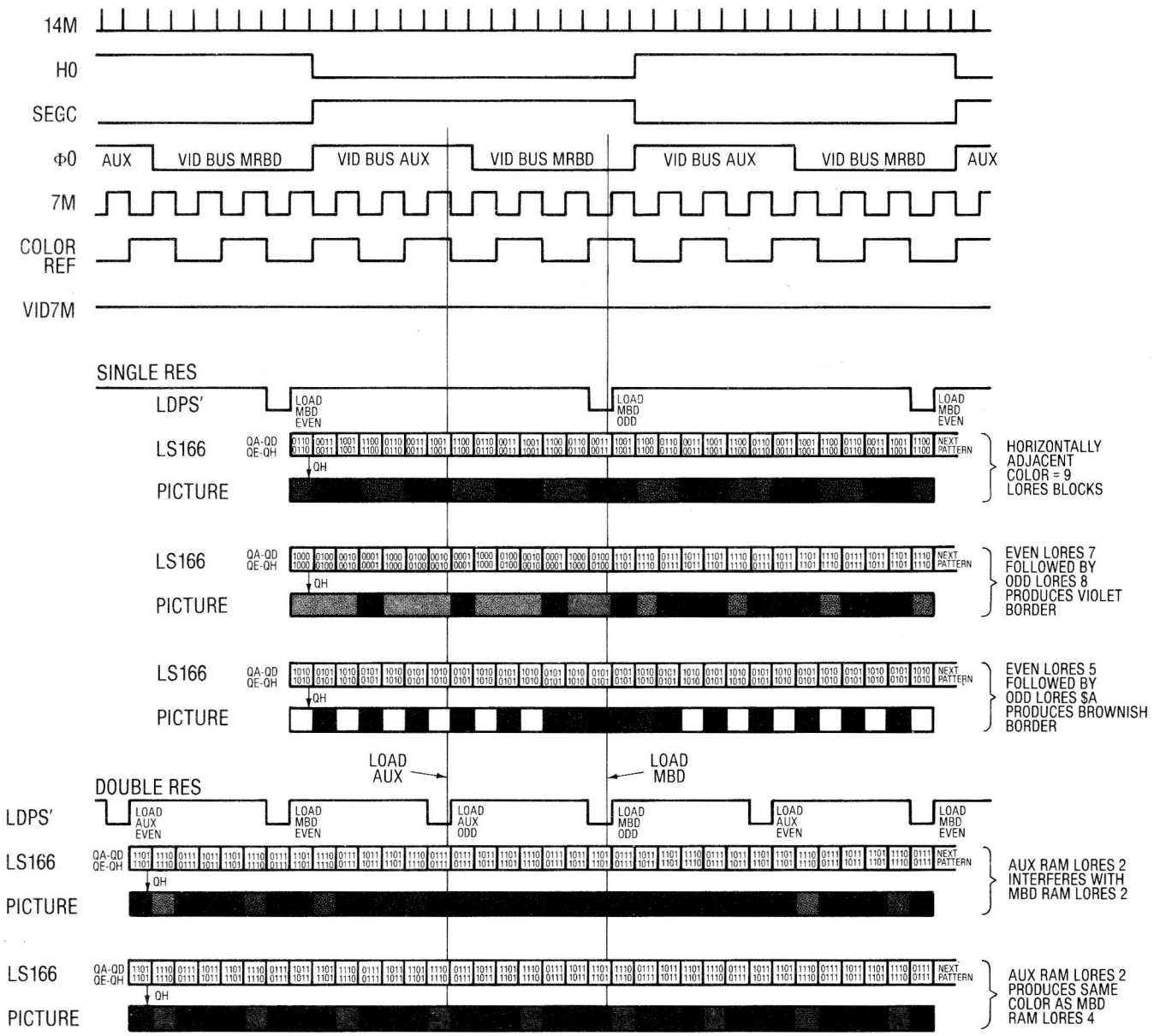


Bild 8.13 Erzeugung und Ausgabe von HiRes

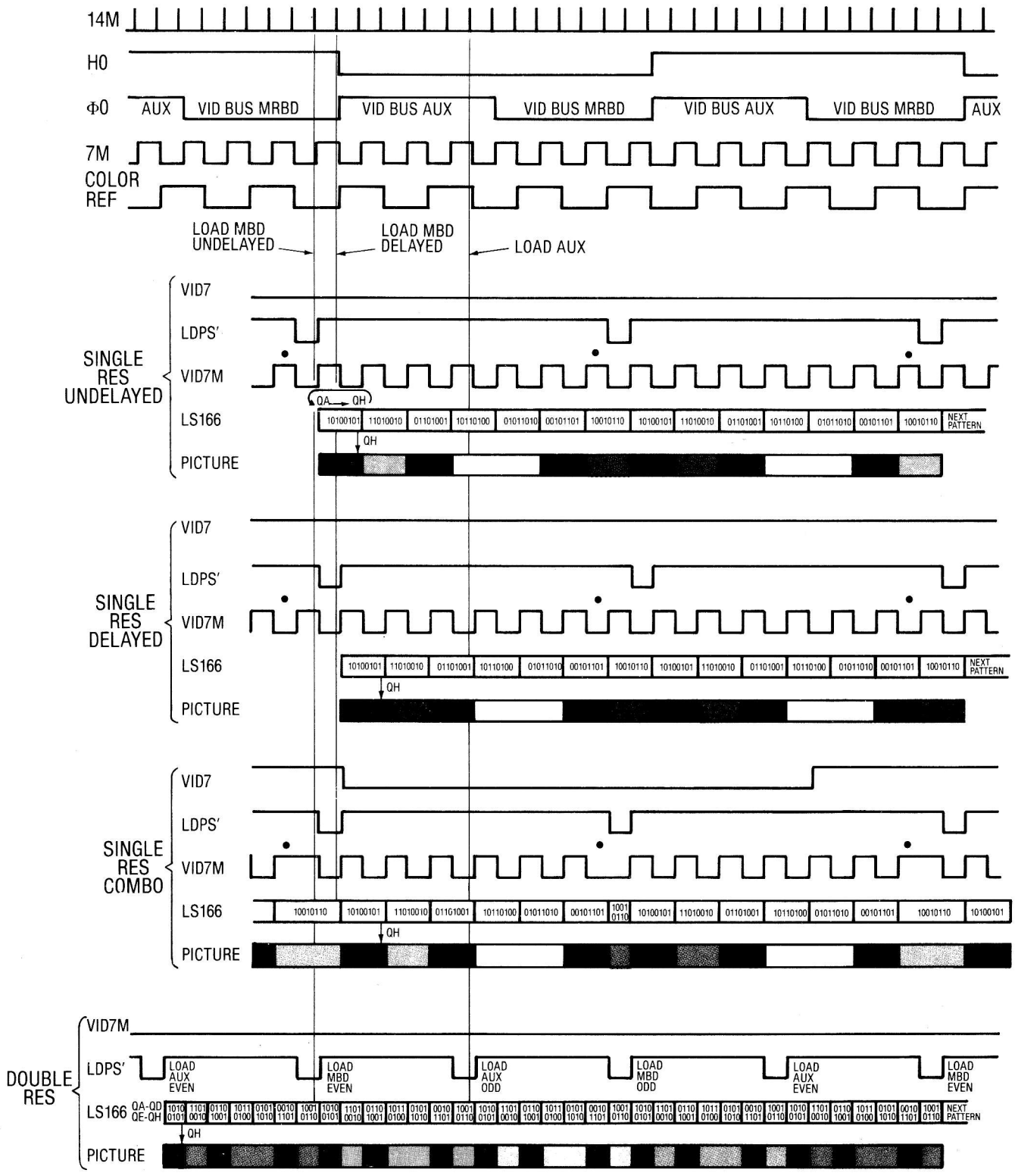


Tabelle 8.7 Wirkungen der Softswitches für die Videomodi

Softswitch gesetzt	Softswitch zurückgesetzt	Wirkung im gesetzten Zustand
TEXT	GRAPHICS	Videoscanner bearbeitet $400-$BFF (Bild 5.3) Segmentierte Adressierung im Video-ROM (Bild 8.5) VID6, VID7 bestimmen INVERSE, NORMAL und FLASH (8.5) COLOR BURST' via CLRGATE abgeschaltet (Bild 8.5 und 8.6) DOUBLE-RES-Timing möglich (Bild 3.9)
MIXED	NOMIX	Umschaltung des Videoscanners auf die Erzeugung von TEXT-Adressen vor den unteren vier Zeilen des Bildschirms (Bild 5.3) Adreßumschaltung des Video-ROMs zum selben Zeitpunkt (Bild 8.5) DOUBLE-RES-Timing zum selben Zeitpunkt auch ohne AN3' möglich (Bild 3.9)
PAGE2	PAGE1	Videoscanner erzeugt Adressen im Bereich $800-$BFF bei TEXT/LORES und 80STORE' (Bild 5.3) Videoscanner erzeugt Adressen im Bereich $4000-$5FFF bei HIRES und 80STORE' (Bild 5.3) Zugriff auf AUX-RAM $400-$7FF bei 80STORE (Bild 5.13b) Zugriff auf AUX-RAM $2000-$3FFF bei 80STORE und HIRES (Bild 5.13b)
HIRES	LORES	Videoscanner erzeugt Adressen im Bereich $2000-$5FFF während GR+1 (Bild 5.3) Video-ROM auf HIRES-Adressen während GR+1 (Bild 8.5) VID7 wird bei einfacher Auflösung und GR+2 als Ver zögerungsbit benutzt (Bild 3.9) PAGE2 schaltet zwischen $2000-$3FFF MAIN und AUX, wenn 80STORE gesetzt ist (Bild 5.13b)
FRCTXT'	FRCTXT	DOUBLE-RES-Timing auch während GRAPHICS, wenn 80COL gesetzt ist (Bild 3.9) Timing für einfache Auflösung (7 Mhz ohne Verzögerung durch VID7), wenn 80COL zurückgesetzt ist (Bild 3.9)
80STORE	80STORE'	Sperrung der Umschaltfunktion von PAGE2 zwischen "Seite 1" und "Seite 2" (Bild 5.3) Entsperrung der Umschaltfunktion von PAGE2 zwischen MAIN und AUX (Bild 5.13b)
80COL	40COL	Aktivierung der Takterzeugung für DOUBLE-RES TEXT oder "erzwungene" DOUBLE-RES (via FRCTEXT') (Bild 3.9)
ALTCHR	NRMCHR	Einschalten der Adressierung von inversen Zeichenmatrizen im Video-ROM auch für Kleinbuchstaben (Bild 8.5)

Die Zeitsignale der Bilderzeugung

Die Bilderzeugung wird hauptsächlich von zwei Taktsignalen bestimmt: **LDPS'** ("LoaD Parallel in, Serial out register" = Laden des Schieberegisters mit parallelen Daten) und **VID7M** (VIDeo 7 MHz). Beide Signale sind Ausgänge des HAL und wurden in Kapitel 3 oberflächlich besprochen. Eine detailliertere Beschreibung folgt erst an dieser Stelle, weil beide Signale einzig und allein für die Erzeugung von PICTURE verwendet werden.

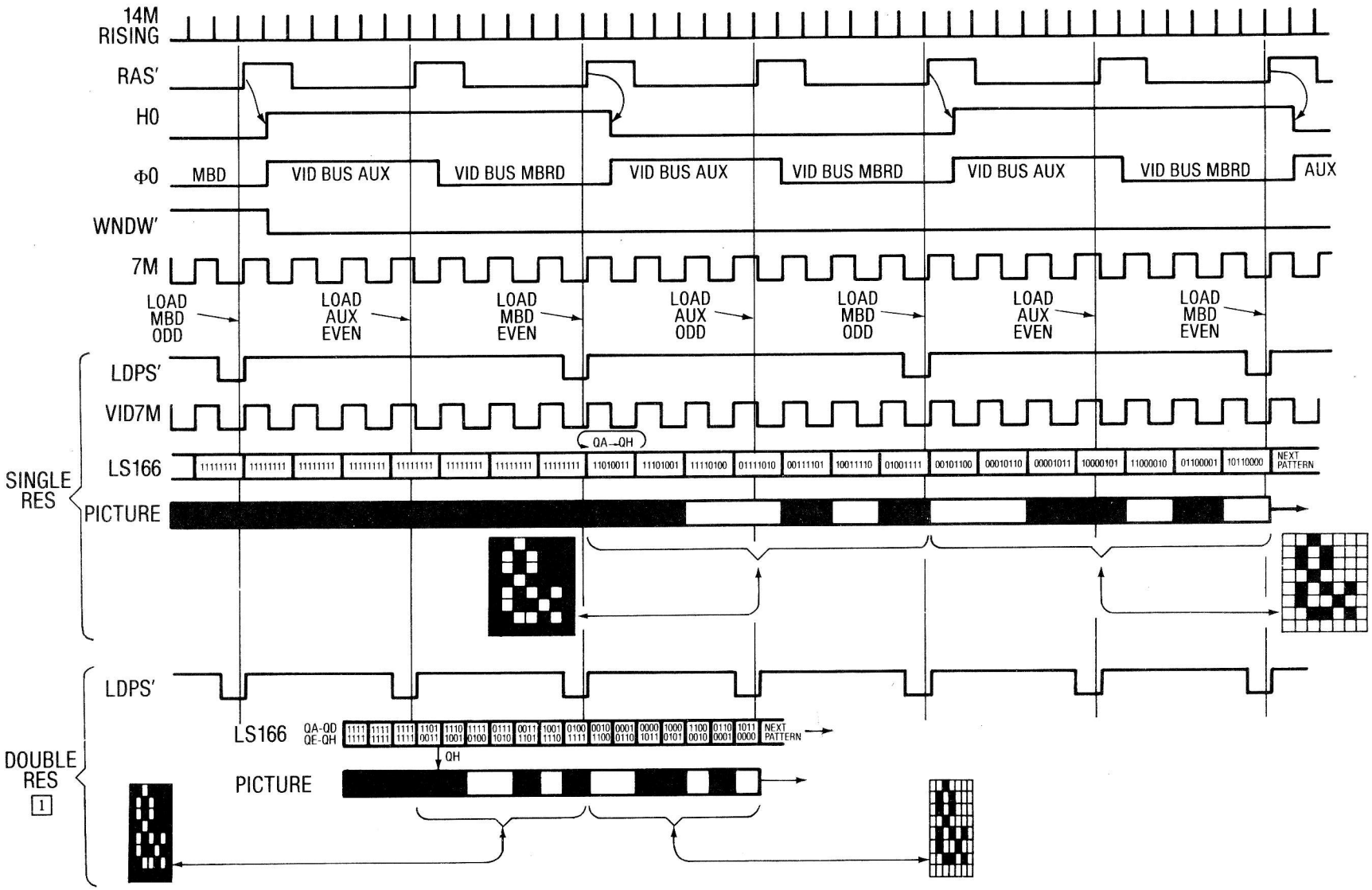
Wie bereits in den letzten Abschnitten gesagt, ist LDPS' das Kontrollsignal für "Laden/Schieben", über VID7M wird die eigentliche Schiebefrequenz (14M) geschaltet. Wenn VID7M den Pegel "0" hat, führt jede steigende Flanke von 14M abhängig von LDPS' entweder zu einer Ladeaktion neuer Daten des Video-ROMs oder zu einer Schiebeaktion bereits geladener Daten.

Wie in Tabelle 8.8 gezeigt, wird eine ganze Reihe verschiedener Darstellungsformen direkt über Variationen des Zusammenspiels von VID7M und LDPS' erreicht. Was dabei im einzelnen passiert, sollte im Verlauf weiterer Erklärungen zu den Bildern 8.7, 8.8 und 8.13 deutlich werden.

Die logischen Gleichungen für die Ausgangsterme des HAL sind bereits in Tabelle 3.4 aufgeführt worden; damit Sie nicht dauernd hin- und herblättern müssen, enthält Tabelle 8.9 die Gleichungen für LDPS' und VID7M noch einmal. Gültig sind sie in dieser Form allerdings nur für Hauptplatinen der Rev. B - der HAL der Rev. A erwartet anstelle von GR+2' das Signal GR+2 und weist einige Differenzen bezüglich der Phasenkorrelation auf. Die folgende Besprechung ist daher in weiten Teilen nur für die Rev. B zutreffend. Die Querverweise in Klammern beziehen sich auf die Gleichungen der Tabelle 8.9.

1. Zwischen TEXT und GRAPHICS wird via GATED GR+2' unterschieden. Dieses Signal findet sich in den Gleichungen als GR' wieder, es ist während GRAPHICS "0", wenn FRCTEXT den Pegel "1" hat. Gültig ist GR' in den Gleichungen während TEXT oder "erzwungenem" TEXT (FRCTXT'). GR'' ist während GRAPHICS gültig, wenn FRCTXT' den Zustand "1" hat.
2. SEGB ist während GRAPHICS äquivalent zu LORES (d.h. HIRES zurückgesetzt), es wird innerhalb des HAL zur Unterscheidung zwischen LORES und HIRES GRAPHICS benutzt.
3. Während LORES GRAPHICS ist VID7M durchgehend "0" (S1). Dadurch wird das Schieberegister durchgehend mit 14 MHz betrieben. Durch Zurücksetzen von 80COL und Aktivierung von FRCTXT' kann ein eigentlich nicht vorgesehener 7M-LORES-Modus gesetzt werden, in dem mit VID7 verzögert wird. In diesem Modus ist VID7M mit 7M identisch.
4. VID7M ist in den 80er Modi (also 80COL zusammen mit TEXT oder "erzwungenem" TEXT) durchgehend "0" (S2).
5. VID7M ist während TEXT (oder "erzwungenem" TEXT) identisch mit 7M, wenn 80COL zurückgesetzt ist (S3). Damit gleichen sich beide Signale aber auch im Modus TEXT40. Als Nebeneffekt ergibt sich, daß HIRES40 ohne Verzögerung betrieben wird, wenn FRCTXT' aktiv ist, und der unter Punkt 3 beschriebene anormale 7M-LORES-Modus.
6. Über die Terme S1 bis S3 sind sämtliche Videomodi außer HiRes40 erzeugbar, die Gleichungen S4, S5 und T1 bis T3 werden ausschließlich für HiRes40 benötigt. Am Ende von PHASE1 * Q3' * AX' (in den Bildern 3.2 und 8.13 durch Punkte gekennzeichnet) innerhalb der Verarbeitung von HiRes40 wird VID7M durch S4 bestimmt: wenn VID7 "0" ist, fällt VID7M - ansonsten bleibt dieses Signal auf "1". Durch diese "abschaltbare Flanke" werden eine Verzögerung um eine Periode von 14M und damit die Farben Blau und Orange anstelle von Violett und Grün (bei einfacher Auflösung) erzeugt. An allen anderen Punkten der HiRes40-Verarbeitung ändert VID7M seinen Zustand immer (d.h. bei T1, T2 und T3).
7. Durch die Definition in S5 wird VID7M innerhalb des ersten "Schwarzzyklus" am rechten Bildschirmrand gezwungen, nach PHASE1 * Q3' * AX' zu fallen. Dadurch wird (zusammen mit einem nicht verzögerten LDPS') der letzte (rechteste) Punkt auf dem Bildschirm abgeschnitten, wenn er verzögert ist (nur HiRes40).
8. In den Modi mit einfacher Auflösung fällt LDPS' nur während PHASE1. Bei doppelter Auflösung wird dieses Signal nach AX' * Q3' sowohl während PHASE1 als auch während PHASE0 aktiv (S1). Etwas einfacher ausgedrückt: In den 80er Modi werden auch die Daten des AUX-RAMs berücksichtigt.
9. Während TEXT (S2), "erzwungenem" TEXT (S3), LORES (S3), unverzögertem HIRES (S4) und während des letzten dargestellten Bytes innerhalb einer Fernsehzeile (S5) fällt LDPS' nach PHASE1 * Q3' * AX'. Während eines verzögerten HiRes40-Zyklus fällt LDPS' eine 14M-Periode später - das nächste Bitmuster wird dadurch ebenfalls verzögert geladen.

Der in den Punkten 3 und 5 erwähnte anormale LoRes-Modus führt ein echtes Schattendasein - nicht nur, daß er fast unbekannt ist, anfangen kann man mit ihm leider auch nicht viel. Dieser Modus verarbeitet im Prinzip die LoRes-Bitmuster mit der Geschwindigkeit von HIRES40 - die Ergebnisse bestehen aus fragmentierten Blocks für alle Bitmuster außer 000, 010, 1010 und 1111. Die letzteren vier Bitfolgen erzeugen dieselben Farben wie unverzögerte HIRES40-Muster: 0101 erscheint auf geraden Horizontalpositionen violett, auf ungeraden grün, bei 1010 ist es genau umgekehrt. Wie zu erwarten, erzeugt 1111 an jeder Stelle des Bildschirms die Farbe Weiß.



NOTE: [1] VID7M is low in double res timing.

Bild 8.7 Erzeugung und Ausgabe von TEXT

Anmerkung:

Dargestellt ist hier das Ende des linken Schwarzbereichs und die Ausgabe der ersten Zeichen am Anfang einer Fernsehzeile.

Tabelle 8.8 Variationsmöglichkeiten der Bildmodi über VID7M und LDPS'

MODE	VARIATION	RESULT
TEXT40	VID7M = 7M LDPS' during PHASE 1	7 MHz load/shift One video cycle per MPU cycle
HIRES40*	VID7M = 7M if VID7' VID7M = 7M' if VID7 LDPS' during PHASE 1 LDPS' delayed if VID7	7 MHz load/shift Delayed 7 MHz load/shift One video cycle per MPU cycle Video cycle delayed if VID7
LORES40*	VID7M constant low LDPS' during PHASE 1	14 MHz load/shift One video cycle per MPU cycle
DOUBLE-RES MODES	VID7M constant low PHASE 1, PHASE 0 LDPS'	14 MHz load/shift Two video cycles per MPU cycle

FRCTXT' = 0 erzwingt Verarbeitung mit 7 MHz ohne Verzögerung

Um 7MHz-LoRes zu aktivieren, müssen Sie LoRes40 wählen und danach AN3 auf den Pegel "0" bringen (vorausgesetzt, Sie haben eine 80-Zeichen-Karte mit 64 kRAM installiert). Ein anderer Weg, diesen Modus kurzfristig einzuschalten, ist das Setzen von LoRes40 und danach ein Druck auf CONTROL-RESET. Solange Sie diese Tasten festhalten, bleibt AN3 durch das aktive RESET' auf dem Pegel "0".

Wesentlich interessanter als der anormale LoRes-Modus scheint mir die Möglichkeit, die Verzögerung von HiRes40 über AN3 an- und abzuschalten. Damit lassen sich die Farben eines kompletten Bildschirms mit einem einzigen Befehl umschalten - vergleichen Sie das doch einmal mit dem Aufwand für ein entsprechendes Programm. Um diese Möglichkeit effektiv zu nutzen, müssen Sie alle HiRes-Bytes mit gesetztem D7 speichern. Danach können Sie via AN3 zwischen Grün/Violett und Orange/Blau umschalten. Voraussetzung dafür ist allerdings, daß 80COL zurückgesetzt und eine 80-Zeichen-Karte mit 64k RAM und einer Verbindung zwischen den Kontakten 50 und 55 des speziellen Steckplatzes installiert ist. Falls Sie nur über einen monochromen Monitor verfügen, können Sie über AN3 den gesamten Bildschirm um einen halben Bildpunkt verschieben - man kann diesen Effekt für Explosionen, Erdbeben und was der reizenden Dinge in Arcade-Spielen sonst noch sind, benutzen.

Wo wir gerade bei interessanten und ansonsten völlig nutzlosen Effekten sind: Diese Umschaltung ist auch in dem 7MHz-LoRes-Modus möglich. Wenn Sie einen LoRes-Bildschirm mit den Bitmustern 5 und A füllen, können Sie danach über AN3 zwischen Grau/Grau und Grün/Violett umschalten. Nicht aufregend!?

Das in Punkt 7 beschriebene Abschneiden des allerletzten Bits von verzögerten HiRes40-Bitmustern ist eine Verbesserung gegenüber dem Apple II, bei dem dieser Vorgang davon abhängig war, ob das erste während HBL verarbeitete Videodatum ein gesetztes Bit 7 hatte oder nicht - sprich: mehr oder weniger per Zufallsfunktion. Schön wäre es gewesen, wenn die Verbesserung soweit gegangen wäre, daß dieses Bit überhaupt nicht abgeschnitten werden muß - das hätte aber eine nicht unerhebliche Änderung der gesamte Bilderzeugung des Apple gefordert (und noch einen weiteren Chip dazu).

Bild 3.2 zeigt den zeitlichen Verlauf dieses Abschneidevorgangs für das letzte Bit eines HiRes40-Musters, zusammen mit anderen Kombinationsmöglichkeiten von LDPS' und VID7M. Bemerkenswert in diesem Diagramm sind zwei Punkte innerhalb des letzten Videozyklus: zum einen ist er nicht verzögert, obwohl VID7 den Wert "1" hat, zum anderen ist der LDPS'-Puls an diesem erzwungenen Schnittpunkt doppelt so breit wie er eigentlich sein sollte.

Tabelle 8.9 Gleichungen der Ausgangsterme des HAL für VID7M und LDPS'

SIGNAL	EQUATIONS	NOTES
VID7M	S1 = GR'' • SEGB S2 = GR' • 80COL'' S3 = GR' • 7M S4 = VID7' • Φ1 • Q3' • AX' S5 = H0' • CLR REF • Φ1 • Q3' • AX' T1 = VID7M • AX T2 = VID7M • Φ0 T3 = VID7M • Q3	LORES GRAPHICS IS HIGH SPEED DOUBLE RES IS HIGH SPEED SAME AS 7M IF NOT HIRES FRCTXT'' • 80COL' ⟶ 7M, UNDELAYED HIRES DELAY CHECK AT Φ1 • Q3' • AX' NO DELAY AT RIGHT DISPLAY EDGE TOGGLE THROUGH AX KEEP TOGGLING THROUGH Φ0 KEEP TOGGLING THROUGH Q3
LDPS'	S1 = Q3' • AX' • 80COL'' • GR' S2 = Q3' • AX' • Φ1 • GR' S3 = Q3' • AX' • Φ1 • SEGB S4 = Q3' • AX' • Φ1 • VID7' S5 = Q3' • AX' • Φ1 • CLR REF • H0' S6 = Q3' • AX • RAS'' • Φ1 • VID7 • SEGB' • GR''	DOUBLE RES CAUSES DOUBLE LDPS' TEXT MODE LORES NOT DELAYED HIRES RIGHT DISPLAY EDGE CUTOFF HIRES DELAYED LDPS'

Die Ausgabe von TEXT

Die Zeitabläufe für die Darstellung von TEXT im Apple //e sind sehr geradlinig und überschaubar. Der Videoscanner liefert eine Adresse, der RAM liefert den entsprechenden Zeichen-Code zum Videodatenbus, wo darüber der Video-ROM adressiert wird. SEGA-SEGC entsprechen direkt VA-VC und bestimmen, welches der acht Teile der Matrix des anliegenden Zeichens ausgegeben wird. Jede 40-Byte-Zeile des TEXT-Bildspeichers wird achtmal hintereinander ausgelesen, bei jedem Durchlauf wird ein weiterer Teil der entsprechenden Matrizen ausgegeben, bis schließlich die gesamte TEXT-Zeile in acht untereinanderliegenden Fernsehzeilen auf dem Bildschirm steht. Die einzelnen Zeichenmatrizen finden Sie in Bild 8.8.

Bild 8.7 zeigt die Ausgabe des siebten von acht Bitmustern für zwei "&"-Zeichen, die hintereinander am Anfang einer Zeile stehen. Das erste Zeichen ist NORMAL, das zweite Zeichen INVERSE gespeichert. Direkt am (linken) Anfang des Diagramms haben die Bits H5-H0 des Videoscanners der Wert 011000: HBL ist soeben auf "0" gegangen und WNDW' wird mit dem nächsten Taktimpuls aktiv, d.h. mit der nächsten steigenden Flanke von RAS' während PHASE1. Der Videodatenbus enthält noch "Schrott" von der letzten während HBL adressierten Speicherstelle, durch den Pegel "1" von WNDW' wird PICTURE' auf "1" gehalten; damit bleibt dieser Teil des Bildschirms schwarz.

Wenn LDPS' im Bild 8.7 zum ersten Mal aktiv wird, hat WNDW' immer noch den Pegel "1", der Ausgang des Video-ROMs wird durch die "pull-up"-Widerstände auf "11111111" gehalten. Etwas anderes bekommt auch das Schieberegister nicht geliefert, es hält PICTURE damit solange weiterhin auf Schwarzpegel, bis LDPS' das nächste Mal aktiv wird.

Ungefähr zur selben Zeit, in der WNDW' nun endlich aktiv wird, werden die Videodaten von den Latches festgehalten, die vom RAM ausgegeben wurden, als H5-H0 auf dem Stand 0110000 war. Die Videodaten vom AUX-RAM werden auf dem Videodatenbus rund 20 ns nach der steigenden Flanke von PHASE0 gültig - wenn COL80 gesetzt ist, fällt LDPS' kurz vor Ende von PHASE0, ansonsten werden sie komplett ignoriert. Für die folgende Besprechung nehmen wir an, daß 80COL gesetzt ist und der Apple sich im Modus TEXT80 befindet. Der untere Teil von Bild 8.7 gibt diesen Zeitablauf wieder.

Bild 8.8 Die Zeichenmatrizen im Video-ROM des Apple //e

Hinweise:
(1) Beim "verbesserten" Apple //e sind die Matrizen für den ASCII-Bereich 40-5F durch "Mauszeichen" ersetzt.
(2) Die deutsche Ausführung des Video-ROMs hat die doppelte Kapazität, der zweite Matrizensatz ist bis auf die Zeichen Ää Öö Üü ß @ identisch mit dem ersten - (Anm. d. Übers.).

Der Videodatenbus enthält zu Anfang unserer Betrachtung den Wert A6, d.h. den ASCII-Wert für ein als NORMAL ausgegebenes "&"-Zeichen. Die IOU übersetzt den Wert von VID7-VID6 (10) auf RAM10-RA9 und erzeugt hier ebenfalls den Wert 10. An den Adreßeingängen des Video-ROMs liegt damit ebenfalls der Wert A6 an. SEGC-SEGA haben den Wert 110 - wir befinden uns im siebten Durchlauf der TEXT-Zeile, der siebte von acht Teilen der Zeichenmatrizen wird ausgegeben. GR+2 hat den Wert "0": die Ausgabe ist nicht auf GRAPHICS, sondern auf TEXT geschaltet. Die komplette ROM-Adresse ergibt lautet damit 0-10-100110-110 (GR+2, RA10-9, VID5-0, SEGC-A). Auf dieser Adresse befindet sich im ROM die zweitunterste Bildzeile für ein "&"-Zeichen in invertierter Form.[1]

Rund 390 Nanosekunden, nachdem die Daten des AUX-RAMs auf dem Videodatenbus gültig geworden sind, findet eine steigende Flanke von 14M statt, während LDPS' und VID7M auf "0" sind. Damit werden die vom Video-ROM ausgegebenen Bitmuster in das Schieberegister geladen. Daraus folgt, daß der Video-ROM eine Zugriffszeit von weniger als 390 ns haben muß. Die Annahme liegt also nahe, daß eine 350 ns-Ausführung eingesetzt wurde - oder auch nicht: Im alten Apple II verläuft das Timing für den Tastatur-ROM ähnlich und Apple, Inc. hat trotzdem einen ROM mit 450 ns eingebaut. Für den //e ist man etwas raffinierter vorgegangen - dieser ROM ist nicht mehr mit der Herstellerkennung, sondern nur noch mit einer Apple-spezifischen Nummer beschriftet, Rückschlüsse auf die Zugriffszeit lassen sich damit nicht mehr so einfach ziehen. Falls Sie den Video-ROM durch ein selbstprogrammiertes EPROM ersetzen wollen, empfehle ich Ihnen allerdings auf jeden Fall einen Typ mit 350 ns Zugriffszeit oder darunter.

Das in unserem Beispiel gerade geladene Bitmuster 11010011 besteht zwar aus acht Bit, repräsentiert aber nur 7 Punkte auf dem Bildschirm, weil nur sechsmal geschoben wird, bevor der nächste Ladevorgang stattfindet. Die erste und die letzte "1" in diesem Byte (d.h. Bit 0 und Bit 6!) sind Bestandteil jeder Zeichenmatrix und sorgen für die notwendigen horizontalen Zwischenräume auf dem TEXT-Bildschirm.

Der Beginn der "sichtbaren" Bildausgabe wird durch den Zeitpunkt festgelegt, zu dem LDPS' das erste Mal nach der fallenden Flanke von WNDW' aktiv wird. Das gilt auch für LoRes und für die verzögerte oder unverzögerte HiRes-Darstellung. Egal, welcher Bildmodus gesetzt ist - der linke Schwarzbereich wird erst durch das Laden des ersten Bitmusters beendet.[2]

Das Bitmuster 1010011 wird bitweise zu PICTURE' geschoben. Auf jede steigende Flanke von 14M findet ein Schiebevorgang statt, weil VID7M konstant auf "0" gehalten wird, wenn TEXT und 80COL gesetzt sind.

Die Videodaten des Hauptplatinen-RAMs werden rund 20 ns nach der fallenden Flanke von PHASE0 gültig, d.h. während der Zeit, in der das durch die Daten des AUX-RAMs erzeugte Bitmuster zu PICTURE' geschoben wird. Der Video-ROM hat auch hier rund 390 ns Zeit, das entsprechende Bitmuster an seinen Ausgängen zur Verfügung zu stellen.

Im in Bild 8.7 dargestellten Beispiel enthält der Videodatenbus jetzt den Wert 26, den Code für ein INVERSEs "&". VID7-6 (00) wird von der IOU zu RA10-9 ebenfalls mit 00 übersetzt, die Gesamtadresse ergibt sich wiederum aus GR+2, RA10-9, VID5-0, SEGC-A und hat den Wert 0-00-100110-110. An der entsprechenden Speicherstelle ist im Video-ROM der Wert 00101100 enthalten, der siebte von acht Teilen eines inversen "&", der ebenfalls invers gespeichert ist. Dieses Muster wird gegen Ende der PHASE1 via LDPS' in das Schieberegister geladen und wird in derselben Weise Bit für Bit zu PICTURE' geschoben wie die vorhergehende Folge, das niederwertigste Bit kommt zuerst.

Im Bildspeicherbereich stehen in unserem Beispiel der Code für das NORMAL ausgegebene "&" und das INVERSE "&" auf derselben Adresse, das erste Zeichen ist im AUX-RAM, das zweite im RAM der Hauptplatine gespeichert. Hier liegt das Grundprinzip aller Modi mit doppelter Auflösung: während eines Videoscanner-Zyklus finden zwei Videoausgaben statt. Darüber lassen sich auch die Modi mit einfacher Auflösung einfach konstruieren: die Schiebegeschwindigkeit wird um die Hälfte heruntergesetzt, es findet nur noch eine Videoausgabe pro Videoscanner-Zyklus statt.

---

1 Sämtliche Bitfolgen sind im Video-ROM in invertierter Form gespeichert, eine "0" steht für "weiß", eine "1" für "schwarz" auf dem Bildschirm. Die Folgen werden über den Ausgang QH des Schieberegisters unverändert weitergegeben und in der PAL-Version erst nach der Verzögerung über das "Einsammel-Register" LS164 ein zweites Mal invertiert, bevor sie zum Summenverstärker gelangen. Der Grund für diese doppelte Invertierung liegt hauptsächlich in der NTSC-Originalversion (vgl. Bild 8.5, 8.6).

2 Hier scheint es eine kleine Differenz zu geben: Nach meinen Messungen ist einzig und allein WNDW' für die Länge der Schwarzperiode verantwortlich, LDPS' wird auch während HBL + VBL erzeugt. Wenn man den OE'-Eingang des Video-ROMs von WNDW' trennt und durchgehend auf "0" legt, erscheinen die während HBL + VBL adressierten Speicherstellen (UNUSED 8, s. Kapitel 5) bzw. deren zugeordnete Zeichen auf dem Bildschirm - (Anm. d. Übers.).

Im oberen Teil von Bild 8.7 finden Sie dasselbe Beispiel mit zwei aufeinanderfolgenden "&"-Zeichen, diesmal allerdings im Modus TEXT40, beide Codes befinden sich im RAM der Hauptplatine. Sie werden während der Horizontalzählerstände (H5-0) 011000 und 1001001 ausgelesen. LDPS' wird während PHASE0 nicht aktiv, die Videodaten des AUX-RAMs werden also ignoriert. *Der linke Schwarzbereich des Bildschirms wird um eine halbe Mikrosekunde gegenüber dem der doppelten Auflösung verlängert*, bis zu dem Punkt, in dem das durch die Daten des Hauptplatinen-RAMs erzeugte Bitmuster geladen und hinausgeschoben wird. Die in diesem Beispiel geladenen Bitfolgen entsprechen denen des vorangegangenen Beispiels, nur die Verarbeitungsgeschwindigkeit und konsequenterweise die Zeichenbreite auf dem Bildschirm ändern sich.

Zwischen einem Taktimpuls des Videoscanners (bzw. dem damit gesetzten Zustand) und dem darauffolgenden LDPS'-Impuls, der das Schieberegister mit den Daten belädt, die innerhalb dieses Zyklus erzeugt wurden, liegt eine relativ lange Zeit (Zugriffszeit des RAMs, Laufzeit der Latches, Zugriffszeit des Video-ROMs). Hier haben wir den Grund, warum WNDW' und SEGA-SEGC um einen vollen Taktzyklus verzögert werden - WNDW' bleibt so lange genug auf "1" und setzt das PICTURE-Signal auf Schwarzpegel, bis das erste Bitmuster, das innerhalb einer Zeile ausgegeben werden soll, wirklich bereit ist. Die Verzögerung von SEGA-SEGC bewirkt, daß diese Signale ziemlich genau zur selben Zeit am Video-ROM "ankommen" wie die dazugehörigen Bits vom Videodatenbus.

Die Erzeugung und Ausgabe von LoRes-Grafik

Wenn Sie die vorhergehenden Abschnitte nicht gelesen hätten, könnten Sie vielleicht auf die Idee kommen, daß ein LoRes-Block durch einen dicken Impuls mit einer Mikrosekunde Breite auf dem Bildschirm erzeugt wird. Tatsächlich gibt es nur eine einzige Gelegenheit, wo das der Fall ist: bei der Ausgabe eines weißen LoRes-Blocks im Modus LoRes40 hat das VIDEO-Signal eine ganze Mikrosekunde lang Weißpegel. Alle anderen farbigen Blocks bestehen aus einer Folge von Pulsen, die so dicht zusammenstehen, daß sie ein Fernseher nicht mehr als einzelne Punkte innerhalb einer Zeile darstellen kann - sie verschwimmen zu einem durchgehenden Strich. Außerdem liegen diese Pulse auf der Frequenz des Farbträgers und werden deshalb vom Chroma-Verstärker des Fernsehers oder Computermonitors als Farbinformation akzeptiert.[3]

Ein LoRes-Farbsignal wechselt mit einer Frequenz von 3.56 MHz zwischen Schwarz- und Weißpegel hin und her. Die dadurch erzeugte Farbe hängt von der Phasenkorrelation dieser Schwingung zu COLOR REFERENCE ab. Außerdem muß das PICTURE-Signal nicht unbedingt symmetrisch verlaufen - entsprechend größer sind die Möglichkeiten der Farberzeugung. Auf einen Nenner gebracht: Es gibt 12 LoRes-Bitfolgen, die Farbe produzieren, nämlich 0001, 0010, 0011, 0100, 0110, 0111, 1000, 1001, 1011, 1100, 1101 und 1110.

Die Adreßeingänge des Video-ROMs (GR+2, RA10-9, VID5-0, SEGC-A) werden im Modus LoRes folgendermaßen gesetzt: "1", VID7-6, Vid5-0, VC, "1", H0. Über GR+2 und SEGB bzw. über die beiden konstanten "Einser" an den entsprechenden Eingängen wird der LoRes-Bereich des ROMs selektiert. Die Unterteilung durch VC und H0 in vier Stücke bringt einige weitere Spezialitäten der LoRes-Ausgabe mit sich, die Thema der folgenden Abschnitte sind.

Der LoRes-Bereich des Video-ROMs enthält vier Arten von Bitfolgen, die allesamt in derselben Weise invertiert sind wie die TEXT-Zeichenmatrizen. Jede Bitfolge ist eine zweifache Wiederholung des COLOR-Wertes, über den sie adressiert wird. Ein Byte des Bildspeichers enthält im Modus LoRes die Information für zwei übereinanderliegende Blocks, jeder Block besteht aus vier Fernsehzeilen. Über VC wird deshalb bestimmt, ob die auszugebende Bitfolge von VID3-0 (= oberer Block) oder von VID7-4 (= unterer Block) bestimmt wird. In beiden Fällen wird zusätzlich über H0 bestimmt, ob die "originale" Bitfolge oder eine um zwei Bit verschobene Folge ausgegeben wird - darüber ergibt sich für ein Muster unabhängig von seinem Speicherplatz (gerade/ungerade) immer dieselbe Farbe.

Tabelle 8.10 zeigt die vier möglichen Kombinationen von VC und H0 am Beispiel eines Videodatums mit dem Wert 6C. Wenn VC den Wert "0" hat, werden die oberen Blocks einer LoRes-Zeile ausgegeben, deren Wert aus den Bits VID3-0 erzeugt wird. Wenn sich die gerade ausgelesene Speicherstelle auf einer geraden Horizontalposition befindet (H0 = "0"), dann gibt der Video-ROM eine einfache Inversion des Bitmusters aus (aus 0111 wird 1000 1000). Falls dieser Wert dagegen aus einer ungeraden Adresse ausgelesen wird, muß die Phasenlage zu COLOR REFERENCE korrigiert werden: H0 hat den Wert "1" und der Video-ROM gibt anstelle von 1000 1000 den Wert

---

3 Wie im Abschnitt über die PAL-Version des //e erklärt, ist das in direkter Form nur für die NTSC-Norm der Fall. Letztendlich erzeugt der PAL-Wandler aber dasselbe Ergebnis, die Voraussetzung ist in beiden Fällen, daß die Folgefrequenz der Signalspitzen mit der von COLOR REFERENCE (NTSC: 3.58, PAL: 356 MHz) übereinstimmt. Im folgenden Text wird von der PAL-Version mit 3.56 MHz ausgegangen - (Anm. d. Übers.).

0010 0010 aus - dieselbe (doppelte) Bitfolge, aber um 2 Bit nach rechts verschoben. An dieser Stelle folgt noch einmal eine etwas genauere Erklärung der "Kompensation":

Ein Videozyklus hat dieselbe Länge wie 3.5 Perioden von COLOR REFERENCE. Wenn wir annehmen, daß mit dem ersten Zyklus einer Zeile ("gerade Adresse") PICTURE und COLOR REFERENCE in Phase sind, dann ist COLOR REFERENCE beim zweiten Zyklus um 180 Grad (0.5 Perioden) voraus, der dritte Zyklus beginnt wieder in Phase, weil COLOR REFERENCE inzwischen 7 komplette Perioden hinter sich hat. Dieser Prozeß setzt sich durch die gesamte Zeile hindurch fort - bei jeder zweiten Speicherstelle stimmt die Phasenbeziehung "automatisch", die dazwischenliegenden "ungeraden" Speicherstellen müssen entsprechend kompensiert werden.

Bei jeder Auslese einer ungeraden Speicherstelle ist COLOR REFERENCE also bereits eine halbe Periode (eine siebtel Mikrosekunde) "zu weit vorne". Im Modus LoRes wird immer mit 14 MHz Schiebefrequenz gearbeitet, ein Bit dauert also eine vierzehntel Mikrosekunde. Durch das vorherige Rechtsverschieben des Bitmusters um 2 Bit wird exakt die siebtel Mikrosekunde "eingespart", die COLOR REFERENCE vorauseilt - das Ergebnis ist eine von der Speicheradresse unabhängige Farbdarstellung.

Tabelle 8.10 Kombinationsmöglichkeiten von H0 und VC in LoRes

Videodatum	VC	H0	Bitmuster des Video-ROMs	PICTURE
67 (0110) 0111)	0	0	$88 (1000 1000)	$77 (0111 0111)
67 (0110) 0111)	0	1	$22 (0010 0010)	$DD (1101 1101)
67 (0110) 0111)	1	0	$99 (1001 1001)	$66 (0110 0110)
67 (0110) 0111)	1	1	$66 (0110 0110)	$99 (1001 1001)

Die unteren beiden Zeilen von Tabelle 8.10 zeigen die Verarbeitung des unteren Blocks, der durch das Videodatum $27 bestimmt wird. Sie ist identisch mit der des oberen Blocks, bis auf die Tatsache, daß VID7-4 zur Erzeugung des Musters benutzt werden.

Bild 8.9 zeigt verschiedene Beispiele für die Zeitabläufe bei der Ausgabe von LoRes-Blocks, insgesamt sind es drei für einfache und zwei für doppelte Auflösung. Der Stand von VC spielt in diesen Beispielen überhaupt keine Rolle - dieses Signal bestimmt lediglich, ob die obere oder die untere Hälfte des Videodatums ausgewertet wird.

Die Ausgabe von LoRes40

Die ersten drei Beispiele in Bild 8.9 zeigen gerade/ungerade Paare von LoRes-Blocks in einfacher Auflösung. Dieser Modus wird durch Zurücksetzen der Softswitches 80COL, TEXT und HIRES und durch Setzen von AN3 (FRCTXT') aktiviert.[4] VID7M ist die gesamte Zeit auf "0", weil GATED GR+2' und SEGB den Pegel "1" haben, die Lade-/Schiebefrequenz ist damit durchgehend 14 MHz. Da der Ausgang des Schieberegisters mit dem höchstwertigen Eingang verbunden ist und die Bits damit im Kreis herumgeschoben werden, gehen dem Bildgenerator die Daten nicht aus, obwohl LDPS' nur einmal pro Videozyklus aktiv wird, nämlich während PHASE1, und damit nur Videodaten vom Hauptplatinen-RAM geladen werden. Die Bitfolge wird 1.75 Mal hintereinander ausgegeben (14 Zyklen von 14M auf 8 Bit), bevor das nächste Muster vom Video-ROM geladen wird. Ein LoRes-Block besteht aus vier Bit, die im Video-ROM einfach doppelt hintereinander gestellt werden. Damit ergeben sich 3.5 Ausgaben dieser Folge innerhalb einer Mikrosekunde oder 3.5 Millionen Ausgaben pro Sekunde - nicht gerade zufällig die Frequenz von COLOR REFERENCE. Alle LoRes-Bitfolgen außer 0000, 0101, 1010 und 1111 erzeugen damit ein PICTURE-Signal, das seinen Pegel mit 3.5 MHz ändert, und damit einen farbigen Block auf dem Bildschirm. (0000 und 1111 erzeugen überhaupt keine Änderung bzw. einen Puls von einer Mikrosekunde Dauer, 0101 und 1010 erzeugen eine konstante Frequenz von 7 MHz. Alle anderen Bitfolgen haben mindestens eine Teilfolge, die eine Frequenz von 3.5 MHz erzeugt.

Das erste Beispiel in Bild 8.9 zeigt das Ergebnis der Bitfolge 1001 in zwei nebeneinanderliegenden Speicherstellen. Die erste Speicherstelle hat eine gerade, die zweite folglicherweise eine ungerade Adresse. Die Auslese der geraden Speicheradresse ergibt eine einfache Inversion und Verdoppelung, das Schieberegister wird mit dem Wert 0110 0110 beladen. Durch bitweises Herausschieben (niederwertigstes Bit zuerst) und die zweite Inversion ergibt sich für PICTURE die Bitfolge 10011001100110. Im "ungeraden" Zyklus wird der Wert 1001 1001 in das Schieberegister geladen, PICTURE hat die Bitfolge 01100110011001. In beiden Fällen ist PICTURE damit ein symmetrisches

4 Wenn FRCTXT' über AN3 aktiviert wird und 80COL nicht gesetzt ist, dann alterniert VID7M und setzt die Schiebefrequenz auf 7 MHz herab. Die Folge ist der im Abschnitt "Die Zeitsignale der Bilderzeugung" beschriebene anormale 7 MHz-LoRes-Modus, mit dem wir uns an dieser Stelle nicht weiter im Detail beschäftigen wollen.

Rechtecksignal und hat eine Phasenbeziehung zu COLOR REFERENCE, die vom Fernseher/Monitor als die Farbe Orange erkannt wird.

Die Farbe des linken und des rechten Randes eines LoRes-Blocks hängt von den Bitmustern seiner Nachbarn ab. Zwei nebeneinanderliegende Blocks gleicher Farbe ergeben eine kontinuierliche Bitfolge und damit einen doppelt so breiten Block ohne eine sichtbare Trennstelle in der Mitte. Zwei Blocks unterschiedlicher Farbe ergeben an der Stelle des Farbwechsels meistens eine dritte Bitfolge als Interferenzerscheinung, die unterschiedlich stark in Erscheinung tritt - je nach Art der beiden Farben und abhängig davon, ob der Übergang von einer geraden auf eine ungerade Adresse stattfindet oder umgekehrt. Auch wenn im ersten Beispiel von Bild 8.9 die linken und die rechten Ränder der Blocks orange dargestellt sind - ihre tatsächliche Farbe hängt von ihren Nachbarn ab.

Das zweite Beispiel in Bild 8.9 ist das Muster 0111 (hellblau) auf einer geraden Adresse, gefolgt von 1000 (dunkelbraun) auf einer ungeraden Speicherstelle. Daraus ergibt sich ein unsymmetrisches Rechteck mit einer Frequenz von 3.56 MHz, dessen Grundschwingung (3.56 Mhz-Sinus) vom Chroma-Verstärker als Farbinformation akzeptiert wird. Unsymmetrische Rechtecke werden von Bitfolgen erzeugt, bei denen nur ein Bit gesetzt oder nur ein Bit zurückgesetzt ist. Eine Folge wie 1000 erzeugt eine dunkle Farbe, weil das PICTURE-Signal sich zu 75 Prozent der Zeit auf dem Schwarzpegel befindet - bei 0111 ist es umgekehrt, hier entstehen helle Farbschattierungen. Wie im Beispiel gezeigt, wird durch die Bitfolge an der Grenzstelle zwischen beiden Blocks ein weiteres Bitmuster erzeugt: 0111 auf einer geraden Adresse, gefolgt von 1000 auf einer ungeraden Speicherstelle, erzeugt beim Übergang einen violetten Strich.

Anhand des hellblauen Blocks läßt sich zeigen, daß die meisten "Grenzfarben" der LoRes-Blocks vorhersagbar sind: Jede Bitfolge, die von einer ungeraden Adresse gelesen wird und mit einer "1" endet, ergibt zusammen mit der linken Seite eines folgenden 0111-Musters eine weiße Grenzstelle. Speziell bei den hellen Farbschattierungen ist das meistens der Fall - schließlich unterscheiden sie sich selbst von der Farbe weiß nur durch ein einziges nicht gesetztes Bit.

Das dritte Beispiel in Bild 8.9 ist das Muster 0101 auf einer geraden Adresse, gefolgt von 1010 auf einer ungeraden Speicherstelle. Beide ergeben die Farbe Grau, weil PICTURE hier mit 7 MHz zwischen Schwarz- und Weißpegel geschaltet wird - der Chroma-Verstärker des Fernsehers/Monitors weigert sich, diese Frequenz als Farbinformation zu erkennen und schaltet auf schwarzweiß. Die Farbe Grau ist somit nichts anderes als weiß und schwarz zu gleichen Teilen gemischt, die Helligkeit des entstehenden Signals ist bei den Bitfolgen 101010.. und 010101.. dieselbe. Das ist der Grund, warum im Überblick am Anfang dieses Buchs behauptet wurde, daß in LoRes40 nur 15 (und nicht 16) Farben inklusive Schwarz und Weiß zur Verfügung stehen.

Auch wenn beide Graufarben miteinander identisch sind - ihr PICTURE-Signal liegt um 180 Grad auseinander. Aus diesem Grund ergibt sich eine kräftige Interferenz, wenn 1010 auf 0101 folgt. Das kann natürlich ausgenutzt werden, um zwei graue Blocks horizontal voneinander zu trennen - wenn man diese Interferenz vermeiden will, sollte man sich dagegen innerhalb eines Bildes für eine einzige Graufarbe entscheiden. In der praktischen Anwendung wird man sich wohl für das Grau entscheiden, bei dem die geringeren Interferenzerscheinungen mit anderen verwendeten Farben zu bemerken sind.

Wenn LoRes-Farben nebeneinander in numerischer Reihenfolge ausgegeben werden, dann ist das Ergebnis alles andere als eine erkennbare Farbfolge. Etwas logischer sieht es dagegen aus, wenn man die Farben kreisförmig ihrer Phasenbeziehung zu COLOR REFERENCE nach anordnet - schließlich werden sie als Bitfolgen interpretiert und nicht als numerische Werte.

Bild 8.10 zeigt die durch die LoRes-Bitfolgen erzeugten PICTURE-Signale im (Phasen-)Vergleich zu COLOR REFERENCE. Je weiter oben im Bild eine Farbe steht, desto näher fällt der Puls im PICTURE-Signal mit einer Flanke von COLOR REFERENCE zusammen. Diese Darstellung macht einen recht interessanten Punkt offensichtlich, nämlich die Aufteilung in die vier Farbtonpaare dunkles/helles Magentarot, dunkles/helles Blau, dunkles/helles Blaugrün und dunkles/helles Braun. Der PICTURE-Impuls, durch dessen Phasenlage dunkles Magentarot erzeugt wird, ist auf beiden Seiten vom Puls für helles Magentarot umgeben, ein "Durchschnitt" der Zentren beider Pulse fällt mit der Mitte von COLOR REFERENCE zusammen. Beide Phasenunterschiede zu COLOR REFERENCE erzeugen dieselbe Farbe, die Bitfolge mit breiteren "1"-Pulsen erzeugt den helleren, die andere Folge den dunkleren Farbton.

Die Farbe 0001 wird normalerweise als "Magentarot" bezeichnet, die Farbe 1011 als "Rosa". Innerhalb dieses Buchs ist Rosa durchgehend als "helles Magentarot" bezeichnet worden, um damit zu unterstreichen, daß die Farbe 1011 nichts weiter ist als eine helle Schattierung von 0001.

Die Ausgabe von LoRes80

Die letzten beiden Beispiele in Bild 8.9 zeigen die Ausgabe von fünf aufeinander folgenden Blocks in LoRes80. Dieser Modus wird durch Zurücksetzen der Softswitches TEXT, HIRES, AN3 (FRCTXT') und Setzen von 80COL aktiviert. Eine höhere Schiebefrequenz ergibt sich dadurch nicht - LoRes40 arbeitet bereits mit 14 MHz, und schneller geht nichts auf dem Apple. Der einzige Unterschied zwischen den beiden Modi liegt darin, daß LDPS' in LoRes80 sowohl während PHASE0 als auch während PHASE1 aktiv wird, die Videodaten des AUX-RAMs werden also ebenfalls berücksichtigt. Für jedes Bitmuster bleibt damit nur noch die Hälfte der Zeit - es wird nicht mehr nach 3.5, sondern bereits nach 1.75 "Rundläufen" im Schieberegister durch das nächste ersetzt. In dieser Zeit (0.5 Mikrosekunden) finden sechs Schiebevorgänge des 4-Bit-Musters statt. Da die Bitfolgen mit derselben Geschwindigkeit hinausgeschoben werden wie in LoRes40, stehen auch dieselben Farben zur Verfügung - die Blocks haben aber nur noch die halbe Breite. Außerdem erzeugt eine im AUX-RAM gespeicherte Bitfolge eine andere Farbe als eine aus dem RAM der Hauptplatine. Diese Tatsache wird vom ersten LoRes80-Beispiel in Bild 8.9 verdeutlicht, bei dem die kontinuierliche Bitfolge 0010 in beiden Bereichen gespeichert ist.

Wenn diese Folge aus einer geraden Speicherstelle des AUX-RAMs gelesen wird (ganz links in Bild 8.9), dann gibt der Video-ROM den Wert 1101 1101 aus, also eine einfache Inversion und Verdoppelung. Dieses Muster wird mit dem niederwertigsten Byte zuerst hinausgeschoben und ein zweites Mal invertiert, für PICTURE kommt 0010 0010 heraus. Wenn danach aus einer geraden Speicherstelle des Hauptplatinen-RAMs ebenfalls 0010 gelesen wird, kommt dafür ebenfalls die PICTURE-Bitfolge 0010 0010 heraus, das PICTURE-Signal für die erste Folge wurde aber bereits nach sechs Schiebevorgängen abgeschnitten. Die sich ergebende Gesamtfolge ist 010 001 0 0 10 0010 - zwischen zwei Blocks gleicher Farbe entsteht ein deutlich sichtbarer Trennstrich. Diese Diskontinuität gilt für das gesamte Beispiel.

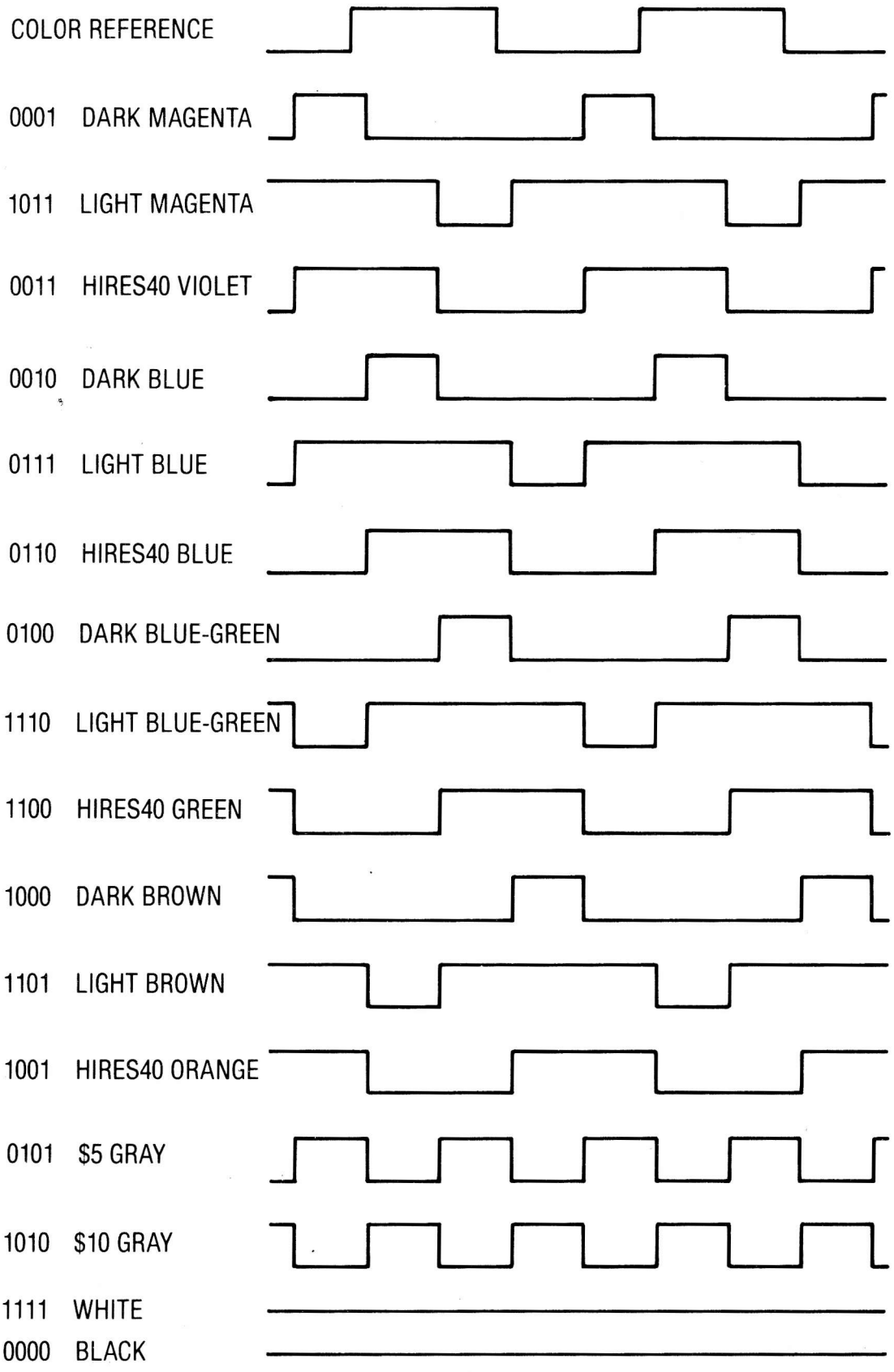
Um die 7-Bit-Folge 010 0010 harmonisch fortzusetzen, wird das Muster 001 0001 benötigt, das durch den Wert 0001 in einer Speicherstelle des Hauptplatinen-RAMs erzeugt wird. Das letzte Beispiel von Bild 8.9 zeigt das sehr deutlich: Um eine Linie mit einheitlicher Farbe zu zeichnen, die die LoRes40-Farbe 0100 hat, muß der Wert 0100 im Hauptplatinen-RAM und der Wert 0010 in den entsprechenden Speicherstellen von AUX gespeichert werden. Diese Beziehung gilt für alle LoRes-Farben: Um eine "Hauptplatinen-Farbe" in AUX darzustellen, müssen Sie das entsprechende Bitmuster um ein Bit nach rechts rotiert speichern. Tabelle 8.11 faßt alle Möglichkeiten für dabei entstehende Bitfolgen zusammen.

Ein Nachteil von LoRes80 liegt darin, daß die Breite eines Blocks wesentlich stärker von seiner Farbe abhängig ist als in LoRes40. Die optische Breite eines Blocks wird auch in LoRes40 dadurch bestimmt, wieviele nicht gesetzte Bits am Anfang und am Ende der Bitfolge stehen - in LoRes80 fällt das aber wesentlich stärker auf, weil ein Block nicht mehr aus 14 Punkten (3.5 Wiederholungen), sondern nur noch aus der Hälfte besteht. Die Bitfolgen 1000 von einer geraden oder 0010 von einer ungeraden Speicherstelle sind dafür ein Extrembeispiel, sie erzeugen "Blocks", die gerade noch einen Punkt breit sind. (In LoRes40 erzeugt 1000 die Folge 00 1000 1000 1000, also einen Block mit 12 Punkten Breite, in LoRes80 wird daraus tatsächlich 000 1000.)

Tabelle 8.11 Gleiche LoRes-Farben durch verschiedene Bitmuster von der Hauptplatine und von AUX

SINGLE RES COLOR	AUXILIARY CARD	MOTHERBOARD
$0 (0000)	$0 (0000)	$0 (0000)
$1 (0001)	$8 (1000)	$1 (0001)
$2 (0010)	$1 (0001)	$2 (0010)
$3 (0011)	$9 (1001)	$3 (0011)
$4 (0100)	$2 (0010)	$4 (0100)
$5 (0101)	$A (1010)	$5 (0101)
$6 (0110)	$3 (0011)	$6 (0110)
$7 (0111)	$B (1011)	$7 (0111)
$8 (1000)	$4 (0100)	$8 (1000)
$9 (1001)	$C (1100)	$9 (1001)
$A (1010)	$5 (0101)	$A (1010)
$B (1011)	$D (1101)	$B (1011)
$C (1100)	$6 (0110)	$C (1100)
$D (1101)	$E (1110)	$D (1101)
$E (1110)	$7 (0111)	$E (1110)
$F (1111)	$F (1111)	$F (1111)

Bild 8.10 Die Phasenlagen von LoRes- und HiRes80-Bitfolgen zu COLOR REFERENCE



Die Firmware des Apple //e enthält leider keine speziellen Befehle, die das Zeichnen in LoRes80 möglich machen. Allerdings ist es nicht übermäßig schwierig, entsprechende Routinen zu schreiben, was ruhigen Gewissens in Integer BASIC oder Applesoft geschehen darf.

Die Erzeugung und Ausgabe von HiRes-Grafik

HiRes ist dem Modus TEXT in einigen Punkten wesentlich ähnlicher als man denkt. Beide Signale werden durch Schieben von Folgen mit 7 Bit erzeugt - aus diesem Grund läßt sich im Modus HiRes Text mit denselben 7 * 8 - Zeichenmatrizen ausgeben wie sie im Video-ROM gespeichert sind. Allerdings werden diese Zeichen etwas bunt ausfallen, weil im Modus HiRes (im Gegensatz zu TEXT) ein COLOR BURST erzeugt wird. Abgesehen davon gibt es natürlich schon ein paar wichtige Unterschiede:

1. HiRes-Bitfolgen sind direkt mit 7 Punkten pro Byte im RAM gespeichert. Bei TEXT stehen dagegen ASCII-Zeichen im Bildspeicher, über die eine Matrix mit 7 * 8 Punkten innerhalb des Zeichen-ROMs adressiert wird.
2. Wenn das höchstwertige Bit (VID7) innerhalb eines HiRes-Bytes gesetzt ist und FRCTXT' den Pegel "1" hat, dann werden die folgenden 7 Bit um eine Periode von 14M verzögert ausgegeben.

Die Übersetzung und Ausgabe von HiRes-Bitfolgen via Video-ROM und Schieberegister besteht schlicht aus einer doppelten Inversion, d.h. es findet letztendlich überhaupt keine Veränderung statt. Die Farberzeugung ist dagegen wirklich geeignet, um einem graue Haare wachsen zu lassen. Bild 8.13 zeigt eine erstaunliche Menge von Farben, die allesamt über ein und dieselbe Bitfolge (1011010) erzeugt wurden. Bei der Besprechung dieses Beispiels werden Sie sehen, daß die Farbe eines Hires-Bitmusters von den folgenden Faktoren *und ihrer Kombination* abhängen kann:

- einfache oder doppelte Auflösung;
- gerade oder ungerade Speicherstelle;
- Speicherung im Hauptplatinen-RAM oder in AUX (nur HiRes80);
- verzögerte oder unverzögerte Ausgabe (HiRes40);
- verzögerte oder unverzögerte Ausgabe der benachbarten Muster (HiRes40).

Die Adreßbits des Video-ROMs (GR+2, RA10-9, VID5-0, SEGC-A) haben in sämtlichen HiRes-Modi die Werte 1, VID7-6, VID5-0, VC, 0, H0. Über GR+2 und SEGB wird der HiRes-Bereich des ROMs angesteuert. (GR+2 ist "1" und unterscheidet GRAPHICS von TEXT, SEGB ist "0" und unterscheidet HiRes von LoRes.) Die Adreßbits VID7, VC und H0 haben dagegen überhaupt keinen Einfluß - alle acht durch sie ansteuerbaren Bereiche haben exakt denselben Inhalt. Einfacher gesagt: *Im Modus HiRes wird die Adresse des Video-ROMs ausschließlich von VID6-0 bestimmt.*

Der Video-ROM enthält auch die HiRes-Bitmuster in invertierter Form. Auf allen Adressen, die sich über "VID6-0 = 111 0011" ergeben, findet sich das Datum 000 1100, auf der Adresse 000 0000 das Datum 111 1111 etc. Das überflüssige achte Bit ist bei allen HiRes-Bitfolgen auf den Wert 1 gesetzt, es spielt keine Rolle.[5] Die Bitfolge wird über das Schieberegister zu PICTURE' geschoben und dort erneut invertiert. Das Ergebnis dieser doppelten Inversion ist bereits bekannt: *PICTURE enthält im Modus HiRes eine Bitfolge, die mit dem aus dem Bildspeicher ausgelesenen Wert identisch ist.* Ein im Bildspeicher gesetztes Bit erzeugt einen Weißpegel in PICTURE und damit einen Punkt auf dem Bildschirm, ein nicht gesetztes Bit erzeugt einen Schwarzpegel - von der Farberzeugung einmal abgesehen, mit der wir uns in den nächsten beiden Abschnitten beschäftigen werden.

Die Ausgabe von HiRes40

Die obersten beiden Beispiele in Bild 8.13 zeigen die vier grundlegenden Möglichkeiten für Phasenverhältnisse zwischen HiRes40-Bitfolgen und COLOR REFERENCE. Das Ergebnis sind vier verschiedene Farben für ein und

---

5 Das höchstwertige Bit einer HiRes-Bitfolge gelangt zwar zum Schieberegister, wird aber unter keinen Umständen hinausgeschoben - nach dem Hinausschieben des siebten Bits erfolgt der nächste Ladevorgang. Apple, Inc. stand damit vor der äußerst schwierigen Entscheidung, welchen Wert sie den achten Bits im Video-ROM geben sollten. Nach langen Konferenzen haben sie sich für eine "1" entschieden. Ob der Computer dadurch mehr wiegt?

dasselbe Bitmuster, die unterschiedlichen Farben kommen dadurch zustande, daß das Muster 1011010 in jeweils zwei aufeinanderfolgende Speicherstellen geschrieben wurde, beim ersten Paar ist D7 gesetzt, beim zweiten zurückgesetzt.

Die im obersten Beispiel von Bild 8.13 gezeigten Farben entstehen daraus, daß Videoscanner und -generator die Bitfolge 01011010 zuerst von einer geraden Adresse, danach von einer ungeraden Adresse lesen. Da in beiden Fällen das höchstwertige Bit zurückgesetzt ist, erfolgt keine Verzögerung bei der Ausgabe.

Um festzustellen, ob eine HiRes-Bitfolge verzögert oder unverzögert ausgegeben werden soll, überprüft der HAL den Stand von VID7 während PHASE1 * AX' * Q3', der in Bild 8.13 mit einem Punkt gekennzeichnet ist. Wenn VID7 zu diesem Zeitpunkt zurückgesetzt ist (wie im ersten Beispiel von Bild 8.13), dann fällt LDPS' zusammen mit VID7M bei der nächsten steigenden Flanke von 14M. Auf jede weitere steigende Flanke von 14M folgt eine Umschaltung von VID7M - dieses Signal ist bei unverzögerter HiRes-Ausgabe mit 7M identisch, der Lade- und Schiebevorgang von HiRes40 gleicht damit dem von TEXT40.

Während des Hinausschiebens der ersten drei Bit (101) im obersten Beispiel von Bild 8.13 geht das Signal zuerst auf den Schwarzpegel, wird danach "weiß" und geht wieder zurück auf Schwarzpegel. Die Frequenz der dabei entstehenden Rechteckschwingung ist 3.56 MHz, also identisch mit der von COLOR REFERENCE. Ein farbfähiger Monitor oder Fernseher wird dieses Signal über seinen Chroma-Verstärker leiten und es danach mit seiner eigenen COLOR REFERENCE vergleichen, deren Frequenz und Phasenlage durch den COLOR BURST bestimmt wird. Als Ergebnis dieses Vergleichs ergibt sich eine Farbinformation, die Bitfolge 101 in unserem Beispiel erzeugt einen grünen Punkt auf dem Bildschirm. Vergleichen Sie einmal die Phasenlage dieser Schwingung mit COLOR REFERENCE: *Jede* Schwingung mit 3.56 MHz und dieser Lage erzeugt die Farbe grün.

Wenn wir das Beispiel weiter verfolgen, erhält PICTURE nach diesem grünen Punkt (010) zwei aufeinanderfolgende "Einser" und danach eine weitere "0", geht also von "0" für die Zeitdauer von zwei Punkten auf "1" und danach wieder auf "0". Das entspricht einer Frequenz von 1.78 MHz, d.h. der Hälfte von COLOR REFERENCE, und wird vom Chroma-Verstärker des Fernsehers/Monitors praktisch komplett ignoriert. Das Ergebnis sind zwei weiße Punkte auf dem Schirm. Die Phasenlage zu COLOR REFERENCE spielt dabei keine Rolle - eine Folge wie 0110 bzw. eine Folgefrequenz von 1.78 MHz erzeugt zu jedem beliebigen Zeitpunkt die Farbe Weiß.

Die letzten Bits unseres Beispiels haben den Wert 010 und erzeugen dasselbe Rechteck wie für einen grünen Punkt. Sie haben aber eine um 180 Grad gedrehte Phasenlage zu COLOR REFERENCE - das Ergebnis ist ein violetter Punkt. Grün und Violett sind deshalb Komplementärfarben.

Wenn genau dieselbe Bitfolge aus einer Speicherstelle mit ungerader Adresse gelesen wird, führt PICTURE exakt dieselben Schwingungen aus wie in den vorhergehenden Absätzen beschrieben. COLOR REFERENCE steht aber bei einer ungeraden Speicherstelle genau um 180 Grad"verkehrt" (weil vom letzten "geraden" Zyklus noch eine halbe Periode übriggeblieben ist). Deshalb ergibt sich in diesem Fall die komplementäre Farbfolge: Grün wird Violett, Violett wird Grün, Weiß bleibt unverändert.

Damit sollte die Natur der Farben Grün und Violett im Modus HiRes40 eigentlich geklärt sein. Ein Punkt in HiRes 40 dauert exakt halb so lange wie eine Periode von COLOR REFERENCE. Eine Folge von 010 oder 101 erzeugt deshalb eine Schwingung von 3.56 MHz und damit ein Farbsignal. COLOR REFERENCE ist am Anfang des ersten dargestellten Bytes einer Zeile "0", am Anfang des nächsten ("ungeraden") Bytes "1", am Anfang des dritten ("geraden") Bytes wieder "0" usw. Wenn Punkte auf eine gerade Position gesetzt werden, beginnt die entsprechende Schwingung eine Viertelperiode vor der steigenden Flanke von COLOR REFERENCE und endet eine viertel Periode nach dieser Flanke, das Ergebnis ist violett. Auf ungeraden Positionen gesetzte Punkte haben exakt das umgekehrte Phasenverhältnis und werden deshalb grün dargestellt, zwei oder mehr aufeinanderfolgend gesetzte Punkte erscheinen weiß.

Ein großer Teil der bekannten HiRes-Bilder des Apple besteht aus unverzögerten Bitmustern einfacher Auflösung. Tatsache ist, daß diese Ausgabeform die einzige war, die auf der Originalversion des Apple II (Rev. 0) zur Verfügung stand. Die unverzögerte Ausgabe von HiRes40 liefert 280 einzeln programmierbare Punkte mit den Farben Grün, Violett und Weiß pro Fernsehzeile. Bei monochromer Darstellung ist damit eine Auflösung von 280 * 192 Punkten erreichbar; wenn mit Grün und Violett gearbeitet werden soll, stehen allerdings nur noch die Hälfte der Punkte zur Verfügung und die Auflösung sinkt auf 140 * 192.

Das zweite Beispiel in Bild 8.13 zeigt die beiden anderen möglichen Ergebnisse für die bereits im ersten Beispiel verwendete Bitfolge 1011010. Sie wird ebenfalls aus zwei aufeinanderfolgenden Speicherstellen gelesen, allerdings ist hier jeweils das Bit 7 gesetzt. Für PICTURE ergibt sich dasselbe Signal wie im ersten Beispiel, nur ist diese Bitfolge um eine Periode von 14M verzögert: der HAL hat während PHASE1 * AX' * Q3' festgestellt, daß VID7 den

Wert "1" hat. Bei mehreren aufeinanderfolgenden Bytes mit gesetztem Bit 7 (wie im Beispiel gezeigt), wird LDPS' während der letzten 14M-Periode von PHASE1 aktiv, VID7M entspricht konstant dem Signal 7M' (also um 180 Grad gegenüber 7M gedreht).

Durch diese Verzögerung von VID7M um eine halbe Periode (also um einen halben Bildpunkt bzw. um eine viertel Periode von COLOR REFERENCE) ergibt sich für dieselbe Bitfolge eine andere Phasenlage zu COLOR REFERENCE als bei unverzögerter Ausgabe. Daraus entstehen die beiden Farben Orange und Blau, außerdem werden die Punkte um eine halbe Punktposition nach rechts im Bildschirm verschoben.

Die Erzeugung dieser beiden Farben über eine einfache Verzögerung ist ein fast genialer Trick - nur wird damit die Programmierung von HiRes40 zu einer noch abstrakteren Kunst. Jede Gruppe aus sieben Punkten bekommt ihre eigene Charakteristik bezüglich Farbe und Position. Zusammengenommen mit den von der Punktposition abhängigen Farben und den wirklich nicht einfach zu berechnenden Bildschirmadressen ergeben sich Hürden, an denen schon mancher Programmierer gescheitert ist.

Dabei haben wir uns um den letzten und schwierigsten Punkt noch überhaupt nicht gekümmert! Bevor wir uns allerdings in die Details der Interferenzerscheinungen zwischen verzögerten und unverzögerten Bitmustern stürzen, fassen wir einmal kurz zusammen, was bis jetzt über HiRes40 bekannt ist.

Wir haben 192 Zeilen mit jeweils 280 (40 * 7) Punkten, die individuell gesetzt werden können. Jede Siebenergruppe kann horizontal um eine halbe Punktposition verschoben werden, die damit erreichbare (monochrome) Auflösung liegt in Spezialfällen tatsächlich bei 560 * 192 Punkten. Die Farbe einzelner Punkte ist abhängig von ihrer Horizontalposition, es gibt 140 Positionen für violette, 140 für grüne, 140 für orange und 140 für blaue Punkte. Zwei direkt nebeneinanderliegende Punkte erscheinen grundsätzlich weiß. (Für Orange stehen effektiv nur 139 Positionen zur Verfügung - ein oranger Punkt ganz rechts auf einer Zeile wird von WNDW' so abgeschnitten, daß er dunkelbraun erscheint.) Eine weitere Merkwürdigkeit: Wenn die beiden rechtesten Punkte einer Zeile beide gesetzt sind und außerdem noch verzögert ausgegeben werden, erscheinen sie nicht weiß, sondern blaugrün.

Wenn Sie sich um die Farbe nicht zu kümmern brauchen, können Sie sich ohne weitere Vorsichtsmaßnahmen in die Programmierung von 280 * 192 voneinander völlig unabhängigen Punkten stürzen. Mit einer einzigen Beschränkung ist sogar eine horizontale Auflösung von 560 Punkten möglich, Sie müssen nur berücksichtigen, daß ein verzögerter (und damit um eine halbe Position nach rechts verschobener) Punkt nicht zusammen mit unverzögerten Punkten innerhalb einer Siebenergruppe steht und umgekehrt. Diagonal über den Schirm verlaufende Linien lassen sich deshalb sehr hoch aufgelöst zeichnen, bei einer anfänglich zur Y-Achse parallel verlaufenden Parabel sinkt die Auflösung um so weiter, je mehr sie sich an die X-Achse anschmiegt, d.h. je "flacher" sie wird.

Falls Sie dagegen gerne Buntes auf dem Bildschirm haben, wird es erheblich komplizierter. Die verfügbare Auflösung beträgt 192 * 140 Punkte - solange sich Punkte verschiedener Farbe nicht allzu nahe kommen. Wenn Sie z.B. einen grünen Punkt in eine Siebenergruppe hineinzeichnen, wird der orange Punkt plötzlich ebenfalls grün, weil Bit 7 dieser Gruppe zurückgesetzt werden muß, um einen grünen Punkt zu erzeugen. Ähnlich erstaunliche Ergebnisse zeitigt ein Kombinationsversuch von Blau und Violett. Zwei direkt nebeneinanderliegende Punkte erscheinen dagegen unabhängig vom Stand von Bit 7 immer weiß.

Es ist bereits im Abschnitt "Die Zeitsignale der Bilderzeugung" erwähnt worden, daß die Verzögerung von VID7M durch ein gesetztes Bit 7 abgeschaltet werden kann, indem man FRCTXT' (via AN3 ($C05E)) auf "0" setzt. Tatsächlich unterscheiden sich die Ergebnisse des ersten und des zweiten Beispiels in Bild 8.13 dadurch - wenn man für das zweite Beispiel über FRCTXT' die Verzögerung abschalten würde, kämen dieselben Farben wie im ersten heraus, das gesetzte Bit 7 würde ignoriert. Aus Orange wird Grün, aus Blau wird Violett, und das gesamte Bitmuster rückt um eine halbe Punktposition nach links im Bildschirm.

Wieso erscheinen farbig gezeichnete Objekte in HiRes40 nicht "löcherig", obwohl mindestens jeder zweite Punkt abgeschaltet ist? Nun - auf einem monochromen Monitor oder einem Farbmonitor mit sehr hoher Auflösung sieht ein blau gezeichneter Vogel wirklich mehr wie eine Punktwolke aus. Ein normaler Fernseher dagegen hat zwar angeblich eine Bandbreite von 5 MHz - das heißt aber noch lange nicht, daß er seinen Elektronenstrahl wirklich mit 3.56 MHz an- und abschalten könnte. Aus diesem Grund verschwimmen zwei HiRes-Punkte gleicher Farbe mit einem dazwischenliegenden schwarzen Punkt zu einer relativ einheitlichen horizontalen Linie. Aus demselben Grund (wenn auch in diesem Fall mehr als unerwünscht) sind auf einem normalen Fernseher bereits im Modus TEXT40 die einzelnen Zeichen nicht mehr scharf konturiert. Je höher die Bandbreite des angeschlossenen Monitors, desto schärfer und lesbarer fällt die TEXT-Ausgabe aus - und umso "wolkiger" die Farbgrafik, weil hier die begrenzte Bandbreite für die Farbdeckung benutzt wird. In den 80er Modi fällt die Grenze noch erheblich deutlicher ins Auge: In LoRes80 und HiRes80 sind auch auf einem guten Farbfernseher keine einzelnen Punkte mehr erkennbar, TEXT80 ist dagegen an der Grenze zur Unleserlichkeit.

Die einzelnen Siebenergruppen passen innerhalb einer Zeile schön nebeneinander, wenn sie allesamt verzögert oder unverzögert sind. Die Probleme beginnen erst dann, wenn verzögerte auf unverzögerte Ausgaben folgen oder umgekehrt. Das dritte Beispiel in Bild 8.13 zeigt unsere mittlerweile sattsam bekannte Bitfolge 1011010, die hier auf geraden Adressen mit gesetztem Bit 7 (d.h. verzögert) und auf ungeraden Adressen mit zurückgesetztem Bit 7 gespeichert ist.

Am Anfang dieses Beispiels ist gerade eine unverzögerte Ausgabe beendet worden, VID7M ist also momentan in Phase mit 7M. Das nächste ausgelesene Videodatum hat Bit 7 gesetzt, was der HAL während PHASE1 * AX' * Q3' erkennt und deshalb eine verzögerte Ausgabe einleitet, LDPS' und VID7M finden eine Periode von 14M später statt. In der Zwischenzeit ist "Sendepause" - das Schieberegister erhält keinen neuen Taktimpuls und verlängert so das letzte Bit des vorherigen Bitmusters um die Breite eines halben Punktes. Generell: *Wenn auf ein "normales" Bitmuster eine verzögerte Ausgabe folgt, wird dadurch das letzte Bit des "normalen" Musters um eine halbe Punktposition verlängert.*

Der nächste Schritt in unserem Beispiel besteht im Laden und Hinausschieben des verzögerten Bitmusters. Das geht solange mit konstanter Geschwindigkeit vonstatten, bis der HAL das nächste Videodatum enthält. Hier ist Bit 7 wieder zurückgesetzt und der HAL registiert: "nicht verzögert". Konsequenterweise bleibt VID7M auf "0", LDPS' fällt mit der nächsten steigenden Flanke von 14M - und das zu einem Zeitpunkt, zu dem das letzte Bit der verzögerten Siebenergruppe erst für die Zeit einer halben Punktbreite ausgegeben wurde. Generell: *Wenn auf eine verzögerte Ausgabe ein normales Bitmuster folgt, wird dadurch das letzte Bit der verzögerten Ausgabe auf die Hälfte abgeschnitten.*

Der Grund, warum ich diese Tatsachen hier so stark in den Vordergrund stelle, liegt darin, daß die meisten HiRes40-Bitmuster nur dann vernünftig "aufeinanderpassen", wenn aufeinanderfolgende Siebenergruppen entweder beide "normal" oder beide verzögert ausgegeben werden. Ansonsten kommt es an den horizontalen Übergängen zwischen zwei verschiedenen Farben zu teilweise starken Rändern. Damit hat der unglückliche HiRes40-Programmierer eine weitere Spielwiese für Experimente und Aha-Erlebnisse, die einen manchmal schon zum HB-Männchen werden lassen können. Apple-Freaks wären allerdings nicht Apple-Freaks, wenn sie dieser Tatsache nicht auch einen positiven Aspekt abgewinnen könnten: Durch diese Interferenzen sind vertikale Linien in acht weiteren Farben möglich, die normalerweise in HiRes40 überhaupt nicht zur Verfügung stehen. Sie werden erzeugt, indem man einen Punkt auf der rechten Seite einer Siebenergruppe setzt und ihn über Bit 7 dieses und des darauffolgenden Bytes entweder um eine halbe Punktposition verlängert oder um denselben Betrag verkürzt.

Jede der LoRes-Farben außer Blaugrün (COLOR = 4) und dunklem Magentarot (COLOR = 1) kann auf einer begrenzten Anzahl von Horizontalpositionen erzeugt werden. Die LoRes-Farben 3, 6, C und 9 sind die normalen Äquivalente der HiRes40-Farben, 2 und 7 können zwischen geraden/ungeraden Adressen, D und 8 können zwischen ungeraden/geraden Adressen produziert werden, B und E zwischen beiden Arten von Adressfolgen. Die LoRes-Farbe E ergibt sich außerdem noch am rechten Bildschirmrand, die Farbe 8 zusätzlich durch einen orangen Punkt auf der rechtesten Horizontalposition.[6]

Bild 8.15 zeigt einige der Bitfolgen, die durch Interferenz zwischen verzögerten und unverzögerten Siebenergruppen entstehen können. Ein Vergleich dieser Darstellung mit Bild 8.10 im vorhergehenden Abschnitt zeigt, daß die Phasenverhältnisse zu COLOR REFERENCE dieselben wie in den LoRes-Modi sind - ausgenommen die Muster, in denen die erste Gruppe verzögert ist, mit 10 endet und durch 10 der folgenden unverzögerten Gruppe abgeschnitten wird. Die dabei entstehenden Farben Grün und Violett unterscheiden sich in ihrem Ton von den ansonsten gleichen LoRes-Farben.

Abgesehen von den seltenen Ausnahmen, wo man über diese Interferenzerscheinung tatsächlich einzelne Punkte oder Linien in Farben zeichnen kann, die normalerweise nicht zur Verfügung stehen, ist die ganze Angelegenheit doch reichlich lästig. Bei jeder Gelegenheit, an der zwei Farben zu nahe aneinander geraten, ergibt sich an der Grenzstelle eine dritte Wellenform mit einem anderen Phasenverhältnis zu COLOR REFERENCE. Nun - nachdem ich jetzt Ihr Bewußtsein für die Ursache dieser Erscheinungen genügend geschärft habe, sollten Sie eigentlich selber dazu in der Lage sein, mit Farbkombinationen zu experimentieren, die das Auge des Betrachters erfreuen.

Die Positionen ganz links und ganz rechts innerhalb einer Zeile stellen für verzögerte Ausgaben einen weiteren Spezialfall dar. Über ein Bitmuster mit gesetztem Bit 7 ist es möglich, das letzte Bit vor Anfang des dargestellten Bereichs um einen halben Punkt zu verlängern. Das einzige, was in diesem Fall passiert, ist eine Verbreiterung des linken Schwarzbereichs - HBL wird erst mit der fallenden Flanke von LDPS' inaktiv.

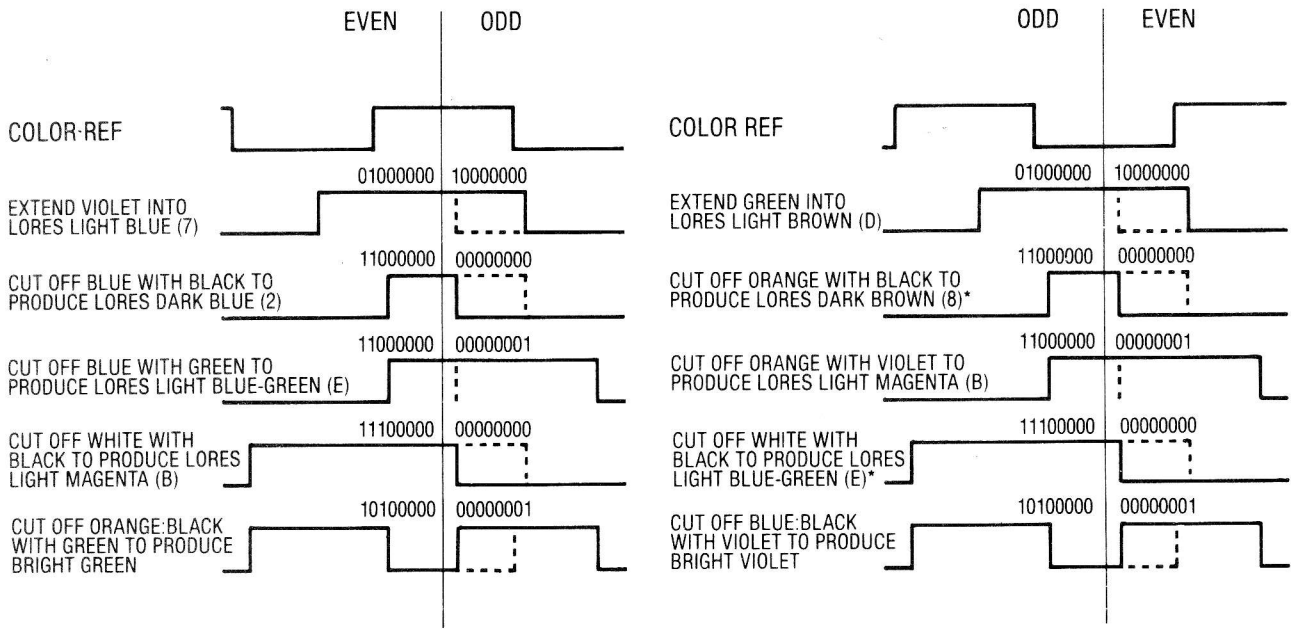
---

6 Die Querverweise auf LoRes-Farben gelten für Farbnummern bzw. Bitfolgen, die vom RAM der Hauptplatine gelesen werden. Vom AUX-RAM im Modus LoRes80 gelesene Werte erzeugen andere Farben (s. Tabelle 8.1).

Bei einer verzögerten Siebenergruppe ganz rechts innerhalb der Zeile sieht das schon etwas anders aus: hier wird das letzte Bit nach der Hälfte der Zeit durch HBL "abgesägt". Das liegt daran, daß die Ablaufsteuerung innerhalb des HAL dafür sorgt, daß die erste nicht mehr ausgegebene Bitgruppe im rechten Schwarzbereich des Bildschirms immer unverzögert ausgegeben wird - egal, welchen Wert VID7 innerhalb der UNUSED 8 hat. Der letzte verzögerte Farbpunkt ganz rechts innerhalb einer Zeile *könnte* orange sein, durch das Abschneiden nach der Hälfte der Zeit wird ein dunkles Braun daraus. Durch denselben Vorgang wird aus zwei hintereinanderfolgenden Punkten auf den beiden letzten Positionen innerhalb einer Zeile nicht weiß, sondern blaugrün.

In der Behandlung von verzögerten Ausgaben für die linke und die rechte Kante des Bildschirms ist der //e dem II(+) einen guten Schritt voraus: Im Videogenerator des II(+) ist es noch möglich, daß ein verzögert ausgegebenes Bitmuster auf der linken Seite einer Zeile das letzte Bit der UNUSED 8 (also des Bereichs, der nie ausgegeben wird) bis in den aktiven Bereich des Bildschirms hinein verlängert, ganz rechts im Bildschirm verzögert ausgegebene Bitmuster werden abhängig davon behandelt, ob das erste Datum im Schwarzbereich sein Bit 7 gesetzt hat oder nicht. Der //e verhält sich in diesen Punkten erheblich berechenbarer - schön wäre es allerdings gewesen, wenn sich ein Weg gefunden hätte, die Abschneiderei ganz zu vermeiden, ohne dabei gleichzeitig unverzögerte Bitmuster in die Länge zu ziehen.

Bild 8.15 Zeitdiagramme der Interferenzmuster an den Grenzstellen von verzögerten und unverzögerten HiRes40-Bitfolgen



* This is identical to the cut off of a delayed pattern by WNDW' at the right side of the screen.

Die Ausgabe von HiRes80

Das letzte Beispiel in Bild 8.13 zeigt (zum letzten Mal!) die Auslese des Bitmusters 1011010 im Modus HiRes80, dabei werdenabwechselnd der RAM der Hauptplatine und der AUX-RAM gelesen. Dieser Modus wird durch Zurücksetzen der Softswitches TEXT, AN3 (FRCTXT') und durch Setzen von HIRES und 80COL aktiviert. In derselben Weise wie die Umschaltung von TEXT40 auf TEXT80 funktioniert das auch im Modus HiRes: die Schiebegeschwindigkeit wird einfach verdoppelt, und Daten vom RAM der Hauptplatine und vom AUX-RAM wechseln sich innerhalb eines Videozyklus ab. Der Verzögerungsmechanismus für die Ausgabe wird durch den Pegel "0" auf FRCTXT' gesperrt, das Timing von LDPS' und VID7M ist damit in HiRes80 dasselbe wie in TEXT80.

Auch in diesem Modus entspricht das PICTURE-Signal genau der Bitfolge im Speicher, nur daß sie in der doppelten Geschwindigkeit ausgegeben wird und deshalb nur die halbe Breite einer HiRes40-Siebenergruppe auf dem

Bildschirm einnimmt. Auf einem monochromen Monitor mit hoher Bandbreite bleibt es bei dieser Komprimierung - auf einem farbfähigen Monitor oder einem Farbfernseher ergibt sich statt dessen ein ziemlich verwischtes und leicht ausgewaschen wirkendes Fleckchen in einer recht undefinierbaren Farbe - nicht nur, daß diese Frequenz für einen normalen Farbfernseher einfach zu hoch ist, 1011010 ergibt auch kein durchgehend konstantes Phasenverhältnis zu COLOR REFERENCE.

Für die Darstellung monochromer Grafiken stehen dem Programmierer im Modus HiRes80 560 anstelle von 280 Punkten pro Zeile zur Verfügung. Die damit verbundenen Vorteile sind so offensichtlich, daß wir an dieser Stelle nicht allzuviele Worte darüber verlieren wollen. Folglicherweise beschäftigt sich der Rest dieses Abschnitts hauptsächlich mit den weniger bekannten Möglichkeiten der farbigen Darstellung in HiRes80.

Eine komplette Periode von COLOR REFERENCE benötigt dieselbe Zeit wie die Ausgabe von zwei Punkten in HiRes40 oder die Ausgabe von 4 Punkten in HiRes80. Das Ergebnis für HiRes40 ist bereits bekannt: alternierende Bits (010 oder 101 etc.) erzeugen in diesem Modus eine Schwingung, deren Frequenz exakt mit der von COLOR REFERENCE übereinstimmt, und damit farbige Punkte auf dem Bildschirm. Für HiRes80 gilt dasselbe: auch hier müssen die Bitfolgen so aufgebaut sein, daß bei der Ausgabe Schwingungen von 3.56 MHz erzeugt werden. Mit der wiederholten Speicherung eines Bitmusters in aufeinanderfolgenden Adressen wie in Bild 8.13 ist es offensichtlich nicht getan: Wenn eine Periode von COLOR REFERENCE vier HiRes80-Punkten entspricht, dann kann das nicht funktionieren, wenn die Bitmuster in Siebenergruppen wiederholt werden. Statt dessen müssen wir die Bits in Vierergruppen aufteilen - wenn PICTURE eine beliebige Vierer-Folge außer 0000 oder 1111 mehr als einmal wiederholt, ergibt sich damit zwangsläufig ein Frequenzanteil von 3.56 MHz und eine definierte Farbe. Durchlaufende Vierer-Folgen über Gruppen zu je sieben Bit zu verteilen ist eine fürchterliche Arbeit für einen Programmierer - dafür können die Resultate aber auch sehr beeindruckend ausfallen.

Die zur Verfügung stehenden Farben sind exakt dieselben wie in LoRes (s. Bild 8.10), wenn es Ihnen Spaß macht, können Sie sogar LoRes40 simulieren, indem Sie dafür sorgen, daß von jeder Vierer-Folge exakt 14 Punkte in jeweils vier untereinanderliegenden Zeilen ausgegeben werden.

Diese Simulation hätte den Vorteil, daß die "LoRes-Blocks" frei im Bildschirm plazierbar sind (zeilenweise verschiebbar in der Höhe und punktweise nach links oder rechts), außerdem könnte man sie beliebig mit HiRes-Linien und -Figuren mischen. Trotzdem können wir den Designern des Apple dankbar sein, daß es einen eigenständigen LoRes-Modus gibt - darin kommt man nämlich mit erheblich weniger Speicherplatz und einem Bruchteil des Rechenaufwandes aus.

Im Modus LoRes ist die Zeit, für die eine ansonsten beliebige Vierer-Folge zu PICTURE geschoben wird, auf ganzzahlige Vielfache von 14 festgelegt. Diese Begrenzung entfällt in HiRes80: falls Sie Lust haben, Blocks mit acht Punkten Breite und zwei Zeilen Höhe zu erzeugen, steht dem nichts entgegen. Selbst einzeln ausgegebene Punkte bekommen die richtige Farbinformation mit - allerdings ist es ganz schön schwierig, die entsprechende Farbe auf dem Bildschirm zu erkennen, wenn der Punkt inmitten einer schwarzen Fläche steht.

In einer Siebenergruppe haben 1.75 Wiederholungen einer Vierer-Folge Platz. Damit die Folge bei der Ausgabe der nächsten Siebenergruppe korrekt fortgesetzt werden kann, muß sie in dieser Gruppe um ein Bit nach links rotiert gespeichert sein.

Die Folge 0010 0010 in einer Speicherstelle mit gerader Adresse im AUX-RAM erzeugt dieselbe Farbe wie 0100 0100 auf einer geraden Adresse im Hauptplatinen-RAM, 1000 1000 in einer ungeraden AUX-Speicherstelle oder 0001 0001 in einer ungeraden Speicherstelle des Hauptplatinen-RAMs. Das gesetzte Bit7 im dritten angegebenen Muster hat übrigens keinen Einfluß auf die Bildausgabe - es ist hier nur der Übersichtlichkeit halber notiert worden.

Je nachdem, aus welcher Speicherstelle eine Vierer-Folge gelesen wird (gerade/ungerade - AUX/Hauptplatine), erzeugt sie eine andere Farbe. Der direkte Zusammenhang mit LoRes40 ist nur bei geraden Adressen des Hauptplatinen-RAMs gegeben, eine komplette Aufstellung finden Sie in Tabelle 8.12.

Dem Programmierer stehen eine Vielzahl von Möglichkeiten zur Einteilung des HiRes80-Bildschirms zur Verfügung - neben der (programmtechnisch kompliziertesten) Methode, an beliebigen Stellen des Bildschirms Punkt für Punkt zu zeichnen, wäre z.B. ein "Super LoRes"-Modus vorstellbar, in dem ein Block einheitlicher Farbe eine beliebige Höhe (1-192), eine beliebige Breite (1-560), eine beliebige Farbe (0-15) und frei wählbare Koordinaten (X = 0-559, Y = 0-191) hat.

Tabelle 8.12 Äquivalente Farben in HiRes80 und LoRes

	EQUIVALENT HIRES80 PATTERNS			
LORES COLOR	AUX/EVEN	MBD/EVEN	AUX/ODD	MBD/ODD
$0 (0000)	$00 (0000 0000)	$00 (0000 0000)	$00 (0000 0000)	$00 (0000 0000)
$1 (0001)	$88 (1000 0000)	$11 (0001 0001)	$22 (0010 0010)	$44 (0100 0100)
$2 (0010)	$11 (0001 0001)	$22 (0010 0010)	$44 (0100 0100)	$88 (1000 1000)
$3 (0011)	$99 (1001 1001)	$33 (0011 0011)	$66 (0110 0110)	$CC (1100 1100)
$4 (0100)	$22 (0010 0010)	$44 (0100 0100)	$88 (1000 1000)	$11 (0001 0001)
$5 (0101)	$AA (1010 1010)	$55 (0101 0101)	$AA (1010 1010)	$55 (0101 0101)
$6 (0110)	$33 (0011 0011)	$66 (0110 0110)	$CC (1100 1100)	$99 (1001 1001)
$7 (0111)	$BB (1011 1011)	$77 (0111 0111)	$EE (1110 1110)	$DD (1101 1101)
$8 (1000)	$44 (0100 0100)	$88 (1000 1000)	$11 (0001 0001)	$22 (0010 0010)
$9 (1001)	$CC (1100 1100)	$99 (1001 1001)	$33 (0011 0011)	$66 (0110 0110)
$A (1010)	$55 (0101 0101)	$AA (1010 1010)	$55 (0101 0101)	$AA (1010 1010)
$B (1011)	$DD (1101 1101)	$BB (1011 1011)	$77 (0111 0111)	$EE (1110 1110)
$C (1100)	$66 (0110 0110)	$CC (1100 1100)	$99 (1001 1001)	$33 (0011 0011)
$D (1101)	$EE (1110 1110)	$DD (1101 1101)	$BB (1011 1011)	$77 (0111 0111)
$E (1110)	$77 (0111 0111)	$EE (1110 1110)	$DD (1101 1101)	$BB (1011 1011)
$F (1111)	$FF (1111 1111)	$FF (1111 1111)	$FF (1111 1111)	$FF (1111 1111)

Eine dritte Möglichkeit ist die Aufteilung in 140 * 192 Punkte mit 16 Farben (wobei Schwarz, Weiß, 1010- und 0101-Grau mitgezählt werden). Jede der Positionen wird durch vier HiRes-Punkte dargestellt und kann in einer frei wählbaren Farbe unabhängig von seiner Position und von seinen Nachbarn gezeichnet werden. Schräge Linien fallen dabei natürlich etwas zackig aus, was aber bei allen Farben außer den 1-Bit-Mustern (Farben 1,2,4 und 8) bis zu einem gewissen Grad ausgeglichen werden kann. Dabei ergeben sich 554 * 192 Zeichenpositionen für Vierer-Folgen. Der Rechenaufwand ist natürlich etwas größer als bei einer einfachen Zeichnung mit 140 * 192 Punkten, dafür verlaufen die Steigungen bei den Farben, die aus drei gesetzten Punkten bestehen, fast so sanft wie bei einer monochromen Darstellung mit 560 Punkten horizontaler Auflösung. Bei Grau könnte man sogar noch einen Schritt weitergehen, indem man mit jeder Zeile zwischen den beiden zur Verfügung stehenden Mustern wechselt. Bei den Farben 1,2,4 und 8 ist leider nichts mehr zu holen - sie sind wirklich nur in 140 verschiedenen Positionen innerhalb einer Zeile darstellbar.

Die in den letzten Abschnitten beschriebenen Programmiermethoden für farbige Darstellung in HiRes80 sind nur als sehr allgemeine Konzepte zu verstehen - sie könnten natürlich in erweiterter Form und mit einigen anderen Ideen gemischt einen Grundstock für eine Unterstützung aller Grafikmodi durch die Firmware des Apple //e abgeben.[7] Etwas Entsprechendes wäre mehr als nötig: bis jetzt unterstützt die Firmware des //e nur die 40er Modi für die Grafik.

Die Umschaltungen im Modus MIXED

Das ist das letzte Thema, mit dem wir uns bei einer Besprechung der Grafikausgabe des Apple //e beschäftigen müssen - dieser Hin- und Herschalterei verdanken wir schließlich die Notwendigkeit für die verzögerten GRAPHICS-Signale GR+1 und GR+2.

Bild 8.17 ist ein Zeitdiagramm des letzten Bildausgabe-Zyklus von Zeile 159 im Modus MIXED. Auf der linken Seite dieses Diagramms schalten HPE und H5.-.H0 des Videoscanners gerade von 1111111 auf 0000000 und leiten so HBL ein. Im selben Moment, in dem die Horizontalsektion des Scanners den Stand 0 erreicht, schaltet die Vertikalsektion auf 110100000, der Term V4 * V2 wird "wahr". Das bedeutet, daß die Zeitabläufe nunmehr auf TEXT

7 Einblicke in die Funktionsweise von HiRes80 von einer anderen Seite aus, zusammen mit den Assemblerlistings für einige Zeichenroutinen in diesem Modus, vermittelt der im Januar 1984 in *Apple Orchard* erschienene Artikel "True Sixteen-Color Hi-Res" von Allen Watson III.

umgeschaltet werden sollen, was aber im Moment noch nicht möglich ist, weil die letzte GRAPHICS-Bitfolge überhaupt noch nicht in das Schieberegister geladen worden ist. Aus diesem Grund sollte auch die Bildausgabe noch nicht auf Schwarz gesetzt werden.

Die Lösung dieses Dilemmas ist einfacher als man denkt: Alle Umschaltungen von GRAPHICS werden um zwei Taktimpulse des Videoscanners verzögert (s. GR, GR+1 und GR+2 in Bild 8.5). Von dieser Verzögerung sind die IOU-Ausgänge SEGA, SEGB, RA9, RA10 und natürlich GR+2 selber betroffen, sowohl der HAL als auch der Video-RAM erhalten dadurch ihre Umschaltsignale bzw. neuen Adreßbits erst dann, wenn der letzte Bildausgabezyklus abgeschlossen ist.

Der Ablauf der in Bild 8.17 dargestellten Ereignisse:

1. RAS' steigt während PHASE1 (Taktimpuls für den Videoscanner), der neue Zählerstand ist 110100000/0000000 nach interner IOU-Laufzeit. V4 * V2 wird "wahr" (=> "TEXT Time").
2. Ungefähr zur selben Zeit, in der der Videoscanner seinen Zustand ändert, erfolgt die steigende Flanke von PHASE0, das letzte Videodatum wird in den Latches festgehalten.
3. In der 80er Modi wird LDPS' während PHASE0 aktiv und lädt das letzte durch Daten des AUX-RAMs erzeugte Bitmuster dieser Zeile in das Schieberegister.
4. LDPS' wird während PHASE1 aktiv und lädt das letzte durch Daten des Hauptplatinen-RAMs erzeugte Bitmuster in das Schieberegister. Unabhängig davon, ob einfache oder doppelte Auflösung selektiert wurde, ist dieses Muster das letzte, das innerhalb dieser Zeile ausgegeben wird, bevor der rechte Schwarzbereich beginnt.
5. RAS' steigt während PHASE1, daraufhin fällt GR+1 und WNDW' geht auf "1". GR+1 setzt die interne Umschaltung von SEGA-SEGC auf die Auswahl für TEXT (d.h. VA-VC), die entsprechenden Pegelwechsel auf den Leitungen SEGA-SEGC finden allerdings erst zusammen mit dem nächsten Taktimpuls des Videoscanners statt. Die Umschaltung wird also erst mit zwei Taktzyklen Verzögerung wirksam. GR+1 hat noch eine weitere Funktion: es sperrt HIRES innerhalb der Adresserzeugung des Videoscanners, die nächste Adresse wird damit für den Speicherbereich TEXT/LoRes erzeugt.
6. Im selben Zeitraum, in dem GR+1 auf "0" geht, steigt PHASE0 und hält damit das erste Videodatum fest, das nicht mehr auf dem Bildschirm dargestellt wird. Wenn HIRES gesetzt ist, dann kommt dieses Datum noch aus einem der HiRes-Bildspeicherbereiche, weil HIRES TIME erst zu diesem Zeitpunkt inaktiv wird.
7. Die letzten Bits des letzten Bitmusters werden hinausgeschoben, bis LDPS' erneut aktiv wird,[8] das Schieberegister mit 1111 1111 beladen wird und so PICTURE auf Schwarzpegel setzt.
8. RAS' steigt während PHASE1, als Folge wird GR+2 zurückgesetzt. SEGA-SEGC, RA9-10 und GR+2 sind jetzt alle auf TEXT gestellt. Durch die Auslösung über den Taktimpuls des Videoscanners kann das erst dann geschehen, wenn das letzte Bitmuster komplett hinausgeschoben wurde.
9. Ungefähr zum selben Zeitpunkt, zu dem GR+2 fällt, steigt PHASE0 und hält so das zweite Videodatum fest, das ebenfalls nicht ausgegeben wird. Selbst dann, wenn HIRES gesetzt ist, kommt dieses Datum bereits aus dem Bereich TEXT/LoRes, weil HIRES TIME mittlerweile für mehr als eine Mikrosekunde zurückgesetzt ist.

Soweit es die Adressierung des Video-ROMs betrifft, ist eine Verzögerung der TEXT-Umschaltung um einen Taktimpuls des Scanners ausreichend, weil das letzte Bitmuster danach bereits in das Schieberegister geladen worden ist. Für die Steuereingänge des HAL wäre das allerdings zuwenig: LDPS' und VID7M müssen im Modus GRAPHICS bleiben, bis das letzte Bitmuster der Zeile komplett hinausgeschoben und das Schieberegister mit "Einsern" beladen worden ist. Aus diesem Grund ist die Verzögerung um zwei Taktzyklen an SEGB und GR+2 eine Notwendigkeit.

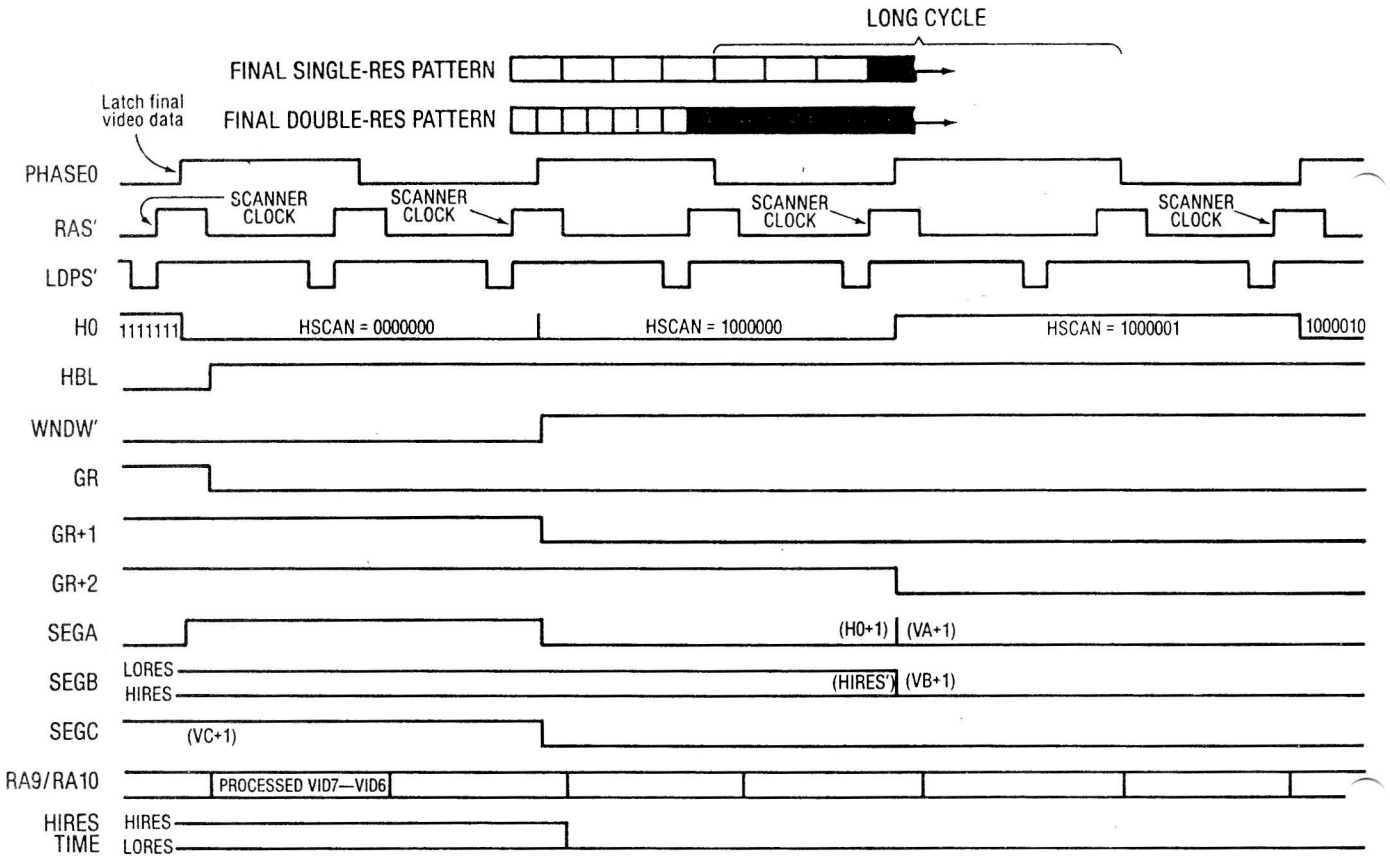
---

8 Durch diesen Ladevorgang wird letztendlich in HiRes40 das letzte Bit einer verzögert ausgegebenen Siebenergruppe abgeschnitten, WNDW' hat die Ausgänge des Video-ROMs bereits als Folge des letzten Scanner-Taktimpulses abgeschaltet. Das Abschneiden findet also nur indirekt durch WNDW' statt - as.

Ein ideales Zeitverhalten für die Umschaltung zwischen GRAPHICS und TEXT in den 40er Modi ließe sich dadurch erreichen, daß die ROM-Adreßbits durch GR+1 und die Steuereingänge des HAL durch GR+2 geschaltet würden. Das wäre im Apple //e allerdings nur dann möglich, wenn der HAL für HIRES und GRAPHICS jeweils eine eigene Termdefinition hätte. In der momentanen Ausführung ist das nicht der Fall, alle Terme werden über GR+2 geschaltet.[9] Das erzeugt ein "Glitching" im Bild, wenn ein Programm während der aktiven Bildausgabe über die Softswitches C050/C051 zwischen GRAPHICS und TEXT umschaltet.

Die Rückschaltung von TEXT auf GRAPHICS findet zu Beginn von HBL vor der ersten aktiven Fernsehzeile statt, die Abläufe sind dieselben wie in Bild 8.17 dargestellt. Sie sind wesentlich unkritischer, weil WNDW' während der gesamten Umschaltung noch inaktiv ("1") ist. Außerdem findet noch während VBL eine doppelte Umschaltung zwischen GRAPHICS und TEXT statt, die bereits in Kapitel 5 erwähnt wurde. Sie hat keine weiteren Konsequenzen für die Bildausgabe, auch während dieser Zeit ist WNDW' durchgehend inaktiv.

Bild 8.17 Die Umschaltung von GRAPHICS auf TEXT im Modus MIXED



9 HIRES TIME wird im Adressgenerator des Videoscanners durch GR+1 umgeschaltet. Aufgrund der Zugriffszeiten der RAMs erfolgt dadurch die Umschaltung des Videodatenbusses zwischen HiRes- und TEXT/LoRes-Videodaten zum selben Zeitpunkt wie die Umschaltung der Video-ROM-Adressen durch GR+2.