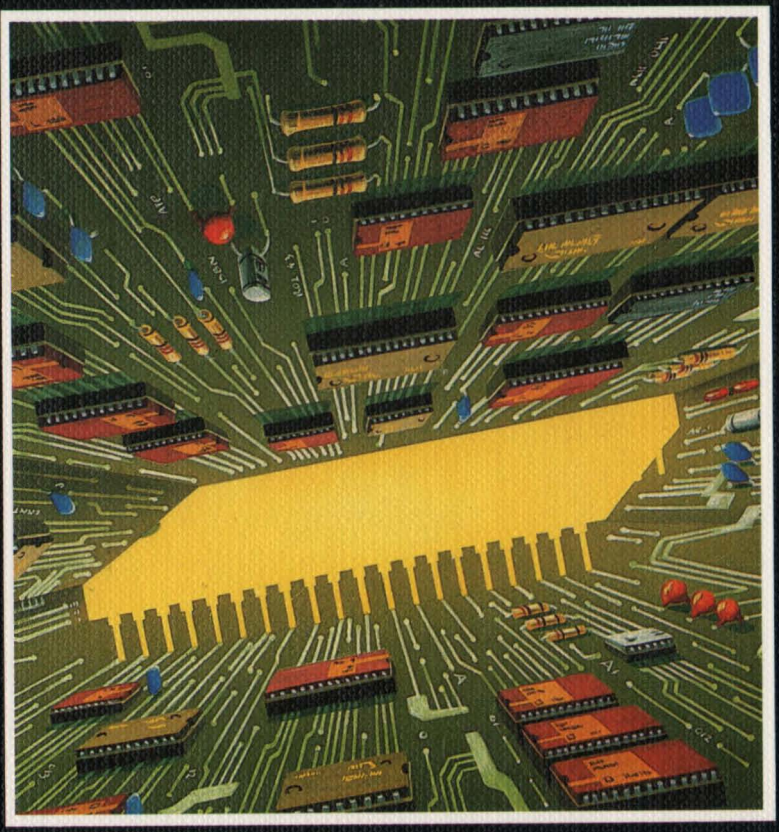
SYBEX

Z80 Anwendungen



James W. Coffron

Z80 Anwendungen

James W. Coffron

BERKELEY · PARIS · DÜSSELDORF

Originalausgabe in Englisch
Titel der englischen Ausgabe: „Z80 Applications"
Original Copyright © 1983 by SYBEX Inc., Berkeley, California, USA

Deutsche Übersetzung: Winfried Wolf
Umschlagentwurf: Jean François Penichoux
Satz: tgr – typo-grafik-repro gmbh, remscheid
Gesamtherstellung: Boss-Druck und Verlag, Kleve

ISBN 3-88745-037-X
1. Auflage 1984
2. Auflage 1985
3. Auflage 1985
4. Auflage 1987

Inhalt

Einleitung

Ist es Ihnen schon passiert, daß Sie eine neue Anwendungsidee für ein Mikroprozessor-gesteuertes System hatten, die dann in einem Wirrwarr von technischen Details aus Datenblättern unterging? Wenn ja, dann sind Sie nicht der einzige. Eine Reihe hervorragender Anwendungsideen für Mikroprozessor-gesteuerte Systeme haben dieses Schicksal erlitten. Die Idee war gut, aber die Person mit dieser Idee hatte nicht genug Wissen über die Details des Mikroprozessors, um das Projekt voll zu verwirklichen.

Dieses Buch ist geschrieben worden, um Benutzern von Z80-Systemen aus diesem Dilemma herauszuhelfen. Es ist entstanden, um den Anwender mit genügend Verständnis für den Z80-Mikroprozessor auszurüsten. Allgemein gesagt, erklärt es detailliert, wie der Z80-Mikroprozessor mit der System-Hardware verbunden wird, um ein komplettes System zu bilden. Im einzelnen werden dabei ROMs, statische RAMs, dynamische RAMs und Eingabe/Ausgabe besprochen. Im weiteren werden dann einige spezielle I/O-Bausteine untersucht. Einige dieser Bausteine, wie das SIO (serial input/output) und das PIO (peripheral input/output), sind speziell zur Verwendung mit dem Z80 entwickelt worden, während andere, wie der Timer 8253 und der Peripheriebaustein 8255, das nicht sind.

Geschrieben sowohl für den Neuling als auch für den erfahrenen Programmierer, enthält dieses Buch eindeutige Beschreibungen, klare, präzise Diagramme und vollständige Beispiele. Der Anwender soll die entsprechenden Informationen zur Realisierung seiner eigenen Anwendungen und Ideen bekommen. Natürlich zeigt dieses Buch nicht alle Anwendungsmöglichkeiten des Z80 – die individuellen Anwendungen müssen von Ihnen kommen –, es bietet jedoch die Informationen zur Realisierung jeglicher Anwendung.

Inhalt

Kapitel 1, *Benutzung von ROMs in Z80-Systemen,* beginnt mit einer Vorstellung der verschiedenen Typen von Nur-Lese-Speichern. Es zeigt

dann, wie übliche ROM- und EPROM-Bausteine an eine Z80-CPU angeschlossen werden.

Kapitel 2, *Statische RAMs in Z80-Systemen,* zeigt, wie statische Schreib-/Lesespeicher in einem Z80-System eingesetzt werden. Es beschreibt sowohl Common-I/O- als auch Separate-I/O-RAMs und stellt die vollständigen Schaltungen zweier RAM-Systeme vor.

Kapitel 3, *Eingabe und Ausgabe beim Z80,* beschreibt, wie der Z80 elektrisch mit Eingabe- und Ausgabebausteinen kommuniziert. Standard-Ein/Ausgabe-Ports werden vorgestellt und detailliert erklärt.

Kapitel 4, *Dynamische RAMs in Z80-Systemen,* zeigt, wie dynamische RAM-Systeme mit dem Z80 funktionieren. Das Kapitel beginnt mit einer Beschreibung von typischen dynamischen RAM-Bausteinen. Es fährt damit fort, die Verbindung zwischen RAM und Z80 zu erklären und berücksichtigt dabei die Besonderheiten der Z80-Architektur.

Kapitel 5, *Interrupts beim Z80,* erklärt die wichtigsten Eigenschaften von Interrupts. Die drei Interrupt-Betriebsarten werden untersucht und für jeden Typ Beispiele gegeben.

Kapitel 6, *Der PIO-Baustein 8255 in Z80-Systemen,* untersucht die Verwendung des Peripherie-I/O-Chips 8255- ein Baustein, der an den Z80-Bus angeschlossen wird. Die einzelnen Operationsarten werden besprochen und viele Programmierbeispiele gegeben.

Kapitel 7, *Der programmierbare Timerbaustein 8253,* zeigt, wie der programmierbare Timerbaustein 8253 zusammen mit dem Z80 benutzt werden kann. Die Verbindung zwischen diesem Baustein und dem Z80 wird erklärt. Außerdem wird eine Reihe von Programmbeispielen vorgestellt.

Kapitel 8, *Der Z80-PIO-Baustein,* fährt mit der Betrachtung von Peripherie-Chips fort. Das Blockschaltbild des Z80-PIO wird vorgestellt sowie die Operationsarten und die Register. Es folgt eine Reihe von Programmbeispielen unter Verwendung des Z80-PIO.

Kapitel 9, *Benutzung des Z80-CTC,* beschreibt den Zähler/Zeitgeber-Baustein und zeigt seine Verbindung mit dem Z80. Viele Programmbeispiele für die Benutzung des CTC werden besprochen.

Kapitel 10, *Einführung in die serielle Kommunikation,* beschreibt die Eigenschaften serieller Kommunikation. Begriffe wie Baudrate, Startbit, Stopbit und Markierungspegel werden definiert. Als praktisches Beispiel wird gezeigt, wie das USART 8251 mit dem Z80 verbunden und betrieben wird.

Kapitel 11, *Benutzung des Z80-SIO,* fährt in der Beschreibung serieller Kommunikation mit dem Z80-SIO-Chip fort. Verschiedene praktische Beispiele für den Einsatz und die Programmierung des Z80-SIO werden vorgestellt.

Kapitel 12, *Statische Testmethode für Z80-Systeme,* schließt diesen Text mit einer Beschreibung der SST-Technik (Static Stimulus Testing) für den Z80 ab. Es zeigt, wie diese Methode benutzt werden kann, ein System auch ohne Software elektrisch zu prüfen und Fehler zu finden. Sie ermöglicht Ihnen weiter, bei Einbau eines neuen Interface in ein vorhandenes Z80-System dieses schnell und einfach zu prüfen.

Abschließend gibt Anhang A eine vollständige Beschreibung aller Operationen der internen Register des Z80-SIO.

Wie Sie sehen, bietet dieses Buch alle wichtigen Punkte zum Verständnis des Z80-Mikroprozessors. Mit diesem Wissen sollten Sie in der Lage sein, jede Idee für ein Mikroprozessor-gesteuertes System auch zu realisieren. Wenn Sie mit der Implementation eigener Ideen und Anwendungen beginnen, werden Sie feststellen, daß die Benutzung des Z80-Mikroprozessors nicht immer leicht fällt, jedoch auch viel Freude machen kann. So lassen Sie Ihrer Phantasie freien Lauf und lernen Sie, wie Sie den Z80 dazu bringen, für Sie zu arbeiten.

Kapitel 1

Benutzung von ROMs in Z80-Systemen

Einführung

Wir wollen unsere Studie des Z80-Mikroprozessors damit beginnen, zu lernen, wie er mit Nur-Lese-Speichern (ROMs) verbunden wird. Wir fangen mit einer allgemeinen Einführung in die Arbeitsweise von ROMs an und untersuchen dann die wichtigen Aspekte im Zusammenspiel mit dem Z80. Am Ende des Kapitels sollten Sie vollständig verstanden haben, wie ROMs elektrisch mit dem Z80 kommunizieren, und in der Lage sein, dieses Wissen bei der Schaltungskonzeption einzusetzen.

Was ist ein ROM?

ROMs sind in den meisten Mikroprozessor-Systemen anzutreffen. Es sind Speicher, deren Informationsinhalt zwar gelesen, jedoch nicht geändert werden kann und diesen auch beim Ausschalten des Systems nicht verlieren. Sie erlauben der Zentraleinheit (CPU) beim Einschalten des Systems die periphere Hardware zu initialisieren.

Es gibt mehrere Arten nicht-flüchtiger Speicher, die in Mikroprozessor-Systemen benutzt werden können. Eingeschlossen sind Festwertspeicher (ROMs), programmierbare Festwertspeicher (PROMs), lösch- und programmierbare Festwertspeicher (EPROMs) und elektrisch änderbare Festwertspeicher (EAROMs) (siehe Abb. 1.1). Normalerweise wird in einem System nur jeweils einer dieser Typen eingesetzt. Da Ihre Anwendung natürlich einen beliebigen Typ vorsehen kann, wollen wir alle näher beschreiben:

ROM

Das ROM wird bereits beim Hersteller fest programmiert. Es wird dort eingesetzt, wo ein Datenträger in großen Stückzahlen benötigt wird.

PROM

Das PROM wird vom Anwender programmiert. Dabei werden durch entsprechend hohe Spannungsimpulse Brücken aus Metall oder polykristalli-

nem Silizium innerhalb der integrierten Schaltung weggebrannt und damit logisch 1 oder 0 erzeugt. Einmal programmiert, können die Daten nicht geändert werden. PROMs sind die weitaus schnellsten programmierbaren Speicher.

EPROM

Das EPROM kann ähnlich wie das PROM vom Anwender durch Anlegen entsprechender Spannungsimpulse programmiert werden. Die Daten können jedoch auch wieder gelöscht werden und zwar durch Bestrahlung des Chips durch ein Quarzglasfenster im Gehäuse mit ultraviolettem Licht. Nach einer bestimmten Zeit sind alle Daten gelöscht, und das Bauteil kann neu programmiert werden. Das EPROM wird bevorzugt bei Entwicklungsarbeiten eingesetzt.

EAROM

Das EAROM wird ähnlich wie das EPROM programmiert. Gelöscht wird es jedoch elektrisch statt mit UV-Licht.

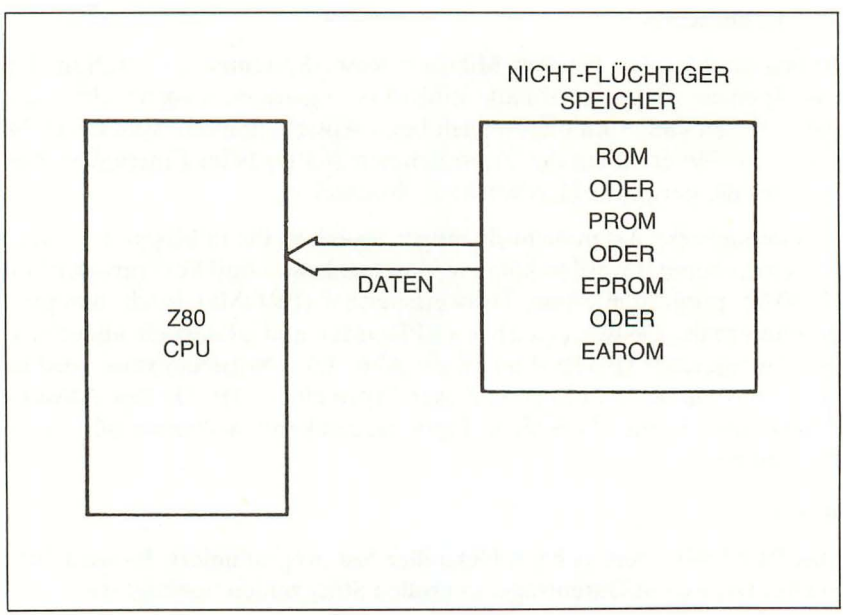


Abb. 1.1: *Dieses Blockschaltbild zeigt eine typische Anwendung von nicht-flüchtigen Speichern in einem Mikroprozessor-System. Der nicht-flüchtige Speicher enthält sinnvollerweise sog. „boot up"-Programme, die beim Einschalten des Systems ausgeführt werden.*

Gleich welchen Typ Sie in Ihrer Anwendung einsetzen: Die Arbeitsweise ist weitgehend identisch. Wir wollen jetzt die wichtigsten Parameter dieser Gruppe von Speichern untersuchen. (Anmerkung: Im weiteren werden wir der Einfachheit halber alle nicht-flüchtigen Speicher als ROM bezeichnen.)

Die wichtigsten Merkmale des ROM

Das ROM kann, wie sein Name ja schon sagt, von der CPU nur gelesen werden. Die gespeicherte Information ist verfügbar, sobald eine Adresse an den Adreß-Eingangs-Leitungen angelegt wird. Die Anzahl der Adreßeingänge ergibt sich aus der internen Organisation des ROM. Übliche Speicherorganisationen sind z.B. 1024 x 8, 2048 * 8 oder 4096 * 8. Die erste Nummer gibt die Anzahl der Adressen an, die zweite Nummer die Anzahl der mit einer Adresse angesprochenen Bits.

Die Anzahl der Adreßleitungen, die benötigt werden, um z.B. ein 2048 * 8 ROM anzusteuern, ergibt sich aus $2^x = 2048$ mit $x=11$ als gesuchte Größe. Bei einem 4096 * 8 ROM gilt dementsprechend $2^x = 4096$ mit $x=12$.

Bei der Bezeichnung eines ROM wird i.a. der Adreßumfang auf den nächsten Tausender abgerundet und dieser zusammen mit dem Buchstaben „K" angegeben. Mit anderen Worten: Ein 1024 * 8 ROM wird als 1K Baustein bezeichnet, ein 4096 * 8 ROM als 4K Baustein usw.

Als letztes wäre noch zum ROM zu sagen, daß die Daten parallel ausgelesen werden. Die Datenbits der angesprochenen Adresse werden also gleichzeitig auf den Datenbus gelegt und dann vom Mikroprozessor in seine internen Register eingelesen.

Abb. 1.2 zeigt ein Funktions-Blockdiagramm der ROM-Arbeitsweise. Den Eingang „Bausteinauswahl" (chip select), der dort zu sehen ist, haben wir bis jetzt noch nicht besprochen. Die Funktion dieser Eingangsleitung ist es, die Datenausgänge des ROM ein- und auszuschalten. Wenn sie aktiv ist, so werden die Daten, logisch 0 oder 1, auf die Ausgänge gelegt. Ist sie nicht aktiv, so befinden sich die Ausgänge in einem hochohmigen Zustand (tri-state). Mit anderen Worten, die Baustein-Auswahl-Leitung gibt die Ausgänge des ROM frei bzw. sperrt sie. Später werden wir noch genauer zeigen, wie diese Leitung zu benutzen ist.

Der Ereignisablauf beim Lesen von Daten aus dem ROM

Betrachten wir das Blockschaltbild in Abb. 1.2. Eine Folge von elektrischen Ereignissen wird bei jedem Lesen des Bausteins veranlaßt. Dabei

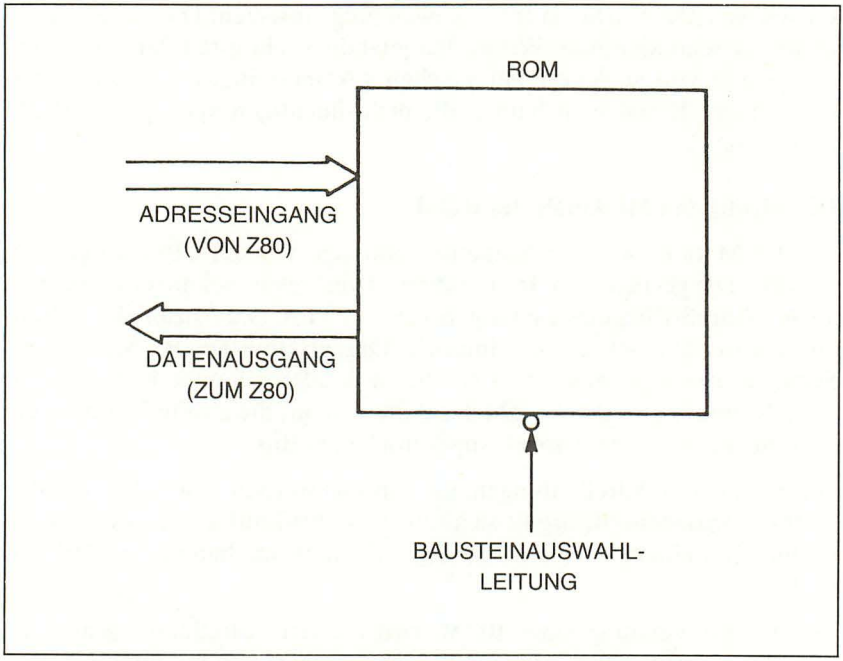


Abb. 1.2: *Das Funktions-Blockschaltbild eines typischen ROM-Bauteils. Die Adreßeingänge selektieren die internen Daten, die ausgegeben werden sollen. Die Daten werden auf die Datenausgangsleitungen gelegt, wenn das Bauteil über die Baustein-Auswahl-Leitung (chip select) selektiert wird.*

können wir davon ausgehen, daß der folgende Ablauf unabhängig von der benutzten CPU ist. (Später werden wir untersuchen, wie der Z80 speziell diesem Ablauf folgt.)

1. Die CPU gibt eine Adresse an das ROM aus. Sie wird vom ROM benutzt, um eines der möglicherweise Tausenden von gespeicherten Bytes anzuwählen.

2. Die CPU wartet jetzt für eine bestimmte Zeit, genannt Zugriffszeit (access time), die i.a. 100 bis 300 Nanosekunden beträgt (abhängig vom verwendeten ROM-Typ), um dem Speicher zu ermöglichen, die Adresse intern zu decodieren und die gewünschte Information bereitzustellen.

3. Das Baustein-Auswahl-Signal wird aktiviert und damit die Daten auf den System-Datenbus gelegt. Sie stehen somit auch an den Datenein-

gangsleitungen der CPU. Als nächstes speichert die CPU die Daten in eines ihrer internen Register.

4. Das Baustein-Auswahl-Signal wird deaktiviert und somit die ROM-Daten vom Datenbus weggeschaltet.

Diese Sequenz wird jedesmal durchlaufen, wenn die CPU Daten vom ROM liest. Abb. 1.3 zeigt das verallgemeinerte Zeitdiagramm.

Mikroprozessoren sind so ausgelegt, daß sie diese Richtlinien einhalten. Deshalb braucht der Anwender sich um diesen Ablauf nicht zu kümmern; die CPU benutzt dafür interne Zeitsteuer-Schaltungen. Es ist aber wichtig, daß Sie den Ablauf verstehen, was Ihnen den Einsatz von ROMs wesentlich erleichtert.

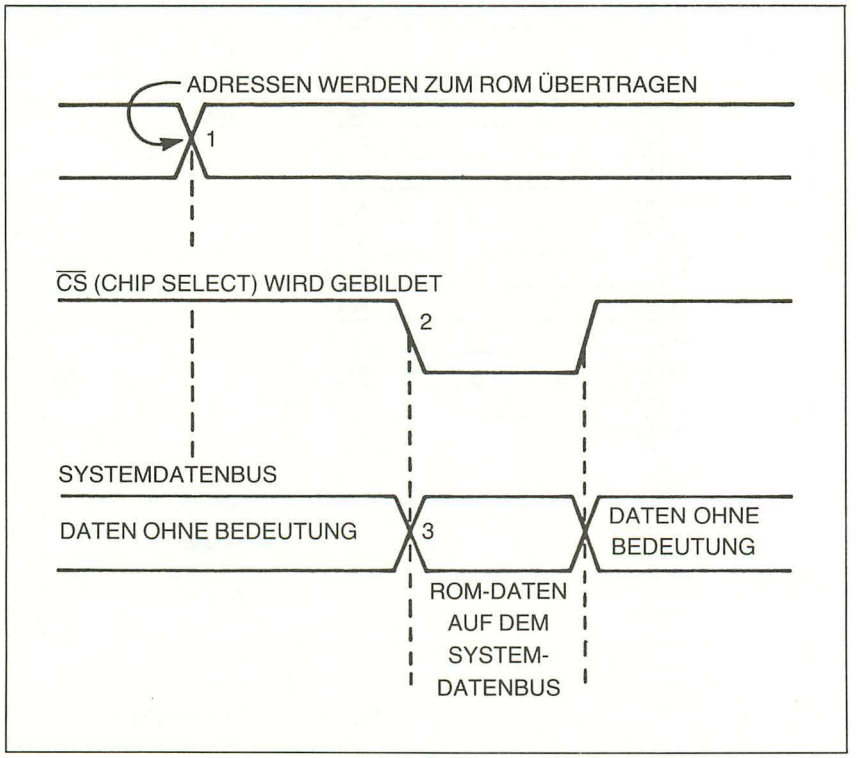


Abb. 1.3: *Ein verallgemeinertes Timing-Diagramm für das Lesen von Daten aus dem ROM:*

1. *Die Adresse von der CPU wird in das ROM eingelesen.*
2. *Das Baustein-Auswahl-Signal (chip select) wird angelegt.*
3. *Das ROM legt die Daten auf den Datenbus.*

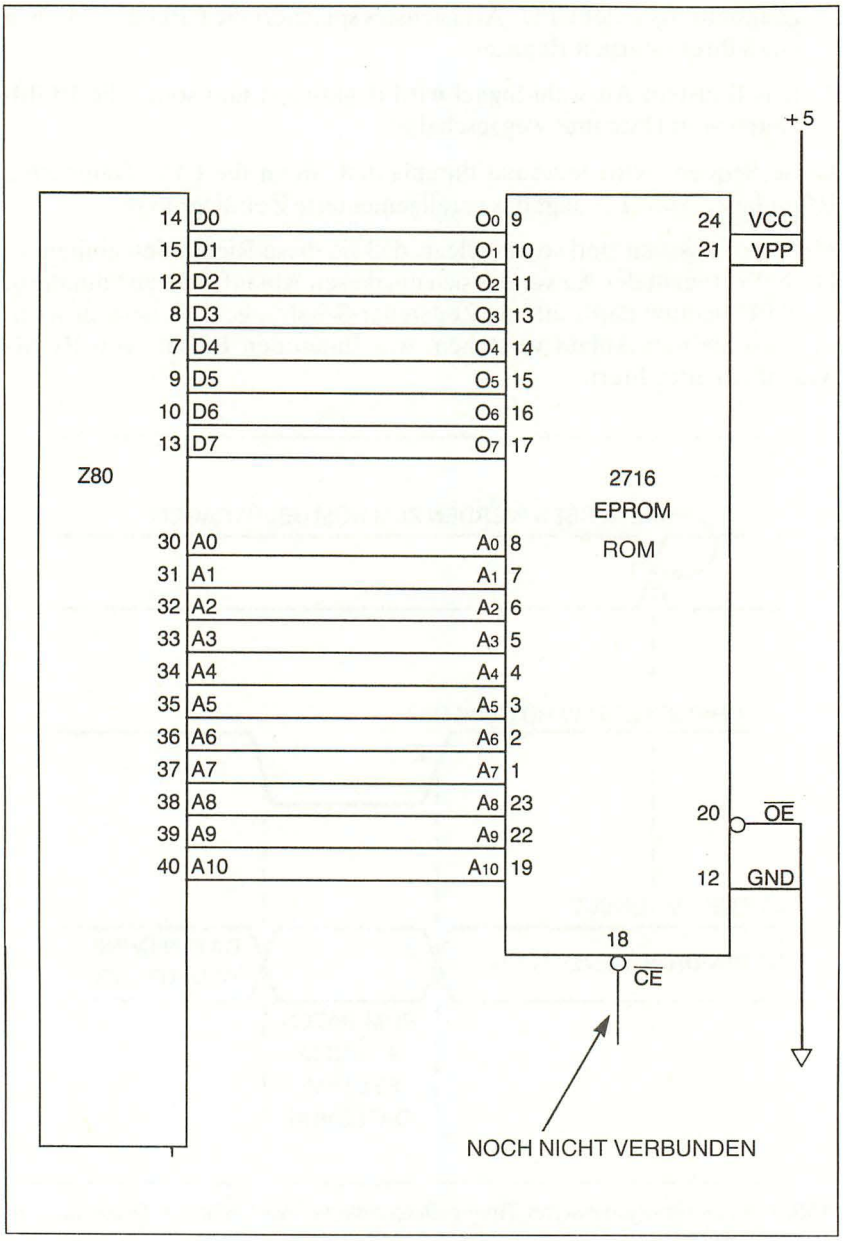


Abb. 1.4: *Schematische Darstellung der Verbindung zwischen einem Z80-Mikroprozessor und einem 2716-EPROM mit ungepufferten Daten- und Adreßleitungen.*

Das Verbinden der Z80-Bus-Signale

Wir wollen jetzt den Z80 mit dem ROM verbinden. Abb. 1.4 zeigt die physikalische Verbindung der Z80-Daten- und Adreßleitungen mit denen des ROM. Sie ist äußerst einfach. Der Baustein-Auswahl-Eingang ist in Abb. 1.4 nicht beschaltet, da wir erst einmal über die seitenweise Adressierung sprechen wollen.

Seitenweise Adressierung (address mapping)

Der Z80 hat 16 physikalische Adreßausgänge, A0 bis A15. Damit kann er $2^{16} = 65536$ verschiedene Speicherplätze direkt adressieren. Ein 2716-EPROM z.B. hat 2048 Speicherplätze. Es muß nun eine Methode gefun-

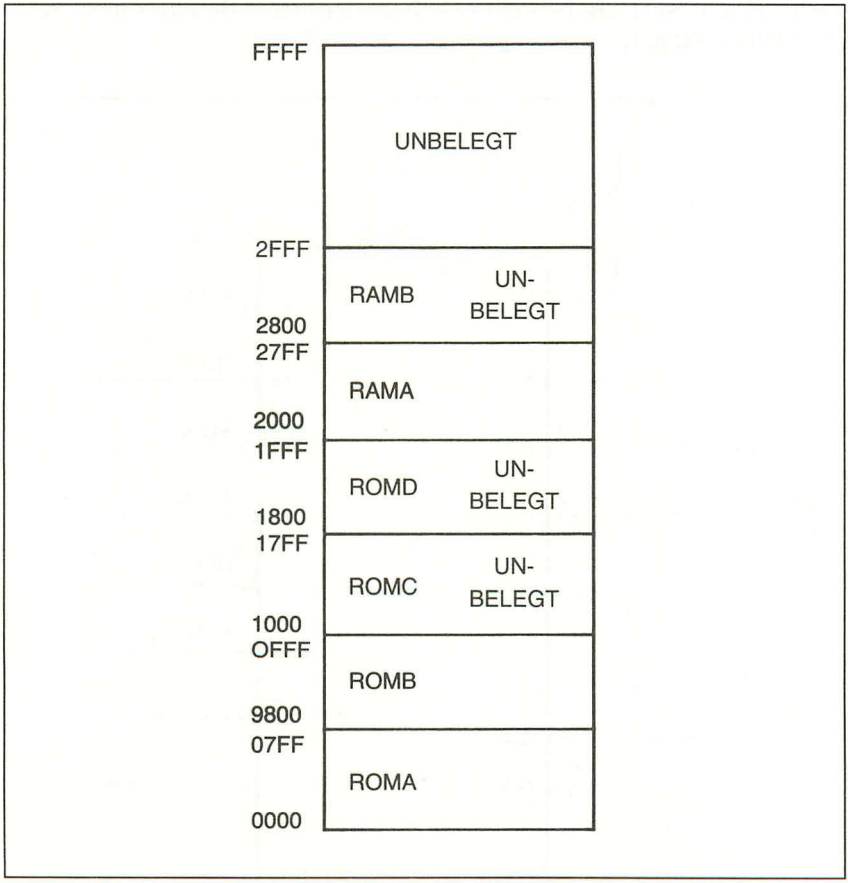


Abb. 1.5: *Ein Speicherbelegungsplan (memory map) eines typischen Mikroprozessor-Systems.*

den werden, diese 2048 Adressen aus dem Gesamtbereich von 65536 bzw. 64K Adressen auszuwählen.

Bevor die Hardware für das System aufgebaut wird, sollte zuerst ein Speicherbelegungsplan (memory map) erstellt werden. In ihm wird festgelegt, welche Adreßbereiche für ROM-, RAM-, Eingangs- und Ausgangsbausteine verwendet werden. Abb. 1.5 zeigt so einen typischen Speicherbelegungsplan. Wir sehen dort, daß der gesamte Adreßbereich in Blöcke unterteilt ist, die jeweils für ein bestimmtes ROM oder RAM Speicherplatz reservieren.

Anmerkung: Der unterste Adreßbereich (0000 hexadezimal) wird sinnvollerweise für das ROM reserviert. Der Grund dafür ist, daß der Z80 nach einem Reset (z.B. beim Einschalten) den ersten Befehl von der Speicherstelle 0000 holt.

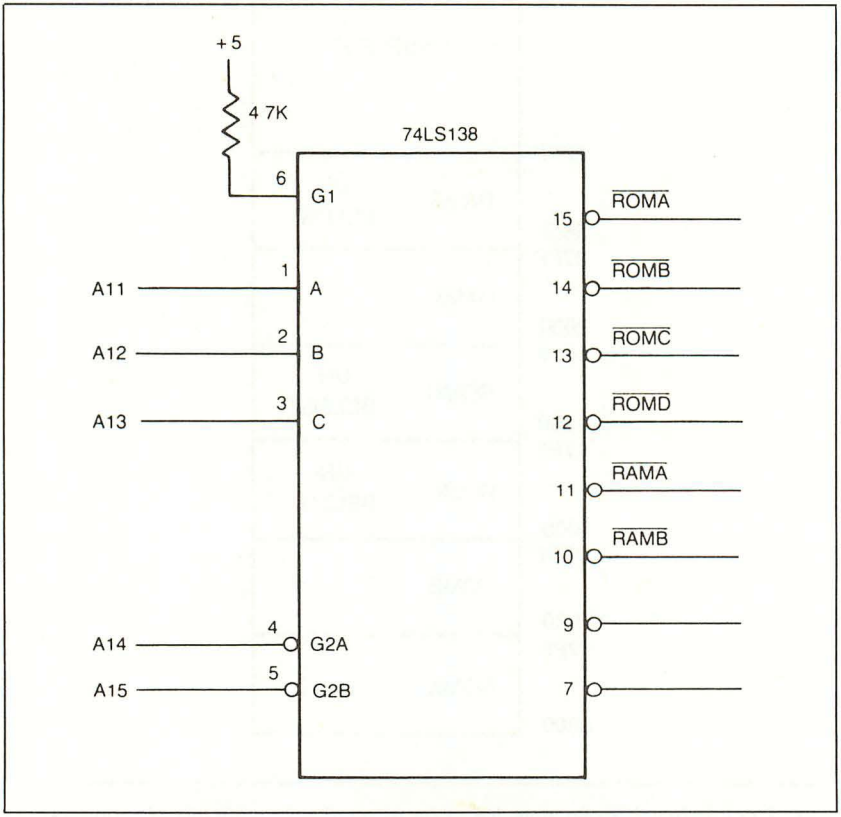


Abb. 1.6: *Eine Hardware-Schaltung, die den Speicher gemäß Abb. 1.5 aufteilt.*

Erzeugung der Baustein-Auswahl-Signale (chip selects)

Aus Abb. 1.5 können wir sehen, daß ROMA angesteuert wird (also seine Daten auf den Datenbus legt), wenn der Z80 eine Adresse im Bereich 0000 bis 07FF (inklusiv) ausgibt. Wir benötigen jetzt ein elektrisches Signal, das nur innerhalb dieser Grenzen aktiv ist. Abb. 1.6 zeigt eine Schaltung, die dieses Signal erzeugt.

In der Schaltung in Abb. 1.6 wird der Binärdecoder 74LS138 verwendet. Von seinen acht Ausgängen ist, abhängig von den Eingangszuständen, max. einer logisch 0. Sind z.B. A11, A12, A13, A14 und A15 logisch 0, so wird der Ausgang Pin 15 logisch 0. Die Tabelle in Abb. 1.7 zeigt die Abhängigkeit der Ausgänge des 74LS138 von den Adressen auf dem Adreßbus.

Die Schaltung in Abb. 1.6 ist natürlich nur eine von vielen Möglichkeiten, einzelne Adreß-Blöcke aus dem gesamten Adreßbereich auszuwählen.

A15	A14	A13	A12	A11	A10 ---------- A0	HEX	PIN # = 0
0	0	0	0	0	0 ---------- 0	0000	15
0	0	0	0	0	1 ---------- 1	07FF	15
0	0	0	0	1	0 ---------- 0	0800	14
0	0	0	0	1	1 ---------- 1	0FFF	14
0	0	0	1	0	0 ---------- 0	1000	13
0	0	0	1	0	1 ---------- 1	17FF	13
0	0	0	1	1	0 ---------- 0	1800	12
0	0	0	1	1	1 ---------- 1	1FFF	12
0	0	1	0	0	0 ---------- 0	2000	11
0	0	1	0	0	1 ---------- 1	27FF	11
0	0	1	0	1	0 ---------- 0	2800	10
0	0	1	0	1	1 ---------- 1	2FFF	10

Abb. 1.7: *Wirkung der Schaltung aus Abb. 1.6.*

Erzeugung des Speicher-Lese-Signals (memory read)

Zur Beschaltung des Chip-Select-Eingangs des ROM fehlt uns jetzt noch ein Steuersignal, das angibt, wann der Z80 den Datentransfer beginnt und beendet. Dieses sog. Memory-Read-Signal ($\overline{\text{MEMR}}$) ermöglicht zusammen mit dem Select-Signal (siehe Abb. 1.6) die Ausgabe der Daten eines bestimmten ROM auf den Datenbus. Abb. 1.8 zeigt eine Möglichkeit zur Erzeugung dieses Signals.

Die $\overline{\text{MEMR}}$-Leitung in Abb. 1.8 wird immer dann logisch 0, wenn der Z80 Daten aus dem Speicher lesen will. Dieser Vorgang wird durch einen Software-Befehl ausgelöst. Dabei wird die $\overline{\text{RD}}$-Leitung immer dann 0, wenn der Prozessor Daten lesen will, und die $\overline{\text{MREQ}}$-Leitung (memory request = Speicheranfrage) dann, wenn auf einen Speicher zugegriffen werden soll.

Verbinden der Chip-Select-Leitungen

Abb. 1.9 zeigt die komplette Schaltung für die Verbindung des Z80 mit einem ROM. Wenn sowohl das Auswahlsignal ($\overline{\text{ROMA}}$) als auch das Speicher-Lese-Signal ($\overline{\text{MEMR}}$) logisch 0 ist, wird der Chip-Select-Eingang ($\overline{\text{CE}}$) des 2716-EPROM logisch 0 und somit aktiv. Abb. 1.10 zeigt das Timing während einer ROM-Lese-Operation.

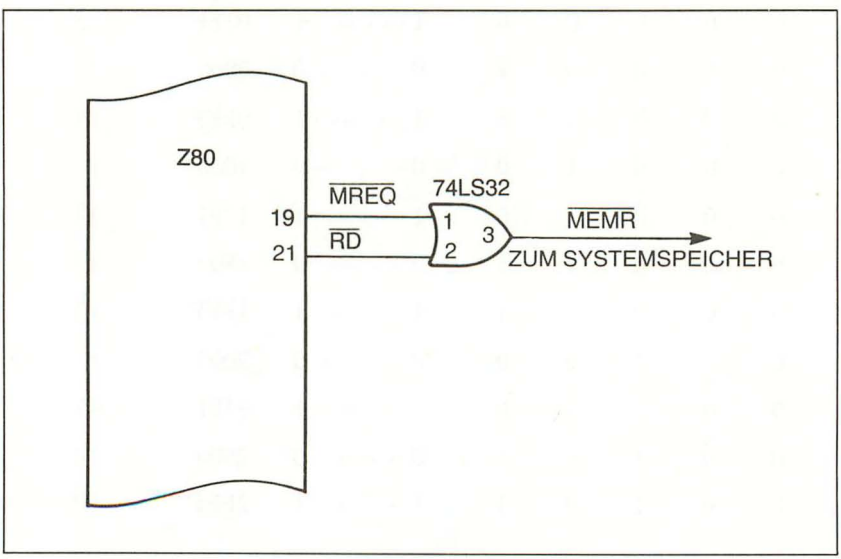


Abb. 1.8: *Schaltung zur Erzeugung des Memory-Read-Signals durch ODERung der Z80 Steuersignale.*

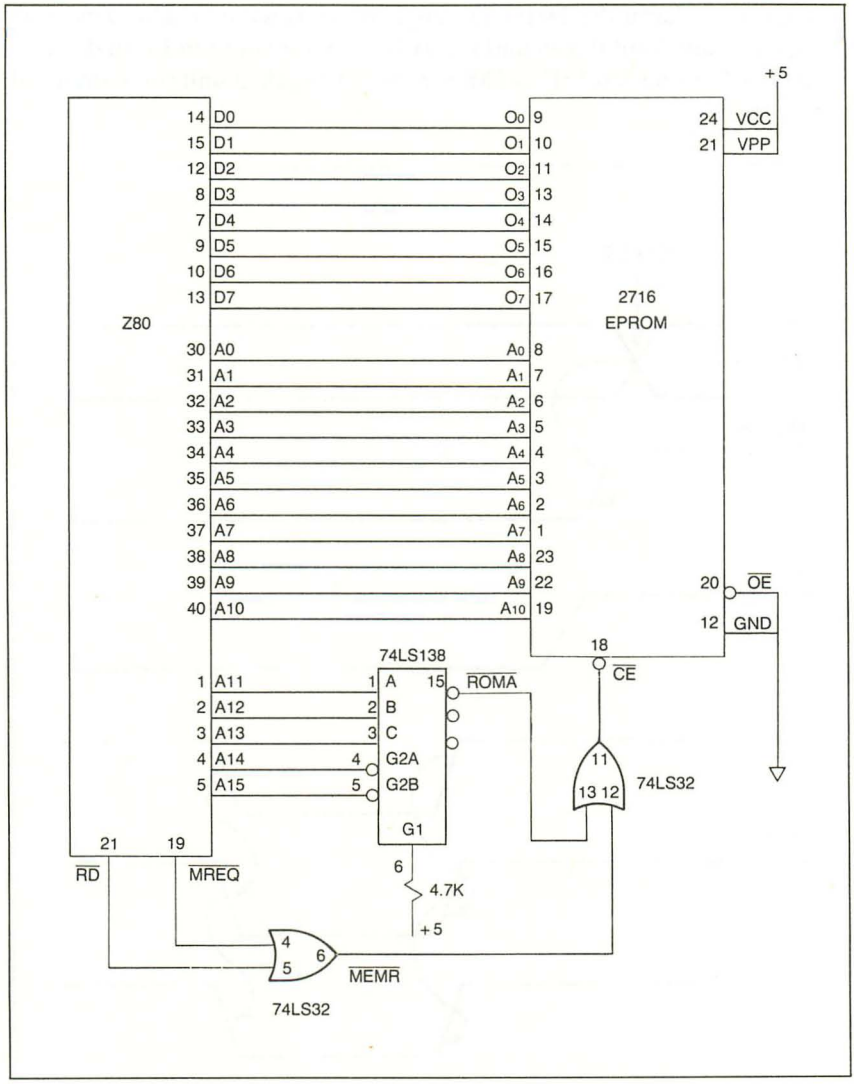


Abb. 1.9: *Die komplette Schaltung für den Anschluß eines 2716-EPROM an den Z80-Mikroprozessor gemäß unseres Speicherbelegungsplans.*

Eine andere Möglichkeit zur Speicheranwahl

Abb. 1.11 zeigt noch eine andere Möglichkeit, die Datenausgabe des ROM zum richtigen Zeitpunkt zu steuern. Dabei wird von der Tatsache Gebrauch gemacht, daß der 2716 zwei Eingänge hat, die beide logisch 0

sein müssen, damit die Datenausgänge aktiviert werden. Dies sind $\overline{OE}$ (Output Enable) auf Pin 20 und $\overline{CE}$ (Chip Enable) auf Pin 18. In der vorherigen Schaltung war $\overline{OE}$ auf Masse-Potential gelegt und somit dauernd aktiviert.

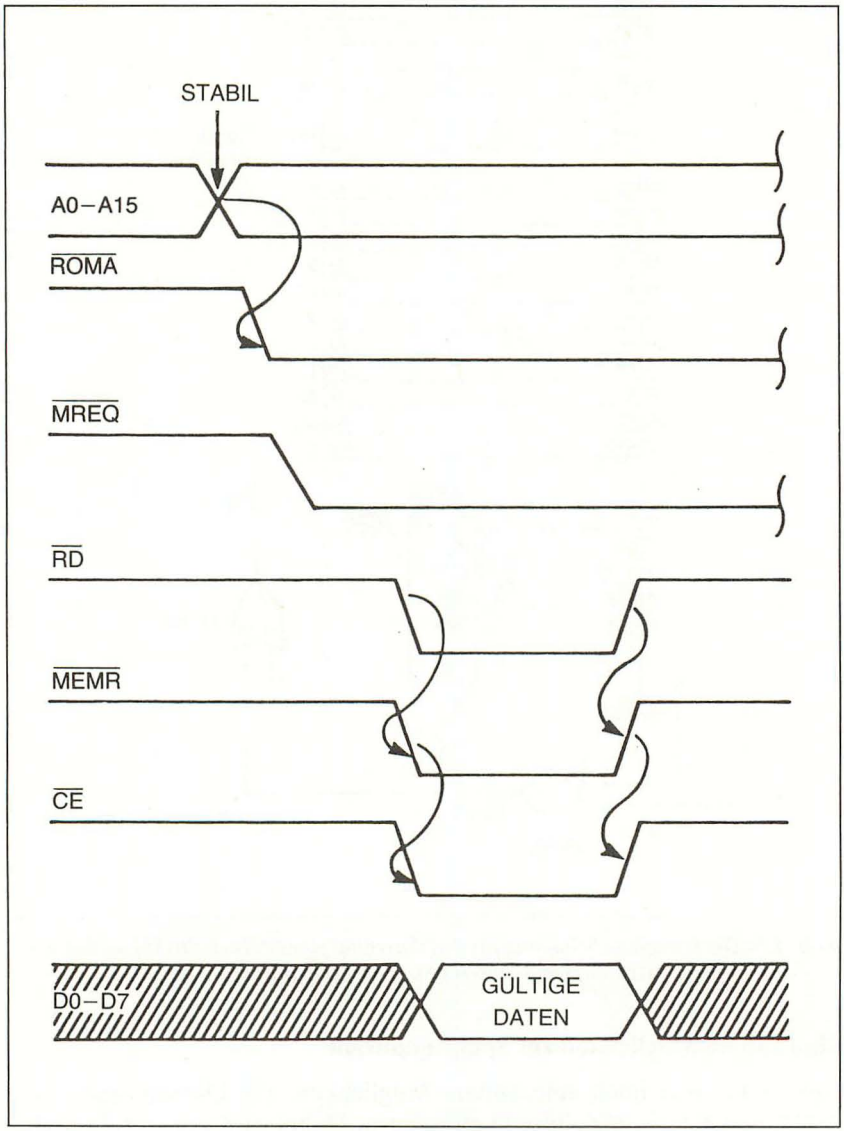


Abb. 1.10: *Timing für die wichtigsten Signale aus Abb. 1.9.*

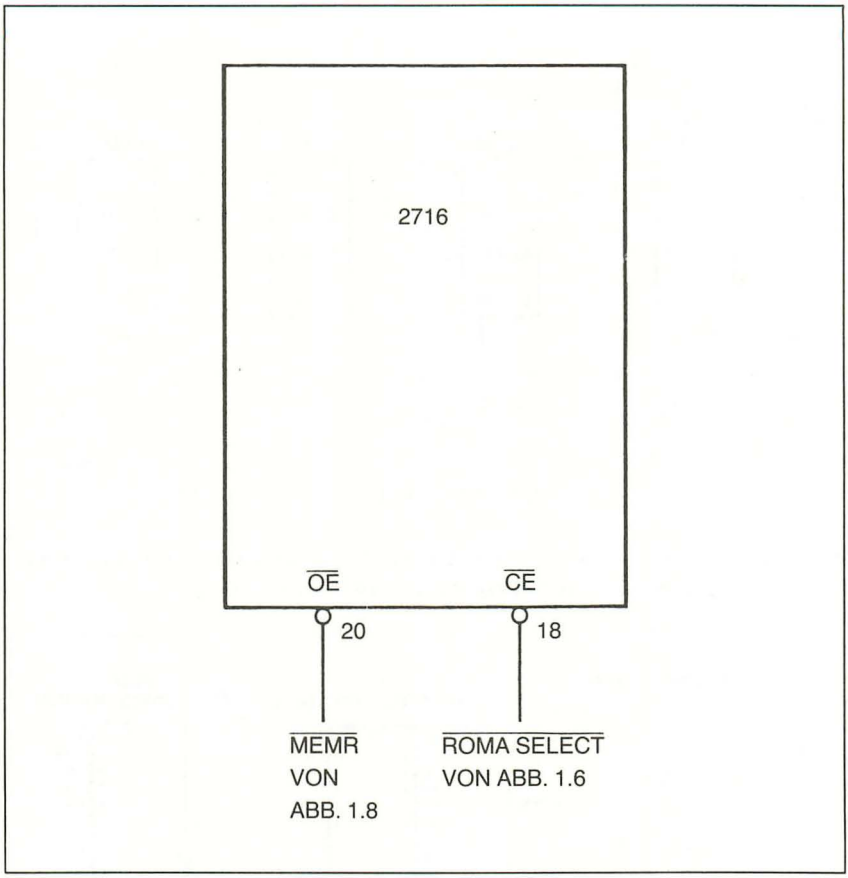


Abb. 1.11: *Schaltung, bei der sowohl Pin 20 als auch Pin 18 zur Speicheranwahl benutzt wird.*

Systeme mit mehreren ROMs

Abb. 1.12 zeigt, daß es möglich ist, mit der in Abb. 1.11 gezeigten Technik mehrere ROMs ohne zusätzliche Gatter-Bausteine anzusteuern. (Die in Abb. 1.10 verwendete Technik würde eine Erweiterung nur mit zusätzlichen Gattern zulassen).

Erinnern wir uns, daß für jedes ROM ein bestimmter Adreßblock reserviert ist. Natürlich macht die Z80-CPU diese Einteilung bei der Abarbeitung der Software nicht. Es ist durchaus möglich, daß ein Mehr-Byte-Befehl zum Teil in einem ROM steht, zum Teil in einem anderen. Es dürfen also keine Lücken im Adreßbereich auftreten. (Es gibt Ausnahmen.)

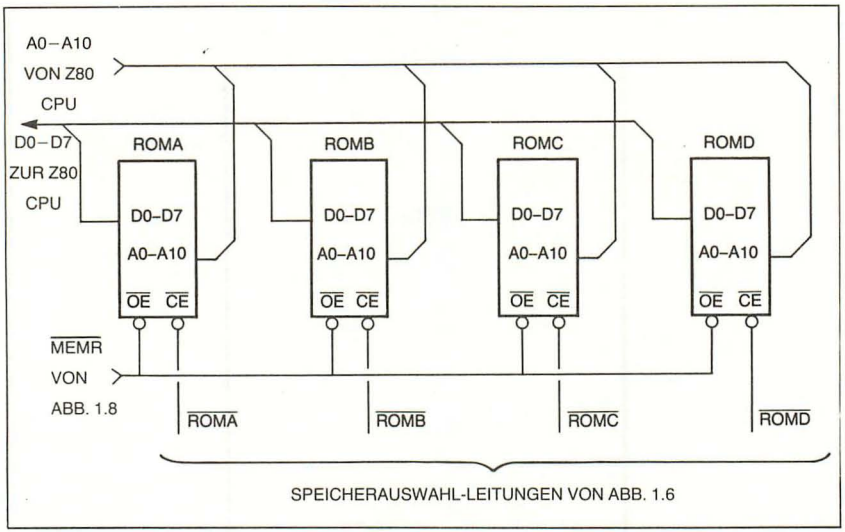


Abb. 1.12: *Schaltung zur Ansteuerung mehrerer ROMs ohne weiteren Hardware-Aufwand. Hierbei wird die $\overline{OE}$-Leitung des 2716 ausgenutzt.*

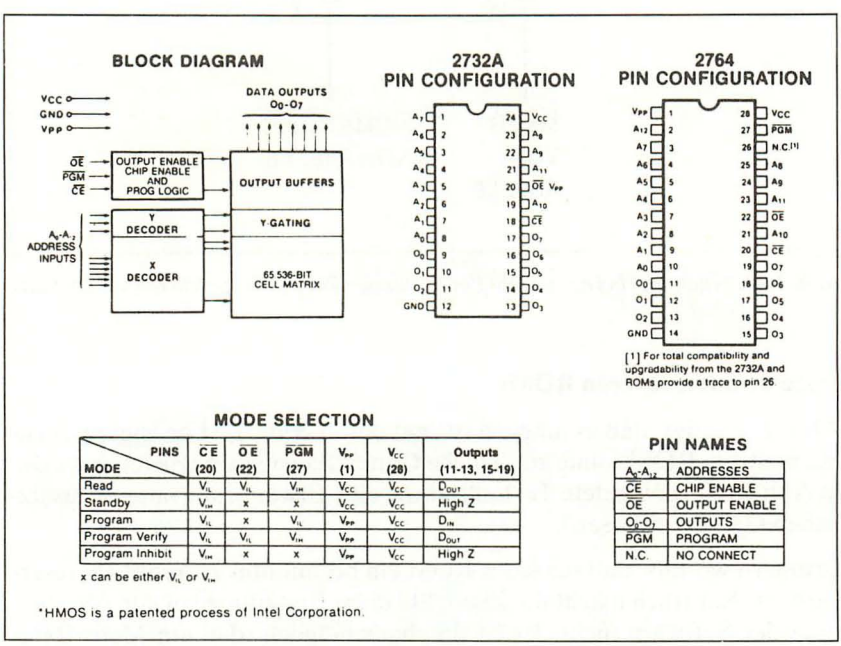


Abb. 1.13: *Pinbelegung und Blockschaltbild der in Mikroprozessor-Systemen häufig verwendeten 2732- und 2764-EPROMs.*

Adressierung größerer ROMs

Zwei Typen von größeren EPROMs werden häufig eingesetzt: Das 2732 mit 4096 * 8 Bit- und das 2764 mit 8192 * 8 Bit-Organisation. Abb. 1.13 zeigt die Pinbelegung dieser beiden Bauteile.

Interessant hierbei ist, daß trotz der verschiedenen Pinbelegungen beide Bauteile wahlweise in einen Sockel eingesetzt werden können. Man benutzt hierzu 28-Pin-Sockel und legt Vcc an Pin 28 und an Pin 26 (siehe Abb. 1.14). Es besteht somit die Möglichkeit, anfangs einen 2732 einzusetzen und später mit einem 2764 den Speicherplatz zu verdoppeln, ohne mehr Platz auf der Platine zu belegen.

Mit der Verwendung größerer Speicher ändert sich auch der Speicherbelegungsplan von Abb. 1.5. Abb. 1.15 zeigt eine Schaltung, die den Speicherbereich in Blöcke von 4K aufteilt. Der Bereich für ROMA geht jetzt von 0000 bis 0FFF.

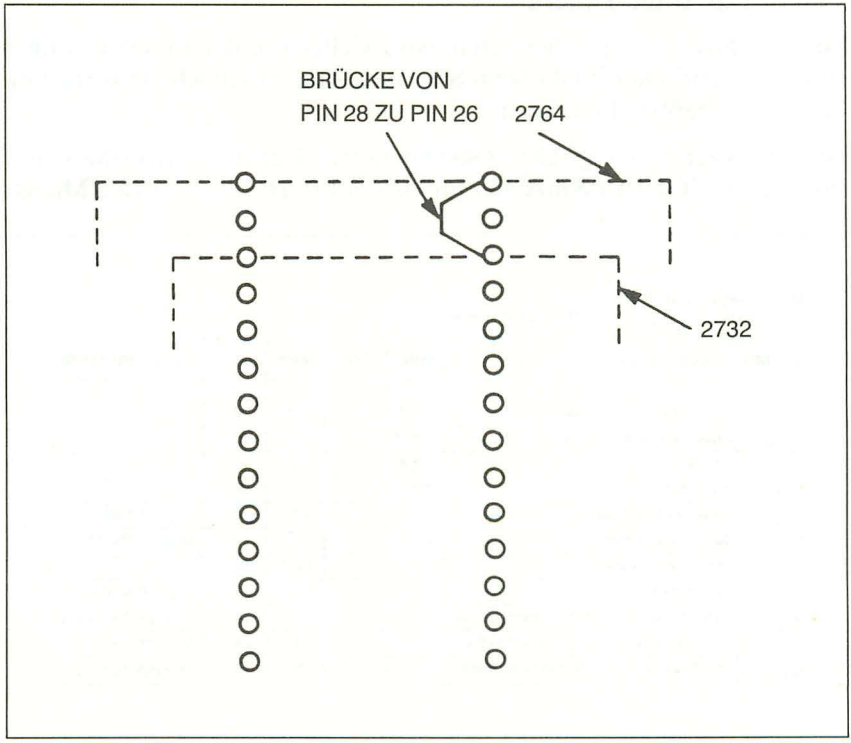


Abb. 1.14: *Auf diese Weise läßt sich ein 28-Pin-Sockel wahlweise für 2732- und 2764-EPROMs benutzen.*

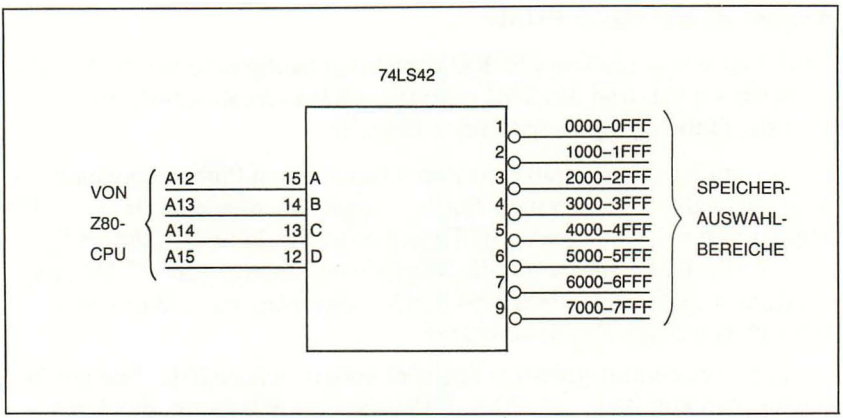


Abb. 1.15: *Memory-Select-Decodierung mit einem 74LS42 BCD-zu-Dezimal-Decoder.*

Einsatz von Adreß-Puffern

Das beschriebene Speichersystem wird vielfach in der Industrie eingesetzt. Ein kritischer Punkt beim Systementwurf ist jedoch die Belastung der Z80-Adreßbus-Leitungen.

Abb. 1.16 zeigt ein typisches Z80-Datenblatt. Der max. Ausgangsstrom für logisch 0 IOL ist 1,8 mA, der für logisch 1 IOH – 250 μA (das Minus-

D.C. CHARACTERISTICS
T_A = 0°C to 70°C, V_{CC} = 5V ± 5% unless otherwise specified

SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNIT	TEST CONDITION
V_{ILC}	Clock Input Low Voltage	−0.3		0.8	V	
V_{IHC}	Clock Input High Voltage	Vcc−.6		Vcc+.3	V	
V_{IL}	Input Low Voltage	−0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 1.8mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = −250 μA
I_{CC}	Power Supply Current			150*	mA	
I_{LI}	Input Leakage Current			±10	μA	V_{IN} = 0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	V_{OUT} = 2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float			−10	μA	V_{OUT} = 0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	$0 \leq V_{IN} \leq V_{CC}$

Abb. 1.16: *Ein Teil-Datenblatt für den Z80-Mikroprozessor.*

zeichen gibt an, daß der Strom vom Z80 abgegeben wird). Das bedeutet, daß eine Standard-TTL-Last (Transistor-Transistor-Logik) von einem Ausgang angesteuert werden kann. Eine TTL-Last entspricht 1,6 mA bei logisch 0 und 40 μA bei logisch 1.

In den bisher beschriebenen Schaltungen wurde von den oberen Adreßleitungen nur ein TTL-Bauteil (z.B. 74LS138) angesteuert. LS-Bausteine (Low-Power-Schottky) haben nur etwa 1/3 der Last von Standard-TTL-Bausteinen. Die unteren Adreßleitungen wurden direkt mit den Adreßeingängen der Speicher verbunden. MOS-Speicher haben eine Eingangslast von ca. 10 μA (logisch 0 und 1). Möglicherweise steuern die Adreßleitungen noch andere Bauteile an. Kabel und Steckverbinder führen zudem zu einer Erhöhung der kapazitiven Belastung.

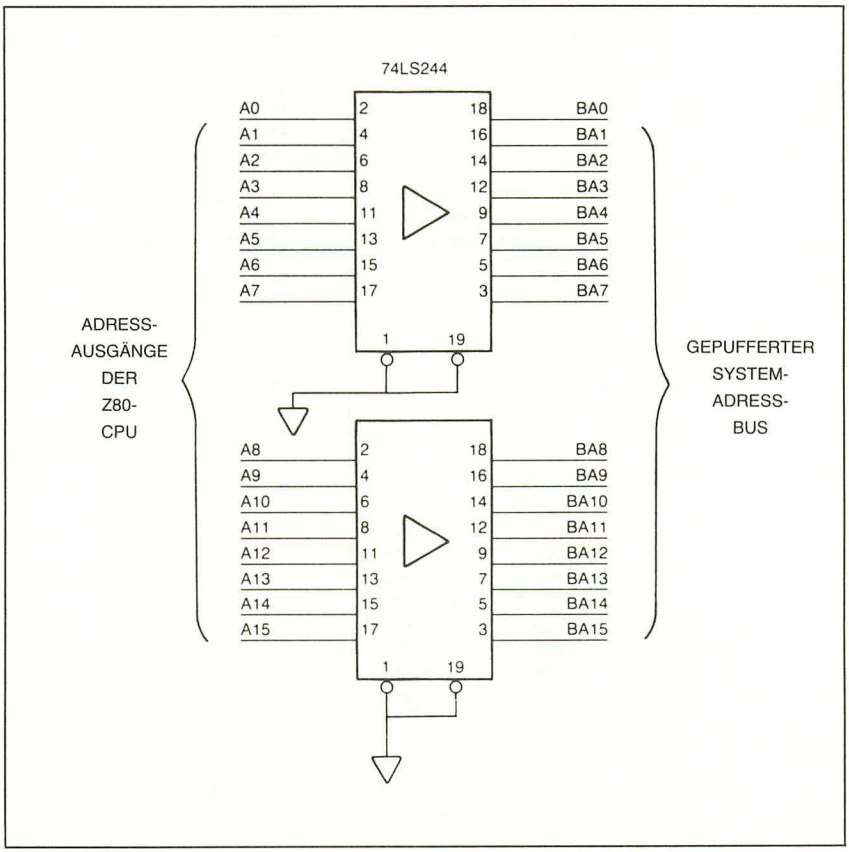


Abb. 1.17: *Pufferung eines Z80-Adreßbusses.*

In Fällen, in denen die max. Ausgangslast der Adreßleitungen überschritten wird, müssen Adreß-Puffer (Buffer) eingesetzt werden. Adreß-Puffer erhöhen den max. Treiberstrom des Z80-Adreßbusses und sollten auf jeden Fall eingesetzt werden, wenn ein Kabel angesteuert wird. Abb. 1.17 zeigt den Einsatz von Adreß-Puffern.

Speicher-Datenpuffer

In einigen Anwendungsfällen reicht der max. Ausgangsstrom der Datenleitungen des ROM nicht aus, den Datenbus direkt zu treiben. In diesem Fall müssen Datenbus-Puffer eingesetzt werden.

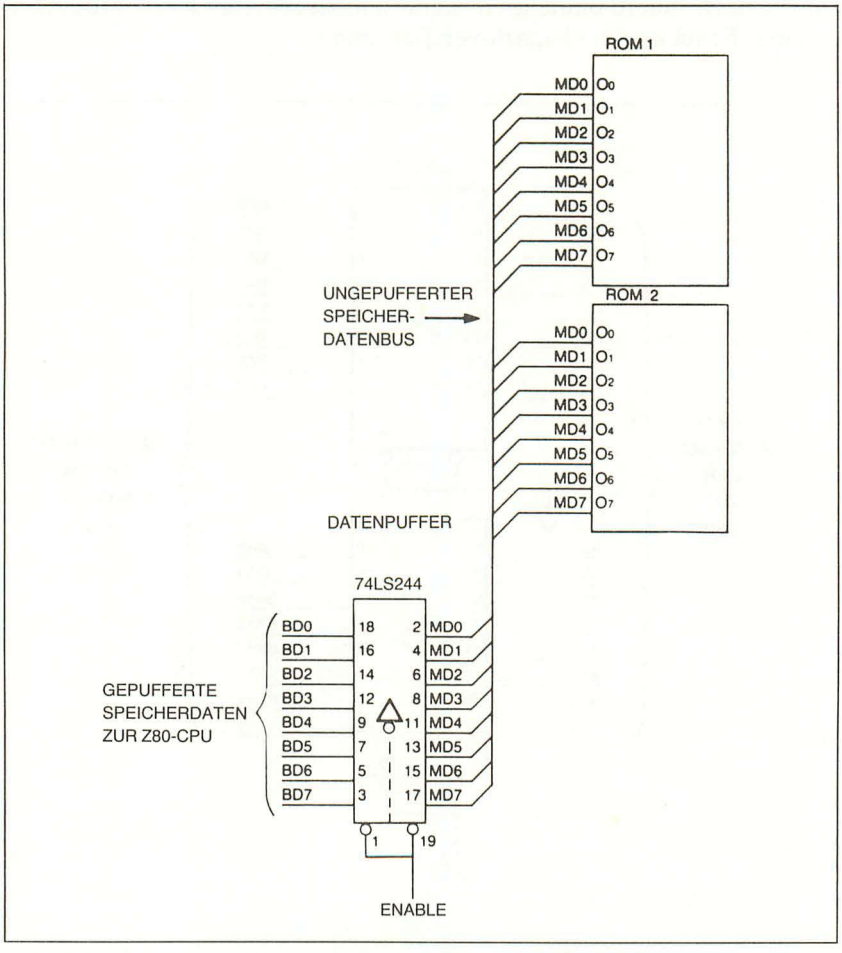


Abb. 1.18: *Einsatz eines Datenbus-Puffers.*

Die Speicher-Datenbus-Puffer müssen die Möglichkeit zur Tri-State-Betriebsart haben. Damit kann die Datenverbindung vom ROM zum Bus elektrisch unterbrochen werden, wenn der Mikroprozessor sie nicht braucht. Die Schaltung in Abb. 1.18 zeigt, wie Datenpuffer in einem typischen ROM-System eingesetzt werden. In dieser Schaltung treiben die ROM-Datenausgänge nur die Datenpuffer-Eingänge. Die Pufferausgänge treiben den Systemdatenbus.

Drei Beispiele für vollständige ROM-Systeme

Die Schaltungen in Abb. 1.19, 1.20 und 1.21 sind vollständige ROM-Systeme, die die in diesem Kapitel beschriebenen Schaltungskonzepte

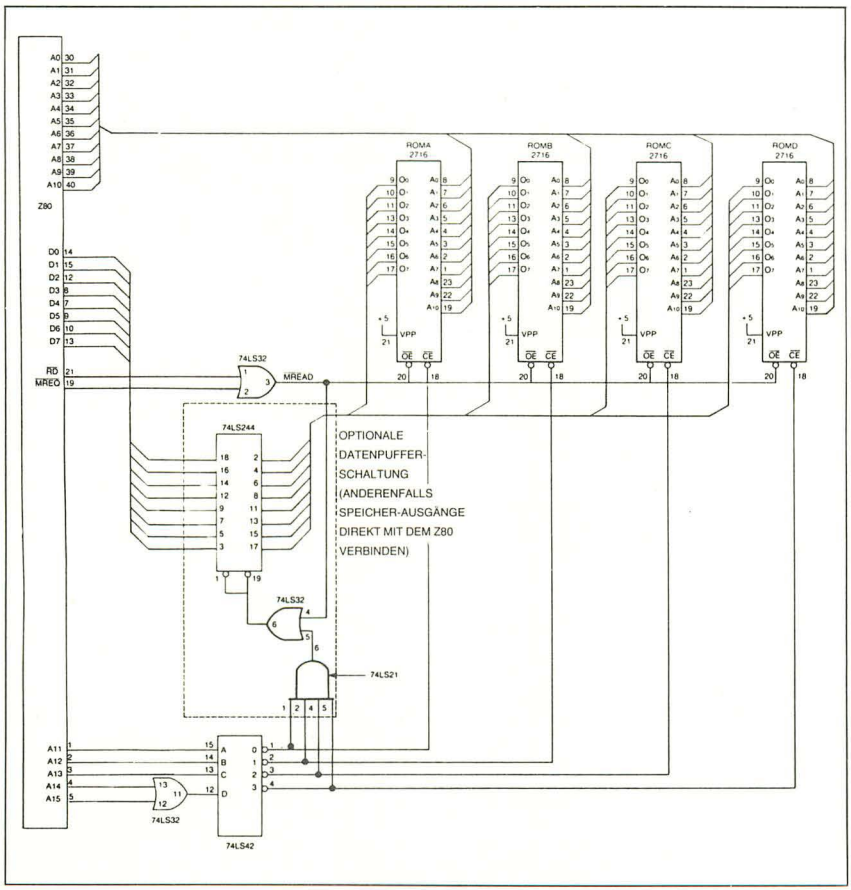


Abb. 1.19: *Ein vollständiges Speichersystem unter Verwendung der Memory-Map aus Abb. 1.22.*

beherzigen. Die Memory-Map hierfür zeigt Abb. 1.22. Speichersysteme solcher Art sind heute im allgemeinen Gebrauch. Die EPROMs sind aus vielen Quellen verfügbar und leicht zu programmieren und einzusetzen.

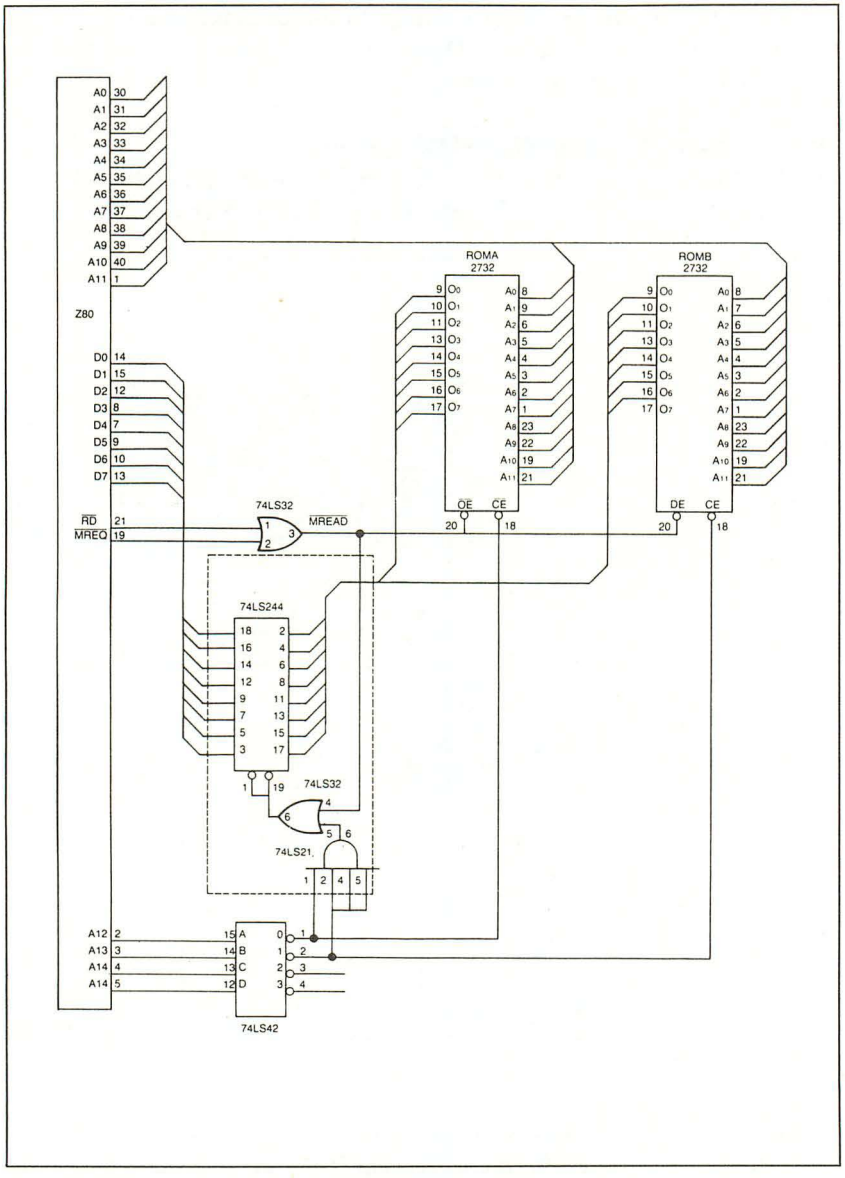


Abb. 1.20: *Ein weiteres Speichersystem mit 2732-Bausteinen.*

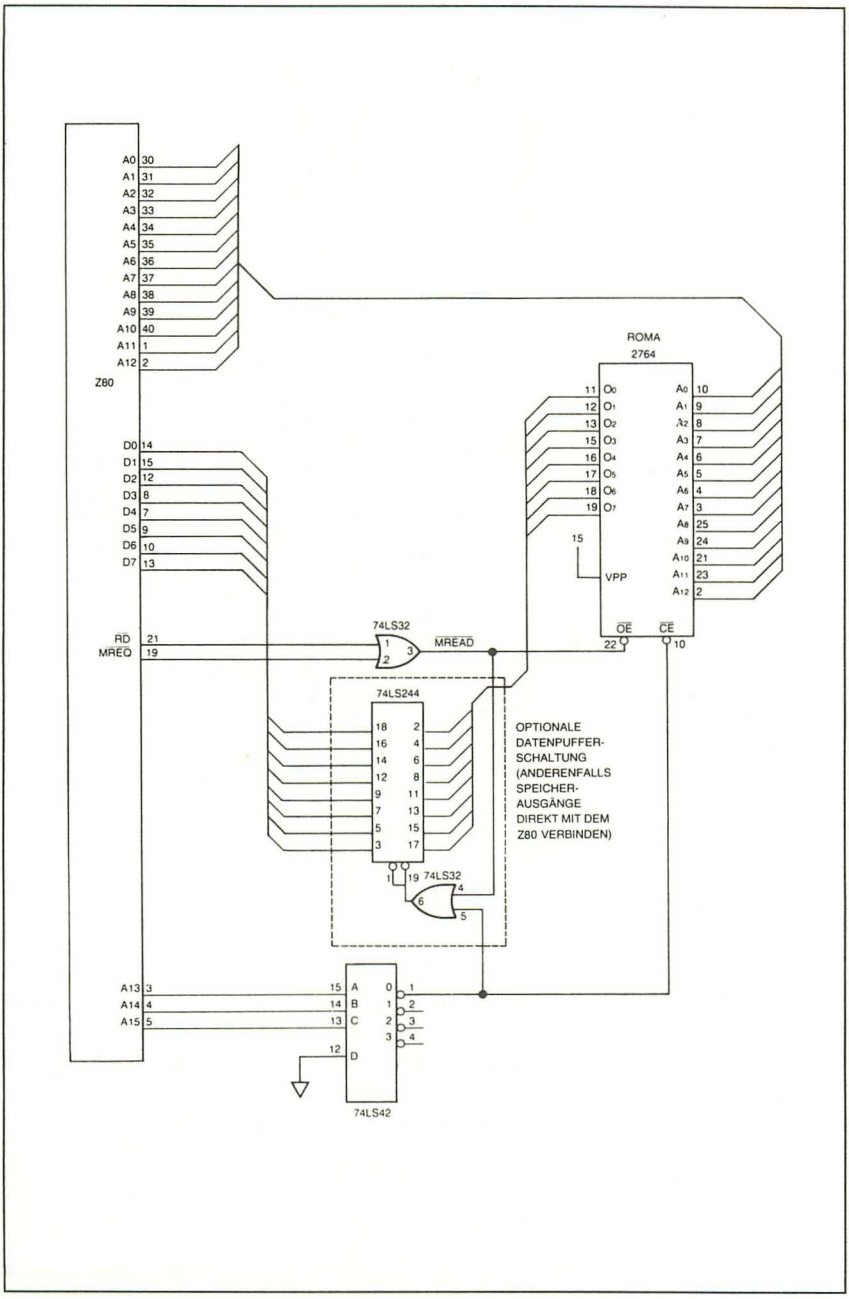


Abb. 1.21: *Ein drittes Beispiel mit 2764-Bausteinen.*

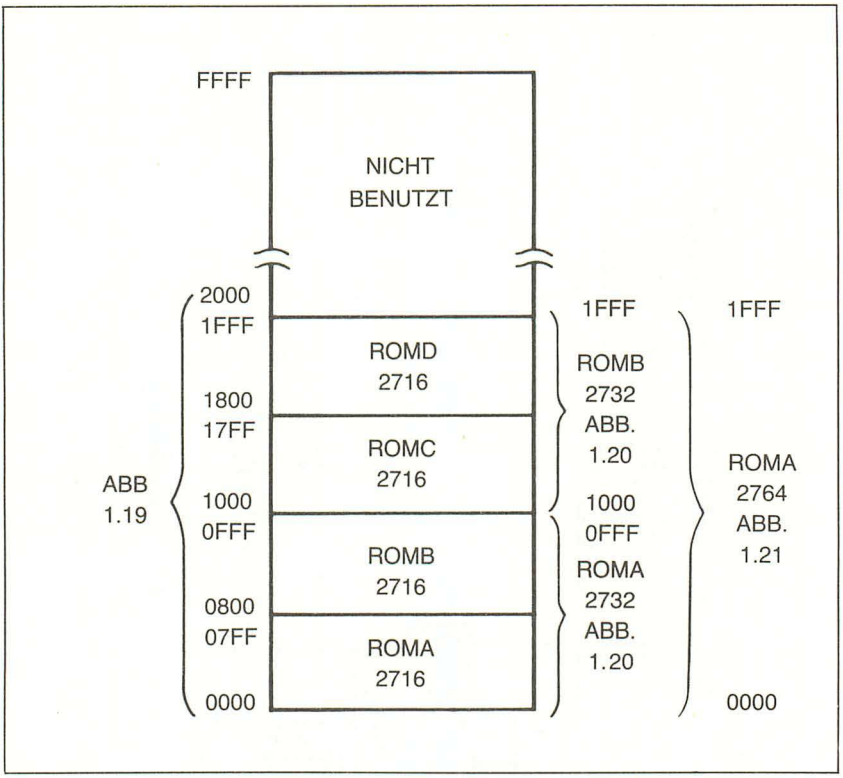


Abb. 1.22: *Die Memory-Map zeigt die Beziehungen der Adressen zu den unterschiedlichen physikalischen Speichereinheiten.*

Zusammenfassung

Wir haben in diesem Kapitel nützliche Informationen erhalten in Hinblick auf die Benutzung von ROMs in Z80-Systemen. Wir haben einige Grundlagen zur Arbeitsweise von ROMs kennengelernt sowie die wichtigsten Elemente zur Verbindung mit einem Z80. Die Beispiele in diesem Kapitel zeigten, daß die Verwendung von ROMs in Z80-Systemen eine relativ einfache Sache ist. Es gibt natürlich einige Regeln, die befolgt werden müssen.

Kapitel 2

Statische RAMs in Z80-Systemen

Einführung

Wir wollen in unserer Studie des Z80-Mikroprozessors damit fortfahren zu lernen, wie er mit Schreib-Lese-Speichern, den RAMs (Random Access Memory), verbunden wird. RAMs werden in Mikroprozessor-Systemen zur vorübergehenden Speicherung von Programmen, Daten und Variablen eingesetzt. Im Gegensatz zu ROMs (Read-Only Memory, beschrieben in Kapitel 1) sind RAMs flüchtige Speicher, das heißt, sie verlieren ihre Information beim Ausschalten der Stromversorgung.

Die elektrische Verbindung zwischen dem Z80 und statischen RAMs ist nicht schwer herzustellen. Einige Richtlinien sind dabei natürlich zu beachten. Diese Richtlinien und alle zur Verwendung von statischen RAMs wichtigen Punkte sind Thema dieses Kapitels. Wir werden außerdem zwei in der Industrie häufig benutzte Systeme vorstellen und diskutieren. Das eine verwendet den 2114, einen 1K * 4 Bit Speicher, das andere den 6116 mit 1K * 8 Bit.

Allgemeines über statische RAMs

Als erstes wollen wir erläutern, wie die Kommunikation mit einem Halbleiter-RAM im allgemeinen funktioniert. Dies wird Ihnen bei dem Verständnis der Kommunikation des Z80 zum RAM bedeutend weiterhelfen. Sind Sie hiermit bereits vertraut, so können Sie diesen Teil des Kapitels überspringen.

RAMs sind fähig, sowohl Daten zur CPU zu senden als auch Daten von ihr zu empfangen. Zur Steuerung dieser beiden Kommunikationsarten dienen spezielle elektrische Signale. Abb. 2.1 zeigt ein Blockschaltbild eines typischen RAM-Bauteils mit seinen Anschlüssen.

Als erstes betrachten wir den Anschluß der Stomversorgung. Üblich hierbei ist eine Spannung von +5 V zwischen Vcc und Masse, wovon wir im weiteren auch ausgehen. (Genaue Angaben stehen im Datenblatt des Herstellers.)

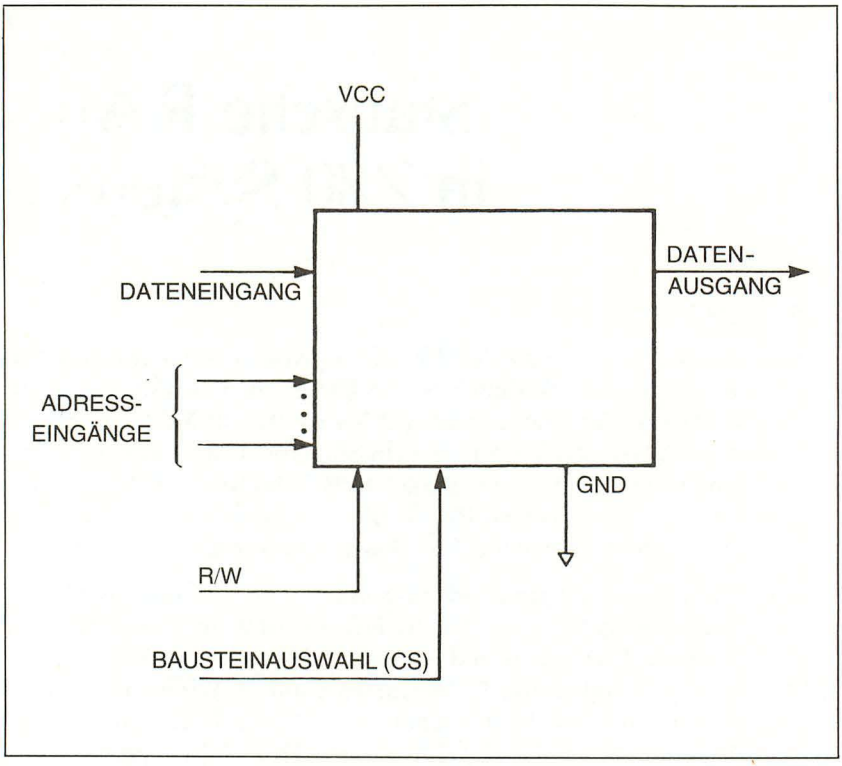


Abb. 2.1: *Blockschaltbild eines typischen RAM.*

Die Leitungen „Dateneingang" in Abb. 2.1 sind physikalische Anschlüsse, die es erlauben, elektrische Informationen in das RAM zu schreiben. Über die Leitungen „Datenausgang" können die gespeicherten Daten wieder elektrisch ausgelesen werden. Das RAM hat mindestens eine Eingangs- und Ausgangsleitung; die Anzahl ist abhängig von der Organisation des Bauteils.

In Kapitel 1 haben wir die interne Organisation eines ROM beschrieben. Das meiste davon läßt sich auf das RAM übertragen. Wie beim ROM gibt die Bezeichnung des RAM schon einige Informationen. Ein RAM kann z.B. als $256 * 1$, $256 * 4$ oder $1024 * 8$ bezeichnet werden, wobei die erste Nummer die Anzahl der Adressen und die zweite die Anzahl der Dateneingänge und Ausgänge beschreibt. Die zweite Nummer gibt also an, wieviel Bits bei einer Lese- oder Schreiboperation parallel aus dem RAM gelesen oder in das RAM geschrieben werden.

Sowohl zum Lesen als auch zum Schreiben werden die gleichen Adreßleitungen benutzt. Die Relation zwichen der Anzahl der Adreßleitungen und der Anzahl der verschiedenen Speicheradressen ist dieselbe wie beim ROM. Ein statisches 1024 * 4 bzw. 1K * 4 RAM hat z.B. 10 Adreßleitungen.

Als letztes nun die R/W-Leitung aus Abb. 2.1. Sie dient zur elektrischen Umschaltung auf Lese- oder Schreibbetrieb.

Wir wollen jetzt unsere Aufmerksamkeit auf die Ereignisse beim Lesen und Beschreiben eines statischen Halbleiter-RAM richten, wenn es mit dem Mikroprozessor kommuniziert. Als erstes betrachten wir die Ereignissse während einer Leseoperation.

Ereignisablauf bei einer RAM-Leseoperation

1. Als erstes wird die Adresse eingelesen. Der Speicherplatz, von dem die Daten gelesen werden sollen, wird decodiert.
2. Die R/W-Steuerleitung wird auf den für die Leseoperation richtigen Pegel gebracht. Bei einigen Speichern ist dies logisch 1, bei anderen logisch 0. (Genauere Angaben stehen im Datenblatt des Herstellers.)
3. Das System muß jetzt eine bestimmte Zeit warten, damit die interne Logik den Speicherplatz decodieren kann. Diese Zeit wird als Lese-Zugriffszeit (read access time) bezeichnet.
4. Nach dieser Wartezeit sind die Daten an den Ausgangsleitungen verfügbar und können vom Mikroprozessor gelesen werden. Liest der Prozessor die Daten zu früh, also vor Ablauf der Zugriffszeit, so können sie fehlerhaft sein.

Abb. 2.2 zeigt das Timing für diesen Ablauf. Wir werden später darauf zurückkommen, wenn wir uns mit der Kommunikation des Z80 mit Halbleiter-RAMs beschäftigen. Der Ablauf beim Lesen von Daten aus dem RAM ist exakt der gleiche wie beim ROM. Tatsächlich unterscheidet der Mikroprozessor nicht zwischen einer RAM- und einer ROM-Leseoperation.

Lassen sie uns jetzt den Ereignisablauf beim Schreiben von Daten zum RAM betrachten.

Ereignisablauf bei einer RAM-Schreiboperation

1. Als erstes wird die Adresse eingelesen, um die Speicherstelle, zu der geschrieben werden soll, anzuwählen.

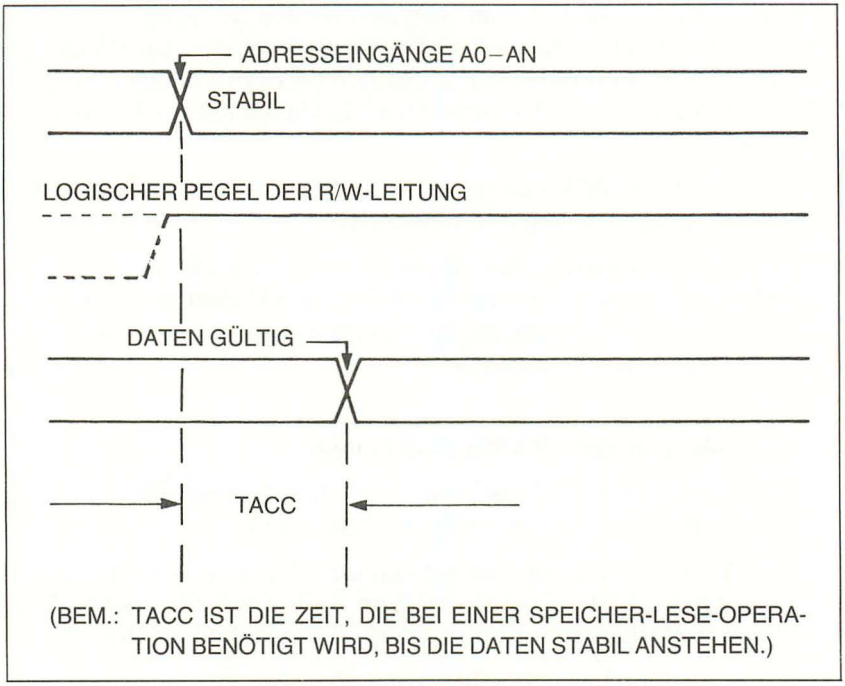


Abb. 2.2: *Allgemeines Timing beim Lesen von Daten aus dem Systemspeicher.*

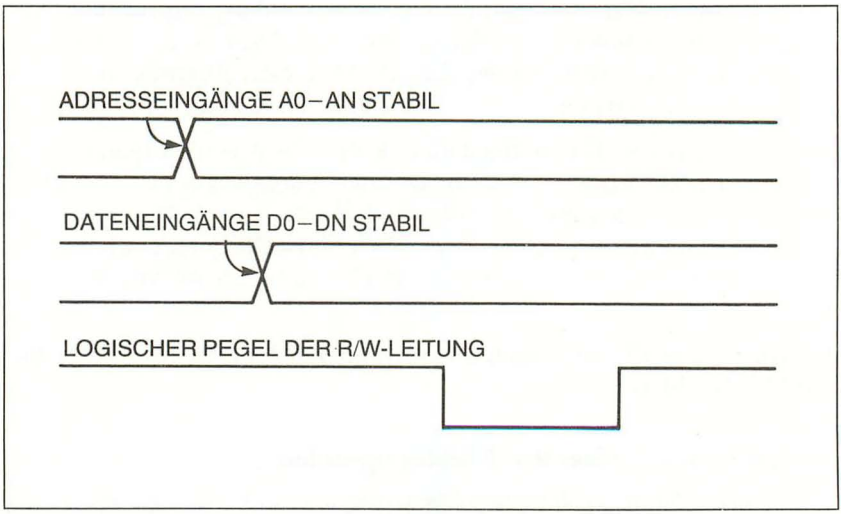


Abb. 2.3: *Diagramm zum zeitlichen Ablauf bei einer RAM-Schreiboperation.*

2. Als nächstes werden die Daten, die in das RAM geschrieben werden sollen, an die Eingangsleitungen angelegt.
3. Das System muß jetzt für eine bestimmte Zeit warten, damit sich die interne Decodierlogik stabilisieren kann. Diese Zeit wird als Schreib-Zugriffszeit (write access time) bezeichnet und liegt i.a. bei bis zu einigen hundert Nanosekunden.
4. Nach dieser Wartezeit wird die R/W-Steuerleitung auf den entsprechenden Pegel gesetzt bzw. gepulst und damit die Daten in das RAM geschrieben.

Das Timing für diesen Ablauf zeigt Abb. 2.3. Die zur Einhaltung dieses Ablaufs nötige Hardware ist abhängig vom eingesetzten Prozessortyp.

Ein reales Speicherbauteil

Lassen Sie uns jetzt ein typisches und in der Industrie weitverbreitetes RAM untersuchen. Wir wollen zuerst die wichtigsten Betriebsdaten dieses Bauteils betrachten und dann sehen, wie die Daten gelesen bzw. geschrieben werden. Das Verständnis der elektrischen Arbeitsweise des RAM ist grundlegend wichtig. Wenn Sie die Arbeitsweise eines RAM detailliert kennen, wird es für Sie auch leicht sein, sie auf beliebige RAMs in anderen Mikroprozessor-Anwendungen zu übertragen.

Das RAM, das wir uns für unsere Betrachtung ausgesucht haben, ist das 2114, ein 1024 * 4 statisches RAM mit gemeinsamen Ein- und Ausgängen (common I/O). Common I/O bezieht sich auf die elektrische Anordnung der Eingangs- und Ausgangsleitungen. Ein RAM kann entweder getrennte oder gemeinsame I/O-Leitungen haben. RAMs mit getrennten

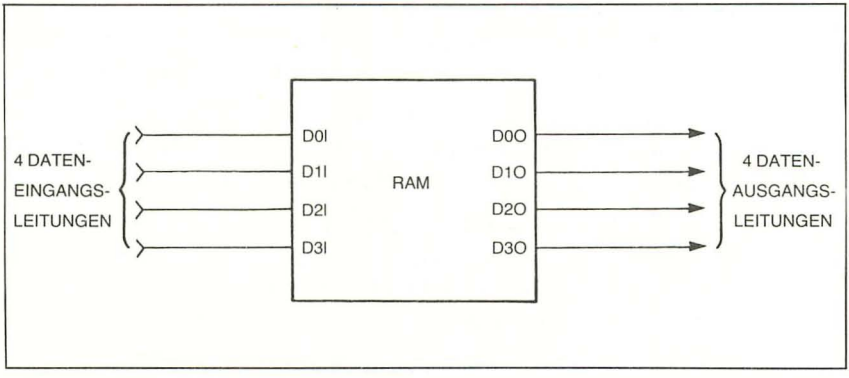


Abb. 2.4: *Blockschaltbild eines RAM mit getrennten Ein- und Ausgängen.*

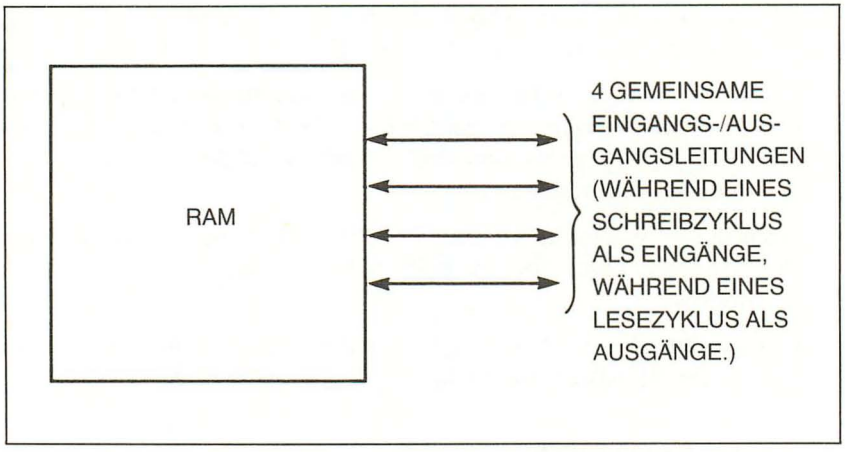


Abb. 2.5: *Blockschaltbild eines RAM mit gemeinsamen Ein- und Ausgängen.*

I/O-Leitungen (separate I/O) benutzen für Dateneingänge bzw. Ausgänge verschiedene Pins (siehe Abb. 2.4); bei RAMs mit gemeinsamem I/O liegen die Ein- und Ausgänge auf denselben Pins (siehe Abb. 2.5).

Daten können sowohl in das RAM geschrieben als auch aus dem RAM gelesen werden. Diese beiden Funktionen schließen sich aber gegenseitig aus: Man kann niemals beides zur gleichen Zeit tun. Das bedeutet, daß niemals die Eingangs- und die Ausgangsleitungen zur gleichen Zeit benutzt werden können. Das erlaubt eine Einsparung von Pins zur Lese-/Schreibumschaltung. Das gilt auch für das oben beschriebene gemeinsame I/O. Die Eingangs- und Ausgangsleitungen werden dabei im Zeit-Multiplex betrieben: Während der Lese-Operation dienen die I/O-Leitungen als Ausgänge und während der Schreiboperation als Eingänge. Der logische Pegel der R/W-Steuerleitung bestimmt die Datenrichtung. Abb. 2.6 zeigt die Pinbelegung und das Blockschaltbild für einen 2114.

Lassen Sie uns nun die wichtigsten Parameter dieses Speichers betrachten. (Anmerkung: Diese Informationen gelten für die meisten Halbleiter-Speicher, auch wenn sie kein gemeinsames I/O haben.)

Bei diesem Beispiel wollen wir das RAM nur von der Benutzerebene her betrachten. Außerdem gehen wir davon aus, daß die Speicherzugriffszeit der Systemgeschwindigkeit angepaßt ist und keine zusätzlichen Wartezyklen generiert werden müssen.

Abb. 2.7 zeigt einen Datenblattauszug und allgemeine Timing-Diagramme für die elektrische Kommunikation mit dem 2114. Als erstes

2114A 1024 X 4 BIT STATIC RAM

	2114AL-1	2114AL-2	2114AL-3	2114AL-4	2114A-4	2114A-5
Max. Access Time (ns)	100	120	150	200	200	250
Max. Current (mA)	40	40	40	40	70	70

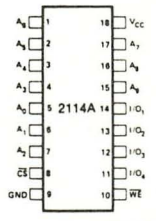
- **HMOS Technology**
- **Low Power, High Speed**
- **Identical Cycle and Access Times**
- **Single +5V Supply ±10%**
- **High Density 18 Pin Package**
- **Completely Static Memory - No Clock or Timing Strobe Required**
- **Directly TTL Compatible: All Inputs and Outputs**
- **Common Data Input and Output Using Three-State Outputs**
- **2114 Upgrade**

The Intel® 2114A is a 4096-bit static Random Access Memory organized as 1024 words by 4-bits using HMOS, a high performance MOS technology. It uses fully DC stable (static) circuitry throughout, in both the array and the decoding, therefore it requires no clocks or refreshing to operate. Data access is particularly simple since address setup times are not required. The data is read out nondestructively and has the same polarity as the input data. Common input/output pins are provided.

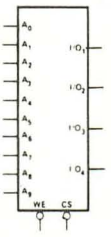
The 2114A is designed for memory applications where the high performance and high reliability of HMOS, low cost, large bit storage, and simple interfacing are important design objectives. The 2114A is placed in an 18-pin package for the highest possible density.

It is directly TTL compatible in all respects: inputs, outputs, and a single +5V supply. A separate Chip Select ($\overline{CS}$) lead allows easy selection of an individual package when outputs are or-tied.

PIN CONFIGURATION



LOGIC SYMBOL



PIN NAMES

A_0–A_9	ADDRESS INPUTS	V_{CC}	POWER (+5V)
$\overline{WE}$	WRITE ENABLE	GND	GROUND
$\overline{CS}$	CHIP SELECT		
I/O_1–I/O_4	DATA INPUT/OUTPUT		

BLOCK DIAGRAM

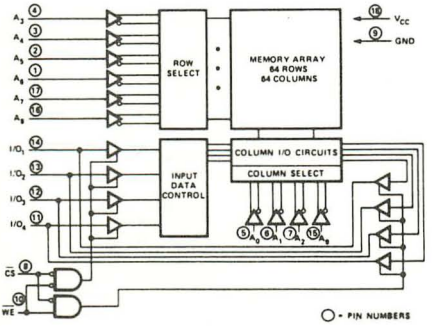


Abb. 2.6: *Teil eines Original-Datenblattes sowie das Blockschaltbild für das statische 1K * 4 RAM 2114.*

betrachten wir den Schreibvorgang. Der Ablauf ist der gleiche wie in Abschnitt 2.3 beschrieben. Wir wollen an dieser Stelle jedoch mehr ins Detail gehen und die aktuellen Spezifikationen des zu untersuchenden RAM berücksichtigen.

Ereignisfolge beim Schreiben der Daten zum 2114

1. Die Adresse für den Datentransfer wird an die Adreßeingänge gelegt.

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ± 10%, unless otherwise noted.

READ CYCLE [1]

SYMBOL	PARAMETER	2114AL-1 Min.	2114AL-1 Max.	2114AL-2 Min.	2114AL-2 Max.	2114AL-3 Min.	2114AL-3 Max.	2114A-4/L-4 Min.	2114A-4/L-4 Max.	2114A-5 Min.	2114A-5 Max.	UNIT
t_{RC}	Read Cycle Time	100		120		150		200		250		ns
t_A	Access Time		100		120		150		200		250	ns
t_{CO}	Chip Selection to Output Valid		70		70		70		70		85	ns
t_{CX}	Chip Selection to Output Active	10		10		10		10		10		ns
t_{OTD}	Output 3-state from Deselection		30		35		40		50		60	ns
t_{OHA}	Output Hold from Address Change	15		15		15		15		15		ns

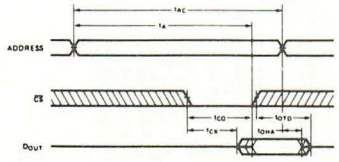
WRITE CYCLE [2]

SYMBOL	PARAMETER	2114AL-1 Min.	2114AL-1 Max.	2114AL-2 Min.	2114AL-2 Max.	2114AL-3 Min.	2114AL-3 Max.	2114A-4/L-4 Min.	2114A-4/L-4 Max.	2114A-5 Min.	2114A-5 Max.	UNIT
t_{WC}	Write Cycle Time	100		120		150		200		250		ns
t_W	Write Time	75		75		90		120		135		ns
t_{WR}	Write Release Time	0		0		0		0		0		ns
t_{OTW}	Output 3-state from Write		30		35		40		50		60	ns
t_{DW}	Data to Write Time Overlap	70		70		90		120		135		ns
t_{DH}	Data Hold from Write Time	0		0		0		0		0		ns

NOTES:
1. A Read occurs during the overlap of a low $\overline{CS}$ and a high $\overline{WE}$.
2. A Write occurs during the overlap of a low $\overline{CS}$ and a low $\overline{WE}$. t_W is measured from the latter of $\overline{CS}$ or $\overline{WE}$ going low to the earlier of $\overline{CS}$ or $\overline{WE}$ going high.

WAVEFORMS

READ CYCLE ③



NOTES:
3. $\overline{WE}$ is high for a Read Cycle.
4. If the $\overline{CS}$ low transition occurs simultaneously with the $\overline{WE}$ low transition, the output buffers remain in a high impedance state.
5. $\overline{WE}$ must be high during all address transitions.

WRITE CYCLE

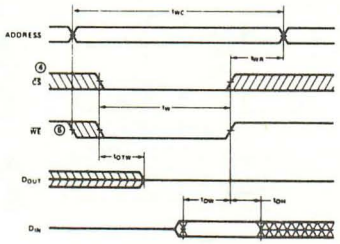


Abb. 2.7: Genaues Datenblatt mit Timing-Parameter für den 2114.

2. Daten werden an die Daten-I/O-Leitungen des Bauteils gelegt. Diese Leitungen sind tri-state. Wäre das nicht der Fall, würde es zu einem Konflikt kommen zwischen den zu schreibenden Daten und den bereits im RAM befindlichen (siehe Abb. 2.8).
3. Als nächstes wird $\overline{WE}$ und etwa zur gleichen Zeit $\overline{CS}$ angelegt. Sobald $\overline{WE}$ logisch 0 wird, werden die I/O-Leitungen zu Eingängen. Vorher sollte der Speicher nicht selektiert werden (siehe Abb. 2.9).

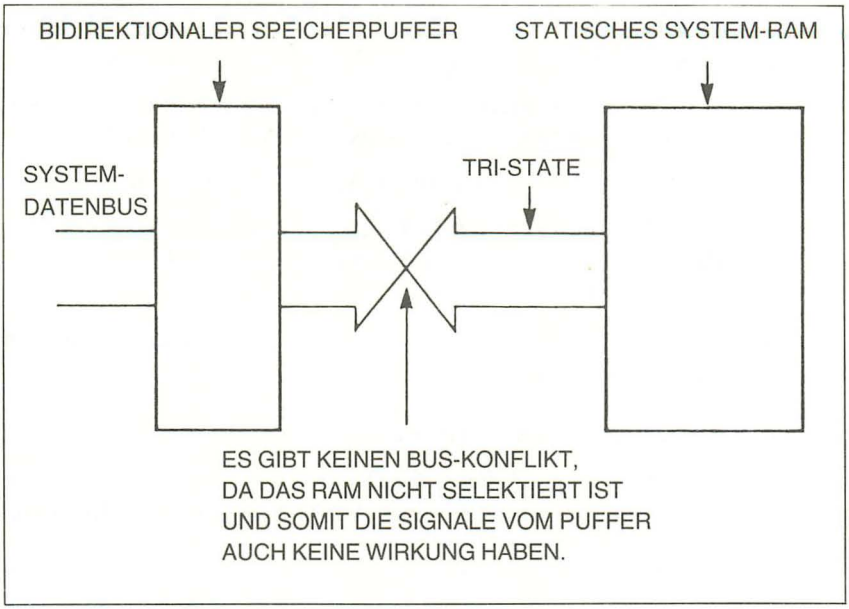


Abb. 2.8: *Blockschaltbild für einen möglichen Buskonflikt zwischen den 2114-Datenausgängen und den Speicher-Datenpuffer-Ausgängen. Dieser Konflikt kann durch entsprechende Behandlung der Richtungs-Steuerleitung des Datenpuffers vermieden werden.*

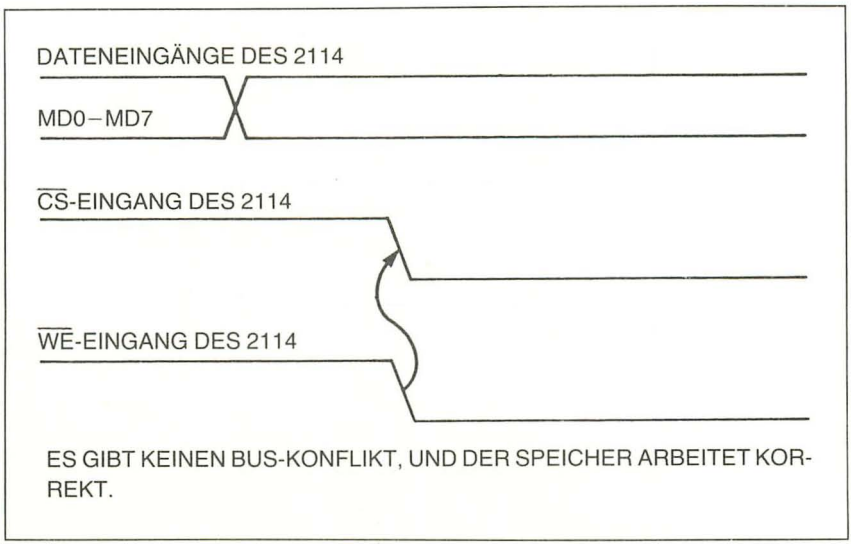


Abb. 2.9: *Timing für die korrekte Ansteuerung des RAM.*

Ereignisfolge beim Lesen der Daten vom 2114

1. Die Adresse für den Datentransfer wird an die Adreßeingänge gelegt.
2. Als nächstes wird $\overline{CS}$ angelegt. Zu diesem Zeitpunkt ist $\overline{WE}$ logisch 1. Die Daten-I/O-Leitungen werden als Ausgänge aktiviert.
3. Die gespeicherten Daten werden über die I/O-Leitungen ausgegeben.

Wie wir nochmals sehen, arbeitet die Leseoperation beim RAM genauso wie beim ROM. Deshalb gelten auch die gleichen Vorschriften für das Timing und die Pufferung.

Nachdem wir nun wissen, wie der 2114 arbeitet, wollen wir ihn an den Z80-Mikroprozessor schließen.

Anschluß der Adreßleitungen an den Z80

Um zu illustrieren, wie der Z80 mit dem RAM kommuniziert, wollen wir jetzt den vollständigen Entwurf eines RAM-Systems für allgemeine

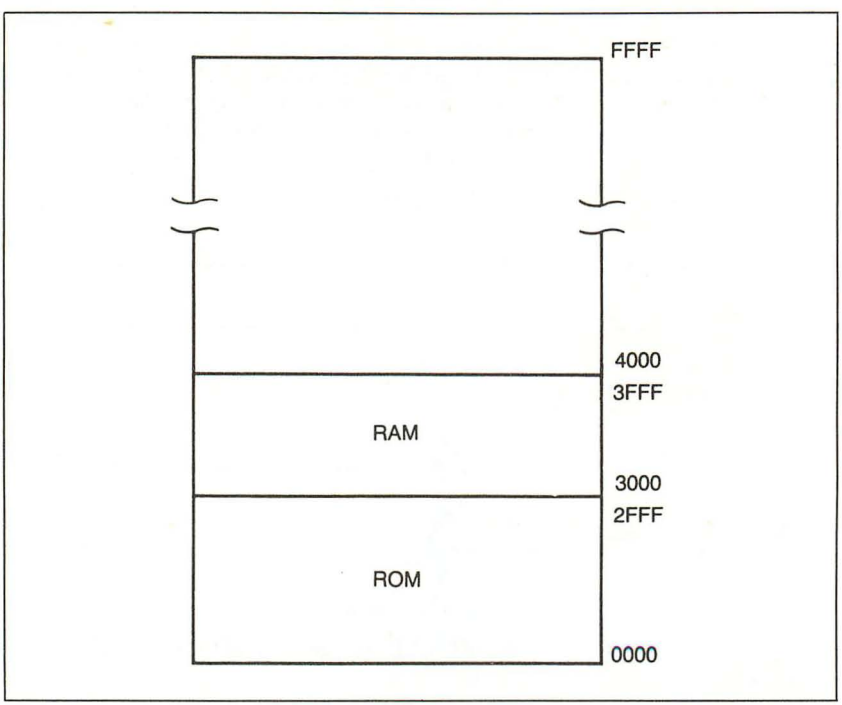


Abb. 2.10: *Ein Ausschnitt aus der Memory-Map mit ROM- und RAM-Belegung im Systemspeicher.*

Anwendungen vorstellen. Wir werden dabei auch die besonderen Problembereiche erforschen, die sich bei der Zusammenschaltung mit dem Z80 ergeben können. Die Informationen, die hierbei anfallen, sind jedoch so allgemein, daß sie für jeden nützlich sind, der statische RAMs einsetzt in Anwendungen mit dem Z80-Mikroprozessor.

Wie der Adreßbus mit dem RAM-System verbunden wird, können Sie aus der Memory-Map Abb. 2.10 ersehen. Die Memory-Map hilft uns zu sehen, wo im (möglichen) 64K Adreßraum das RAM liegt. In unserem Fall liegt es zwischen 3000 und 3FFF hexadezimal. Das ergibt eine RAM-Größe von 4096 bzw. 4K * 8 Bits für dieses System.

Wir wollen also einen 4K * 8 RAM-Speicher für unser System entwickeln. Wenn wir uns das Datenblatt in Abb. 2.6 ansehen, bemerken wir, daß jeder Speicherbaustein 4 I/O-Leitungen besitzt. Da der Z80-Mikropro-

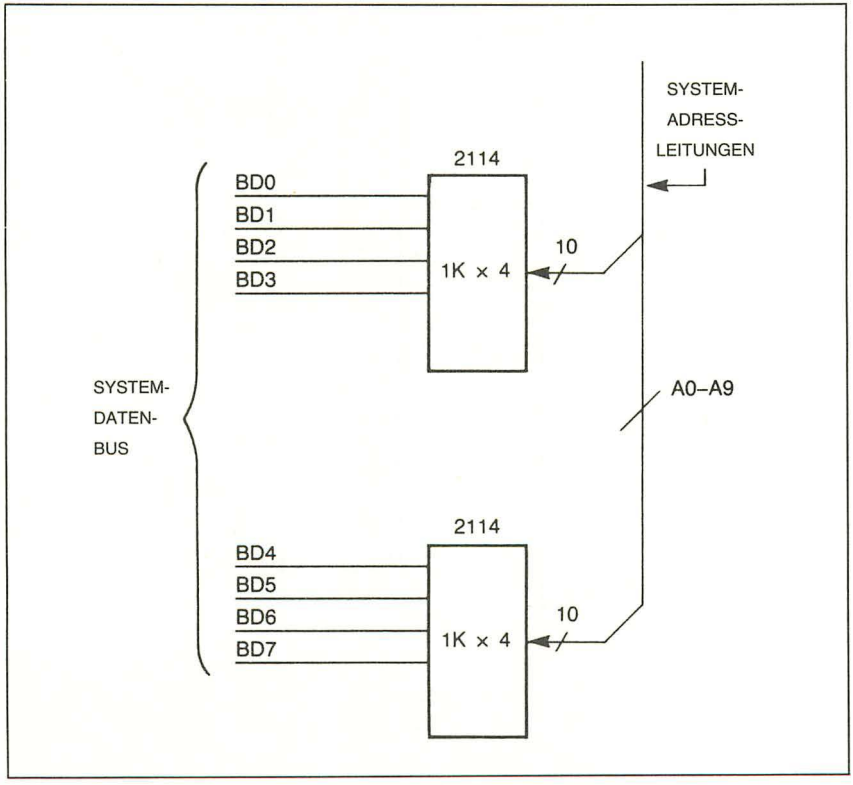


Abb. 2.11: *Zwei 2114-Bausteine werden parallel geschaltet, um ein komplettes 8-Bit-weites Speichersystem zu konstruieren.*

zessor 8 Datenleitungen benutzt, müssen zwei RAM-Bausteine parallel geschaltet werden (siehe Abb. 2.11).

Als erstes müssen wir berechnen, wieviele RAMs für den geforderten Speicher benötigt werden. Das kann auf folgende Weise geschehen: Benötigt werden 4096 Bytes RAM. Jeder 2114-Baustein speichert 1024 Halbbytes, das heißt, für jeweils 1024 Bytes werden 2 Bausteine benötigt. Wir brauchen also acht 2114-Bausteine für den gesamten Speicher.

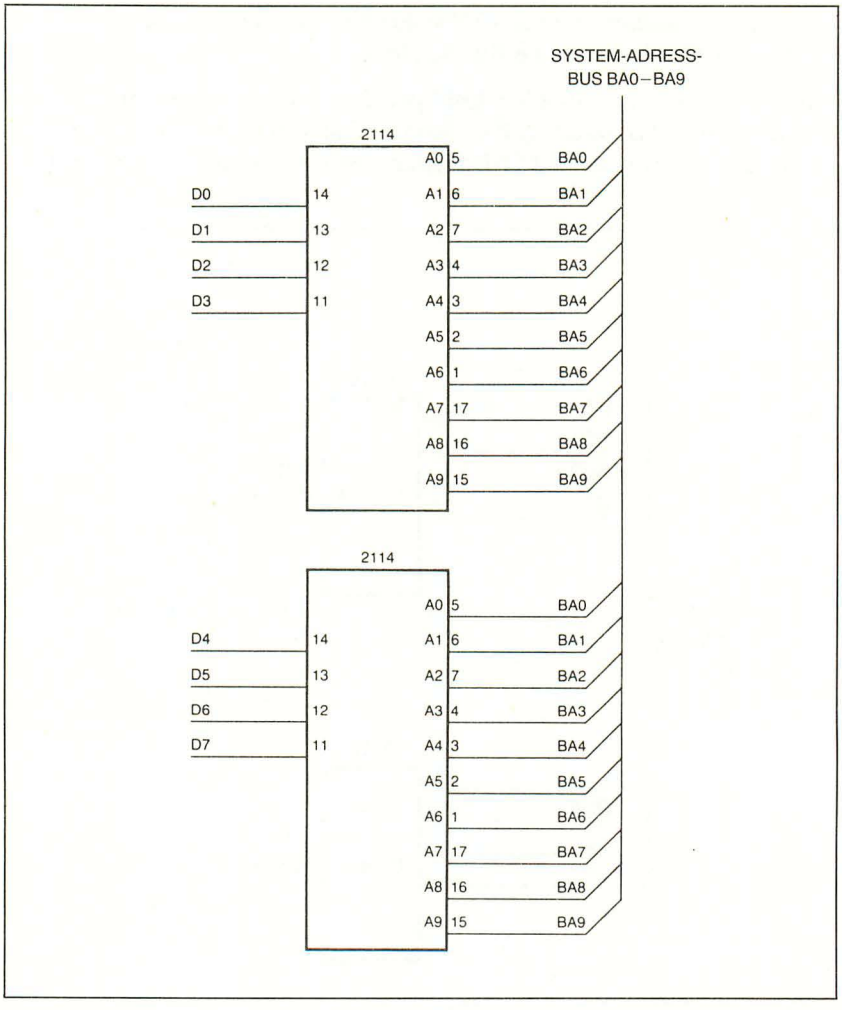


Abb. 2.12: *Die Adreßleitungen A0-A9 aller Speicher werden in paraller Anordnung geschaltet.*

In unserem Beispiel braucht das Z80-System nur 1 KByte RAM. Wir nehmen jedoch an, daß das System für 4 KByte ausgelegt ist. (Anm.: Es ist normalerweise eine gute Idee, ein System für einen größeren Speicherbereich auszulegen, als tatsächlich benötigt wird. Eine spätere Erweiterung wird dadurch wesentlich vereinfacht. Beachten Sie, daß eine spätere Erweiterung ohne entsprechende Auslegung des Systemkonzepts sehr schwierig ist.)

Wir wissen jetzt, daß unser System zur Realisierung des Speichers acht 2114-Bausteine benötigt. Jeder 2114 hat 10 Adreß-Eingangsleitungen A0 bis A9, die in paralleler Anordnung verbunden werden müssen. Das bedeutet, daß alle A0-Leitungen zusammengelegt werden, alle A1-Leitungen usw. (siehe Abb. 2.12).

Die anderen Adreßleitungen A10 bis A15 werden benutzt, um den RAM-Bereich aus dem Gesamt-Speicherbereich zu decodieren. Zwei Ebenen der Decodierung werden verwendet. Die Adreßleitungen A10 und A11 dienen zur Auswahl des richtigen 2114-Speicherpaares, während die Leitungen A10 bis A15 benutzt werden, um den Speicherbereich 3000 bis 3FFF aus dem Gesamtspeicherbereich herauszugreifen.

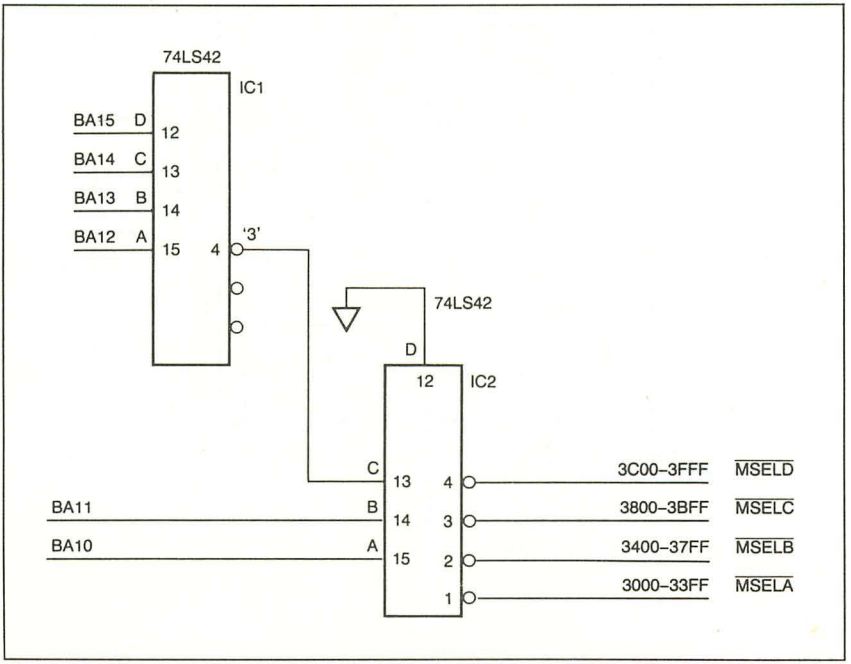


Abb. 2.13: *Schaltung zur Speicherselektierung.*

Es gibt verschiedene Techniken zum Decodieren von Adreßleitungen. Eine typische Schaltung wird in Abb. 2.13 gezeigt. Sie soll Ihnen eine allgemeine Vorstellung darüber geben, wie eine logische Speicherselektierung aussehen kann. Obwohl diese Schaltung lauffähig ist, stellt sie keine Universallösung dar. Alle Lösungen müssen den Systemvoraussetzungen angepaßt sein und sind normalerweise abhängig von der Systemgeschwindigkeit, der Busbelastung und der endgültigen Speicheraufteilung. Abb. 2.13 zeigt die Hardware-Anforderungen zur Realisierung der Zwei-Ebenen-Decodierung des RAM für unser Beispiel.

Beachten Sie, daß wahlweise Adreßpuffer verwendet werden können. Der Einsatz von Puffern wurde in Kapitel 1 beschrieben. Für RAM und ROM gelten die gleichen Regeln bezüglich Last und Geschwindigkeit.

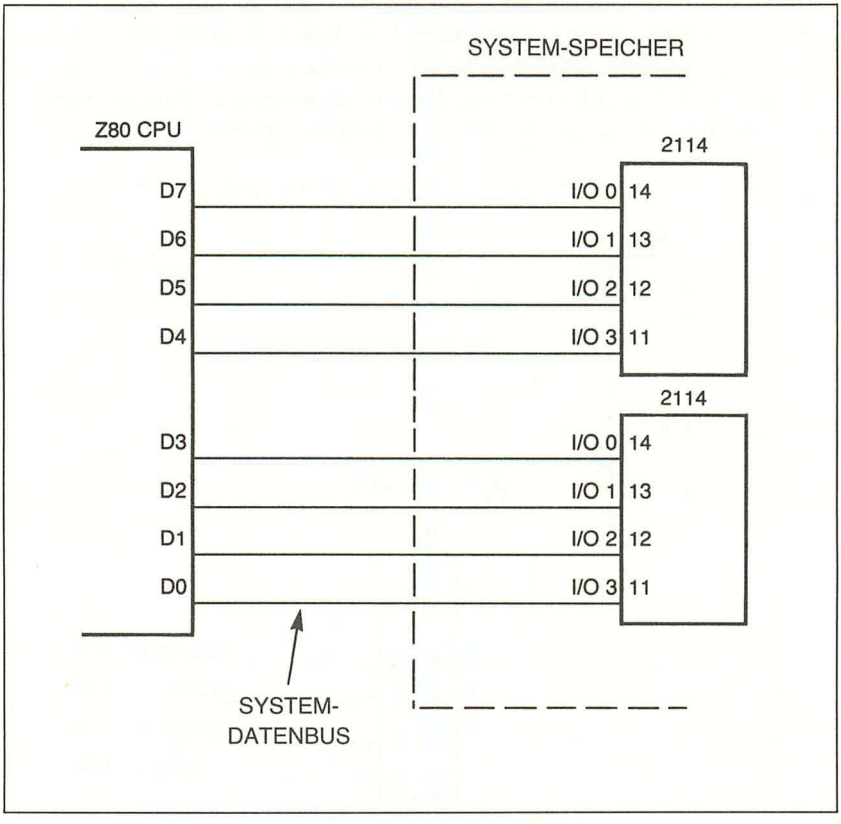


Abb. 2.14: *Physikalische Verbindung zwischen 2114-Speicher und Z80-Mikroprozessor.*

Verbinden der Datenleitungen – ungepuffert

Wir wollen nun zeigen, wie die Datenleitungen des Speichers an den Z80-Datenbus gekoppelt werden. In diesem Beispiel benutzen wir keine Datenpuffer. Wir gehen davon aus, daß die Leitungsbelastung die in den Datenblättern angegebenen Werte nicht überschreitet. Die Datenleitungen der 2114-Bausteine werden dabei einfach parallel miteinander und mit dem Z80-Datenbus verbunden (siehe Abb.2.14).

Die I/O-Leitungen des 2114-Speichers werden über die Anschlüsse $\overline{WE}$ und $\overline{CS}$ gesteuert. Wenn Ihre Anwendung keine Datenpuffer benötigt, reicht eine einfache Direktverbindung zwischen Speicher und Z80-Datenbus aus.

Erzeugung der Systemspeicher-Lese- und Schreibsignale

Wenn der Z80-Mikroprozessor elektrisch mit dem Speicher kommuniziert, so stehen die drei physikalischen Ausgänge $\overline{MREQ}$,$\overline{RD}$,$\overline{WR}$ zur Verfügung, um die Speicher-Steuersignale $\overline{MEMR}$ und $\overline{MEMW}$ zu generieren. Abb.2.15 zeigt eine typische Schaltung zur Generierung des Systemspeicher-Lese- und Schreibsignals aus den drei Z80-Ausgängen.

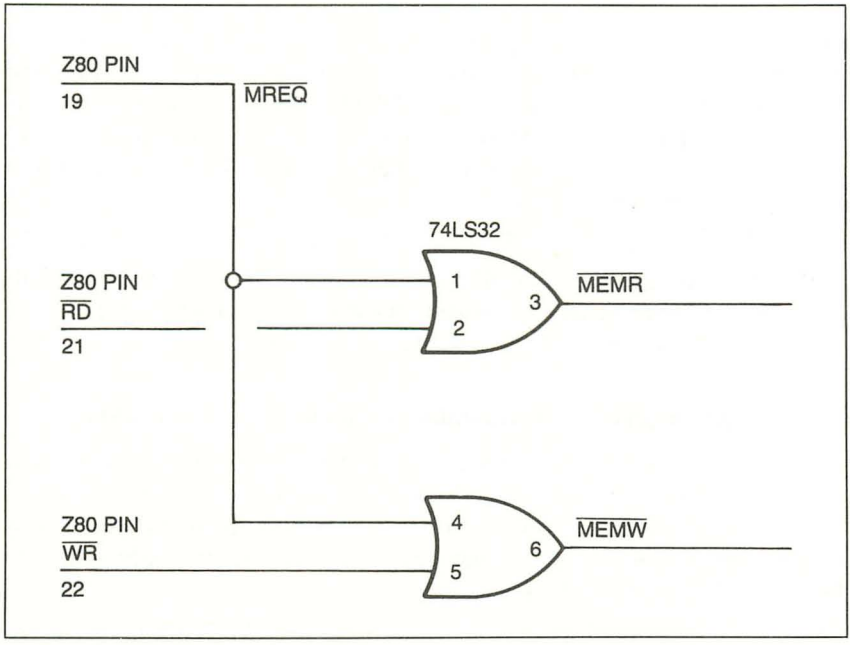


Abb. 2.15: *Schaltung zur Erzeugung des Systemspeicher-Lese- und Schreibsignals.*

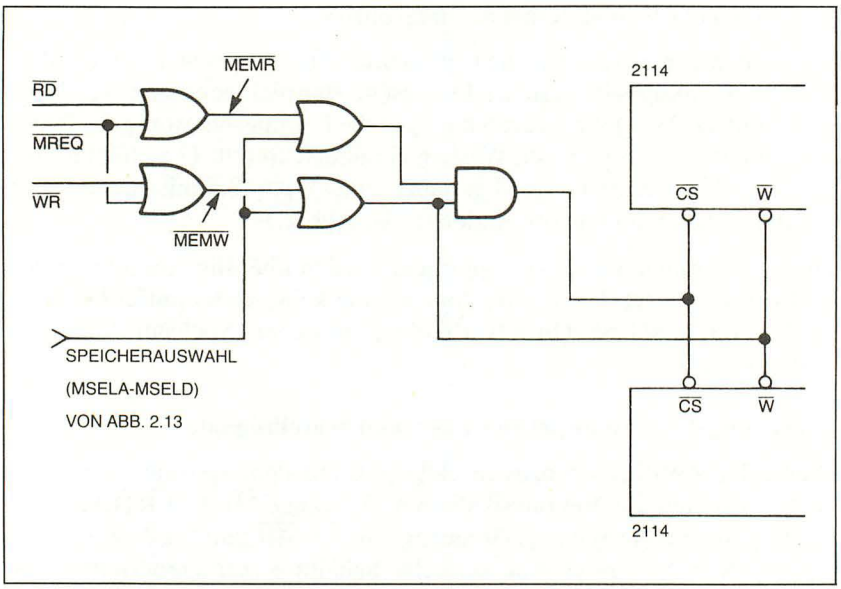


Abb. 2.16: *Beschaltung der $\overline{CS}$- und $\overline{WE}$-Eingänge der 2114-Bausteine.*

Wir wollen jetzt die Ausgangsleitungen aus Abb.2.15 zusammen mit der Speicher-Adreß-Decodierung aus Abb.2.13 benutzen, um die $\overline{WE}$- und $\overline{CS}$-Eingänge der RAMs zu steuern. (Anm.: Die hier gegebenen Informationen beziehen sich nicht ausschließlich auf den 2114-Bausstein, sondern sind für die meisten statischen Speicher gültig. Der 2114 dient hier nur als Mittel zur Darstellung.)

Abb.2.16 zeigt, wie die $\overline{MEMR}$- und $\overline{MEMW}$-Leitungen mit den $\overline{WE}$- und $\overline{CS}$-Leitungen des 2114-Bausteins verbunden werden. Die Schaltung aktiviert zum Schreiben sowohl die $\overline{WE}$- als auch die $\overline{CS}$-Leitung, während zum Lesen nur die $\overline{CS}$-Leitung des 2114 aktiviert wird.

Benutzung von gepufferten Datenleitungen mit statischen RAMs

Im letzten Abschnitt haben wir die 2114-Datenleitungen direkt mit den Z80-Datenleitungen verbunden. Diese Verbindungsart wird benutzt, wenn die gesamte Systemkonzeption keine Datenpufferung vorsieht. Wir wollen jetzt davon ausgehen, daß Ihre Anwendung eine bidirektionale Datenbus-Pufferung benötigt.

Abb.2.17 zeigt eine Möglichkeit, ein statisches RAM-System mit Datenpuffern auszurüsten. Die 2114-Datenleitungen sind dabei wie im vorigen

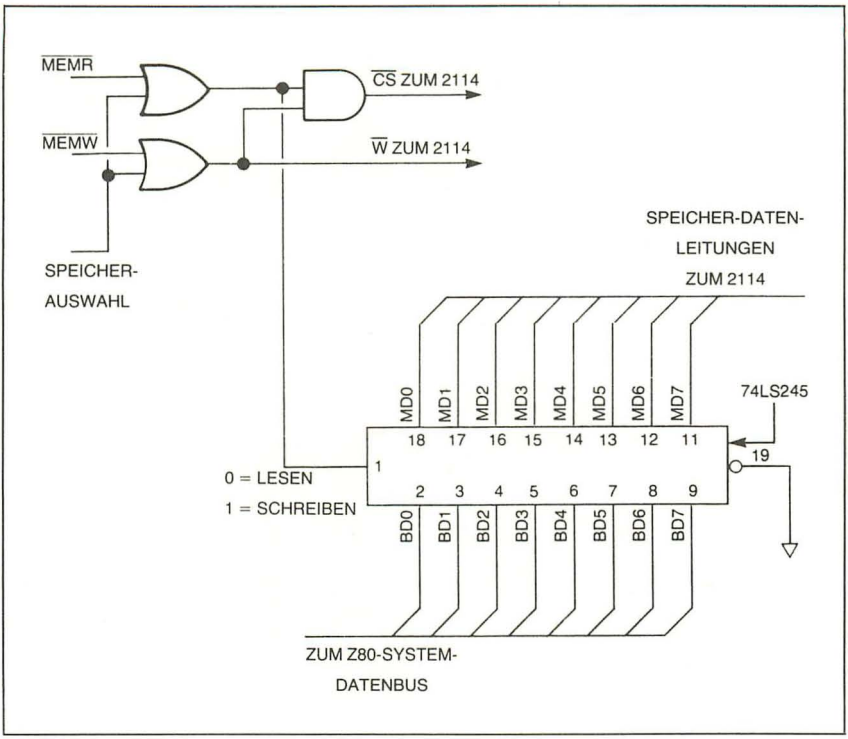


Abb. 2.17: *Schaltungsauszug einer Datenpufferung für den 2114-Speicherbaustein. Es wird hierzu der bidirektionale Pufferbaustein 74LS245 verwendet, da der Speicher mit gemeinsamen Ein- und Ausgangsleitungen (common I/O) arbeitet.*

Beispiel miteinander verbunden, jedoch nicht direkt an den Z80-Datenbus gekoppelt. Statt dessen liegen sie auf der einen Seite eines bidirektionalen Puffers. Die Richtungssteuerung (direction control) für diesen Puffer ist abhängig vom logischen Zustand des $\overline{\text{MEMR}}$-Signals. Immer, wenn dieses Signal logisch 0 und der Speicherbereich selektiert ist, werden die Daten über den Puffer auf den Datenbus gelegt.

Komplettes 2K * 8Bit statisches RAM

Abb.2.18 zeigt ein komplettes statisches RAM-System für den Z80-Mikroprozessor. Es enthält einen Datenpuffer. Wird er in Ihrer Anwendung nicht benötigt, so kann er einfach weggelassen werden, indem Sie die MD-Leitungen direkt an die Datenbusleitungen anschließen.

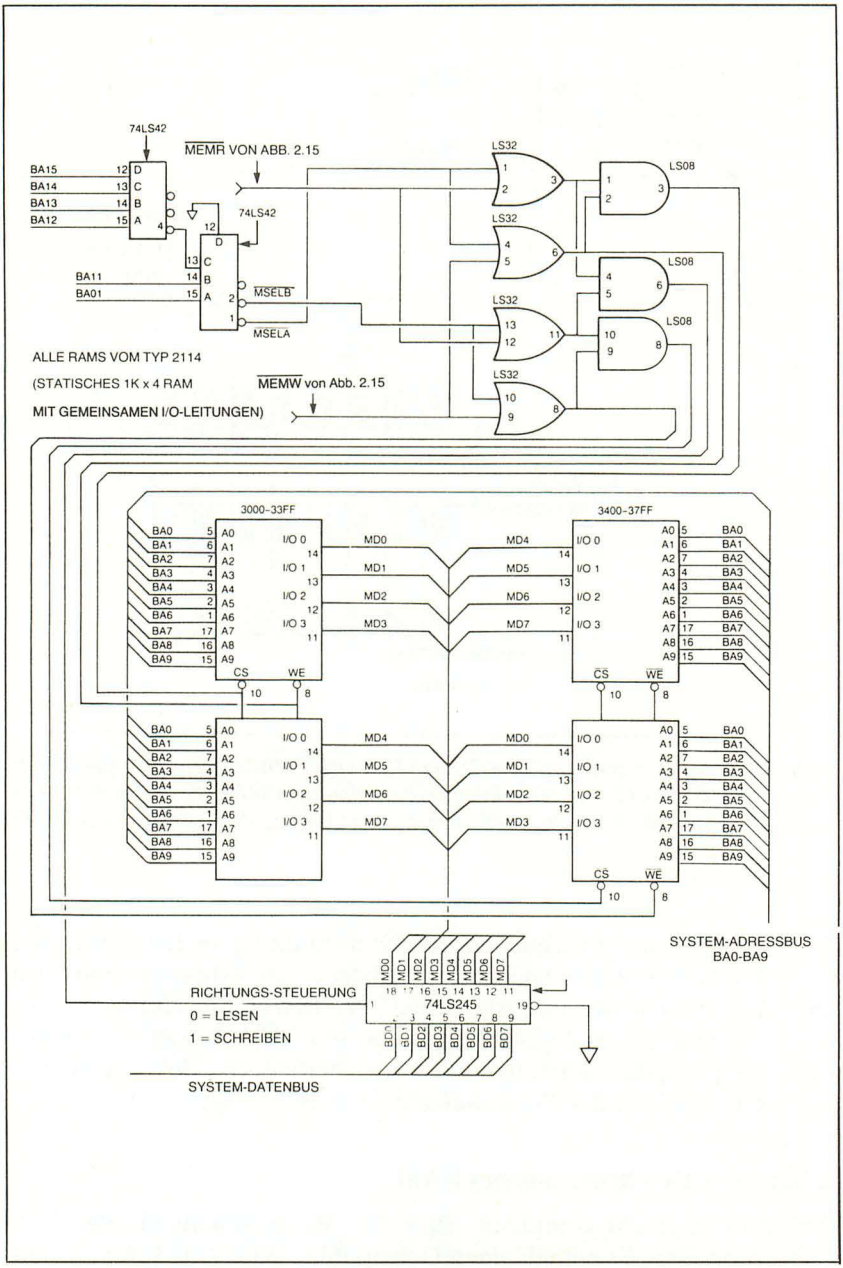


Abb. 2.18: *Komplette Schaltung eines 2K * 8 Bit Speichersystems für den Z80-Mikroprozessor.*

Der 6116: Ein anderes statisches RAM-Bauteil

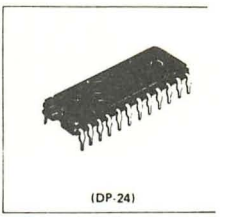
In diesem Abschnitt wollen wir ein anderes, häufig benutztes RAM vorstellen: den 6116, ein 2K * 8 statisches RAM mit gemeinsamen I/O-Leitungen. Abb. 2.19 zeigt einen Datenblattauszug für dieses Bauteil. Die Arbeitsweise des 6116 entspricht weitgehend der des 2114-RAM.

HM6116P-2, HM6116P-3, HM6116P-4

2048-word X 8-bit High Speed Static CMOS RAM

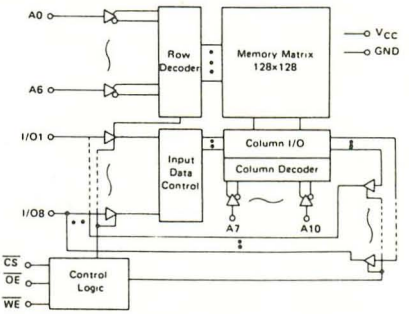
■ FEATURES

- Single 5V Supply and High Density 24 pin Package
- High Speed: Fast Access Time 120ns/150ns/200ns (max.)
- Low Power Standby and Low Power Operation; Standby: 100μW (typ.)
  Operation: 180mW (typ.)
- Completely Static RAM: No clock or Timing Strobe Required
- Directly TTL Compatible: All Input and Output
- Pin Out Compatible with Standard 16K EPROM/MASK ROM
- Equal Access and Cycle Time

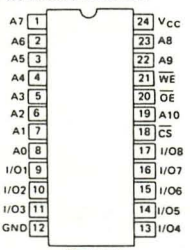


(DP-24)

■ FUNCTIONAL BLOCK DIAGRAM



■ PIN ARRANGEMENT



(Top View)

■ ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Rating	Unit
Voltage on Any Pin Relative to GND	V_{IN}	−0.5 to +7.0	V
Operating Temperature	T_{opr}	0 to +70	°C
Storage Temperature	T_{stg}	−55 to +125	°C
Temperature Under Bias	T_{bias}	−10 to +85	°C
Power Dissipation	P_T	1.0	W

■ TRUTH TABLE

$\overline{CS}$	$\overline{OE}$	$\overline{WE}$	Mode	V_{CC} Current	I/O Pin	Ref. Cycle
H	X	X	Not Selected	I_{SB}, I_{SB1}	High Z	
L	L	H	Read	I_{CC}	D_{out}	Read Cycle (1) ~ (3)
L	H	L	Write	I_{CC}	D_{in}	Write Cycle (1)
L	L	L	Write	I_{CC}	D_{in}	Write Cycle (2)

Abb. 2.19: *Original-Datenblatt des 6116 - ein 2K * 8 Bit statisches RAM mit gemeinsamen I/O-Leitungen.*

■ RECOMMENDED DC OPERATING CONDITIONS (T_a = 0 to +70°C)

Item	Symbol	min	typ.	max	Unit
Supply Voltage	V_{CC}	4.5	5.0	5.5	V
	GND	0	0	0	V
Input Voltage	V_{IH}	2.2	3.5	6.0	V
	V_{IL}	−1.0*	–	0.8	V

* Pulse Width: 50 ns, DC: V_{IL} min = −0.3V

■ DC AND OPERATING CHARACTERISTICS (V_{CC} = 5V ±10%, GND = 0V, T_a = 0 to +70°C)

Item	Symbol	Test Conditions	HM6116P-2			HM6116P-3/-4			Unit
			min	typ*	max	min	typ*	max	
Input Leakage Current	$\|I_{LI}\|$	V_{CC} = 5.5V, V_{in} = GND to V_{CC}	–	–	10	–	–	10	μA
Output Leakage Current	$\|I_{LO}\|$	$\overline{CS}$ = V_{IH} or $\overline{OE}$ = V_{IH}, $V_{I/O}$ = GND to V_{CC}	–	–	10	–	–	10	μA
Operating Power Supply Current	I_{CC}	$\overline{CS}$ = V_{IL}, $I_{I/O}$ = 0mA	–	40	80	–	35	70	mA
	I_{CC1}**	V_{IH} = 3.5V, V_{IL} = 0.6V, $I_{I/O}$ = 0mA	–	35	–	–	30	–	mA
Average Operating Current	I_{CC2}	Min. cycle, duty = 100%	–	40	80	–	35	70	mA
Standby Power Supply Current	I_{SB}	$\overline{CS}$ = V_{IH}	–	5	15	–	5	15	mA
	I_{SB1}	$\overline{CS} \geq V_{CC}$ −0.2V, $V_{in} \geq V_{CC}$ −0.2V or $V_{in} \leq$ 0.2V	–	0.02	2	–	0.02	2	mA
Output Voltage	V_{OL}	I_{OL} = 4mA	–	–	0.4	–	–	–	V
		I_{OL} = 2.1mA	–	–	–	–	–	0.4	
	V_{OH}	I_{OH} = −1.0mA	2.4	–	–	2.4	–	–	V

*: V_{CC} = 5V, T_a = 25°C
**: Reference Only

■ AC CHARACTERISTICS (V_{CC} = 5V ±10%, T_a = 0 to +70°C)

● AC TEST CONDITIONS

Input Pulse Levels: 0.8 to 2.4V
Input Rise and Fall Times: 10 ns
Input and Output Timing Reference Levels: 1.5V
Output Load: 1TTL Gate and C_L = 100pF (including scope and Jig)

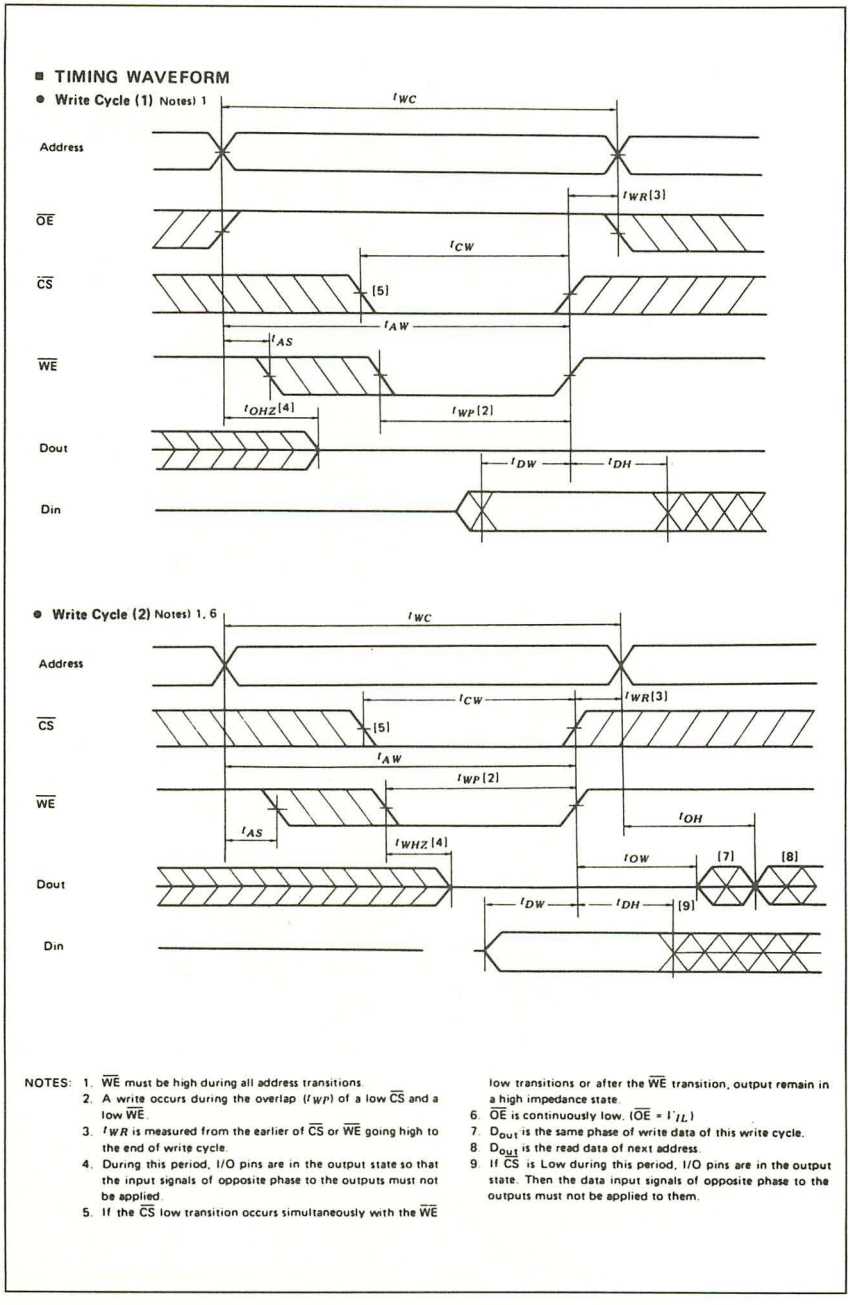
● READ CYCLE

Item	Symbol	HM6116P-2		HM6116P-3		HM6116P-4		Unit
		min	max	min	max	min	max	
Read Cycle Time	t_{RC}	120	–	150	–	200	–	ns
Address Access Time	t_{AA}	–	120	–	150	–	200	ns
Chip Select Access Time	t_{ACS}	–	120	–	150	–	200	ns
Chip Selection to Output in Low Z	t_{CLZ}	10	–	15	–	15	–	ns
Output Enable to Output Valid	t_{OE}	–	80	–	100	–	120	ns
Output Enable to Output in Low Z	t_{OLZ}	10	–	15	–	15	–	ns
Chip deselection to Output in High Z	t_{CHZ}	0	40	0	50	0	60	ns
Chip Disable to Output in High Z	t_{OHZ}	0	40	0	50	0	60	ns
Output Hold from Address Change	t_{OH}	10	–	15	–	15	–	ns

● WRITE CYCLE

Item	Symbol	HM6116P-2		HM6116P-3		HM6116P-4		Unit
		min	typ	min	max	min	max	
Write Cycle Time	t_{WC}	120	–	150	–	200	–	ns
Chip Selection to End of Write	t_{CW}	70	–	90	–	120	–	ns
Address Valid to End of Write	t_{AW}	105	–	120	–	140	–	ns
Address Set Up Time	t_{AS}	20	–	20	–	20	–	ns
Write Pulse Width	t_{WP}	70	–	90	–	120	–	ns
Write Recovery Time	t_{WR}	5	–	10	–	10	–	ns
Output Disable to Output in High Z	t_{OHZ}	0	40	0	50	0	60	ns
Write to Output in High Z	t_{WHZ}	0	50	0	60	0	60	ns
Data to Write Time Overlap	t_{DW}	35	–	40	–	60	–	ns
Data Hold from Write Time	t_{DH}	5	–	10	–	10	–	ns
Output Active from End of Write	t_{OW}	5	–	10	–	10	–	ns

Abb. 2.19: *Fortsetzung des Original-Datenblattes.*



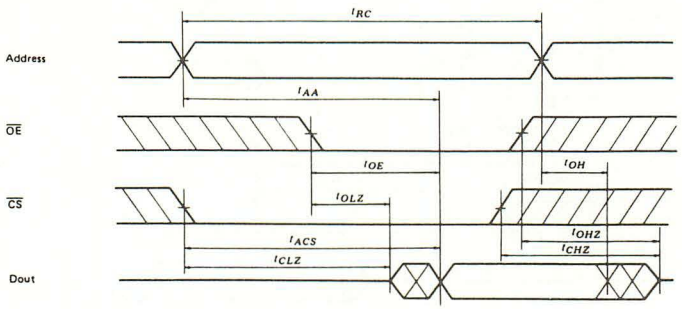
NOTES:
1. $\overline{WE}$ must be high during all address transitions.
2. A write occurs during the overlap (t_{WP}) of a low $\overline{CS}$ and a low $\overline{WE}$.
3. t_{WR} is measured from the earlier of $\overline{CS}$ or $\overline{WE}$ going high to the end of write cycle.
4. During this period, I/O pins are in the output state so that the input signals of opposite phase to the outputs must not be applied.
5. If the $\overline{CS}$ low transition occurs simultaneously with the $\overline{WE}$ low transitions or after the $\overline{WE}$ transition, output remain in a high impedance state.
6. $\overline{OE}$ is continuously low. ($\overline{OE}$ = V_{IL})
7. D_{out} is the same phase of write data of this write cycle.
8. D_{out} is the read data of next address.
9. If $\overline{CS}$ is Low during this period, I/O pins are in the output state. Then the data input signals of opposite phase to the outputs must not be applied to them.

Abb. 2.19: *Fortsetzung des Original-Datenblattes.*

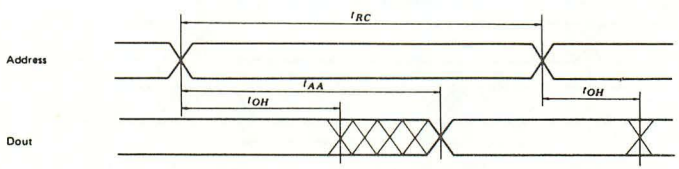
■ **CAPACITANCE** (f = 1MHz, T_a = 25°C)

Item	Symbol	Test Conditions	typ.	max.	Unit
Input Capacitance	C_{in}	V_{in} = 0V	3	5	pF
Input/Output Capacitance	$C_{I/O}$	$V_{I/O}$ = 0V	5	7	pF

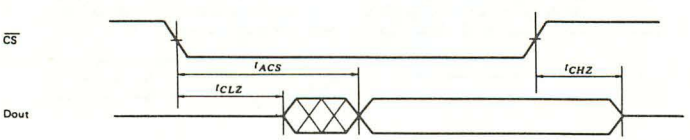
● **Read Cycle (1)** Notes) 1, 5



● **Read Cycle (2)** Notes) 1, 2, 4, 5



● **Read Cycle (3)** Notes) 1, 3, 4, 5



NOTES:
1. $\overline{WE}$ is High for Read Cycle.
2. Device is continuously selected, $\overline{CS} = V_{IL}$.
3. Address Valid prior to or coincident with $\overline{CS}$ transition Low.
4. $\overline{OE} = V_{IL}$.
5. When $\overline{CS}$ is Low, the address input must not be in the high impedance state.

Abb. 2.19: *Fortsetzung des Original-Datenblattes.*

Abb. 2.20 zeigt eine Möglichkeit, das 6116-RAM an ein Z80-System zu schließen. Aufgrund der Organisation dieses Bausteins wird für den gesamten RAM-Bereich nur ein zusätzliches Bauteil benötigt. Das macht den 6116 zu einem idealen Baustein vor allem in kleineren Systemen.

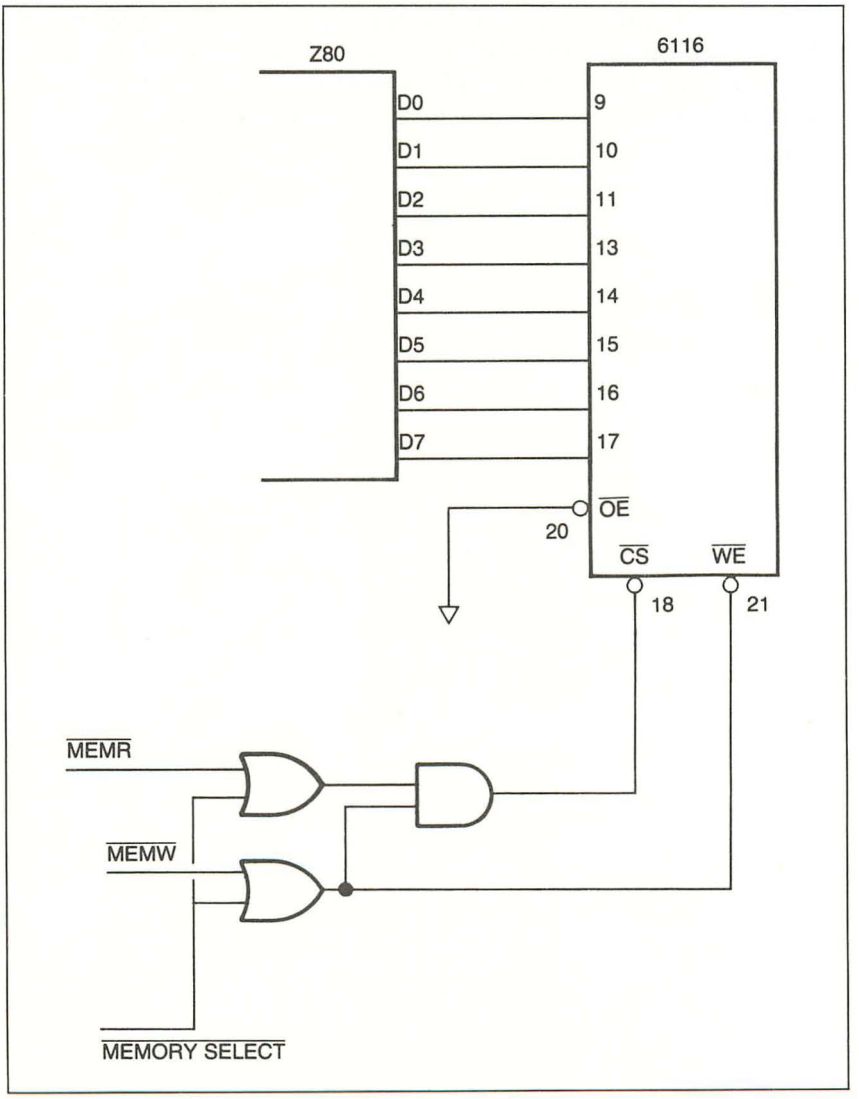


Abb. 2.20: *In dieser Schaltung wird die Verbindung des 6116-Speichers mit dem Z80-Mikroprozessor gezeigt.*

Zusammenfassung

In diesem Kapitel haben wir wichtige Informationen über den Gebrauch von statischen RAMs in einem Z80-Mikroprozessor-System gegeben. Wir begannen mit einer Beschreibung des allgemeinen Ereignisablaufes bei Lese- und Schreiboperationen. Wir stellten dann ein typisches RAM-System vor mit dem 2114, einem 1K * 4 Speicher. Wichtige Speicherdaten und Details beim Lesen und Schreiben wurden aufgezeigt. Dann untersuchten wir ein vollständiges 2K * 8 Bit Speichersystem, das den 2114 einsetzt. Zum Schluß wurde das gleiche Speichersystem noch mit dem 6116 aufgebaut.

Kapitel 3

Ein- und Ausgabe beim Z80

Einführung

Zwei der Hauptoperationen eines Mikroprozessors sind das Lesen von einem Eingabebauteil und das Schreiben zu einem Ausgabebauteil. In diesem Kapitel lernen wir, wie der Z80 elektrisch mit Ein- und Ausgabebauteilen (I/O Devices) kommuniziert. Außerdem werden wir ein allgemeines I/O-Port mit diskreten logischen Bauteilen entwickeln.

Wir haben die Besprechung von Ein- und Ausgabeoperationen an diese Stelle gesetzt, da die elektrische Kommunikation mit I/O-Bauteilen weitgehend der mit statischen RAMs entspricht. Tatsächlich werden Sie beim Fortschreiten in diesem Kapitel starke Ähnlichkeiten in der Kommunikation zwischen dem Mikroprozessor und statischen RAMs und dem Mikroprozessor und I/O-Bauteilen feststellen.

Die in diesem Kapitel gegebenen Informationen sind wichtig. Denn Sie benötigen ein grundlegendes Verständnis von I/O-Operationen, um den späteren Kapiteln folgen zu können.

Übersicht über Ein- und Ausgabe beim Z80

Der Z80 kann Daten von einem I/O-Bauteil lesen bzw. zu einem I/O-Bauteil schreiben. Diese beiden Operationsarten werden in Abb.3.1a und 3.1b gezeigt. Wenn der Z80 eine I/O-Operation durchführt, wird eine bestimmte Sequenz von elektrischen Signalen in der Systemhardware erzeugt. Das heißt, Adreß-, Daten- und Steuerleitungen werden in festgelegter Folge angesprochen. Dieser Ablauf gleicht dem beim Lesen und Schreiben von Speichern.

Es gibt bestimmte Software-Befehle, die, vom Z80 ausgeführt, den Ereignisablauf für eine I/O-Operation starten. (Wir werden diese Befehle später näher untersuchen, wenn wir die Bedienung verschiedener I/O-Bauteile vom Z80 aus besprechen.) Unabhängig von der Art des Befehls ist der Ereignisablauf für die Systemhardware immer derselbe.

Wie in Abb. 3.1a und 3.1b gezeigt wird, kann der Z80 jeweils 8 Bit von einem Eingangsbauteil einlesen oder 8 Bits zu einem Ausgangsbauteil ausgeben.

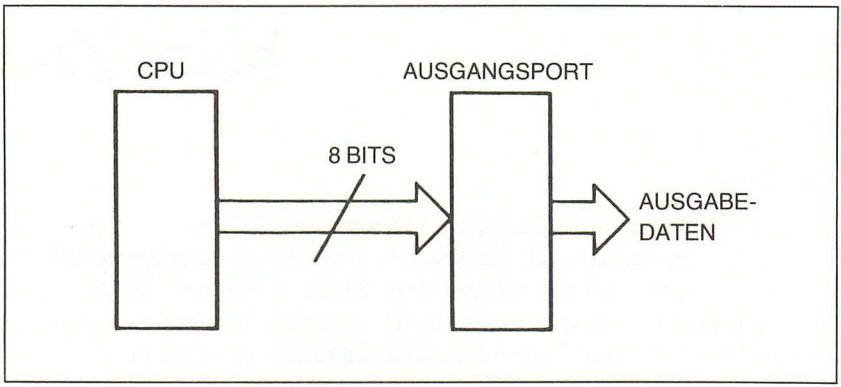


Abb. 3.1a: *Ein Mikroprozessor gibt 8 Bits parallel aus.*

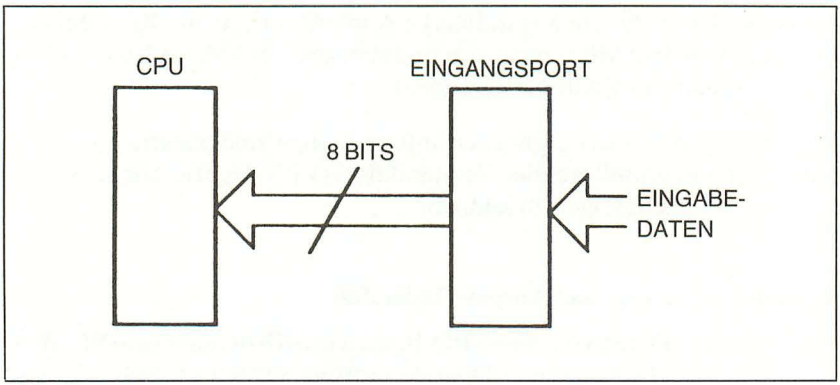


Abb. 3.1b: *Ein Mikroprozessor liest 8 Bits parallel ein.*

Port-Adressen

In einem Z80-System werden 8 der 16 Adreßleitungen für die I/Os benutzt. Das bedeutet, daß der Z80 bis zu 256 8-Bit I/O-Ports bedienen kann. (Vorausgesetzt, das System benutzt Standard „I/O mapped" Architektur und jedes Port belegt eine einzige Adresse.) Wird „Linear Select I/O" verwendet, wobei jedem Baustein eine Adreßleitung zugeordnet ist, so reduziert sich die Anzahl auf acht Ports. „Linear Select I/O" ist in

Systemen mit wenigen I/O-Bauteilen recht nützlich. (Anm.: In diesem Kapitel gehen wir von der normalen „I/O mapped" Architektur aus. Wenn Sie diese Architektur verstanden haben, dürften Sie keine Probleme beim Verständnis anderer Typen haben, da sie nur Variationen darstellen.)

Speicher- und I/O-Adressierung verwenden bei Z80-Systemen die gleichen Adreßleitungen. Allein der Mikroprozessor macht eine elekrische Unterscheidung zwischen Speicher- und I/O-Zugriff. Wie wir wissen, braucht der Systemspeicher nicht in allen Fällen alle 16 Adreßleitungen (A0-A15). Dasselbe gilt auch für das I/O-System: Oft werden nicht alle acht Adreßleitungen (A0-A7) benutzt. Werden z.B. nur fünf I/O-Ports eingesetzt, so genügen drei Adreßleitungen (da 3 Leitungen 8 verschiedene Kombinationen ergeben). Das allgemeine I/O-Port, das in diesem Kapitel vorgestellt wird, benutzt jedoch alle 8 Adreßleitungen zur Decodierung.

Wie wir wissen, ist jedes I/O-Port durch eine eindeutige Kombination der 8 Bits der System-Adreßleitungen A0 bis A7 ansprechbar. Die diesem elektrischen Zustand entsprechende Adreßkombination wird als Port-Adresse bezeichnet. FF ist die in diesem Kapitel benutzte Port-Adresse. Abb.3.2 zeigt die Schaltung zur Dekodierung dieser (und nur dieser) Port-Adresse.

In der Schaltung aus Abb. 3.2 ist der Ausgang Pin 8 des 74LS30 logisch 0, wenn alle Eingangsleitungen logisch 1 sind. Beachten Sie, daß der Ausgang Pin 8 des 74LS30 (genannt Port-Select-Leitung) immer logisch 0

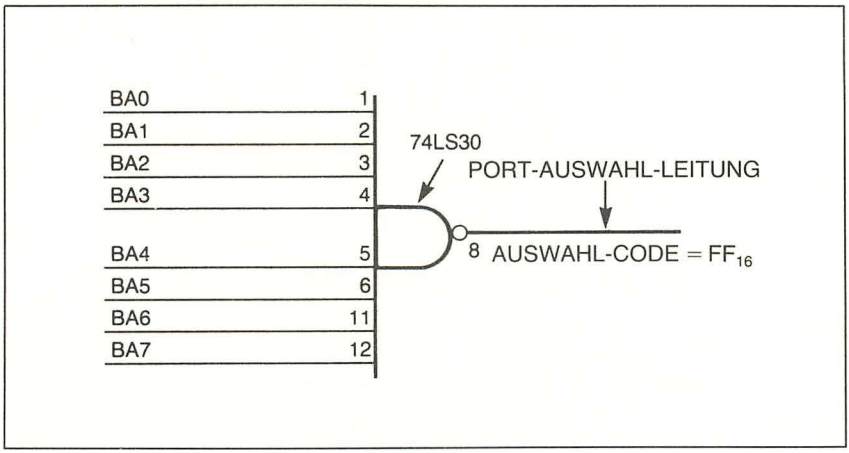


Abb. 3.2: *Schaltung zur Detektierung von FF aus dem Adreßbus als Port-Adresse.*

wird, wenn der System-Adreßbus logisch mit der eindeutigen Adresse des Ports übereinstimmt. In unserem Beispiel ist die Port-Select-Leitung bei logisch 0 aktiv.

Es ist wichtig, daran zu denken, daß die Port-Select-Leitung auch aktiv werden kann, wenn keine I/O-Operation vorliegt. Das liegt daran, daß der Systemspeicher die gleichen Adreßleitungen benutzt. Es kann z.B. sein, daß der Mikroprozessor Daten von der Speicheradresse XXFFh liest und damit die Port-Select-Leitung aktiv wird, da ja die Adreßleitungen A0 bis A7 dem Port-Select-Code entsprechen. Wenn also die Port-Select-Leitung aktiv wird, ohne daß ein I/O-Befehl vorliegt, so ist das kein Grund zur Beunruhigung.

Generierung der $\overline{IOW}$- und $\overline{IOR}$-Steuersignale

Wenn der Z80 eine I/O-Operation durchführt, so aktiviert er seine Steuerleitung $\overline{IOREQ}$. (Anm.: Diese Leitung gleicht der früher besprochenen $\overline{MREQ}$-Leitung für Speicherzugriffe. Die beiden Zeit-Steuerleitungen, $\overline{RD}$ und $\overline{WR}$, werden logisch kombiniert, um die System-Steuerleitungen $\overline{IOR}$ (I/O-Read) und $\overline{IOW}$ (I/O-Write) zu bilden.

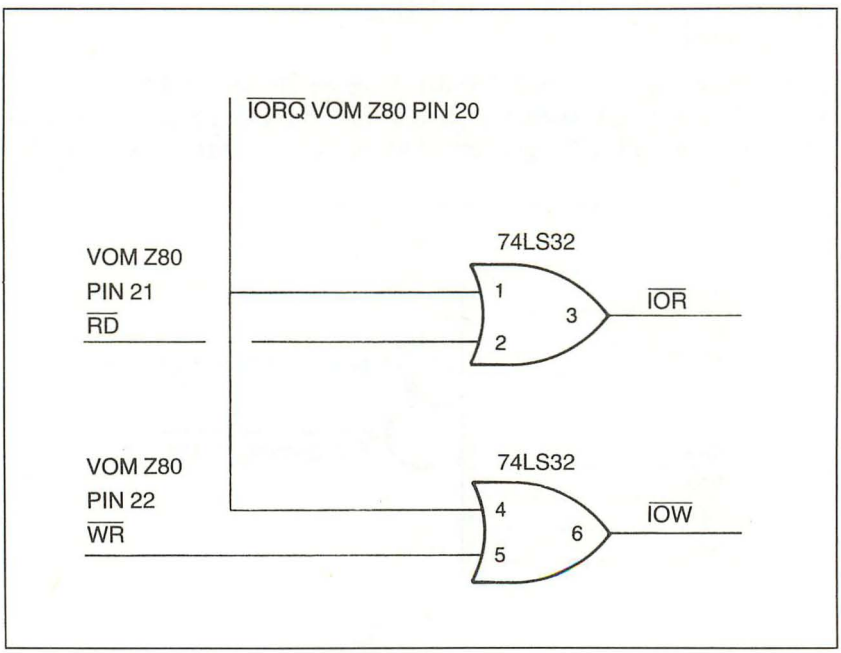


Abb. 3.3: *Schaltung zur Generierung der $\overline{IOR}$- und $\overline{IOW}$-Steuersignale.*

Abb.3.3 zeigt eine Schaltung zur Generierung des $\overline{\text{IOR}}$- und $\overline{\text{IOW}}$-Signals in einem Z80-System. Beachten Sie, daß diese Signale während jeder Eingangs-Lese- bzw. Ausgangs-Schreiboperation aktiv sind, wie in Abb. 3.4a und 3.4b zu sehen ist.

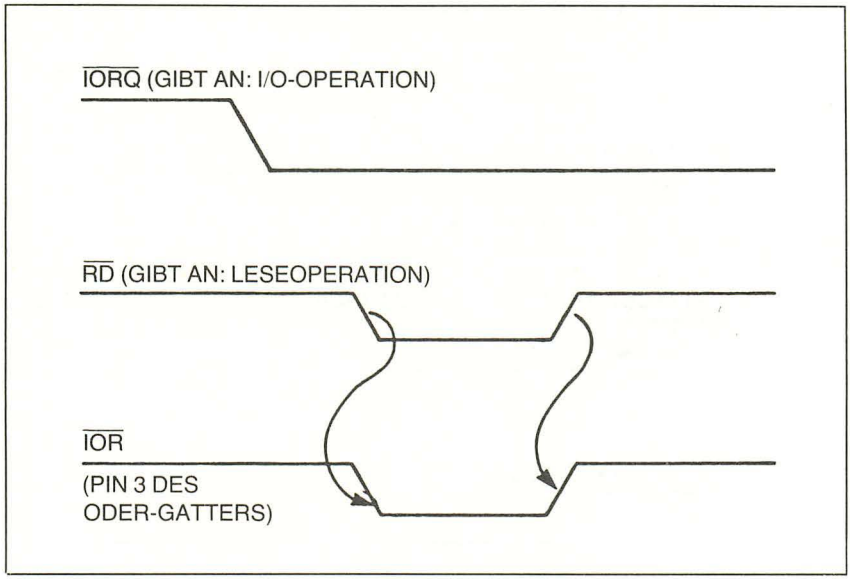


Abb. 3.4a: *Timing der Arbeitsweise der Schaltung aus Abb.3.3 bei einer Leseoperation.*

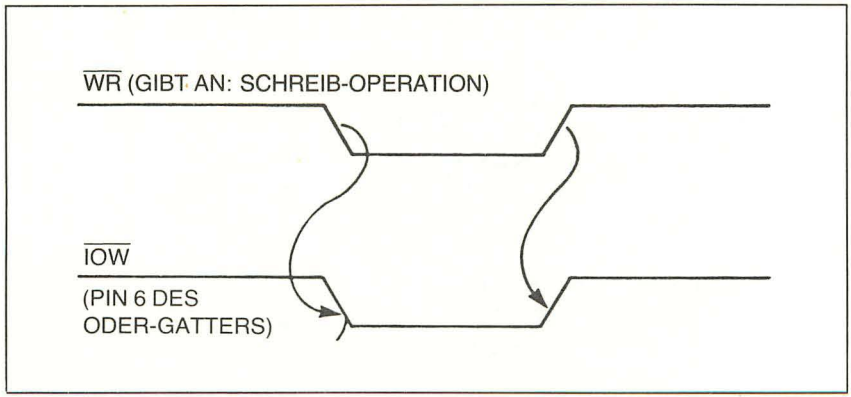


Abb. 3.4b: *Timing der Arbeitsweise der Schaltung aus Abb.3.3 bei einer Schreiboperation.*

Das Port-Schreib-Signal eines Z80-Systems ist definiert als „Schreib-Impuls (write strobe) für ein selektiertes Ausgabe-Port". Dieses Signal wird generiert, wenn sowohl die Port-Select-Leitung als auch die $\overline{IOW}$-Steuerleitung aktiv sind. Das Port-Schreib-Signal stellt ein aktives digitales Signal dar, das die Ausgabe bzw. Speicherung der vom Mikroprozessor kommenden Daten im Ausgabeport auslöst.

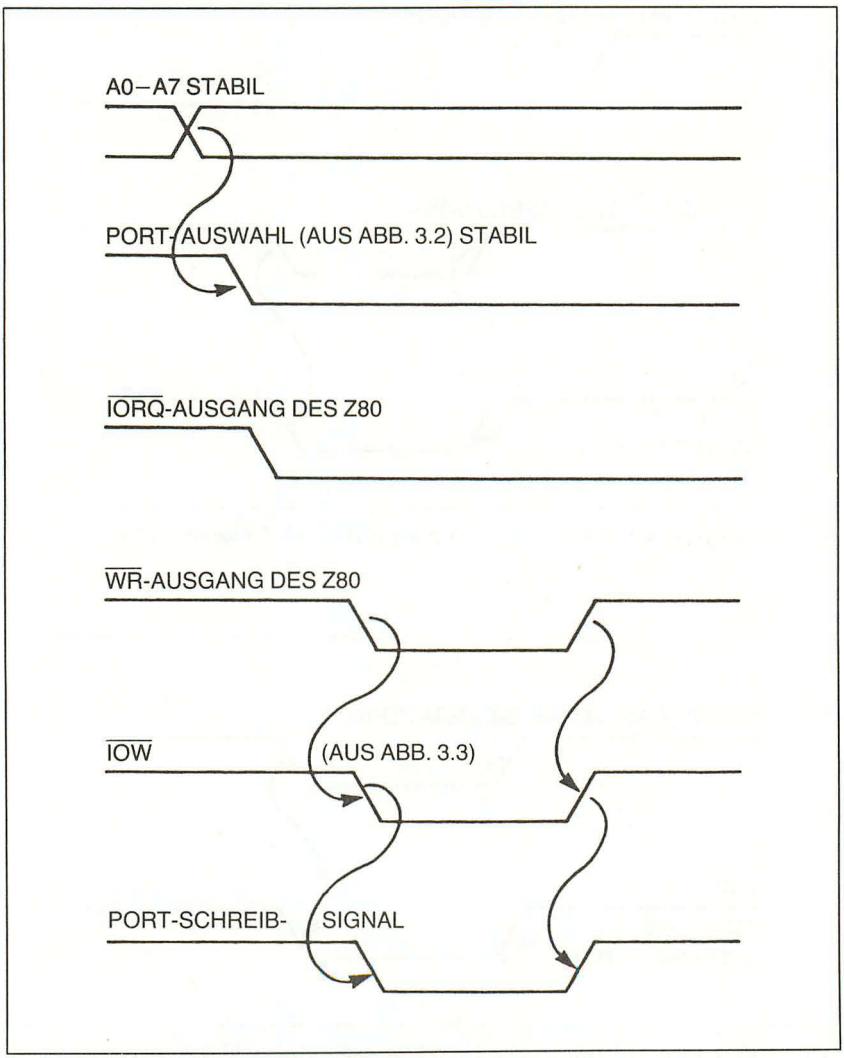


Abb. 3.5: *Allgemeines Timing bei einer Port-Schreib-Operation.*

Das Timing-Diagramm in Abb. 3.5 zeigt das allgemeine Timing beim Schreiben zu einem Ausgangsport. In dieser Darstellung können alle Signale, mit Ausnahme vom $\overline{WR}$, als statische logische Pegel gedacht werden. Die Signalspannung bleibt während der entsprechenden Hardware-Operation konstant. Dieses Timing-Diagramm haben wir bei der Konzeption der Schaltung in Abb. 3.6 verwendet. Sie zeigt eine Möglichkeit zur Erzeugung des Port-Schreib-Signals.

Lassen Sie uns die Schaltung betrachten und sehen, wie sie funktioniert. Das Verständnis dieser Operation wird dazu beitragen, alle Z80-Signale besser zu verstehen. (Beachten Sie, daß es auch andere Möglichkeiten zur Erzeugung des Port-Strobe-Signals gibt.)

Die elektrische Funktion der Schaltung aus Abb.3.6 ist die Erzeugung eines logischen 0-Impulses beim Schreiben in das vom Mikroprozessor angewählte Port. Die System-Adreßleitungen (A0-A7) liegen an den Eingängen eines NAND-Gatters. Das ist die gleiche Schaltung wie in Abb.3.2.

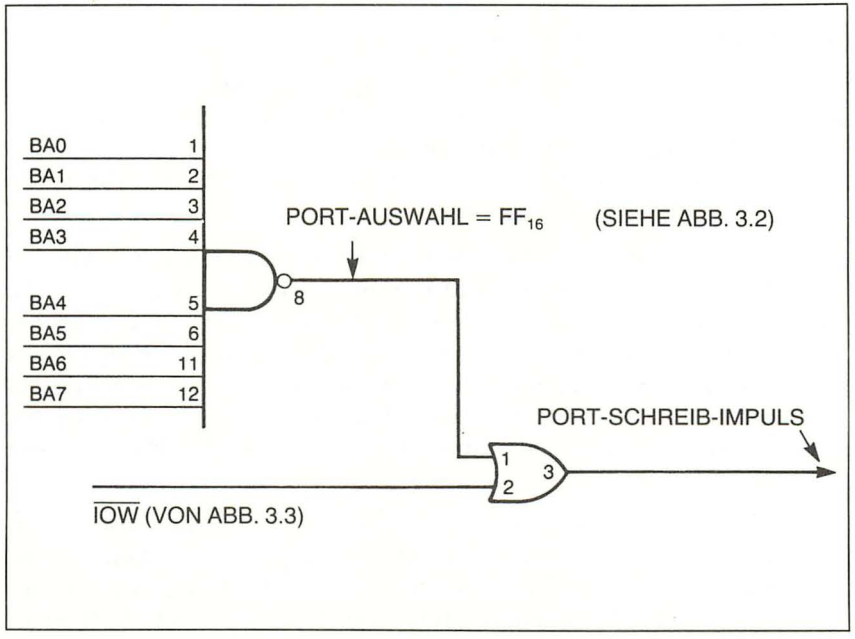


Abb. 3.6: *Diese Schaltung zeigt die zur Erzeugung des Port-Schreib-Signals erforderliche Hardware. Beachten Sie, daß dieses Signal ein eingeschränktes $\overline{IOW}$-Signal ist. Es wurde durch die Port-Select-Leitung eingeschränkt.*

Wenn alle Adreßleitungen (A0-A7) logisch 1 sind (Port-Select-Code: 0FF), wird der NAND-Ausgang logisch 0. Er ist verbunden mit einem Eingang des ODER-Gatters 74LS32. Der andere Eingang dieses Gatters ist mit dem $\overline{IOW}$-Signal von Abb.3.3 verbunden. Wenn das $\overline{IOW}$-Signal 0 ist, wird angezeigt, daß die CPU versucht, ein Byte zu einem System-Ausgabe-Port zu schreiben.

Ist also sowohl das $\overline{IOW}$-Signal als auch das Port-Select-Signal logisch 0, so sollen die Daten in das selektierte Port geschrieben werden. Wir haben also das $\overline{IOW}$-Signal durch Verknüpfung mit der Port-Select-Leitung spezieller gemacht. Das Ergebnis ist ein einzelner logischer 0 Impuls, der bei einer Port-Schreib-Operation zur Adresse FF erzeugt wird. Wir bezeichnen ihn als Port-Schreib-Impuls (Port-Write-Strobe). Er wird benutzt um die Daten zum selektierten Ausgabeport zu schreiben.

Dies ist natürlich nur eine Möglichkeit zur Erzeugung eines Port-Schreib-Impulses. Die Folge von elektrischen Ereignissen und das Konzept der Einschränkung eines $\overline{IOW}$-Impulses sind jedoch bei den meisten Mikroprozessor-Systemen zu finden.

Generierung des Port-Lese-Signals

Lassen Sie uns nun untersuchen, wie der Z80 Daten von einem 8 Bit Eingangsport liest. Das Timing-Diagramm in Abb.3.7 zeigt eine allgemeine Folge elektrischer Ereignisse bei einer Port-Lese-Operation. In diesem Diagramm hat das $\overline{IOR}$-Signal die gleiche Bedeutung wie das $\overline{IOW}$-Signal bei einer Schreiboperation.

Lassen Sie uns nun den elektrischen Effekt des $\overline{RD}$-Signals aus Abb.3.7 betrachten. $\overline{RD}$ ist das zeitgesteuerte Signal, das das $\overline{IOR}$-Signal für die Eingabe-Hardware erzeugt. Dieses Signal startet elektrisch den Datentransfer. Wenn $\overline{RD}$ logisch 0 wird, werden die Daten vom Eingabeport auf den Datenbus gelegt und dann vom Z80 zu einem internen Register geschrieben. Danach wird $\overline{RD}$ wieder logisch 1. Damit ist das Eingabeport elektrisch wieder vom Datenbus getrennt und der Datentransfer beendet.

Abb. 3.8 zeigt eine Möglichkeit zur Generierung eines Port-Lese-Signals mit den logischen Bedingungen von Abb.3.7. Wir können sehen, daß die Schaltung der in Abb. 3.6 stark ähnelt. Tatsächlich liegt der einzige Unterschied in der Benutzung des $\overline{IOR}$-Signals anstelle des $\overline{IOW}$-Signals. Abgesehen davon sind sie identisch. Der Port-Lese-Impuls (port read strobe) wird dann und nur dann aktiv, wenn der Z80 Daten vom angewählten Port FF liest.

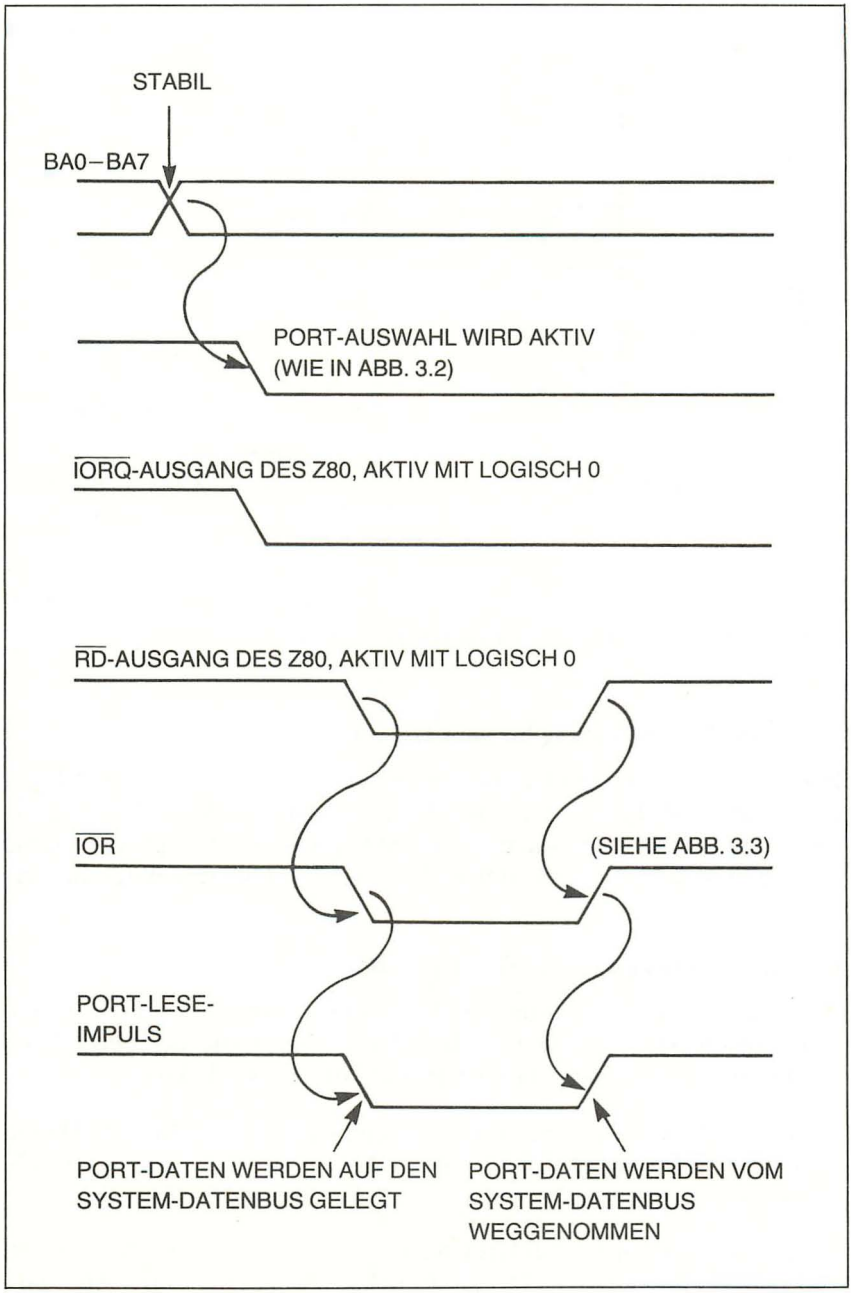


Abb. 3.7: *Timing bei einer Port-Lese-Operation.*

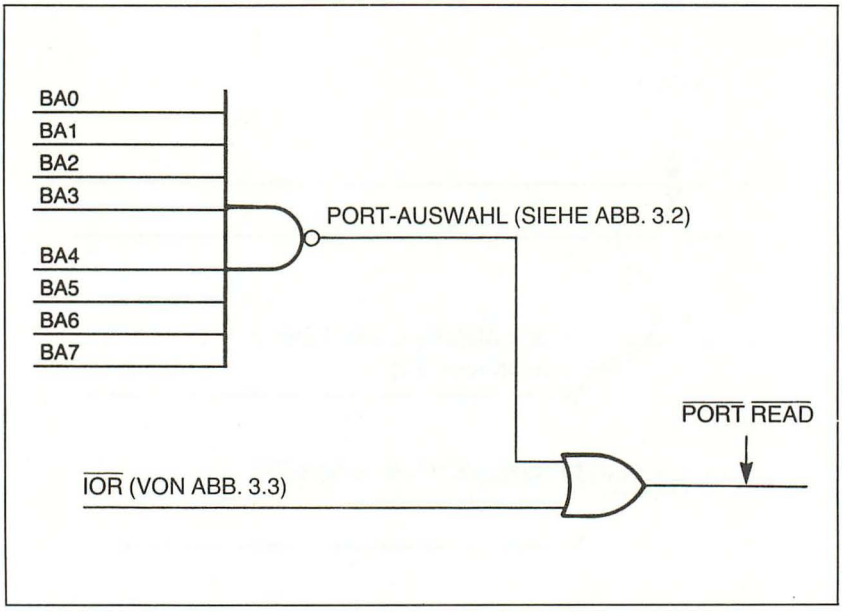


Abb. 3.8: *Dieses Diagramm zeigt die Hardware zur Erzeugung des Port-Lese-Signals.*

Eine komplette Schaltung für ein I/O-Port

In diesem Abschnitt wollen wir die Ereignisse während einer I/O-Lese- und einer I/O-Schreiboperation aufzeigen. Benutzen Sie dabei bitte die Schaltung aus Abb.3.9, die ein allgemeines 8 Bit-Port zeigt. Lassen Sie uns zuerst untersuchen, wie der Z80 eine Ausgabeoperation durchführt.

Ereignisfolge bei einer Ausgabe-Operation

1. Als erstes wird die gewünschte Port-Adresse vom Z80 auf die Adreßleitungen A0 bis A7 gelegt. Die Adresse wird von der Port-Select-Hardware decodiert. Pin 8 des IC 1 in Abb.3.9 wird aktiv 0.

2. Als nächstes werden die auszugebenen Daten vom Z80 auf die Datenleitungen D0 bis D7 gelegt. Sie stehen jetzt am Baustein 74LS374 (IC4) an.

3. Als nächstes wird der $\overline{\text{IORQ}}$-Ausgang des Z80 auf logisch 0 gesetzt. Das zeigt an, daß der Z80 eine I/O-Operation durch führt. Alle statischen Decodierungen der Z80-Signale sind nun abgeschlossen.

4. Als nächstes wird der zeitgesteuerte $\overline{WR}$-Ausgang des Z80 logisch 0. Dies aktiviert die $\overline{IOW}$-Leitung, was wiederum zur Erzeugung des logisch 0 aktiven Port-Schreib-Impulses führt. Als letztes wird der WR-Ausgang des Z80 wieder logisch 1. In diesem Augenblick werden die Daten zum 74LS374 geschrieben. Die Schreiboperation ist damit beendet.

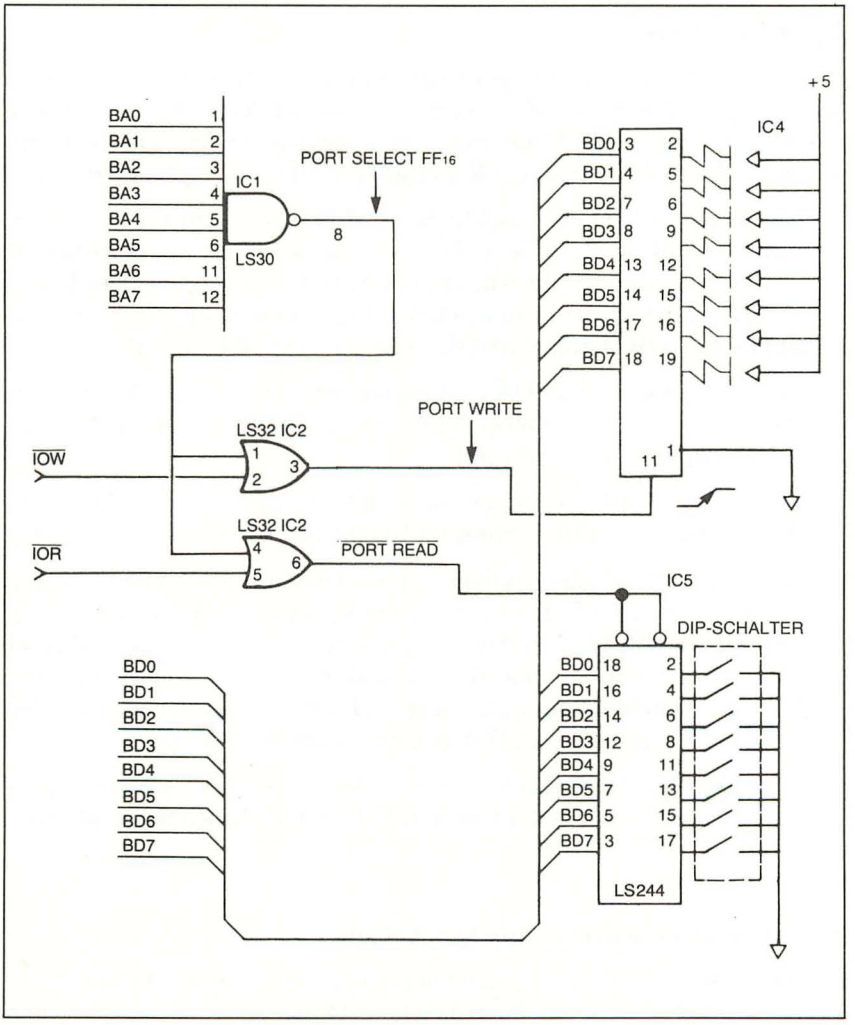


Abb. 3.9: *Vollständiges Schema eines 8-Bit-Eingangs- und Ausgangsports. Die Adresse für Ein- und Ausgang ist FF.*

Bei diesem Ausgabeport werden die Leuchtdioden (LEDs) am Ausgang, abhängig von den Daten, ein- und ausgeschaltet. Eine logische 0 im entsprechenden Datenbit schaltet die Leuchtdiode ein, eine logische 1 schaltet sie aus.

Spätere Kapitel zeigen, wie durch verschiedene Ausgabedaten an spezielle Ports deren Arbeitsweise komplett geändert werden kann.

Lesen eines Eingabeports

Lassen Sie uns nun die Ereignisfolge beim Lesen von Daten aus einem Eingangsport durch den Z80 betrachten (wie in Abb.3.9 gezeigt). Die Hauptereignisse beim I/O-Lesen entsprechen denen beim Schreiben, mit der Ausnahme, daß statt der $\overline{WR}$-Leitung die $\overline{RD}$-Leitung benutzt wird.

1. Als erstes wird die gewünschte Port-Adresse vom Z80 auf die Adreßleitungen A0 bis A7 gelegt. Die Adresse wird von der Port-Select-Hardware decodiert. Pin 8 des IC1 in Abb. 3.9 wird aktiv 0. Die Daten werden jetzt noch nicht auf den Bus gelegt, sondern die Hardware wartet auf das $\overline{IORQ}$-Signal und das zeitgesteuerte $\overline{RD}$-Signal.
2. Als nächstes wird der $\overline{IORQ}$-Ausgang des Z80 auf logisch 0 gesetzt. Das zeigt an, daß der Z80 eine I/O-Operation (im Gegensatz zu einer Speicheroperation) durchführt.
3. Als nächstes wird der zeitgesteuerte $\overline{RD}$-Ausgang des Z80 logisch 0. Dies aktiviert die $\overline{IOR}$-Leitung und damit auch die Port-Lese-Leitung.
4. Wenn die $\overline{IOR}$-Leitung aktiviert ist, ist der Mikroprozessor elektrisch bereit, Daten vom Eingabeport zu empfangen. Das Port-Lese-Signal (aktiv logisch 0) liegt an den Steuereingängen 1 und 19 des 74LS244 (IC5). Damit werden seine Ausgänge aktiviert und somit der logische Wert der Schalterstellungen auf den Datenbus übertragen. Der Z80 speichert die anliegenden Daten in einem internen Register.
5. Als letztes wird das $\overline{RD}$-Signal des Z80 wieder 1. Das Eingabeport wird elektrisch vom Datenbus getrennt, und die Eingabeoperation ist beendet.

Zusammenfassung der elektrischen Abläufe

Lassen Sie uns noch schnell einen kurzen Rückblick auf die Abläufe beim Lesen und Schreiben von Ports machen. Diese Informationen werden später beim detaillierten Besprechen einzelner I/O-Bausteine nützlich sein.

Ausgabeablauf

1. Die gewünschte Port-Adresse wird auf A0-A7 gelegt.
2. Die Ausgabedaten werden auf den Datenbus D0-D7 gelegt.
3. $\overline{\text{IORQ}}$ wird logisch 0 gesetzt.
4. $\overline{\text{WR}}$ wird logisch 0 gesetzt.
5. $\overline{\text{WR}}$ wird logisch 1 gesetzt.

Eingabeablauf

1. Die gewünschte Port-Adresse wird auf A0-A7 gelegt.
2. $\overline{\text{IORQ}}$ wird logisch 0 gesetzt.
3. $\overline{\text{RD}}$ wird logisch 0 gesetzt.
4. $\overline{\text{RD}}$ wird logisch 1 gesetzt.

Zusammenfassung

In diesem Kapitel haben wir die Details der Ein- und Ausgabe des Z80 kennengelernt. Wir haben mehrere Hardware-Dekodierungstechniken erforscht und die Ereignisse bei Ein- und Ausgabeoperationen genau untersucht.

Wenn Sie vorhaben, Z80-Systeme zu entwerfen oder zu reparieren, sollten sie mit der Ein-/Ausgabetechnik bekannt sein. Die Informationen in diesem Kapitel sind dafür wichtig. Außerdem sind sie nützlich als Vorbereitung für die Besprechung von LSI-I/O-Bauteilen in späteren Kapiteln.

Kapitel 4

Benutzung von dynamischen RAMs

Einführung

In diesem Kapitel lernen wir die Benutzung von dynamischen RAMs in Z80-Systemen kennen. Zuerst untersuchen wir ein typisches dynamisches RAM. Dann zeigen wir, wie der Adreß-, Daten- und Steuerbus des Z80 elektrisch mit dynamischen RAM-Bauteilen kommuniziert. Zuletzt untersuchen wir noch ein vollständiges RAM-System für den Z80.

Das dynamische RAM, das wir in diesem Kapitel betrachten, ist der 4116, ein 16K * 1 Baustein. Wir haben diesen Baustein ausgewählt, weil er in seiner Arbeitsweise den meisten heute eingesetzten dynamischen RAMs entspricht. Wenn Sie verstanden haben, wie dieses Bauteil funktioniert, werden Sie auch die Arbeitsweise anderer dynamischer RAM-Bausteine (z.B. 4164, 64K * 1) verstehen.

Übersicht über den 4116

Der 4116 hat eine 16.384 (16K) * 1 Organisation. Er benutzt getrennte Ein- und Ausgangsleitungen. Ein solcher Speicher benötigt normalerweise 14 Adreßleitungen (um alle 16.384 Adressen ansprechen zu können) sowie eine Dateneingangs- und eine Datenausgangsleitung.

Um die Daten intern abzuspeichern, ist eine Schreib-Freigabe-Leitung notwendig. Außerdem braucht der 4116 Stromversorgungen +12V, +5V, −5V und Masse. Wenn wir alle Anschlüsse auflisten, ergibt sich folgendes Bild:

14 Adreßleitungen
1 Dateneingangsleitung
1 Datenausgangsleitung
4 Stromversorgungsleitungen
1 Schreib-Freigabe-Leitung

also im ganzen 21 physikalische Leitungen. Abb. 4.1 macht dies deutlich.

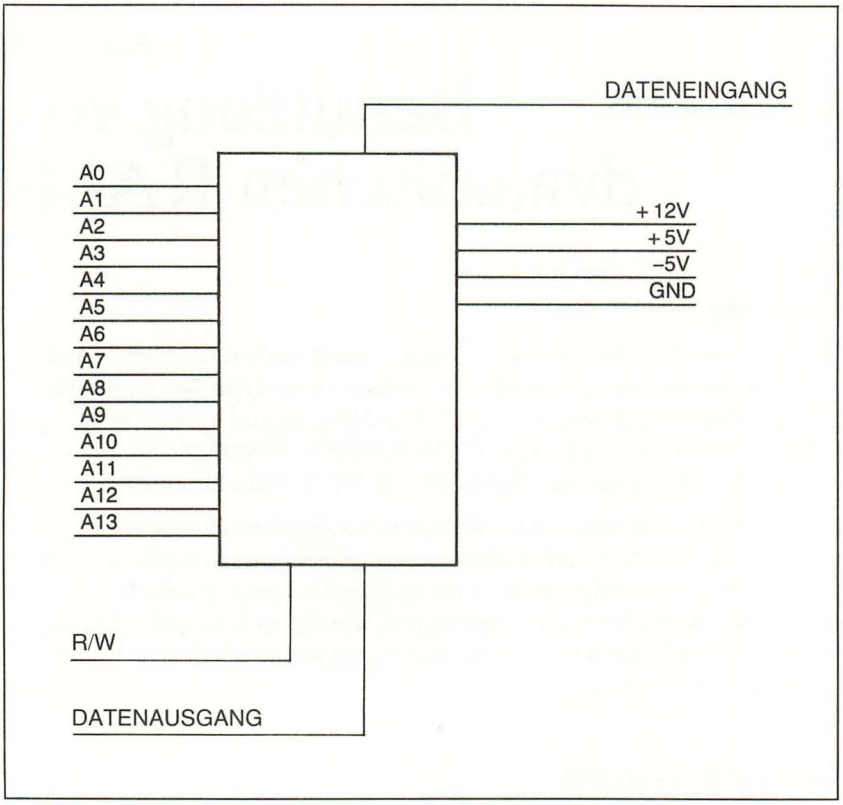


Abb. 4.1: *Ein Blockschaltbild, das die physikalischen Leitungen zeigt, die ein 16K * 1 RAM benötigt. Der 4116 muß auf irgendeine Weise diese physikalischen Bedingungen erfüllen.*

Der 4116 ist jedoch in einem 16-Pin-Gehäuse (siehe Abb. 4.2) untergebracht. Auch wenn das 16-Pin-Gehäuse auf dem ersten Blick unzureichend erscheint: Es liegt kein Fehler vor. Die Hersteller dieses Bauteils machen von der Zeit-Multiplex-Technik Gebrauch, um physikalische Anschlüsse einzusparen. Zeit-Multiplex beim 4116 bedeutet, daß die erforderlichen 14 Adreßleitungen in 2 Gruppen zu je 7 eingelesen werden. Die eine Gruppe wird nach der anderen eingelesen. Es gibt also 7 Adreß-Pins beim 4116. Sie werden, wie in Abb.4.3 zu sehen ist, mit A0-A6 bezeichnet. Wenn auf ihnen die unteren 7 Bits des Adreßbusses übertragen werden, so handelt es sich um die sogenannte Zeilenadresse (row address). Die höheren 7 Bits A7-A13 bilden die sogenannte Spaltenadresse (column address) (siehe Abb. 4.4).

16,384 X 1-BIT DYNAMIC RAM

MK4116(J/N/E)-2/3

FEATURES

- Recognized industry standard 16-pin configuration from MOSTEK
- 150ns access time, 320ns cycle (MK 4116-2) 200ns access time, 375ns cycle (MK 4116-3)
- ± 10% tolerance on all power supplies (+12V, ±5V)
- Low power: 462mW active, 20mW standby (max)
- Output data controlled by $\overline{CAS}$ and unlatched at end of cycle to allow two dimensional chip selection and extended page boundary
- Common I/O capability using "early write" operation
- Read-Modify-Write, $\overline{RAS}$-only refresh, and Page-mode capability
- All inputs TTL compatible,low capacitance, and protected against static charge
- 128 refresh cycles
- ECL compatible on VBB power supply (-5.7V)
- MKB version screened to MIL-STD-883
- JAN version available to MIL-M-38510/240

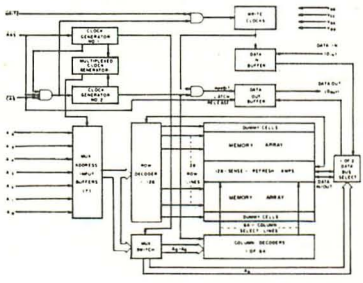
DESCRIPTION

The MK 4116 is a new generation MOS dynamic random access memory circuit organized as 16,384 words by 1 bit. As a state-of-the-art MOS memory device, the MK 4116 (16K RAM) incorporates advanced circuit techniques designed to provide wide operating margins, both internally and to the system user, while achieving performance levels in speed and power previously seen only in MOSTEK's high performance MK 4027 (4K RAM).

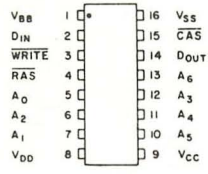
The technology used to fabricate the MK 4116 is MOSTEK's double-poly, N-channel silicon gate, POLY II Ⓣ process. This process, coupled with the use of a single transistor dynamic storage cell, provides the maximum possible circuit density and reliability, while maintaining high performance capability. The use of dynamic circuitry throughout, including sense amplifiers, assures that power dissipation is minimized without any sacrifice in speed or operating margin. These factors combine to make the MK 4116 a truly superior RAM product.

Multiplexed address inputs (a feature pioneered by MOSTEK for its 4K RAMS) permits the MK 4116 to be packaged in a standard 16-pin DIP. This recognized industry standard package configuration, while compatible with widely available automated testing and insertion equipment, provides highest possible system bit densities and simplifies system upgrade from 4K to 16K RAMs for new generator applications. Non-critical clock timing requirement allow use of the multiplexing technique while mair taining high performance.

FUNCTIONAL DIAGRAM



PIN CONNECTIONS



PIN NAMES

A_0-A_6	ADDRESS INPUTS	$\overline{WRITE}$	READ/WRITE INPUT
$\overline{CAS}$	COLUMN ADDRESS STROBE	V_{BB}	POWER (-5V)
D_{IN}	DATA IN	V_{CC}	POWER (+5V)
D_{OUT}	DATA OUT	V_{DD}	POWER (+12V)
$\overline{RAS}$	ROW ADDRESS STROBE	V_{SS}	GROUND

Abb. 4.2: *Datenblattauszug für den 4116, ein 16K * 1 dynamisches RAM.*

Die Ausdrücke Zeilen- und Spaltenadresse entspringen dem internen Aufbau des Bausteins als ein Speicherarray mit 128 * 128 Speicherzellen. Diese Speicherzellen sind matrizenartig angeordnet. Jede Zelle hat eine eindeutige Zeilen- und Spaltenadresse.

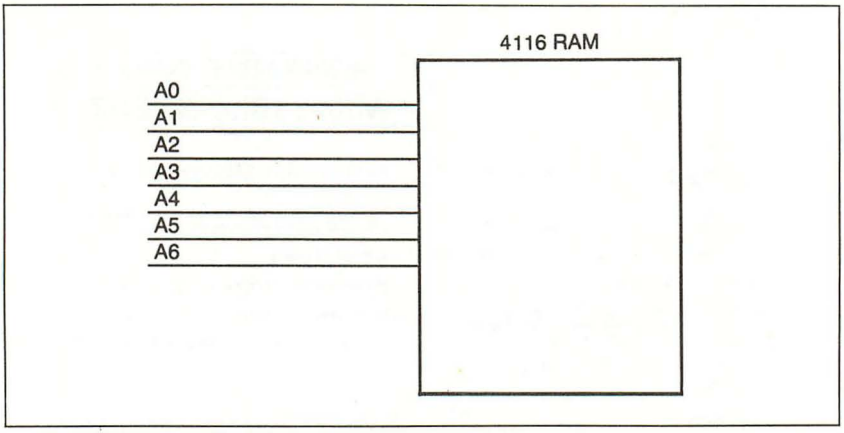


Abb. 4.3: *Blockschaltbild mit den tatsächlichen am 4116 verfügbaren Adreßleitungen. Die Gesamtadresse von 14 Bit wird in zwei separaten Gruppen zu je 7 Bit auf die 7 physikalischen Leitungen gelegt.*

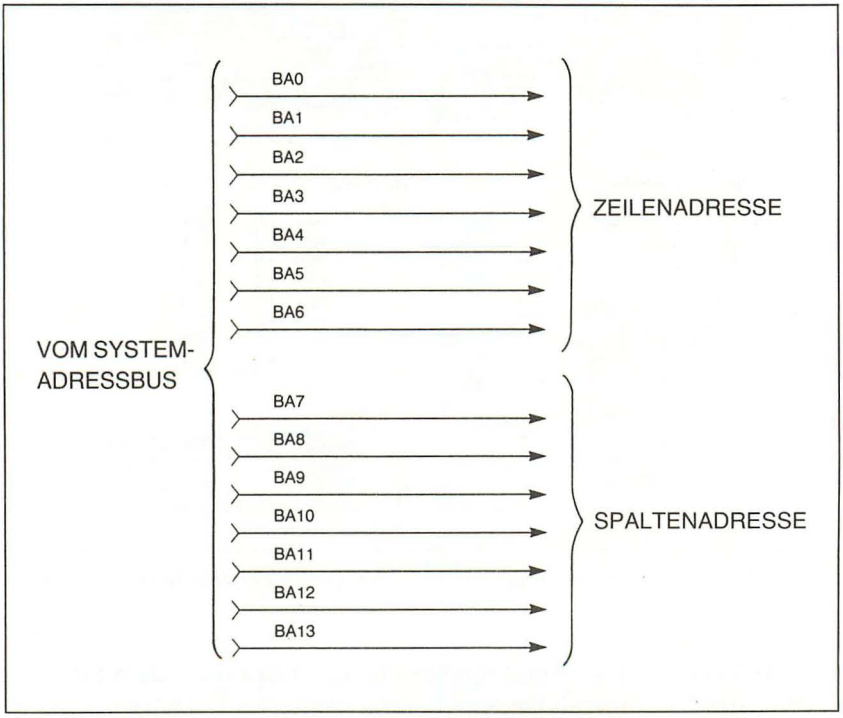


Abb. 4.4: *Die System-Adreßleitungen sind aufgeteilt in eine Zeilen- und eine Spaltengruppe.*

Zur Steuerung des Zeit-Muliplex wird ein Signal benötigt, das den Wechsel der Adreßhälften steuert. Der 4116 muß darüber unterrichtet werden, wann die Information auf seinen Adreßleitungen A0-A6 die Zeilenadresse ist und wann die Spaltenadresse. Hierfür sind zwei zusätzliche Anschlüsse am Bauteil vorgesehen. Sie werden als $\overline{RAS}$ (row address strobe) und $\overline{CAS}$ (column address strobe) bezeichnet. Wenn $\overline{RAS}$ aktiv (logisch 0) wird, so werden die unteren 7 Bits der Adresse als Zeilenadresse in den Baustein eingelesen. Logisch 0 am $\overline{CAS}$-Eingang führt zum Einlesen der oberen 7 Bits als Spaltenadresse. Abb. 4.5 zeigt das Timing beim Einlesen der Adresse in den 4116.

Die Leitungen A0-A6, $\overline{RAS}$ und $\overline{CAS}$ sind Eingänge des 4116. Das heißt, das Bauteil ist, was die Multiplex-Steuerung angeht, selber passiv. Diese Arbeit muß von externer Hardware übernommen werden. Mit anderen Worten: Externe Hardware muß die Zeilenadresse anlegen, das $\overline{RAS}$-Signal aktivieren, die Spaltenadresse anlegen und das $\overline{CAS}$-Signal aktivieren. Das Ganze muß außerdem zeitlich aufeinander abgestimmt sein.

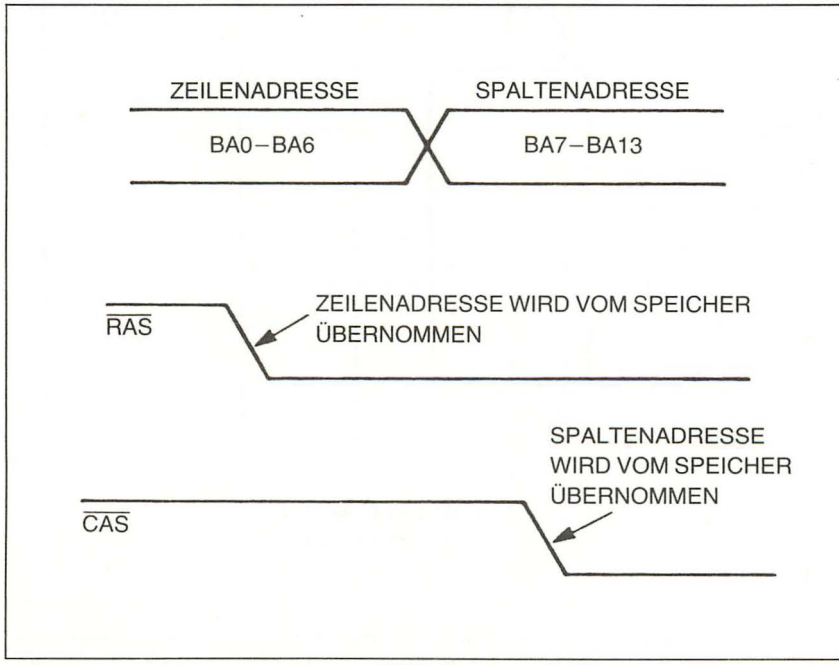


Abb. 4.5: *Zusammenhang zwischen $\overline{RAS}$-Signal und $\overline{CAS}$-Signal bei der Adressierung eines 4116 RAM.*

Adressen-Multiplex

Eine Schaltung in der Art, wie sie in Abb. 4.6 gezeigt wird, kann das Adressen-Multiplex für den 4116 durchführen. Wie arbeitet diese Schal-

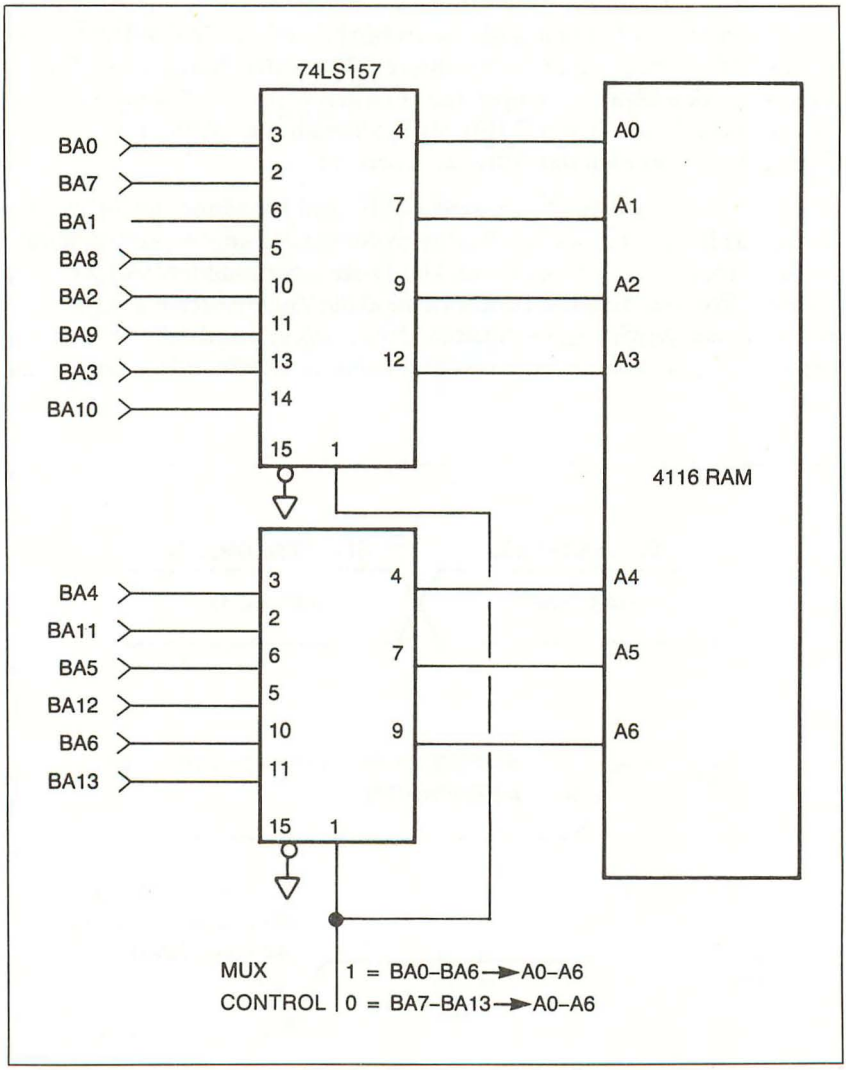


Abb. 4.6: Schaltung zum Anschluß der 14 Adreßleitungen an ein 4116-Bauteil. Die Adreßleitungen vom Mikroprozessor gehen an die Eingänge von 74LS157 Multiplexern. Die Multiplexer-Ausgänge sind mit den Adreßeingängen des RAM verbunden.

tung? Die 14 Adreßleitungen A0-A13 gehen zu den Eingängen von logischen Bauteilen, die als Multiplexer bezeichnet werden. Wenn die Steuerleitungen der Multiplexer logisch 1 sind, werden die Adreßbus-Leitungen A0-A6 zum 4116 geleitet. Wenn sie logisch 0 sind, werden die übrigen 7 Leitungen A7-A13 zu den Adreßeingängen des RAM geleitet.

Wir wissen jetzt, daß drei verschiedene Vorgänge von der Hardware gesteuert werden müssen, um die benötigte 14-Bit-Adresse zum 4116 zu übertragen:

1. Generierung des $\overline{\text{RAS}}$-Signals zur richtigen Zeit
2. Umschaltung der Multiplexer-Steuerleitung auf logisch 0
3. Generierung des $\overline{\text{CAS}}$-Signals zur richtigen Zeit

Abb. 4.7 zeigt diese drei Ereignisse in einem verallgemeinerten Zeitdiagramm (später werden wir sehen, wie der Z80 dies einhält). Jetzt wollen wir lernen, wie der 4116 die Adresse einliest.

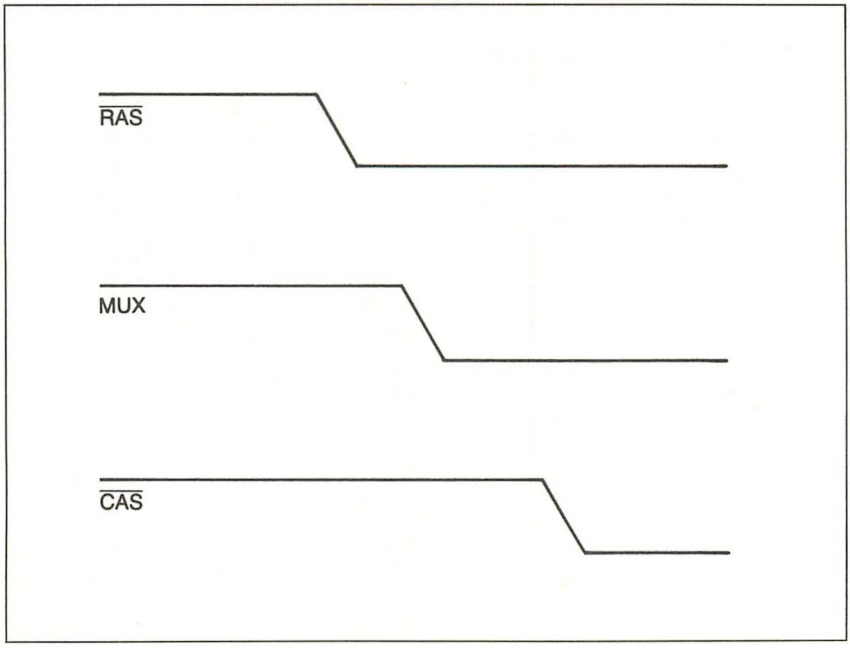


Abb. 4.7: *Beziehung zwischen $\overline{RAS}$-, MUX- und $\overline{CAS}$-Signal als Steuersignale eines dynamischen RAM. Blockschaltbild für ein 16 K * 8-Bit dynamisches RAM-System.*

Abb. 4.8 zeigt das Blockschaltbild eines 16K * 8-Bit dynamischen RAM-Systems für den Z80. Lassen Sie es uns nun näher untersuchen.

Wenn Sie diese Zeichnung sehen, sollten Sie bedenken, daß es viele Möglichkeiten zum Anschluß dynamischer RAMs an einen Mikroprozessor gibt. Wir haben die hier vorgestellte Technik ausgewählt, weil sie besonders klar und einfach ist und nicht viel Hardware-Erfahrung erfordert. Das Interface für ein dynamisches RAM kann auch sehr kompliziert sein, besonders im industriellen Bereich. Jedoch auch die kompliziertesten Schaltungen benutzen die hier vorgestellte Technik als Grundlage. Lassen Sie uns Abb.4.8 untersuchen.

Die Speicheradresse wird in der in Abb. 4.6 vorgestellten Art eingebracht (dieser Block ist in Abb. 4.8 weggelassen worden). Die $\overline{RAS}$-Leitung liegt an allen Speicherbausteinen parallel an. Die $\overline{CAS}$-Leitung kann, aber muß nicht, bei einer Speicheroperation angesprochen werden (dies wird später noch erklärt).

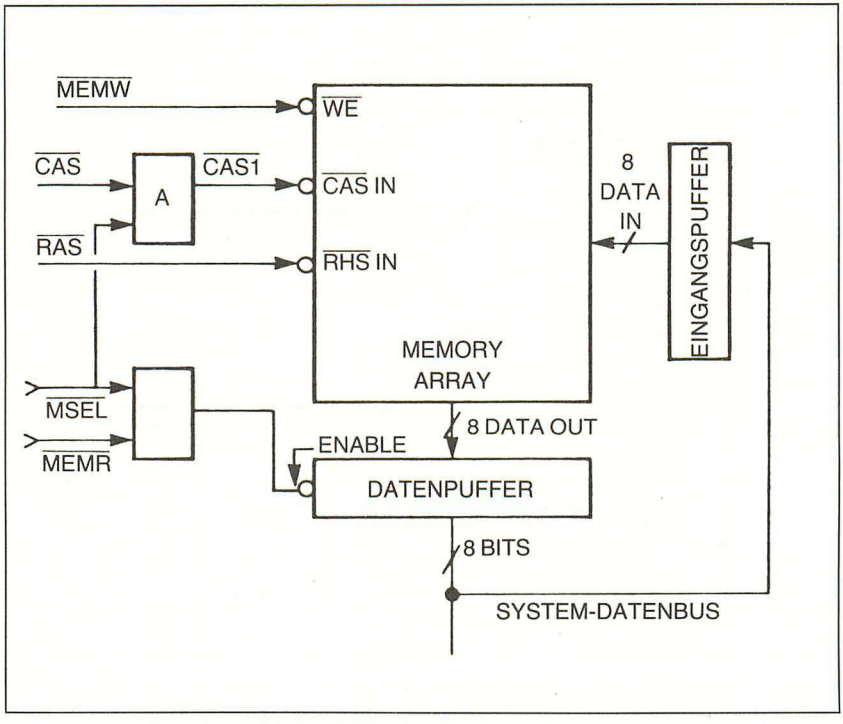


Abb. 4.8: *Das Blockschaltbild zeigt die benötigte Hardware-Peripherie für ein dynamisches RAM-Speichersystem.*

Die Logik zur Generierung des $\overline{CAS}$-Signals steckt in Block A. Sie decodiert den Zustand der Speicher-Auswahl-Leitungen und aktiviert zur richtigen Zeit die $\overline{CAS}$-Eingangsleitung zum Speicher. Die Speicherauswahl wird durch Decodierung der Systemadresse gewonnen.

Der Adreßraum für dieses dynamische RAM ist 16 KBytes. Die Auswahlleitung für diesen Bereich wird aus den Adreßleitungen A14 und A15 decodiert. Sind beide logisch 1, so wird das RAM von der CPU angesprochen.

Im Blockschaltbild von Abb. 4.8 bekommt die RAM-Schreibfreigabe-Leitung immer dann ihren aktiven logischen 0-Pegel, wenn $\overline{MREQ}$ und $\overline{WR}$ logisch 0 sind. Sie meinen möglicherweise, daß das zu Problemen führen könnte, da diese Signale ja bei jeder Schreiboperation zu jedem Systemspeicher auftreten. Wir werden jedoch später sehen, wie das System ungewollte Schreiboperationen zum RAM verhindert.

Generierung des $\overline{RAS}$-, $\overline{CAS}$- und MUX-Signals

Wir werden jetzt zeigen, wie das $\overline{RAS}$-, $\overline{CAS}$- und MUX-Steuersignal erzeugt wird. Auch hier gibt es viele mögliche Lösungstechniken. Die hier vorgestellte Technik erlaubt es auch dem Anfänger, zu verstehen, was genau passiert und in welcher Reihenfolge. Außerdem funktioniert sie.

Aus Abb. 4.5 können wir die geforderte Reihenfolge für das $\overline{RAS}$-, $\overline{CAS}$- und MUX-Signal ersehen. Abb. 4.9 zeigt, wie diese Signale in der richtigen Folge erzeugt werden können. Das $\overline{RAS}$-Signal ist hier ein gepuffertes $\overline{MREQ}$-Signal. Erinnern wir uns, daß das $\overline{RAS}$-Signal den unteren Adreßteil in einen internen Zwischenspeicher des 4116 abspeichert. Die Adreßleitungen des Mikroprozessors sind zu diesem Zeitpunkt bereits stabil.

Das MUX-Signal wird zum Umschalten der Adreßleitungen von A0-A6 auf A7-A13 benötigt. Es wird von der $\overline{RAS}$-Leitung erzeugt. Es läuft dabei jedoch durch eine Verzögerungs-Leitung. Jede Standard-Verzögerungs-Leitung kann benutzt werden. Manchmal wird die Verzögerung auch durch Reihenschaltung mehrerer logischer Bausteine erzeugt, wobei sich die Schaltverzögerungen addieren. Als Verzögerungszeit zwischen $\overline{RAS}$ und MUX werden etwa 50 Nanosekunden genommen.

Nachdem die MUX-Leitung logisch 0 geworden ist, kann auch die $\overline{CAS}$-Leitung aktiv werden. Das $\overline{CAS}$-Signal wird ebenso über eine Verzögerungsleitung gewonnen.

Diese Technik der Signal-Erzeugung ist leicht durch Hardware zu reali-

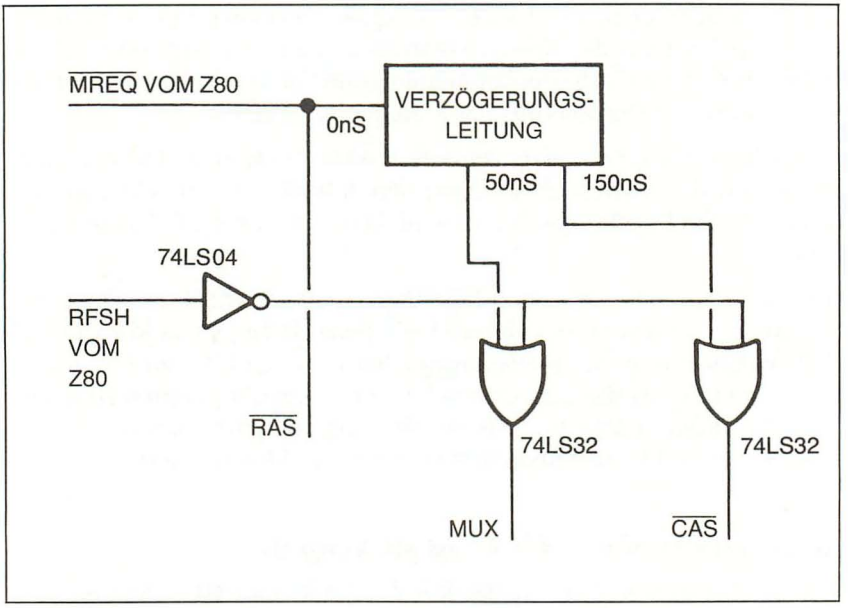


Abb. 4.9: *So werden das RAS-, CAS- und MUX-Signal generiert.*

sieren. Es gibt jedoch auch ein paar Nachteile. Der Hauptnachteil liegt darin, daß der Speicher eine sehr kurze Zugriffszeit haben muß, wenn er bei der maximalen Prozessorgeschwindigkeit arbeiten soll, da die Speicherzugriffszeit ab der fallenden Flanke des CAS-Signals gemessen wird. Eine Lösungsmöglichkeit ist die Herabsetzung der Prozessorgeschwindigkeit.

Wenn die Taktfrequenz auch langsam genug sein muß, damit die Zugriffszeiten eingehalten werden können, so darf sie nicht zu langsam sein, damit die Auffrisch- oder Refresh-Bedingungen, die wir später noch beschreiben werden, erhalten bleiben.

Dateneingangs-Pufferung des dynamischen RAM

Das dynamische RAM 4116 hat getrennte Dateneingänge und Ausgänge, das heißt, es werden unterschiedliche physikalische Anschlüsse zur Eingabe und zur Ausgabe benutzt. Wir wollen jetzt die Dateneingabe vom Mikroprozessor zum RAM betrachten.

Abb. 4.10 zeigt eine Puffer-Schaltung, die den System-Datenbus mit den 4116 Eingangs-Pins verbindet. Wie in der Schaltung zu sehen ist, sind die

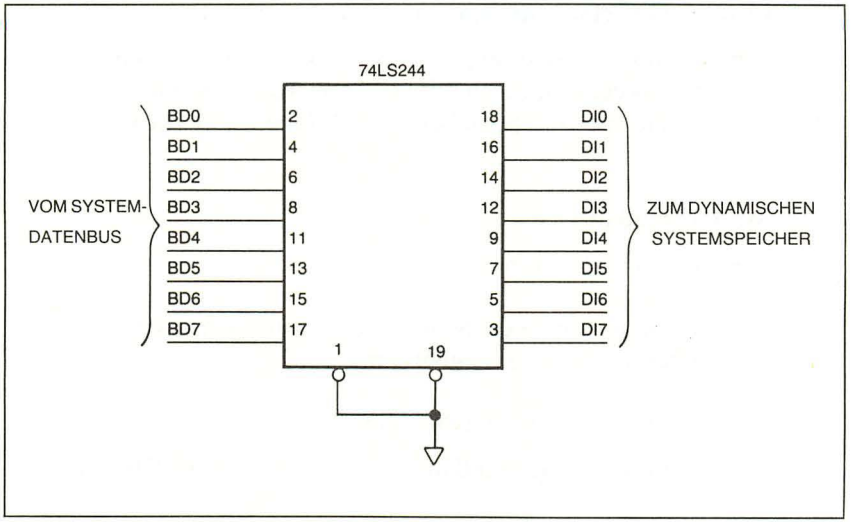


Abb. 4.10: *Pufferung der RAM-Eingangs-Datenleitungen. In einigen Systemen wird diese Pufferung weggelassen, und die RAM-Eingänge werden direkt mit dem Datenbus verbunden.*

Puffer dauernd durchgeschaltet. Die Daten vom Datenbus legen elektrisch also dauernd an den RAM-Eingangs-Pins an. Das stört nicht, da die Daten ja nicht ohne ein spezielles Schreibfreigabe-Signal in das RAM eingeschrieben werden.

Vielleicht werden sie fragen, warum Puffer eingesetzt werden. Da der Datenbus doch dauernd mit dem RAM verbunden ist, könnten sie ja auch wegfallen. Tatsächlich gibt es auch einige Systeme, die auf eine Pufferung verzichten, jedoch gibt es Gründe, die für die Puffer sprechen:

- um die RAM-Eingänge von dem angeschlossenen Datenbus zu isolieren. Das ist hilfreich bei der Fehlersuche in defekten Systemen. Wenn ein einzelnes RAM-Bauteil ausfällt, könnte die ganze Datenbusleitung gestört werden. Dies Problem vergrößert sich, wenn viele RAMs auf dem Bus liegen.

- um das Rauschen zu vermindern. Der System-Datenbus ist normalerweise ziemlich „schmutzig", das bedeuted, es befinden sich ungewollte Störsignale auf den Leitungen. Datenbus-Puffer schützen das RAM vor solchen Signalen.

Die meisten Hersteller fordern den Einsatz von Daten-Eingangs-Puffern.

Schreibvorgang bei einem dynamischen RAM

In diesem Abschnitt wird gezeigt, wie der Mikroprozessor Daten in den 4116 schreibt. Aus dem Blockschaltbild in Abb. 4.8 ist zu ersehen, daß die folgenden Signale für eine Speicher-Schreib-Operation aktiv sein müssen:

- $\overline{MREQ}$ muß logisch 0 sein
- $\overline{WR}$ muß logisch 0 sein

Diese beiden Signale bilden die Schreibfreigabe für den Systemspeicher. Zur Auswahl des Speichers gilt außerdem:

- A14 und A15 müssen logisch 1 sein
- das dynamische RAM muß im Bereich C000 bis FFFF liegen

Unter diesen Bedingungen wird das $\overline{CAS}$-Signal an das System-RAM gelegt und es erfolgt eine Schreiboperation.

Lassen Sie uns nun das Timing-Diagramm für zwei unterschiedliche Fälle betrachten: Das RAM wird angesprochen bzw. das RAM wird nicht angesprochen, obwohl der Prozessor Daten ausgibt (zu einem anderen Speicherbereich oder I/O).

Abb. 4.11 zeigt das Timing bei einer Speicher-Schreib-Operation für ein angesprochenes RAM. Als erstes wird die Speicher-Anwahl-Leitung (Memory Select) aktiv (0). Das passiert dadurch, daß die Adreßleitungen A14 und A15 vom Z80 auf logischen 1-Pegel gebracht werden. Als nächstes Ereignis wird das $\overline{RAS}$-Signal aktiv (0). $\overline{RAS}$ wird vom $\overline{MREQ}$-Ausgang des Z80 generiert. Nach einer Verzögerung von 50 Nanosekunden (ab der fallenden Flanke von $\overline{RAS}$) wird das MUX-Signal logisch 0. Damit werden die Adreßeingänge des 4116 von der Zeilenadresse auf die Spaltenadresse umgeschaltet. Nach weiteren 100 Nanosekunden (erzeugt durch Verzögerung des MUX-Signals) wird das $\overline{CAS}$-Signal aktiv (0). Zuletzt wird aus dem $\overline{WR}$-Ausgang des Z80 das Schreib-Freigabe-Signal für den $\overline{WE}$-Eingang des Speichers erzeugt.

Das Timing-Diagramm in Abb. 4.11 zeigt den Ablauf bei jeder Schreiboperation des Mikroprozessors zum dynamischen Systemspeicher. Abb. 4.12 zeigt das Timing eines Schreibvorgangs, bei dem nicht das dynamische RAM angesprochen wird, da die Daten anderweitig bestimmt sind. Ein Blick auf das Timing-Diagramm in Abb.4.12 zeigt, daß die Speicher-Freigabe-Leitung (Memory Select) nicht aktiv ist. Der Grund dafür ist,

daß A14 und A15 nicht beide logisch 1 sind. Die $\overline{RAS}$-Leitung ist aktiv, da auch die $\overline{MREQ}$-Steuerleitung aktiv ist. MUX wird dadurch ebenfalls erzeugt. Die System-$\overline{CAS}$-Leitung wird aktiv, jedoch nicht die $\overline{CAS}1$-Leitung zum Speicher, da das MSEL-Signal fehlt (siehe Abb. 4.8).

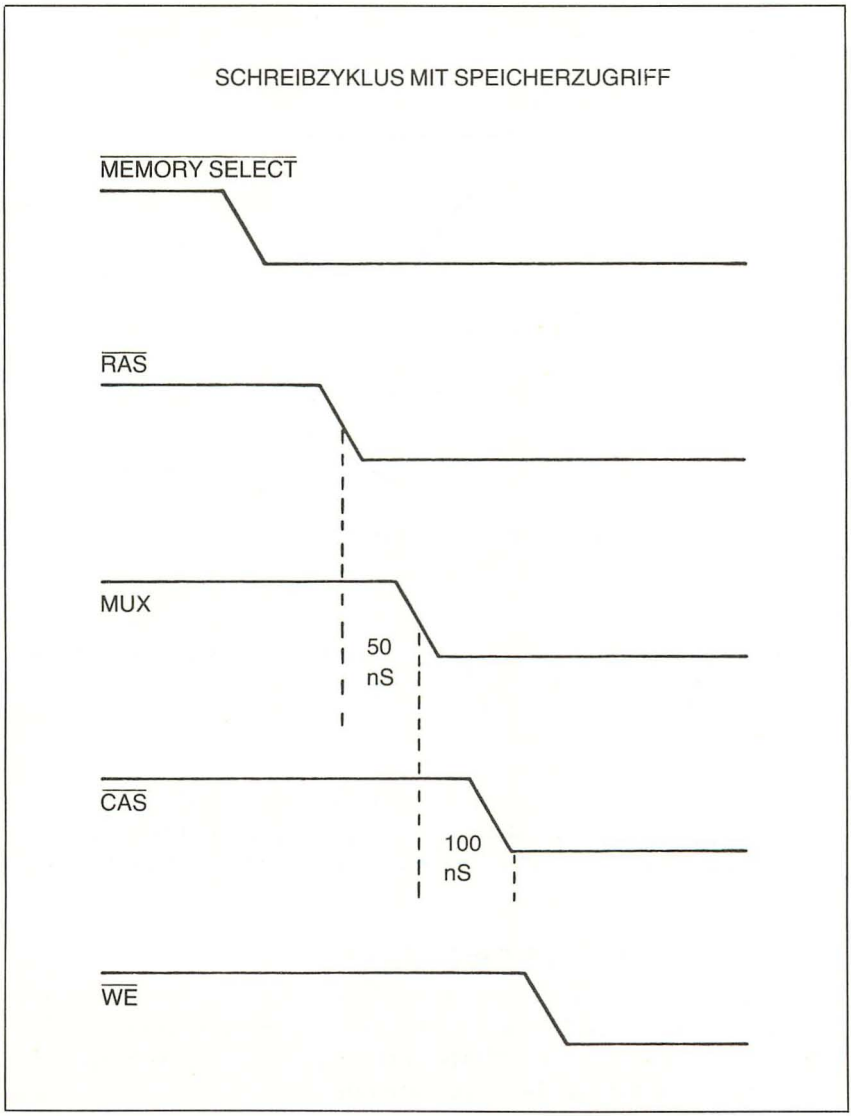


Abb. 4.11: *Timing für ein angesprochenes dynamisches RAM bei einem Schreibzyklus.*

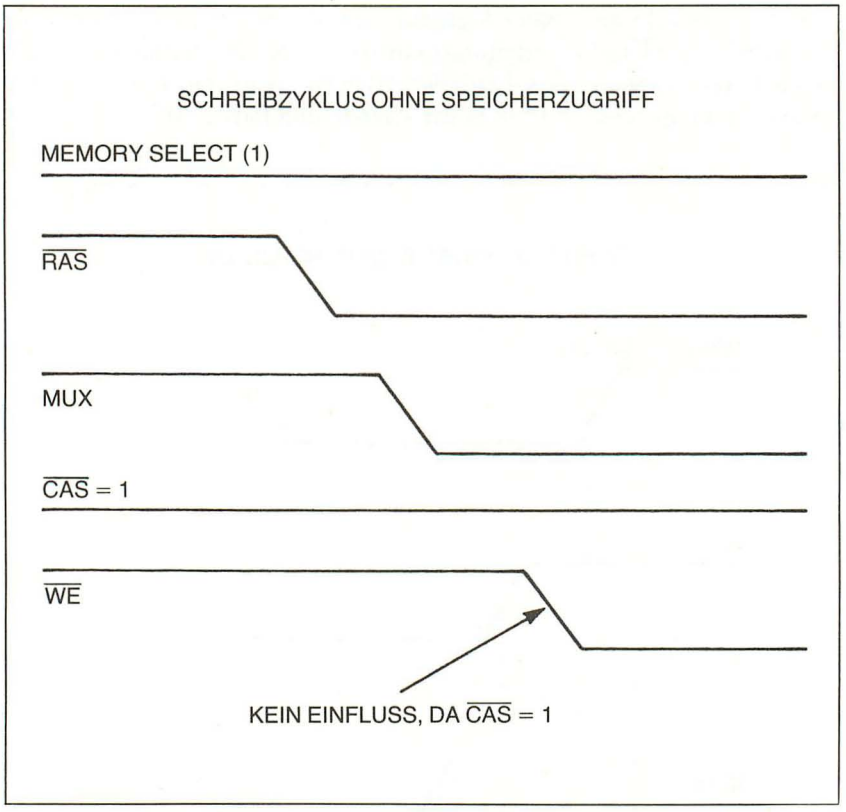


Abb. 4.12: *Timing für ein nicht angesprochenes dynamisches RAM bei einem Schreibzyklus. Da die Speicher-Zugriffs-Leitung logisch 1 bleibt, wird auch kein CAS-Signal generiert.*

Zum Schluß wird die $\overline{WE}$-Leitung am dynamischen RAM aktiv. Daten werden jedoch nicht in das RAM geschrieben, da sowohl $\overline{RAS}$ als auch $\overline{CAS}$ logisch 0 sein müssen, damit der 4116 Daten liest oder schreibt.

Lesevorgang bei einem dynamischen RAM

Wir wollen jetzt zeigen, wie die Daten des dynamischen RAM bei einer Leseoperation auf den Datenbus gebracht werden. Die Technik ist ähnlich der für statische RAMs, ROMs und I/O verwendeten. Abb. 4.13 zeigt eine Möglichkeit der Hardware-Realisierung.

In dieser Schaltung wird der uni-direktionale Puffer 74LS244 benutzt.

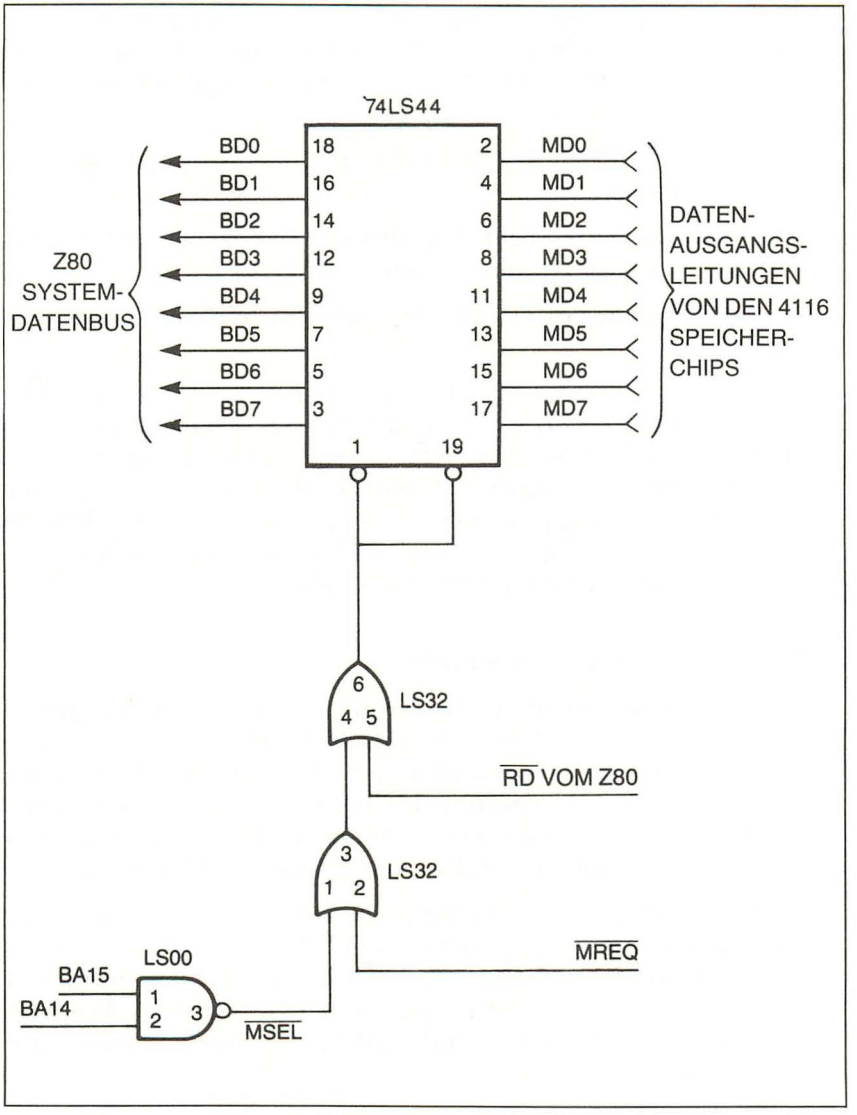


Abb. 4.13: *Schaltung für die Datenübertragung vom Speicher zum Datenbus.*

Der Eingang dieses Puffers liegt an den 4116-Datenausgängen. Der Ausgang des 74LS244 ist direkt mit dem Systemdatenbus verbunden. Sobald der Mikroprozessor Daten vom RAM lesen will, wird der Puffer durchgeschaltet und die Speicherdaten auf den Bus gelegt.

Der Adressierungsvorgang des RAM bei einer Leseoperation ist der gleiche wie bei einer Schreiboperation. $\overline{RAS}$, MUX und $\overline{CAS}$ haben exakt das gleiche Timing. Damit die Daten auf den Bus gelegt werden, müssen folgende elektrische Vorgänge ablaufen:

1. Der $\overline{MREQ}$-Ausgang des Z80 muß logisch 0 sein, was eine Speicheroperation anzeigt.
2. A15 und A14 müssen beide logisch 1 sein, da daraus das Speicher-Selektierungs-Signal gewonnen wird.
3. Der $\overline{RD}$-Ausgang des Z80 muß logisch 0 sein, was eine Lese-Operation anzeigt.

Wie wir in Abb. 4.13 sehen können, sind die Eingänge Pin 1 und 2 des ODER-Gatters dann logisch 0, wenn BA14 und BA15 logisch 1 sind und $\overline{MREQ}$ aktiv ist. Erst wenn die $\overline{RD}$-Leitung auch aktiv (logisch 0) ist, wird der Puffer 74LS244 durchgeschaltet. Die $\overline{RD}$-Leitung bleibt für eine Zeit, die von der Taktfrequenz abhängig ist, auf logischem 0-Pegel. Wenn der Z80 sie wieder auf logisch 1 setzt, werden die Daten elektrisch vom Bus weggenommen und der Lesezyklus beendet.

Auffrischung des dynamischen RAM

Bis jetzt haben wir ignoriert, daß das dynamische RAM aufgefrischt (refreshed) werden muß. Das Auffrischen bedeutet, daß die Informationen in den Speicherzellen des Bauteils periodisch angesprochen werden müssen, damit sie erhalten bleiben. Typischerweise bleiben die Informationen in den Speicherzellen nur einige Millisekunden erhalten und gehen verloren, wenn sie nicht innerhalb dieser Zeit angesprochen werden.

Es gibt verschiedene Techniken zum Daten-Refreshing bei dynamischen RAMs. Die hier gezeigte Technik wird vom Z80 unterstützt. Nehmen wir an, daß zur Datenerhaltung jede Speicherzelle des dynamischen RAM mindestens alle 2 Millisekunden angesprochen werden muß. Das muß auch geschehen, während das RAM bei der Programmausführung nicht selektiert wird.

Wenn Daten vom 4116 gelesen werden, wird die gesamte Speicherzeile des RAM parallel aufgefrischt. Das RAM kann also dadurch komplett aufgefrischt werden, indem alle 128 Zeilenadressen angesprochen werden. Wenn also 128 Zeilen in 2 Millisekunden angesprochen werden müssen, so bleibt für jede Zeile 16 Mikrosekunden Zeit.

Der Z80 ist hervorragend zur Ansteuerung von dynamischen RAMs geeignet, da er eine interne Einrichtung, den Refresh-Counter, besitzt,

der den Anschluß von dynamischen RAMs besonders einfach gestaltet. Der Wert dieses Zählers wird zu bestimmten Zeitpunkten auf den Adreßbus gelegt. Danach wird der Zählerstand jedesmal um eins erhöht.

Diese interne Einrichtung des Z80 nimmt automatisch Rücksicht auf dynamische RAM-Systeme. Bei jedem Befehl-Lese-Zyklus des Prozessors wird der Refresh-Counter-Inhalt auf die Adreßleitungen A0 bis A6 gelegt. Das geschieht während der Zeit, die der Z80 zur Decodierung des Befehls verwendet. Dieser Typ der Refresh-Adressierung wird auch als Cycle-Stealing bezeichnet.

Während die Refresh-Adresse auf dem Adreßbus liegt, ist auch der RFSH-Ausgang des Z80 aktiv. Bei dem dynamischen RAM braucht nur der $\overline{RAS}$-Eingang aktiv werden, um einen Refresh auszulösen. (Erinnern wir uns, daß die $\overline{RAS}$-Steuerleitung die Zeilenadresse im Speicher ablegt.) Die $\overline{CAS}$-Leitung soll nicht aktiv werden, weil das eine Speicher-Lese-Operation auslösen würde. (Das genaue Timing für die Refresh-Operation finden Sie im vollständigen Datenblatt des 4116.)

Abb. 4.14 zeigt den geforderten Signalablauf für eine Refresh-Operation des 4116. Während eines Refresh-Zyklus aktiviert der Z80 die $\overline{MREQ}$-Leitung. Dadurch entsteht ein $\overline{RAS}$-Signal. Die $\overline{RD}$-Leitung ist jedoch nicht aktiv. Die Logik der $\overline{CAS}$-Schaltung verhindert ein Aktivieren die-

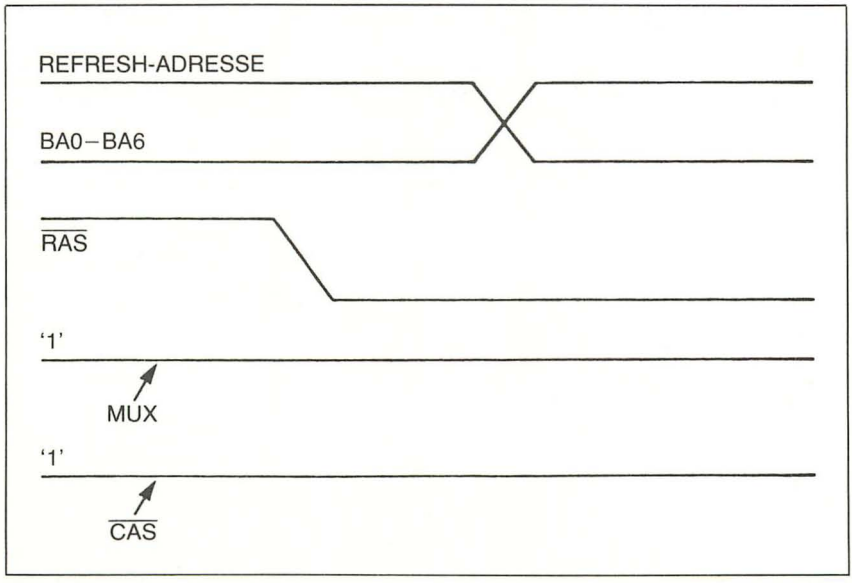


Abb. 4.14: *Darstellung des Timings bei einer Speicher-Refresh-Operation.*

ses Signals. Außerdem verhindert das Fehlen des $\overline{RD}$-Signals, daß die Tri-State-Puffer zum Datenbus durchgeschaltet werden.

Es soll angemerkt werden, daß das Speicher-Refreshing nur solange läuft, wie der Prozessor ein Programm ausführt, das heißt, solange er Befehle

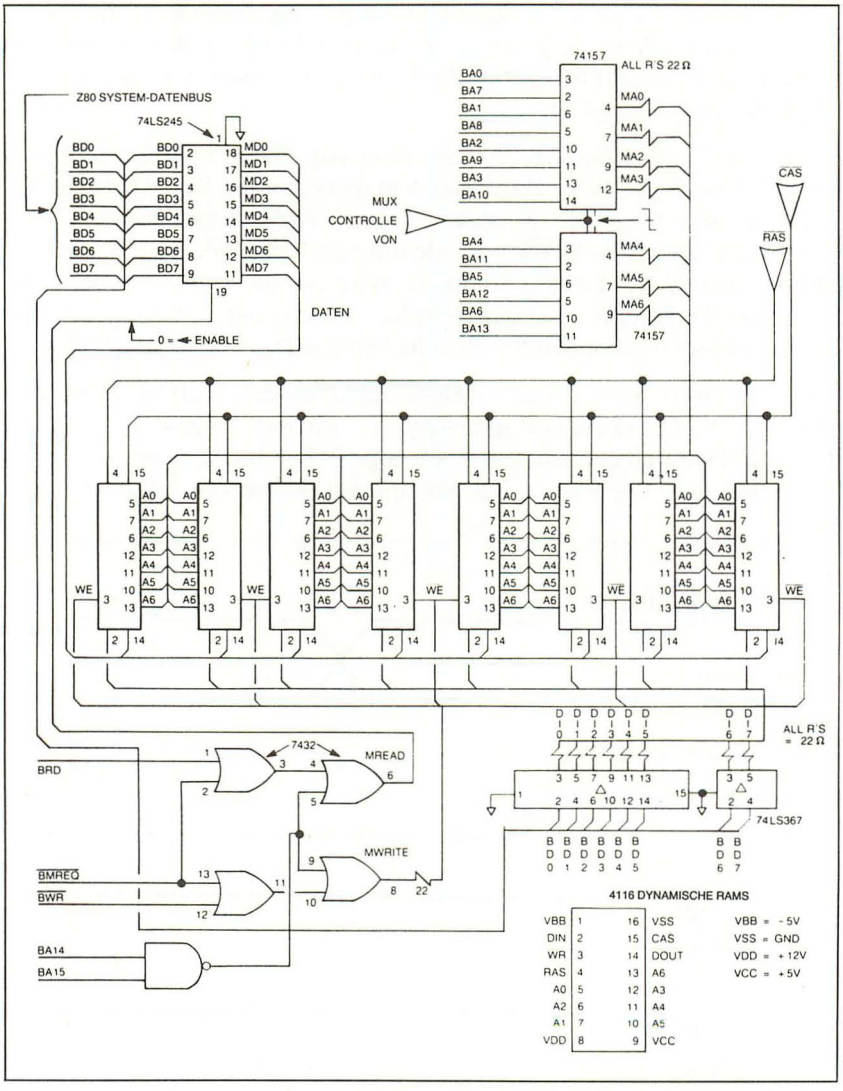


Abb. 4.15: *Komlettes 16K * 8-Bit Speichersystem mit dynamischen RAMs für den Z80-Mikroprozessor.*

liest. Die Zeitintervalle zwischen den Befehlszyklen sind jedoch nicht länger, als sie zum Speicher-Refreshing sein dürfen. Da einige Befehle länger dauern als andere, muß bei der Berechnung der Refresh-Zeiten vom schlimmsten Fall (worst case) ausgegangen werden.

Wenn der Z80 angehalten wird, fällt auch das Refreshing aus. Das gilt z.B. beim Anlegen eines Resets. Dies ist bei der Benutzung von dynamischen RAMs unbedingt zu beachten.

Komplettes Schaltbild eines 16K * 8Bit dynamischen RAM

Abb. 4.15 zeigt die vollständige Schaltung eines dynamischen RAM-Systems für den Z80. Erinnern wir uns, daß die Hardware-Realisierung verschiedene Wege gehen kann. Dieser Entwurf wurde ausgewählt, weil er die grundlegenden Details des Interfacing vom Z80 zu einem typischen dynamischen RAM besonders gut wiedergibt.

Zusammenfassung

In diesem Kapitel haben wir die Grundlagen der elektrischen Kommunikation zwischen dem Z80 und dynamischen RAM besprochen. Dabei haben wir ein typisches Bauteil, den 4116, näher untersucht. Dieses RAM repräsentiert eine allgemeine Gruppe dynamischer RAMs, die gemultiplexte Adreßleitungen benutzt.

Weiter haben wir in diesem Kapitel erklärt, wie das Adreß-Multiplexing durch Hardware realisiert werden kann. Wir haben die Funktion der $\overline{\text{RAS}}$-, MUX- und $\overline{\text{CAS}}$-Steuersignale besprochen und gelernt, wie jedes Signal mit Hilfe des Z80-Mikroprozessors erzeugt werden kann.

Zuletzt haben wir noch etwas über das Refreshing bei dynamischen RAMs kennengelernt. Der Z80 ist durch seine Refresh-Technik im Einsatz mit dynamischen RAMs äußerst beliebt geworden.

Kapitel 5

Interrupts beim Z80

Einführung

In diesem Kapitel beschäftigen wir uns mit der Thematik des Interrupts (Unterbrechung) beim Z80. Wir wollen mit einer Einführung in die Interrupt-Technik zunächst ganz allgemein anfangen. Wir zeigen dann, wie die verschiedenen Interrupt-Typen vom Z80 elektrisch gehandhabt werden. Für jeden neuen Interrupt-Modus wollen wir an einem Beispiel zeigen, was während der Operation genau passiert.

Es ist für das Verständnis der folgenden Kapitel für Sie wichtig zu wissen, wie der Z80 externe Interrupt-Anfragen behandelt. Wenn Sie mit dieser Thematik nicht vollständig vertraut sind, sollten Sie dieses Kapitel sorgfältig lesen.

Was ist ein Interrupt?

Stellen Sie sich folgende Situation vor: Sie haben ein Gespräch mit einer anderen Person. Eine zweite Person kommt herbei und sagt Ihren Namen, um Ihre Aufmerksamkeit zu gewinnen. Es folgt eine Liste von möglichen Reaktionen für diese externe Anfrage:

1. Sie können die zweite Person vollständig ignorieren und mit dem Gespräch fortfahren, als wäre sie nicht da.
2. Sie können an einem geeigneten Punkt Ihre Unterhaltung unterbrechen und Ihre Aufmerksamkeit auf die zweite Person richten.
3. Sie können unmittelbar Ihr Gespräch mit der ersten Person abbrechen und beginnen, mit der zweiten Person zu sprechen.

In jedem Fall fahren Sie mit dem Gespräch mit der ersten Person fort, sobald sie das Gespräch mit der zweiten Person beendet haben.

Die beschriebene Szene mag simpel sein, doch sie beschreibt das Interrupt-Konzept innerhalb eines Mikroprozessor-Systems.

Stellen Sie sich vor, Sie seien die Z80-CPU. Die Person, mit der Sie zuerst sprechen, ist das Hauptprogramm und die zweite Person ist eine externe

Interrupt-Anforderung. (Da ist ein Teil der Hardware, das die Aufmerksamkeit der CPU wünscht.) Die Z80-CPU muß diese externe Anfrage irgendwie handhaben. Dafür gibt es verschiedene Wege. Die oben genannten drei Möglichkeiten sind die häufigsten.

Mit Hilfe dieser kleinen Einführung wollen wir jetzt die Details der Z80-Interrupt-Technik untersuchen.

Wo kommen die Interrupt-Anfragen her?

Ein Mikroprozessor-System kann aus vielen verschiedenen Hardware-Komponenten zusammengesetzt sein. Das kann z.B. ein Datensichtgerät sein, ein Drucker, Floppy- oder Festplattenlaufwerke, ein Zeitgeber, ein Steuermotor oder ein Digital-Analog-Wandler, um nur ein paar zu nennen. Die meisten der externen Hardware-Komponenten brauchen die Aufmerksamkeit der CPU nur zu bestimmten Zeiten; im übrigen arbeiten sie selbständig.

Lassen Sie uns z.B. annehmen, Ihr System beeinhaltet eine Hardware-Uhr und die Uhrzeit soll auf dem Bildschirm angezeigt werden. Die Ziffern der Uhr ändern sich nur einmal pro Sekunde. Mit anderen Worten, nur einmal pro Sekunde braucht die Uhr den Service der CPU, um die Zeit zu lesen und auf dem Bildschirm zu schreiben. Während der übrigen Zeit kann die CPU andere Aufgaben erfüllen.

Der Punkt hierbei ist, daß die externe Uhren-Hardware die Aufmerksamkeit der CPU nicht die ganze Zeit braucht. Die Hardware fordert nur dann die CPU an, wenn sie gebraucht wird. Das kann mit Hilfe des CPU-Interrupt-Systems erfolgen, wobei die CPU jede Sekunde eine elektrische Interrupt-Anfrage von der Uhren-Hardware bekommt. Daraufhin unterbricht die CPU, was immer sie auch gerade macht, ihre Arbeit und liest die Uhrzeit. Nachdem die Zeit gelesen und auf dem Bildschirm angezeigt wurde, setzt die CPU die Programmausführung an der Stelle fort, wo sie unterbrochen wurde.

Dies ist ein einfaches Beispiel für einen Interrupt. Einen ähnlichen Ablauf wird auch andere Hardware auslösen, die eine elektrische Interrupt-Anforderung zur CPU macht. Die Eingangsfrage, von wo die Interrupts kommen, läßt sich also so beantworten: Sie kommen von der Mikroprozessor-System-Hardware.

Nicht-maskierbare Interrupts

Es gibt vier verschiedene Interrupt-Leitungen beim Z80-Mikroprozessor: $\overline{\text{INT}}$, $\overline{\text{NMI}}$, $\overline{\text{BUSRQ}}$ und $\overline{\text{RES}}$. Jede davon kann einen Interrupt auslö-

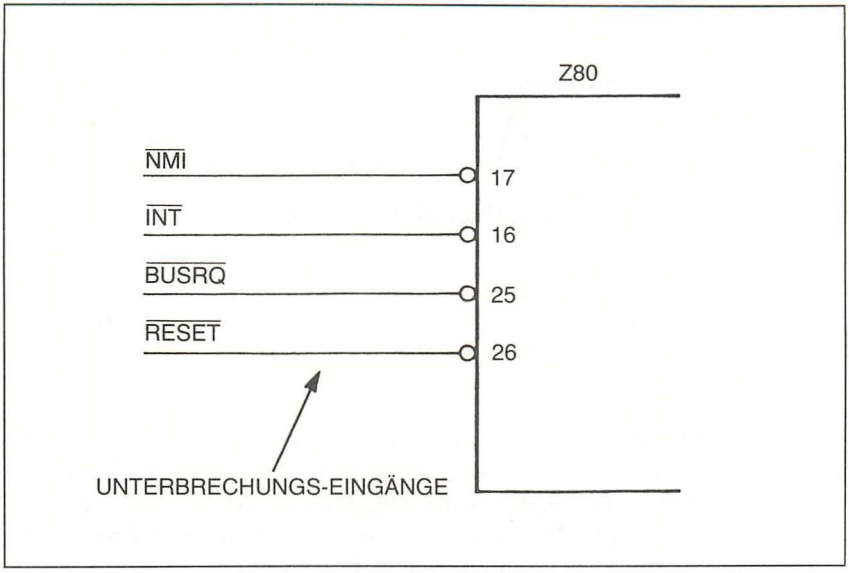


Abb. 5.1: *Die physikalischen Interrupt-Leitungen des Z80.*

sen. Abb. 5.1 zeigt die vier Eingänge. In diesem Abschnitt wollen wir die $\overline{\text{NMI}}$- oder nicht-maskierbare Interrupt-Leitung besprechen. Wir wollen mit diesem Interrupt beginnen, da er auf nur eine Art und Weise funktioniert. Die meisten hierbei besprochenen Punkte gelten jedoch auch für die anderen Interrupt-Eingänge.

Ein nicht-maskierbarer Interrupt wird vom Z80 so schnell, wie es elektrisch möglich ist, berücksichtigt. Diese Interrupt-Art fällt unter die dritte Möglichkeit unserer Liste am Anfang dieses Kapitels. Da der Z80 einen nicht-maskierbaren Interrupt hat, kann man folgern, daß er auch einen maskierbaren Interrupt haben muß (den $\overline{\text{INT}}$-Eingang, den wir später besprechen).

Pin 17 des Z80 wird $\overline{\text{NMI}}$ genannt (non-maskable interrupt). Dieser Interrupteingang reagiert auf eine negative Flanke des Signals (siehe Abb. 5.2). Der $\overline{\text{NMI}}$-Eingang arbeitet völlig asynchron zum Z80. Das bedeutet, daß die externe Hardware zu einem beliebigen Zeitpunkt ihre Anforderung machen kann, ohne den internen Arbeitszustand der CPU zu berücksichtigen.

Die nächste Frage ist, wann der $\overline{\text{NMI}}$-Interrupt-Eingang elektrisch von der CPU berücksichtigt wird. Generell wird die $\overline{\text{NMI}}$-Anforderung am

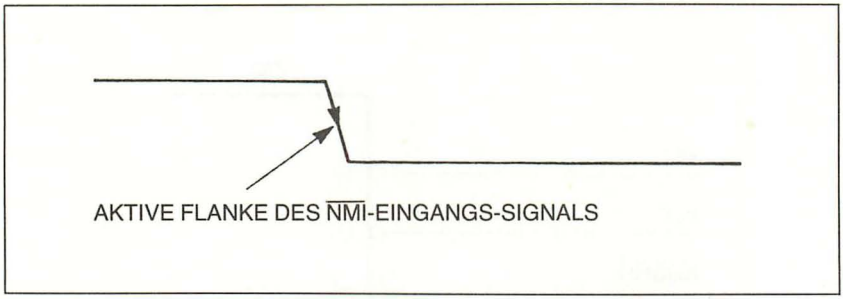


Abb. 5.2: *Der NMI-Interrupt-Eingang ist aktiv mit der fallenden Flanke des Signals. Intern wird hierdurch eine Markierung gesetzt.*

Ende des laufenden Befehlszyklus berücksichtigt. Das ermöglicht dem Z80, den laufenden Befehlsablauf zu beenden, bevor irgendeine Reaktion auf den Interrupt gegeben wird. Durch diese Art des Ablaufes kann die interne Logik der CPU einfacher gestaltet werden. Die Antwortzeit auf einen Interrupt wird dadurch auch nicht wesentlich verlängert.

Die erste Aktion einer NMI-Operation ist das Abspeichern des PC-Registers auf den System-Stack. Das rettet die Adresse, zu der der Z80 nach Beendigung des Interrupt-Servive zurückkehren soll, im Speicher.

Als nächstes wird der Zustand des Interrupt-Flip-Flops IFF1 in IFF2 gespeichert. IFF1 wird als „Interrupt-Enable Flip-Flop" bezeichnet, und seine Funktion ist die Freigabe oder das Sperren von Interrupts vom INT-Eingang des Z80. Für nicht-maskierbare Interrupts wird es nicht benötigt.

IFF1 wird in IFF2 gepeichert, weil sein logischer Zustand während der NMI-Behandlung erhalten bleiben soll. IFF1 selber wird während des NMI-Servive dann auf logisch 0 gesetzt, um damit andere Interrupts (vom INT-Eingang) zu sperren. (Später bei der Beendigung des NMI-Service muß der ursprüngliche Arbeitszustand wieder hergestellt werden. Das geschieht dadurch, daß IFF1 seinen Originalzustand zurückerhält.)

Nachdem IFF1 in IFF2 gespeichert worden ist und IFF1 logisch 0 gesetzt wurde, springt der Z80 zur Adresse 0066H im Speicher.

Lassen Sie uns noch einmal schnell die vier Ereignisse durchgehen, die auftreten, wenn der Z80 einen NMI annimmt:

1. Der PC wird auf den Stack geschoben (pushed).
2. IFF1 wird in IFF2 gespeichert.

Abb. 5.3: *Der typische Anfang einer Interrupt-Service-Routine.*

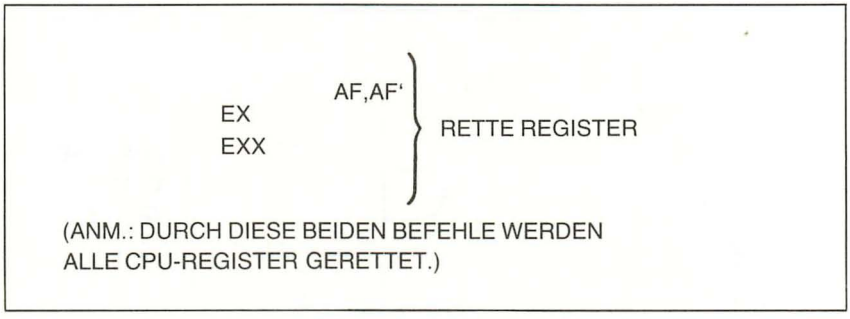


Abb. 5.4: *Eine Alternative zum Retten der Z80-internen Register.*

3. IFF1 wird zu logisch 0 gesetzt.
4. Der Z80 springt zur Adresse 0066H.

Adresse 0066H beeinhaltet die Service-Routine, die bei einem $\overline{\text{NMI}}$ ausgeführt wird. (Beachten Sie, daß die CPU-Register bei einem Interrupt nicht gerettet werden. Der Programmierer muß alle verwendeten Register in der Interrupt-Service-Routine retten.) Ein typischer Start einer Interrupt-Service-Routine wird in Abb. 5.3 gezeigt.

Eine Alternative zum Retten aller Register ist die Benutzung der Register-Tausch-Befehle (exchange instructions). Dies wird ermöglicht durch den Gebrauch des zweiten Registersatzes während der Ausführung eines Interrupts. Abb. 5.4 zeigt die hierzu verwendeten Befehle.

Löschen der $\overline{\text{NMI}}$-Anforderung

Die $\overline{\text{NMI}}$-Anforderung wurde elektrisch in Hardware außerhalb des Z80 generiert. Eine der Aufgaben der Service-Routine ist das Löschen der

zum Interrupt führenden Bedingung (wenn möglich oder erforderlich). Eine Möglichkeit zum Löschen der Interrupt-Anforderung kann das Schreiben zu einem Ausgabeport sein. (Später werden wir noch eine andere Möglichkeit kennenlernen.) Der Punkt hier ist der, daß der Programmierer für das elektrische Abschalten der Interrupt-Anforderung verantwortlich ist.

Ende der $\overline{\text{NMI}}$-Service-Routine

Am Ende der $\overline{\text{NMI}}$-Service-Routine führt der Z80 einen RETN-Befehl (return from non-maskable interrupt) aus. An diesem Punkt werden die

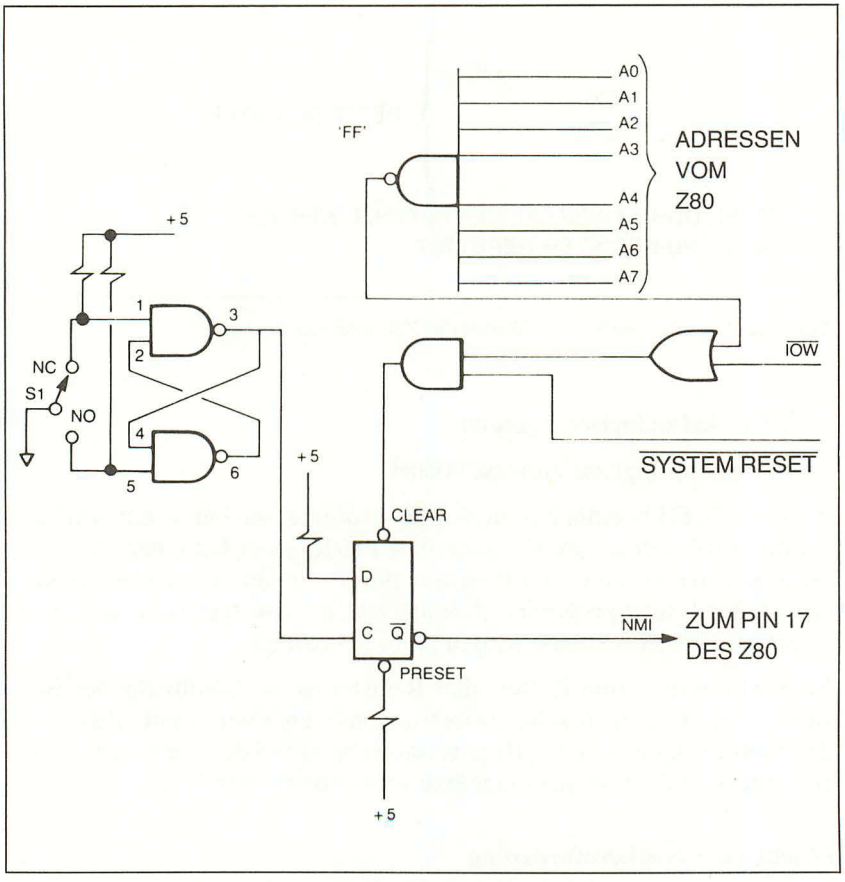


Abb. 5.5: *Schaltung zur Erzeugung einer Interrupt-Anforderung durch externe Hardware.*

beiden obersten Bytes auf dem Stack als Rücksprungadresse benutzt. IFF2 wird zu IFF1 kopiert. (Anm.: Sie können den RET-Befehl auch benutzen, jedoch wird dabei IFF2 nicht zu IFF1 kopiert.)

Beispiel für einen $\overline{\text{NMI}}$

Hier nun ein Beispiel, das die Benutzung des nicht-maskierbaren Interrupts illustriert. Wir zeigen eine Hardware-Technik zur Auslösung einer $\overline{\text{NMI}}$-Anforderung und die zum Service notwendige Basis-Software. Abb.5.5 zeigt die Schaltung für diese Interrupt-Anforderung. Wir wollen sie einmal näher untersuchen.

Wenn der Tastenschalter S1 betätigt wird, bekommt Pin 3 des 7400 logisches 0-Potential. Dadurch wird auch der Takteingang des D-Flip-Flops logisch 0. Zu diesem Zeitpunkt passiert nichts. Wenn der Schaltknopf losgelassen wird, geht der Takteingang auf logisch 1. Damit wird der $\overline{\text{Q}}$-Ausgang des D-Flip-Flops auf logisch 0 gesetzt, was einen $\overline{\text{NMI}}$ am Z80 auslöst.

Der $\overline{\text{NMI}}$-Eingang bleibt logisch 0, bis der Z80 eine Ausgabe-Schreib-Operation zum Port 0FFH ausführt. Dann wird der $\overline{\text{Q}}$-Ausgang des D-Flip-Flops und somit der $\overline{\text{NMI}}$-Eingang des Z80 in den logischen 1-Zustand zurückgeschaltet.

Es muß angemerkt werden, daß der $\overline{\text{NMI}}$-Eingang flanken-sensitiv und nicht zustands-sensitv ist. Das bedeutet, daß der Wechsel von logisch 1 auf logisch 0 als Unterbrechungsanforderung gedeutet wird (siehe Abb.5.6).

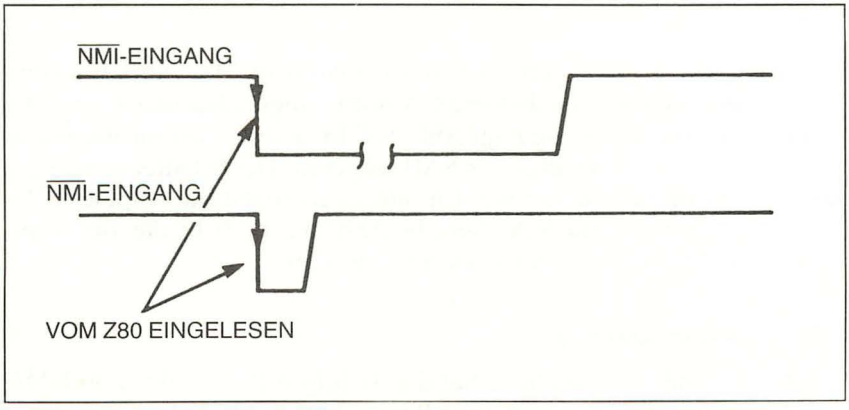


Abb. 5.6: *Timing des $\overline{\text{NMI}}$-Zustands. Der $\overline{\text{NMI}}$ kann weggeschaltet werden, bevor er vom Mikroprozessor beachtet wird.*

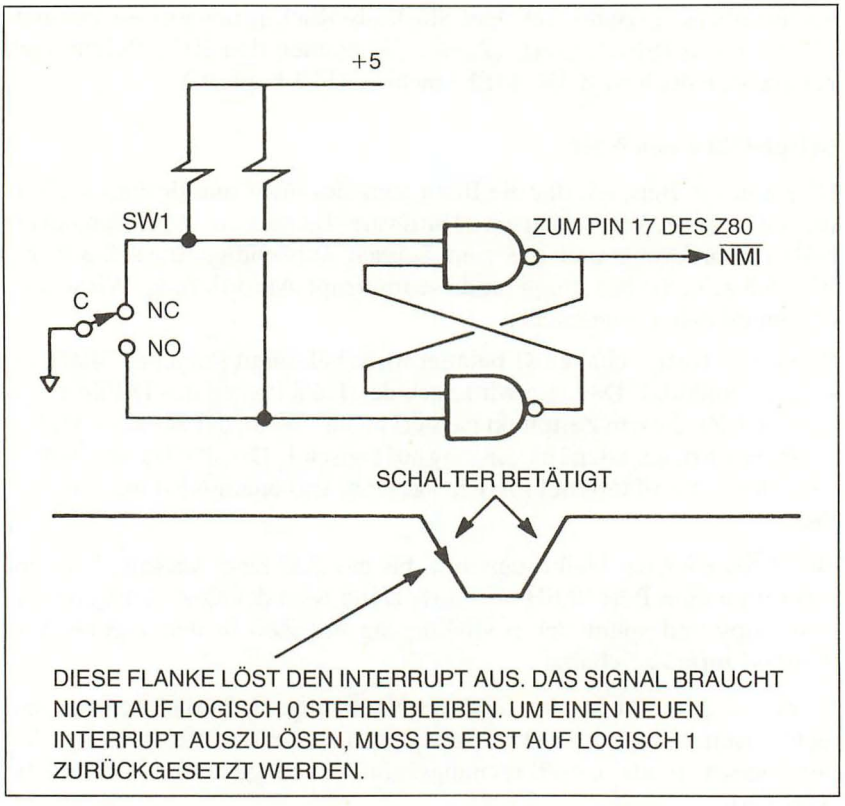


Abb. 5.7: *Besonders einfache Methode zur Erzeugung einer externen $\overline{NMI}$-Anforderung.*

Aufgrund der flanken-getriggerten Charakteristik des $\overline{NMI}$-Eingangs kann eine sehr einfache Interrupt-Anforderungs-Schaltung verwendet werden. So eine Schaltung zeigt Abb. 5.7. In dieser Schaltung wird beim Betätigen des Druckschalters ein $\overline{NMI}$ ausgelöst. Der Schalter muß losgelassen und erneut gedrückt werden, um einen neuen Interrupt auszulösen. Im Gegensatz zur Schaltung in Abb. 5.5 braucht die Interrupt-Anfrage nicht vom Z80 zurückgeschaltet werden.

$\overline{NMI}$ -Zusammenfassung

Die vorstehende Besprechung hat die wichtigen Punkte des Z80-$\overline{NMI}$-Eingangs gezeigt. Wir wollen diese Punkte kurz wiederholen:

1. Der $\overline{NMI}$ ist nicht maskierbar, wird also immer angenommen.

2. Der $\overline{\text{NMI}}$ ist flanken-getriggert mit dem Übergang von logisch 1 auf 0.
3. Der Z80 schiebt den PC auf den Stack.
4. IFF1 wird in IFF2 gespeichert, was den Zustand des Interrupt-Flip-Flops rettet.
5. IFF1 wird logisch 0 gesetzt und somit andere Interrupts verhindert.
6. Der Z80 springt zur Adresse 0066H und beginnt, den dort stehenden Code auszuführen.
7. Der RETN-Befehl holt die PC-Adresse und den logischen Zustand des Interrupt-Freigabe-Flip-Flops zurück.

Der $\overline{\text{INT}}$-Eingang

Wir wollen jetzt die Arbeitsweise des $\overline{\text{INT}}$-Eingangs Pin 16 des Z80 untersuchen. Den größten Teil des für den $\overline{\text{NMI}}$ präsentierten Materials gilt auch für den $\overline{\text{INT}}$-Eingang.

Einer der elektrischen Eigenschaften des $\overline{\text{INT}}$-Eingangs des Z80 ist es, daß er logisch ausmaskiert werden kann. Das bedeutet, daß mit dem Z80-Befehl DI (Disable Interrupts) der $\overline{\text{INT}}$-Eingang logisch abgeschaltet wird. Wie wir wissen, ist IFF1 das Z80-interne Flip-Flop, das den $\overline{\text{INT}}$-Pin sperrt oder freigibt. Es wird durch die DI-Anweisung auf logisch 0 gesetzt.

Bislang haben wir die Verwendung von IFF1 noch nicht detailliert besprochen. Im Gegensatz zum $\overline{\text{NMI}}$ ist der $\overline{\text{INT}}$-Eingang zustands-sensitiv. Das bedeutet, daß der Eingang logisch 0 bleiben muß, bis der Z80 ihn abfragt. Diese Abfrage wird am Ende des letzten Maschinenzyklus einer Befehlsausführung gemacht. Erinnern wir uns, daß der $\overline{\text{NMI}}$-Eingang abgeschaltet werden konnte, bevor der Z80 auf ihn reagiert. Wir wissen nun zwei Hauptpunkte über den Z80-$\overline{\text{INT}}$-Eingang:

1. Der $\overline{\text{INT}}$ kann vom Z80 mit dem DI-Befehl elektrisch ausmaskiert werden.
2. Der $\overline{\text{INT}}$-Eingang muß logisch 0 bleiben, bis er vom Z80 abgefragt wird. Der $\overline{\text{INT}}$-Eingang ist zustands-sensitiv im Gegensatz zum flanken-senitiven $\overline{\text{NMI}}$. Das bedeutet, daß der $\overline{\text{INT}}$-Eingang abgeschaltet werden muß, nachdem er bedient wurde. Wird das nicht getan, wird eine neue Interrupt-Anfrage durchgeführt. Es ist Aufgabe der Interrupt-Service-Routine, die Interrupt-Anforderung ordentlich zurückzusetzen, um ungewollte weitere Interrupts zu verhindern.

Es gibt eine Reihe spezieller Peripheriebausteine für den Z80, die automatisch die Interrupt-Anforderung zum richtigen Zeitpunkt zurücksetzen. Ein Beispiel dafür ist das Z80-PIO. Das PIO erkennt, wenn der Z80 eine RETI-Anweisung (Return from Interrupt) ausführt und nimmt die Anforderung von der System-$\overline{\text{INT}}$-Leitung. Wir werden das PIO und einige andere Bausteine später besprechen.

Es gibt nur einen $\overline{\text{INT}}$-Eingang am Z80, aber er kann in drei verschiedenen Betriebsarten (modes) arbeiten. Wenn Sie die Z80-Interrupt-Struktur benutzen wollen, müssen Sie den gewünschten Modus programmieren. Das muß geschehen, bevor irgendein Interrupt bei der CPU eintrifft. Zu unterscheiden sind Modus 0, Modus 1 und Modus 2. Die drei Befehle zur Programmierung des jeweiligen Modus sind: IM0, IM1 und IM2.

Wenn der Z80 eingeschaltet oder zurückgesetzt wird, ist automatisch Modus 0 selektiert. Außerdem ist IFF1 logisch 0 gesetzt und damit die Interrupts gesperrt. Lassen Sie uns nun die drei Interrupt-Arten einzeln aufzeigen und ihre Unterschiede und Gleichheiten ausarbeiten.

Interrupt-Modus 1

Lassen Sie uns mit dem Interrupt-Modus 1 beginnen. Wir fangen mit diesem Modus an, weil er am ehesten dem $\overline{\text{NMI}}$ entspricht und weil wir von bekanntem Boden ausgehen wollen. Der Z80 wird mit dem IM1-Befehl in den Interrupt-Modus 1 gesetzt.

Bei dieser Betrachtung gehen wir davon aus, daß die Interrupts auf der $\overline{\text{INT}}$-Leitung mit dem EI-Befehl (enable interrupt) freigegeben wurden.

Wenn die $\overline{\text{INT}}$-Leitung logisch 0 ist und der Z80 darauf reagiert, geschieht folgendes:

1. IFF1 wird auf logisch 0 gesetzt, was weitere Interrupts von der $\overline{\text{INT}}$-Leitung sperrt.
2. IFF2 wird auf logisch 0 gesetzt. IFF2 wird, wie wir wissen, bei einem $\overline{\text{NMI}}$ als Zwischenspeicher für IFF1 benutzt. Bei diesem Interrupt wird es jedoch nicht benutzt.
3. Der PC wird auf den Stack geschoben. Damit ist die Adresse, bei der der Prozessor nach Beendigung der Interrupt-Service-Routine fortfahren soll, gesichert.
4. Der Z80 springt zur Adresse 0038H des Systemspeichers und startet den dort befindlichen Programmcode.

Wie Sie sehen, ist das im wesentlichen der gleiche Ablauf, wie er auch vom $\overline{\text{NMI}}$ eingeleitet wird.

Während der Ausführung der Interrupt-Service-Routine müssen die CPU-Register gesichert werden. Dies kann entweder durch Benutzung des zweiten Registersatzes oder durch PUSH-Befehle geschehen.

Aufgabe der Interrupt-Service-Routine ist es, die Interrupt-Anfrage vom Z80 Pin 16 zu entfernen. Die dafür benötigte Software ist von System zu System verschieden und abhängig von der Hardware, die den Interrupt ausgelöst hat. Abb. 5.5 zeigt eine Hardware-Lösung zur Erzeugung einer Interrupt-Anforderung. In dieser Schaltung dient eine Ausgabe-Schreib-Operation zum Port 0FFH zum Rücksetzen der Anforderung. Damit der Z80 weitere Interrupts akzeptiert, muß erst IFF1 auf logisch 1 gesetzt werden. Dies geschieht, wenn die CPU vor dem Rücksprung aus der Service-Routine einen EI-Befehl verarbeitet.

Als letztes muß der Programmierer der CPU erlauben, das Programm an der Stelle fortzusetzen, an der es unterbrochen wurde. Dafür wird der RET oder der RETI-Befehl eingesetzt. Der RET-Befehl holt die letzten beiden Bytes des System-Stack und schreibt sie in den PC (Program Counter) des Z80. Diese beiden Bytes bilden die Adresse des Befehls, der ausgeführt worden wäre, wenn kein Interrupt gekommen wäre. Der RET-Befehl steht üblicherweise auch am Ende von Unterprogrammen.

Der RETI-Befehl hat den gleichen Ablauf wie der RET-Befehl. Der Hauptunterschied liegt jedoch darin, daß verschiedene Peripheriebausteine, wie z.B. der Z80-PIO, daraufhin ausgelegt sind, den RETI-Befehlscode mitzuberücksichtigen. Wenn ein externes Bauteil diesen Befehl sieht, wird die Interrupt-Anforderung auf den neuesten Stand gebracht. (Mehr darüber werden Sie in späteren Kapiteln kennenlernen.)

Abb. 5.8 zeigt das Beispiel einer Interrupt-Service-Routine. Diese Routine geht davon aus, daß der Z80 im Modus 1 arbeitet und daß der Interrupt mit der Hardware aus Abb. 5.5 erzeugt wurde. Der Befehl OUT (0FFH),A wird benutzt, um die Interrupt-Anforderung zu löschen, bevor mit dem EI-Befehl die Interrupts wieder freigegeben werden.

Es soll angemerkt werden, daß die Interrupts erst nach Ausführung des dem EI-Befehl folgenden Befehls freigegeben werden. Das erlaubt dem Z80, einen Rücksprung auszuführen, bevor eine neue Interrupt-Anforderung angenommen wird.

Die Interrupts sind beim Start der Interrupt-Service-Routine immer gesperrt. Das geschieht dadurch, daß der Z80 automatisch IFF1 logisch 0 setzt. IFF1 bleibt logisch 0, bis es durch die System-Software auf logisch

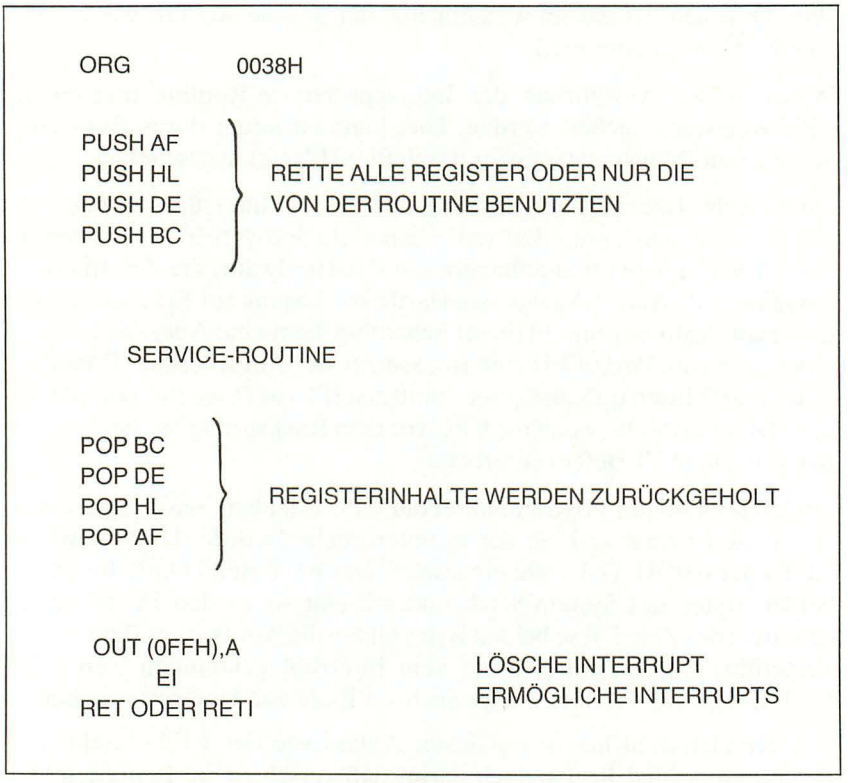


Abb. 5.8: *Einfache Interrupt-Service-Routine für den Interrupt-Modus 1.*

1 gesetzt wird. Das erfolgt am Ende der Service-Routine, wenn keine weiteren Interrupts während ihrer Ausführung auftreten sollen. Der Programmierer kann natürlich auch innerhalb der Service-Routine die Interrupts schon wieder freigeben. Es ist Aufgabe des System-Designers zu entscheiden, ob eine Interrupt-Routine von einem neuen Interrupt unterbrochen werden darf.

Interrupt-Modus 0

In diesem Abschnitt besprechen wir die zweite Betriebsart für das Z80-Interrupt-System: den Modus 0. Modus 0 ist angewählt, wenn die CPU eingeschaltet wird, wenn ein Reset gegeben wird oder wenn eine IM0-Anweisung ausgeführt wird. Diese Interrupt-Art wird häufig mit der des 8080-Mikroprozessors gleichgesetzt.

Beim Modus 0 reagiert die CPU auf eine Interrupt-Anforderung in gleicher Weise wie im Modus 1. Doch statt nur zur Adresse 0038H zu springen, kann sie jetzt auf eine von acht vorgewählten Adressen oder sogar zu jeder möglichen Speicheradresse springen. Dies ist eine sehr schöne Möglichkeit. Wir wollen jetzt sehen, wie sie funktioniert.

Wenn wir annehmen, daß eine Interrupt-Anforderung eingetroffen ist und vom Z80 akzeptiert wurde, wird als nächstes folgendes vor sich gehen.

1. IFF1 und IFF2 werden logisch 0 gesetzt und damit weitere Interrupts gesperrt.
2. Der $\overline{M1}$-Ausgang und der $\overline{IORQ}$-Ausgang werden logisch 0. Dieser Zustand tritt nur bei einem Interrupt ein und trägt daher den Namen „Interrupt-Bestätigung" (interrupt acknowledge).
3. Externe Hardware decodiert den Zustand, daß $\overline{M1}$ und $\overline{IORQ}$ logisch 0 sind und gibt ein einzelnes Datenbyte auf den Z80-Datenbus aus. Dieses Datenbyte ist der Befehlscode für einen RST 0-RST 7. Es kann auch ein CALL-Befehl sein. Wir gehen hier jedoch davon aus, daß ein Byte auf den Datenbus gelegt wird, das einem RST-Befehl entspricht.
4. Der Z80 liest das Byte und interpretiert es als Befehlscode.
5. Der PC wird auf den System-Stack geschoben.
6. Zuletzt springt der Z80 zu der vom RST-Befehl angegebenen Adresse.

Diese Adressen sind:

RST	Datenbyte	Speicheradresse (Hex)
0	C7	0000
1	CF	0008
2	D7	0010
3	DF	0018
4	E7	0020
5	EF	0028
6	F7	0030
7	FF	0038

Abb. 5.9 zeigt die Ereignisfolge für die Hardware bei dieser Operation. Ein Schaltbild für eine mögliche Realisierung zeigt die Abb. 5.10. Die Schaltung arbeitet folgendermaßen:

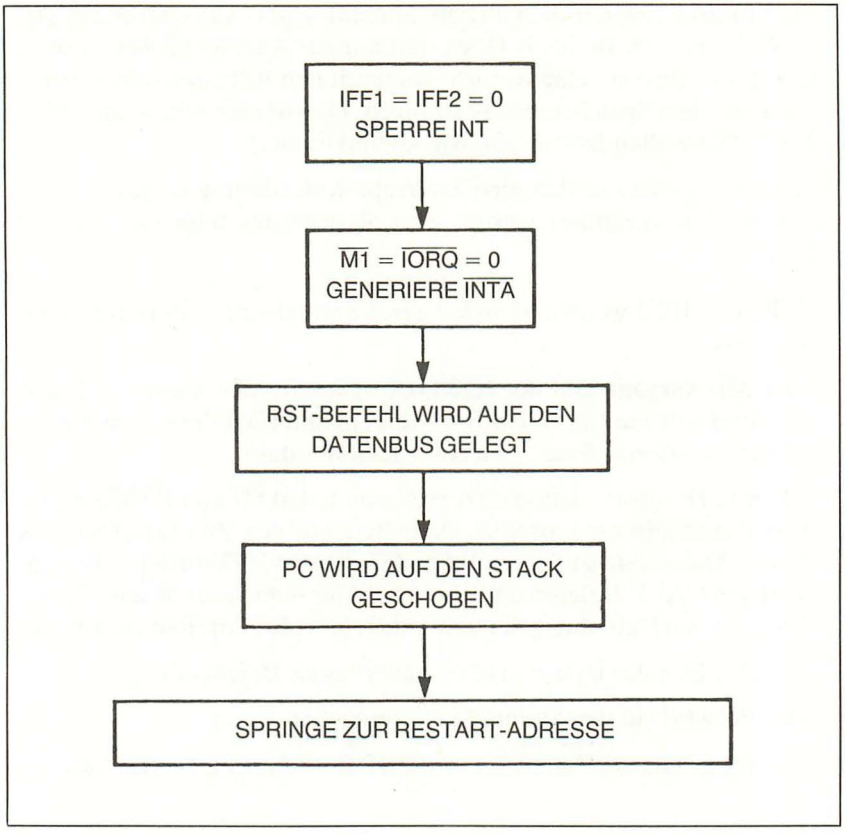


Abb. 5.9: *Flußdiagramm des internen Ablaufs nach einer Interrupt-Anforderung im Modus 0.*

$\overline{M1}$ und $\overline{IORQ}$ in der Schaltung gehen zu den Eingängen eines ODER-Gatters. Wenn die Interrupt-Anfrage beim Z80 eingegangen ist und akzeptiert wurde, werden $\overline{M1}$ und $\overline{IORQ}$ logisch 0, was eine Interruptbestätigung bedeutet. Dadurch wird der Ausgang des ODER-Gatters logisch 0. In unserem Beispiel schaltet das die Tri-State-Puffer durch und legt DF hexadezimal auf den Datenbus. DF ist ein RST 3-Befehl. Für den Z80 bedeutet das einen CALL zur Speicheradresse 0018H.

Dieses Beispiel zeigte eine Möglichkeit, ein Datenbyte auf den System-Datenbus zu legen. Es wurden alle Aspekte für die Benutzung des Interrupt-Modus 0 berücksichtigt. Es ist natürlich nur für ein externes Bauteil, das eine Interrupt-Anforderung macht, ausgelegt. In einem späteren

Abschnitt dieses Kapitels werden wir auch behandeln, wie Mehrfach-Interrupt-Anforderungen vom Z80 gehandhabt werden. Im Moment ist es nur wichtig, daß Sie verstehen, wie die CPU die Interrupts im Modus 0 behandelt.

Wie wir am Anfang dieses Abschnitts angemerkt haben, kann statt des RST-Befehls auch ein CALL-Befehl auf den Datenbus gelegt werden. Wenn der Z80 den CALL-Befehlscode liest, erwartet er noch zwei weitere Bytes.

Die beiden nächsten Zyklen nach der Interrupt-Bestätigung sind Speicher-Lese-Zyklen. Während dieser Zyklen wird die Adresse für die CALL-Anweisung in die CPU eingelesen. Es ist Aufgabe der externen Hardware, die Adresse zur richtigen Zeit auf den Datenbus zu legen. Das Flußdiagramm in Abb. 5.11 zeigt genau, was geschieht, wenn ein RST- oder ein CALL-Befehl während des Interrupt-Bestätigungs-Zyklus auf

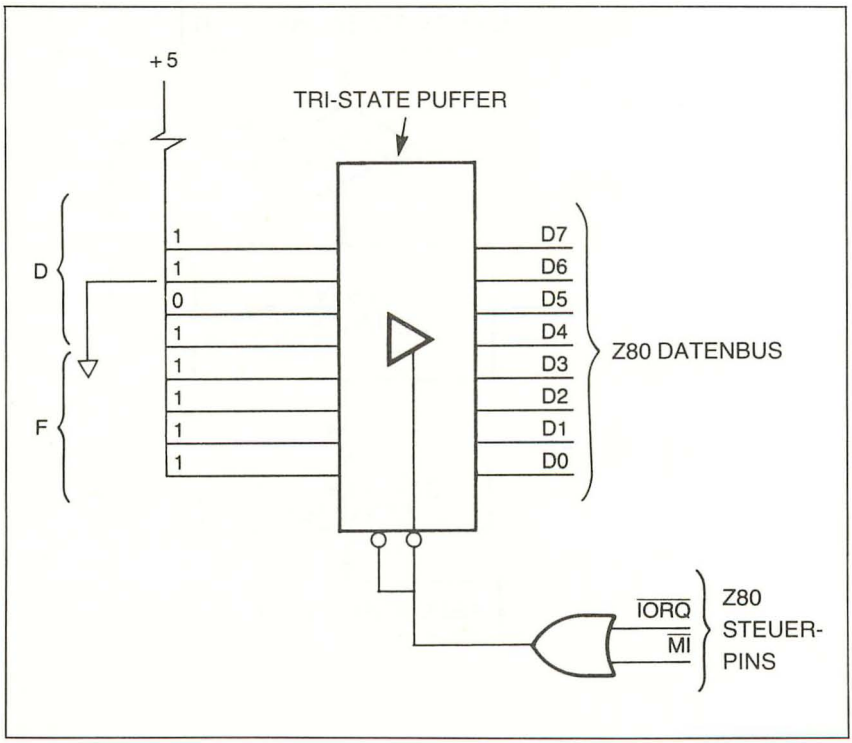


Abb. 5.10: *So kann ein RST-Vektor bei einem Interrupt-Bestätigungs-Zyklus auf den Datenbus gelegt werden.*

den Datenbus gelegt wird. Abb. 5.11 zeigt auch, daß die RST-Anweisung schneller ausgeführt wird, aber die CALL-Anweisung bietet die Möglichkeit, zu jeder beliebigen Adresse zu springen um die Interrupt-Service-Routine auszuführen.

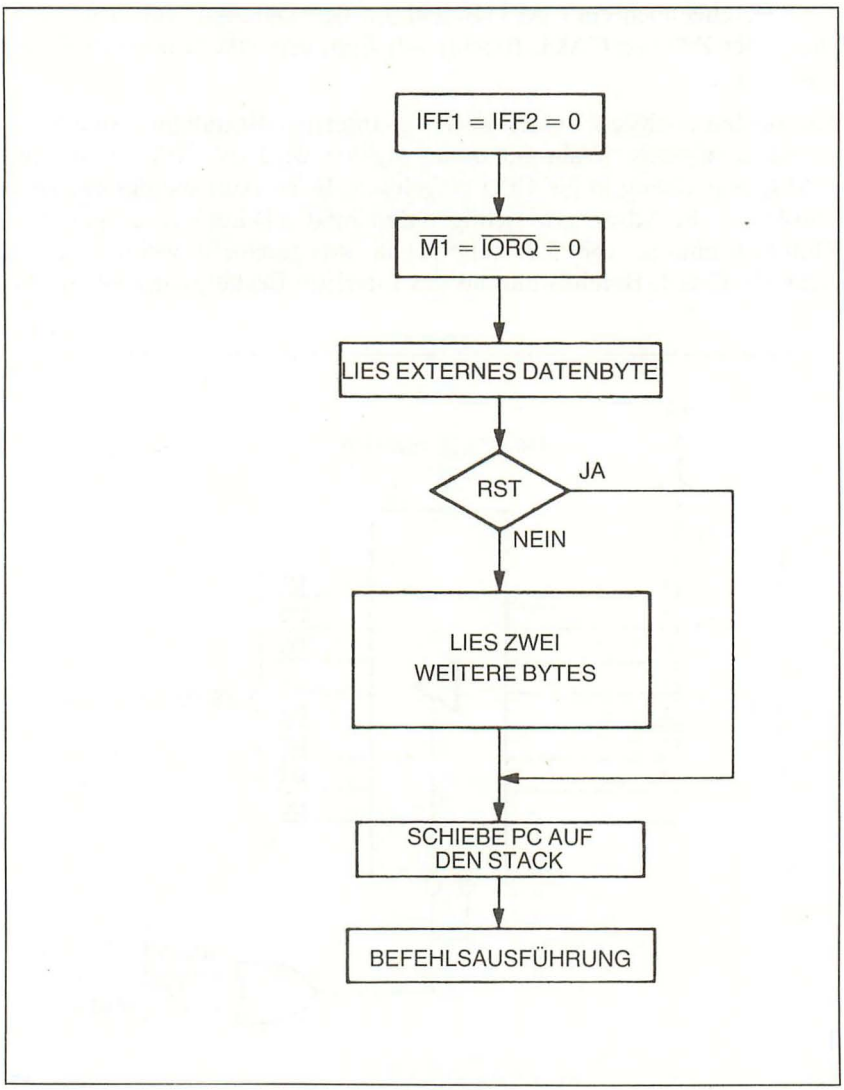


Abb. 5.11: *Flußdiagramm der Ereignisse, wenn der Z80 eine RST- oder CALL-Anweisung während des Interrupt-Bestätigungs-Zyklus erhält.*

Interrupt-Modus 2

In diesem Abschnitt wollen wir die dritte Interrupt-Betriebsart des Z80 behandeln, den Modus 2. Er ist besonders leistungsfähig. Der Modus 2 ist gesetzt, wenn der Z80 einen IM2-Befehl ausführt. Die Interrupt-Anforderung arbeitet in genau der gleichen Weise wie beim Modus 0 und 1. Wir wollen uns jetzt aber darauf konzentrieren, wie der Z80 auf eine Interrupt-Anforderung im Modus 2 reagiert.

Das allgemeine Konzept der Modus-2-Struktur ist folgendes: Wenn der Z80 die Interrupt-Anfrage dadurch bestätigt, daß er $\overline{M1}$ und $\overline{IORQ}$ zur gleichen Zeit aktiviert, legt die externe Hardware ein Datenbyte auf den System-Datenbus. Das ist der gleiche Vorgang wie beim Interrupt-Modus 0, doch hier endet die Gleichheit auch. Der Z80 nimmt das Datenbyte und bildet mit Hilfe eines internen 8-Bit-Registers ein 16-Bit-Wort daraus. Das interne Register heißt I-Register. (siehe Abb. 5.12)

Die Daten auf dieser 16-Bit-Adresse bilden eine weitere Adresse. Diese neue Adresse ist die absolute Speicheradresse der Interrupt-Service-Routine (siehe Abb.5.13). Das ist die generelle Arbeitsweise des Interrupt-Modus 2. Wir wollen jetzt ein Beispiel untersuchen.

Bei diesem Beispiel gehen wir davon aus, daß der Interrupt bestätigt wurde und die externe Hardware 52H auf den Datenbus legt. Des weiteren nehmen wir an, daß das I-Register des Z80 mit 19H initialisiert wurde und daß die Speicherstellen 1952H und 1953H folgendes enthalten:

1952H = 38H als Daten
1953H = 2FH als Daten

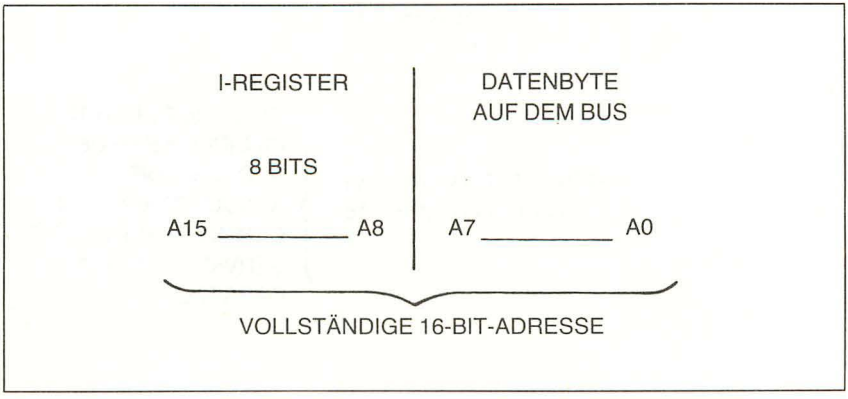


Abb. 5.12: *Das I-Register bildet zusammen mit dem externen Byte die 16-Bit-Adresse, von der das erste Byte des Interrupt-Vektors geholt wird.*

Stellen wir uns vor, der Interrupt ist gerade eingetroffen. Die 52H vom Datenbus und die 19H vom I-Register werden zusammengelegt und bilden die Speicheradresse 1952H. Das I-Register beinhaltet die oberen 8 Bit und das Datenbyte vom Bus die unteren 8 Bit. Der Z80 holt sich die Daten, die bei dieser Adresse gespeichert sind und benutzt sie als unteres Byte einer neuen Adresse. In diesem Beispiel würde das untere Byte der neuen Adresse also 38H sein. Der Z80 liest nun das nächste Byte von der Speicheradresse 1952H +1. Dieses Datenbyte wird als oberer Teil der neuen Speicheradresse benutzt. Die komplette Adresse ist also 2F38H. Der Z80 springt nun zu dieser Adresse und führt die Interrupt-Service-Routine aus.

Das mag sehr umständlich aussehen, um eine einfache Interrupt-Adresse zu bilden, doch lassen Sie uns betrachten, was diese Architektur im Systemzusammenhang bewirkt. Stellen Sie sich vor, Sie haben ein System mit verschiedenen (sagen wir 10) externen Bausteinen, die Interrupts

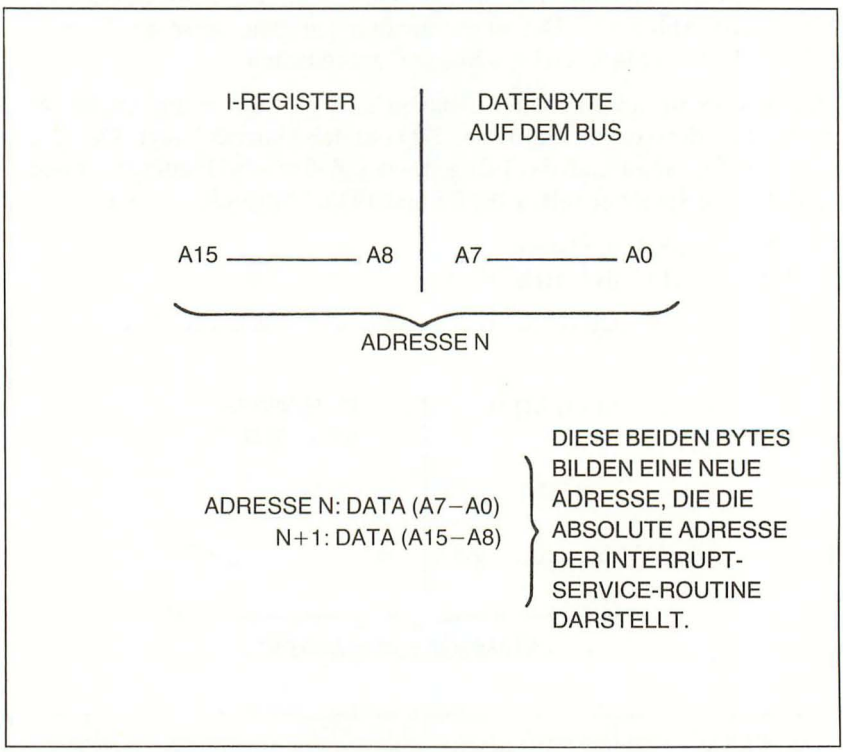


Abb. 5.13: *So wird die 16-Bit-Zeigeradresse bei einem Interrupt-Modus 2 gebildet.*

anfordern können. Jeder Baustein kann ein anderes Byte bei einer Interrupt-Bestätigung auf den Datenbus legen. Durch das Lesen dieses einfachen Bytes kann der Z80 einen Sprung zu einer belieben Speicheradresse ausführen. Des weiteren kann der Z80 für jeden dieser 10 Bausteine zu einer anderen Adresse springen. Lassen Sie uns an einem Beispiel deutlich machen, wie das in einem System aussehen kann.

Zuerst ist es notwendig, die Speicheradressen zu kennen, bei denen die Interrupt-Service-Routinen stehen. In unserem Beispiel nehmen wir folgende Adressen an:

Baustein	**Speicheradresse**
Dev 1	20F5H
Dev 2	3802H
Dev 3	1951H
Dev 4	F318H
Dev 5	E821H
Dev 6	2568H
Dev 7	1585H
Dev 8	CE80H
Dev 9	3597H
Dev 10	211EH

Wir müssen nun eine Vektortabelle mit den Speicheradressen der Service-Routinen bilden. Im vorigen Beispiel war 2F38H der Vektor, der zur Service-Routine für eine bestimmte Hardware-Komponente zeigte.

Diese Service-Routine steht also an der angegebenen Stelle im Systemspeicher. (Anm.: Die Adressen in unserem Beispiel sind aus der Luft gegriffen und nur zur Illustration gedacht.)

Als nächstes müssen wir eine Stelle im Speicher wählen, an der die Vektortabelle stehen soll. Die Tabelle kann maximal 256 Bytes lang sein. Jeder Vektor besteht aus zwei Bytes und damit ergeben sich maximal 128 Speicheradressen oder Vektoren für die Service-Routinen des Interrupts im Modus 2. Dies ist jedoch eine seltene Begrenzung im Systementwurf.

In diesem Beispiel wollen wir die Vektortabelle in den Speicherbereich 1100H-11FFH legen. Dies ist eine willkürliche Wahl (normalerweise vom Systementwickler getroffen). Wenn dieser Bereich festgelegt ist, muß entschieden werden, welches Datenbyte die externen Bausteine bei einer Interrupt-Bestätigung auf den Datenbus legen sollen. Wir wollen folgende Wahl treffen:

Baustein	Datenbyte
Dev 1	00H
Dev 2	02H
Dev 3	04H
Dev 4	06H
Dev 5	08H
Dev 6	0AH
Dev 7	0CH
Dev 8	0EH
Dev 9	10H
Dev 10	12H

Beachten Sie, daß jedes Byte eine gerade Zahl darstellt. Das kommt daher, daß jeder Vektor zwei aufeinander folgende Adressen im Speicher belegt. Mit diesen Informationen können wir unsere Vektortabelle ausfüllen. In unseren Beispiel würde das so aussehen:

Speicheradresse	Daten	Vektor
1100H	F5H	Dev 1
1101H	20H	
1102H	62H	Dev 2
1103H	38H	
1104H	51H	Dev 3
1105H	19H	
1106H	18H	Dev 4
1107H	F3H	
1108H	21H	Dev 5
1109H	E8H	
110AH	68H	Dev 6
110BH	25H	
110CH	85H	Dev 7
110DH	15H	
110EH	80H	Dev 8
110FH	CEH	
1110H	97H	Dev 9
1111H	35H	
1112H	1EH	Dev 10
1113H	21H	

Ein Z80-Programm für dieses Beispiel zeigt Abb. 5.14.

```
                ;-------------------------------------------------------
                ;
                ; Programm zum Aufsetzen des Z80 in den
                ; Interrupt-Modus 2
                ;
                ;-------------------------------------------------------

                        ORG     0

0000    ED 5E           IM      2       ; Interrupt-Modus 2
0002    3E 11           LD      A,11H   ; oberes Byte der Vektor-Tabelle
0004    ED 47           LD      I,A     ; Vektor-Register
0006    FB              EI              ; Interrupt Freigabe

                ; Es folgen die Interrupt-Service-Routinen
                ; (hier nicht gezeigt)

                ;-------------------------------------------------------
                ; Interrupt-Vektor-Tabelle (die Adressen sind
                ; als Beispiele zu verstehen)
                ;-------------------------------------------------------

                   ORG     1100H   ; Anfangsadresse der Tabelle

1100    20F5            DEFW    20F5H   ; Baustein 1
1102    3802            DEFW    3802H   ; Baustein 2
1104    1951            DEFW    1951H   ; Baustein 3
1106    F318            DEFW    0F318H  ; Baustein 4
1108    E821            DEFW    0E821H  ; Baustein 5
110A    2568            DEFW    2568H   ; Baustein 6
110C    1585            DEFW    1585H   ; Baustein 7
110E    CE80            DEFW    0CE80H  ; Baustein 8
1110    3597            DEFW    3597H   ; Baustein 9
1112    211E            DEFW    211EH   ; Baustein 10

                ; Ende der Tabelle
```

Abb. 5.14: *Einfaches Programm zum Aufsetzen des Interrupt-Modus 2 und zum Generieren der Vektortabelle.*

Mehrere Bausteine stellen Interrupt-Anforderungen

In diesem Abschnitt werden wir untersuchen, wie mehr als ein externer Baustein eine Interrupt-Anforderung an den Z80 stellen kann. Wir werden diese Informationen in späteren Kapiteln benutzen, wenn wir über Z80-Peripherie-Bausteine sprechen. Abb. 5.15 zeigt ein Blockschaltbild eines möglichen Mehrfach-Interrupt-Systems. Es gibt verschiedene Wege für die CPU zur Lösung dieses Problems. In diesem Abschnitt werden wir die Haupttechniken behandeln: Polling, priorisierte Interrupts und verkettete Prioritäten (daisy chain).

Polling

Die erste Technik wird als Polling (Abfrage) bezeichnet. Lassen Sie uns annehmen, der Z80 bekommt eine Interrupt-Anfrage. Die Interrupt-Service-Routine befiehlt der CPU, ein spezielles Byte, genannt Status-Byte,

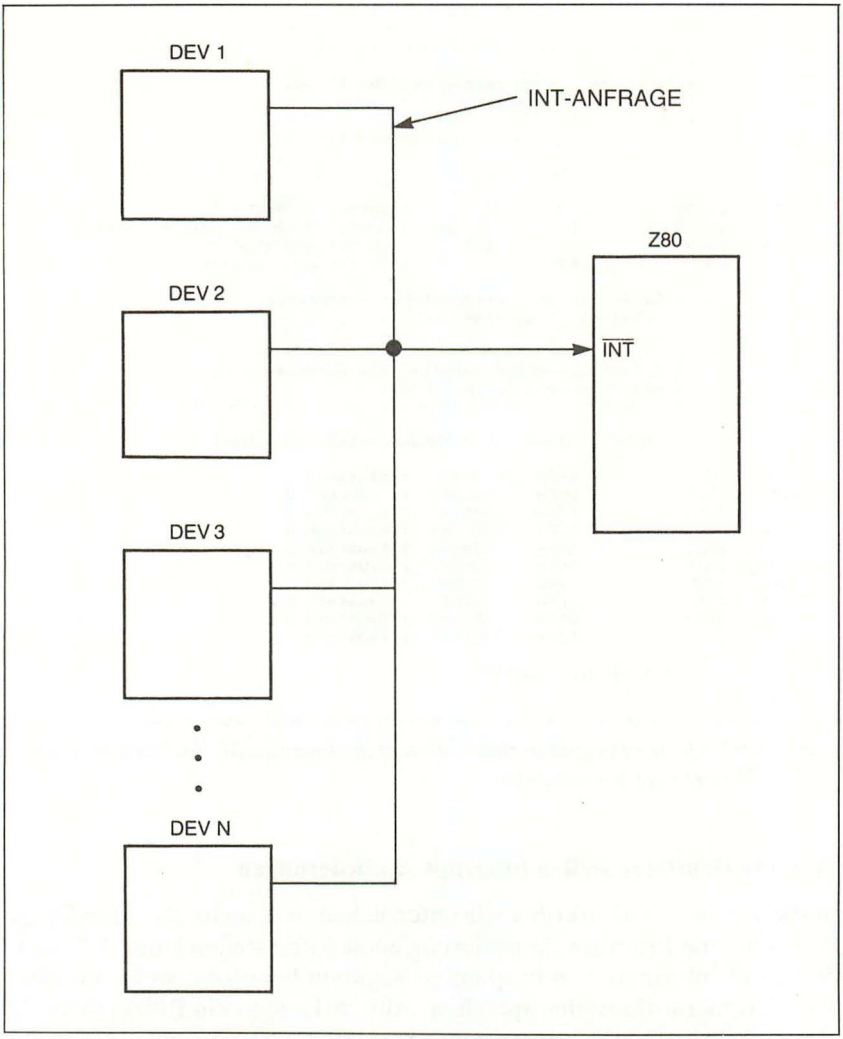


Abb. 5.15: *Blockschaltbild eines Systems, das mehrfache Interrupt-Anforderungen generieren kann.*

von jedem Peripheriebaustein zu lesen. Im allgemeinen zeigt ein Bit dieses Bytes dem Z80 an, daß der entsprechende Baustein eine Interrupt-Anforderung gestellt hat. Wenn natürlich mehrere Bausteine eine Interrupt-Anfrage eingebracht haben, muß die Service-Routine entscheiden, welcher Baustein zuerst bedient wird. Dies kann durch Polling gemacht werden.

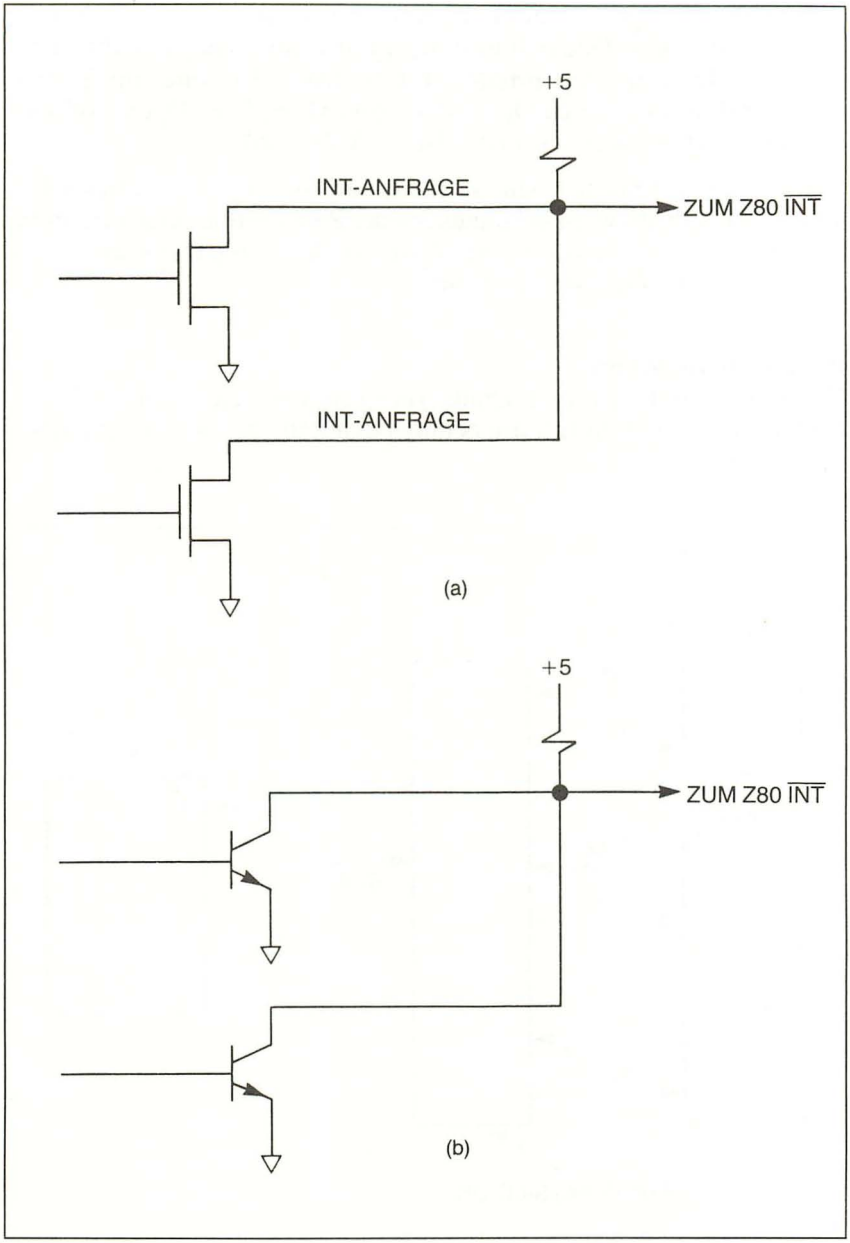


Abb. 5.16: *Schema der Ausgangsstruktur von MOS- und TTL-Bauteilen. Diese Typen werden als Open-Drain und Open-Collector bezeichnet. Die Ausgänge können miteinander verbunden werden.*

Beim Polling kann jeder der externen Bausteine zur beliebigen Zeit einen Interrupt auslösen. Dadurch ist es dem Prozessor unmöglich, direkt die Quelle des Interrupts festzustellen. Tatsächlich sind die Interrupt-Leitungen aller Bausteine durch Open-Drain (bei MOS) bzw. Open-Collector (bei TTL) miteinander verbunden (siehe Abb. 5.16).

Die Polling-Technik hat den Nachteil, daß Interrupts nicht besonders schnell gehandhabt werden können, da der Z80 alle Bausteine erst abfragen muß. Der Vorteil dieser Technik ist jedoch, daß die Hardware besonders einfach ausgelegt werden kann.

Priorisierte Interrupts

Die priorisierte Interrupt-Technik ermöglicht eine sehr effiziente Handhabung von Mehrfach-Interrupts durch den Z80. Wir wollen sehen, wie sie arbeitet.

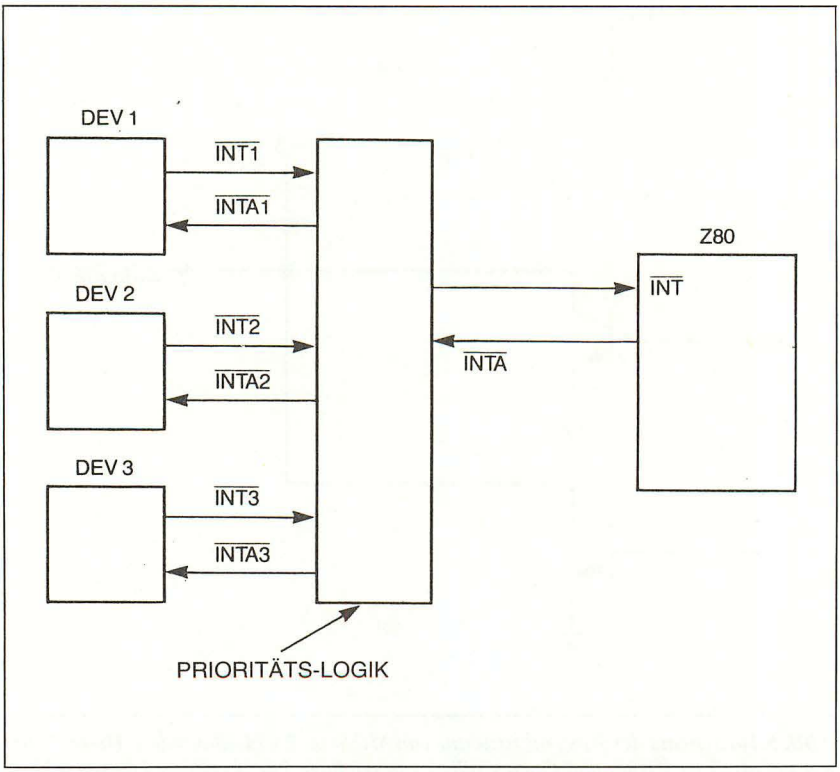


Abb. 5.17: *Eine Möglichkeit zur Priorisierung von Interrupt-Anforderungen.*

Jeder Baustein ist elektrisch darauf vorbereitet, bei einer Interrupt-Bestätigung ein entsprechendes Datenbyte auf den Datenbus zu legen. (Dieses Konzept wurde schon im Zusammenhang mit den Interrupts im Modus 0 und 2 besprochen.) In einem Mehrfach-Interrupt-System dürfen die einzelnen Bausteine jedoch nicht ohne weiteres das Datenbyte auf den Bus legen, da nur immer eine Quelle zum Bus schreiben darf. Die CPU muß also die Interrupt-Anforderung selektiv bestätigen.

Zu diesem Zweck muß die Hardware ein Prioritätsschema für die externen Bausteine bilden. Die Logik dieses Systems (im Blockschaltbild Abb. 5.17 gezeigt) ist es, daß die einzelnen Interrupt-Anforderungs-Leitungen mit einem logischen Block, der Prioritätslogik, verbunden werden. Dieser Block entscheidet, welche Interrupt-Leitung zur CPU freigegeben werden soll. Wenn die CPU den Interrupt bestätigt, wird das Bestätigungssignal von der Prioritätslogik zu dem Baustein geleitet, der den Interrupt angefordert hat. Die Wirkungsweise der Prioritätslogik ist die einer Verkehrsregelung für die Interrupt-Anfrage und -Bestätigung.

Ein Vorteil dieses Typs von Interrupt-Schema ist, daß Interrupts schnell und effizient von der CPU behandelt werden können. Ein Nachteil ist das Erfordernis zusätzlicher Hardware.

Daisy-Chain-Priorität

Eine Variation des Prioritäts-Interrupt-Schema ist die Daisy-Chain-Architektur. Unter „Daisy-Chain" vesteht man das Hintereinanderschalten von Bausteinen als Kette, wobei jeweils der Ausgang eines Bauteils an den Eingang des nächsten gelegt wird. Abb. 5.18 zeigt ein Blockschaltbild eines Daisy-Chain-Systems für den Z80. Diese Anordnung sperrt automatisch für alle Bausteine im unteren Teil der Kette die Interrupts, wenn ein Baustein oberhalb in der Kette eine Interrupt-Anforderung stellt. Wenn z.B. DEV 1 eine Anforderung erbittet, sind DEV 2 bis DEV 4 gesperrt. Wenn die CPU den Interrupt bestätigt, kann nur der aktive Baustein in der Kette reagieren.

Wenn ein gerade gesperrter Baustein in der Kette einen Interrupt anfordern will, wird dieser so lange bewahrt, bis er wieder freigegeben wurde. Dadurch geht keine Interrupt-Anforderung verloren.

Viele Z80-Peripherie-Bausteine sind für die Daisy-Chain-Technik vorbereitet. Sie haben entsprechende interne Hardware sowie physikalische Bauteile-Pins, die die Verbindung in der Daisy-Chain besonders einfach machen. In späteren Kapiteln wird das noch näher gezeigt.

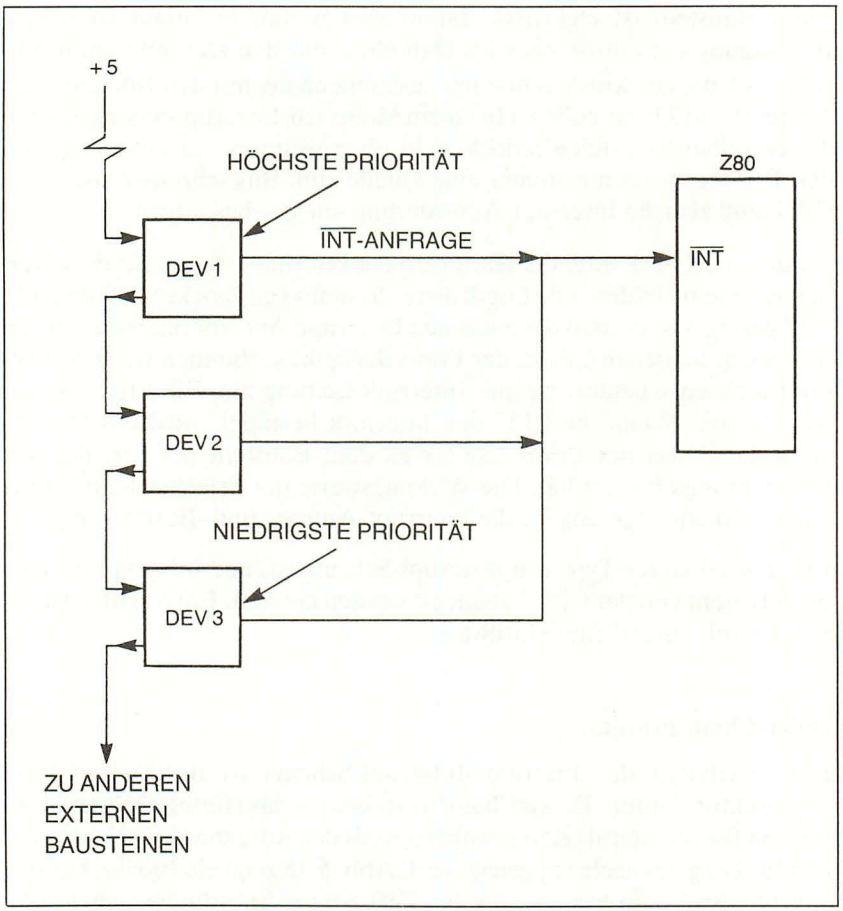


Abb. 5.18: *Benutzung der Daisy-Chain-Prioritäts-Logik in einem Z80-System.*

Zusammenfassung

In diesem Kapitel haben wir verschiedene Techniken zur Unterbrechung des Z80 kennengelernt. Wir begannen mit dem allgemeinen Konzept von Interrupts. Dann betrachteten wir jede der drei möglichen Betriebsarten beim Z80 im einzelnen. Zum Abschluß wurden noch Techniken der Mehrfach-Interrupt-Behandlung vorgestellt: Polling, Priorisierung und Daisy-Chain.

Kapitel 6

Benutzung des 8255 mit dem Z80

Einführung

Der erste hochintegrierte Peripherie-Baustein, den wir besprechen wollen, ist der programmierbare Interface-Adapter 8255. Wir wollen folgendes berücksichtigen:

1. wie er arbeitet.
2. wie er elektrisch mit dem Z80-Bus verbunden wird.
3. wie er als vielseitiges I/O-Bauteil programmiert werden kann.

Übersicht über den 8255

Der 8255 ist ein LSI-Bauteil in einem 40-poligen Dual-in-line Gehäuse. Es ermöglicht eine Vielzahl verschiedener Interface-Funktionen innerhalb eines Mikroprozessor-Systems. Im Gegensatz zu vielen anderen in diesem Buch beschriebenen Bausteinen ist der 8255 ursprünglich nicht für den Z80-Mikroprozessor entworfen worden. Er wurde zuerst von der Firma Intel für den 8080-Mikroprozessor gebaut.

Lassen Sie uns, bevor wir auf den Anschluß an den Z80 und auf die Programmierung eingehen, den 8255 allgemein beschreiben. Wenn Sie ihn auf unterster Ebene verstehen, wird es für Sie auch nicht schwer sein, seinen Einsatz in Ihren oder anderen Systementwürfen zu verstehen.

Abb. 6.1 zeigt das Blockschaltbild des 8255. Wir wollen jetzt die Funktion jeder dieser Blöcke betrachten.

Wie wir in der Zeichnung sehen, gibt es vier Blöcke, die an externe Bauteile angeschlossen werden können. Die Anschlüsse werden bezeichnet mit PA0-PA7, PB0-PB7 und PC0-PC7. Diese Signalgruppen sind logisch unterteilt in drei verschiedene I/O-Ports, Port A (PA), Port B (PB) und Port C (PC). Die vier I/O-Blöcke sind auf der anderen Seite mit dem internen Datenbus des 8255 verbunden. Über diesen Datenbus werden Dateninformationen innerhalb des 8255 ausgetauscht.

Zwei der Blöcke aus Abb.6.1, als Group-A-Control und Group-B-Control bezeichnet, definieren die Art, wie die drei I/O-Ports arbeiten sollen. (Es gibt verschiedene Betriebsarten (Modes), die durch die CPU definiert werden müssen, indem Steuerwörter zum 8255 geschrieben werden.) Beachten Sie, daß Port C noch einmal in zwei 4-Bit-Hälften unterteilt ist. Eine Hälfte ist mit der Gruppe A verbunden und die andere Hälfte mit Gruppe B. (Anm.: Den Grund dafür erfahren Sie später. Im Augenblick soll nur die Tatsache festgestellt werden.)

Die letzten logischen Blöcke in Abb.6.1 werden als Datenbus-Puffer und Lese-Schreib-Steuerlogik (read/write control logic) bezeichnet. Diese

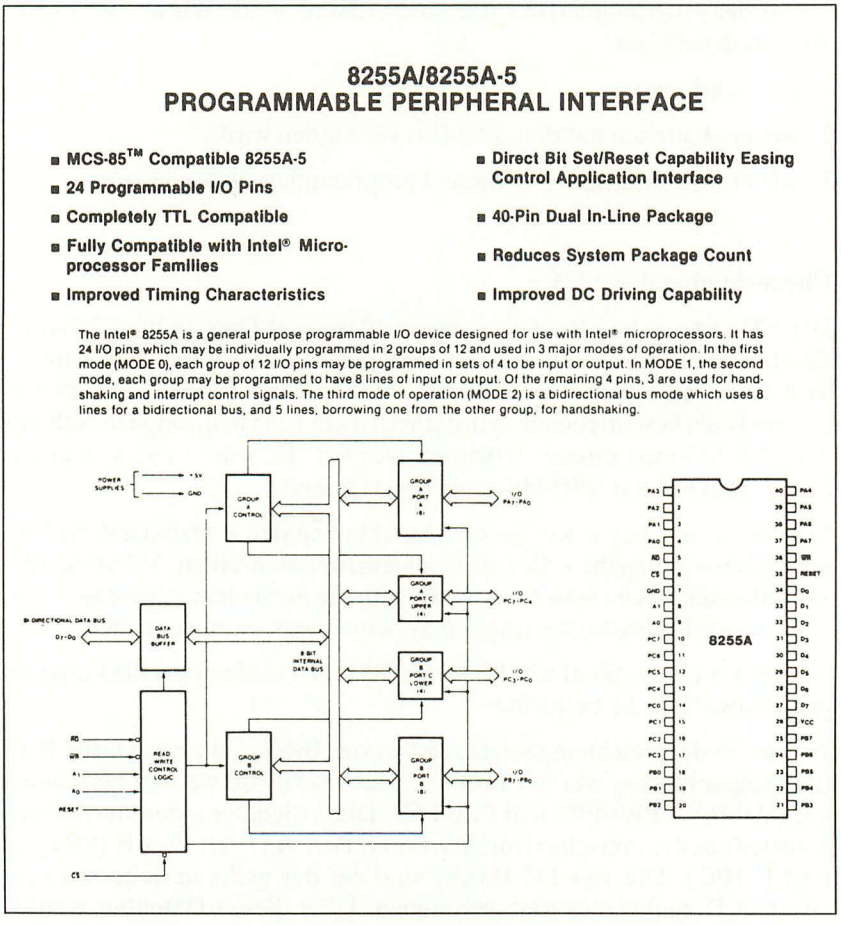


Abb. 6.1: *Blockschaltbild und Pinbelegung des 8255.*

Blöcke dienen als elektrisches Interface zwischen dem 8255 und dem Mikroprozessor. Die Datenbus-Puffer puffern die Daten-I/O-Leitungen vom und zum CPU-Datenbus. Die Lese-Schreib-Steuerlogik leitet die Daten zum richtigen internen Register mit dem richtigen Timing. Der interne Datenpfad ist abhängig von der Operation der CPU.

Pinbelegung des 8255

In diesem Abschnitt wollen wir die Funktion von jedem einzelnen Pin des 8255 besprechen. Das gibt uns nützliche Informationen, die später bei der Besprechung über die Verbindung mit dem Z80 gebraucht werden. Die Pinbelegung des 8255 wird in Abb. 6.1 gezeigt. Lassen Sie uns nun die einzelnen Anschlüsse vorstellen:

D0-D7 Dies sind die Dateneingangs- und Ausgangsleitungen des Bauteils zum Systembus. Der Datenaustausch zwischen dem 8255 und dem Mikroprozessor wird über diese acht Leitungen abgewickelt.

$\overline{\text{CS}}$ *(Chip Select)* Sobald dieser Eingang logisch 0 ist, kann der Mikroprozessor elektrisch mit dem 8255 kommunizieren.

$\overline{\text{RD}}$ *(Read)* Wenn der $\overline{\text{RD}}$-Eingang und der $\overline{\text{CS}}$-Eingang logisch 0 sind, werden die Datenausgangsleitungen des 8255 auf den System-Datenbus geschaltet.

$\overline{\text{WR}}$*(Write)* Wenn der $\overline{\text{WR}}$-Eingang und der $\overline{\text{CS}}$-Eingang logisch 0 sind, werden die Daten des System-Datenbusses in ein internes Register des 8255 geschrieben.

A0-A1 Die logische Kombination dieser zwei Adreßeingänge bestimmt das interne Register, das bei Lese/Schreiboperationen angesprochen wird.

$\overline{\text{RESET}}$ Wenn dieser Eingang auf logisch 1 gebracht wird, werden alle internen Register des 8255 gelöscht und die Port-Leitungen als Eingänge geschaltet.

PA0-PA7 Dies sind die acht I/O-Signalleitungen des Port A zum Anschluß an Peripheriegeräte.

PB0-PB7 Dies sind die acht I/O-Signalleitungen des Port B.

PC0-PC7 Dies sind die acht I/O-Signalleitungen des Port C. Sie können auch in zwei 4-Bit-Gruppen aufgeteilt werden, wobei sie als Steuerleitungen für Port A und Port B dienen.

Jetzt, da wir die Bedeutung der einzelnen Pins kennen, wollen wir das Chip an den Z80-Systembus anschließen.

Verbindung des 8255 mit der Z80-CPU

Lassen Sie uns jetzt den 8255 als Port-Bauteil in einem Z80-System einsetzen. Die Kommunikation mit der Z80-CPU soll über I/O-Operationen abgewickelt werden. Der 8255 hat zwei Adreßeingänge (A0 und A1) und belegt somit vier verschiedene I/O-Adressen. Im Normalfall werden die beiden Adreßeingänge mit den Adreß-Ausgangsleitungen A0 und A1 des Z80 verbunden.

Die $\overline{CS}$-Eingangsleitung des 8255 wird benutzt, um das Bauteil logisch im I/O-Adreßraum einzufügen. Als Beispiel wollen wir hierfür die Adressen 10H, 11H, 12H und 13H vorsehen. Dies kann logisch und elektrisch durch Verwendung der Schaltung in Abb. 6.2 realisiert werden.

Die Decodierlogik wurde dabei so gewählt, daß Sie bis zu 8 unabhängig voneinander arbeitende I/O-Bausteine mit dieser Schaltung adressieren, bzw. auswählen können. Jeder Ausgang (Y) belegt hierbei 4 I/O-Adressen (s. Abb. 6.2), es kann also jeder Peripherie-Baustein, der für die Mikroprozessorserie 8080/8085 entwickelt wurde, ohne zusätzliche Deco-

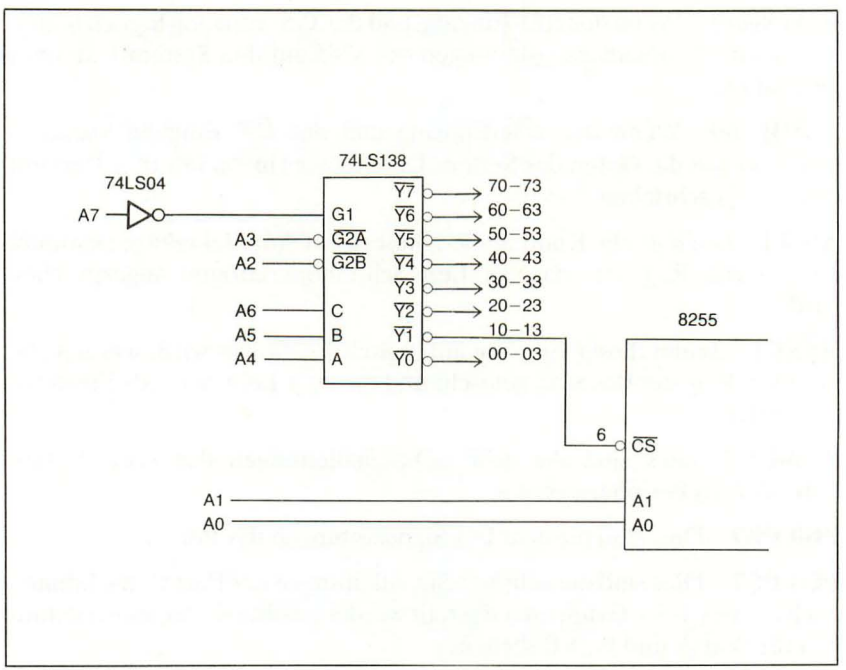


Abb. 6.2: *Logikplan mit der Decodierung der Adreßleitungen A0-A7 zur Bildung des $\overline{CS}$-Signals.*

dierlogik angeschlossen werden. Sollen mehr als 8 I/O-Bausteine an ein Mikrocomputersystem angeschlossen werden, dann benutzen Sie die gleiche Decodierlogik. Die Leitung A 7 wird in diesem Fall direkt mit dem Eingang G 1 des Bausteins 74 LS 138 verbunden. In diesem Fall müssen Sie zu den Adressen an den einzelnen Ausgängen (Y) den Wert 80 H addieren, weil die Decodierung nur dann erfolgt, wenn die Leitung A 7 = 1 ist.

Wie wir aus Kapitel 3 wissen, benötigen I/O-Operationen eine Decodierung der unteren acht Adreßleitungen des Z80, A7-A0. In Abb. 6.2 wird das $\overline{CS}$-Signal (Y1) aktiv, wenn diese Leitungen den logischen Zustand 000100XX haben. (Die unteren zwei Adreßbits werden nicht mit decodiert, sondern dienen im 8255 zur Anwahl des internen Registers.) Diese I/O-Architektur wird manchmal als „Device Port I/O" bezeichnet. Das bedeutet, daß das Bauteil selber mehrere I/O-Adressen (hier sind es vier) belegt.

Als nächstes müssen wir den $\overline{RD}$- und den $\overline{WR}$-Eingang mit den $\overline{IOR}$- und $\overline{IOW}$-Steuerleitungen des Z80-Systems verbinden. Eine direkte Verbindung mit den $\overline{RD}$- und $\overline{WR}$-Ausgängen des Z80 ist hier nicht möglich, da diese Leitungen auch bei Speicheroperationen aktiv sind. Abb. 6.3 zeigt die Erzeugung des $\overline{IOR}$- und $\overline{IOW}$-Signals und ihre Verbindung mit dem 8255.

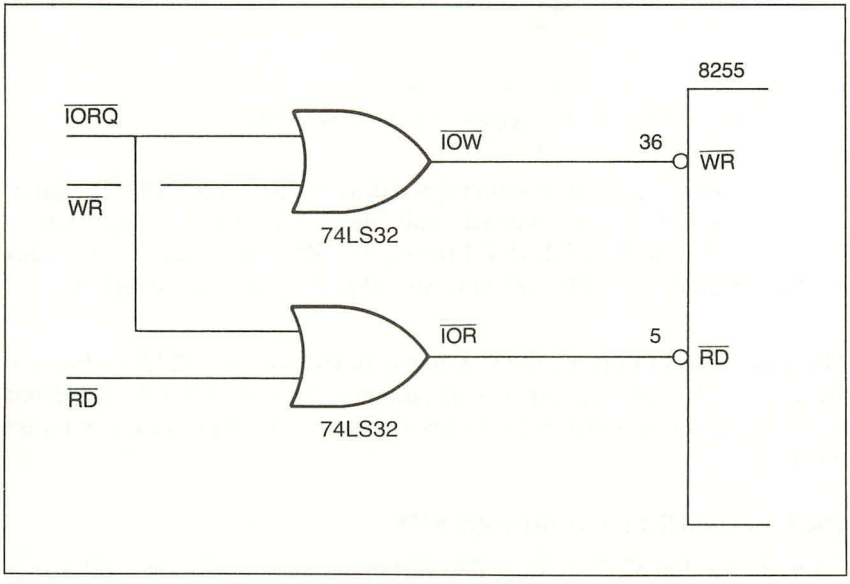


Abb. 6.3: *Beschaltung des $\overline{RD}$- und des $\overline{WR}$-Eingangs des 8255.*

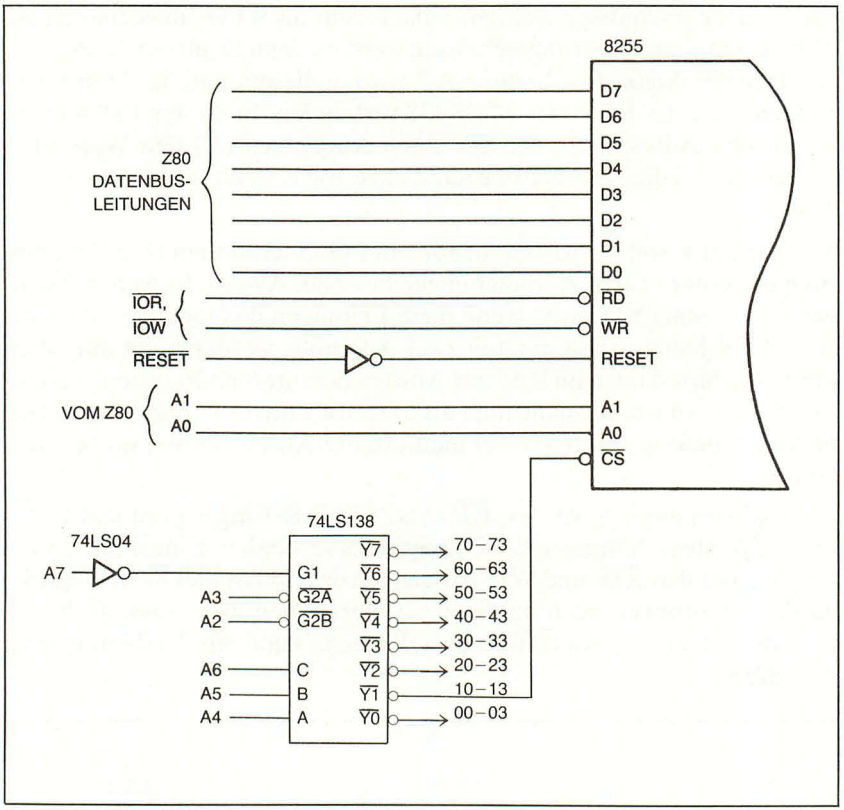


Abb. 6.4: *Kompletter Verbindungsplan zwischen Z80 und 8255.*

Das letzte zu beschaltende Steuersignal des 8255 ist der RESET-Eingang. Ein wichtiger Punkt hierbei ist, daß dieser Eingang im logischen 1-Zustand aktiv ist. Der RESET-Eingang des Z80 ist logisch 0 aktiv. Das RESET-Signal vom Z80 muß zum Anschluß des 8255 also invertiert werden.

Die Datenbusleitungen D0-D7 können direkt mit den 8255 verbunden werden. Abb. 6.4 zeigt einen vollständigen Anschlußplan des 8255 mit dem Z80. Wir gehen dabei davon aus, daß keine Datenpufferung benötigt wird.

Die Lese- und Schreibregister des 8255

Jetzt, da wir den 8255 mit dem Z80 verbunden haben, wollen wir darüber sprechen, wie er programmiert werden muß. Wir fangen dabei mit der

Besprechung der vier internen Register an (genauer: vier Lese- und vier Schreibregister). In unserem Beispiel sind die Adressen dieser Register 10H, 11H, 12H und 13H. Ihre Definition ist folgende:

Bauteilanschlüsse				**Registername**
$\overline{RD}$	$\overline{WR}$	A1	A0	
1	0	0	0	Datenport A schreiben
0	1	0	0	Datenport A lesen
1	0	0	1	Datenport B schreiben
0	1	0	1	Datenport A lesen
1	0	1	0	Datenport C schreiben
0	1	1	0	Datenport C lesen
1	0	1	1	Steuerwort schreiben
0	1	1	1	nicht erlaubt

Die Funktion der Register 0 bis 2 wird durch das in das Steuerregister 3 geschriebene Steuerwort definiert. Abb. 6.5 zeigt die Bit-Definition für das Steuerwort. Lassen Sie uns nun einige wichtige Operationsarten an Beispielen untersuchen.

Modus 0: Einfaches I/O-Register

Um den 8255 in den Modus 0 zu setzen, muß zuerst ein Steuerwort in das Steuerregister geschrieben werden. Dieses Steuerwort definiert, wie die einzelnen Register des 8255 verwendet werden sollen. Die Bits zur Programmierung der Standard-I/O-Funktion sollten wie folgt gesetzt sein:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0

Wenn wir Abb. 6.5 verwenden, können wir folgendes sehen:

- Bit D7 definiert das Wort als Steuerwort.
- Bit D6 und D5 definieren den Arbeitsmodus für Port A. Dies ist der Modus 0.
- Bit D4 = 0 schaltet Port A als Ausgang.
- Bit D3 = 0 definiert die oberen 4 Bits von Port C als Ausgänge.
- Bit D2 setzt den Arbeitsmodus für Port B (hier Modus 0).
- Bit D1 = 0 definiert Port B als Ausgabeport.
- Bit D0 = 0 definiert die unteren vier Bits von Port C als Ausgänge.

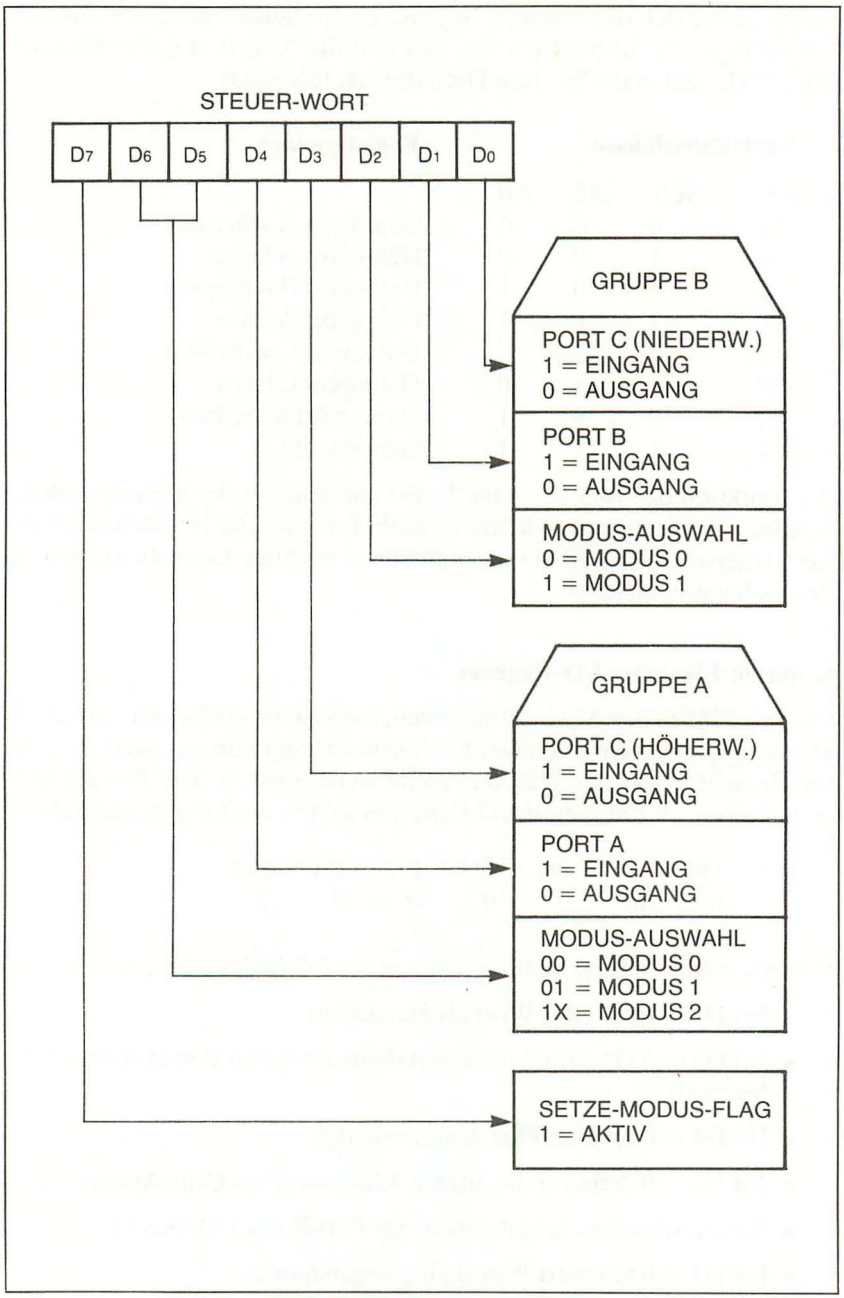


Abb. 6.5: *Bit-Definition für das Steuerregister des 8255.*

Das zum 8255 geschriebene Steuerwort schaltet alle drei Ports (A, B und C) als Ausgabeports. Dadurch erhält der 8255 vierundzwanzig einzeln ansteuerbare Ausgangsleitungen zur Ansteuerung externer Geräte. Die Z80-Anweisungen zur Programmierung des 8255 in den Modus 0 sind folgende:

```
LD    A,80H       ; Setze das Steuerwort
OUT   (13H),A     ; Gib das Steuerwort zum 8255 aus
```

Wenn der 8255 einmal mit diesen beiden Befehlen programmiert wurde, können wir mit einer einfachen OUT-Anweisung Daten in jedes Ausgabeport schreiben. Wir wollen z.B. 23H zum Port A, 41H zum Port B und 73H zum Port C schreiben. Die hierfür benötigte Anweisungsfolge ist folgende:

```
LD    A,23H       ; Daten für Port A
OUT   (10H),A     ; Ausgabe der Daten zum 8255
LD    A,41H       ; Daten für Port B
OUT   (11H),A     ; Ausgabe der Daten zum 8255
LD    A,73H       ; Daten für Port C
OUT   (12H),A     ; Ausgabe der Daten zum 8255
```

Nach Ausführung dieser Anweisungen liegen die programmierten Daten an den Portleitungen der Ports A, B und C. Dies ist eine einfache Methode zur Benutzung dreier Ausgabeports in einem Baustein.

Es ist auch möglich, die Ports als Kombination von Ein- und Ausgängen zu programmieren. Port A und C könnten z.B. als Ausgabeports und Port B als Eingabeport programmiert werden. Mit Abb. 6.5 als Referenz müßte das Steuerwort dann so aussehen:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	1	0

Nachdem dieses Steuerwort in das Register 3 des 8255 geschrieben wurde, arbeitet das Bauteil wie in Abb. 6.6 gezeigt. Wir können einen einfachen IN-Befehl benutzen, um Daten vom Port B zu lesen.

```
IN    A,(11H)     ; Lies Daten vom Port B
```

Wenn der 8255 in der richtigen Weise programmiert wurde, ist es sehr einfach, Daten zu einem beliebigen Port zu schreiben oder aus einem beliebigen Port zu lesen. Dies waren nur zwei der möglichen Kombinationen.

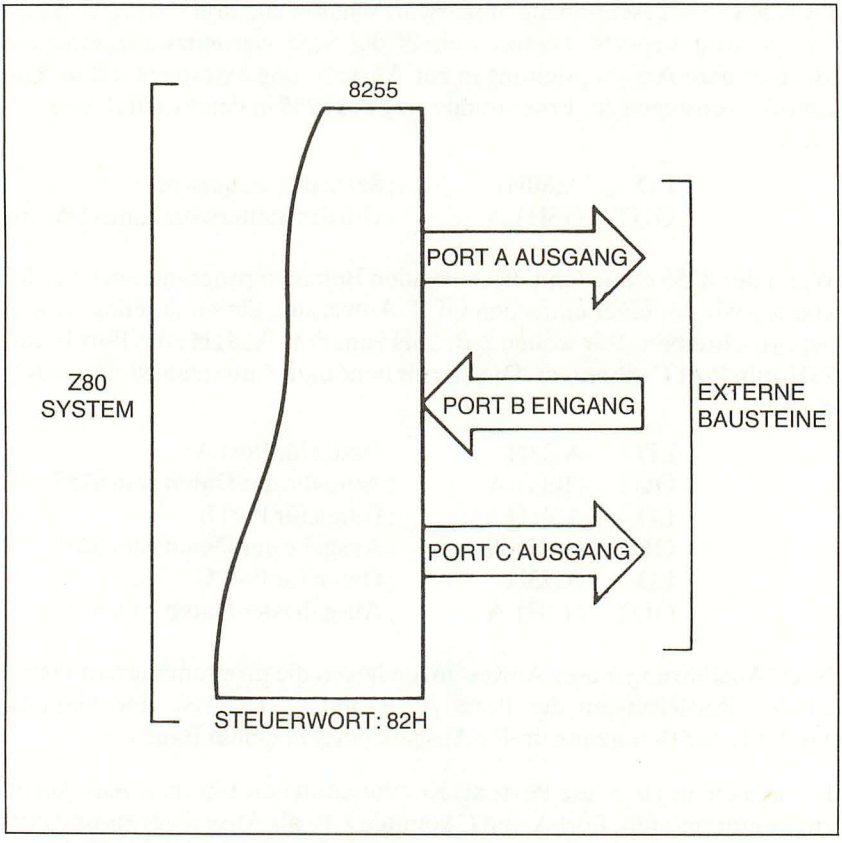


Abb. 6.6: *Konfiguration des 8255 nach dem Schreiben des Steuerworts.*

Abb. 6.7 zeigt eine Aufstellung aller möglichen Kombinationen zur Programmierung des 8255 im Modus 0.

Ein Beispiel für den Modus 0

Als Beispiel für die Benutzung des Modus 0 des 8255 wollen wir hier ein Tastatur-Interface vorstellen. Wir betrachten sowohl die Hardware als auch die Software. Die Tastatur ist als 4 * 4 Matrix einpoliger Schließkontakte aufgebaut. Der 8255 dient als Interface zwischen diesen Schaltern und dem Z80. Die Hardware für dieses Beispiel wird in Abb.6.8 gezeigt. Hier nun eine Übersicht über die Funktionsweise der Schaltung und des Programms:

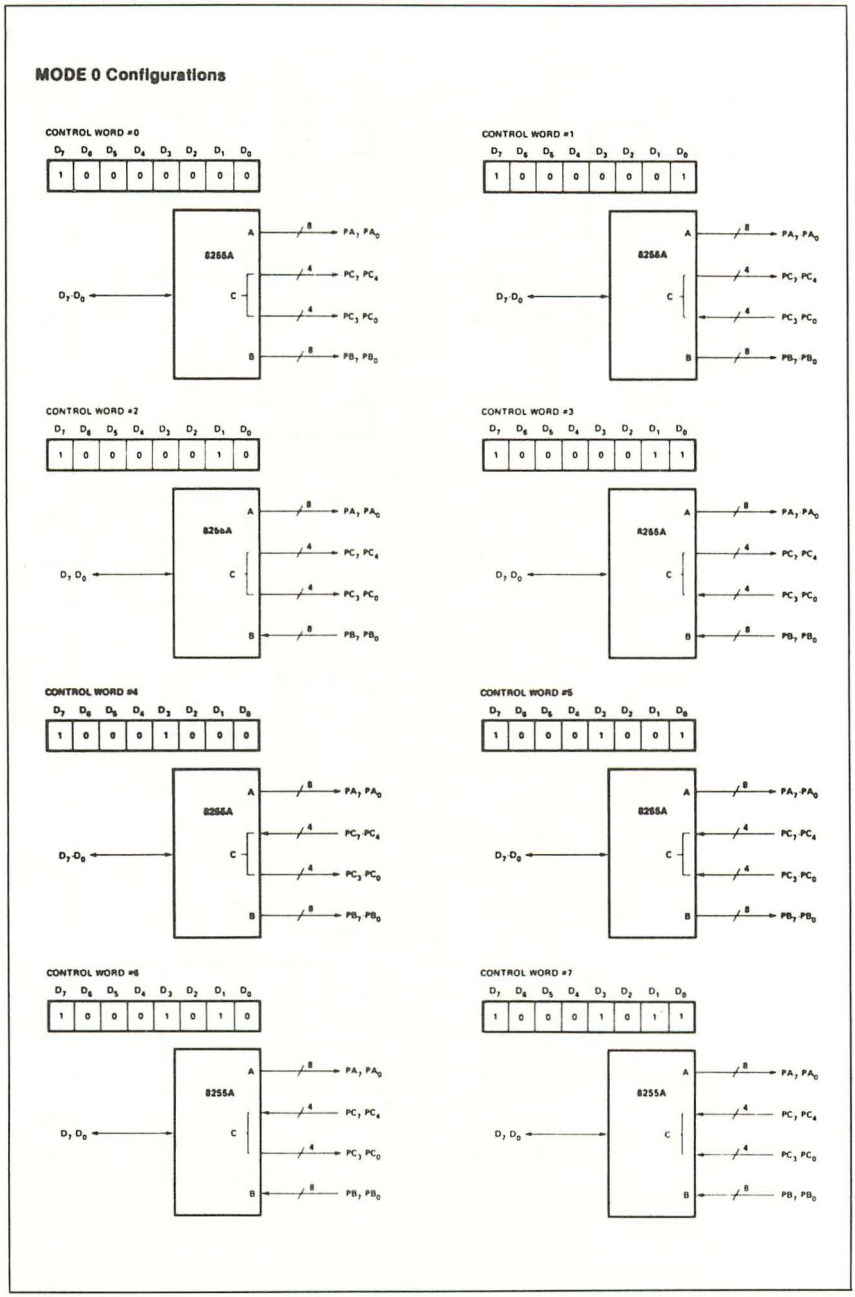


Abb. 6.7: *Mögliche Konfigurationen im Modus 0.*

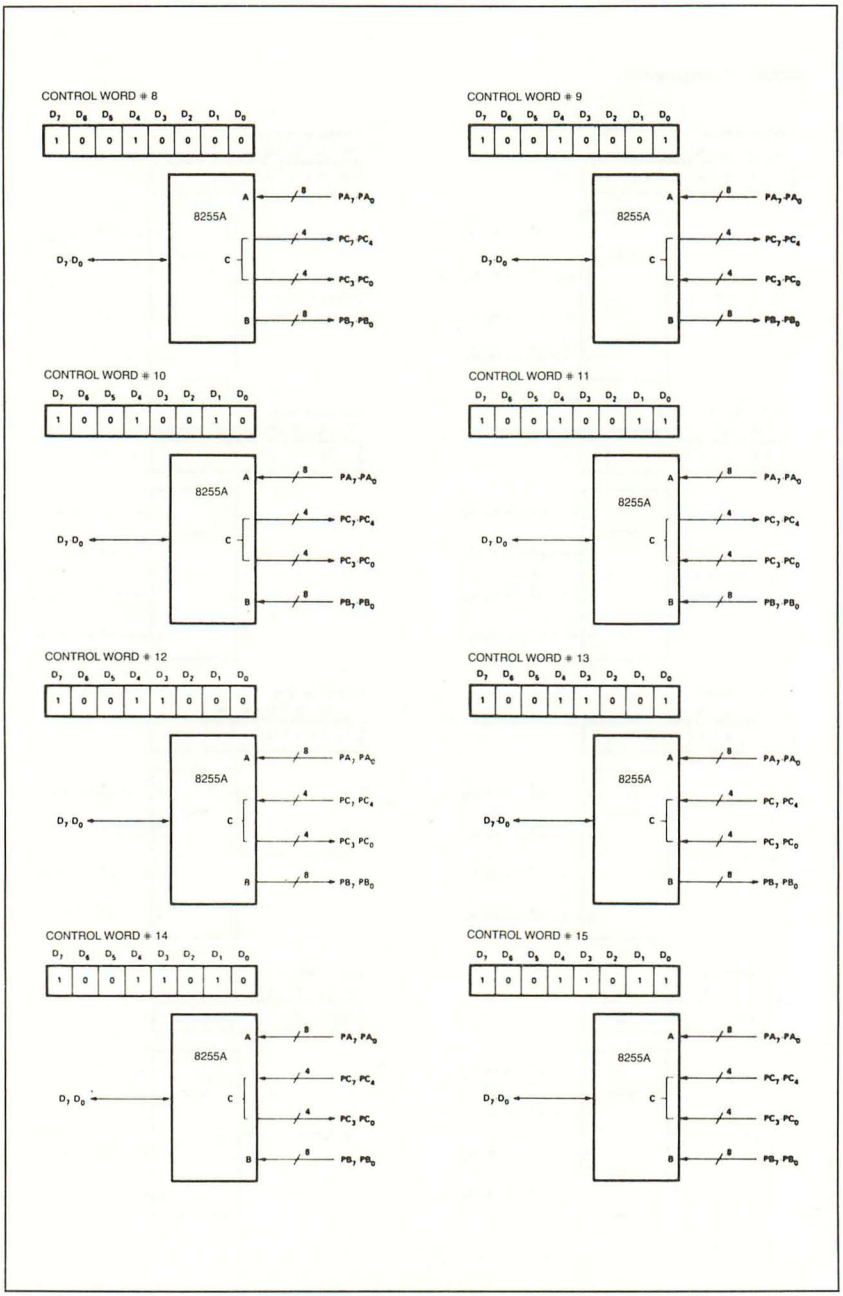


Abb. 6.7: *Mögliche Konfigurationen im Modus 0 (Fortsetzung).*

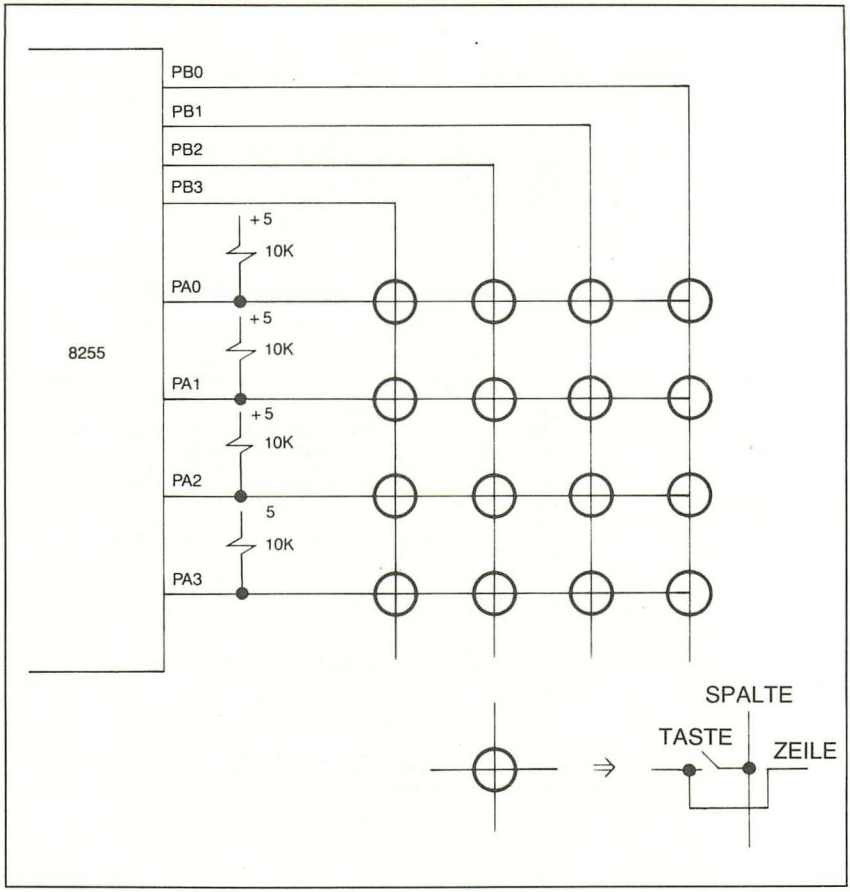


Abb. 6.8: *Verbindung des 8255 mit einer 4 * 4 Tastatur-Matrix.*

Port B wird als Ausgabeport programmiert; seine Leitungen sind mit den Spalten der Tastatur verbunden. Port A dient als Eingangsport; seine Leitungen sind mit den Tastaturzeilen verbunden. Diese Zeileneingänge werden über 10 KOhm Widerstände auf +5V hochgezogen (pull-up). Die Spaltenleitungen werden vom Programm einzeln auf logisch 0 gesetzt. Jedesmal, wenn eine Leitung logisch 0 gesetzt wurde, wird das Eingabeport A gelesen. Wenn dabei irgendein Bit logisch 0 ist, muß eine Taste gedrückt sein. Die Position der gedrückten Taste kann ermittelt werden, wenn man berücksichtigt, welche Eingangsleitung und welche Ausgangsleitung gerade logisch 0 ist. Abb.6.9 zeigt ein in dieser Weise arbeitendes Programm.

```
                ;
                ;-------------------------------------------------
                ; Programm zur Auswertung der Tastatureingabe
                ;-------------------------------------------------
                ;
0010                PORTA   EQU     10H     ; PIO Eingabe-Port
0011                PORTB   EQU     11H     ; PIO Ausgabe-Port
0012                PORTC   EQU     12H     ; PIO Ausgabe-Port
0013                CONP    EQU     13H     ; PIO Steuer-Register
0099                CONWD   EQU     99H     ; Steuerwort

                            ORG     1800H

1800    3E 99               LD      A,CONWD
1802    D3 13               OUT     (CONP),A    ; Aufsetzen des 8255

1804    0E 11               LD      C,PORTB     ; Adresse von Port B

1806    06 FE       COL:    LD      B,0FEH      ; Spalte 0 aktiv
1808    ED 41       COL1:   OUT     (C),B       ; Ansprechen der aktiven Spalte
180A    DB 10               IN      A,(PORTA)   ; Einlesen der Zeile
180C    2F                  CPL                 ; invertieren
180D    E6 0F               AND     0FH         ; unbenutzte Bits ausblenden
180F    FE 00               CP      0           ; irgendwelche Bits = 1 ?
1811    C2 181F             JP      NZ,KEYIN    ; ja, Taste wurde betätigt
1814    CB 00               RLC     B           ; verschiebe aktive Spalte
1816    78                  LD      A,B
1817    FE EF               CP      0EFH        ; Durchgang beendet ?
1819    C2 1808             JP      NZ,COL1     ; nein, teste nächste Spalte
181C    C3 1806             JP      COL

                ; Zur Feststellung der betätigten Taste wird
                ; Zeile und Spalte benötigt

181F    4F          KEYIN:  LD      C,A         ; Zeilen Position
1820    7C                  LD      A,H         ; Spalten Position
1821    2F                  CPL                 ; invertieren
1822    E6 0F               AND     0FH         ; unbenutzte Bits auf 0
1824    C9                  RET

                ; Beim Rücksprung gibt das gesetzte Bit vom
                ; A-Register die Spaltenposition an und das
                ; gesetzte Bit vom C-Register die Zeilenposition

                            END
```

Abb. 6.9: *Programm zur Abfrage der Tastatur und zur Ermittlung der gedrückten Taste.*

Obwohl in diesem Beispiel nur eine einfache 4 * 4 Matrix bedient wird, kann es auf einfache Weise auf eine fast beliebige Größe erweitert werden. Die Benutzung des 8255 ist eine recht einfache Möglichkeit zum Aufbau solcher Interfaces.

Arbeitsmodus 1 des 8255

Ein anderer Arbeitsmodus des 8255 ist sein Einsatz als I/O-Baustein in einer Handshake-Umgebung. Port A und B sind dabei weiterhin 8 Bit Ports, während die oberen 4 Bits des Port C als Handshake-Leitungen für Port A und die unteren 4 Bits als die Handshake-Leitungen für Port B dienen.

Die grundsätzliche Idee des I/O-Transfer mittels Handshake ist folgende: Das Peripheriegerät informiert den 8255 darüber, daß es bereit ist, Daten zu senden oder zu empfangen. Die Handshake-Leitungen übermitteln diese Informationen (siehe Abb. 6.10).

Im Fall (a) werden Daten vom 8255 zum Peripheriegerät gesendet. Bevor die Daten zum 8255 geschrieben werden, muß das Peripheriegerät prüfen, ob das Eingangspuffer-voll-Flag gesetzt ist. Ist es gesetzt, sind die im Eingangsregister des 8255 stehenden Daten noch nicht vom Prozessor

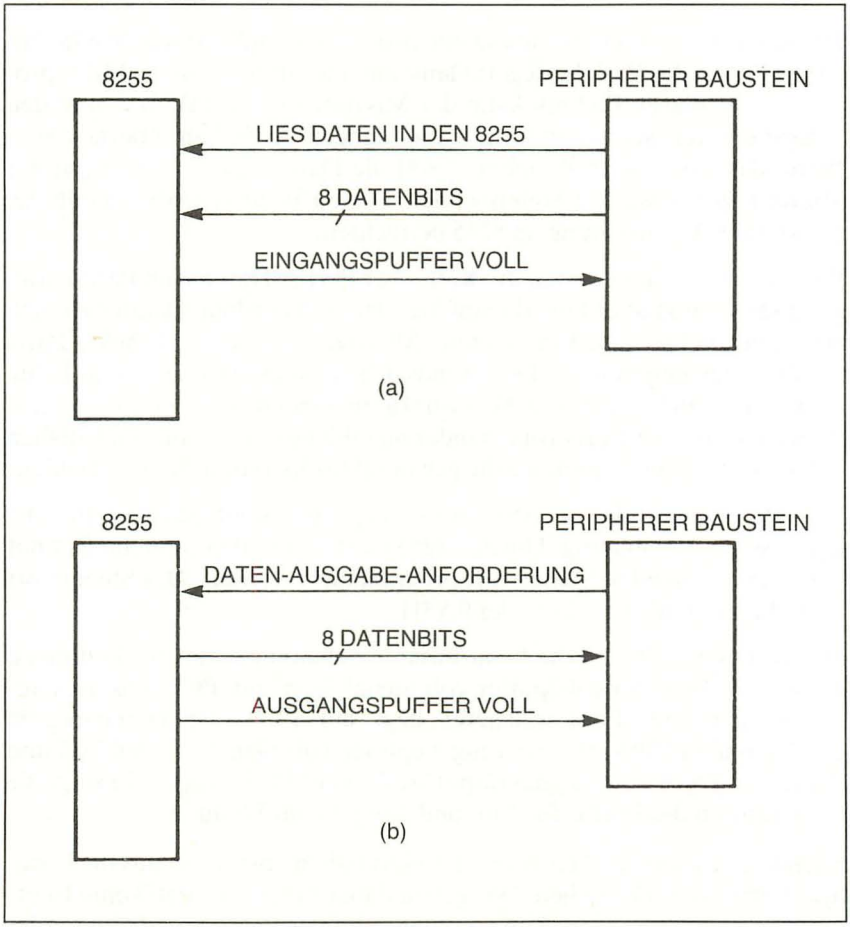


Abb. 6.10: *Benutzung der Handshake-Leitungen zwischen dem 8255 und einem Peripheriegerät.*

gelesen worden. Ist das Flag nicht gesetzt, schreibt das Peripheriegerät neue Daten zum 8255, und das Eingangspuffer-voll-Flag wird gesetzt. Liest der Z80 die Daten, wird das Flag zurückgesetzt.

Im Fall (b) der Abb.6.10 sendet der 8255 Daten zum Peripheriegerät. Bevor er die Daten sendet, muß er das Ausgabepuffer-voll-Flag prüfen. Dieses Flag beinhaltet die Information für das Peripheriegerät, daß der 8255 sendebereit ist. Wenn die Daten ausgelesen werden (strobe data out), wird das Flag zurückgesetzt und somit angezeigt, daß keine Daten mehr zur Verfügung stehen. Der Prozessor kann jetzt neue Daten für die Ausgabe zum 8255 schreiben.

Die Handshake-Technik zum Datenaustausch ist äußerst nützlich in Fällen, in denen das Peripheriegerät langsamer ist als der System-Mikroprozessor. Bei dieser Technik kann der Mikroprozessor z.B. Daten in den Ausgabepuffer schreiben und dann andere Aufgaben übernehmen. Wenn das langsamere Peripheriegerät die Daten gelesen hat, kann der Mikroprozessor weitere Daten senden. Lassen Sie uns nun die Details der Handshake-Verarbeitung im 8255 betrachten.

Wir wollen annehmen, daß der 8255 über Port A Daten zum Peripheriegerät sendet und über Port B empfängt. Diese Anordnung kann natürlich auch umgekehrt sein. Eine andere Alternative ist die, daß beide Ports senden oder empfangen. Die grundsätzliche Idee ist dabei, daß beide Ports fähig sind Daten mit Handshake zu senden oder zu empfangen. Weiterhin sind die Ports voneinander unabhängig. Sie brauchen lediglich das Steuerwort zu ändern, um die gewünschte Operationsart anzuwählen.

Das Blockschaltbild in Abb. 6.11 zeigt die Konfiguration für die gewünschte Anwendung. Um den 8255 hierfür vorzubereiten, muß er mit dem entsprechenden Steuerwort programmiert werden. Das Steuerwort ist in diesem Fall 10100110 oder 0A3H.

Die Datenausgabe geschieht in unserer Schaltung über die Leitungen PA0-PA7. Das Ausgabepuffer-voll-Signal liegt auf PC7, das Bestätigungssignal vom Peripheriegerät liegt auf PC6, die Dateneingabe geschieht über PB0-PB7, das Eingabepuffer-voll-Signal liegt auf PC1 und das Datenübernahme-Signal (input strobe) auf PC2. Abb. 6.12 zeigt die Pindefinition des Port C für Ein- und Ausgabe im Modus 1.

Nachdem wir nun die Hardware definiert haben, lassen Sie uns die benötigte Software untersuchen. Wir gehen dabei davon aus, daß keine Interrupts verwendet werden. Um zu sehen, ob Daten gelesen oder gesendet werden sollen, fragt der Z80 die Statusleitungen des 8255 ab (polling). Wir gehen weiter davon aus, daß das Peripheriegerät elektrisch fähig ist,

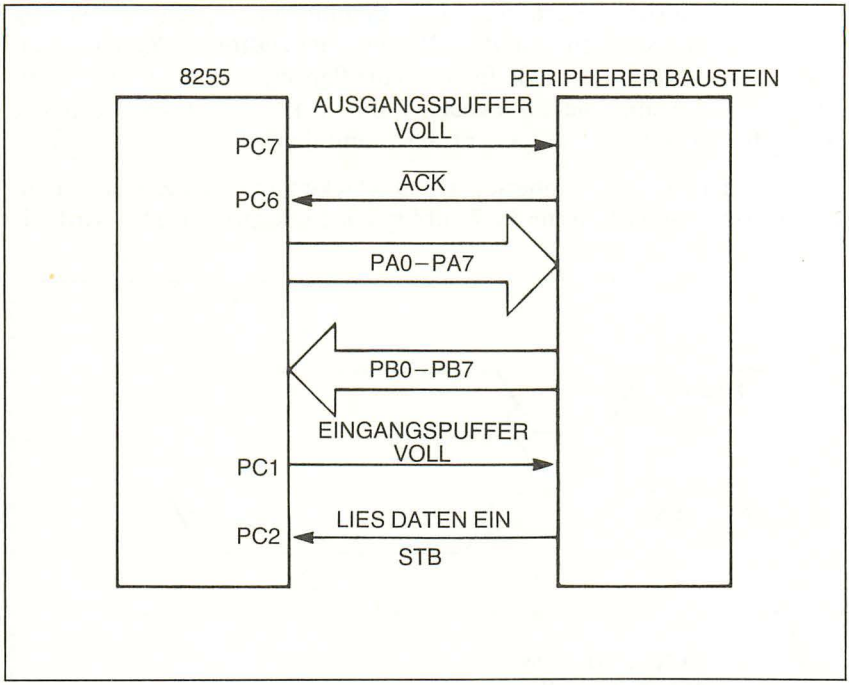


Abb. 6.11: *8255-Konfiguration mit Datenausgabe über Port A und Eingabe über Port B.*

	AUS	EIN
PC0	$INTR_B$	$INTR_B$
PC1	IBF_B	$\overline{OBF}_B$
PC2	$\overline{STB}_B$	$\overline{ACK}_B$
PC3	$INTR_A$	$INTR_A$
PC4	$\overline{STB}_A$	I/O
PC5	IBF_A	I/O
PC6	I/O	$\overline{ACK}_A$
PC7	I/O	$\overline{OBF}_A$

Abb. 6.12: *Pindefinition für Port C im Modus 1.*

Daten im Handshake-Verfahren zu senden und zu empfangen. Wichtig dabei ist es zu bedenken, daß der 8255 nur eine Hälfte der Kommunikation abwickelt. Die andere Hälfte muß vom Peripheriegerät in einer kompatiblen Weise abgewickelt werden. Abb. 6.13a und 6.13b zeigt das Timing für einen typischen Datentransfer mit dem 8255.

Um ein Zeichen zum Peripheriegerät zu schicken, muß der Z80 zuerst die Ausgabepuffer-voll-Leitung PC7 auf logisch 1 überprüfen. Das wird mit

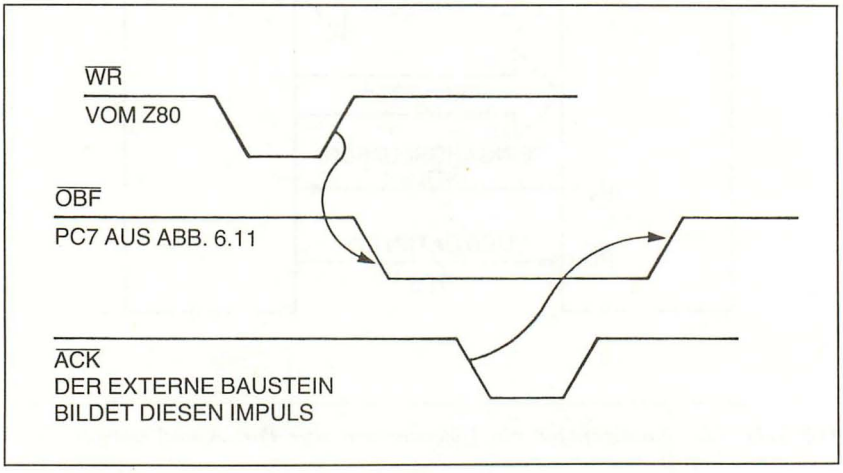


Abb. 6.13a: *Timing bei der Ausgabe im Modus 1.*

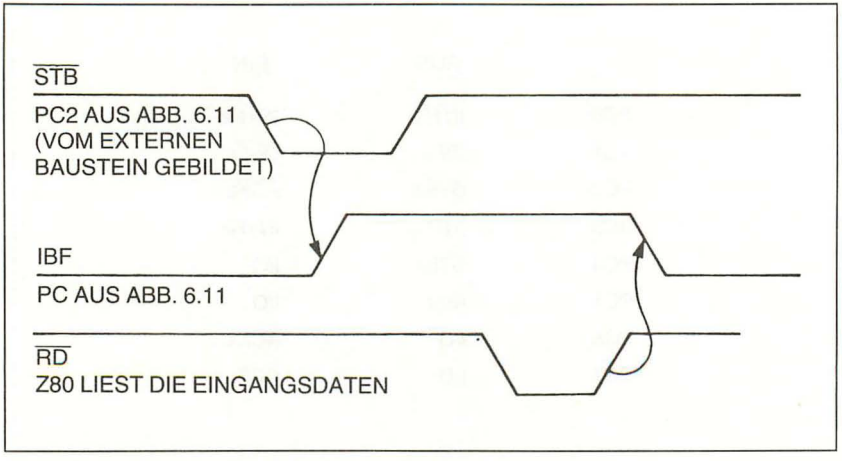


Abb. 6.13b: *Timing bei der Eingabe im Modus 1.*

einer einfachen Eingabeanweisung vom Port C gemacht. Wenn das Bit D7 vom Port C logisch 1 ist, sind die Daten aus dem Ausgabepuffer vom Peripheriegerät abgeholt worden. Wenn D7 logisch 0 ist, sollte der Z80 keine weiteren Daten schicken. Wenn das Bit logisch 1 wird, können weitere Daten zum Port A geschrieben werden. PC7 wird logisch 0 und meldet dem Peripheriegerät, daß der Ausgabepuffer voll ist. Das Peripheriegerät bestätigt den Datenempfang, indem die Leitung PC6 einen logischen 0-Impuls bekommt. Damit wird auch PC7 wieder logisch 1. Abb. 6.14 zeigt das Flußdiagramm für die Ausgabeoperation mit Handshake. Das Z80-Programm hierfür wird in Abb. 6.15 gezeigt.

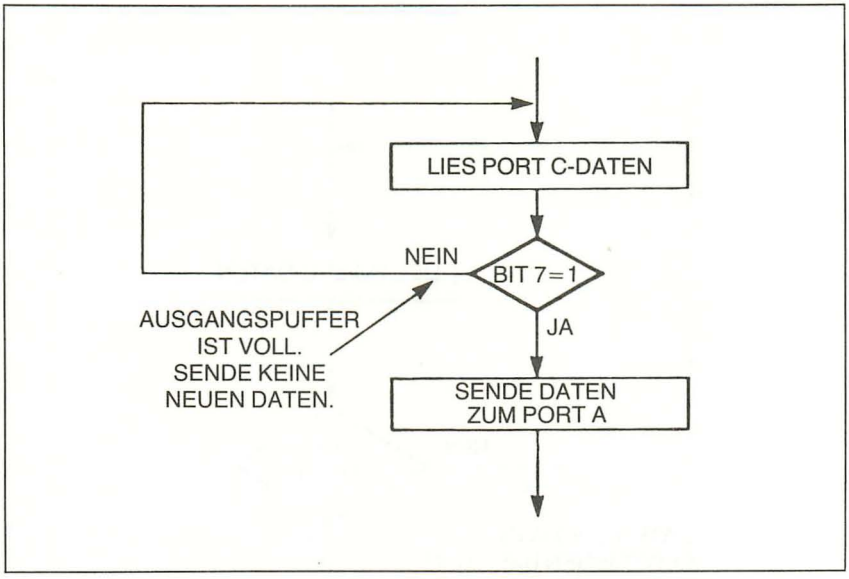


Abb. 6.14: *Flußdiagramm für eine Port-Schreiboperation mit Handshake.*

```
24E8                ODATA   EQU     24E8H           ; Adresse der Ausgabedaten
                            ORG     1800H

1800    DB 12       BACK:   IN      A,(12H)         ; lese Port C
1802    CB 7F               BIT     7,A             ; Bit 7 = 1 ?
1804    CA 1800             JP      Z,BACK          ; nein, wiederhole
1807    3A 24E8             LD      A,(ODATA)       ; Ausgabedaten
180A    D3 10               OUT     (10H),A         ; schreibe zum Port A
180C    C9                  RET
                            END
```

Abb. 6.15: *Z80-Programm zur Realisierung des Flußdiagramms aus Abb. 6.14.*

Wir wollen jetzt sehen, wie der Z80 über den 8255 Daten von einem Peripheriegerät empfängt. Als erste Operation überprüft der Z80 die Eingangspuffer-voll-Leitung PC1. Ist diese Leitung logisch 1, dann bedeutet das, daß das Peripheriegerät Daten in das Eingangsregister vom Port B geladen hat. Zum Laden der Daten dient die Strobe-Leitung PC2. Der Z80 kann jetzt mit einer Eingabeanweisung das Datenregister Port B lesen. Dadurch wird das Bit 1 des Port C logisch 0 und zeigt dem Peripheriegerät an, daß es neue Daten schicken kann. Abb. 6.16 zeigt das Flußdiagramm für diesen Transfer. Das zugehörige Z80-Programm ist in Abb. 6.17 zu sehen.

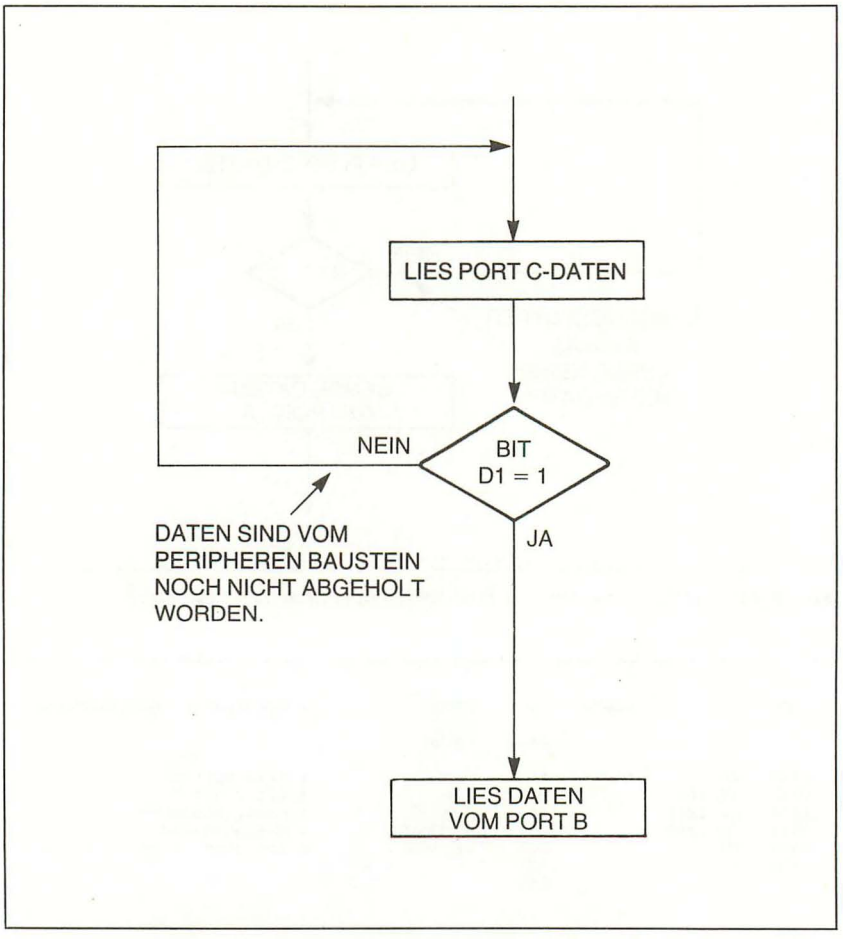


Abb. 6.16: *Flußdiagramm für eine Port-Leseoperation mit Handshake.*

```
1800   DB 12      BACK:   IN      A,(12H)        ; lese Port C
1802   CB 4F              BIT     1,A            ; Bit 1 = 1 ?
1804   CA 1800            JP      Z,BACK         ; nein, wiederhole
1807   DB 11              IN      A,(11H)        ; Lese Port B
1809   C9                 RET
                          END
```

Abb. 6.17: *Z80-Programm zur Realisierung des Flußdiagramms aus Abb. 6.16.*

Arbeitsmodus 2 des 8255

Ein anderer Arbeitsmodus des 8255 ist Modus 2. Hierbei kann Port A als Bidirektional-Port arbeiten. Das bedeutet, daß die acht Portleitungen Daten sowohl senden als auch empfangen können. Ist der 8255 in diesem Modus programmiert, entspricht der interne Ablauf dem Blockschaltbild aus Abb. 6.18.

In Abb. 6.18 ist ein Ausgabe- und ein Eingabezwischenspeicher (latch) zu sehen. Das Ausgabe-Latch speichert die vom Z80 stammenden Ausgabedaten und wartet darauf, daß das Peripheriegerät es ermöglicht, die Daten auf die Portleitungen PA0-PA7 zu legen. Das Eingabe-Latch speichert die vom Peripheriegerät gesendeten Daten.

Lassen Sie uns nun das Blockschaltbild in Abb. 6.18 untersuchen und sehen, wie die Daten zum Peripheriegerät gesendet werden. Der erste Vorgang dabei ist, daß der Z80 Daten in das Ausgabe-Latch des Port A schreibt. Dadurch wird die $\overline{\text{OBF}}$-Leitung (output buffer full) aktiv (logisch 0), und das Peripheriegerät wird darüber informiert. Für den Z80 bedeutet das außerdem, daß der Ausgabepuffer voll ist und noch nicht gelesen wurde.

Zum Abholen der Daten schickt das Peripheriegerät eine Bestätigung zum 8255 ($\overline{\text{ACK}}$-Leitung logisch 0-Impuls). Dadurch werden die Pufferdaten auf die Port-Leitungen gelegt. Außerdem wird das $\overline{\text{OBF}}$-Signal zurückgesetzt (logisch 1) und damit die CPU informiert, daß weitere Daten zum Port A gesendet werden können.

Bevor der 8255 Daten vom Peripheriegerät empfangen kann, muß es den logischen Zustand der IBF-Leitung (input buffer full) überprüfen. Ist die Leitung logisch 1, so sind die letzten Eingabedaten vom Mikroprozessor noch nicht gelesen worden. Wir wollen jetzt annehmen, daß die Leitung logisch 0, also der Eingabepuffer leer ist.

Das Peripheriegerät legt die Daten auf die Datenleitungen im Port A und

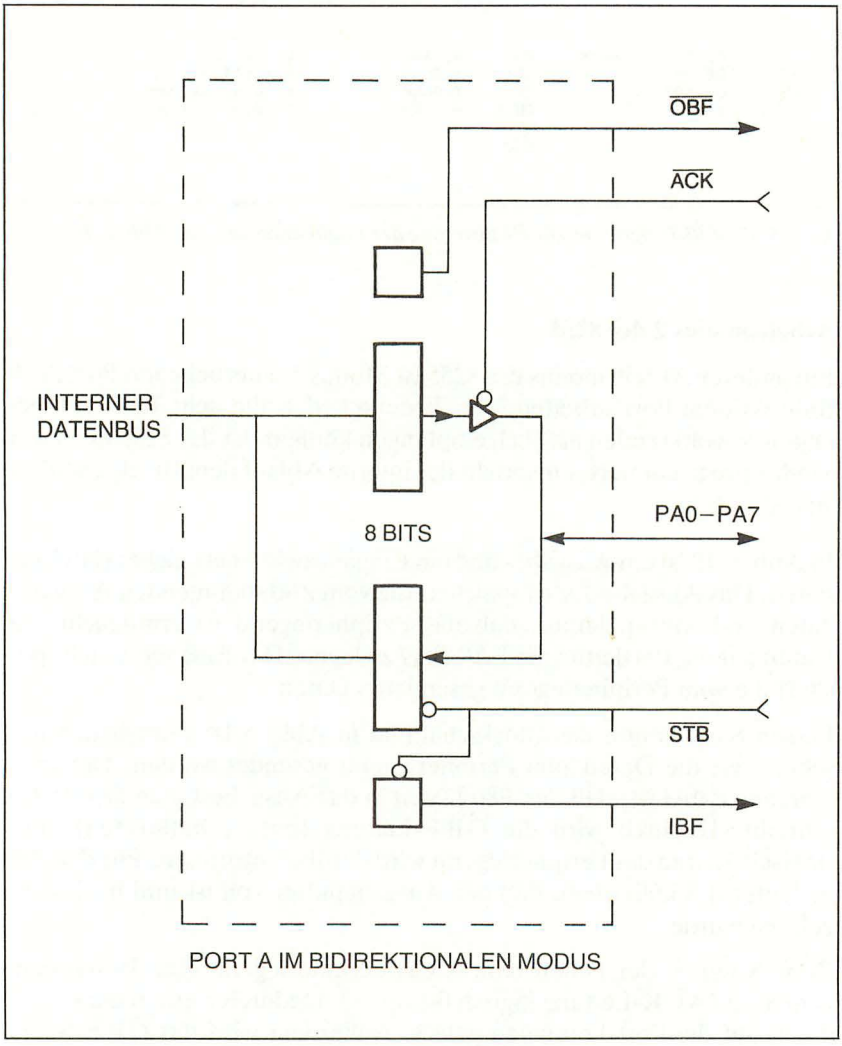


Abb. 6.18: *Interne Konfiguration im Modus 2.*

spricht den $\overline{\text{STB}}$-Eingang des 8255 an. Damit werden die Daten in das interne Eingangs-Latch abgespeichert und die IBF-Leitung logisch 1 gesetzt. Zum Lesen der Daten überprüft der Z80 den logischen Zustand dieser Leitung. Mit der Leseoperation des Prozessors wird die IBF-Leitung wieder zurückgesetzt. Das Peripheriegerät erfährt damit, daß es weitere Daten zum 8255 senden kann.

PORT-C-LEITUNG	DEFINITION
PC0	I/O
PC1	I/O
PC2	I/O
PC3	$INTR_A$
PC4	$\overline{STB}_A$
PC5	IBF_A
PC6	$\overline{ACK}_A$
PC7	$\overline{OBF}_A$

Abb. 6.19: *Pindefinition des Port C für Modus 2.*

Um der CPU zu ermöglichen, den Status der Handshake-Leitungen festzustellen, gibt das Port C ihren logischen Zustand wieder. Abb. 6.19 zeigt die Bit-Definition für das Port C im Modus 2.

Zusammenfassung

In diesem Kapitel haben wir die Benutzung des 8255 in Z80-Mikroprozessor-Systemen kennengelernt. Wir haben seine interne Organisation untersucht und eine Möglichkeit zur Verbindung mit dem Z80-Systembus vorgestellt. Weiterhin haben wir die benötigte Software besprochen.

Der 8255 wurde ursprünglich nicht zur Benutzung mit der Z80-CPU entwickelt. Er ist allerdings ein äußerst vielseitiges Bauteil. Seine einfache Einsetzbarkeit machen ihn auch für Z80-Anwendungen interessant.

Für den Einsatz dieses Bausteins in einem 4 MHz-Computersystem müssen aufgrund der schnellen Schaltzeiten Bausteine mit der Bezeichnung 8255 A-5 verwendet werden.

Kapitel 7

Der programmierbare Timerbaustein 8253

Einführung

Dieses Kapitel zeigt Ihnen die Benutzung des programmierbaren Timerbausteins 8253 in Z80-Mikroprozessor-Systemen. Wir werden mit der allgemeinen Beschreibung des Bausteins beginnen. Dann wollen wir ihn mit dem Z80-Mikroprozessor verbinden. Zuletzt stellen wir die für verschiedene Anwendungen erforderliche Software vor.

Blockschaltbild des Timerbausteins 8253

Lassen Sie uns mit der Untersuchung des Blockschaltbilds aus Abb. 7.1 beginnen und die internen Register des Bausteins besprechen. Das Schaltbild zeigt, daß der Timerbaustein drei identische und voneinander unabhängige Zähler besitzt. In diesem Kapitel werden wir die Benutzung dieser Zähler noch kennenlernen.

In Abb. 7.1 ist auch der Datenbus-Puffer zu sehen. Dieser Block puffert den Datenbus in beide Richtungen zwischen den internen Registern des 8253 und dem Mikroprozessor. Außerdem gibt es einen Block, als Lese/Schreibsteuerung bezeichnet, der die Lese- und Schreiboperationen der Zählerregister steuert. Der letzte Block, das Steuerwortregister (control word register), beinhaltet die vom Mikroprozessor kommenden Programmier-Informationen. Dieses Register bestimmt die logische Arbeitsweise des Chips. Abb. 7.2 zeigt die Pinbelegung des 8253.

Die drei Zählerleitungen: Clock, Gate und Out

Jeder der Zähler im Blockschaltbild Abb.7.1 besitzt drei nach außen führende Anschlüsse. Zwei dieser Anschlüsse, Clock (Takt) und Gate (Tor), sind Eingänge. Die Funktion dieser Leitungen ist abhängig von der Programmierung des Bausteins. Hier die allgemeine Definition:

Clock *(Takt)* Dies ist der Takteingang des Zählers. Der Zähler hat 16 Bit. Die maximale Taktfrequenz ist 2,6 MHz (für die Standardversion). Die Mindesttaktfrequenz ist 0 Hz (DC).

PROGRAMMABLE INTERVAL TIMER

- MCS-85™ Compatible 8253-5
- 3 Independent 16-Bit Counters
- DC to 2 MHz
- Programmable Counter Modes
- Count Binary or BCD
- Single +5V Supply
- 24-Pin Dual In-Line Package

The Intel® 8253 is a programmable counter/timer chip designed for use as an Intel microcomputer peripheral. It uses nMOS technology with a single +5V supply and is packaged in a 24-pin plastic DIP.

It is organized as 3 independent 16-bit counters, each with a count rate of up to 2 MHz. All modes of operation are software programmable.

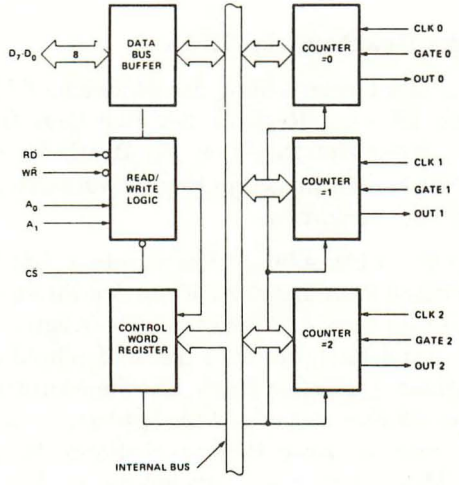


Abb. 7.1: *Blockschaltbild des Timerbausteins 8253.*

Gate *(Tor)* Dieser Eingang kann als Tor für den Takteingang dienen oder für einen Startimpuls (Triggerung). Seine Funktion ist vom programmierten Modus des Zählers abhängig.

Out *(Ausgang)* Die genaue Wirkungsweise dieses Zählerausgangs ist von der Programmierung abhängig.

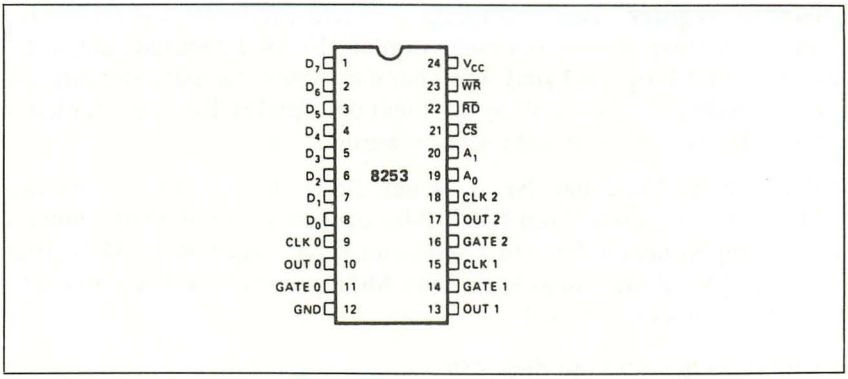


Abb. 7.2: *Pinbelegung des Timerbausteins 8253.*

Interne Register des 8253

Eine Liste der internen Register des 8253 wird in Abb. 7.3 vorgestellt. Lassen Sie uns als erstes das Moduswortregister besprechen. Dieses Register bestimmt die Arbeitsweise des Bausteins. Da die Zähler voneinander unabhängig sind, können sie jeweils durch Ausgabe eines bestimmten Datenworts in das Modusregister programmiert werden. Später werden wir das noch näher untersuchen. Zuerst sollen einmal die vier internen Register aus Abb. 7.3 definiert werden.

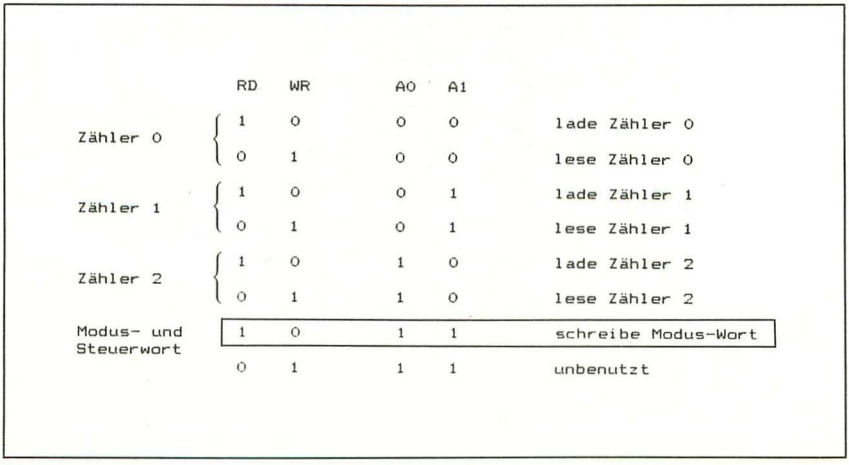


Abb. 7.3: *Die internen Register und ihre Ansteuerung.*

Steuerwortregister Bevor mit dem Baustein gearbeitet werden kann, muß dieses Register programmiert werden. Es wird dadurch adressiert, daß A0 und A1 logisch 1 sind. Die Daten in diesem Register steuern den Arbeitsmodus und das Zählformat (binär oder BCD). Das Register kann nur beschrieben, jedoch nicht gelesen werden.

Zähler Nr. 1, Nr. 2 und Nr. 3 Jeder dieser drei identischen Blöcke besteht aus einem setzbaren 16-Bit-Abwärtszähler. Sie sind vollkommen unabhängig voneinander und können im Binär- oder im BCD-Format arbeiten. Der Zählerstand kann vom Mikroprozessor gelesen werden, ohne daß er beeinflußt wird.

Verbindung des 8253 mit dem Z80

Bevor wir die Programmierung des 8253 kennenlernen, soll erst einmal die elektrische Verbindung mit dem Z80-Mikroprozessor behandelt werden.
Den 8253 kann man sich als Zusammenfassung von vier separaten I/O-Ports vorstellen. Die Gesamtanwahl des Bausteins wird über die $\overline{CS}$-Leitung gemacht. Wenn dieser Eingang logisch 0 ist, kann der Z80 mit dem 8253 kommunizieren.

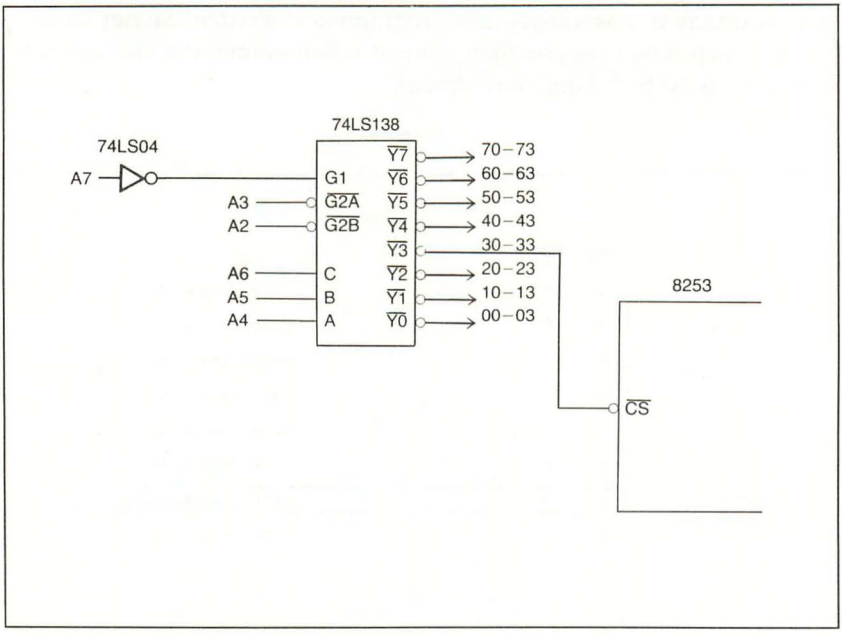


Abb. 7.4: Decodierung der CS-Leitung.

Die Adreßleitungen A0 und A1 bestimmen das anzusprechende interne Register. Die Ansteuerung entspricht der von Port-Bausteinen, wobei jedes interne Register einem Port entspricht.

Die Decodierung für die $\overline{CS}$-Leitung wird aus den oberen 6 Bits des unteren Adreßbytes gewonnen (A6-A2). Wir wollen den Baustein in die I/O-Architektur unseres Systems einfügen. Wir legen dazu fest, daß der 8253 den Adreßbereich 30H-33H im I/O-Belegungsplan einnimmt. Das bedeutet, daß die I/O-Adressen 30H, 31H, 32H und 33H für den 8253 reserviert sind (siehe Abb. 7.4).

Der $\overline{RD}$- und der $\overline{WR}$-Eingang wird direkt mit der $\overline{IOR}$- und der $\overline{IOW}$-Systemleitung verbunden. Das entspricht der Anschlußweise des 8255 in Kapitel 6. Abb. 7.5 zeigt die Verbindung.

Die Datenleitungen des 8253 können direkt an den Z80-Datenbus geschlossen werden. Abb. 7.6 zeigt einen vollständigen Verbindungsplan des 8253 zum Z80 ohne Datenpuffer; Abb. 7.7 zeigt den gleichen Plan mit Datenpuffer. Als Pufferbaustein wird der 74LS245 benutzt. Nur wenn der 8253 selektiert ist, und die $\overline{RD}$-Leitung aktiv ist, werden die Daten vom

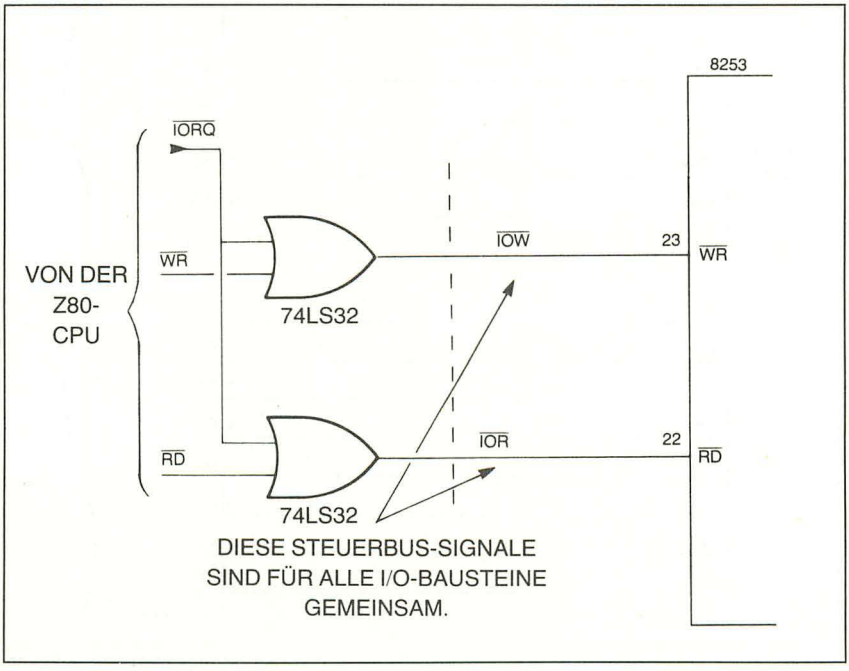


Abb. 7.5: *Erzeugung des $\overline{IOR}$- und des $\overline{IOW}$-Signals.*

Timerbaustein zum Datenbus übertragen. Während der übrigen Zeit verläuft die Datenrichtung vom Bus zum 8253.

Die Verwendung der Schaltungen aus Abb. 7.6 und Abb. 7.7 erlauben eine zuverlässige Kommunikation zwischen dem Z80 und dem 8253. Der Timerbaustein muß jedoch noch auf bestimmte Weise programmiert werden.

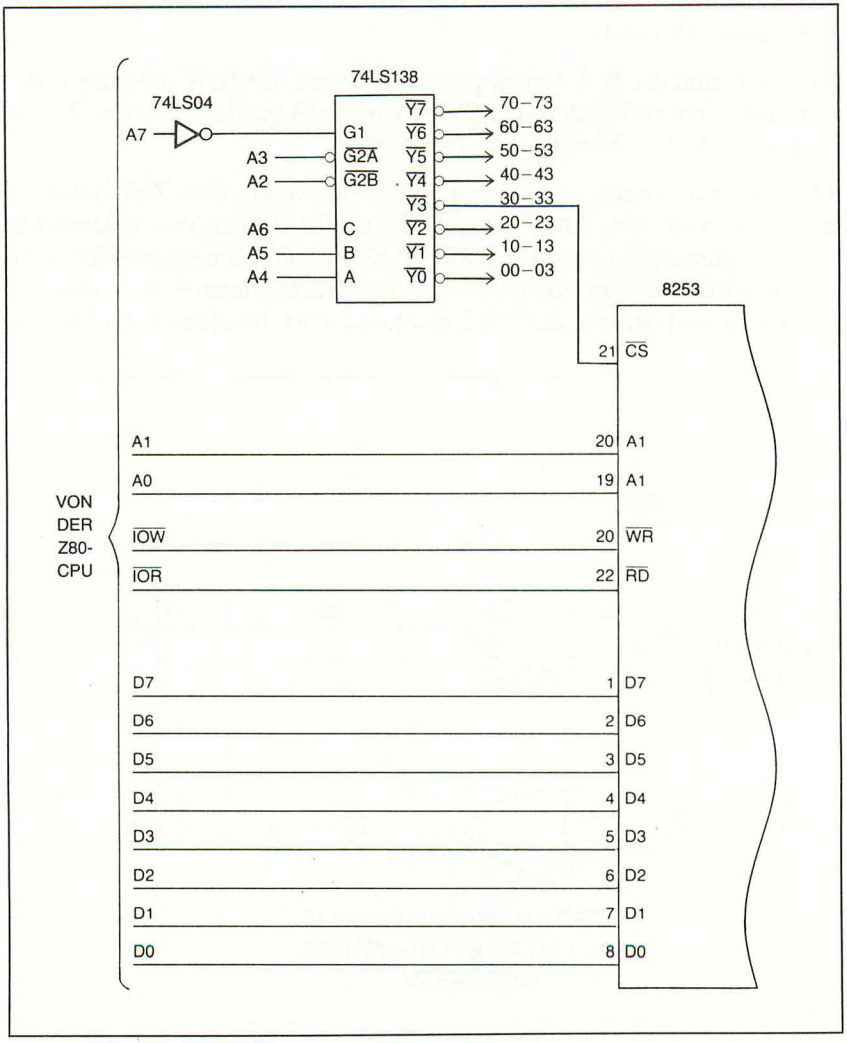


Abb. 7.6: *Verbindung des 8253 mit dem Z80 ohne Datenpuffer.*

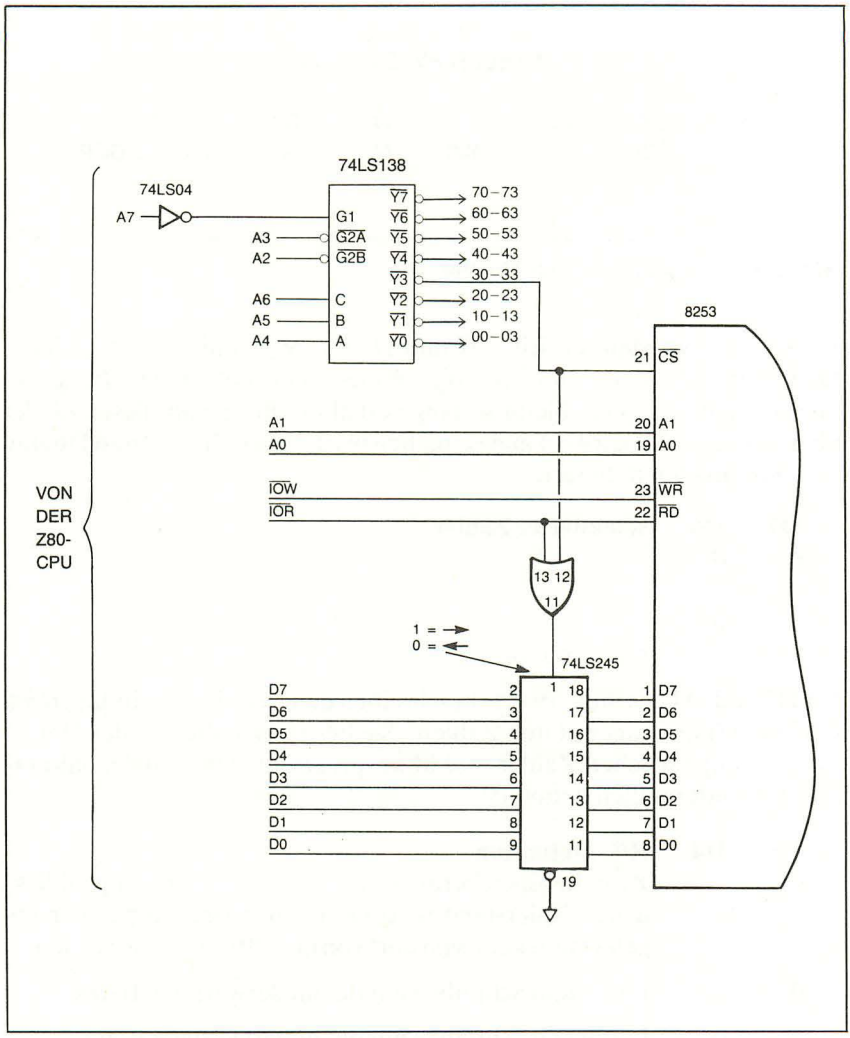


Abb. 7.7: *Verbindung des 8253 mit dem Z80 mit Datenpuffer.*

Programmierung des Timerbausteins (Steuerwortformat)

Durch Programmierung des Steuerregisters lassen sich verschiedene Operationsarten des Timers anwählen. Abb.7.8 zeigt das Format der dazu verwendeten Steuerwörter. Das Steuerregister läßt sich elektrisch mit A0=1 und A1=1 anwählen. In unserem System ist seine Adresse 33H.

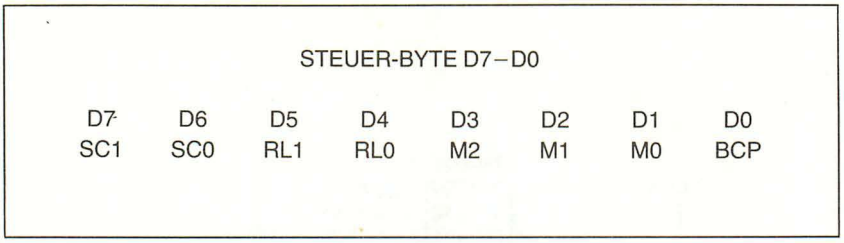


Abb. 7.8: *Bit-Definition für das Steuerwort.*

In Abb. 7.8 werden die Bits D7 und D6 mit SC1 und SC0 bezeichnet. Diese Bits selektieren den zu programmierenden Zähler. Die Programmierung eines Zählers bleibt so lange erhalten, bis genau dieser Zähler über ein neues Steuerwort angesprochen wird. Diese Bits D7 und D6 sind folgendermaßen definiert:

D7	**D6**	**selektierter Zähler**
0	0	0
0	1	1
1	0	2
1	1	nicht erlaubt

Bit D5 und D4 des Steuerworts beschreiben den Lese/Lade-Modus (read/load mode) des angewählten Zählers. Sie bestimmen die Art der Datenübertragung zwischen Zähler und Mikroprozessor. Die Bits D5 und D4 sind folgendermaßen definiert:

D5	**D4**	**R/L-Definition**
0	0	Zwischenspeicherungs-Anweisung. Der augenblickliche Zählerstand wird in einem Zwischenspeicher abgelegt und kann von dort von der CPU gelesen werden.
0	1	Lesen und Schreiben nur des niederwertigen Bytes.
1	0	Lesen und Schreiben nur des höherwertigen Bytes.
1	1	Lesen und Schreiben beider Bytes, das niederwertige zuerst.

Die Zwischenspeicherung des augenblicklichen Zählerinhalts (D5=0 und D4=0) ermöglicht das Lesen eines stabilen Zählerwerts auch bei laufendem Zähler. Wird diese Methode nicht benutzt, kann es sein, daß sich der Zählerstand gerade während des Auslesens ändert (siehe Abb.7.9). Dadurch werden falsche Daten gelesen. Die Zwischenspeicherungs-Anweisung wird jeweils vor einer Lese-Anweisung gegeben.

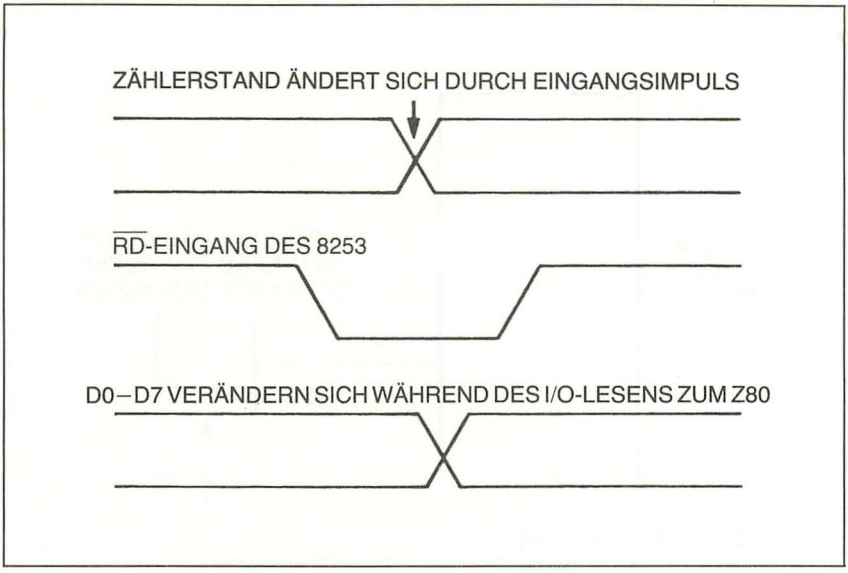


Abb. 7.9: *Timing, das zeigt, wie sich der Zählerstand während einer Leseoperation ändern kann.*

Auf die Zwischenspeicherung kann verzichtet werden, wenn der Zähler während des Auslesens angehalten wird. Das kann durch Benutzung des Gate-Eingangs oder mit externer Hardware realisiert werden (siehe Abb. 7.10). Jede dieser Methoden hat ihre Nachteile. Bei der Zwischenspeicherung bekommt der Mikroprozessor einen um mehrere Zyklen veralteten Zählerstand (abhängig von der Taktgeschwindigkeit und der Art des Auslesens). Die direkte Methode erfordert dagegen zusätzliche Hardware. Außerdem kann sie u.U. die gesamte Arbeitsweise des Systems beeinflussen.

Die nächsten drei Bits des Steuerworts sind D3, D2 und D1. Sie bestimmen den grundlegenden Betriebsmodus des Zählers. Hier sind die Definitionen:

D3	**D2**	**D1**	**Modusart**
0	0	0	Modus 0: Interrupt beim letzten Takt
0	0	1	Modus 1: Programmierbarer Zeitgeber (one shot)
0	1	0	Modus 2: Taktgenerator
0	1	1	Modus 3: Rechteckgenerator
1	0	0	Modus 4: Software – getriggerter Impuls
1	0	1	Modus 5: Hardware – getriggerter Impuls

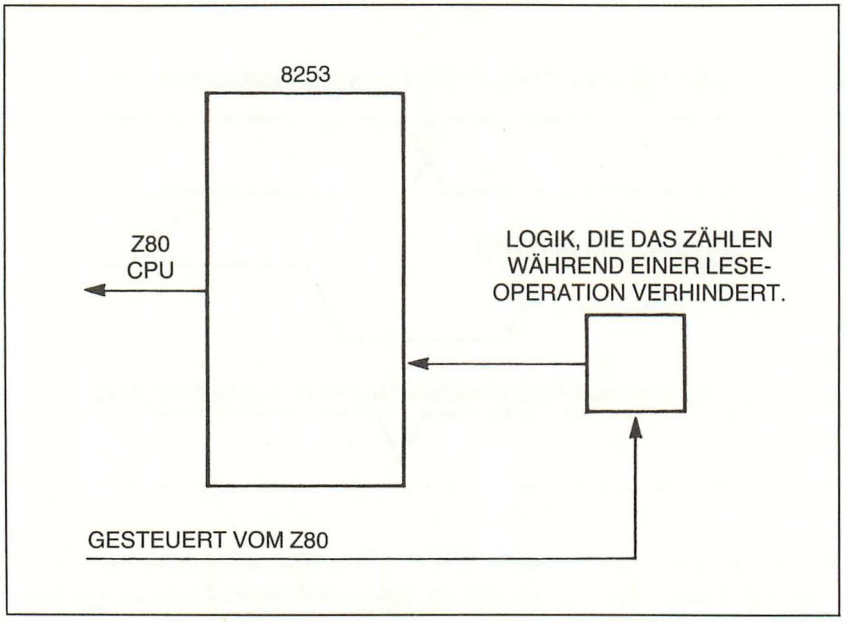


Abb. 7.10: *Durch externe Logik kann der Zählerstand während des Lesens festgehalten werden.*

Das letzte Bit (D0) des Steuerworts bestimmt, in welcher Art gezählt wird, binär (logisch 0) oder in BCD (logisch 1). Der höchste Zählerstand bei binärer Zählweise ist 2^{16} und bei dezimaler Zählweise 10^4.

Beispiel für den Modus 0: Interrupt beim letzten Takt

Wir wollen hier den Betriebsmodus 0 des 8253 kennenlernen und die Funktion der beteiligten Anschlüsse untersuchen. Zum Schluß des Abschnitts soll auch eine Routine zum Initialisieren des 8253 vorgestellt werden.

Der Modus 0 bedeutet am 8253 folgendes: Der Zähler wird auf einen Anfangswert gesetzt und zählt dann abwärts in einer der Eingangsfrequenz entsprechenden Rate. Wenn der Zählerstand 0000 erreicht hat, wird der OUT-Pin des Zählers logisch 1. Von diesem Pin könnte z.B. ein Interrupt am Mikroprozessor ausgelöst werden. Der Ausgang bleibt logisch 1 bis der Zählermodus geändert wird oder ein neuer Anfangswert programmiert wird. Legt man den Gate-Eingang auf logisch 0, so wird die Zählung unterbrochen.

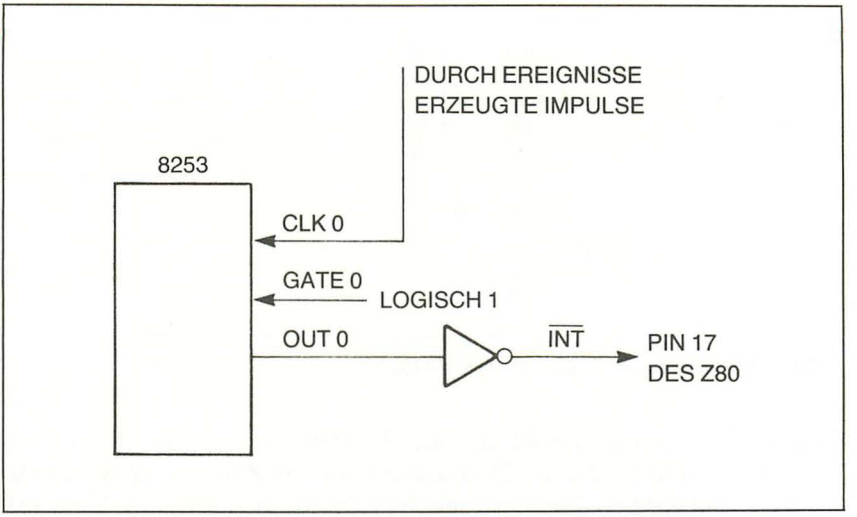


Abb. 7.11: *Blockschaltbild für das Anwendungsbeispiel im Modus 0.*

Wir wollen die Arbeitsweise des Modus 0 anhand eines Beispiels untersuchen. Es sollen dabei Ereignisse mit dem Zähler 0 gezählt werden. Die Zählweise ist dezimal (BCD). Nach 125 Zählimpulsen soll der Z80 einen Interrupt bekommen. Die Interrupt-Service-Routine liegt bei Adresse 0038H; der Z80 arbeitet im Interrupt-Modus 1. Das Blockschaltbild für dieses Beispiel zeigt Abb. 7.11.

Zum Aufsetzen des Zählers muß das Steuerregister programmiert werden. Basierend auf der Bit-Definition von Abb. 7.8 wählen wir als Steuerwort 00110001. Es setzt sich folgendermaßen zusammen:

Bit D7 und D6 = 00	Zähler 0
Bit D5 und D4 = 11	Es werden beide Bytes geladen.
Bit D3, D2 und D1 = 000	Betriebsmodus 0
Bit D0 = 1	dezimale Zählung

Damit hat das Steuerwort für Port 33H den Wert 31H.

Als nächstes muß der Anfangszählerstand in das Zählerregister geschrieben werden. Da wir in BCD arbeiten, muß das niederwertige Byte 25H und das höherwertige Byte 01 sein. Diese beiden Bytes werden zur Adresse 30 geschrieben. Abb.7.12 zeigt einen Ausschnitt aus dem Z80-Assembler-Listing für die Initialisierung des 8253.

```
                ; Wir gehen davon aus, daß die Interrupts gesperrt sind

0000    3E 31           LD      A,31H           ; Steuerwort
0002    D3 33           OUT     (33H),A         ; zum Steuerregister des 8253
0004    3E 25           LD      A,25H
0006    D3 30           OUT     (30H),A         ; unteres Byte des Zählers
0008    3E 01           LD      A,01H
000A    D3 30           OUT     (30H),A         ; oberes Byte des Zählers

                ; gezählt wird in BCD

                ; Zählung beginnt nach dem Setzen des oberen Bytes

000C    FB              EI
                        END
```

Abb. 7.12: *Initialisierung des 8253 für Modus 0.*

Der Zähler startet, sobald das zweite Datenbyte in das Register 0 geschrieben wurde. Nach 125 Impulsen hat der Zähler den Wert 0000 erreicht, und der Out-Pin wird logisch 1. Das löst den Interrupt beim Z80 aus. Die Interrupt-Service-Routine erledigt die gewünschte Arbeit. Teil dieser Routine ist die erneute Programmierung des 8253. Abb. 7.13 zeigt als Beispiel eine Interrupt-Service-Routine für den Z80 im Interrupt-Modus 1.

Modus 1: Programmierbarer Zeitgeber (one-shot)

Im Modus 1 kann der 8253 einen Ausgangsimpuls in der Länge der programmierten Taktzyklen ausgeben. Der Zeitgeber wird dabei mit der ansteigenden Flanke eines Signals am Gate-Eingang getriggert. Trifft ein

```
                ; Der Z80 arbeitet im Interrupt-Modus 1

                ; Zu diesem Zeitpunkt sind 125 Impulse gezählt worden
                ; und wir laden Zähler 0 neu.

0000    3E 25   CNTINT: LD      A,25H
0002    D3 30           OUT     (30H),A         ; lade unteres Byte
0004    3E 01           LD      A,01H
0006    D3 30           OUT     (30H),A         ; lade oberes Byte

                ; Die neue Zählung hat begonnen

0008    FB              EI
0009    C9              RET

                ; Als Teil der Interrupt-Routine könnten bestimmte
                ; Aktionen ausgelöst werden. In diesem Beispiel wird
                ; lediglich der Zähler neu geladen.

                        END
```

Abb. 7.13: *Interrupt-Service-Routine für den Z80-Interruptmodus 0.*

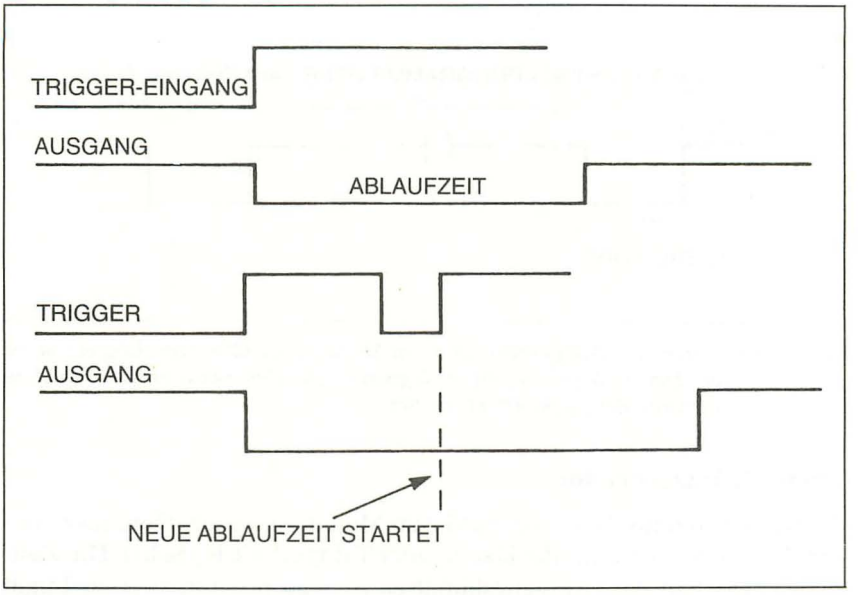


Abb. 7.14: *Timing für den 8253 als programmierbaren Zeitgeber (one-shot). Wenn der Triggereingang logisch 1 wird, geht der Ausgang für die programmierte Zeitdauer auf logisch 0. Trifft während des Zeitablaufs ein weiterer Triggerimpuls ein, so startet die Zeit von neuem.*

Triggerimpuls während der Laufzeit des Zeitgebers ein, so startet die Zeit von neuem (nachtriggerbar, siehe Abb. 7.14).
Als Beispiel lassen Sie uns annehmen, die Eingangsfrequenz sei 1 MHz. Der Ausgangsimpuls des Zeitgebers soll exakt 75 Mikrosekunden lang sein. Das Z80-Programm zur Einleitung dieses Ablaufs ist in Abb. 7.15 zu sehen. Als programmierbarer Zeitgeber wird dabei der Zähler 1 benutzt.

```
0000   3E 72            LD      A,01110010B
0002   D3 33            OUT     (33H),A          ; Ausgabe des Steuerwortes

            ; Zähler 1, RL = 3, M = 1, binäre Zählung

0004   3E 4B            LD      A,75             ; dezimal 75
0006   D3 31            OUT     (31H),A          ; niederw. Byte
0008   3E 00            LD      A,00
000A   D3 31            OUT     (31H),A          ; höherw. Byte

            ; Der Baustein ist jetzt programmiert und wartet auf einen
            ; Trigger-Impuls

                        END
```

Abb. 7.15: *Z80-Programm zum Aufsetzen des 8253 in den Modus 1.*

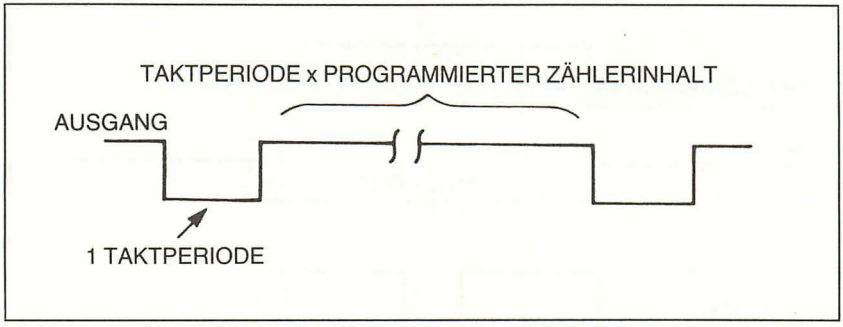


Abb. 7.16: *Timing des Ausgangssignals beim Modus 2 des 8253. Der Ausgang ist für die Dauer einer Taktperiode logisch 0. Die Zeit zwischen den Impulsen wird von der Zeitkonstante bestimmt.*

Modus 2: Taktgenerator

Im Betriebsmodus 2 wird der 8253 zum Modulo-n-Teiler. Der Ausgangspin des Zählers wird für die Dauer einer Taktperiode logisch 0. Die Zeitdauer zwischen den Ausgangsimpulsen ist vom programmierten Inhalt des Zählerregisters abhängig (siehe Abb. 7.16). Wird zwischen zwei Ausgangsimpulsen eine neue Zeitkonstante geladen, so wird die laufende Zeit nicht beeinflußt. Erst nach dem nächsten Ausgangsimpuls wird die neue Zeitkonstante wirksam.

Als Beispiel nehmen wir einmal an, daß die gewünschte Frequenz am Ausgang des Zählers 2 638 Hz betragen soll. Das entspricht einer Periodendauer von 1567 Mikrosekunden. Wenn die Eingangsfrequenz 1 MHz beträgt, so muß als Zeitkonstante 1567 programmiert werden. Das kann sowohl in BCD als auch binär geschehen. Abb. 7.17 zeigt ein Z80-Programm zur Initialisierung des 8253 für diese Operation.

```
0000   3E B5          LD     A,10110101B
0002   D3 33          OUT    (33H),A        ; Ausgabe des Steuerwortes
0004   3E 67          LD     A,67H
0006   D3 32          OUT    (32H),A        ; Zähler 2, unteres Byte BCD
0008   3E 15          LD     A,15H
000A   D3 32          OUT    (32H),A        ; Zähler 2, oberes Byte BCD

               ; Der Baustein ist jetzt initialisiert und die Ausgangs-
               ; Frequenz liegt bei 638 Hz (für 1 MHz Eingangstakt).

                      END
```

Abb. 7.17: *Z80-Programm, das den 8253 als programmierbaren Taktgenerator benutzt. Bei 1 MHz Eingangsfrequenz ist die Ausgangsfrequenz 683 Hz.*

Rechteckgenerator

Der Modus 3 entspricht weitgehend dem Modus 2. Der einzige Unterschied ist der, daß das Ausgangssignal symmetrisch ist, das heißt, es ist eine halbe Periode lang logisch 1 und die andere Hälfte logisch 0. Hat die programmierte Zeitkonstante einen ungeraden Wert, so wird sie aufgeteilt in (n+1)/2 für logisch 1 und (n-1)/2 für logisch 0. Abb. 7.18 zeigt das Z80-Programm zum Aufsetzen des Zählers 0 als 10-kHz-Rechteckgenerator, wobei wieder von einer Eingangsfrequenz von 1 MHz ausgegangen wird.

```
0000   3E 36          LD      A,00110110B
0002   D3 33          OUT     (33H),A         ; Ausgabe des Steuerwortes
0004   3E 64          LD      A,100
0006   D3 30          OUT     (30H),A         ; Zähler 0, unteres Byte bin.
0008   3E 00          LD      A,0
000A   D3 30          OUT     (30H),A         ; Zähler 0, oberes Byte bin.

               ; Der Baustein ist jetzt initialisiert und die Ausgangs-
               ; Frequenz liegt bei 10 kHz (für 1 MHz Eingangstakt).

                       END
```

Abb. 7.18: *Z80-Programm, das den 8253 als programmierbaren Rechteckgenerator benutzt. Bei 1 MHz Eingangsfrequenz ist die Ausgangsfrequenz 10 kHz.*

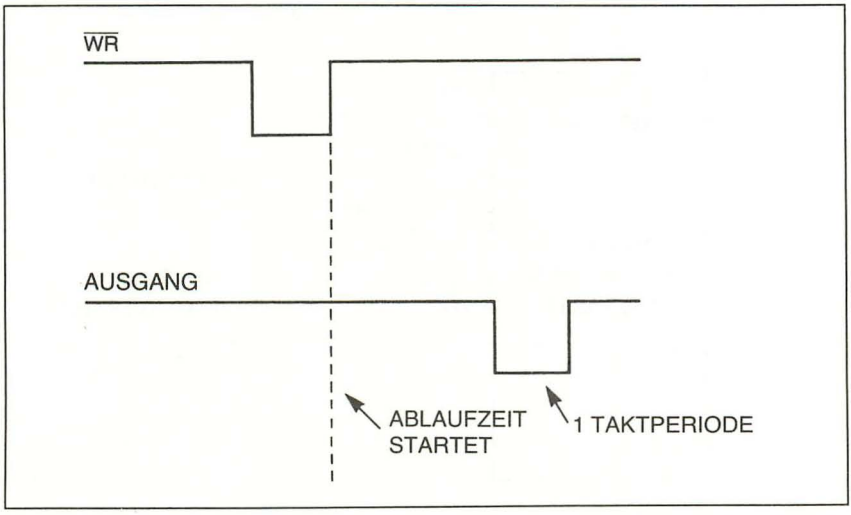


Abb. 7.19: *Timing des Ausgangssignals des 8253 bei Benutzung als software-getriggerter Impulsgeber. Der Timer startet nach der letzten Schreiboperation des Prozessors. Am Ende des Zählvorgangs wird der Ausgang für die Dauer einer Taktperiode logisch 0.*

Modus 4: Software-getriggerter Impuls

In diesem Modus kann das Programm den Zähler für den Ablauf einer bestimmten Wartezeit ab dem Laden der Zeitkonstante aufsetzen. Mit dem letzten Takt, wenn der Zählerstand 0000 ist, wird der Ausgang für die Dauer einer Taktperiode logisch 0 und danach wieder logisch 1 (siehe Abb.7.19). Sobald der Modus 4 gesetzt ist, wird der Ausgang logisch 1.

Ein Beispiel für die Benutzung von Modus 4

Für unser Beispiel wollen wir den Zähler 2 des 8253 für einen software-getriggerten Impuls von 100 Millisekunden programmieren. Die Eingangsfrequenz ist 1 MHz und damit die Zeitkonstante 10^5. Die Zählung soll im BCD-Format erfolgen. Da die größtmögliche Zeitkonstante 10^4 ist, müssen wir zwei Zähler hintereinander schalten. Der erste Zähler teilt die Frequenz auf 1 kHz herunter. Der zweite erzeugt den software-getriggerten Impuls (siehe Abb. 7.20). Abb. 7.21 zeigt ein Z80-Programm, das den 8253 in der oben beschriebenen Art verwendet.

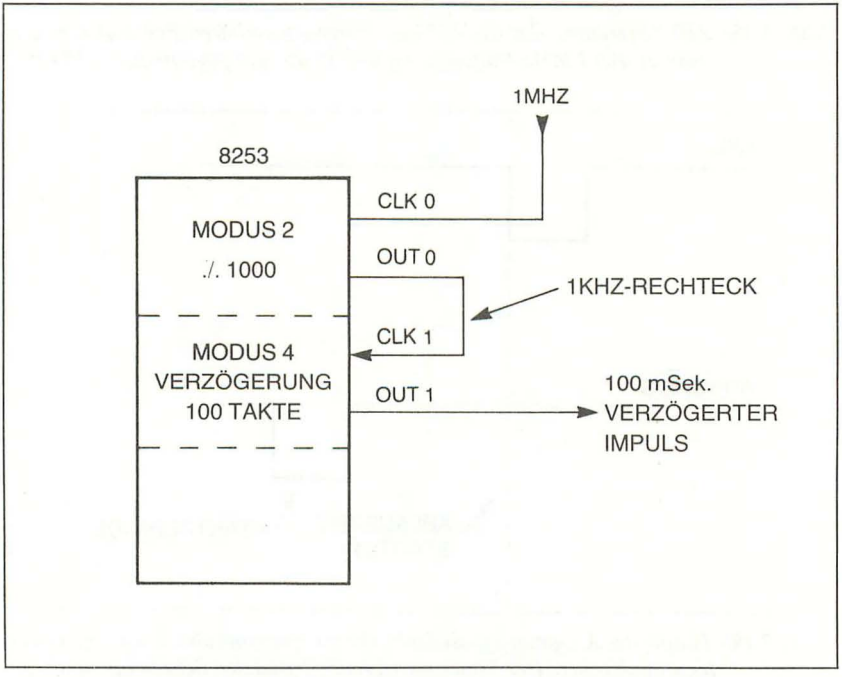


Abb. 7.20: *Auf diese Weise kann der 8253 einen um 100 Millisekunden verzögerten Impuls erzeugen.*

```
0000   3E 35            LD      A,00110101B
0002   D3 33            OUT     (33H),A         ; Ausgabe des Steuerwortes
0004   3E 00            LD      A,00H
0006   D3 30            OUT     (30H),A         ; Zähler 0, unteres Byte
0008   3E 10            LD      A,10H
000A   D3 30            OUT     (30H),A         ; Zähler 0, oberes Byte

               ; Der Zähler 0 ist jetzt initialisiert und gibt einen 1 kHz-
               ; Rechtecktakt bei 1 MHz Eingangsfrequenz ab. Der Ausgang
               ; wird mit dem Takt-Eingang vom Zähler 1 verbunden.

000C   3E 79            LD      A,01111001B
000E   D3 33            OUT     (33H),A         ; Ausgabe des Steuerwortes
0010   3E 00            LD      A,00H
0012   D3 31            OUT     (31H),A         ; Zähler 1, unteres Byte
0014   3E 01            LD      A,01H
0016   D3 31            OUT     (31H),A         ; Zähler 1, oberes Byte

               ; Die letzte Ausgabe zum Zähler 1 startet den
               ; Zeitablauf

                        END
```

Abb. 7.21: *Z80-Programm zur Erzeugung des software-getriggerten Impulses von 100 Millisekunden. Das Programm setzt dabei die in Abb. 7.20 verwendete Struktur voraus.*

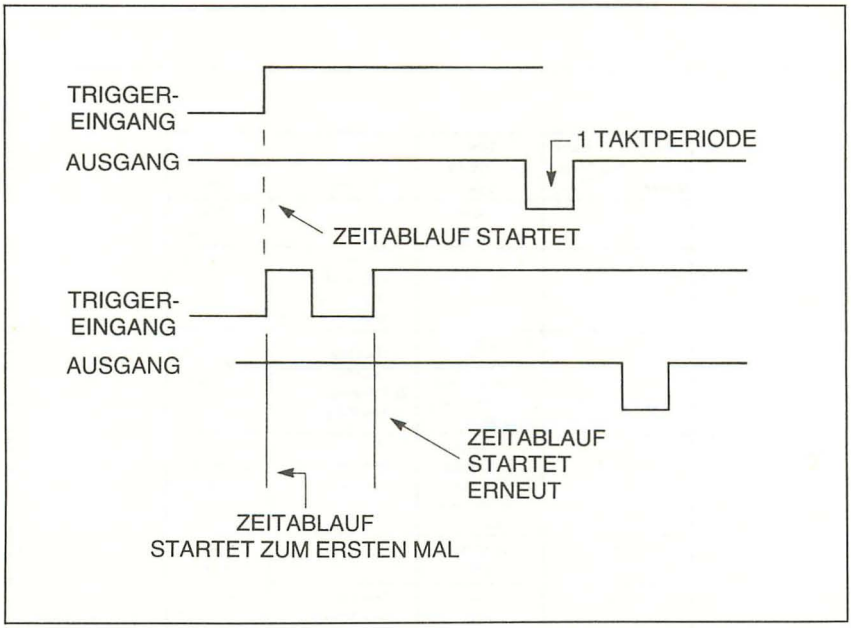


Abb. 7.22: *Timing beim Einsatz des 8253 im Modus 5. Wenn der Triggereingang logisch 1 wird, startet der Zeitablauf. Trifft vor Ablauf der Zeit ein neuer Triggerimpuls ein, so startet der Zeitablauf erneut.*

Modus 5: Hardware-getriggerter Impuls

In diesem Modus wird der Zähler durch eine steigende Flanke am Gate-Eingang gestartet. Beim Zählerstand 0000 wird der Ausgang für die Dauer einer Taktperiode logisch 0. Der Timer ist nachtriggerbar, das heißt, der Zeitablauf beginnt von neuem, wenn der Gate-Eingang eine steigende Flanke erhält (siehe Abb. 7.22). Abb. 7.23 zeigt das Z80-Programm zur Initialisierung des Zählers 1 als hardware-getriggerter Impulsgeber.

```
0000   3E 7A          LD      A,01111010B
0002   D3 33          OUT     (33H),A         ; Ausgabe des Steuerwortes
0004   3E 55          LD      A,85
0006   D3 31          OUT     (31H),A         ; Zähler 1, unteres Byte bin.
0008   3E 00          LD      A,00H
000A   D3 31          OUT     (31H),A         ; Zähler 1, oberes Byte bin.

               ; Der Zähler ist jetzt initialisiert und erwartet einen
               ; Trigger-Impuls auf TRG1.

                       END
```

Abb. 7.23: *Z80-Programm zum Aufsetzen des 8253 im Modus 5.*

Modes \ Signal Status	Low Or Going Low	Rising	High
0	Disables counting	— —	Enables counting
1	— —	1) Initiates counting 2) Resets output after next clock	— —
2	1) Disables counting 2) Sets output immediately high	**1) Reloads counter** **2) Initiates counting**	Enables counting
3	1) Disables counting 2) Sets output immediately high	Initiates counting	Enables counting
4	Disables counting	— —	Enables counting
5	— —	Initiates counting	— —

Abb. 7.24: *Tabelle für die verschiedenen Benutzungsarten des Gate-Eingangs.*

Benutzung des Gate-Eingangs

Jeder Betriebsmodus benutzt den Gate-Eingang in anderer Weise. Die Tabelle in Abb.7.24 gibt darüber eine Zusammenstellung.

Zusammenfassung

In diesem Kapitel lernten wir den Timerbaustein 8253 im Zusammenhang mit Z80-Systemen kennen. Wir haben das Blockschaltbild des Bausteins vorgestellt und die einzelnen Betriebsarten besprochen. Der 8253 eignet sich aufgrund seiner Vielseitigkeit für fast alle Timer-Funktionen. Die Beispiele in diesem Kapitel zeigten, daß sein Einsatz keine schwierige Sache ist.

Für den Einsatz dieses Bausteins in einem 4 MHz-Computersystem müssen aufgrund der schnellen Schaltzeiten Bausteine mit der Bezeichnung 8253-5 verwendet werden.

Kapitel 8

Der Z80-PIO-Baustein

Einführung

In diesem Kapitel wollen wir den Z80-PIO-Baustein (Mostek MK3881) vorstellen. Er wird als programmierbarer I/O-Baustein in Z80-Systemen eingesetzt. Im Gegensatz zu dem in Kapitel 6 besprochenen 8255 ist das PIO speziell für die Zusammenschaltung mit dem Z80 entwickelt worden.

Das Z80-PIO (PIO = peripherial I/O port) ist ein I/O-Baustein, der für verschiedene Betriebsarten programmiert werden kann. In diesem Kapitel beschreiben wir die Arbeitsweise jeder Betriebsart und zeigen die Anwendung im Z80-System.

Blockschaltbild des PIO

Wir wollen als erstes das Blockschaltbild in Abb. 8.1 untersuchen. Im Schaltbild sind zwei unabhängige Portkanäle zu sehen: A und B. Wir werden später sehen, daß sie in ihrer Arbeitsweise fast identisch sind.

Nach außen hin hat jedes Port acht Datenleitungen und zwei Steuerleitungen. Die Ein- und Ausgänge sind TTL-kompatibel. Jeder Ausgang kann im 0-Zustand 2 Milliampere und im 1-Zustand 250 Mikroampere treiben. Das ist genug zur Ansteuerung einer Standard-TTL-Last mit 1,6 Milliampere bei 0 und 40 Mikroampere bei 1.

Zwei weitere Blöcke in Abb.8.1 sind die interne Steuerlogik und die Interrupt-Steuerung. Die interne Steuerlogik regelt den internen Datenverkehr im richtigen Timing. Die Interrupt-Steuerung dient der Behandlung von Interrupt-Anforderungen und -Bestätigungen in Z80-Systemen. Wir werden der Interrupt-Programmierung des PIO später einen eigenen Abschnitt widmen.

Die Pinbelegung des Z80-PIO

Abb. 8.2 zeigt die Pinbelegung des Z80-PIO im 40-poligen DIP-Gehäuse. +5V und Masse (GND) dienen der Stromversorgung. Lassen Sie uns nun die Bedeutung der einzelnen Pins besprechen. Danach wollen wir sehen, wie das PIO mit dem Z80-Mikroprozessor zusammengeschaltet wird.

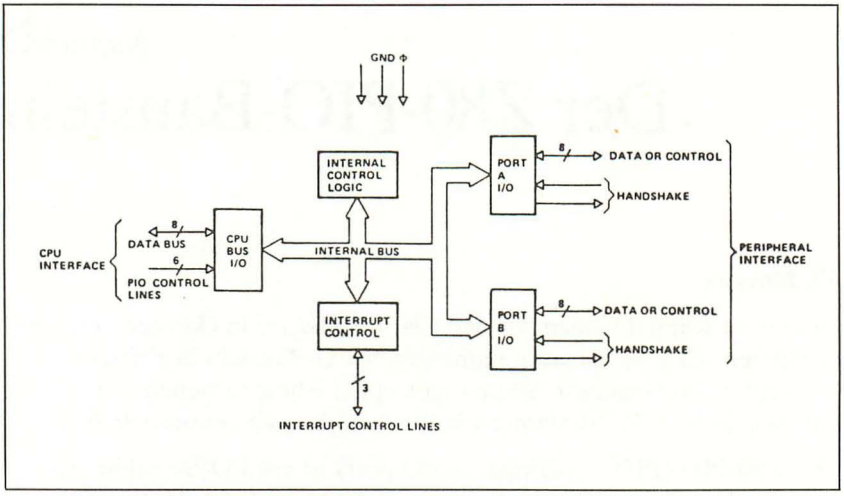


Abb. 8.1: *Blockschaltbild des Z80-PIO.*

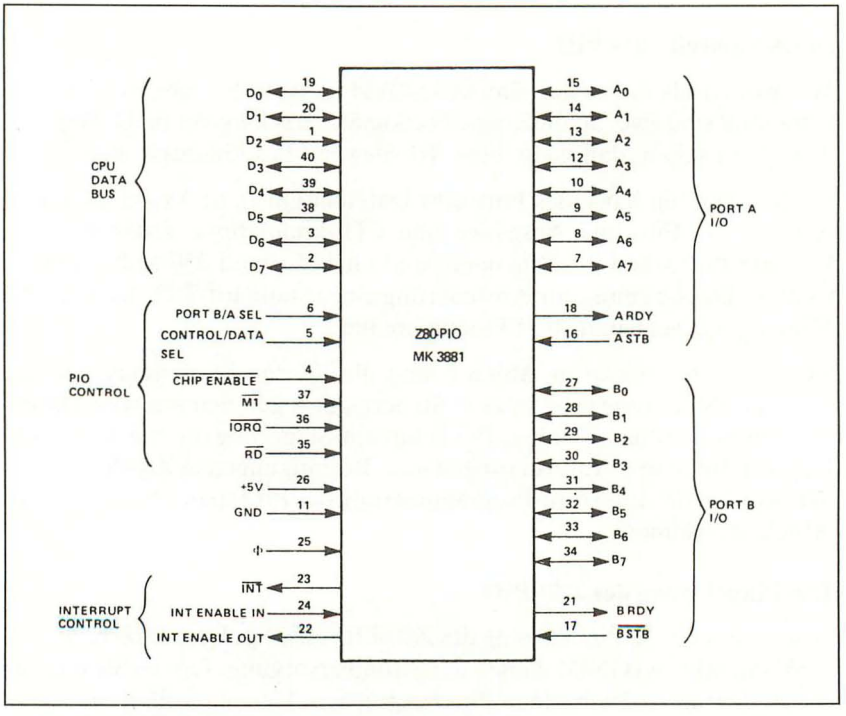


Abb. 8.2: *Pinbelegung des Z80-PIO.*

D7-D0 Dies sind die Daten-I/O-Leitungen zur Kommunikation mit dem Z80-Mikroprozessor.

B/A-SEL Diese Eingangsleitung bestimmt, ob der Datenbus mit Port B (logisch 1) oder Port A (logisch 0) kommuniziert. Die Leitung wird normalerweise mit A0 des Adreßbusses verbunden.

C/D-SEL Der logische Zustand dieser Leitung bestimmt, ob das Datenport oder das Steuerwortregister des selektierten Ports angesprochen werden soll. Das Steuerwortregister wird zur Programmierung der verschiedenen Betriebsarten benutzt.

$\overline{\text{CE}}$ *(Chip Enable)* Ist diese Eingangsleitung logisch 0, so kann das PIO mit dem Mikroprozessor kommunizieren. Die $\overline{\text{CE}}$-Leitung wird normalerweise aus den Adreßleitungen A7-A2 decodiert.

CLOCK Dies ist der Eingang für einen einphasigen Takt, der zur internen Ablaufsteuerung benutzt wird. Normalerweise wird er mit der Systemtaktleitung des Z80 verbunden.

$\overline{\text{M1}}$ Dieser Eingang wird mit dem $\overline{\text{M1}}$-Ausgang des Z80-Mikroprozessors verbunden. Das PIO benutzt ihn zur Erkennung von Befehlscode-Lesezyklen (im Zusammenhang mit der $\overline{\text{RD}}$-Leitung) und zur Erkennung der Interrupt-Bestätigung (im Zusammenhang mit der $\overline{\text{IORQ}}$-Leitung). Das PIO berücksichtigt einen speziellen Befehlscode des Z80: die RETI-Anweisung.

Dazu kommen noch zwei weitere Funktionen des $\overline{\text{M1}}$-Eingangs:

- Er synchronisiert die Interrupt-Logik des PIO.
- Er erzeugt einen internen Reset, wenn weder die $\overline{\text{RD}}$-Leitung noch die $\overline{\text{IORQ}}$-Leitung aktiv sind. Dies erspart den Pin für einen speziellen Reset-Eingang.

$\overline{\text{IORQ}}$ Dieser Eingang wird mit dem $\overline{\text{IORQ}}$-Ausgang des Z80 verbunden. Eine interne Verknüpfung mit dem $\overline{\text{CE}}$-Eingang sorgt dafür, daß der Datentransfer zwischen PIO und Prozessor nur stattfindet, wenn $\overline{\text{IORQ}}$ und $\overline{\text{CE}}$ logisch 0 sind.

Die zweite Funktion des $\overline{\text{IORQ}}$-Eingangs ist es, das PIO darüber zu informieren, daß eine Interrupt-Bestätigung vorliegt. $\overline{\text{IORQ}}$ und $\overline{\text{M1}}$ sind während des Interrupt-Bestätigungs-Zyklus beide logisch 0. Ist das PIO das unterbrechende Bauteil, so wird automatisch der Interrupt-Vektor auf den Datenbus gelegt. In einem späteren Abschnitt dieses Kapitels werden wir die Interrupt-Programmierung des PIO noch kennenlernen.

$\overline{RD}$ Dieser Eingang wird mit dem $\overline{RD}$-Ausgang des Mikroprozessors verbunden. Wie wir aus früheren Kapiteln wissen, ist diese Leitung immer dann logisch 0, wenn der Z80 Speicher- oder I/O-Daten liest. Daten vom PIO zur CPU werden übertragen, wenn der $\overline{IORQ}$-, der $\overline{RD}$- und der $\overline{CE}$-Eingang aktiv sind.

Beachten Sie, daß das PIO keinen $\overline{WR}$-Anschluß besitzt. Daten werden vom Z80 zum PIO geschrieben, wenn der $\overline{CE}$- und der $\overline{IORQ}$-Eingang aktiv ist, der $\overline{RD}$-Eingang aber nicht.

IEI und IEO *(Interrupt Enable In/Out)* Dies sind Eingangs- und Ausgangsleitungen der Interrupt-Prioritätskette im Z80-System. Wir werden sie später eingehender besprechen.

$\overline{INT}$ Dieser Open-Drain-Ausgang wird bei einer Interrupt-Anforderung des PIO aktiv. Wir werden später noch näher darauf eingehen.

A0-A7 Dies sind die Peripherie-I/O-Leitungen des Ports A. Hierüber kommuniziert das PIO mit der Außenwelt. A0 ist das niederwertigste Bit.

$\overline{ASTB}$ *(Register A Strobe)* Dieser Eingang dient dem Handshaking zwischen Peripheriegeräten und dem PIO-Port A. Die Arbeitsweise des Eingangs ist von der programmierten Betriebsart abhängig. Hier die möglichen Verwendungsarten:

- Wenn das Port im Nur-Ausgabe-Modus ist, zeigt die steigende Flanke am $\overline{ASTB}$-Eingang an, daß das Peripheriegerät die Daten empfangen hat.
- Im Nur-Eingabe-Modus wird die Leitung zum Abspeichern der empfangenen Daten in das Eingangsregister benutzt.
- Wenn Port A im Bidirektional-Modus programmiert ist, dient die $\overline{ASTB}$-Leitung im Zusammenhang mit den anderen Handshake-Leitungen zur Umschaltung der Datenrichtung. Wir werden später noch näher darauf zu sprechen kommen.

ARDY *(Register A Ready)* Die Arbeitsweise dieses Ausgangs ist vom programmierten Modus abhängig. Die möglichen Funktionen des Pins sind folgende:

- Wenn das Port im Nur-Ausgabe-Modus ist, zeigt diese Leitung an, daß Daten abholbereit im Ausgaberegister stehen.
- Im Nur-Eingabe-Modus zeigt die ARDY-Leitung an, daß die empfangenen Daten von der CPU gelesen wurden (Eingabepuffer leer).

- Im Bidirektional-Modus ist die Leitung aktiv, wenn Daten am Ausgaberegister des Ports verfügbar sind. Das Peripheriegerät muß jetzt die $\overline{\text{ASTB}}$-Leitung aktivieren, damit die Daten auf die Portleitungen gelegt werden.

B0-B7 Dies sind die Portleitungen des Ports B.

$\overline{\text{BSTB}}$ *(Port B Strobe)* Die Funktion dieses Eingangs entspricht der von ASTB mit folgender Ausnahme:

- Wenn Port A im Bidirektional-Modus arbeitet, werden mit der $\overline{\text{BSTB}}$-Leitung die vom Peripheriegerät kommenden Daten in das Eingangsregister des Ports A abgespeichert. Port B kann nicht im Bidirektional-Modus arbeiten.

BRDY *(Register B Ready)* Die Arbeitsweise dieses Ausgangs entspricht der von ARDY. Eine Ausnahme besteht im Bidirektional-Modus des Ports A: Die BRDY-Leitung wird logisch 1, wenn der Prozessor die Daten vom Eingangsregister des Ports A abgeholt hat und das PIO neue Daten vom Peripheriegerät empfangen kann.

Verbindung des Z80-PIO mit dem Z80-Mikroprozessor

Wir wollen jetzt sehen, wie das PIO für eine zuverlässige Kommunikation mit dem Z80-Mikroprozessor verbunden wird. Abb. 8.3 zeigt eine typische Verbindung. Diese Schaltung wollen wir hier näher untersuchen.

Die B/A-Auswahlleitung des PIO wird physikalisch mit der A0-Adreßleitung des Z80 verbunden. Die C/D-Leitung verbinden wir mit A1 des Z80. Sind die Adreßleitungen in dieser Weise angeschlossen, ergibt sich folgende Port-Definition:

A0	A1	angesprochenes Port
0	0	A Daten
0	1	B Daten
1	0	A Steuerwort
1	1	B Steuerwort

Die $\overline{\text{CE}}$-Eingangsleitung des PIO wird aus den System-Adreßleitungen A2-A7 decodiert. Die I/O-Adressen in Abb. 8.3 sind 2C, 2D, 2E und 2F. In dieser Schaltung sind die Interrupt-Leitungen nicht beschaltet. Später, bei der Behandlung der Interrupt-Arbeitsweise, werden wir auch diese Leitungen verwenden.

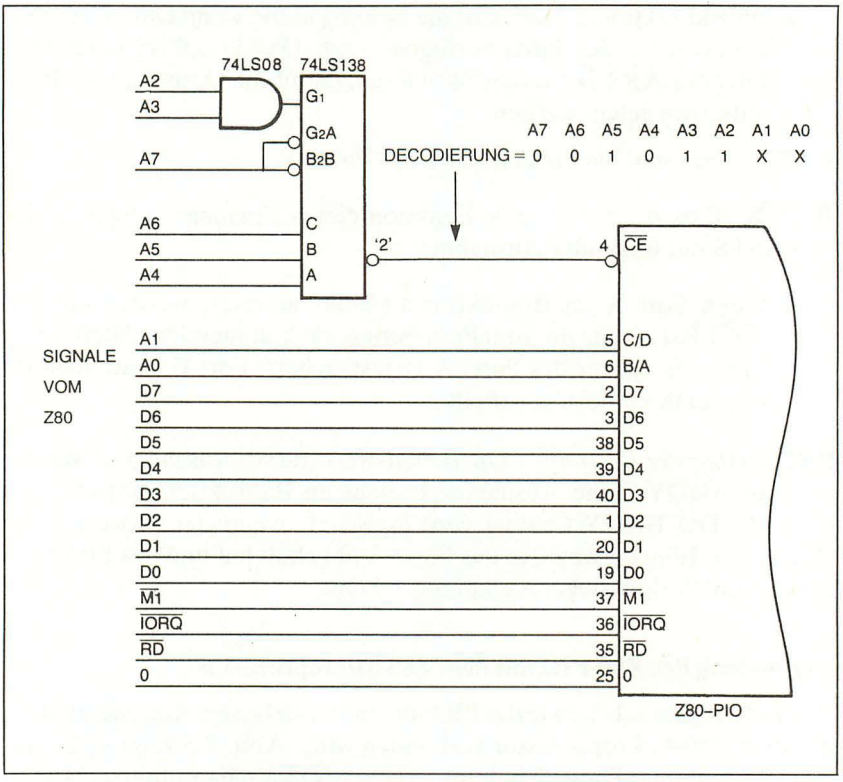


Abb. 8.3: *Physikalische Verbindung zwischen Z80 und PIO.*

Rücksetzen des PIO (Reset)

Beim Einschalten der Stromversorgung wird das PIO in den Reset-Zustand gebracht. Der Reset-Zustand bedeutet folgendes:

1. Beide Maskenregister werden für alle Datenbits zurückgesetzt. (Da wir die Maskenregister noch nicht besprochen haben, ist es hier nur wichtig zu wissen, daß sie zurückgesetzt werden.)
2. Die Port-Datenleitungen werden in einen hochohmigen Zustand gesetzt, und die READY-Handshake-Leitungen sind inaktiv. Beide Kanäle werden in den Modus 1 (Nur-Eingabe) gebracht.
3. Die Interrupt-Vektor-Register werden nicht zurückgesetzt. Diese Register beinhalten die Vektoren, die bei einer Interrupt-Bestätigung auf den Datenbus gelegt werden.

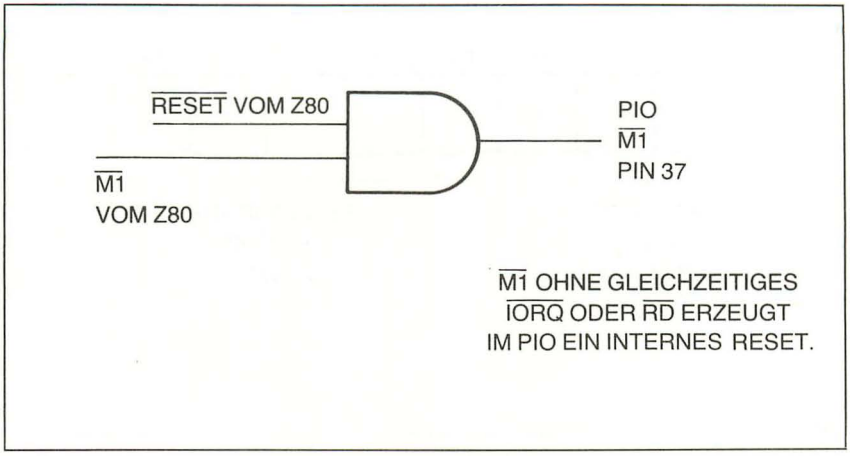


Abb. 8.4: *Hardware-Schaltung zur Erzeugung eines Reset am Z80-PIO. Das Bauteil hat keinen speziellen Reset-Eingang.*

4. Beide Interrupt-Freigabe-Flipflops werden zurückgesetzt. Das PIO ist erst interruptfähig, wenn es dafür programmiert wird.
5. Beide Ausgaberegister werden zurückgesetzt.

Der Reset-Zustand wird immer nach dem Einschalten der Versorgungsspannung eingenommen. Mit Hilfe der Schaltung in Abb. 8.4 kann auch ein externer Hardware-Reset gegeben werden. Wie aus der Schaltung hervorgeht, wird ein $\overline{M1}$-Signal erzeugt, ohne daß $\overline{RD}$ oder $\overline{IORQ}$ aktiv sind. Dieser Signalzustand tritt während der normalen Programmausführung des Z80 nicht auf.

Programmierung des PIO im Modus 0 (Nur-Ausgabe)

Als erstes wollen wir jetzt den Betriebsmodus 0 des PIO besprechen. In diesem Modus wird das Port nur als Ausgang benutzt. Beide Ports des Bausteins können unabhängig voneinander programmiert werden. In unserem Beispiel verwenden wir Port A.

Wir gehen bei unserer Betrachtung davon aus, daß das Bauteil gerade zurückgesetzt wurde. Als erstes muß das Steuerregister mit dem Moduswort programmiert werden. Die Adreßleitung A0 wird logisch 0 (für Port A) und A1 wird logisch 1 (für das Steuerregister). Das entspricht der Port-Adresse 2E. Das Datenwort für das Steuerregister ist in Abb. 8.5 zu sehen. Die unteren vier Bits sind logisch 1, womit angezeigt wird, daß das

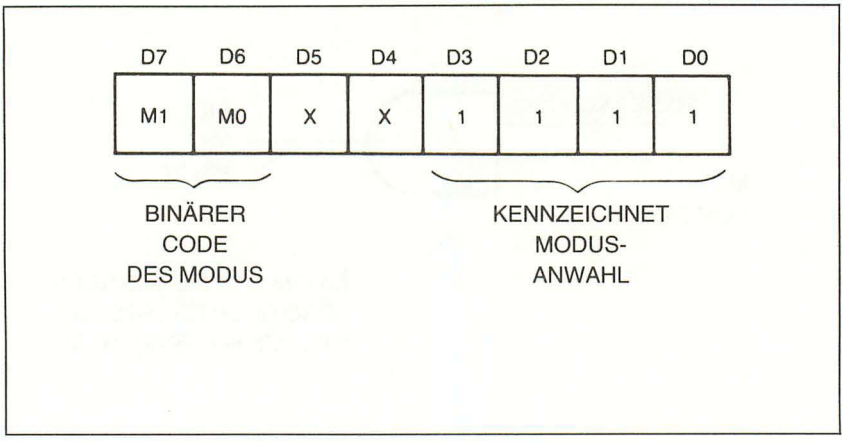


Abb. 8.5: *Modus-Wort für das Z80-PIO.*

Moduswort gesetzt wird. Bit D7 und D6 geben den gewünschten Betriebsmodus an. Daraus ergeben sich die möglichen Wörter 0F (Modus 0), 4F (Modus 1), 8F (bidirektionaler Modus) und CF (Steuermodus). Die Bits D4 und D5 werden hier ignoriert.

Das Programmieren dieses Modusworts ist alles, was gemacht werden muß, um das PIO als Ausgabeport benutzen zu können. Abb. 8.6 zeigt ein Z80-Programm, das das PIO als Ausgabeport programmiert und einen Binärwert ausgibt.

```
1800   3E 0F            LD      A,0FH
1802   D3 2E            OUT     (2EH),A        ; Ausgabe des Steuerwortes

               ; Port A = Ausgabe, B/A = 0, C/D = 1

               ; Ausgabe von 53H zum Port A

1804   3E 53            LD      A,53H
1806   D3 2C            OUT     (2CH),A        ; Ausgabe der Daten

               ; Programmieren von Port B

1808   3E 0F            LD      A,0FH
180A   D3 2F            OUT     (2FH),A        ; Ausgabe des Steuerwortes

               ; Port B = Ausgabe, B/A = 1, C/D = 1

               ; Port B kann als universelles Ausgabeport benutzt werden.
               ; Dazu werden Daten zur Port-Adresse 2DH geschrieben.
```

Abb. 8.6: *Z80-Programm zum Aufsetzen des PIO in den Modus 0 und anschließender Datenausgabe.*

Programmierung des Modus 1 (Nur-Eingabe)

In diesem Modus arbeitet das PIO als Eingabeport. Das Steuerwort zur Programmierung hierfür ist 4F. Es kann zur Port-Adresse 2E (Port A) oder 2F (Port B) geschrieben werden. Wir wollen mit Port-Adresse 2F das Port B als Eingabeport programmieren. Als Eingangsleitungen werden damit, wie in Abb.8.7 gezeigt, B0-B7 benutzt.

Werden keine Handshakes benutzt, so muß die $\overline{\text{B S T B}}$-Leitung fest auf logisch 0 liegen (siehe Abb. 8.8). Dadurch kann der Zustand der Eingangsleitungen B0-B7 direkt gelesen werden. Abb. 8.9 zeigt das Z80-Programm zur Initialisierung des PIO mit Port A als Ausgang und Port B als Eingang. Das Programm liest danach Daten vom Port B und schreibt sie zum Port A.

Wird der Modus 1 mit Handshakes benutzt, so werden die Leitungen $\overline{\text{A S T B}}$, $\overline{\text{B S T B}}$, ARDY und BRDY benötigt. Mit der $\overline{\text{B S T B}}$-Leitung lassen sich Daten in das Eingangsregister des Ports B abspeichern, und die BRDY-Leitung zeigt an, ob dieses Register leer ist, also vom Z80 gelesen wurde. Abb. 8.10 zeigt ein Blockschaltbild für die Benutzung der Handshake-Signale.

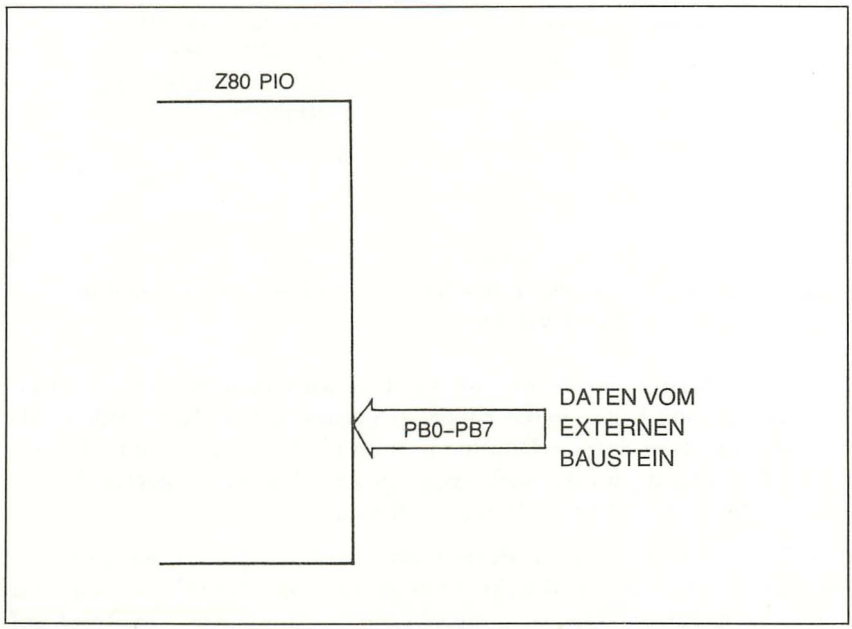


Abb. 8.7: *Blockschaltbild für die Dateneingabe über Port B.*

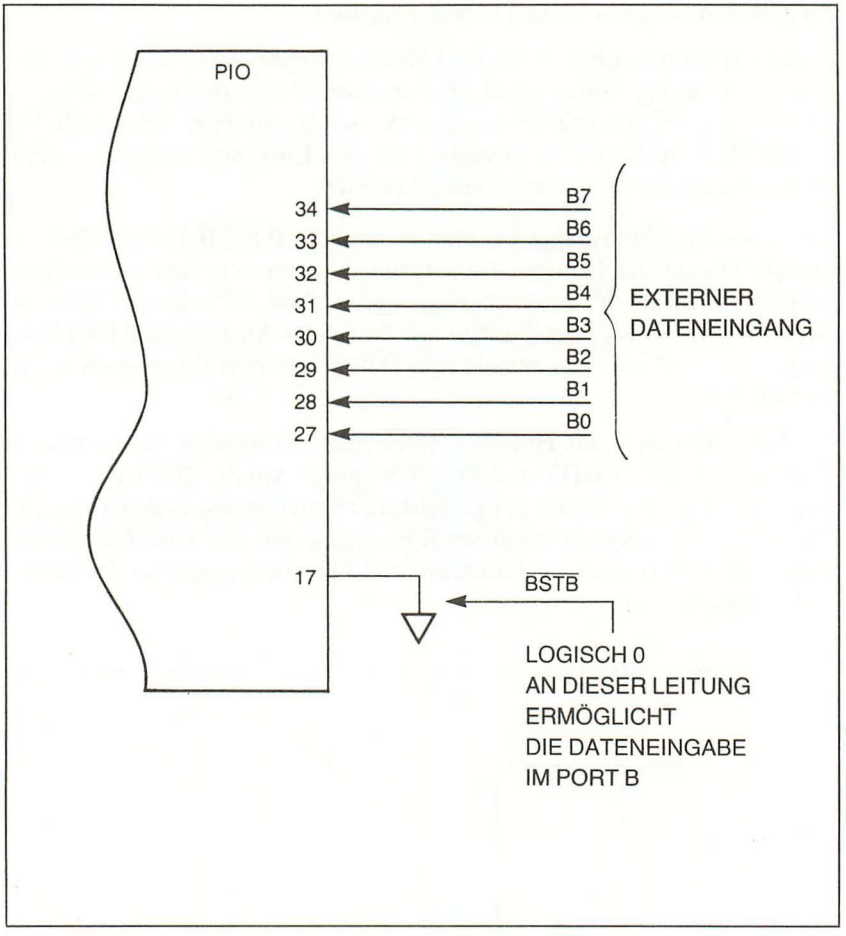


Abb. 8.8: *Wenn das Port nicht im Handshake-Modus betrieben wird, muß die $\overline{STB}$-Leitung auf logisch 0 liegen.*

Ist das PIO für Interrupts programmiert, so wird auch die Interrupt-Leitung aktiv, wenn Daten in das Eingangsregister geschrieben werden. Als Teil der Interrupt-Service-Routine werden die Daten gelesen und somit die BRDY-Leitung wieder auf logisch 1 gesetzt. Das signalisiert dem Peripheriegerät, daß es weitere Daten senden kann.

Bei der Benutzung dieser Betriebsart ist es nötig, am Anfang eine Dummy-Leseoperation durchzuführen, um die BRDY-Leitung auf logisch 1 zu setzen. Das ist nötig, da die Leitung nach dem Reset auf logisch 0 liegt.

```
1800   3E 0F            LD      A,0FH
1802   D3 2E            OUT     (2EH),A      ; Ausgabe des Steuerwortes

               ; Port A ist jetzt ein Ausgabeport

1804   3E 4F            LD      A,4FH
1806   D3 2F            OUT     (2FH),A      ; Ausgabe des Steuerwortes

               ; Port B ist jetzt ein Eingabeport

1808   DB 2D            IN      A,(2DH)      ; Lese Daten vom Port B
180A   D3 2C            OUT     (2CH),A      ; Ausgabe der Daten zum Port A

                        END
```

Abb. 8.9: *Z80-Programm zur Initialisierung beider PIO-Ports mit anschließender Datenübertragung von Port B auf Port A.*

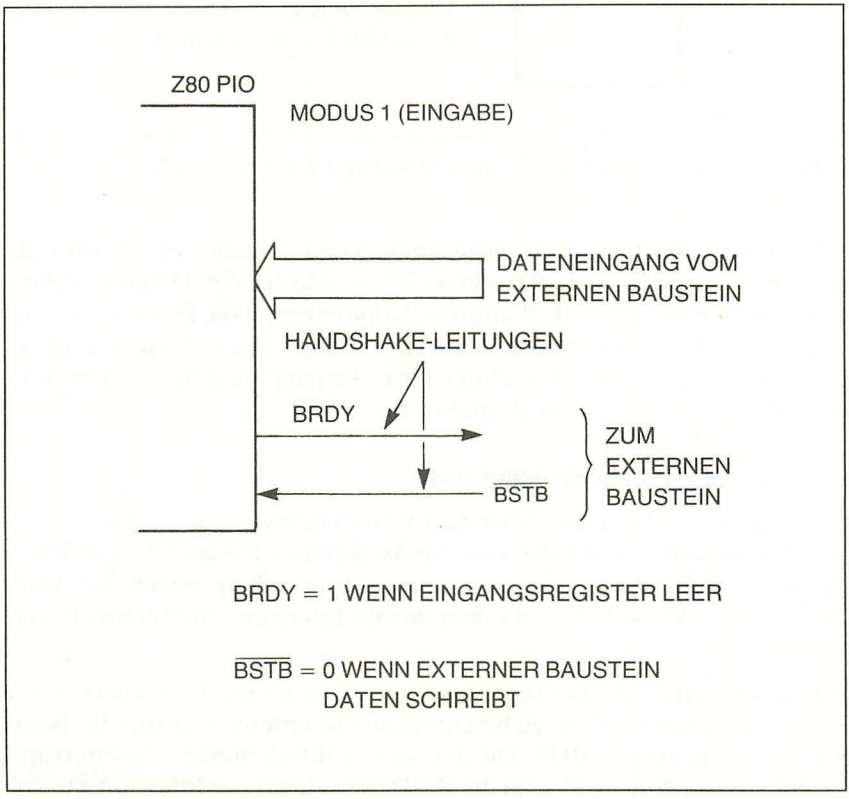


Abb. 8.10: *Benutzung der Handshake-Leitungen bei der Dateneingabe.*

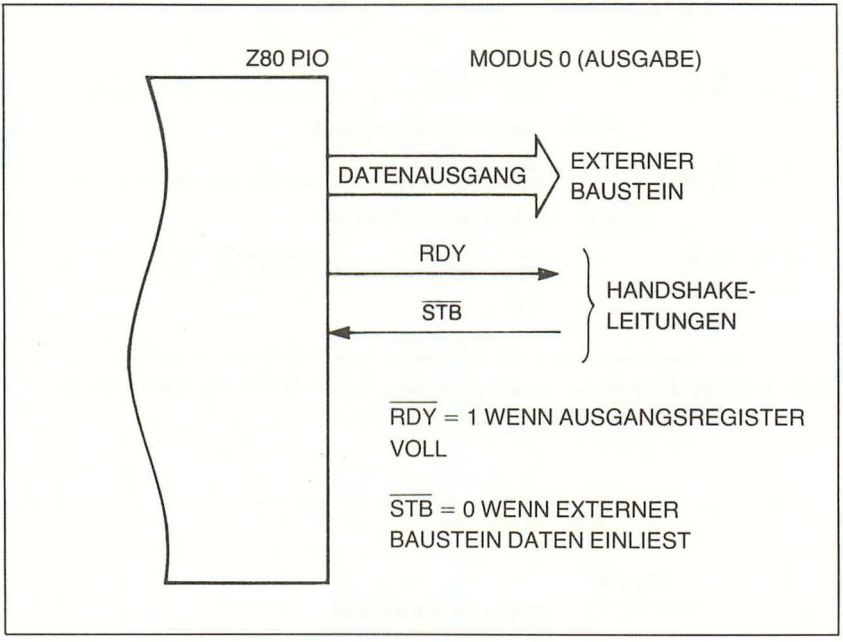


Abb. 8.11: *Benutzung der Handshake-Leitungen bei der Datenausgabe.*

Im Modus 0 können die Handshake-Signale ebenfalls benutzt werden. Hierbei wird die RDY-Leitung logisch 1, sobald der Z80 Daten zum Ausgabeport geschrieben hat. Wenn das Peripheriegerät die Daten liest, quittiert es dies durch Ansprechen der STB-Leitung. Dadurch wird die RDY-Leitung wieder logisch 0, und der $\overline{\text{INT}}$-Ausgang wird aktiv. Abb. 8.11 zeigt die Benutzung dieser Betriebsart.

Aufsetzen des Interrupt-Steuerworts

Im vorigen Abschnitt haben wir das PIO im Modus 0 und 1 programmiert. Im Zusammenhang mit den Handshake-Signalen haben wir auch Interrupts erwähnt, doch über ihre genaue Anwendung haben wir nicht gesprochen. Wir wollen jetzt sehen, wie die Interrupts programmiert werden.

Um das Interrupt-Steuerwort zu setzen, ist es nötig, das Steuerregister des entsprechenden Ports zu beschreiben. Die unteren vier Bits des Interrupt-Steuerworts sind 0111. Die oberen vier Bits definieren die Interrupt-Arbeitsweise. Abb. 8.12 zeigt die Bit-Definition für das Interrupt-Steuerwort.

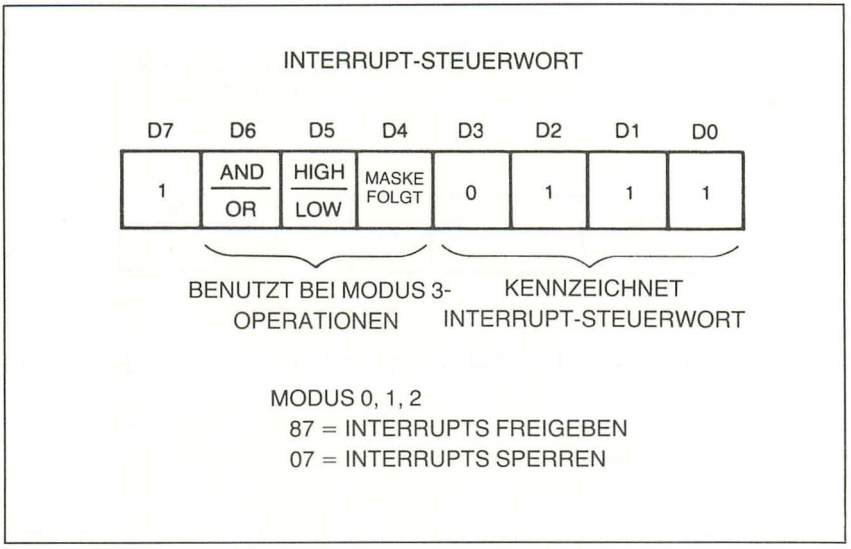


Abb. 8.12: *Die unteren vier Bits mit dem Wert 0111 zeigen an, daß es sich um das Interrupt-Steuerwort handelt.*

Wenn das Bit D7 des Interrupt-Steuerworts logisch 1 ist, wird das Interrupt-Freigabe-Flipflop des PIO gesetzt und das Bauteil kann Interrupts erzeugen. Wird das Bit auf logisch 0 gesetzt, sperrt das die Interrupts. Tritt eine Interrupt-Bedingung ein, ohne daß das Freigabe-Flipflop gesetzt ist, so wird der Interrupt bei einer späteren Freigabe generiert. Anstehende Interrupts werden gelöscht, wenn Bit D4 des Interrupt-Steuerworts logisch 1 ist.

Die Bits D6, D5 und D4 (mit obiger Ausnahme) werden nur im Modus 3 des PIO benutzt. Wir werden diesen Modus später besprechen. Für Modus 0 und 1 wird nur die Interrupt-Freigabe benötigt.

Das Interrupt-Freigabe-Flipflop wird mit folgenden Wörtern gesetzt bzw. zurückgesetzt:

Interrupts freigeben 87H
Interrupts sperren 07H

Es ist auch möglich, die Interrupt-Freigabe zu steuern, ohne die anderen Parameter (beim Modus 3) zu setzen, wenn folgende Wörter benutzt werden:

Interrupts freigeben 83H
Interrupts sperren 03H

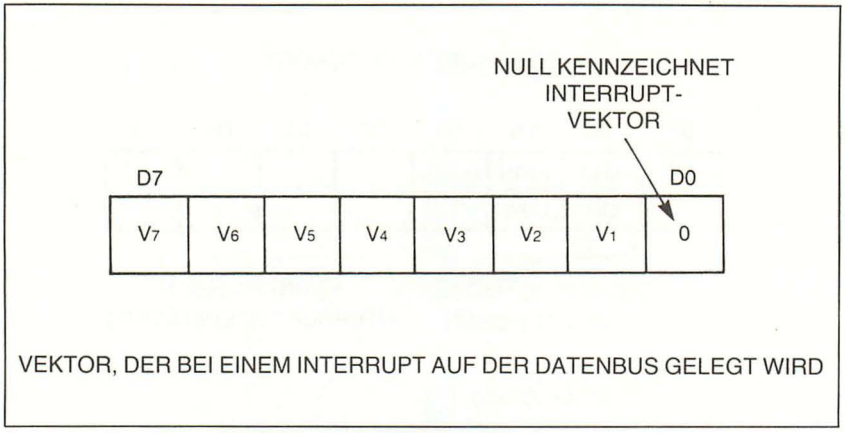


Abb. 8.13: *Bit-Definition für den Interrupt-Vektor.*

```
1800   3E 76            LD     A,76H          ; Interrupt-Vektor
1802   D3 2E            OUT    (2EH),A        ; Ausgabe des Vektors

1804   3E 0F            LD     A,0FH
1806   D3 2E            OUT    (2EH),A        ; Ausgabe des Steuerwortes

            ; Jetzt wird das Interrupt-Steuerwort geschrieben

1808   3E 87            LD     A,87H
180A   D3 2E            OUT    (2EH),A        ; eine Möglichkeit zur
                                              ; Freigabe der Interrupts

            ; Es folgt eine andere Möglichkeit zum Einschalten
            ; der Interrupts:

180C   3E 83            LD     A,83H
180E   D3 2E            OUT    (2EH),A        ; Interrupt-Freigabe

            ; Man braucht nur eine der beiden Möglichkeiten zum
            ; Einschalten der Interrupts verwenden.

                        END
```

Abb. 8.14: *Z80-Programm zum Aufsetzen des Interrupt-Vektors 76H und zum Einschalten des Interrupt-Systems.*

Werden die Interrupts freigegeben, so muß zur Benutzung des Z80-Interrupt-Modus 2 auch ein Interrupt-Vektor programmiert sein. Der Interrupt-Vektor wird geladen, indem das Steuerregister in der Form von Abb. 8.13 beschrieben wird. Abb. 8.14 gibt einen Auszug aus einem Z80-Programm wieder, das den Interrupt-Vektor aufsetzt und die Interrupts freigibt. Der benutzte Vektor ist 76H.

Zusammenfassung des Timings von Modus 0 und 1

Jetzt, da wir an Beispielen die Arbeitsweise von Modus 0 und 1 untersucht haben, dürfte das Verständnis der Timing-Diagramme in Abb. 8.15 keine Schwierigkeiten mehr bereiten.

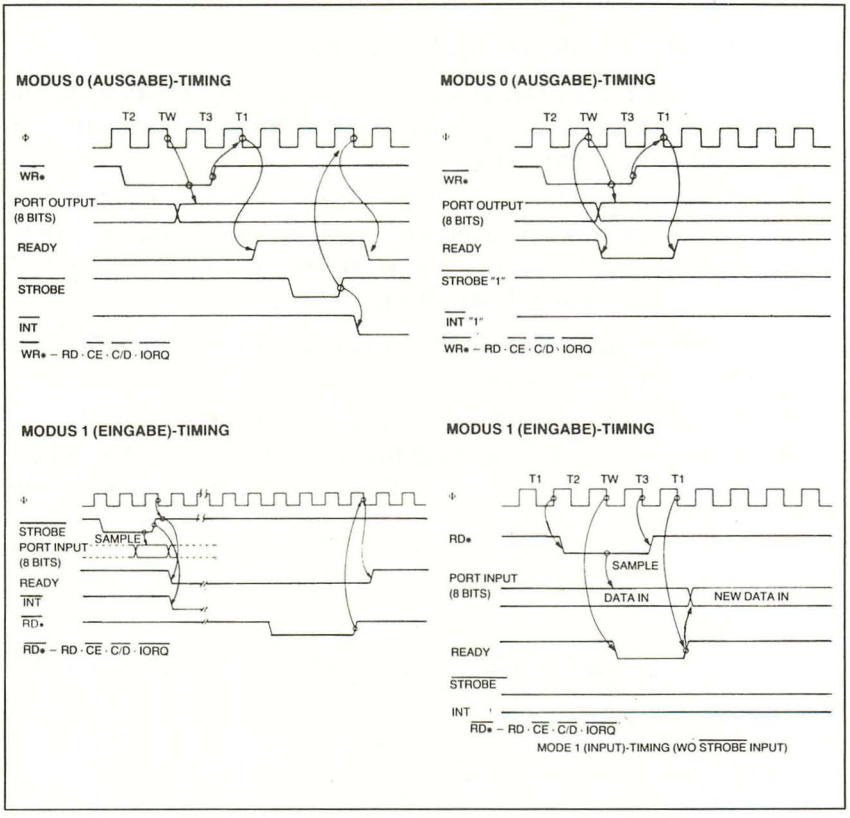


Abb. 8.15: *Timing-Diagramme für den Modus 0 und 1 des Z80-PIO.*

Benutzung des PIO im Modus 2 (Bidirektionaler Modus)

Der Bidirektional-Modus ist eine Kombination von Modus 0 und Modus 1. Es werden dabei alle vier Handshake-Leitungen des Bauteils benötigt. Deshalb ist dieser Modus auch nur für Port A verfügbar. Port B kann zur gleichen Zeit nur im Steuermodus arbeiten. Der Interrupt-Vektor für die Ausgabe wird in Port A programmiert, der für die Eingabe in Port B.

Abb. 8.16 zeigt das Timing für einen typischen Datentransfer mit dem PIO im Bidirektional-Modus. Lassen Sie uns nun einige Details des Timings untersuchen. Die Handshake-Leitungen in Port A dienen der Ausgabesteuerung und die Handshake-Leitungen in Port B der Eingabesteuerung. Sie arbeiten genauso, als wären die entsprechenden Ports im Modus 0 bzw. Modus 1 programmiert. Wir beginnen mit dem Transfer vom PIO zum Peripheriegerät:

1. Das PIO wird im Bidirektional-Modus programmiert.
2. Der Z80 schreibt ein Datenwort zum Ausgangsregister Port A. Die ARDY-Leitung wird dadurch logisch 1 und zeigt dem Peripheriegerät an, daß es Daten abholen kann.
3. Das Peripheriegerät spricht den $\overline{\text{ASTB}}$-Eingang des PIO an. Dadurch werden die Ausgabedaten auf die Portleitungen A0-A7 gelegt. Wenn das Peripheriegerät die Daten eingelesen hat, wird die $\overline{\text{ASTB}}$-Leitung wieder deaktiviert (logisch 1).
4. Die $\overline{\text{INT}}$-Leitung des PIO wird aktiv und signalisiert dem Z80, daß die Ausgabedaten abgeholt wurden.
5. Wir wollen jetzt annehmen, daß das Peripheriegerät Daten an das PIO schickt. Die $\overline{\text{ASTB}}$-Leitung ist in Ruhestellung (logisch 1). Das Peripheriegerät legt Daten an die Portleitungen A0-A7 und spricht die

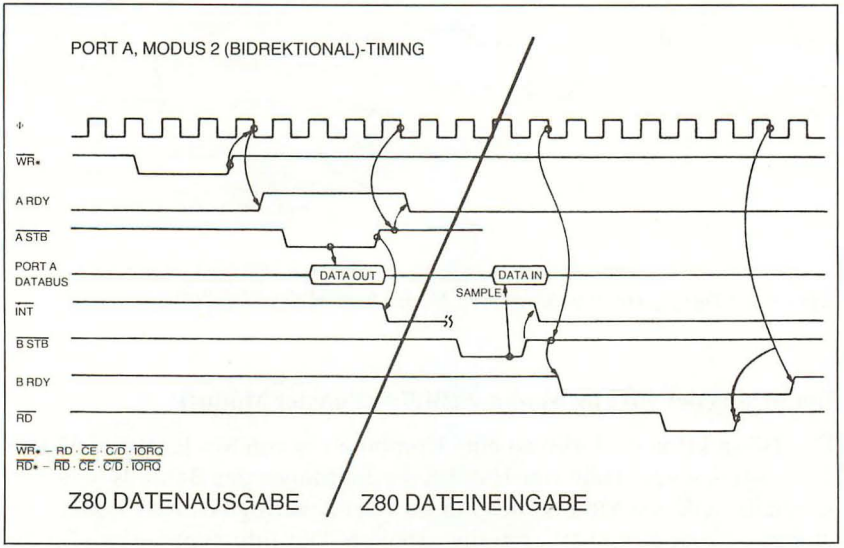


Abb. 8.16: *Timing beim Bidirektional-Modus des PIO.*

$\overline{\text{BSTB}}$-Leitung an. Wenn das $\overline{\text{BSTB}}$-Signal von logisch 0 auf logisch 1 zurückgeht (siehe Abb. 8.16), wird der $\overline{\text{INT}}$-Ausgang aktiv. Das signalisiert dem Z80, daß das Peripheriegerät Daten in das Eingangsregister von Port A geschrieben hat. Der Z80 kann sie jetzt lesen.

6. Die 0-1-Flanke des $\overline{\text{BSTB}}$-Signals setzt auch den BRDY-Ausgang auf logisch 0. Das meldet dem Peripheriegerät, daß die Daten vom Z80 noch nicht gelesen wurden und keine weiteren Daten gesendet werden dürfen, bis BRDY wieder logisch 1 ist. Das passiert, sobald der Z80 die Daten vom Port A gelesen hat.

Wie wir gesehen haben, wird immer dann eine Interrupt-Anforderung durchgeführt, wenn das Peripheriegerät Daten gesendet oder empfangen hat. Der Z80 muß natürlich wissen, um welche der beiden Operationen es sich gehandelt hat. Das kann dadurch geschehen, daß zwei verschiedene Interrupt-Vektoren in Port A und Port B geladen werden.

Wenn das Peripheriegerät Daten zum PIO gesendet hat, wird bei der Interrupt-Bestätigung der Vektor von Port B auf den Datenbus gelegt. Wenn das Peripheriegerät Daten empfangen hat, wird der Vektor von Port A benutzt.

Es ist natürlich möglich, daß der im Modus 3 arbeitende Port B ebenfalls einen Interrupt auslöst. Soll dies verhindert werden, dürfen am Port B die Interrupts nicht freigegeben werden. Obwohl wir den Modus 3 noch nicht besprochen haben, wollen wir festhalten, daß im gemischten Betrieb von Modus 2 und 3 die Interrupts zu Problemen führen können.

Abb. 8.17 zeigt ein Z80-Programm zum Aufsetzen des PIO in einer Modus 2-Anwendung. Für die Dateneingabe und Ausgabe werden Interrupt-Service-Routinen benutzt. Wir gehen davon aus, daß das Peripheriegerät die Handshake-Leitungen richtig bedient.

```
2700                OUTDAT  EQU     2700H       ; Speicherplatz für Ausgabedaten
2701                INDAT   EQU     OUTDAT+1    ; Speicherplatz für Eingabedaten

                ; Dieses Programm setzt Port A als Ein/Ausgabeport auf.

1800    3E 8F               LD      A,8FH       ; Moduswort
1802    D3 2E               OUT     (2EH),A     ; Ausgabe zum Port A
1804    3E 00               LD      A,00H       ; Interrupt-Vektor
1806    D3 2E               OUT     (2EH),A     ; Ausgabe zum Port A

1808    3E CF               LD      A,0CFH      ; Moduswort
180A    D3 2F               OUT     (2FH),A     ; schreibe zum Port B
```

Abb. 8.17: *Z80-Programm zum Aufsetzen des PIO in einer Anwendung mit Modus 2. Die Dateneingabe und Ausgabe erfolgt über Interrupt-Service-Routinen.*

```
                ; Durch das Moduswort wird Port B in den Modus 3
                ; gesetzt. Die Bit-Definition folgt.

180C   3E FF            LD      A,0FFH          ; alle Bits werden Eingänge
180E   D3 2F            OUT     (2FH),A         ; schreibe zum Port B
1810   3E 17            LD      A,17H           ; Int.-Steuerwort
1812   D3 2E            OUT     (2EH),A         ; Int. aus, Maske folgt
1814   3E FF            LD      A,0FFH          ; alle Bits ausmaskiert
1816   D3 2F            OUT     (2FH),A         ; schreibe zum Port B

1818   3E 02            LD      A,02H           ; Interrupt-Vektor
181A   D3 2F            OUT     (2FH),A         ; schreibe zum Port B

181C   3E 3A            LD      A,3AH           ; Vektor-Adresse oberes Byte
181E   ED 47            LD      I,A

                ; Der Interrupt-Vektor für Port A ist 3A00 und für Port B 3A02.
                ; Die Adressen der Service-Routinen sind FC81 und FD00.

                ; Die Interrupts Port A sind für die Datenausgabe
                ; Die Interrupts Port B sind für die Dateneingabe

1820   ED 5E            IM      2               ; Z80 Interrupt-Modus 2
1822   3E 83            LD      A,83H           ; PIO-Inter. einschalten
1824   D3 2E            OUT     (2EH),A         ; Port A
1826   D3 2F            OUT     (2FH),A         ; Port B
1828   FB               EI                      ; Z80 Interrupts ein

1829   C3 1829  LOOP:   JP      LOOP            ; warte auf Interrupts

                ; Es folgt die Interrupt-Vektor-Tabelle und Beispiele für
                ; die Interrupt-Service-Routinen

                         ORG     3A00H
3A00   FC81              DEFW    0FC81H          ; Interrupt-Vektor Port A
3A02   FD00              DEFW    0FD00H          ; Interrupt-Vektor Port B

                ; Hier die Service-Routinen

                         ORG     0FC81H

FC81   3A 2700           LD      A,(OUTDAT)      ; Lade Accu mit Ausgabedaten
FC84   D3 2C             OUT     (2CH),A         ; schreibe zum Port A
FC86   FB                EI                      ; Interrupts ein
FC87   ED 4D             RETI                    ; Rücksprung

                         ORG     0FD00H

FD00   DB 2D             IN      A,(2DH)         ; Lese Daten vom Port B
FD02   32 2701           LD      (INDAT),A       ; speichere Daten
FD05   FB                EI                      ; Interrupts ein
FD06   ED 4D             RETI                    ; Rücksprung

                         END
```

Abb. 8.17: *Fortsetzung des Z80-Programms.*

Benutzung des PIO im Modus 3

Der Modus 3 wird in einer Nicht-Handshake-Umgebung eingesetzt. Die acht Bits jedes Ports können in beliebiger Anordnung als Eingangs- oder Ausgangsleitungen dienen. Zum Beispiel könnten die Bits B0, B3 und B6 Eingänge und die übrigen Bits des Ports B Ausgänge sein. Dies kann auf folgende Weise bewerkstelligt werden:

Als erstes muß der Programmierer das Moduswort in das Steuerregister schreiben. Für den Modus 3 hat das Moduswort den Wert 0CFH. Das unmittelbar folgende Wort definiert, wie die acht Bits des Ports benutzt werden sollen. Eine logische 1 markiert die entsprechende Bit-Position als Eingang. Eine logische 0 markiert sie als Ausgang. Die folgenden Befehle setzen das PIO in den Modus 3 und definieren die Bits B0, B3 und B6 als Eingänge. Die Leitungen B1, B2, B4, B5 und B7 werden zu Ausgängen.

```
LD    A,0CFH      ; Lade CF in den Akku
OUT   (2FH),A     ; Selektiere Modus 3 für Port B
LD    A,49H       ; Bit D0, D3 und D6 = 1
OUT   (2FH),A     ; Setze B0, B3 und B6 als Eingänge
```

Das PIO ist jetzt für die gewünschte I/O-Bit-Definition aufgesetzt. Der Programmierer kann jetzt das Port lesen oder zum Port schreiben. Wenn vom Port gelesen wird, werden die logischen Werte der als Eingänge definierten Leitungen übernommen. Die als Ausgänge definierten Bit-Positionen enthalten den bei der letzten OUT-Anweisung geschriebenen Wert. Das bedeutet, daß alle Bits bei einer Eingabeoperation gelesen werden, jedoch nur die als Eingänge definierten Leitungen die Eingangsdaten wiedergeben.

Eine andere Besonderheit des Modus 3 sind seine Interrupt-Fähigkeiten. Das PIO kann im Falle einer bestimmten logischen Kombination der Eingangsleitungen eine Interrupt-Anforderung auslösen. Dazu muß das Interrupt-Steuerwort entsprechend aufgesetzt werden. Erinnern wir uns, daß wir bei der Besprechung des Interrupt-Steuerworts für Modus 0, 1 und 2 die Bits D6, D5 und D4 ignoriert haben. Jetzt machen wir von diesen Bits Gebrauch.

Bit 6 des Interrupt-Steuerwortes gibt an, in welcher Weise die Eingangsbits logisch verknüpft werden sollen. Die zwei Möglichkeiten sind AND und OR. Wenn die AND-Verknüpfung gewählt wurde, müssen alle selektierten Bits im Aktivzustand sein, um einen Interrupt auszulösen. Bei der OR-Verknüpfung muß mindestens ein Bit im Aktivzustand sein.

Die AND- und OR-Funktion ist nicht einfach eine Boolsche Verknüpfung der Leitungen, sondern eine Verknüpfung der Aktivzustände.

Der Aktivzustand kann entweder bei logisch 1 oder bei logisch 0 sein. Alle Eingänge werden jedoch auf denselben Aktivzustand überprüft. Das bedeutet, alle Eingänge werden entweder auf logisch 1 oder auf logisch 0 überprüft.

Bit D5 bestimmt den Aktivzustand der selektierten Portleitungen. Wenn D5 logisch 1 ist, werden die Portleitungen auf logisch 1 überprüft. Wenn D5 logisch 0 ist, werden sie auf logisch 0 überprüft.

Nachdem die Verknüpfungsart und der Aktivzustand festeht, muß noch definiert werden, welche der acht Bits überprüft werden sollen. Wenn das Bit D4 des Interrupt-Steuerworts logisch 1 ist, wird das folgende Byte als Maskenwort benutzt. Abb. 8.18 zeigt das Maskenwort. Wenn der Programmierer ein bestimmtes Bit kontrollieren will, muß das entsprechende Maskenbit logisch 0 sein. Bits, die für die Verknüpfung nicht benutzt werden sollen, werden mit logisch 1 „ausmaskiert".

Um die Arbeitsweise des Modus 3 zu verdeutlichen, wollen wir sie an einem Beispiel untersuchen. In unserem Beispiel gehen wir folgendermaßen vor:

1. Das Port B wird in den Modus 3 aufgesetzt.
2. Wir programmieren die Datenleitungen B2, B3 und B4 als Eingänge. Damit sind B0, B1, B5, B6 und B7 Ausgänge.
3. Wir wollen für den Interrupt die Leitungen B3 und B4 kontrollieren. Dazu maskieren wir alle anderen Leitungen aus.
4. Der Aktivzustand von B3 und B4 soll logisch 0 sein.
5. Der Interrupt soll ausgelöst werden, wenn beide Bits logisch 0 sind. Das ist eine AND-Verknüpfung der Aktivzustände.
6. Der Interrupt-Vektor soll 08 sein.

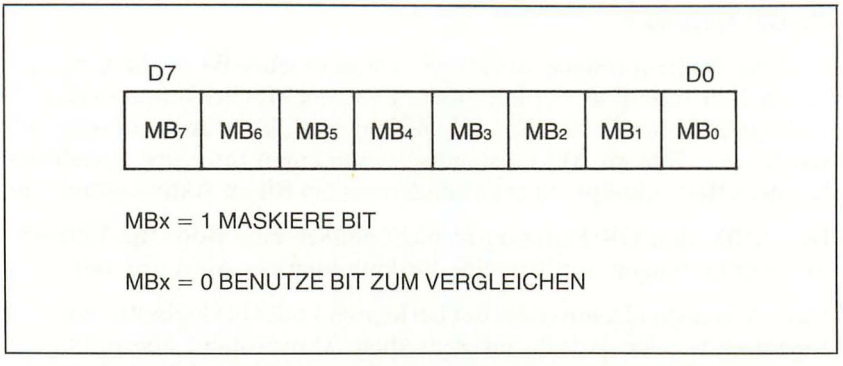


Abb. 8.18: *Bit-Definition des Maskenworts. Dieses Byte informiert das PIO elektrisch darüber, welche Bits kontrolliert werden sollen.*

Wir gehen in diesem Beispiel davon aus, daß das PIO wie schon zuvor die I/O-Adressen 2C, 2D, 2E und 2F belegt. Abb. 8.19 zeigt das Blockschaltbild der Verbindung zum Peripheriegerät.

Hier nun das Z80-Programm zum Aufsetzen des PIO:

```
LD    A,11001111B
OUT   (2FH),A       ; Setze Modus 3 für Port B
LD    A,00011100B
OUT   (2FH),A       ; Setze Bit B2, B3 und B4 als
                      Eingang
LD    A,00001000B
OUT   (2FH),A       ; Interrupt-Vektor 08
LD    A,11010111B
OUT   (2FH),A       ; INT, AND, LOW, Maske folgt
LD    A,11100111B
OUT   (2FH),A       ; Bit B3 und B4 sind nicht maskiert
```

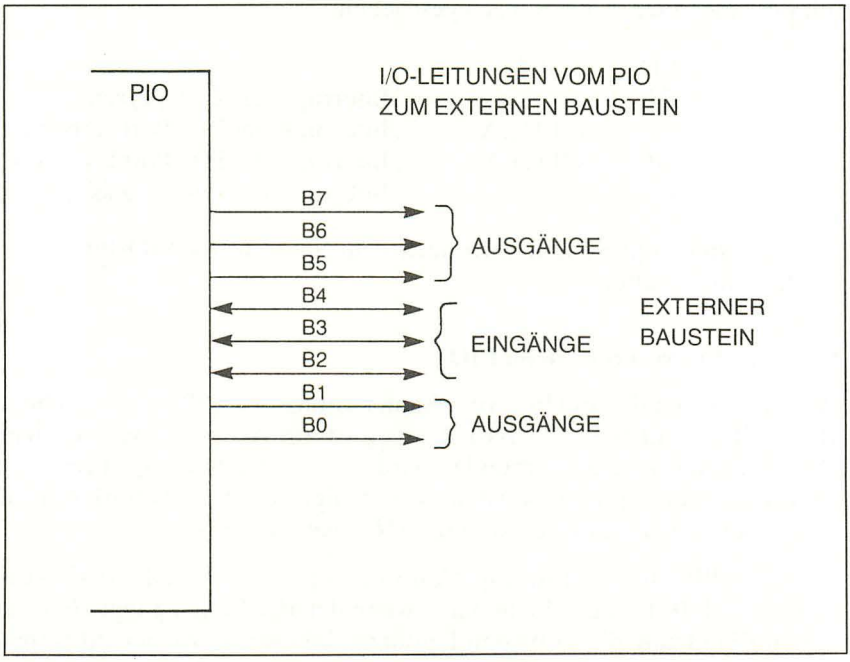


Abb. 8.19: *Verbindung des PIO mit einem Peripheriegerät. Beachten Sie, daß einige I/O-Leitungen Eingänge und andere Ausgänge sind. Das Z80-PIO wird im Steuermodus betrieben.*

Das vorstehende Programm setzt das PIO in der beschriebenen Weise auf. Als weiterer Programmteil muß noch die Interrupt-Service-Routine geschrieben werden. (In Kapitel 4 finden Sie die genaue Beschreibung der Z80-Interrupts.)

Eine nützliche Eigenschaft des PIO ist es, daß es den Datenbus beobachtet und die RETI-Anweisung decodiert und damit automatisch den Interrupt zurücknimmt. Erinnern wir uns, daß es normalerweise die Aufgabe des Programmierers ist, in der Interrupt-Service-Routine die Interrupt-Anforderung explizit zurückzunehmen. Davon ist der Programmierer bei Benutzung des PIO befreit.

Interrupt-Freigabe und Interrupt-Sperrung

Beim ersten Aufsetzen des PIO könnte eine ungewollte Interrupt-Anforderung auftreten. Dies kann problematisch werden, da eine wirkliche Interrupt-Bedingung noch nicht existiert. Es ist deshalb ratsam, vor dem Aufsetzen des PIO die Interrupts zu sperren. Der beste Weg ist der, zuerst die Z80-Interrupts zu sperren und danach die des PIO. Das kann mit den folgenden Anweisungen geschehen:

```
LD    A,03H
DI                  ; Interrupts des Z80 sperren
OUT   (2EH),A       ; Interrupts des PIO Port A sperren
OUT   (2FH),A       ; Interrupts des PIO Port B sperren
EI                  ; Interrupts des Z80 freigeben
```

Das Z80-Interrupt-System wird nach dem Sperren der PIO-Interrupts wieder eingeschaltet.

Interrupt-Priorisierung beim PIO

Wir wollen jetzt die drei Interrupt-Steuerleitungen des PIO untersuchen: $\overline{\text{INT}}$, IEO sind IEI. Die $\overline{\text{INT}}$-Leitung ist ein Ausgang, der mit dem $\overline{\text{INT}}$-Eingang des Z80 verbunden wird. IEI ist der Interrupt-Freigabe-Eingang. Ist er logisch 1, so ist das PIO fähig, Interrupt-Anforderungen zu machen. Ist er logisch 0, so ist der $\overline{\text{INT}}$-Ausgang gesperrt.

Wenn das PIO in einer Interrupt-Anforderung steckt, wird der IEO-Ausgang logisch 0. Das geschieht auch, wenn der IEI-Eingang logisch 0 ist. Durch Benutzung dieser beiden Leitungen können bis zu vier PIOs (mit zusätzlicher Hardware auch mehr) in einem Interrupt-Prioritäts-Schema verkettet werden. Abb. 8.20 zeigt eine solche Verkettung, wobei beim Baustein mit der höchsten Priorität der IEI-Eingang auf logisch 1 liegt.

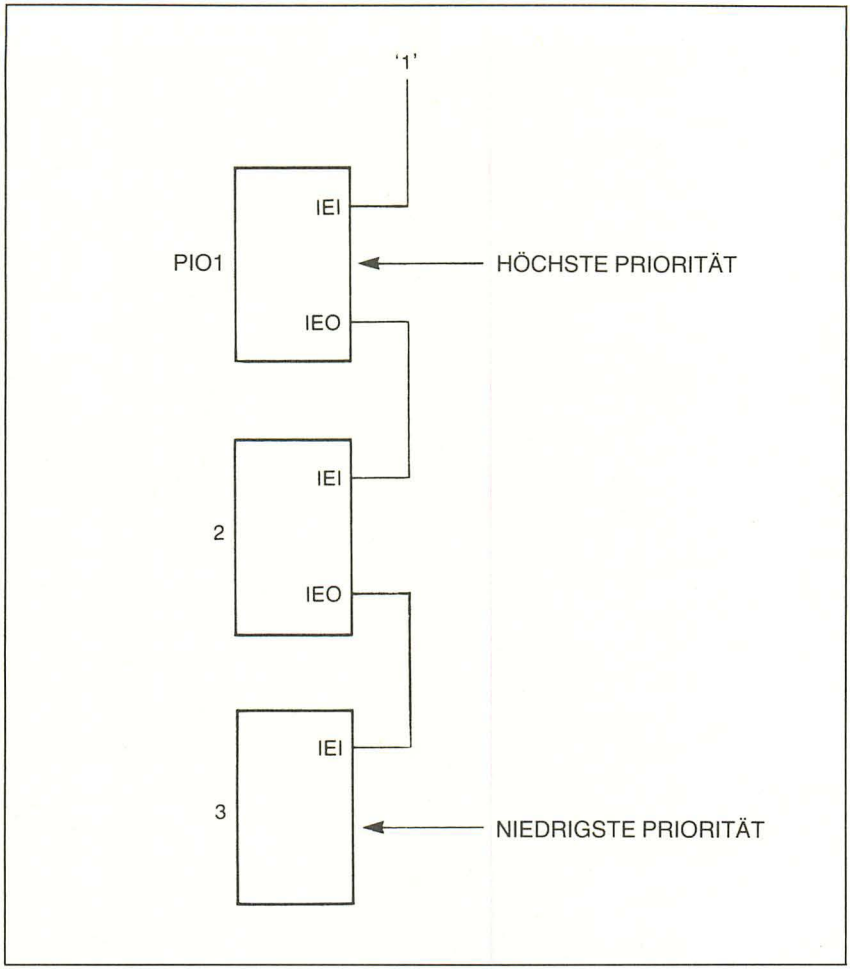


Abb. 8.20: *Interrupt-Prioritätskette (interrupt daisy chain) mit Z80-PIOs.*

Zusammenfassung

In diesem Kapitel lernten wir das Z80-PIO kennen. Wir begannen mit der Untersuchung des Blockschaltbildes und behandelten dann jeden einzelnen Betriebsmodus. Für jeden Modus stellten wir Programmierbeispiele vor.

Die zur Verbindung mit dem Z80 erforderliche Hardware wurde vorgestellt und besprochen. Die vermittelten Informationen dürften für die allermeisten Systeme anwendbar sein.

Kapitel 9

Benutzung des Z80-CTC

Einführung

In diesem Kapitel wollen wir den als Z80-CTC bekannten Zähler-Zeitgeber-Baustein vorstellen. Der CTC ist ein LSI-Chip, das speziell für den Einsatz in Z80-Mikroprozessor-Systemen konzipiert ist. Der Baustein ist besonders vielseitig. Wir werden seine Fähigkeiten nach und nach in diesem Kapitel kennenlernen.

Als erstes werden wir das Blockschaltbild des CTC vorstellen und dann die Register und die Hardware-Verbindungen besprechen. Am Ende des Kapitels sollten Sie in der Lage sein, den Z80 in Ihrem System einzusetzen und zu programmieren.

Blockschaltbild des CTC

Der CTC (Counter-Timer-Chip) verdankt seinen Namen den zwei Grundfunktionen Zählung und Timing. Er besteht aus vier unabhängigen Zähler-Timer-Kanälen in einem 28-poligen DIP-Gehäuse. Abb. 9.1 zeigt das Blockschaltbild des CTC. Wir wollen es einmal untersuchen.

Wie wir in Abb. 9.1 sehen, hat der CTC vier weitgehend identische Kanäle, Ch0, Ch1, Ch2 und Ch3. Drei dieser Kanäle sind mit jeweils zwei Signalleitungen verbunden: dem Zählerausgang und dem Takt/Trigger-Eingang. Der Kanal 3 hat nur eine Eingangsleitung. Der Grund dafür liegt in der Begrenzung durch das 28-Pin-Gehäuse.

Weiter ist in Abb. 9.1 die interne Steuerlogik zu sehen. Dieser Block sorgt für ein sauberes Timing des internen Datenverkehrs.

Die im Blockschaltbild zu sehende Interrupt-Steuerlogik ist besonders leistungsfähig. Wir werden später die Funktion dieser Logik genauer untersuchen.

Der letzte Block in Abb. 9.1 beinhaltet das CPU-Bus-I/O-Interface. Hier wird der Datenverkehr zwischen dem Z80 und dem CTC abgewickelt. Das geschieht über acht Daten- und sechs Steuerleitungen.

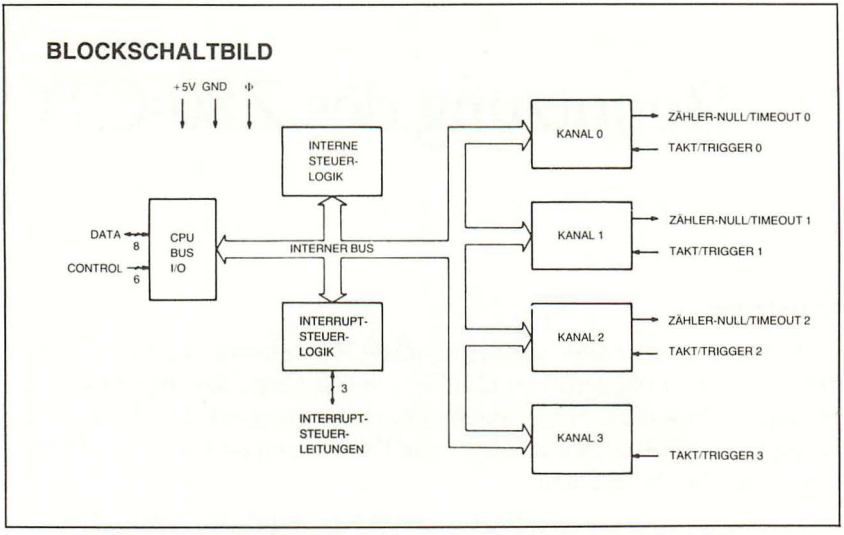


Abb. 9.1: *Blockschaltbild des Z80-CTC. Beachten Sie, daß die vier Zählerkanäle weitgehend identisch sind.*

Genauere Betrachtung eines Kanals

In Abb. 9.1 können wir sehen, wie alle Blöcke zusammenarbeiten. Um den CTC einsetzen und programmieren zu können, müssen wir jedoch mehr ins Detail gehen. Wir suchen uns dafür den als Kanal 0 bezeichneten Block aus. (Beachten Sie, daß dieser Kanal stellvertretend für die anderen Kanäle betrachtet werden kann. Eine Ausnahme bildet nur Kanal 3, dem die Ausgangsleitung fehlt.)

Abb. 9.2 zeigt das Blockschaltbild eines Kanals. Wir können in dieser Abbildung sehen, daß ein einzelner Kanal aus einem Kanal-Steuerregister, einem Zeitkonstanten-Register und einem Abwärtszähler zusammengesetzt ist. Diese wollen wir hier näher untersuchen.

Der Programmierer schreibt Informationen in das Kanal-Steuerregister. Dieses Register definiert die Betriebsart des Kanals.

Neben dem Kanal-Steuerregister liegt das Zeitkonstanten-Register. Dieses Register hat eine Breite von 8 bit. Es enthält eine Binärzahl im Bereich von 00H bis FFH. Diese Zahl wird zum Aufsetzen des Abwärtszählers benutzt.

Der Abwärtszähler hat ebenfalls eine Breite von 8 bit. Sein Ausgang wird als „zero count/timeout" bezeichnet. Bei jedem Nulldurchgang des Zäh-

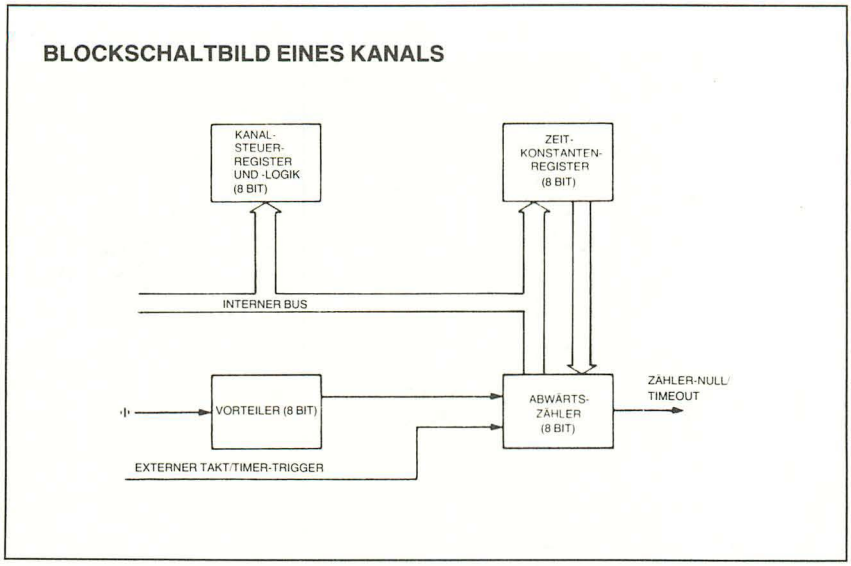


Abb. 9.2: *Blockschaltbild eines einzelnen Zählerkanals. Dieses Schaltbild gilt für alle vier Kanäle des CTC.*

lers wird der Ausgang aktiv. Vor dem Abwärtszähler befindet sich der 8-Bit-Vorteiler. Er kann den Systemtakt, von der Programmierung abhängig, durch 16 oder durch 256 teilen.

Ein anderer Eingang des Abwärtszählers ist die „external clock/timer trigger"-Leitung. Sie kann als direkter Takteingang oder als Freigabeeingang für den vom Vorteiler kommenden Takt fungieren. Die Funktion dieser Leitung ist von der Programmierung des CTC abhängig.

Pinbelegung des Z80-CTC

Bevor wir mit der Programmierung des CTC beginnen, soll sichergestellt werden, daß wir ihn mit dem Z80-Mikroprozessor verbinden können. Abb. 9.3 zeigt die Pinbelegung und die Signalbezeichnungen für die Ein- und Ausgänge des Bausteins. Wir wollen uns jede Leitung ansehen und ihre Funktion definieren.

Signal-Definition beim CTC

D7-D0 Dies sind die Daten-I/O-Leitungen des Bausteins zur Verbindung mit dem Z80-Datenbus. Das niederwertigste Bit ist D0.

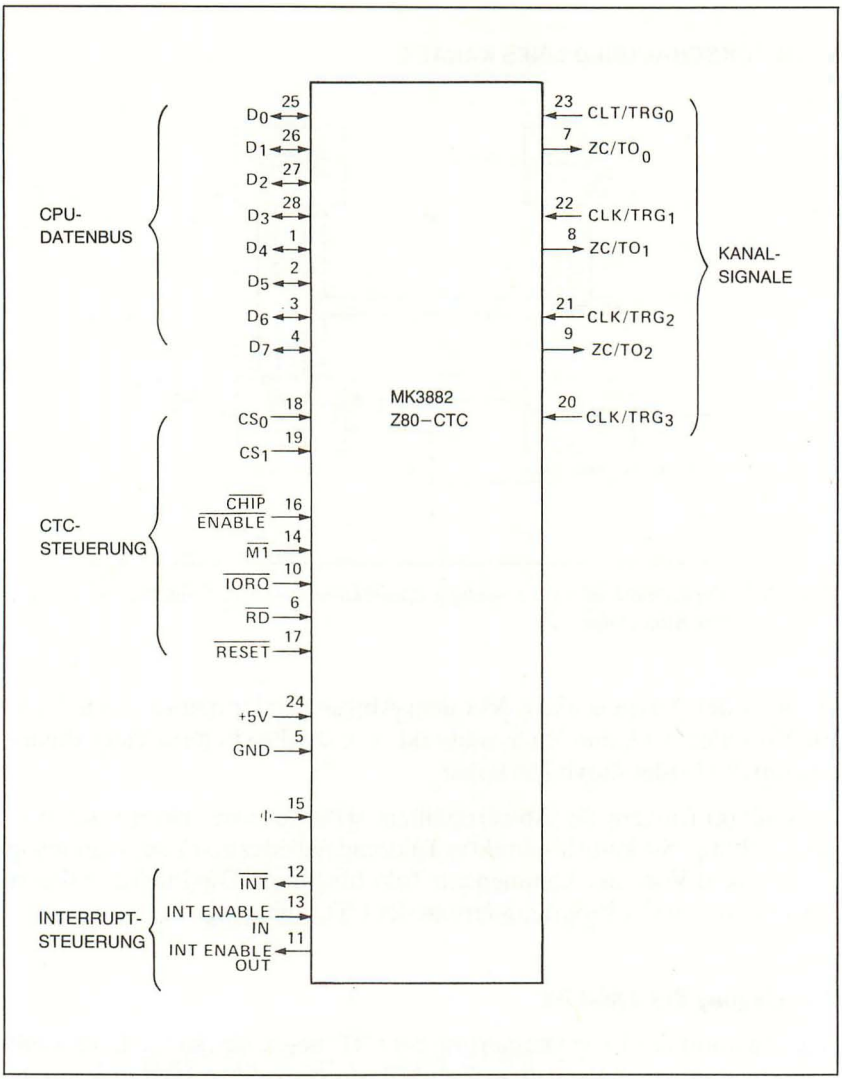


Abb. 9.3: *Pinbelegung und Signaldefinition beim Z80-CTC.*

CS1,CS0 Dies sind die Kanal-Auswahlleitungen. Sie bilden die binäre Adresse des Kanals, mit dem während einer I/O-Operation kommuniziert wird. Im allgemeinen werden die Adreßleitungen A1 und A0 des Z80 mit diesen Leitungen verbunden. Die Wahrheitstafel für diese Eingänge ist folgende:

CS1	CS0	aktiver Kanal
0	0	Null
0	1	Eins
1	0	Zwei
1	1	Drei

$\overline{CE}$ *(chip enable)* Dies ist der Baustein-Freigabe-Eingang. Wenn diese Leitung logisch 0 ist, kann der CTC elektrisch mit dem Z80 kommunizieren. Der $\overline{CE}$-Eingang wird normalerweise aus den Adreßleitungen A7-A2 decodiert.

CLOCK Dieser Eingang wird mit dem Systemtakt verbunden. Der Takt wird u.a. zur Steuerung der internen Datentransfers benötigt.

$\overline{M1}$ Die $\overline{M1}$-Eingangsleitung des CTC wird mit dem $\overline{M1}$-Ausgang des Z80 verbunden. Dieses Signal wird in Verbindung mit dem $\overline{RD}$-Signal zur Erkennung von Befehlscode-Leseoperationen benötigt. In Verbindung mit dem $\overline{IORQ}$-Signal zeigt es außerdem die Interrupt-Bestätigung des Z80 an.

$\overline{IORQ}$ Dieser Eingang wird mit dem $\overline{IORQ}$-Ausgang des Z80 verbunden. Er zeigt an, daß der Z80 eine I/O-Operation durchführt.

$\overline{RD}$ Der $\overline{RD}$-Eingang wird mit der $\overline{RD}$-Leitung des Z80 verbunden. Die Leitung informiert den CTC darüber, daß die CPU eine Speicher- oder I/O-Leseoperation durchführt. Es soll hier angemerkt werden, daß der CTC keinen speziellen Schreibeingang besitzt. Wenn der Z80 Daten zum CTC schreibt, ist die $\overline{CE}$- und die $\overline{IORQ}$-Leitung logisch 0 und die $\overline{RD}$-Leitung logisch 1.

IEI *(interrupt enable input)* Wenn dieser Eingang logisch 1 ist, wird dem CTC ermöglicht, die $\overline{INT}$-Ausgangsleitung anzusprechen. Ist IEI logisch 0, so ist die $\overline{INT}$-Leitung des Bausteins abgeschaltet.

IEO *(interrupt enable output)* Wenn dieser Eingang logisch 1 ist, bedeutet das, daß der Baustein gerade keinen Interrupt anfordert oder zugeteilt bekommen hat. Die Leitung ist außerdem mit dem IEI-Eingang verknüpft und dient somit als Verbindungsleitung in der Interrupt-Prioritätskette.

$\overline{INT}$ Dies ist der Interrupt-Anforderungs-Ausgang des CTC. Er ist als Open-Drain-Ausgang ausgeführt und kann mit den $\overline{INT}$-Ausgängen anderer Bausteine direkt verbunden werden (wired AND).

$\overline{\text{RESET}}$ Ein logischer 0-Pegel auf diesem Eingang setzt den CTC in einen definierten Anfangszustand. In diesem Zustand werden alle Zähler gestoppt und alle Interrupt-Freigabe-Bits zurückgesetzt. Alle Ausgangsleitungen werden hochohmig.

CLK/TRG_3-CLK/TRG_0 Dies sind die externen Takt- und Timer-Trigger-Eingänge.

ZC/TO_2-ZC/TO_0 Zeitablauf-Ausgänge, aktiv logisch 1.

Verbindung des CTC mit dem Z80

Nachdem wir die einzelnen Anschlüsse des CTC kennen, wollen wir ihn nun mit dem Z80-Mikroprozessor verbinden. Wir gehen davon aus, daß keine Datenbus-Pufferung benötigt wird. (Wenn Sie mit dieser Thematik nicht vertraut sind, kann Ihnen das Kapitel 2 helfen, zu entscheiden, ob Ihre Anwendung Datenbus-Puffer benötigt.)

Bei der Beschaltung des CTC berücksichtigen wir auch die Interrupt-Fähigkeiten des Chips. Es ist jedoch auch möglich, den CTC ohne Benutzung der Interrupts zu betreiben.

Abb. 9.4 zeigt die Verbindung zwischen Z80 und CTC. Wir wollen jetzt einige wichtige Punkte dieser Schaltung besprechen.

Als erste Verbindung zwischen Z80 und CTC sehen wir den Datenbus. Wir verbinden dazu einfach D0-D7 des Z80 mit D0-D7 des CTC. Über diese Leitungen findet der physikalische Datenaustausch zwischen beiden Bausteinen statt.

Die Adreß-Ausgangsleitungen A0 und A1 des Z80-Adreßbusses sind mit den Eingängen CS0 und CS1 des CTC verbunden. Der Baustein-Freigabe-Eingang $\overline{\text{CE}}$ wird aus den Adreßleitungen A2-A7 decodiert. In Abb. 9.4 belegt der CTC die I/O-Adressen 40H, 41H, 42H und 43H. Die komplette Decodierung für die vier I/O-Ports entsteht aus der logischen Kombination der Adreßleitungen A0-A7.

$\overline{\text{M1}}$, $\overline{\text{IORQ}}$ und $\overline{\text{RD}}$ sind die nächsten zu verbindenen Signale. Die entsprechenden Pins am Z80 haben die gleiche Bezeichnung.

Die Stromversorgung des CTC liegt zwischen +5V und GND. Der Takteingang (Pin 15) wird mit dem Z80-Takteingang (Pin 6) verbunden. Bei der Standardversion ist die maximale Taktfrequenz 2,5 MHz.

Jetzt muß noch der Open-Drain-INT-Ausgang des CTC mit dem INT-Eingang des Z80 verbunden werden. Die Leitung wird über einen 10 kOhm-Widerstand auf 5V hochgezogen.

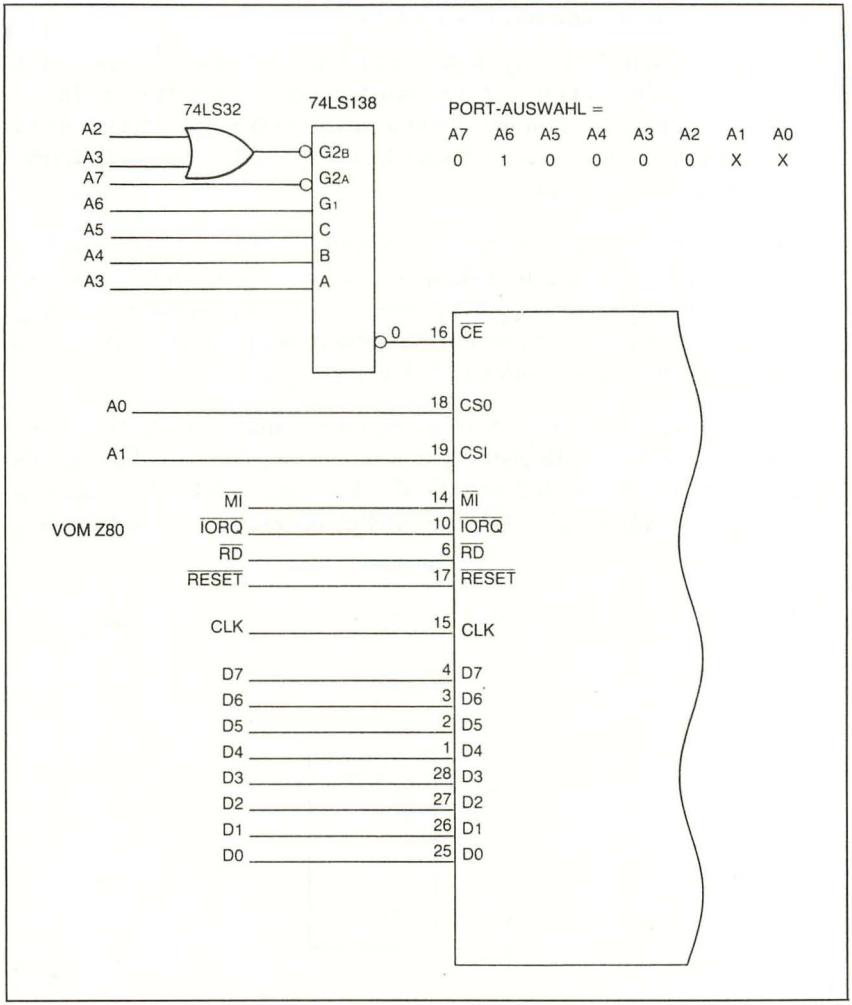


Abb. 9.4: *Physikalische Verbindung zwischen Z80-CTC und Z80-Mikroprozessor. Die Decodierung des $\overline{CE}$-Eingangs des CTC erfolgt aus den Adreßleitungen A2-A7.*

Wird der CTC nicht in einem verketteten Interrupt-Schema eingesetzt, so muß der IEI-Eingang lediglich auf Vcc gelegt werden. Dadurch ist der Baustein immer interruptfähig. Den IEO-Ausgang kann man unbeschaltet lassen. Soll der CTC innerhalb einer Interrupt-Prioritätskette liegen, so muß der IEI-Eingang mit dem IEO-Ausgang eines anderen Bauteils verbunden werden. Das wird in Abb. 9.5 gezeigt.

Übersicht über den Zählermodus des CTC

Jetzt haben wir den CTC physikalisch mit dem Z80-Mikroprozessor verbunden, und wollen sehen, wie der Baustein programmiert wird. In diesem Abschnitt untersuchen wir die Hardware- und Software-Details für den Zählermodus. Abb. 9.6 zeigt das Blockschaltbild eines im Zählermodus arbeitenden CTC-Kanals.

Wie wir in Abb. 9.6 sehen, ist der Zähler zum Zählen eines externen Taktes aufgesetzt. Dieser externe Takt wird dem Zähler über den CLK/TRG-Eingang zugeführt. Zum Beispiel könnte die Anzahl bestimmter Ereignisse gezählt werden. Der Mikroprozessor könnte dann beim Überschreiten eines Grenzwertes irgendeine Aktion auslösen.

Um einen Abwärtszähler des CTC zu benutzen, muß eine Zeitkonstante in das Zeitkonstanten-Register geschrieben werden. Sie kann einen Maximalwert von 0FFH haben. Die Zeitkonstante bildet den Anfangswert für den Abwärtszähler. Im Nulldurchgang des Zählers wird der Aus-

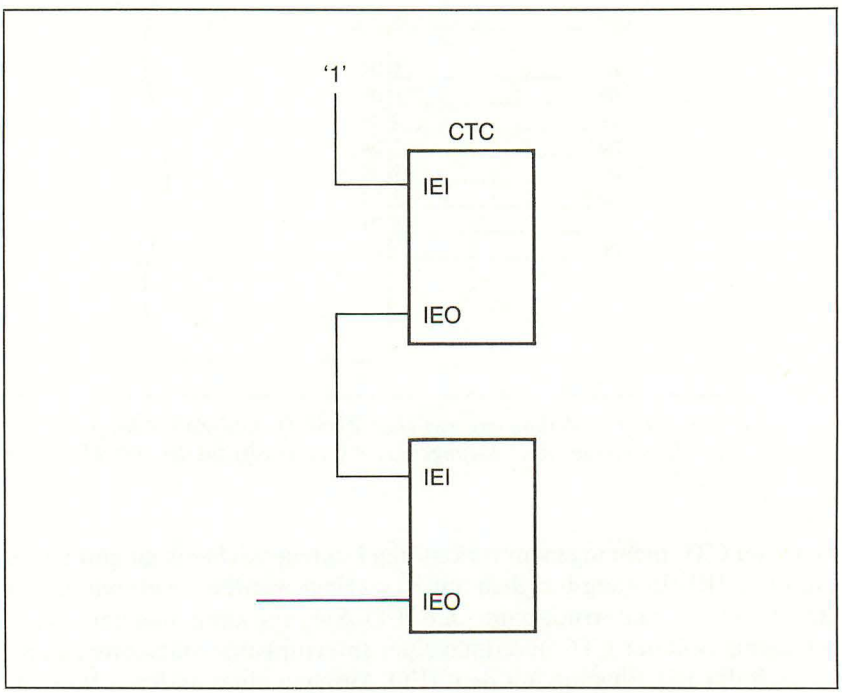


***Abb. 9.5**: Benutzung des CTC in einer Interrupt-Prioritätskkette (daisy chain).*

gang aktiv. Später werden wir genau zeigen, wie das Zeitkonstanten-Register geladen wird. Im Moment ist nur wichtig zu wissen, was im CTC allgemein realisiert werden muß.

Bevor der Kanal im Zählermodus arbeiten kann, muß er dafür programmiert werden. Dazu muß ein bestimmtes Datenwort in das Kanal-Steuerregister geschrieben werden.

Nachdem das Steuerregister aufgesetzt ist, startet der Abwärtszähler mit dem Zählen des externen Taktes. Dabei kann wahlweise die positive oder die negative Flanke berücksichtigt werden.

Ein wichtiger Punkt dabei ist es, daß der Takt jeweils bei der positiven Flanke des Systemtaktes eingelesen wird. Deshalb kann der externe Takt niemals eine höhere Frequenz haben als der Systemtakt an Pin 15 des CTC. Tatsächlich wird in der Bausteinspezifikation die externe Taktfrequenz auf den halben Wert der Systemtaktfrequenz begrenzt.

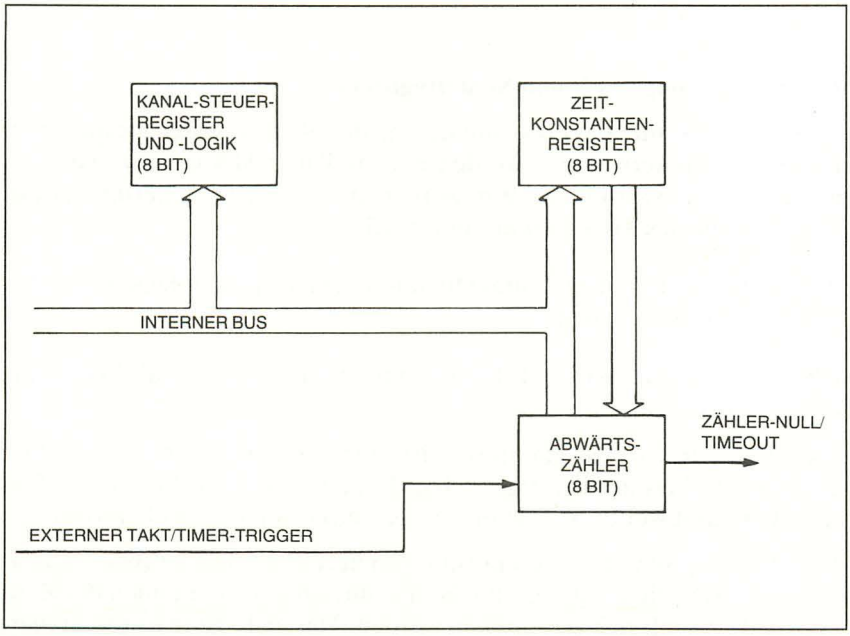


Abb. 9.6: *Blockschaltbild eines CTC-Kanals im Zählermodus.*

Der Zähler zählt nun abwärts. Sobald er den Wert 0 erreicht, wird die Zeitkonstante neu geladen, und der Ausgangspin wird etwa für die Dauer einer Periode des Systemtaktes logisch 1. Sind Interrupts freigegeben, so wird eine Interrupt-Anforderung an den Z80 generiert.
Der Zählvorgang wird nicht unterbrochen, sondern mit der neu geladenen Zeitkonstante weitergeführt. Dadurch können bis zur Interrupt-Verarbeitung noch weitere Takte gezählt werden, und kein Ereignis geht verloren.

Wird während des Zählens das Zeitkonstanten-Register vom Programmierer neu beschrieben, so wird der neue Wert erst beim nächsten Nulldurchgang berücksichtigt. Will der Programmierer die laufende Operation abbrechen und mit einer neuen Zeitkonstante fortfahren, so muß er das Steuerregister entsprechend programmieren.

Sowohl das Steuerregister als auch das Zeitkonstanten-Register müssen nach dem Einschalten des Systems oder nach einem Reset erst einmal geladen werden.

Zu jedem Zeitpunkt kann der augenblickliche Zählerstand gelesen werden. Das wird durch eine einfache Leseanweisung vom entsprechenden Kanal gemacht.

Programmierung des Kanal-Steuerregisters

Lassen Sie uns nun die Programmierung des Steuerregisters besprechen. Einige der Steuerwortbits werden nur im Timer-Modus, den wir noch nicht kennen, verwendet. Wir werden diese Bits detaillierter bei der Besprechung des Timer-Modus untersuchen.

Abb. 9.7 zeigt die Bit-Definition für das Steuerwort. Im folgenden wollen wir jedes Bit besprechen.

D0 Dieses Bit muß logisch 1 sein, damit das Byte als Kanal-Steuerwort interpretiert wird.

D1 *(Kanal-Reset)* Wenn dieses Bit logisch 1 ist, wird der Zähler gestoppt. Keines der Bits irgendeines Kanalregisters wird geändert. Der Kanal arbeitet weiter, sobald eine neue Zeitkonstante geladen wurde.

D2 Eine logische 1 in diesem Bit informiert den CTC darüber, daß das folgende Byte die Zeitkonstante ist. Es müssen also zum Laden der Zeitkonstante zwei Bytes geschrieben werden. Das erste Byte ist das Steuerwort (mit Bit D2 gesetzt) und das zweite ist die Zeitkonstante.

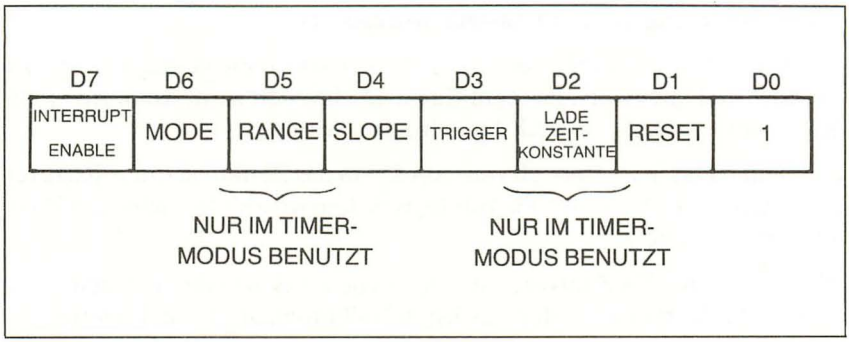


Abb. 9.7: *Bit-Definition des Kanal-Steuerregisters für den Z80-CTC.*

D3 Dieses Bit wird nur im Timer-Modus gebraucht. Ist es logisch 1, so startet der Timer mit einem externen Trigger-Impuls. Ist das Bit logisch 0, startet der Timer sobald die Zeitkonstante geladen ist.

D4 Dieses Bit definiert die aktive Flanke des externen TRG/CLK-Eingangs. Die Definition ist folgende:

Timer-Modus:	D4 = 1	positive Flanke startet Timer
	D4 = 0	negative Flanke startet Timer
Zähler-Modus:	D4 = 1	positive Flanke dekrementiert Zähler
	D4 = 0	negative Flanke dekrementiert Zähler

D5 Dieses Bit wird nur im Timer-Modus benutzt. Es bestimmt den Wert des Vorteilers. Eine logische 1 bedeutet eine Teilung durch 256, eine logische 0 eine Teilung durch 16.

D6 Ist dieses Bit logisch 1, so wird der entsprechende Kanal in den Zählermodus geschaltet. Der Abwärtszähler wird vom CLK/TRG-Eingang getaktet. Ist das Bit logisch 0, so arbeitet der Kanal im Timer-Modus und der Abwärtszähler wird vom Ausgang des Vorteilers getaktet. Die Periode am Zählerausgang ist dann gleich: Taktperiode * Vorteilerwert * Zeitkonstante.

D7 Wenn dieses Bit logisch 1 ist, werden Interrupts von diesem Kanal freigegeben. Dazu muß das Interrupt-Vektor-Register aufgesetzt sein. Ist Bit D7 logisch 0, so werden die Interrupts gesperrt.

Programmierung des Zeitkonstanten-Registers

Der Kanal kann erst arbeiten, wenn die Zeitkonstante geladen ist. Sie ist das nächste Datenwort, das zum Kanal geschrieben wird, wenn zuvor ein Steuerwort mit D2 = logisch 1 geschrieben wurde.

Es können Daten im Bereich von 0 bis 255 in das Zeitkonstanten-Register geschrieben werden. Sind alle Bits logisch 0, so wird als tatsächlicher Wert mit 256 gearbeitet.

Wenn während des Zählvorgangs eine neue Zeitkonstante geladen wird, so wird ihr Wert erst bei dem nächsten Nulldurchgang berücksichtigt.

Programmierung des Interrupt-Vektors

Abb. 9.8 zeigt die Bit-Definition für den Interrupt-Vektor. Der CTC unterstützt den Interrupt-Modus 2 des Z80. Für das ganze Chip braucht nur ein Interrupt-Vektor programmiert zu werden. In Abb. 9.8 ist zu sehen, daß Bit D0 immer logisch 0 ist. Damit wird dem CTC angezeigt, daß es sich nicht um ein Steuerwort, sondern um den Interrupt-Vektor handelt. Die Bits D1 und D2 werden bei einer Interrupt-Bestätigung automatisch gesetzt. Ihre logische Kombination bestimmt den Vektor des entsprechenden Kanals. Deshalb braucht der Benutzer nur einen Vektor pro Chip zu laden.

Die oberen 5 Bits des Vektors werden vom Benutzer gesetzt. Abhängig vom unterbrechenden Kanal setzt dann der CTC die Bits D1 und D2. D0 wird immer logisch 0 gesetzt. Als Beispiel für die Vergabe der vier Interrupt-Vektoren gehen wir einmal davon aus, der Benutzer hätte den Vektor 58 zum Kanal 0 geschrieben.

Das setzt die Interrupt-Vektor-Bits wie folgt:

0 1 0 1 1 0 0 0

Die entsprechenden Kanal-Interrupt-Vektoren sind dann:

Kanal 0 = 0 1 0 1 1 0 0 0
Kanal 1 = 0 1 0 1 1 0 1 0
Kanal 2 = 0 1 0 1 1 1 0 0
Kanal 3 = 0 1 0 1 1 1 1 0

Es ist durchaus möglich, daß mehrere Kanäle des CTC Interrupt-Anforderungen stellen. Die interne Priorität dafür ist vordefiniert. Kanal 0 hat die höchste Priorität, Kanal 3 hat die niedrigste.

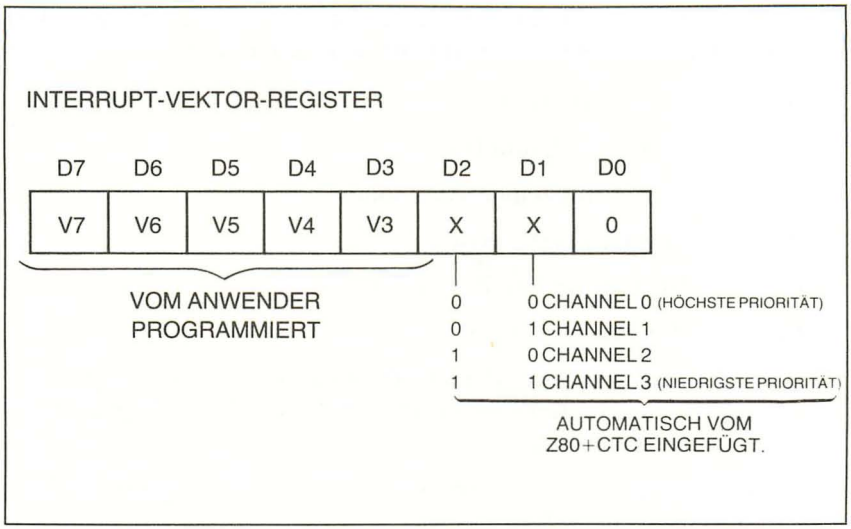


Abb. 9.8: *Bit-Definition des CTC-Interrupt-Vektors.*

Programmierung des CTC im Zählermodus

Lassen Sie uns nun an einem Beispiel den Zählermodus des CTC untersuchen. Wir gehen zuerst von einem nicht-Interrupt-getriebenen System aus. Danach untersuchen wir das gleiche System mit Benutzung von Interrupts. Abb. 9.9 zeigt das Blockschaltbild für unser Beispiel. Wir zählen die Fehlermeldungen von einem zu testenden System. Jedesmal, wenn ein Fehler auftritt, bekommt der CTC einen Zählimpuls. Der gesamte Aufbau arbeitet folgendermaßen:

1. Das System zählt Impulse. Gezählt wird mit den positiven Flanken am externen Takteingang des CTC-Kanals.

2. Der CTC-Kanal 1 wird benutzt.

3. Der CTC wird im Abfragemodus (polling) betrieben. Wenn der Zähler 46 Impulse gezählt hat, löst der Mikroprozessor irgendwelche Aktionen aus.

Als erstes müssen wir die Bits des Steuerregisters definieren. Wir wissen aus der Schaltung Abb.9.4, daß die I/O-Port-Adressen 40H, 41H, 42H

und 43H sind. Da wir Kanal 1 benutzen, müssen wir zur Adresse 41H schreiben. Das folgende sind die Bits für das Steuerregister:

Bit 7 = 0 Sperre Interrupts

Bit 6 = 1 Wähle Zählermodus

Bit 5 = 0 Nicht benutzt im Zählermodus

Bit 4 = 1 Positive Flanke zählt

Bit 3 = 0 Nicht benutzt im Zählermodus

Bit 2 = 1 Zeitkonstante folgt

Bit 1 = 1 Rücksetzen des Zählers. Dieser Zustand wird nicht gespeichert, sondern generiert lediglich einen Reset-Impuls für diesen Kanal.

Bit 0 = 1 Zeigt an, daß dies ein Steuerwort und kein Interrupt-Vektor ist.

Das Steuerwort wird vom Z80 in folgender Weise zum CTC geschrieben:

```
LD    A,57H
OUT   (41H),A     ; Ausgabe des Steuerworts an den CTC
```

Als nächstes muß die Zeitkonstante zum Port 41H geschrieben werden. Es ist möglich, zwischendurch andere Wörter in die Ports 40H, 42H und 43H zu schreiben. Das nächste Wort zum Port 41H wird jedoch vom CTC als Zeitkonstante für den Kanal 1 interpretiert, da Bit 2 des Steuerworts gesetzt war.

Die gewünschte Zeitkonstante ist 46, was einem Hexadezimalwert von 2EH entspricht. Die folgenden Anweisungen laden die Zeitkonstante zum CTC:

```
LD    A,2EH
OUT   (41H),A     ; Ausgabe der Zeitkonstanten an den
                  CTC
```

Sobald die Zeitkonstante geladen ist, beginnt der Kanal zu arbeiten. Hier ist das komplette Z80-Programm, das den CTC programmiert und daraufhin in einer Warteschleife abfragt:

```
LD    A,57H
OUT   (41H),A     ; Setze Steuerwort.
LD    A,2EH
OUT   (41H),A     ; Setze Zeitkonstante 46 dezimal.
```

```
LOOP1:  IN    A,(41H)    ; Lese Zählerstand.
        CP    2EH        ; Erster Impuls gekommen?
        JP    Z,LOOP1    ; Wenn nein, warte.
LOOP2:  IN    A,(41H)    ; Lese Zählerstand.
        CP    2EH        ; 46 Impulse gekommen?
        JP    NZ,LOOP2   ; Wenn nein, warte.
        ....  ;          Hier beginnt die Aktion des Z80,
                         ; nachdem 46 Fehlerimpulse
                         ; empfangen wurden.
```

Wie in dem Listing zu sehen ist, wird der CTC nicht auf Zählerstand 0 abgefragt. Das liegt daran, daß der Zähler, sobald er auf 0 gezählt hat, die Zeitkonstante neu lädt. Um den Endzustand vom Anfangszustand zu unterscheiden, wird erst einmal gewartet, bis der erste Zählimpuls eingetroffen ist (LOOP1).

Wir wollen jetzt das gleiche Programm unter Verwendung von Interrupts vorstellen. Nachdem der Z80 den CTC aufgesetzt hat, wartet er nur noch auf den Interrupt. Das sollte in Ihrer Anwendung besser nicht der Fall sein. Es wird hier nur zur Veranschaulichung so dargestellt.

Die Problembeschreibung ist die gleiche wie beim letzten Beispiel. Das Steuerwort für den CTC muß die Benutzung von Interrupts berücksichtigen. Es folgt das Z80-Programm:

```
        DI                 ; Sperre Z80-Interrupts.
                           ; Das verhindert ungewollte Inter-
                             rupts vom CTC.
        IM    2            ; Setze Interrupt-Modus 2.
        LD    A,80H
        LD    I,A          ; Setze oberes Byte der Interrupt-
                           ; Tabelle. Das System geht davon aus,
                           ; daß die Tabelle auf Adresse 8000 –
                           ; 8FFF steht. Die CTC-Vektoren ste-
                             hen auf Adresse 8030-8037.
        LD    A,0D7H       ; Steuerwort für Kanal 1.
        OUT   (41H),A      ; Sende es zum CTC.
        LD    A,2EH        ; Zeitkonstante = 46 dez.
        OUT   (41H),A      ; Lade Zeitkonstante in den CTC.
        LD    A,30H        ; Interrupt-Vektor.
        OUT   (41H),A      ; Schreibe ihn zum CTC.
        EI                 ; Interrupt-Freigabe.
LOOP:   JP    LOOP         ; Warte hier auf den Interrupt.
```

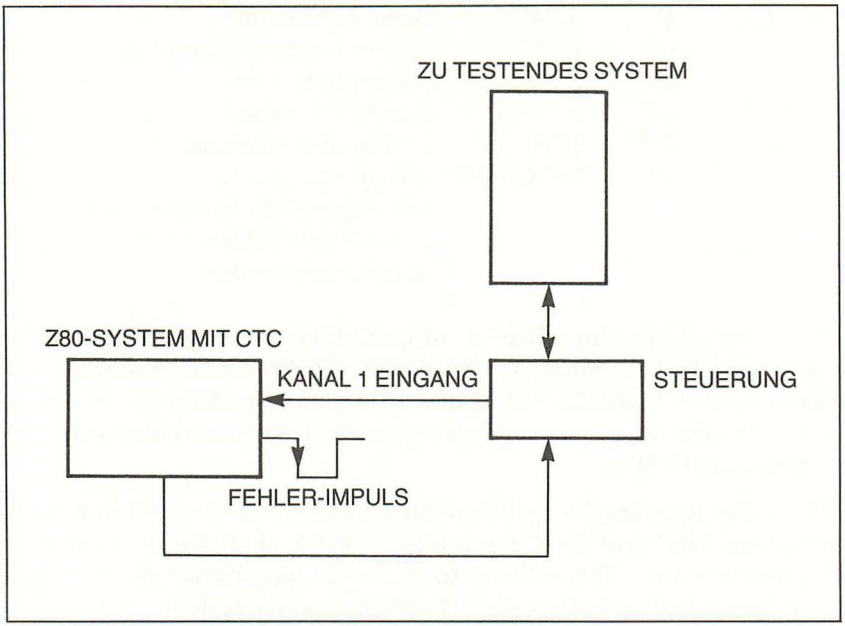


Abb. 9.9: *Blockschaltbild für die Benutzung des CTC in einer typischen Anwendung. Eine externe Schaltung sendet immer dann einen Impuls zum CTC, wenn das zu testende Gerät Fehler meldet.*

Die Adresse der Interrupt-Service-Routine muß auf Adresse 8032 im Speicher stehen. Wenn der Z80 die Service-Routine beendet, sollte die RETI-Anweisung benutzt werden. Dadurch nimmt der CTC automatisch die Interrupt-Anforderung zurück.

Ein Beispiel für den Timer-Modus des CTC

In diesem Abschnitt wollen wir sehen, wie der CTC im Timer-Modus programmiert wird. Abb. 9.10 zeigt das Blockschaltbild eines im Timer-Modus arbeitenden Kanals. Wir wollen den CTC so programmieren, daß seine Ausgangsfrequenz 2400 Hz beträgt. Solch eine Applikation kann z.B als Baudraten-Generator Anwendung finden. Wir gehen davon aus, daß die Systemtaktfrequenz 1 MHz beträgt.

Wir müssen also den Teilerfaktor berechnen, um die Eingangsfrequenz auf 2400 Hz zu bringen. Das Problem wird in Abb. 9.11 veranschaulicht. Die Periodendauer bei 2400 Hz ist 416,7 Mikrosekunden. Zur Vereinfachung runden wir auf 417 Mikrosekunden auf. Die Periodendauer des Eingangstaktes ist eine Mikrosekunde.

Der Vorteiler des CTC kann durch 16 oder durch 256 teilen. Wenn wir 417 durch 16 teilen, erhalten wir 26,06 oder abgerundet 26. Das bedeutet, daß die Zeitkonstante 26 sein muß. Der Gesamtteiler ist also $16 * 26 = 416$.

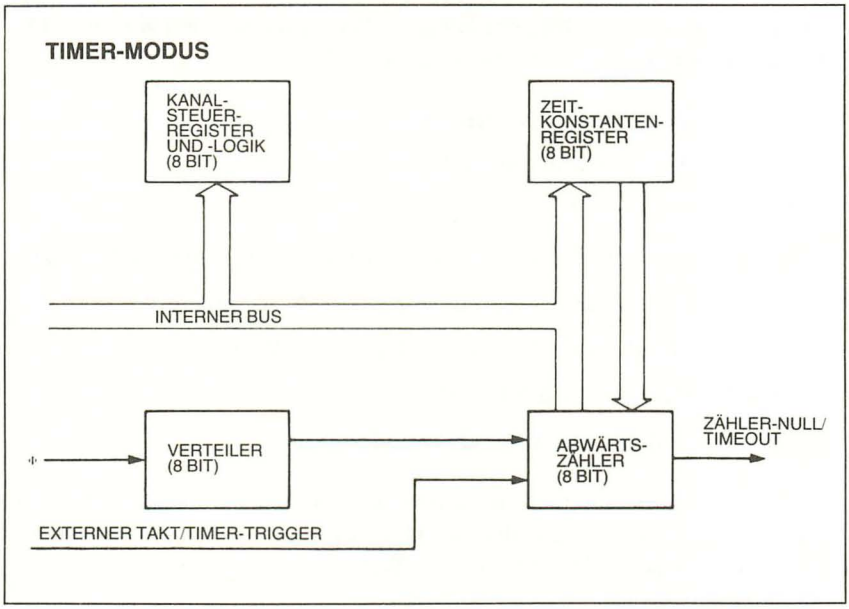


Abb. 9.10: *Blockdiagramm des Z80-CTC im Timer-Modus.*

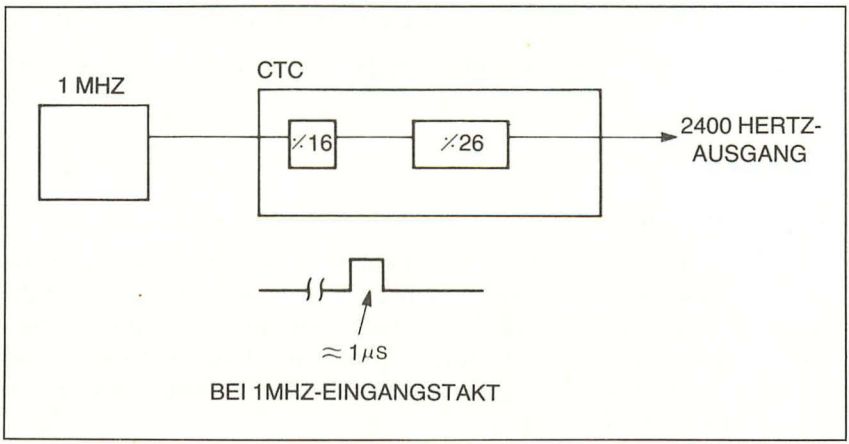


Abb. 9.11: *Verwendung des CTC als Baudraten-Generator. Die Ausgangsfrequenz ist etwa 2400 Hz bei einer Eingangsfrequenz von 1 MHz.*

Alle 416 Mikrosekunden geht der ZC/TO-Ausgang kurz auf logisch 1 und dann zurück auf logisch 0. Mit dieser Schaltung können wir eine Frequenz, die nahe an 2400 Hz liegt, erzeugen.

Wir wollen jetzt die Programmierung des CTC besprechen. Als erstes muß das Steuerwort aufgesetzt werden. Wir benutzen den Kanal 2. Die Bits des Steuerworts werden wie folgt gesetzt:

D7 = 0 keine Interrupts benutzen

D6 = 0 Timer-Modus anwählen

D5 = 0 Vorteiler 16

D4 = 0 keine Bedeutung, da wir den Triggereingang nicht benutzen

D3 = 0 Timer startet, sobald die Zeitkonstante geladen ist.

D2 = 1 Die Zeitkonstante folgt als nächstes Wort.

D1 = 1 Rücksetzen des Kanals

D0 = 0 definiert das Wort als Steuerwort

Das Datenwort für das Zeitkonstanten-Register ist 26 dezimal oder 1AH. Wenn wir davon ausgehen, daß die Hardware nach Abb. 9.4 verschaltet ist, können wir den CTC in folgender Weise programmieren:

```
LD   A,07H
OUT  (42H),A     ; Ausgabe des Steuerworts zum Kanal 2.
LD   A,1AH
OUT  (42H),A     ; Setze Zeitkonstante und starte Timer.
```

Der Timer arbeitet jetzt mit einer Ausgangsfrequenz des Kanals 2 von etwa 2400 Hertz.

Zusammenfassung

In diesem Kapitel haben wir die Verwendung des Z80-CTC-Bausteins kennengelernt. Wir begannen mit einer Untersuchung des Blockschaltbildes und betrachteten dann die für die Kommunikation mit dem Z80-Mikroprozessor wichtigen Details. Nachdem das Bauteil mit dem Z80 verbunden war, untersuchten wir die Details der Programmierung und Anwendung des CTC.

In diesem Kapitel erforschten wir alle wichtigen internen Register. Dazu zeigten wir typische Z80-Software zur Programmierung des CTC. Zum

Schluß stellten wir drei Beispiele für die Benutzung des CTC vor. Die Auswahl dieser Beispiele geschah in Hinblick auf die allgemeine Anwendbarkeit der vermittelten Informationen.

Der CTC ist ein äußerst vielseitiges Chip und kann entscheidend zur Verringerung der Bauteile-Anzahl in vielen Z80-Systemen beitragen. Wenn Sie mit diesem Baustein arbeiten, werden Sie mehr und mehr nützliche Anwendungsmöglichkeiten entdecken. Dieses Kapitel ist nur als Einstiegspunkt gedacht. Die hierin vermittelten Informationen sollen Ihnen dabei helfen, den CTC in beliebigen Systemanwendungen einzusetzen und zu programmieren.

Kapitel 10

Einführung in die serielle Kommunikation

Einführung

In diesem Kapitel wollen wir das grundlegende Konzept der seriellen Kommunikation vorstellen. Die Hardware-Implementation serieller Kommunikation werden wir dabei anhand eines häufig benutzten LSI-Bausteins untersuchen: den 8251.

Wenn Sie mit dem Konzept der seriellen Kommunikation noch nicht vertraut sind, sollten Sie dieses Kapitel sehr aufmerksam lesen. Es ist wichtig, daß Sie das vorgestellte Material vollständig verstehen. Am Ende des Kapitels sollten Sie vollständig vertraut sein mit dem Einsatz serieller Kommunikation in Mikroprozessor-Systemen.

Was ist serielle Kommunikation?

Serielle Kommunikation ist die Übertragung von Daten in einem Bitstrom, jeweils ein Bit zur Zeit. Die ganze Übertragung erfolgt in einer zeitsequentiellen Weise.

Parallele Kommunikation ist das Gegenteil von serieller Kommunikation. Alle Bits der Daten werden zur gleichen Zeit empfangen oder gesendet. Ein gutes Beispiel für die parallele Kommunikation ist die I/O-Lese- oder Schreiboperation, wobei alle acht Datenbits zur gleichen Zeit gesendet (geschrieben) oder empfangen (gelesen) werden. Tatsächlich wurden alle bisher beschriebenen Mikroprozessor-Kommunikationen in paralleler Weise ausgeführt.

Um jetzt zu veranschaulichen, was serielle Kommunikation ist und wie sie sich von paralleler Kommunikation unterscheidet, lassen Sie uns ein Beispiel untersuchen. Wir wollen acht Datenbits von einem Stück Hardware in einem Mikroprozessor-System zu einem anderen schicken. Wir planen die Übertragung auf zwei Arten: parallel und seriell.

Abb. 10.1a zeigt die Übertragung in serieller Form; Abb. 10.1b zeigt sie in paralleler Form. Beachten Sie, daß die parallele Übertragung 8 separate Datenleitungen erfordert, für jedes Bit eine. Bei der seriellen Über-

tragung wird lediglich eine physikalische Leitung benötigt – die acht Datenbits werden Bit für Bit über einen einzelnen Draht gesendet.

Serielles Timing

Wir wollen jetzt das einfache Beispiel von oben etwas erweitern, um weitere Konzepte der seriellen Kommunikation vorzustellen. Einer der kritischen Punkte bei der seriellen Kommunikation ist die Frequenz des gesendeten Datenstroms. Diese Frequenz wird als Baudrate bezeichnet. Die Baudrate ist definiert als die Anzahl der über eine einfache serielle Leitung übertragenen Bits pro Sekunde.

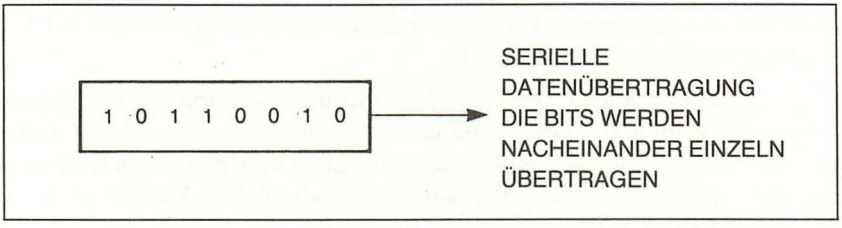


Abb. 10.1a: *Ein serieller Bitstrom wird übertragen. Das komplette Datenwort wird Bit für Bit gesendet.*

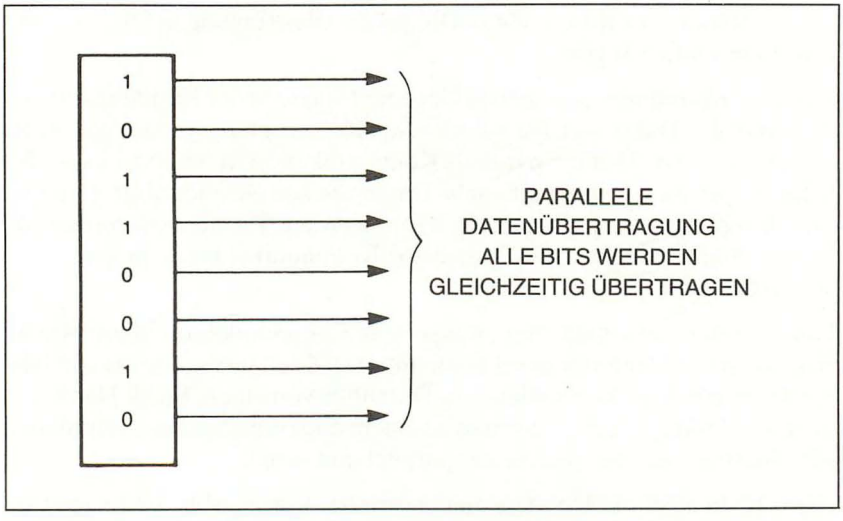


Abb. 10.1b: *Ein paralles Datenbyte wird übertragen. Die Bits werden alle gleichzeitig gesendet.*

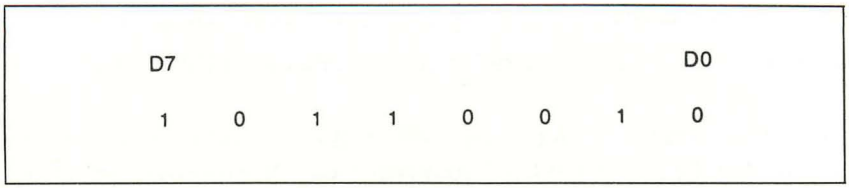


Abb. 10.2a: *Die acht auf serielle Art zu sendenden Datenbits.*

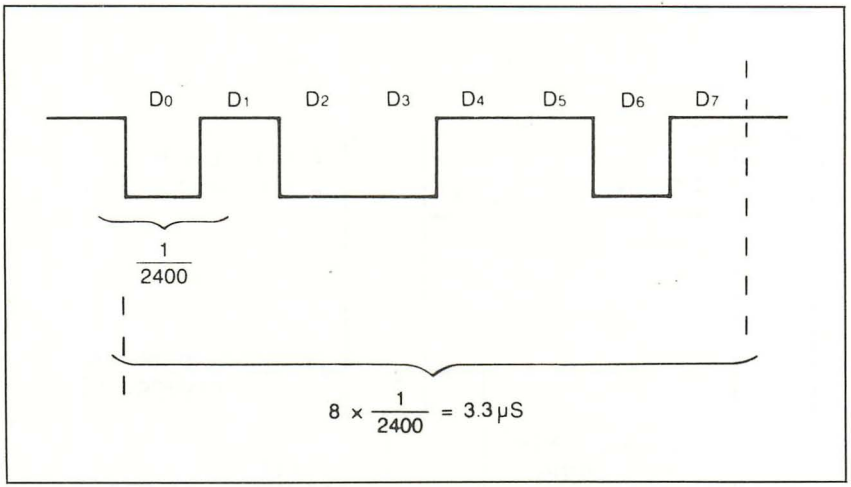


Abb. 10.2b: *Kurvenform des generierten seriellen Bitstroms anfangend mit D0.*

Typische Baudraten sind 110, 150, 300, 1200, 2400, 4800 und 9600. Wir wollen hier acht Datenbits mit 2400 Baud übertragen. Abb. 10.2a zeigt das zu sendende Datenbyte. Abb. 10.2b zeigt den Pegelverlauf des gesendeten Signals.

In Abb. 10.2 ist auch zu sehen, daß die Dauer eines gesendeten Bits gleich 1/Baudrate ist. In diesem Beispiel ist sie somit 1/2400 = 416 Mikrosekunden. Damit können wir auch die Übertragungsdauer für alle acht Bit errechnen. Sie ist 8 * 416 μS = 3328 μS. Beim parallelen Transfer würden die gleichen acht Bit normalerweise in weniger als 1 μS übertragen.

Umsetzung paralleler Daten in serielle Daten

Der größte Teil der seriellen Kommunikation ist die Umsetzung der parallelen Daten in einen seriellen Bitstrom. Die Umsetzungsschritte sind folgende:

1. Speichere das parallele 8-Bit-Wort in einem Schieberegister.
2. Schiebe die 8 Bits mit der richtigen Baudrate Bit für Bit aus dem Schieberegister.

Diese beiden Schritte sind in Abb.10.3 als Blockschaltbild zu sehen. Wir sehen, daß die zu sendenden Daten zuerst vom Mikroprozessor erzeugt werden und dann in einem parallel ladbaren 8-Bit-Schieberegister gespeichert werden. Die Daten werden mit D0 beginnend und D7 als letztes ausgeschoben.

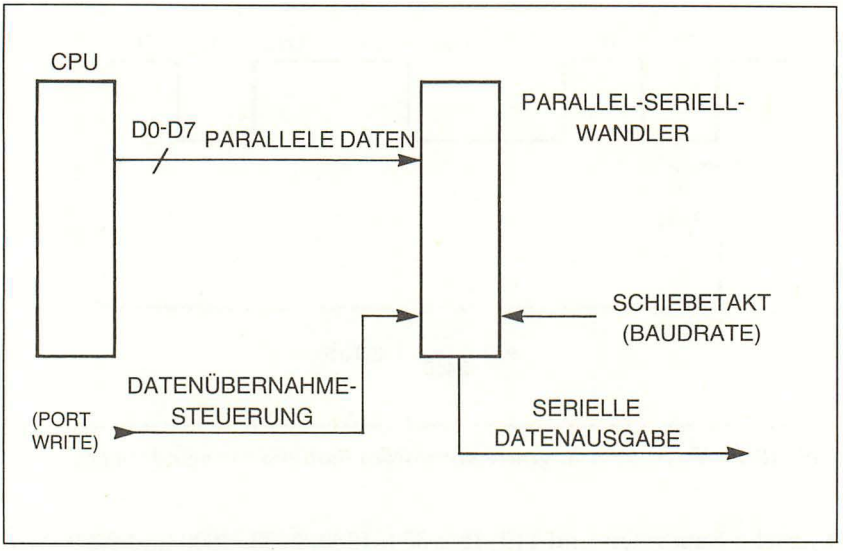


Abb. 10.3: *Umsetzung eines parallelen Bytes in einen seriellen Bitstrom. Der Mikroprozessor lädt die Daten in ein Schieberegister, wo sie dann Bit für Bit ausgeschoben werden.*

Start-Bit

Bislang haben wir den Begriff der Baudrate und die Parallel-zu-Seriell-Umsetzung kennengelernt. Die gesendeten Daten müssen jedoch auch auf der Empfängerseite richig interpretierbar sein. Dazu wird ein weiteres Bit, das Startbit, automatisch dem Bitstrom zugefügt.

Die Funktion des Startbits ist es, dem Empfänger mitzuteilen, wann ein neuer Datenstrom beginnt und ihm somit zu ermöglichen, seinen Takt mit dem Bitstrom zu synchronisieren. Jeder Datenstrom repräsentiert ein einzelnes Datenzeichen. Sie können sich jeden Datenstrom als ein Byte

paralleler Daten vorstellen. Natürlich müssen die Daten nicht unbedingt acht bit lang sein, doch macht diese Vorstellung das Verständnis der seriellen Kommunikation leichter.

Solange die Datenausgangsleitung keine Daten sendet, ist sie in einem Zustand, der als Markierung (marking) bezeichnet wird. Dies ist der Ruhezustand der seriellen Übertragungsleitung. Lassen Sie uns annehmen, daß der Markierungszustand einer Übertragungsleitung logisch 1 ist. Das Startbit am Anfang jedes Datenstroms hat den entgegengesetzten Pegel, also hier logisch 0.

Das Startbit ist tatsächlich ein am Anfang des Datenbitstroms angefügtes einzelnes Bit. Wie auch die Baudrate des Bitstroms sein mag, das Startbit hat immer eine Länge von einem bit (siehe Abb. 10.4). Die Empfänger-Hardware detektiert dieses Startbit und ermöglicht das Einlesen der Daten.

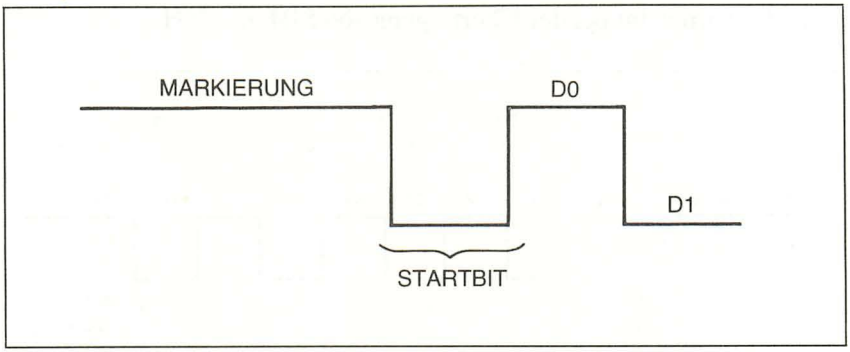


Abb. 10.4a: *Timing des Startbits für den Fall, daß das erste Datenbit logisch 1 ist.*

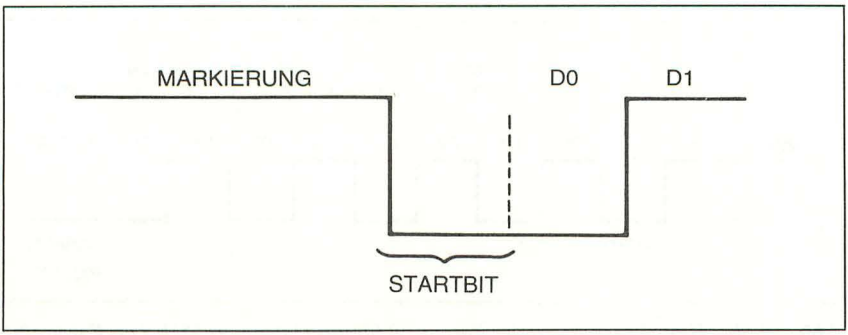


Abb. 10.4b: *Timing des Startbits für den Fall, daß das erste Datenbit logisch 0 ist.*

Paritätsbit

Ein anderes Bit, das dem seriellen Bitstrom zugefügt werden kann, ist das Paritätsbit. Dieses Bit wird vom Sender eingefügt und vom Empfänger benutzt. Hier die Beschreibung seiner Funktion:

Wenn ein Wort zur Übertragung bereitgestellt wird, enthält es eine Anzahl logischer Einsen. Diese Anzahl kann gerade oder ungerade sein. Zum Beispiel hat das 8-Bit-Wort 54H drei Einsen, das Wort 55H aber vier. Die Empfänger-Hardware zählt die empfangenen Einsen und erwartet entweder eine gerade oder eine ungerade Anzahl.

Nehmen wir an, die Empfänger-Hardware untersucht jeden Bitstrom auf eine gerade Anzahl logischer Einsen. Die Zahl 55H wäre in Ordnung, die Zahl 54H aber nicht. Deshalb würde der Sender eine zusätzliche Eins dem Datenstrom zufügen. Es wird also ein zusätzliches Bit übertragen. Dieses Bit kann entweder eine logische 1 oder eine logische 0 sein, abhängig davon, wie eine gerade Anzahl der Einsen zu erreichen ist. Abb. 10.5 zeigt das Paritätsbit bei der Übertragung von 54H und 55H.

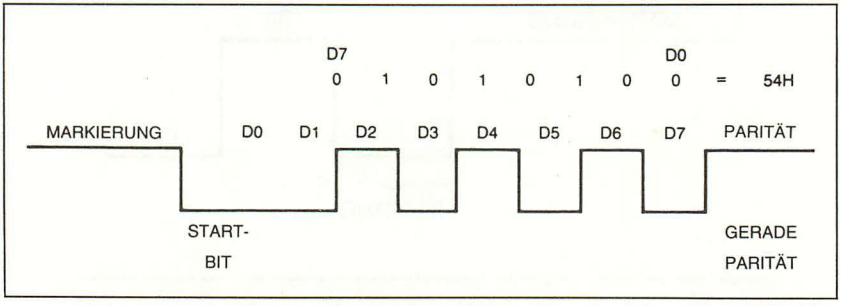


Abb. 10.5a: *Timing für ein gerades Paritätsbit für das Datenwort 54H. Das Paritätsbit ist in diesem Fall logisch 1.*

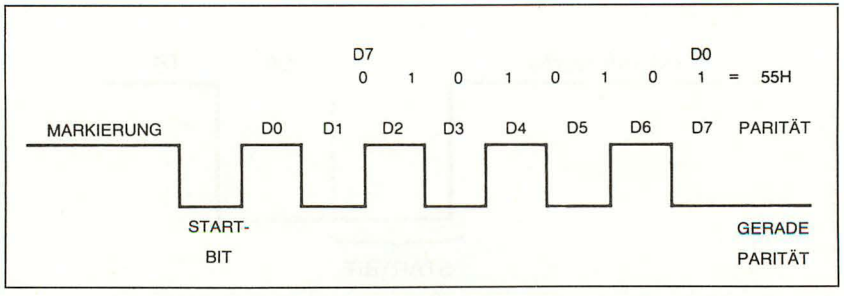


Abb. 10.5b: *Timing für ein gerades Paritätsbit für das Datenwort 55H. Das Paritätsbit ist in diesem Fall logisch 0.*

Stopbit

Das letzte Bit, das dem Bitstrom vom Sender zugefügt wird, ist das Stopbit. Der Empfänger erwartet am Ende jedes Bitstroms ein Stopbit. Standard sind 1, 1½ oder 2 Stopbits. Abb. 10.6 zeigt ein komplettes 8-Bit-Datenwort mit Startbit, Paritätsbit und Stopbit. Der komplette Datenstrom besteht also aus 12 Bit anstatt der 8 Bit, mit denen wir begonnen haben. Bei 2400 Baud ist die Gesamtübertragungszeit der in Abb. 10.6 gezeigten Daten gleich 12 * 416 μS = 4,99 mS.

Rückblick auf das Konzept der seriellen Kommunikation

In einem Rückblick wollen wir hier die wichtigsten Punkte der seriellen Kommunikation auflisten:

1. Serielle Daten werden Bit für Bit übertragen, das niederwertigste Bit zuerst.
2. Daten werden in einer festen Rate, der Baudrate übertragen. Die Baudrate ist gleich der Anzahl der übertragenen Bits pro Sekunde. Bei 1200 Baud würde z.B. die Taktfrequenz 1200 Hz sein.

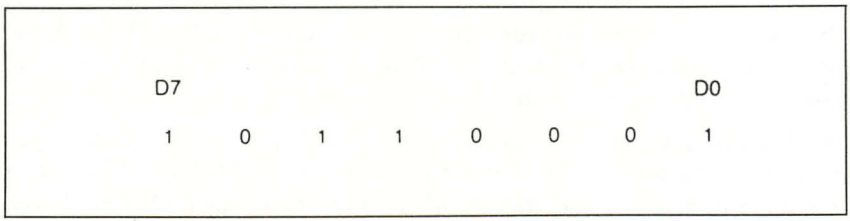


Abb. 10.6a: *Die seriell zu übertragenen Daten.*

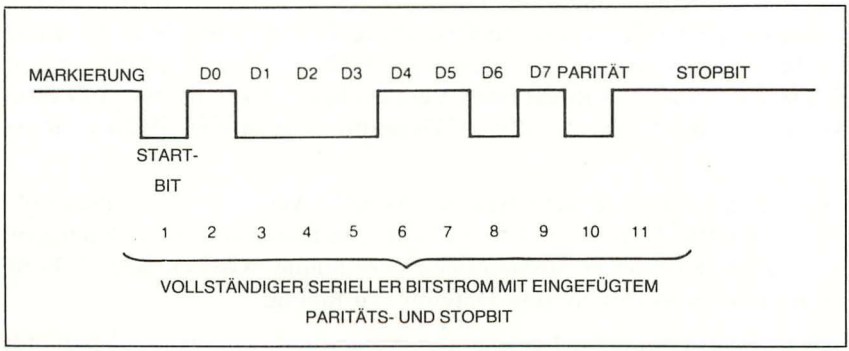


Abb. 10.6b: *Kurvenform des kompletten Bitstroms mit Start-, Paritäts- und Stopbit.*

3. Die parallelen Daten müssen zur seriellen Kommunikation erst in serielle Daten umgewandelt werden.
4. Markierung ist die Bezeichnung für den logischen Zustand der Verbindungsleitung, solange keine Daten gesendet werden.
5. Der Sender fügt ein Bit am Anfang des seriellen Datenstroms zu: das Startbit. Dieses Bit hat den entgegengesetzten logischen Pegel der Markierung.
6. Der Sender hat die Möglichkeit, ein einfaches Bit, das Paritätsbit, am Ende des Datenstroms einzufügen. Dieses Paritätsbit kann wahlweise gerade oder ungerade Parität generieren. Die Parität wird vom Empfänger zur Fehlererkennung benutzt.
7. Der Sender fügt zum Schluß noch Stopbits an. Sie können 1, 1½ oder 2 bits lang sein. Stopbits haben denselben logischen Pegel wie die Markierung auf der Übertragungsleitung.

Übersicht über den 8251

Serielle Kommunikation wird in Mikroprozessor-Systemen sehr häufig eingesetzt. Eines der Haupteinsatzgebiete ist das Terminal-Interface. Da die Übertragungsart sehr verbreitet ist, wurden spezielle LSI-Bausteine dafür entwickelt. Der 8251 ist so ein Baustein. Abb.10.7 zeigt den 8251 in einem Mikroprozessor-System.

Wir können aus Abb. 10.7 ersehen, daß der 8251 genauso wie andere spezielle I/O-Bausteine behandelt wird (einschließlich PIO und CTC). Der Z80 kommuniziert mit dem 8251 über I/O-Befehle. Der 8251 muß vom Programm für seine Arbeitsweise erst einmal aufgesetzt werden. In diesem Kapitel werden Sie lernen, wie der 8251 mit dem Z80 zu verbinden ist und wie die Kommunikation abläuft. Zuerst wollen wir jedoch die wichtigsten Software-Konzepte zur Benutzung des 8251 aufzeigen. Ab Ende sollten Sie wissen, wie ein typisches serielles I/O-Bauteil funktioniert. Diese Kenntnis wird Ihnen beim Verständnis anderer serieller I/O-Bausteine, wie der Z80-SIO, helfen. (Wir besprechen die Z80-SIO im Kapitel 11.)

Abb. 10.8 zeigt das Blockschaltbild des 8251. Wir wollen hier die Funktion jedes Blocks untersuchen und mit dem Datenbus-Puffer beginnen. Dieser Block ist für die physikalische Verbindung zwischen dem 8251 und dem Mikroprozessor-System-Datenbus zuständig.

Als nächstes kommt die Lese/Schreib-Steuerlogik. Dieser Block sorgt für die Steuerung der internen Datentransfers im richtigen Timing.

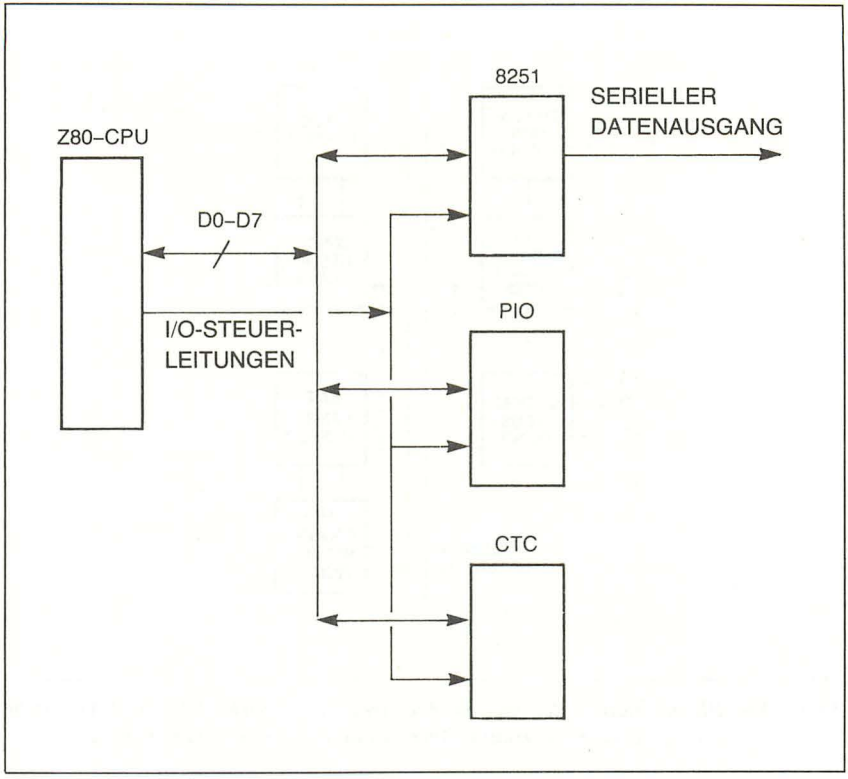


Abb. 10.7: *Der 8251 in einem Z80-gesteuerten Mikroprozessor-System.*

Der als Modemsteuerung bezeichnete Block wird zur Vereinfachung der Verbindung mit einem Modem benutzt. Für die Leser, die nicht wissen, was ein Modem ist, sei hier nur angemerkt, daß es zur seriellen Verbindung über eine Telefonleitung verwendet wird. Abb. 10.9 zeigt den Einsatz von Modems innerhalb eines seriellen Kommunikationssystems.

Weiter können wir in Abb. 10.8 zwei Blöcke sehen, die als Sendepuffer (P–S) und Sendesteuerung bezeichnet sind (P–S steht für Parallel-zu-seriell-Umsetzung). Beide Blöcke sind zusammen zuständig für die serielle Datenausgabe über den TxD-Ausgang. Tx steht für Transmit (senden), D für Data. Die Sendesteuerung prüft elektrisch den Status des Sendepuffers, um zu sehen, ob er leer ist und neue Daten gesendet werden können.

Die letzten zwei Blöcke in Abb. 10.8 sind der Empfangspuffer und die Empfangssteuerung. Der Empfangspuffer empfängt einen Bitstrom von

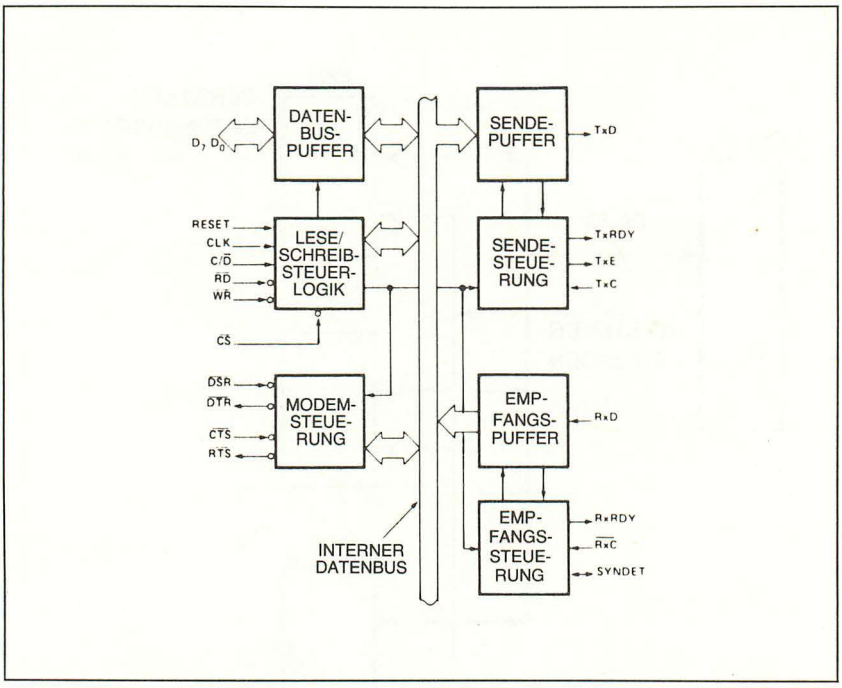


Abb. 10.8: *Blockschaltbild der internen Architektur des USART 8251. (USART steht für Universal Synchronous Asynchronous Receiver Transmitter.)*

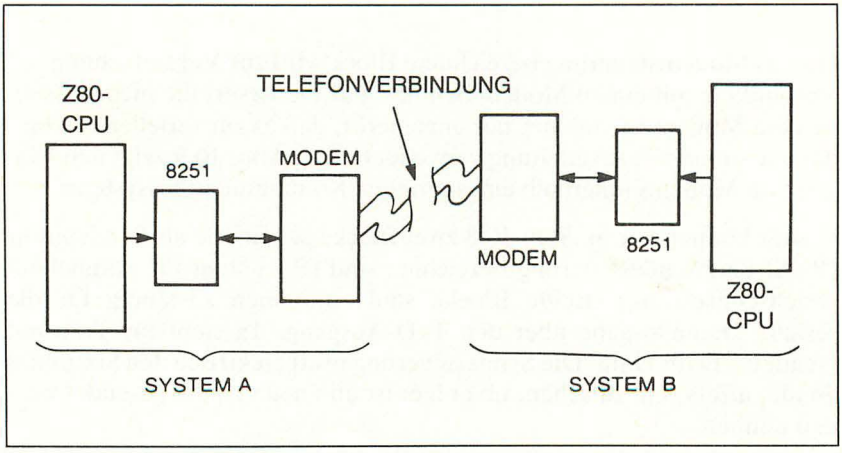


Abb. 10.9: *Dieses Blockschaltbild zeigt die Verwendung von Modems zur seriellen Kommunikation über eine Telefonleitung.*

einem anderen seriellen Baustein über den RxD-Eingang (RxD steht für Receive Data) und wandelt ihn in ein paralleles Wort um. Die Empfangssteuerung überprüft elektrisch den Status des Empfangspuffers und ermittelt so, ob der Puffer voll ist und die Daten vom Mikroprozessor abgeholt werden können.

Pinbelegung des 8251

Abb. 10.10 zeigt die Pinbelegung des 8251. Wir wollen jetzt jeden einzelnen Pin besprechen. Das wird uns beim Verständnis der physikalischen Verbindung mit dem Z80-Systembus hilfreich sein.

D0-D7 Dies sind die acht Datenleitungen zur Verbindung mit dem Mikroprozessor-Datenbus. Alle Informationen und Programmierungsdaten werden über diese Leitungen übertragen.

$\overline{\text{RESET}}$ Dieser Eingang ist logisch 1 aktiv. Der 8251 wird in den Ruhezustand gesetzt, wenn der $\overline{\text{RESET}}$ angelegt wird. Der 8251 bleibt in diesem Zustand, bis er die entsprechenden Programmierungsworte empfangen hat.

CLK *(Clock)* Dieser Takteingang wird zur Synchronisation interner Datentransfers benutzt. Er ist nicht mit dem Baudraten-Takteingang identisch. Die Taktfrequenz muß mindestens 30 mal (bei asynchronem Betrieb 4,5 mal) so hoch sein wie die des Baudraten-Eingangs des Senders oder des Empfängers.

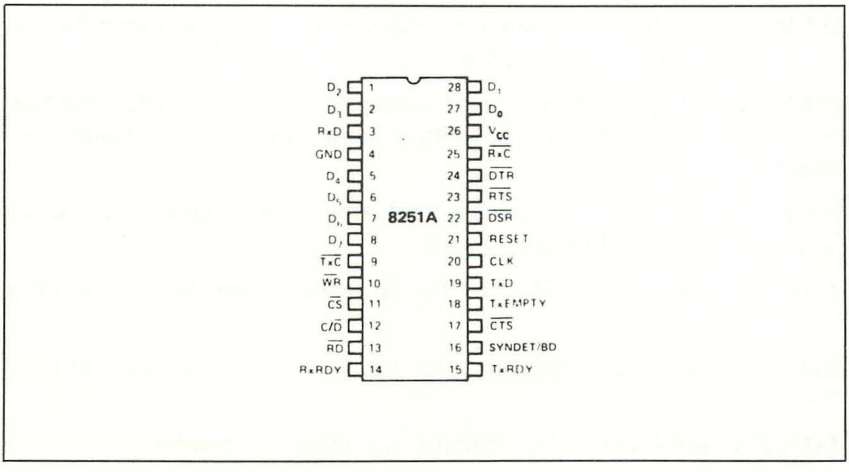


Abb. 10.10: *Pinbelegung des 8251.*

$\overline{\text{WR}}$ *(Schreibeingang,* aktiv logisch 0) Wenn dieser Eingang logisch 0 ist, werden Daten vom System-Datenbus in die internen Register des 8251 geschrieben.

$\overline{\text{RD}}$ *(Leseeingang,* aktiv logisch 0) Wenn diese Eingangsleitung logisch 0 ist, werden Daten aus den internen Registern des 8251 auf den System-Datenbus gelegt.

C/$\overline{\text{D}}$ *(Control/Data-Eingang)* Diese Leitung wird zur Auswahl des internen Registers während eines I/O-Befehls verwendet. Wenn sie logisch 1 ist, so wird das Steuerregister angesprochen. Ist sie logisch 0, so wird das Datenregister angesprochen.

$\overline{\text{CS}}$ *(Chip Select,* Bausteinauswahl) Eine logische 0 an diesem Eingang ermöglicht die Kommunikation mit dem Chip über den Datenbus.

MODEM-STEUERUNG Die folgenden vier Pins werden zur Vereinfachung des Modem-Anschlusses benutzt:

$\overline{\text{DSR}}$ Data Set Ready
$\overline{\text{DTR}}$ Data Terminal Ready
$\overline{\text{CTS}}$ Clear to Send
$\overline{\text{RTS}}$ Ready to Send

Wir wollen sie im Detail untersuchen.

$\overline{\text{DSR}}$ Dieser Eingang kann von der CPU abgefragt werden. Er wird normalerweise zum Test eines Modem-Zustandes benutzt.

$\overline{\text{DTR}}$ Dieser Ausgang kann durch das Senden eines bestimmten Bitmusters auf logisch 0 gesetzt werden.

$\overline{\text{CTS}}$ Eine logische 0 an diesem Eingang gibt den Sendepuffer zum Senden der Daten frei. Die $\overline{\text{CTS}}$-Leitung kann zum Hardware-Handshake benutzt werden.

$\overline{\text{RTS}}$ Dieser Ausgang kann gesetzt werden, wenn das entsprechende Steuerwort zum 8251 geschickt wird.

TxD *(Transmit Data)* Dies ist die Ausgangsleitung für die seriellen Daten.

RxD *(Receive Data)* Dies ist die Eingangsleitung für die seriellen Daten.

TxC *(Transmit Clock)* Baudraten-Eingang für den Sender.

RxC *(Receive Clock)* Baudraten-Eingang für den Empfänger.

TxRDY Dieser Ausgang kann benutzt werden, um der CPU mitzuteilen, daß der 8251 bereit ist, ein weiteres Zeichen zu senden. Sie ist auch zur Interrupt-Anforderung geeignet.

TxEMPTY Solange das Bauteil kein Zeichen zum Senden hat, ist dieser Ausgang logisch 1. Er wird automatisch logisch 0, wenn die CPU ein Zeichen in den Sendepuffer schreibt.

RxRDY Dieser Ausgang wird logisch 1, wenn der Empfangspuffer voll ist. Wie der TxRDY-Ausgang kann auch mit dieser Leitung ein Interrupt ausgelöst werden.

Verbindung des 8251 mit dem Z80

Wir wollen jetzt sehen, wie der 8251 mit dem Z80-Mikroprozessor verbunden wird. Um die Programmierung wollen wir uns im Augenblick noch nicht kümmern. Solange der Z80 noch nicht mit dem 8251 kommunizieren kann, nützt uns die Programmierung des Bauteils wenig, da die Programmierungsinformationen elektrisch nicht ankommen würden.

Abb. 10.11 zeigt eine komplette Schaltung zur Verbindung des 8251 mit dem Z80-System-Bus. Die seriellen Eingangs- und Ausgangsverbindungen sind hier noch nicht dargestellt. Abb. 10.11 zeigt einen Datenpuffer zwischen dem 8251 und dem System-Datenbus. In Ihrem System kann dieser Puffer nötig sein oder auch nicht. Wird er nicht benötigt, können die Chip-Datenleitungen direkt mit dem Datenbus verbunden werden.

Der $\overline{CS}$-Eingang des 8251 wird aus den Adreßleitungen A1–A7 decodiert. In diesem Beispiel sind die Port-Adressen 7CH und 7DH. A0 des Adreßbusses wird mit dem C/$\overline{D}$-Eingang verbunden.

Der $\overline{RD}$- und der $\overline{WR}$-Eingang werden mit der $\overline{IOR}$ und $\overline{IOW}$-Steuerleitung verbunden. Der $\overline{RESET}$-Eingang des 8251 ist aktiv logisch 1. Das ist der entgegengesetzte Pegel der Z80-RESET-Leitung, so daß ein Inverter eingesetzt wird. Abb. 10.11 verdeutlicht, daß die Verbindung des Z80-Systembusses mit dem 8251 der Verbindung mit anderen, früher besprochen Peripheriebausteinen, entspricht.

Der Baudraten-Takt wird über die Eingänge TxC und RxC an den 8251 gelegt. Die beiden Taktfrequenzen müssen nicht gleich sein. Der Baustein ist elektrisch in der Lage, Daten mit einer anderen Baudrate zu senden als zu empfangen. Natürlich muß die Baudrate mit der des Bausteins auf der anderen Seite der seriellen Leitung übereinstimmen.

Ein weiterer Takteingang des 8251 liegt am Pin 20. Dieser Takt wird zur Synchronisation der internen Datentransfers benutzt. Die Frequenz muß

wesentlich über der der Baudraten-Takteingänge liegen. In der Regel wird hierfür einfach der Systemtakt vom Z80 benutzt.

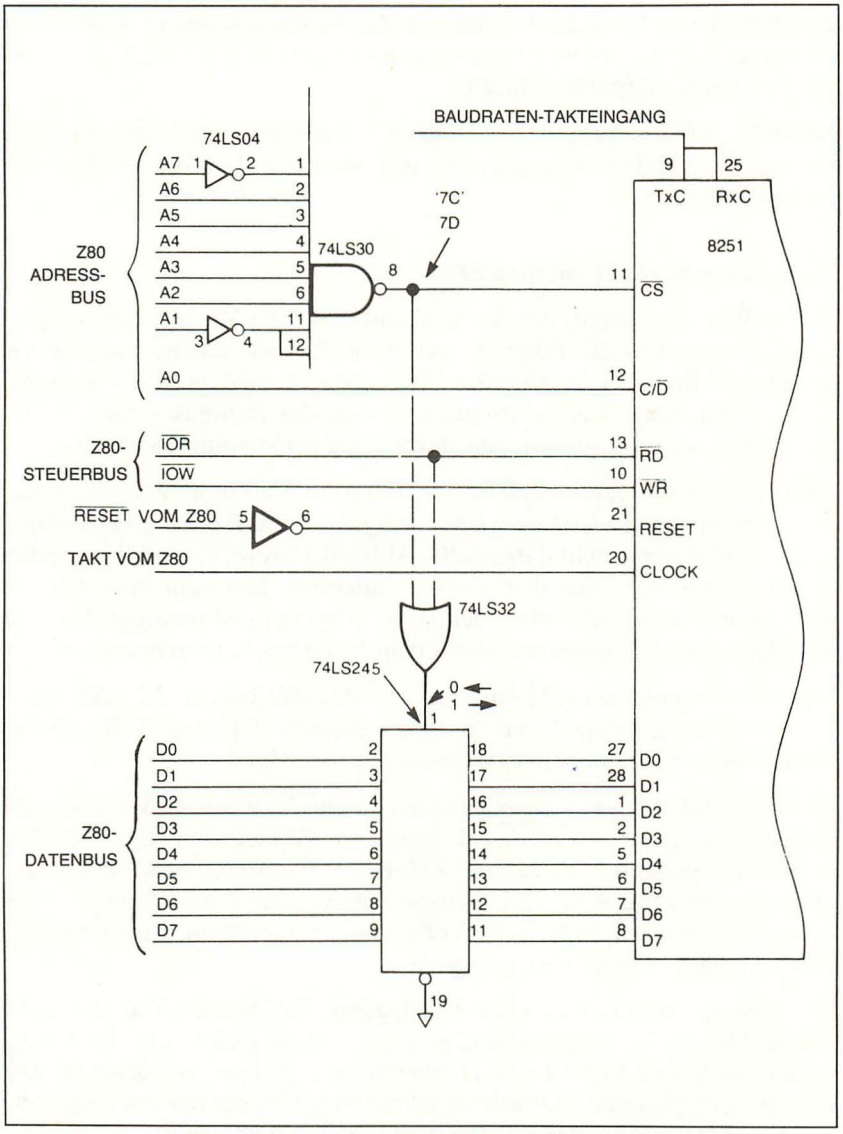


Abb. 10.11: *Verbindung des 8251 mit dem Z80-Systembus. In diesem Fall liegt ein Datenpuffer zwischen dem Z80-Datenbus und den Datenleitungen des 8251.*

Die serielle Verbindung

Lassen Sie uns nun den 8251 mit den seriellen Übertragungsleitungen verbinden. Es gibt für die serielle Übertragung mehrere Standards. In unserem Beispiel wollen wir den heute am häufigsten eingesetzten Standard benutzen: die RS-232-Norm zur elektrischen seriellen Übertragung.

Die RS-232 benutzt anstelle des normalen TTL-Pegels (+5V) eine höhere Spannung. Sie liegt bei etwa ±12V. Abb. 10.12 zeigt als Blockschaltbild ein typisches Interface zur RS-232-Leitung. Wie zu sehen ist, müssen zwischen den TTL-Leitungen des 8251 und den RS-232-Leitungen spezielle Konverter geschaltet werden.

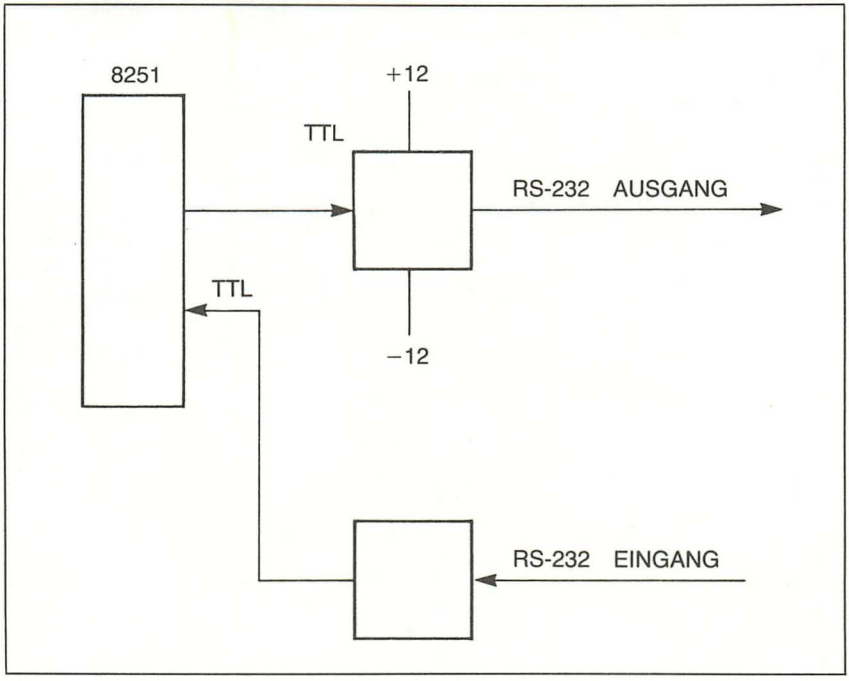


Abb. 10.12: *Zur Umsetzung des TTL-Pegels des 8251 in den RS-232-Pegel sowie umgekehrt werden Konverter eingesetzt.*

Die Umsetzung der TTL-Pegel in RS-232-Pegel und umgekehrt kommt so häufig vor, daß spezielle Bausteine dafür konstruiert wurden. Als Beispiel seien hier die integrierten Schaltungen MC1488 und MC1489 genannt. Datenblattauszüge für diese beiden Bausteine finden Sie in Abb. 10.13 und 10.14.

MOTOROLA

MC1488

QUAD LINE DRIVER

The MC1488 is a monolithic quad line driver designed to interface data terminal equipment with data communications equipment in conformance with the specifications of EIA Standard No. RS-232C.

Features:

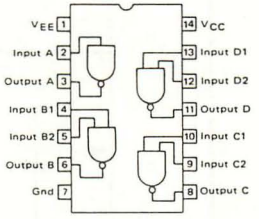
- Current Limited Output
  ±10 mA typ
- Power-Off Source Impedance
  300 Ohms min
- Simple Slew Rate Control with External Capacitor
- Flexible Operating Supply Range
- Compatible with All Motorola MDTL and MTTL Logic Families

QUAD MDTL LINE DRIVER
RS-232C
SILICON MONOLITHIC
INTEGRATED CIRCUIT

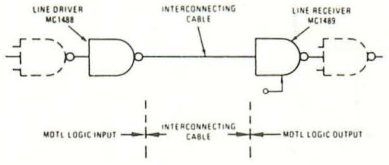
L SUFFIX
CERAMIC PACKAGE
CASE 632
TO-116

P SUFFIX
PLASTIC PACKAGE
CASE 646

PIN CONNECTIONS



TYPICAL APPLICATION



CIRCUIT SCHEMATIC
(1/4 OF CIRCUIT SHOWN)

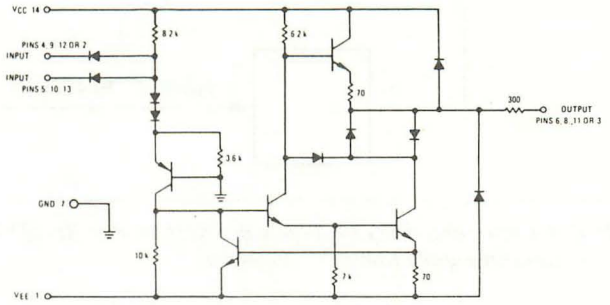


Abb. 10.13: *Datenblattauszug für den TTL-zu-RS-232-Umsetzer MC1488 von Motorola.*

MOTOROLA

MC1489L
MC1489AL

QUAD LINE RECEIVERS

The MC1489 monolithic quad line receivers are designed to interface data terminal equipment with data communications equipment in conformance with the specifications of EIA Standard No. RS-232C.

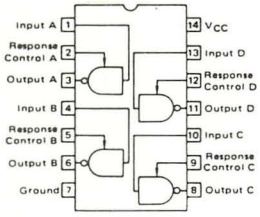
- Input Resistance – 3.0 k to 7.0 kilohms
- Input Signal Range – ±30 Volts
- Input Threshold Hysteresis Built In
- Response Control
  a) Logic Threshold Shifting
  b) Input Noise Filtering

QUAD MDTL LINE RECEIVERS RS-232C

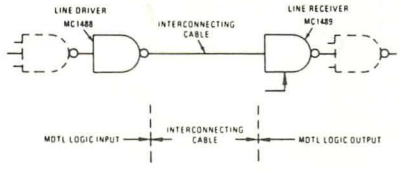
SILICON MONOLITHIC INTEGRATED CIRCUIT

L SUFFIX
CERAMIC PACKAGE
CASE 632
TO-116

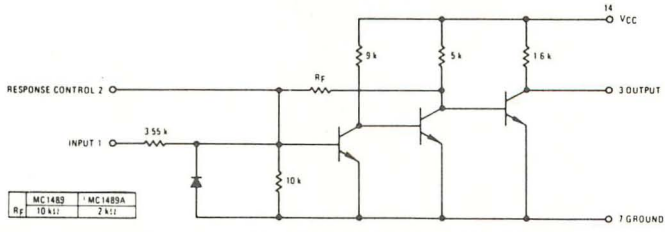
P SUFFIX
PLASTIC PACKAGE
CASE 646



TYPICAL APPLICATION



CIRCUIT SCHEMATIC (1/4 OF CIRCUIT SHOWN)



RF	MC1489	MC1489A
	10 kΩ	2 kΩ

Abb. 10.14: *Datenblattauszug für den RS-232-zu-TTL-Umsetzer MC1489 von Motorola.*

Abb. 10.15 zeigt, wie der MC1488 und der MC1489 in einem typischen System zur seriellen Kommunikation eingesetzt werden. Wie Sie sehen, werden für die vollständige Kommunikation zwischen zwei seriellen Bausteinen nur drei Leitungen benötigt: Tx (senden), Rx (empfangen) und GND (Masse).

In der obigen Besprechung der System-Hardware sind die wichtigsten Punkte der seriellen Übertragung vorgestellt worden, die Sie zur weiteren Verwendung in Erinnerung behalten sollten. In Systemen, die serielle Übertragung benutzen, müssen die vorstehenden Hardware-Einheiten in irgendeiner Form vorhanden sein. Wenn Sie wissen, was gebraucht wird, wird Ihnen die Entwicklung und Analyse solcher Schaltungen leichter fallen.

Programmierung des 8251

Jetzt, da wir den 8251 mit dem Z80 verbunden haben, wollen wir die für eine asynchrone serielle Kommunikation notwendige Software bespre-

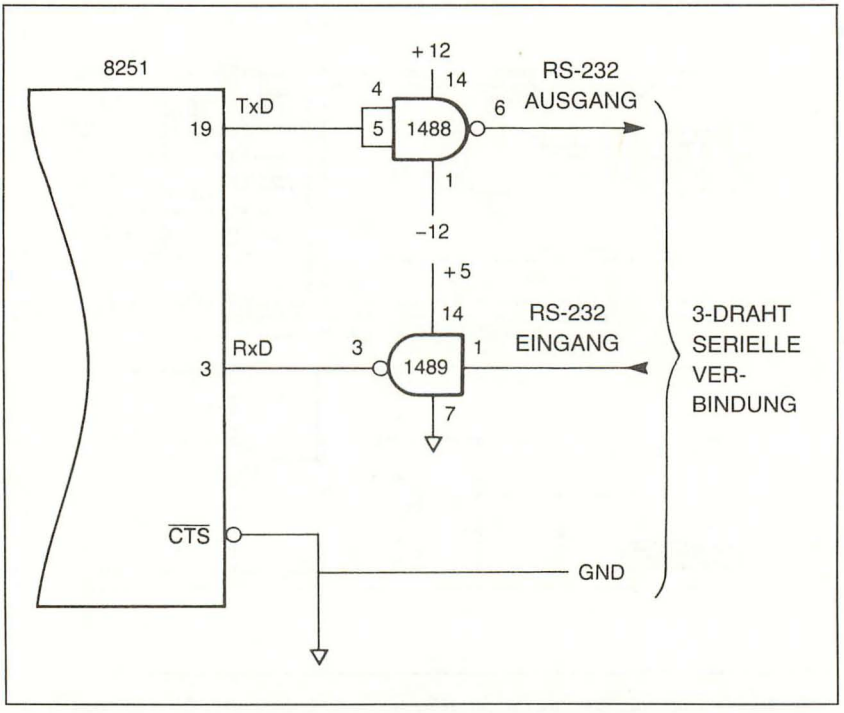


Abb. 10.15: *Verwendung des MC1488 und des MC1489 zur seriellen Übertragung.*

chen. Allgemein gesagt, versorgt die Software den 8251 mit den Übertragungsparametern. Das schließt die Baudrate ein, die Anzahl der Bits, die Anzahl der Stopbits und die Paritätsinformation.

Nachdem das Bauteil einmal programmiert ist, kann der allgemeine Datenverkehr stattfinden. Der Mikroprozessor prüft erst, ob der Sendepuffer leer ist. Wenn das der Fall ist, lädt er den Puffer mit einem Zeichen, das dann über die serielle Leitung gesendet wird. Dieser Vorgang muß bei jedem gesendeten Zeichen stattfinden.

Auch bei der Zeicheneingabe muß die CPU zuerst das Statusregister lesen, um zu sehen, ob ein Zeichen empfangen worden ist.

Die vorstehende Beschreibung des Empfangens und Sendens eines seriellen Zeichens ist eine Vereinfachung. Wir wollen im folgenden sehen, was unter Verwendung des 8251 genau praktiziert werden muß.

Bevor mit dem Datenverkehr begonnen wird, muß ein Satz von Steuerwörtern in den 8251 geladen werden. Dieser wird von der CPU in das Steuerregister geschrieben. Der 8251 interpretiert das erste nach einem Reset in das Steuerregister geschriebene Byte als Modus-Wort. Das folgende Byte wird dann als Befehlswort interpretiert. Das Befehlswort kann, abängig vom Moduswort, auch aus mehreren hintereinander folgenden Bytes bestehen.

Abb. 10.16 zeigt die Bit-Definition des Moduswortes. Wir wollen diese Abbildung benutzen, um den 8251 für eine typische Übertragung zu programmieren. Wir definieren diese Übertragung folgendermaßen:

1. Die Baudrate ist 2400. Unser Baudraten-Generator soll mit der 16-fachen Frequenz (= 38,4 kHz) arbeiten. Sie wird intern heruntergeteilt. Es ist auch möglich, einen Faktor von 1 oder 64 zu programmieren. Um die Synchronisation des Empfängers bei nicht gekoppelten Taktleitungen zu gewährleisten, sollte der Baudraten-Generator mit einer vielfachen Frequenz der Baudrate arbeiten.

2. Jedes Zeichen soll mit acht Bits übertragen werden.

3. Das Paritätsbit soll für gerade Parität gesetzt sein.

4. Der Sender soll pro Zeichen 2 Stopbits einsetzen.

   Mit der vorstehenden Definition bekommt das Moduswort den Wert:

   1 1 1 1 1 1 1 0

Nehmen wir das Wort auseinander, so sehen wir folgendes:

Bit D7,D6 = 11. Das setzt 2 Stopbits.

Bit D5,D4 = 11. Parität wird benutzt und auf gerade gestellt.

Bit D3,D2 = 11. Die Zeichenlänge ist 8 bit.

Bit D1,D0 = 10. Der Baudraten-Faktor ist 16.

Die Z80-Anweisungen zur Programmierung dieses Moduswortes sind:

```
LD    A,0FEH
OUT   (7DH),A     ; Ausgabe zum Steuerregister
```

Als nächstes muß das Befehlswort zum 8251 geschrieben werden. Die Bit-Definition hierfür finden Sie in Abb. 10.17. Wir schreiben folgenden Wert zum Steuerregister:

0 0 0 1 0 1 0 1

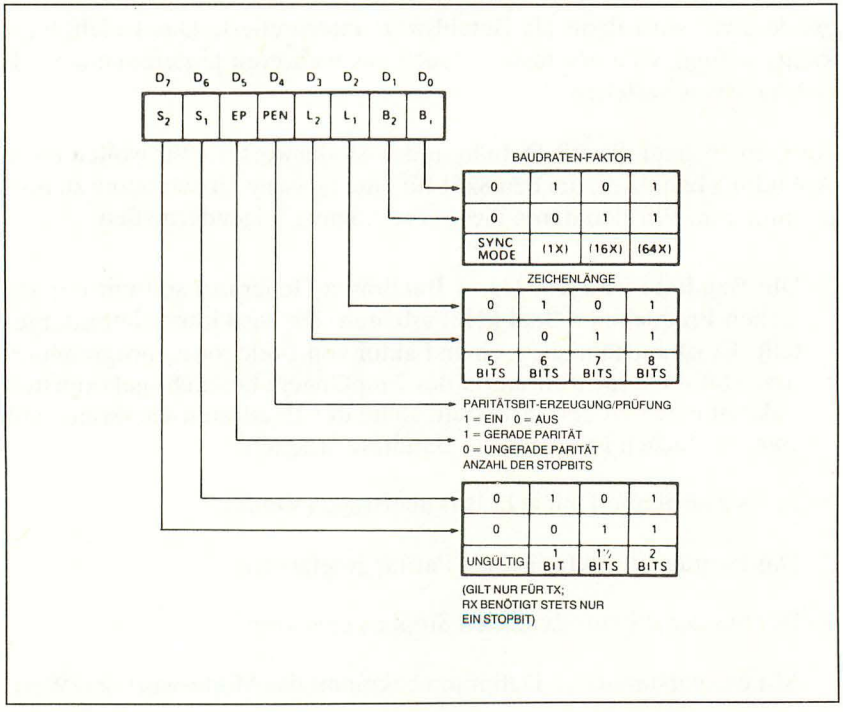


Abb. 10.16: *Bit-Definition für das Moduswort des 8251 (asynchrone Übertragung).*

Dieses Byte schaltet Empfänger und Sender ein und löscht die Fehlerregister. Ab jetzt ist der 8251 zum Senden und Empfangen der Zeichen bereit.

Als letztes soll jetzt noch das Statusregister besprochen werden. Dieses Register kann von der CPU nur gelesen werden. Der Mikroprozessor benutzt es, um den Status des Sende- und des Empfangsregisters abzufragen und um Fehler im empfangenen Zeichen zu erkennen. Die Bit-Defi-

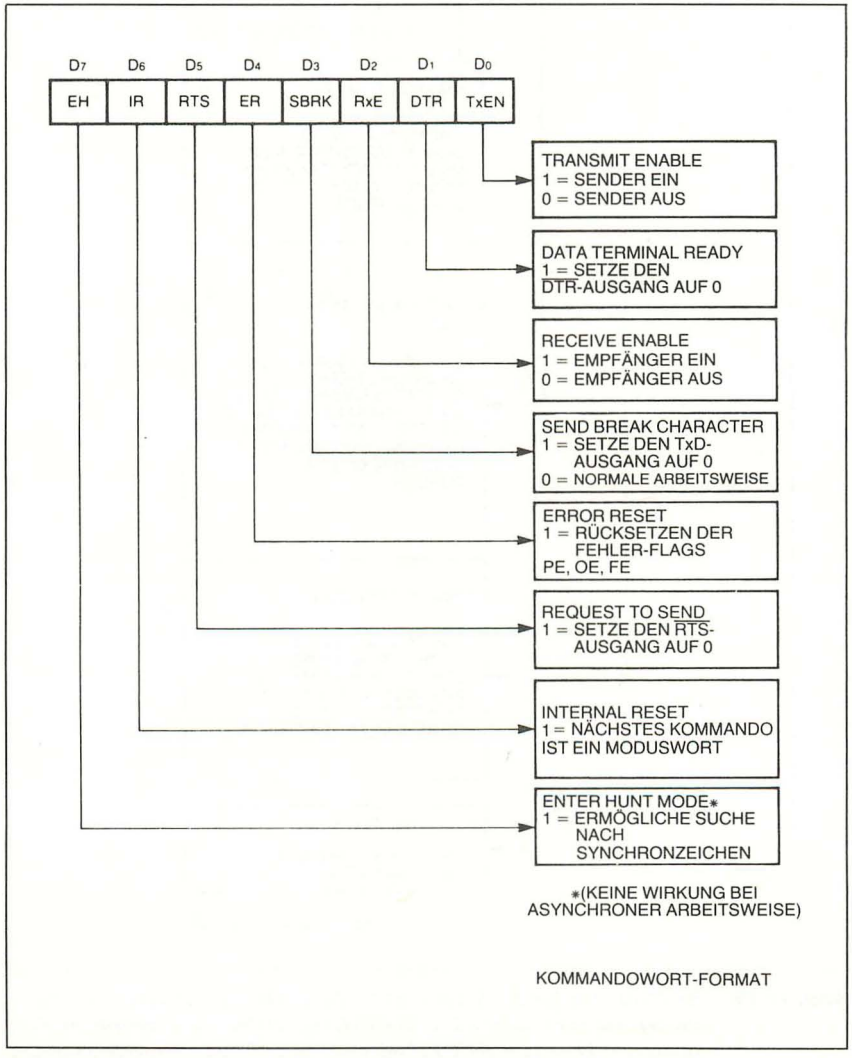


Abb. 10.17: *Bit-Definition für das Befehlswort des 8251.*

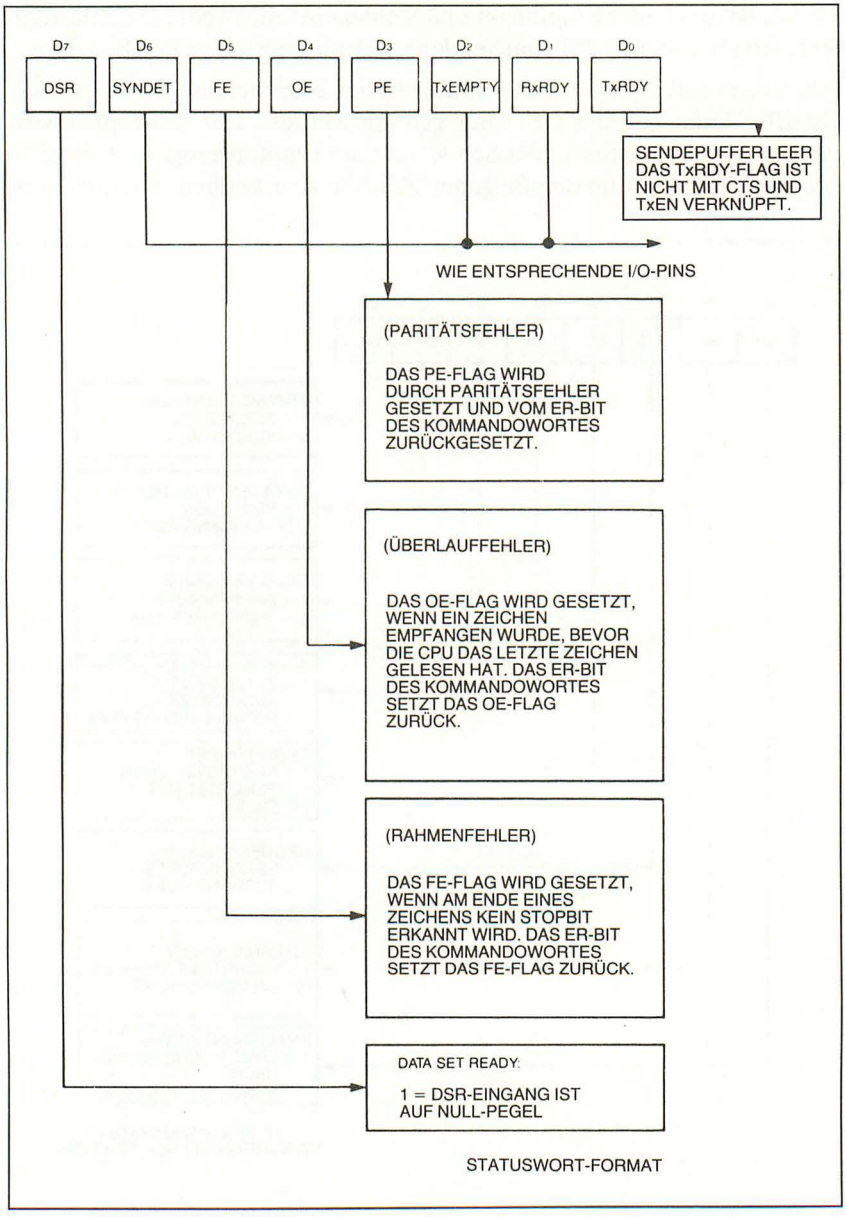


Abb. 10.18: *Bit-Definition für das Statuswort des 8251. Die einzelnen Bits informieren den Mikroprozessor über den elektrischen Status des Bausteins. Das Statuswort kann von der CPU mit einer einfachen Leseoperation geholt werden.*

nition des Statusregisters ist in Abb. 10.18 zu sehen. Das Statusregister wird mit einer einfachen Lese-Operation abgefragt; der C/D-Eingang des 8251 muß dazu logisch 1 sein.

Rahmenfehler

Wenn Sie sich mit serieller Übertragung noch nicht auskennen, mag eine Beschreibung der möglicherweise auftretenden Übertragungsfehler hilf-

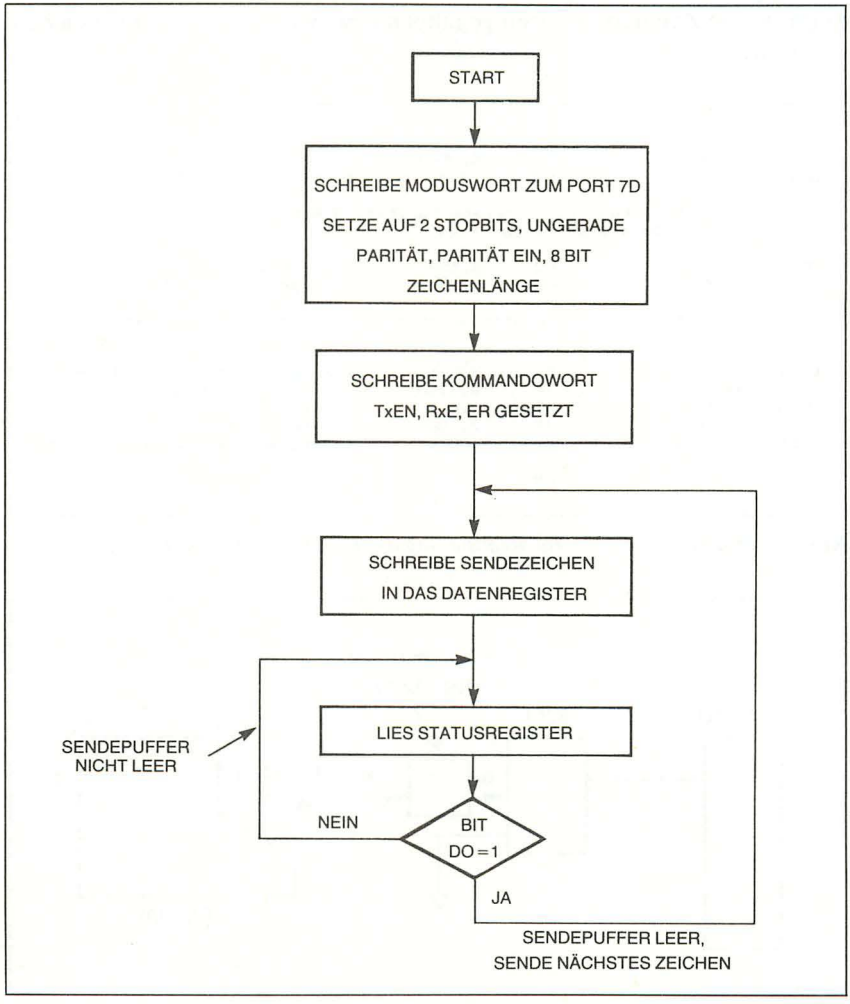


Abb. 10.19: *Flußdiagramm des Ablaufs beim wiederholten Senden eines Zeichens zu einem anderen seriellen Baustein.*

reich sein. Der erste Fehlertyp, den wir besprechen wollen, ist der sogenannte Rahmenfehler (framing error). Dieser Fehler tritt auf, wenn ein Zeichen empfangen wurde und kein Stopbit im Bitstrom vorhanden war. Das deutet darauf hin, daß das ganze Zeichen fehlerhaft ist.

Überlauffehler

Ein Überlauffehler (overrun error) tritt auf, wenn ein neues Zeichen empfangen wurde, bevor das alte von der CPU eingelesen war. Dadurch ist das letzte Zeichen verloren gegangen bzw. wurde vom neuen Zeichen überschrieben.

```
1800   3E FE            LD      A,0FEH       ; Modus-Wort
1802   D3 7D            OUT     (7DH),A      ; Ausgabe zum 8251

               ; 2 Stopp-Bits, gerade Parität, 8 Bits pro Zeichen
               ; Baudraten Faktor * 16

1804   3E 15            LD      A,15H        ; Steuerwort
1806   D3 7D            OUT     (7DH),A      ; Ausgabe zum 8251

               ; Sender und Empfänger einschalten, Fehler rücksetzen

1808   3E 49     LOOP1: LD      A,49H        ; ASCII-Zeichen "I"
180A   D3 7C            OUT     (7CH),A      ; Ausgabe zum Senderegister
180C   DB 7D     LOOP2: IN      A,(7DH)      ; lese Status-Register
180E   CB 47            BIT     0,A          ; Puffer leer (Bit 0 = 1) ?
1810   CA 180C          JP      Z,LOOP2      ; nein, warte
1813   C3 1808          JP      LOOP1        ; Ausgabe des nächsten Zeichens

                        END
```

Abb. 10.20: *Z80-Programm zur Realisierung des Flußdiagramms Abb. 10.19.*

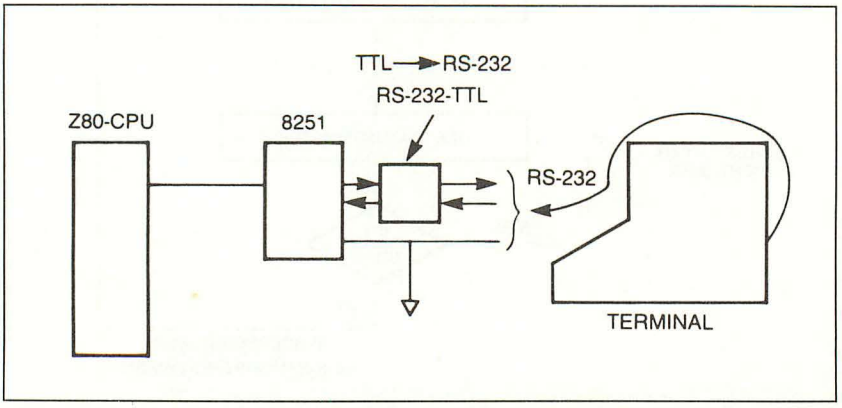


Abb. 10.21: *8251-USART in einem Hardware-Aufbau zur Kommunikation zwischen Mikroprozessor und Terminal.*

Ein einfaches Anwendungsprogramm für den 8251

Wir wollen jetzt ein einfaches Z80-Programm zur Benutzung des 8251 vorstellen. Das Programm setzt den Baustein auf und sendet dann wiederholt ein Zeichen zum Terminal. Obwohl es keinen praktischen Nutzen hat, zeigt das Programm doch die wichtigsten Punkte der Flag-Abfrage (polling). Abb. 10.19 zeigt das Flußdiagramm und Abb. 10.20 das dazugehörende Assembler-Listing.

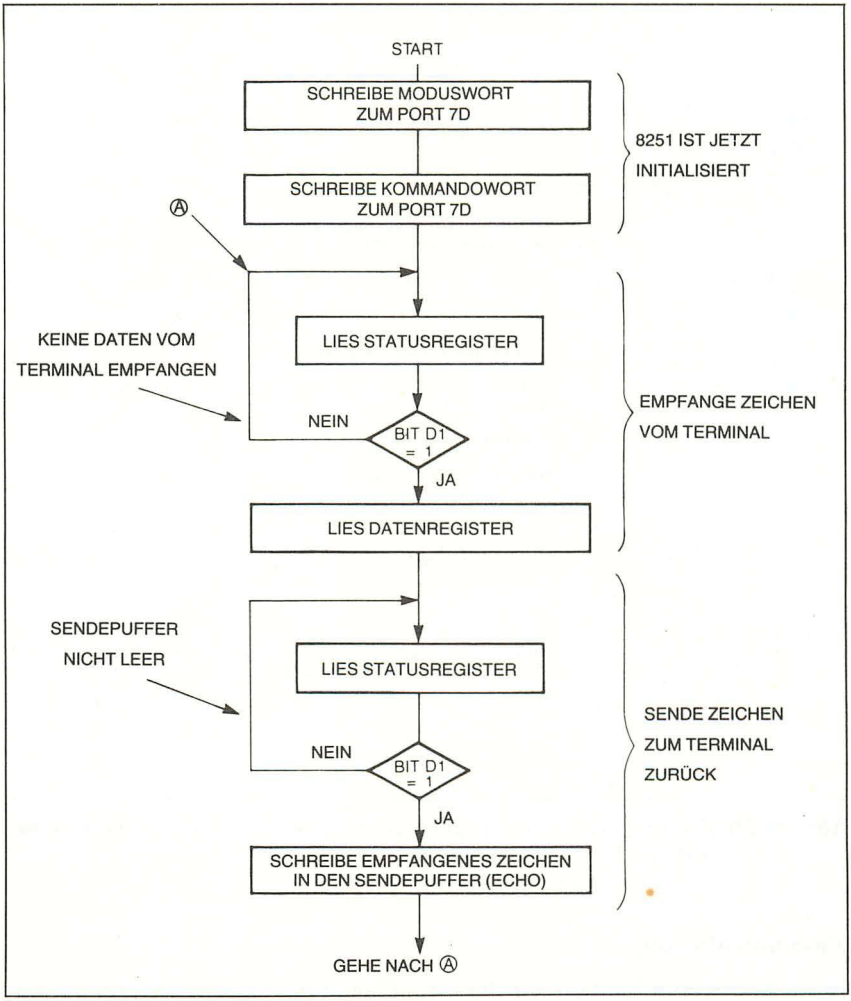


Abb. 10.22: *Flußdiagramm für ein Programm zum Aufsetzen des 8251 und zum Rücksenden der vom Terminal empfangenen Zeichen.*

Eine erweiterte Anwendung für den 8251

Wir wollen jetzt ein Z80-Programm untersuchen, das Zeichen von einem Terminal empfängt und zurückschickt (Echo-Betrieb). Die Arbeitsweise des 8251 beim Empfangen und Senden wird dadurch deutlich. Abb. 10.21 zeigt das Blockschaltbild des Hardware-Aufbaus. Das Flußdiagramm für das Programm sehen Sie in Abb. 10.22, das Z80-Programm selber in Abb. 10.23.

```
HE

1800   3E FE            LD      A,0FEH          ; Modus-Wort
1802   D3 7D            OUT     (7DH),A         ; Ausgabe zum 8251
1804   3E 15            LD      A,15H           ; Steuerwort
1806   D3 7D            OUT     (7DH),A         ; Ausgabe zum 8251

               ; Der 8251 ist jetzt inititialisiert

               ; Als erstes wartet der Baustein auf ein vom
               ; Terminal gesendetes Zeichen.

1808   DB 7D     LOOP1: IN      A,(7DH)         ; lese Status-Register
180A   CB 4F            BIT     1,A             ; Zeichen im Puffer ?
180C   CA 1808          JP      Z,LOOP1         ; nein, warte

               ; ein Zeichen ist empfangen worden

180F   DB 7C            IN      A,(7CH)         ; lese das Zeichen
1811   47               LD      B,A

               ; Wir machen keine Fehlerprüfung

               ; Jetzt werden die Daten gesendet


1812   DB 7D     LOOP2: IN      A,(7DH)         ; lese Status-Register
1814   CB 47            BIT     0,A             ; Puffer leer (Bit 0 = 1) ?
1816   CA 1812          JP      Z,LOOP2         ; nein, warte

               ; Sender ist bereit, ein Zeichen auszugeben

1819   78               LD      A,B
181A   D3 7C            OUT     (7CH),A         ; Zeichen zum 8251
181C   C3 1808          JP      LOOP1           ; Lese nächstes Zeichen

               ; Ende der Echo-Routine

                        END
```

Abb. 10.23: *Z80-Programm zur Realisierung der Abläufe im Flußdiagramm aus Abb. 10.22.*

Zusammenfassung

In diesem Kapitel haben wir die Grundlagen der seriellen Kommunikation kennengelernt. Wir haben eine Reihe wichtiger Begriffe eingeführt wie Baudrate, Startbits, Stopbits und Paritätsbits. Außerdem wurden

zwei der möglichen Übertragungsfehler vorgestellt: Rahmenfehler und Überlauffehler.

Im zweiten Teil des Kapitels behandelten wir ein reales LSI-Bauteil für die serielle Kommunikation: den 8251. Wir haben dieses Bauteil ausgewählt, da es einfach zu verstehen ist und häufig in der Industrie eingesetzt wird. Es löst die Probleme der seriellen Kommunikation zudem sehr einfach und elegant.

Im Kapitel 11 werden wir noch einen weiteren seriellen Baustein besprechen: das Z80-SIO. Dieser Baustein ist noch etwas komplexer als der 8251. Die meisten aus Kapitel 10 bekannten Informationen können jedoch auch auf das Z80-SIO übertragen werden.

Kapitel 11

Benutzung des Z80-SIO

Einführung

In diesem Kapitel wollen wir den als Z80-SIO bekannten seriellen I/O-Baustein besprechen. Der Baustein hat viele verschiedene Operationsarten und kann für die verschiedensten Anwendungen eingesetzt werden. In diesem Kapitel wollen wir uns auf die häufigste Anwendungsart konzentrieren: als asynchroner I/O-Baustein. Wenn Sie verstanden haben, wie das Z80-SIO in diesem Modus programmiert und benutzt wird, werden Sie auch fähig sein, es in anderen Betriebsarten zu betreiben. (Anmerkung: In diesem Kapitel gehen wir davon aus, daß Sie die Grundlagen der asynchronen seriellen Übertragung bereits kennen. Sollte das nicht der Fall sein, sei hier auf das Kapitel 10 verwiesen.)

Blockschaltbild des Z80-SIO-Bausteins

Abb. 11.1 zeigt das Blockschaltbild des Z80-SIO. Lassen Sie es uns untersuchen und die einzelnen Blöcke besprechen. Wir wollen mit den beiden als Kanal A und Kanal B bezeichneten Blöcken beginnen. Sie zeigen, daß das SIO aus zwei unabhängigen seriellen Kommunikationskanälen besteht. Jeder Kanal hat seinen eigenen Steuer/Status-Block. Die Blöcke sind mit verschiedenen Eingangs- und Ausgangsleitungen verbunden. Wir werden diese Leitungen später im Kapitel besprechen.

Die Blöcke links neben den Kanalblöcken sind die Lese/Schreib-Register. In ihnen wird die Programmierung für eine bestimmte Operationsart abgelegt.

Alle Operationsarten des SIO sind durch Software anwählbar. Bevor das SIO zur Kommunikation eingesetzt werden kann, müssen die Lese/Schreib-Blöcke entsprechend aufgesetzt werden. Wir werden die Programmierung dieser Blöcke an späterer Stelle detailliert besprechen.

Ein weiterer Block in Abb.11.1 ist die Interrupt-Steuerlogik. Das Z80-SIO hat eine sehr leistungsfähige Interrupt-Architektur. Dieser Block bil-

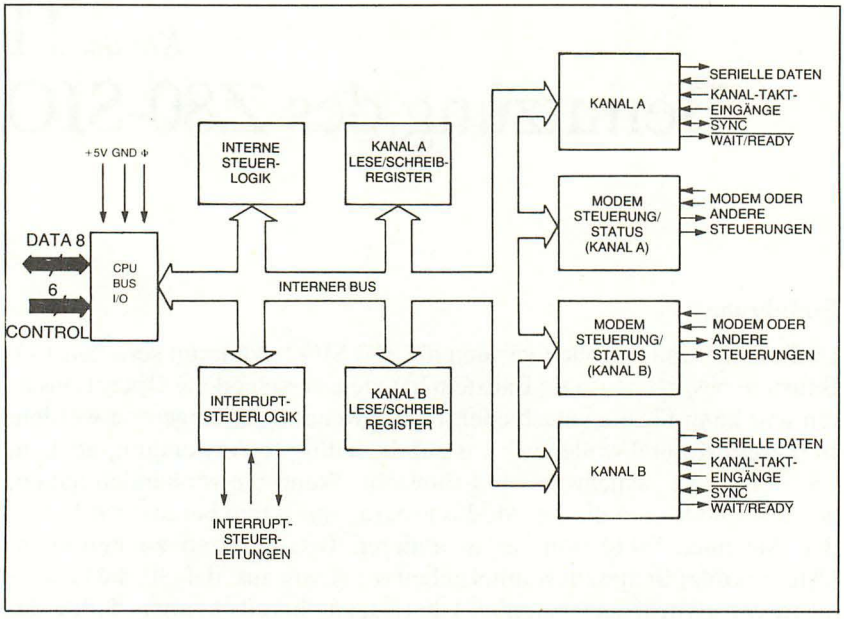


Abb. 11.1: *Blockschaltbild des Z80-SIO.*

det das Interface zur prioritätsgesteuerten Interrupt-Struktur in einem Z80-System.

Der als interne Steuerlogik bezeichnete Block dient der Steuerung des gesamten internen Datenbusses. Er sorgt für das korrekte Timing des gesamten internen Datentransfers des Bausteins.

Als letztes sehen wir in Abb. 11.1 den Block „CPU Bus-I/O". Er ist das elektrische Interface zwischen dem SIO und der CPU bzw. dem Systembus. Später werden wir auch sehen, wie das SIO mit dem Z80-Mikroprozessor verbunden wird.

Pindefinition des SIO

Abb. 11.2 zeigt die Pinbelegung für das Z80-SIO. Wir wollen jetzt jeden Pin besprechen.

D7-D0 Dies sind die Dateneingangs- und Ausgangsleitungen zur Verbindung mit dem System-Datenbus des Mikroprozessors. Der gesamte Informationsaustausch zwischen SIO und Z80 geht über diese Leitungen. D7 ist das höchstwertigste Bit.

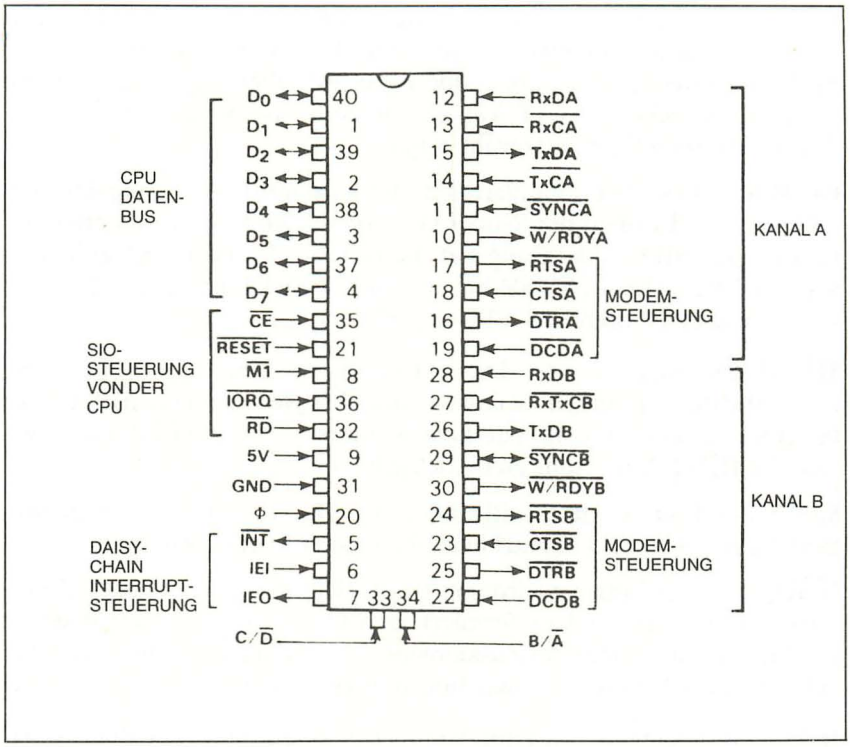


Abb. 11.2: *Pinbelegung des Z80-SIO.*

B/$\overline{\text{A}}$ *(Kanal B oder A Anwahl)* Ist die Leitung logisch 1, so verkehrt der Mikroprozessor mit dem Kanal B, ist sie logisch 0, so verkehrt er mit dem Kanal A. Die B/$\overline{\text{A}}$-Leitung wird sinnvollerweise mit der A0-Adreßleitung des Z80 verbunden.

C/$\overline{\text{D}}$ *(Steuerung/Daten-Anwahl)* Wenn diese Leitung logisch 1 ist, kommuniziert der Mikroprozessor mit den Steuerregistern des Kanals A oder B. Ist sie logisch 0, wird mit den Datenregistern des SIO kommuniziert. Die C/$\overline{\text{D}}$-Leitung wird sinnvollerweise mit der A1-Adreßleitung des Z80 verbunden.

Sind die Adreßleitungen A0 und A1 mit den SIO-Pins B/$\overline{\text{A}}$ und C/$\overline{\text{D}}$ verbunden, belegt der Baustein vier verschiedene Port-Adressen. Diese sind: Kanal A Daten, Kanal B Daten, Kanal A Steuerung und Kanal B Steuerung. (Anmerkung: Die Adressierung des SIO werden wir im nächsten Abschnitt noch näher behandeln.)

$\overline{CE}$ Dies ist der logisch 0 aktive Baustein-Freigabe-Eingang. Der Eingang muß zur Kommunikation mit dem Mikroprozessor aktiv sein. Er wird sinnvollerweise aus den oberen 6 Adreßleitungen des unteren Adreßbytes deckodiert (A7-A2). Die unteren zwei Adreßleitungen werden zur internen Registeranwahl benutzt.

CLOCK Dies ist der Standardtakt, der auch am Takteingang der Z80-CPU anliegt. Er dient der Synchronisation der SIO-internen Kommunikation. Der SIO-Takteingang hat die gleiche elektrische Spezifikation wie der CPU, also Vcc-0,6V für logisch 1. Beachten Sie, daß dies eine Abweichung zum Standard-TTL-Pegel ist.

$\overline{M1}$ *(Maschinen-Zyklus 1)* Dieser Eingang wird mit dem $\overline{M1}$-Ausgang der Z80-CPU verbunden. Wenn $\overline{M1}$ und $\overline{RD}$ gleichzeitig logisch 0 sind, bedeutet das eine Befehlscode-Leseoperation. Das SIO ist dafür ausgelegt, die RETI-Anweisung zu berücksichtigen.

Sind $\overline{M1}$ und $\overline{IORQ}$ gleichzeitig logisch 0, liefert die CPU eine Interrupt-Bestätigung. Das SIO benutzt auch diesen elektrischen Zustand.

$\overline{IORQ}$ Dieser Eingang wird mit dem $\overline{IORQ}$-Ausgang des Z80-Mikroprozessors verbunden. Er informiert das SIO elektrisch darüber, daß eine I/O-Operation vorliegt. Im Zusammenhang mit dem $\overline{M1}$-Eingang dient er außerdem der Erkennung einer Interrupt-Bestätigung.

$\overline{RD}$ Dieser Eingang wird mit dem $\overline{RD}$-Ausgang der Z80-CPU verbunden. Er informiert das SIO darüber, daß die CPU eine Speicher- oder I/O-Leseoperation durchführt. Sind $\overline{RD}$ und $\overline{M1}$ logisch 0, so liest sie einen Befehl aus dem Speicher. Sind $\overline{RD}$, $\overline{IORQ}$ und $\overline{CE}$ logisch 0, so liest die CPU mit einer Eingabeoperation Daten aus dem SIO. Die Quelle im SIO wird durch die C/$\overline{D}$- und die B/$\overline{A}$-Leitung bestimmt.

Ist der $\overline{RD}$ -Eingang logisch 1 und $\overline{IORQ}$ und $\overline{CE}$ logisch 0, so liegt eine Ausgabeoperation der CPU zum SIO vor. Einen speziellen $\overline{WR}$-Schreibeingang gibt es nicht.

$\overline{RESET}$ Eine logische 0 an diesem Eingang schaltet Sender und Empfänger beider Kanäle aus. Die Sendeausgänge beider Kanäle gehen in den Markierungszustand. Alle Modem-Steuerausgänge werden logisch 1. Interrupts sind abgeschaltet. Das SIO muß nach einem Hardware-Reset software-mäßig neu aufgesetzt werden.

IEI *(Interrupt-Freigabe-Eingang)* Nur wenn dieser Eingang logisch 1 ist, kann das SIO Interrupt-Anforderungen stellen. Er wird, angeschlossen an den IEO-Ausgang eines anderen Bausteins, zur Bildung der Interrupt-Prioritätskette benutzt.

IEO *(Interrupt-Freigabe-Ausgang)* Dieser Ausgang ist logisch 1 aktiv. Er informiert in einer Interrupt-Prioritätskette die nachfolgenden Bausteine, daß keine Interrupt-Anforderung eines höherpriorisierten Bausteins vorliegt. In Systemen ohne Interrupt-Daisy-Chain bleibt der Ausgang unbeschaltet.

$\overline{\text{INT}}$ *(Interrupt-Anforderungs-Ausgang)* Dies ist der Open-Drain-Ausgang des SIO zur direkten Verbindung mit dem $\overline{\text{INT}}$-Eingang der Z80-CPU. Stellt das SIO eine Interrupt-Anforderung, wird der Ausgang logisch 0, sonst ist er offen. Ein Widerstand zu +5V (pull-up) legt den $\overline{\text{INT}}$-Eingang der CPU auf logisch 1-Potential, solange er nicht aktiv ist.

$\overline{\text{W/RDYA}}$, $\overline{\text{W/RDYB}}$ *(Warte- bzw. Fertig-Ausgang)* Dies sind die WAIT/READY- Ausgänge der Kanäle A und B des SIO. Die Funktion der Ausgänge wird von der Programmierung bestimmt. Als Ready-Ausgänge werden sie in Verbindung mit DMA-Steuerbausteinen benutzt, um zu melden, daß das SIO Daten senden oder empfangen kann. Als Wait-Ausgänge können sie zur Synchronisation der CPU dienen.

$\overline{\text{CTSA}}$, $\overline{\text{CTSB}}$ *(Clear to Send A,B)* Dies sind 0-aktive Eingänge. Durch entsprechende Programmierung können sie zur automatischen Senderfreigabe benutzt werden. Sie können jedoch auch als frei verwendbare Eingänge programmiert werden.

DCDA, DCDB *(Data Carrier Detect A,B)* Diese Eingänge können, abhängig von der Programmierung, zur automatischen Empfängerfreigabe benutzt werden.

$\overline{\text{RTSA}}$, $\overline{\text{RTSB}}$ *(Request to Send A,B)* Diese Ausgänge können vom Programm auf logisch 0 (aktiv) gesetzt werden. Zurückgesetzt werden sie im Asynchron-Modus jedoch erst, nachdem der Sendepuffer leer ist.

$\overline{\text{DTRA}}$, $\overline{\text{DTRB}}$ *(Data Terminal Ready A,B)* Diese Ausgänge können vom Programm gesetzt und zurückgesetzt werden.

Anmerkung: Die letzten vier Signale $\overline{\text{CTS}}$, $\overline{\text{DCD}}$, $\overline{\text{RTS}}$ und $\overline{\text{DIR}}$ können zur Modem-Steuerung benutzt werden. Das Blockschaltbild in Abb. 11.3 zeigt die Verbindung des SIO mit einem Modem.

RxDA, RxDB Dies sind die nicht-invertierten seriellen Dateneingänge.

TxDA, TxDB Dies sind die nicht-invertierten seriellen Datenausgänge.

$\overline{\text{RxCA}}$, $\overline{\text{RxCB}}$ Dies sind die Baudraten-Takteingänge der Empfänger. Die Taktfrequenz kann im Asynchronbetrieb, von der Programmierung abhängig, das 1-, 16-, 32- oder 64-fache der Baudrate sein.

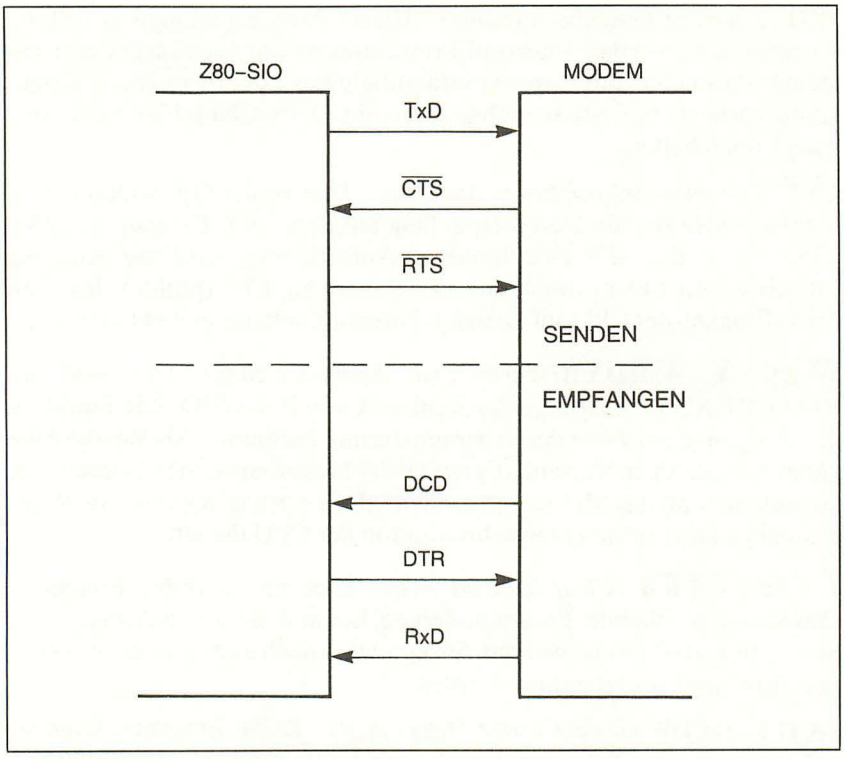


Abb. 11.3: *Verbindung des SIO mit einem Modem.*

TxCA, TxCB Dies sind die Baudraten-Takteingänge der Sender. Die Taktfrequenz kann im Asynchronbetrieb, von der Programmierung abhängig, das 1-, 16-, 32- oder 64-fache der Baudrate sein. Diese Teiler-Faktoren müssen für den Sender und Empfänger eines Kanals gleich sein. Durch unterschiedliche Frequenzen an den Takteingängen können jedoch Sende- und Empfangs-Baudrate verschieden sein.

Verbindung des SIO mit dem Z80-Systembus

Abb. 11.4 zeigt eine typische Verbindung zwischen SIO und Z80-Mikroprozessor. Es wird in dieser Schaltung angenommen, daß die Busbelastung keine Pufferung erfordert.

Der $\overline{CE}$-Eingang Pin 35 wird aus den Adreßleitungen A7–A2 dekodiert. Dafür wird ein 74LS138 benutzt. Die $\overline{CE}$-Leitung wird aktiv, wenn die

Adreßleitungen den Zustand 100011XX annehmen. XX gibt an, daß der Zustand von A1 und A0 für $\overline{CE}$ ohne Bedeutung ist. Sie sind mit dem C/$\overline{D}$- und dem B/$\overline{A}$-Eingang verbunden.

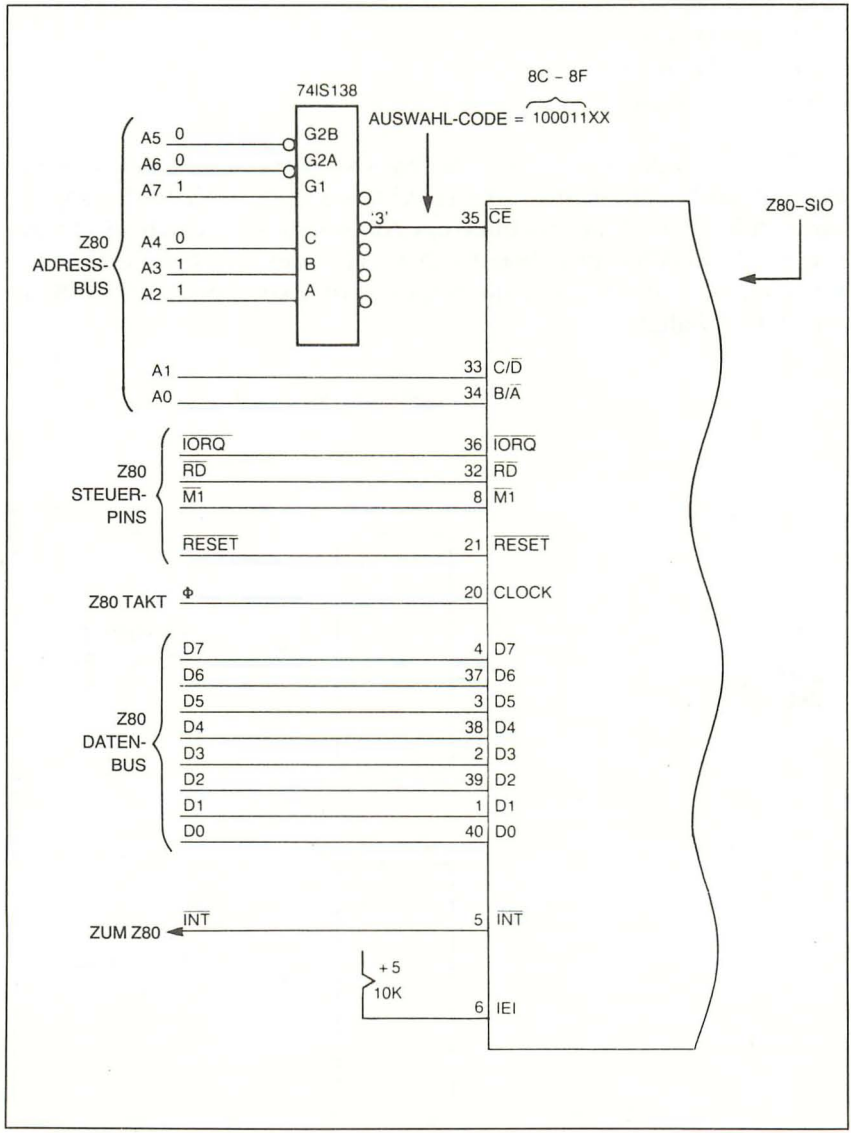


Abb. 11.4: *Die komplette Verbindung zwischen Z80-CPU und Z80-SIO. In diesem Fall werden keine Datenbuspuffer eingesetzt.*

Basierend auf der elektrischen Verbindung und Decodierung sind die Portadressen des SIO 8C, 8D, 8E und 8F. Sie verteilen sich folgendermaßen:

8C = A Daten
8D = B Daten
8E = A Steuerung
8F = B Steuerung

Der $\overline{INT}$-Ausgang des SIO ist mit dem $\overline{INT}$-Eingang des Z80 verbunden. IEI wird über einen 10 KOhm-Widerstand auf +5V hochgezogen. Dadurch kann der Baustein Interrupt-Anforderungen stellen. IEO wird in dieser Anwendung nicht benutzt. Abb. 11.5 zeigt, wie IEI und IEO bei Benutzung des SIO in einer Interrupt-Prioritätskette mit anderen Bausteinen verschaltet werden.

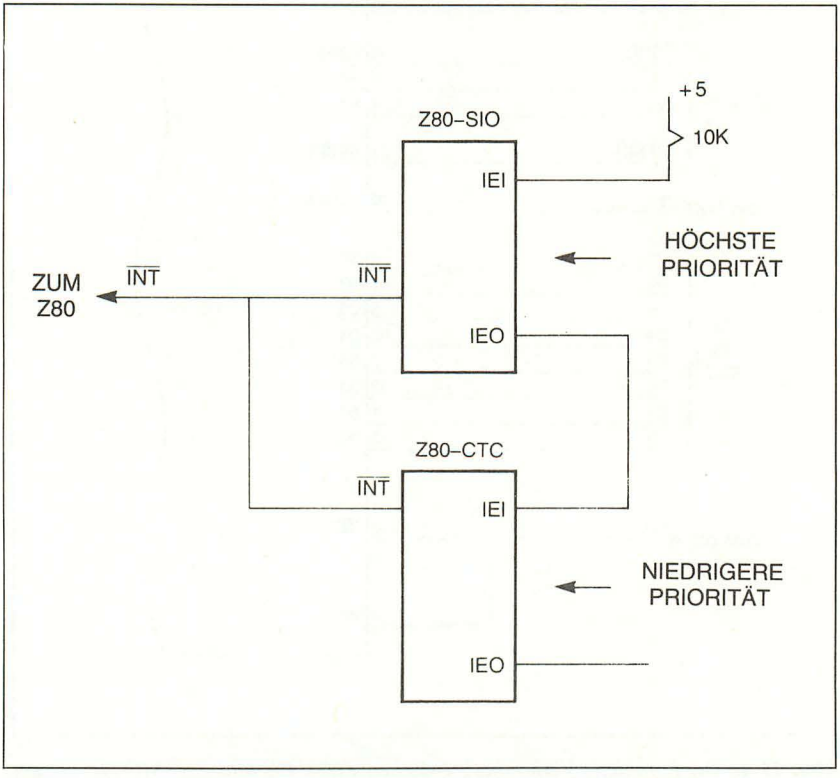


Abb. 11.5: *Einsatz des SIO in einer Interrupt-Prioritätskette.*

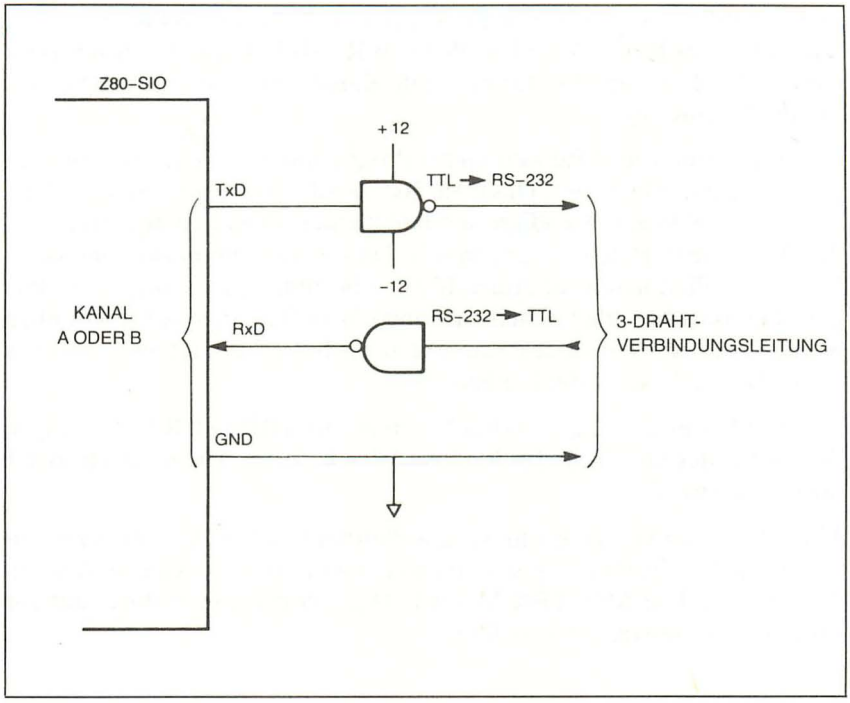


Abb. 11.6: *Einsatz von Pegel-Umsetzern als Interface zwischen den TTL-Pegeln des SIO und den RS-232-Leitungen.*

Verbindung des SIO mit den seriellen Leitungen

Bei dieser Verbindung gehen wir davon aus, daß beide SIO-Kanäle asynchrone serielle Kommunikation benutzen. Die elelektrische serielle Verbindung geht über RS-232. Abb. 11.7 zeigt die Verbindungen für diesen SIO-Modus. Es muß nur die TxD-, RxD-, $\overline{\text{TxC}}$- und $\overline{\text{RxC}}$-Leitung jedes Kanals beschaltet werden.

Die SIO-Register

Nachdem die Hardware, wie im vorigen Abschnitt beschrieben, verbunden ist, ist es jetzt unsere Aufgabe, das SIO so zu programmieren, daß es in der gewünschten Weise mit anderen seriellen Bausteinen kommunizieren kann. Im folgenden Abschnitt wollen wir von den SIO-internen Registern Gebrauch machen und einige Programmbeispiele zum Aufsetzen des SIO vorstellen.

Das Z80-SIO hat acht interne Schreibregister im Kanal B (WR0–WR7) und sieben im Kanal A (WR0, WR1, WR3–R7). Schreibregister WR2 enthält den Interrupt-Vektor für beide Kanäle. Das höchstwertigste Bit für alle Register ist D7.

Mit Ausnahme von WR0 werden zur Programmierung jedes Schreibregisters zwei Bytes benötigt. Das erste Byte wird in WR0 geschrieben. Dieses Byte wird intern decodiert und bildet einen Zeiger zu dem Register, das das nächste Byte empfangen soll. Dies ist deshalb notwendig, da es keine Adreßleitungen zur Auswahl eines bestimmten internen Registers gibt. Die Basisbefehle können auch mit einem Byte in WR0 geschrieben werden. Sie werden dieses Konzept noch besser verstehen, wenn wir einige Beispiele vorgestellt haben.

Kanal B hat einen Satz von drei Leseregistern (RR0–RR2). Sie zeigen den Status des SIO. Kanal A hat kein RR1-Register, wie Sie auch später noch sehen werden.

Mit Hilfe dieser kurzen Einführung wollen wir uns dem Anhang A zuwenden, wo jedes einzelne Register genauer beschrieben ist. Anhang A ist ein Auszug aus dem MOSTEK-Manual. Die Informationen dort sind zur Programmierung des SIO wichtig.

Initialisierung des SIO

Wir wollen jetzt die zur Initialisierung des SIO notwendige Software vorstellen. Wir betrachten dabei ein System mit asynchroner serieller Datenübertragung im Abfragebetrieb (polling).

Lassen Sie uns ein paar allgemeine Merkmale des SIO betrachten. Zuerst wollen wir ein paar wichtige Punkte, die die internen Register des SIO betreffen, näher anschauen (detaillierter beschrieben im Anhang A). Als erstes stellen wir fest, daß das SIO wahlweise 5, 6, 7 oder 8 Bit pro Zeichen senden oder empfangen kann. Die Anzahl der Bits ist softwaremäßig programmierbar.

Beim asynchronen Senden setzt das SIO automatisch ein Startbit, logisch 0, an den Anfang jedes Zeichens. Ebenso werden am Ende jedes Zeichens 1, 1½ oder 2 Stopbits eingefügt. Die Anzahl der Stopbits ist programmierbar. Der Pegel ist logisch 1, was dem Markierungszustand des SIO entspricht.

Vor den Stopbits kann wahlweise ein Paritätsbit eingefügt werden. (Anmerkung: Das Konzept der Paritätsbits wurde in Kapitel 10 vorgestellt.) Wenn Parität programmiert wurde, berücksichtigt das SIO wahl-

weise gerade oder ungerade Parität. Wenn Parität nicht programmiert wurde, wird auch kein Paritätsbit eingesetzt.

Liest die CPU Daten vom SIO, so sind das natürlich immer acht Bit. Es müssen jedoch nicht notwendigerweise alle Bits zum empfangenen Zeichen gehören. Wenn zum Beispiel das empfangene Zeichen nur sechs Bit hat, sind die ersten sechs Bit (D0–D5) des gelesenen Bytes das Zeichen, wobei D0 das niederwertigste Bit ist. Die übrigen Bits sind die Stopbits und ggf. auch das Paritätsbit. Die Stopbits sind immer logisch 1. Sind mehrere Bits unbenutzt, so werden sie auf den Markierungspegel (logisch 1) gesetzt.

Wurde zum Beispiel das SIO für eine Zeichenlänge von 5 bit, keine Parität und 1 Stopbit programmiert, so sind D5, D6 und D7 des von der CPU gelesenen Bytes logisch 1.

Das SIO hat vier Takteingänge für die serielle Kommunikation sowie einen Takteingang zur Steuerung interner Datentransfers. Die vier Takte für die Kommunikation heißen RxA, RxB, TxA und TxB. Die Taktfrequenz an diesen Eingängen kann das 1-, 16-, 32- oder 64-fache der seriellen Empfangs- oder Sende-Baudrate sein.

Wir wollen jetzt, wo wir einige grundsätzliche Fähigkeiten des SIO kennengelernt haben, mit unserem Beispiel asynchroner Kommunikation fortfahren. Die folgende Sequenz zur Registerinitialisierung sollte für den Asynchron-Modus in etwa eingehalten werden:

1. *Beschreiben des WR0.* Dieser Vorgang setzt das SIO zurück. Für einen kompletten Baustein-Reset und zur Programmierung einiger spezieller Befehle können mehrere Bytes benötigt werden.
2. *Beschreiben von WR1.* Dieser Vorgang setzt den Empfangs- und Sendetakt-Vorteiler auf und programmiert Stopbits und Parität.
3. *Beschreiben von WR3.* Dieser Vorgang programmiert die Anzahl der Empfangsbits, setzt Auto-Enable und schaltet den Empfänger ein.
4. *Beschreiben von WR5.* Dieser Vorgang setzt die Anzahl der Sendebits auf und schaltet den Sender ein.
5. *Beschreiben von WR2.* Dies gilt nur für Kanal B. Im Falle eines interrupt-getriebenen Systems wird hiermit der Interrupt-Vektor aufgesetzt.
6. *Beschreiben von WR1.* Dieser Vorgang spezifiziert den Interrupt-Betrieb des SIO. Es dient außerdem zum Ein- und Ausschalten der Interrupt-Fähigkeit.

Die Register WR6 und WR7 werden beim asynchronen Betrieb nicht benötigt. Sie beinhalten spezielle Informationen zur synchronen Kommunikation.

Um zu zeigen, wie die vorstehende Initialisierung in einem typischen System mit SIO funktioniert, wollen wir ein Beispiel vorstellen. Wir setzen den Kanal A des SIO folgendermaßen auf:

- Baudraten-Faktor 64
- 2 Stopbits
- 8 Bit pro Datenzeichen
- keine Interrupts
- keine Parität

Ein Z80-Programm, das das SIO mit den vorstehenden Werten aufsetzt, ist in Abb. 11.7 zu sehen. Das SIO hat dabei die I/O-Adressen 8CH, 8DH, 8EH und 8FH. Abb. 11.3 zeigt die elektrische Decodierung dafür.

```
                ; Dies ist ein Z80-Programm zum initialisieren der SIO
                ; *64 Takt, 2 Stopp-Bits, 8 Rx-Bits, 8 Tx-Bits, kein Int.


                ; Port-Vereinbarungen:

008C              ADATA   EQU     8CH     ; SIO Kanal A Daten
008D              BDATA   EQU     8DH     ; SIO Kanal B Daten
008E              ACON    EQU     8EH     ; SIO Kanal A Steuerung
008F              BCON    EQU     8FH     ; SIO Kanal B Steuerung

1800  0E 8E              LD      C,ACON  ; Port-Adresse
1802  3E 18              LD      A,18H
1804  ED 79              OUT     (C),A   ; WR0, Kanal Reset
1806  3E 04              LD      A,4
1808  ED 79              OUT     (C),A   ; Zeiger zu WR4
180A  3E CC              LD      A,0CCH
180C  ED 79              OUT     (C),A   ; *64 Takt, 2 Stopp-Bits, keine Par.
180E  3E 03              LD      A,3
1810  ED 79              OUT     (C),A   ; Zeiger zu WR3
1812  3E C1              LD      A,0C1H
1814  ED 79              OUT     (C),A   ; 8 Empf. Bits, Empf. ein
1816  3E 05              LD      A,5
1818  ED 79              OUT     (C),A   ; Zeiger zu WR5
181A  3E 68              LD      A,68H
181C  ED 79              OUT     (C),A   ; 8 Sende-Bits, Sender ein
181E  3E 01              LD      A,1
1820  ED 79              OUT     (C),A   ; Zeiger zu WR1
1822  3E 00              LD      A,0
1824  ED 79              OUT     (C),A   ; Interrupts aus

                ; Nach Ausführung dieser Anweisungen ist der SIO-Kanal A
                ; initialisiert und zur Kommunikation bereit.

                        END
```

Abb. 11.7: *80-Programm zur Initialisierung des Z80-SIO zur asynchronen sequentiellen Kommunikation.*

Empfangen serieller Daten

Nachdem das SIO jetzt initialisiert ist, können wir serielle Daten empfangen und mit der CPU einlesen. Im Abfragemodus (polling) gehen wir davon aus, daß der Z80 die Zeichen ebenso schnell abholen kann, wie sie über die serielle Leitung übertragen werden. Ist das nicht der Fall, wird eine Art Handshake benötigt, um sicherzustellen, daß die Daten nicht schneller übertragen werden als sie von der CPU verarbeitet werden können. Sollte das nämlich passieren, so würde der Empfänger einen Überlauffehler melden.

Das Z80-SIO hat im Empfängerteil einen eingebauten FIFO-Speicher (first- in, first- out) mit einer Tiefe von drei Bytes. Diese drei Bytes können empfangen werden, bevor das Überlaufproblem entsteht. Dadurch bekommt die CPU mehr Zeit zur Verarbeitung einzelner Zeichen.

In der Empfangsroutine prüfen wir nach jedem Zeichen auf Überlauffehler. Wenn ein Fehler auftritt, muß die CPU das Fehler-Flag des SIO zurücksetzen, bevor sie das nächste Zeichen prüfen kann. Die Fehler-Flags bleiben gesetzt, bis sie explizit zurückgesetzt werden. Ein Flußdiagramm für den Zeichenempfang ist in Abb. 11.8 zu sehen.

Wie wir der Abb. 11.8 entnehmen, wird als erste Operation das Statusregister RR0 gelesen. Das wird getan, um festzustellen, ob der Eingangspuffer voll ist. Wenn mindestens ein Zeichen im FIFO steht, ist Bit D0 von RR0 logisch 1. Ist Bit D0 logisch 0, bleibt die CPU in einer Warteschleife, um auf ein Zeichen zu warten (oder sie erledigt andere Aufgaben und kehrt später zu dieser Abfrage zurück). In Abb. 11.8 wartet die CPU so lange in der Schleife und fragt den Status ab, bis ein Zeichen empfangen wurde.

Nachdem ein Zeichen empfangen wurde und D0 von RR1 logisch 1 ist, wird das Zeichen gelesen. Dann wird RR1 auf eventuell aufgetretene Fehler abgefragt. Bit D6, D5 und D4 von RR1 zeigen Fehler an. D6 steht für einen Rahmenfehler, D5 für einen Überlauffehler und D4 für einen Paritätsfehler. Wenn ein Fehler auftrat, muß ein Fehler-Reset durch Beschreiben von WR0 durchgeführt werden.

Treten Fehler auf, kann der Fehlertyp auf der Konsole mitgeteilt werden oder eine andere Aktion, die für die entsprechende Anwendung sinnvoll ist, durchgeführt werden.

Ein Programm zur Realisierung des Flußdiagramms zum Zeichenempfang ist in Abb. 11.9 zu sehen.

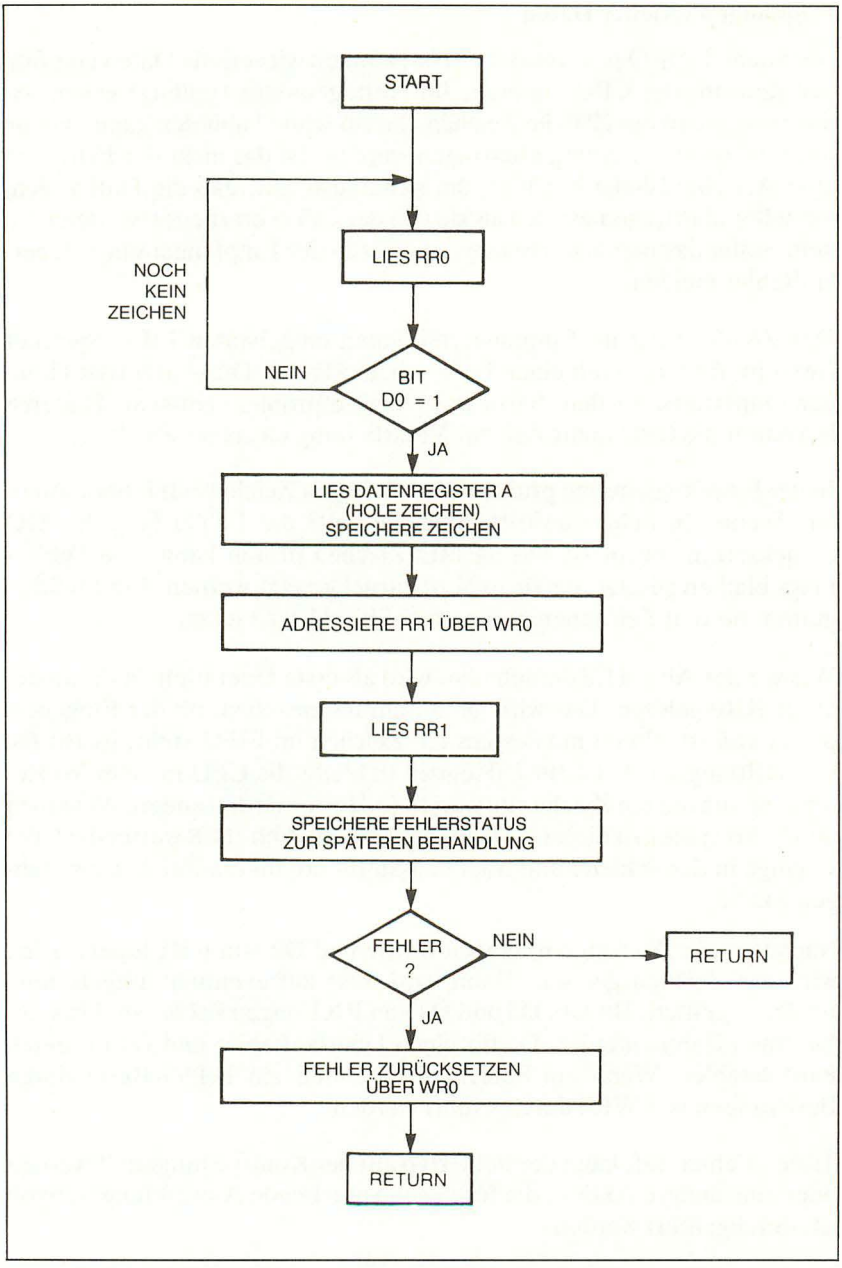


Abb. 11.8: *Flußdiagramm des Ereignisablaufs beim Lesen eines Zeichens vom Z80-SIO.*

```
                ;-------------------------------------------------------
                ; Diese Routine empfängt einzelne Zeichen im Abfrage-Modus
                ; (polling). Danach springt sie zum aufrufenden Programm
                ; zurück. Das empfangene Zeichen wird im Speicher unter der
                ; symbolischen Adresse "CHIN" abgelegt; der Fehlerstatus
                ; unter der Adresse "CHER". Wenn Bit D6, D5 und D4 von "CHER"
                ; = 0 sind, liegt kein Fehler vor.
                ;-------------------------------------------------------

008C                ADATA   EQU     8CH     ; SIO Kanal A Daten
008E                ACON    EQU     8EH     ; SIO Kanal A Steuerung
5900                CHIN    EQU     5900H   ; Speicherplatz für Zeichen
5901                CHER    EQU     CHIN+1  ; Speicherplatz für Fehlerstatus


2400    DB 8E       LOOP:   IN      A,(ACON)        ; lese RR0
2402    E6 01               AND     01H             ; prüfe Bit D0
2404    CA 2400             JP      Z,LOOP          ; kein Zeichen, warte

                ; Jetzt kann das Zeichen eingelesen werden

2407    DB 8C               IN      A,(ADATA)       ; hole das Zeichen
2409    32 5900             LD      (CHIN),A        ; speichere Zeichen

                ; Jetzt prüfen wir auf Fehler beim empfangen

240C    3E 01               LD      A,01H
240E    D3 8E               OUT     (ACON),A        ; Zeiger auf WR1
2410    DB 8E               IN      A,(ACON)        ; lese Fehler-Register
2412    32 5901             LD      (CHER),A        ; speicher Fehler-Status
2415    E6 70               AND     70H             ; nur D6, D5 und D4
2417    C8                  RET     Z               ; zurück wenn kein Fehler

                ; Wenn keine Fehler auftraten, springen wir jetzt zurück.
                ; Sind Fehler aufgetreten, so werden sie vorher rückgesetzt.

2418    3E 30               LD      A,30H           ; "Fehler rücksetzen"
241A    D3 8E               OUT     (ACON),A        ; zum WR0
241C    C9                  RET

                            END
```

Abb. 11.9: *Z80-Programm zur Realisierung des Flußdiagramms von Abb. 11.8.*

Senden eines Zeichens im Abfragemodus

Im vorigen Abschnitt haben wir ein Programm zum Empfangen von Zeichen im Abfragemodus beschrieben. Wir wollen jetzt ein Programm zum Senden von Zeichen im Abfragemodus vorstellen. Abb. 11.10 zeigt das Flußdiagramm dazu.

Die erste Operation in Abb. 11.10 ist das Lesen von RR0. Dieses Register kann gelesen werden, ohne daß vorher ein Zeiger in WR0 geschrieben werden muß. Wir testen Bit D2 von RR0.

Ist Bit D2 logisch 1, so ist der Sendepuffer leer. Ist es logisch 0, so ist der Sendepuffer noch voll. Das Programm muß warten, bis der Puffer leer ist. Erst dann kann ein Zeichen zum Sendepuffer geschrieben werden, und

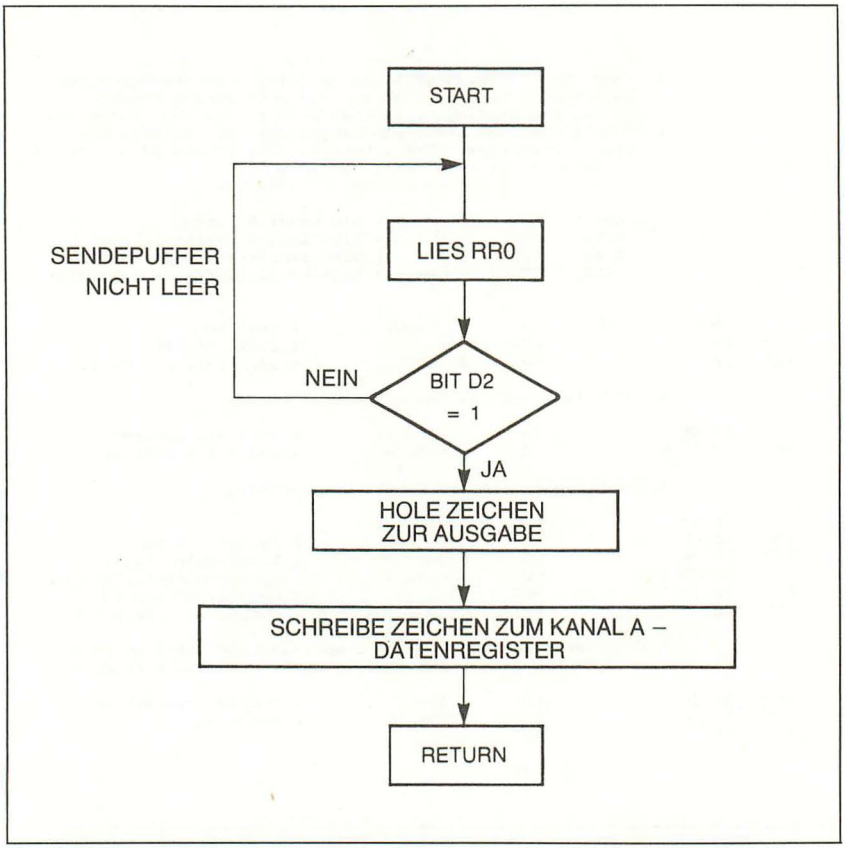


Abb. 11.10: *Flußdiagramm des Ereignisablaufs beim Senden eines Zeichens über das Z80-SIO.*

das SIO startet automatisch mit der seriellen Übertragung. Abb. 11.11 zeigt das Z80-Programm zur Realisierung des Flußdiagramms von Abb. 11.10.

Interrupts beim Z80-SIO

Jetzt, wo wir wissen, wie Zeichen im Abfragemodus gesendet und empfangen werden, wollen wir sehen, wie das SIO in einem interrupt-getriebenen System arbeitet. Wir beginnen mit einer Rückschau auf die SIO-Parameter. Wir untersuchen dann Beispiele zum Aufsetzen des SIO und zum Senden und Empfangen von Zeichen im Interrupt-Betrieb. Zum

```
                ;-------------------------------------------------------
                ;
                ; Diese Routine sendet einzelne Zeichen im Abfrage-Modus
                ; (polling) zum SIO-Kanal A.
                ;
                ; Das zu sendende Zeichen ist auf Speicherplatz "CHTX"
                ; abgelegt. Wir nehmen an, daß die SIO inititialisiert
                ; ist und im I/O-Bereich 8C-8F liegt.
                ;
                ;-------------------------------------------------------

008C                ADATA   EQU     8CH     ; SIO Kanal A Daten
008E                ACON    EQU     8EH     ; SIO Kanal A Steuerung
5902                CHTX    EQU     5902H   ; Speicherplatz für Zeichen


2500    DB 8E       LOOP1:  IN      A,(ACON)        ; lese RR0
2502    CB 57               BIT     2,A             ; prüfe Bit D2
2504    CA 2500             JP      Z,LOOP1         ; nicht fertig, warte

                ; Der Ausgabepuffer ist jetzt leer und kann ein
                ; Zeichen aufnehmen.

2507    3A 5902             LD      A,(CHTX)        ; lade Zeichen in A
250A    D3 8C               OUT     (ADATA),A       ; zum Sendepuffer

                ; Das Zeichen ist nun im Sendepuffer und wir kehren aus
                ; der Routine zurück.

250C    C9                  RET
                            END
```

Abb. 11.11: *Z80-Programm zur Realisierung des Flußdiagramms aus Abb. 11.10.*

Ende des Kapitels sollten Sie eine gute Vorstellung über den Einsatz des SIO in einem interrupt-getriebenen System haben.

Das SIO-Register WR2 wird zum Speichern des Interrupt-Vektors benutzt. Dieser Vektor wird als Antwort auf eine Interrupt-Bestätigung des Z80 auf den Datenbus gelegt. Der Z80 muß dazu im Interrupt-Modus 2 arbeiten. Nur der Kanal B des SIO hat ein Interrupt-Vektor-Register. Das schließt jedoch Kanal A nicht vom Interrupt-Betrieb aus.

Das Z80-SIO kann automatisch acht verschiedene Vektoren auf den Datenbus legen. Diese acht Vektoren werden aus dem einen Vektor gewonnen, der während der Initialisierungsphase in das SIO geschrieben wird. Damit diese acht Vektoren gebildet werden können, muß Bit D2 des Registers WR1 logisch 1 gesetzt werden (Status beeinflußt Vektor).

Wenn dieses Bit gesetzt ist und Interrupts freigegeben sind, dann modifiziert der Typ des Interrupts einige Bits des Interrupt-Vektors. Die modifizierten Bits sind D3, D2 und D1. Damit ergeben sich acht verschiedene Möglichkeiten.

Die acht verschiedenen Vektoren sind in zwei Gruppen von jeweils vier aufgeteilt. Jede Gruppe steht für einen Kanal. Dadurch ist es möglich, Interrupts genauso vom Kanal A wie vom Kanal B zu bekommen. Eine mögliche Frage ist nur, was passiert, wenn beide Kanäle zur selben Zeit Interrupt-Anfragen machen wollen. Zur Lösung dieses Problems sind die Interrupts intern im SIO priorisiert. Die folgende Liste der möglichen Interrupts ist nach der Priorität sortiert; der erste Vektor hat die höchste Priorität.

D3	D2	D1	Interrupt-Typ	Kanal
1	1	1	spezieller Empfängerzustand	A
1	1	0	Zeichen empfangen	A
1	0	0	Sendepuffer leer	A
1	0	1	externe Statusänderung	A
1	1	1	spezieller Empfängerzustand	B
1	1	0	Zeichen empfangen	B
1	0	0	Sendepuffer leer	B
1	0	1	externe Statusänderung	B

Als Beispiel wollen wir annehmen, wir hätten als Interrupt-Vektor den Wert 10010000 programmiert. Weiter gehen wir davon aus, Interrupts wären freigegeben und ein Interrupt tritt auf. Kanal A empfängt ein Zeichen. Der bei der Interrupt-Bestätigung auf den Datenbus gelegte Vektor wäre dann 10011100. Beachten Sie, daß die Bits D3, D2 und D1 gegenüber dem programmierten Vektor modifiziert wurden, um die Art des Interrupts zu kennzeichnen.

Lassen Sie uns nun genauer sehen, was die einzelnen Interrupt-Bedingungen bedeuten. Wir beginnen mit dem speziellen Empfangszustand. Er schließt die Paritätsfehler ein, den Empfänger-Überlauf, die Rahmenfehler und die „End-of-frame"-Nachricht beim synchronen SDLC-Protokoll. Im vorigen Abschnitt haben wir gesehen, daß wir diese Fehler durch Lesen des Registers RR1 feststellen können.

Der nächste Interrupt-Vektor tritt auf, wenn ein empfangenes Zeichen verfügbar ist. Der Interrupt wird angefordert, sobald mindestes ein Zeichen im FIFO steht. Anzumerken ist, daß gleichzeitig auch der spezielle Empfängerzustand auftreten kann, wenn ein Fehler auftritt. Damit gibt es zwei mögliche Interrupt-Vektoren. Der spezielle Empfängerzustand hat jedoch die höhere Priorität.

Der nächste Vektor wird erzeugt, wenn der Sendepuffer bereit ist, das nächste Zeichen aufzunehmen.

Die letzte Interrupt-Bedingung wird durch eine Änderung des Zustands einer der Statusleitungen CTS, DCD oder SYNC ausgelöst.

Initialisierung des SIO für Interrupts

Die Initialisierungs-Sequenz für den Interrupt-Betrieb ist der Sequenz ohne Benutzung von Interrupts sehr ähnlich. Zwei Register, die im Abfragemodus ohne Bedeutung waren, müssen jedoch zusätzlich programmiert werden. Das sind WR2 (Interrupt-Vektor) und WR1 (Interrupt-Modus). Im Abfragemodus mußte WR1 nur zum Abschalten der Interrupts beschrieben werden.

```
                ;-------------------------------------------------------
                ;
                ; Dieses Programm initialisiert die SIO für ein Interrupt-
                ; getriebenes System. Der Interrupt-Vektor soll 98H sein.
                ; Kanal B ist der aktive Kanal. Die SIO liegt im I/O-Bereich
                ; 8C-8F.
                ;
                ; Die wichtigsten SIO-Parameter sind *64 Takt, 2 Stopp-Bits,
                ; 8 Rx-Bits, 8 Tx-Bits, keine Parität.
                ; Int. von Empf. und Sender, Status beinfl. Vektor.
                ;
                ;-------------------------------------------------------

008D            BDATA   EQU     8DH     ; SIO Kanal B Daten
008F            BCON    EQU     8FH     ; SIO Kanal B Steuerung

1800    3E 18           LD      A,18H
1802    D3 8F           OUT     (BCON),A    ; WR0, Kanal Reset
1804    3E 02           LD      A,02H
1806    D3 8F           OUT     (BCON),A    ; WR0, Zeiger zu WR2
1808    3E 98           LD      A,98H
180A    D3 8F           OUT     (BCON),A    ; WR2, Int. Vektor = 98H
180C    3E 04           LD      A,04H
180E    D3 8F           OUT     (BCON),A    ; WR0, Zeiger zu WR4
1810    3E CC           LD      A,0CCH
1812    D3 8F           OUT     (BCON),A    ; WR4, *64, 2 Stopp-Bits, kein Par.
1814    3E 03           LD      A,03H
1816    D3 8F           OUT     (BCON),A    ; WR0, Zeiger zu WR3
1818    3E C1           LD      A,0C1H
181A    D3 8F           OUT     (BCON),A    ; WR3, 8 Rx-Bits, Rx ein
181C    3E 05           LD      A,05H
181E    D3 8F           OUT     (BCON),A    ; WR0, Zeiger zu WR5
1820    3E 68           LD      A,68H
1822    D3 8F           OUT     (BCON),A    ; WR5, 8 Tx-Bits, Tx ein
1824    3E 01           LD      A,01H
1826    D3 8F           OUT     (BCON),A    ; WR0, Zeiger zu WR1
1828    3E 16           LD      A,16H

182A    D3 8F           OUT     (BCON),A    ; WR1, Int. Rx, Tx
182C    C9              RET

                ; Ende der Initialisierung

                        END
```

Abb. 11.12: *Z80-Programm zur Initialisierung des Z80-SIO in einer Interrupt-Anwendung.*

Um zu sehen, wie das SIO für den Interrupt-Betrieb aufgesetzt wird, wollen wir ein Beispiel untersuchen. Die Übertragungsbedingungen sind die gleichen wie beim Abfragemodus. Der kleine Unterschied liegt in der Freigabe der Interrupts. Wir benutzen Kanal B. Das Z80-Programm zur Initialisierung ist in Abb. 11.12 zu sehen.

Nach der Initialisierung

Nachdem das SIO initialisiert ist, kann es in einer interruptgetriebenen Umgebung arbeiten. Natürlich müssen spezielle Routinen geschrieben

```
                ;-------------------------------------------------
                ; SIO Kanal B Interrupt-Service-Routinen
                ;-------------------------------------------------

                ; Port-Vereinbarungen:

008D                BDATA   EQU     8DH     ; SIO Kanal B Daten
008F                BCON    EQU     8FH     ; SIO Kanal B Steuerung
5900                RXCH    EQU     5900H   ; Speicherplatz für Eingabe
5901                TXCH    EQU     RXCH+1  ; Speicherplatz für Ausgabe


                ; Diese Service-Routine wird beim Zeichen-Empfang benutzt.

1800    F5          RXINT:  PUSH    AF              ; rette Register
1801    DB 8D               IN      A,(BDATA)       ; hole Zeichen
1803    32 5900             LD      (RXCH),A        ; speichere Zeichen
1806    F1                  POP     AF              ; Register zurück
1807    FB                  EI                      ; CPU-Interrupts ein
1808    ED 4D               RETI                    ; zurück


                ; Diese Routine wird vom Sendepuffer-leer-Interrupt
                ; aufgerufen, wenn keine weiteren Zeichen gesendet
                ; werden sollen.

180A    F5          TXINT:  PUSH    AF              ; rette Register
180B    3E 28               LD      A,28H
180D    D3 8F               OUT     (BCON),A        ; WR0, Tx Int. aus
180F    F1                  POP     AF
1810    FB                  EI
1811    ED 4D               RETI


                ; Diese Routine wird vom Sendepuffer-leer-Interrupt
                ; aufgerufen, wenn ein weiteres Zeichen gesendet
                ; werden soll.

1813    F5          TXINT1: PUSH    AF
1814    3A 5901             LD      A,(TXCH)        ; hole Zeichen
1817    D3 8D               OUT     (BDATA),A       ; zur SIO
1819    F1                  POP     AF
181A    FB                  EI
181B    ED 4D               RETI

                            END
```

Abb. 11.13: *Beispiele für Interrupt-Service-Routinen zum Senden und Empfangen von Zeichen mit dem Z80-SIO.*

```
                ;-------------------------------------------------------
                ;
                ; Interrupt-Service-Routine zur Behandlung spezieller
                ; Interrupt-Bedingungen
                ;
                ;-------------------------------------------------------

008D                BDATA   EQU     8DH     ; SIO Kanal B Daten
008F                BCON    EQU     8FH     ; SIO Kanal B Steuerung

1B00    DB 8D       SPCINT: IN      A,(BDATA)       ; lese empf. Zeichen

                ; Das Eingabezeichen muß gelesen werden, um den
                ; Empfangspuffer rückzusetzen. Das Zeichen wird
                ; bei der Fehlerbehandlung nicht gebraucht.

1B02    3E 01               LD      A,01H
1B04    D3 8F               OUT     (BCON),A        ; WR0, Zeiger auf WR1

                ; Wir lesen RR1, um den Fehler-Typ festzustellen.

1B06    DB 8F               IN      A,(BCON)

                ; Bit D6, D5 und D4 beschreiben den Fehler-Typ.

1B08    3E 30               LD      A,30H
1B0A    D3 8F               OUT     (BCON),A        ; WR0, "Fehler rücksetzen"

                ; Der vorstehende Befehl wird benötigt, um den Fehlerstatus
                ; rückzusetzen.

                ; Wir schalten jetzt die Interrupts wieder ein.

1B0C    FB                  EI

1B0D    ED 4D               RETI

                ; Die RETI-Anweisung wird zum Rücksetzen der Interrupt-
                ; Bedingung in der SIO-Hardware benötigt.

                            END
```

Abb. 11.14: *Interrupt-Service-Routine zur Behandlung spezieller Empfängerzustände.*

werden, um jeden Interrupt-Typ, den das SIO generiert, zu handhaben. Für den Empfangspuffer-voll-Interrupt ist es nur nötig, das Zeichen vom Kanal-Datenregister zu lesen – das leert den Puffer und schaltet damit die Interrupt-Quelle aus. Die Ausführung der RETI-Anweisung setzt die Interrupt-Anforderung dann auch hardware-mäßig zurück.

Tritt ein Interrupt durch die Senderegister-leer-Bedingung auf, wird von der Service-Routine entweder ein neues Zeichen in den Ausgabepuffer geschrieben oder durch ein spezielles Kommando der anstehende Tx-Interrupt zurückgesetzt. Abb. 11.13 zeigt als Beispiel die Interrupt-Routinen zum Senden und Empfangen der Zeichen. Sie sind normalerweise Teile größerer Service-Routinen. Sie zeigen jedoch verschiedene interessante Punkte über die Verarbeitung von SIO-Interrupts.

Zum Schluß sei noch auf Abb. 11.14 hingewiesen, wo die Interrupt-Routine zur Behandlung spezieller Empfängerzustände gezeigt wird.

Zusammenfassung

In diesem Kapitel haben wir die Arbeitsweise und Programmierung des Z80-SIO kennengelernt. Wir haben das Blockschaltbild des Bausteins untersucht und beschrieben, wie die internen Register programmiert werden. Die Informationen aus diesem Kapitel und aus Anhang A sollten für Sie ausreichen, das SIO in Ihrem eigenen System für beliebige Anwendungen der seriellen Kommunikation einzusetzen.

Kapitel 12

Statische Testmethode für Z80-Systeme

Einführung

In diesem lezten Kapitel wollen wir eine Hardware-Test- und Reparaturmethode vorstellen, die im englischen als Static Stimulus Testing (SST) bezeichnet wird. Wir nennen sie „Statische Einspeisungs-Testmethode", verwenden aber weiter die englische Abkürzung SST. Ursprünglich für die Industrie entwickelt, eröffnet diese Methode einen einfachen und preiswerten Weg, ohne Software die Hardware eines Systems zu prüfen. Alle in diesem Buch beschriebenen Bausteine und die angeschlossene Peripherie können Sie hiermit überprüfen.

Die SST wurde ursprünglich zum Testen und Reparieren von Mikroprozessor-Systemen folgender zwei Kategorien entwickelt:

- Systeme, die einen katastrophalen Fehler aufweisen und keine Software mehr verarbeiten können (also Systeme, die schon einmal liefen).
- Systeme im Entwicklungsstadium, die noch nie arbeiteten. Dies sind Systeme, die noch nicht „debugged", als fehlerfrei gemacht wurden, oder Systeme, für die noch keine Software vorhanden ist. Trotzdem möchte der Hardware-Entwickler vielleicht schon ohne die System-Software sämtliche Komponenten testen.

In beiden Kategorien sind die klassischen Hardware-Testmethoden, wie Logik-Zustands-Analyse und Signatur-Analyse, von geringem Nutzen. Das liegt daran, daß diese beiden Methoden nur sinnvoll sind, wenn zumindest etwas Software bereits arbeitet. Obwohl neue Techniken zur Signatur-Analyse entwickelt wurden, gibt es noch eine große Lücke im Bereich der Testtechnik von Mikroprozessor-Systemen.

Eine wichtige Frage ist: „Wie und wo fängt man an, ein System zu debuggen, das total laufunfähig ist? Was kann man tun, wenn das System keinerlei Software verarbeiten kann?"

Die SST kann tatsächlich diese Lücke schließen. Es ist mit ihr möglich, ein nicht-lauffähiges System bis zu dem Punkt zu bringen, wo Diagnostik-Software laufen kann. Dies kann in einer effizienten und übersichtlichen Weise mit Hilfe einfacher und preiswerter Instrumente erfolgen.

Eine wichtige Eigenschaft der SST ist die totale Unabhängigkeit von irgendeiner System-Software. Das bedeutet, daß auch ein Techniker oder Ingenieur mit wenig Software-Erfahrung die SST erfolgreich anwenden kann. Andererseits kann eine software-erfahrene Person mit wenig Hardware-Erfahrung die Grundlagen der SST leicht lernen.

In diesem Text wollen wir Beispiele für den Gebrauch der SST-Technik zum Testen eines vollständigen Systems vorstellen. Wir gehen dabei davon aus, daß keine Software existiert. Wir nehmen also an, daß wir gerade einen Prototyp entwickelt haben und seine Funktion überprüfen wollen, oder daß ein System fehlerhaft arbeitet und wir kein Software-Listing haben (oder daß die System-Software-Kenntnis minimal ist).

Wenn Sie mit der Benutzung der SST beginnen, werden Sie bald ein Gefühl dafür bekommen, was in der Hardware passiert. Tatsächlich wird Ihnen der Gebrauch dieses Werkzeugs ein besseres Verständnis der elektrischen Kommunikation des Z80 in einer System-Umgebung vermitteln.

Übersicht über die SST

Das Grundkonzept der SST geht davon aus, daß die elektrische Kommunikation in einem Mikroprozessor-System an sich statischer Natur ist. Es gibt zwei Spannungspegel, die logisch 1 und logisch 0 repräsentieren. Das Mikroprozessor-System springt zwischen diesen beiden statischen Basiszuständen hin und her. Die elektrischen Ereignisse laufen sinnvollerweise in hoher Geschwindigkeit ab, doch es besteht kein Zwang dazu. Tatsache ist, daß es eine obere Geschwindigkeitsgrenze gibt. Eine untere Grenze gibt es zum Glück im allgemeinen aber nicht.

In einem Mikroprozessor-System ist nicht nur die Kommunikation zwischen CPU und Speicher oder I/O statisch, sondern alle Signalleitungen haben zu einem bestimmten Zeitpunkt einen bestimmten elektrischen Zustand. Während einer Speicheroperation zeigen die Adreßleitungen zum Beispiel logisch auf einen bestimmten Speicherplatz. Das geschieht unabhängig vom Zustand der anderen Systemsignale. Mit der SST kann jede Signalleitung des Systems als unabhängiges logisches Signal aufgefaßt werden.

Jedes Signal hat in jedem Falle einen Ausgangspunkt und einen Zielpunkt. Mit der SST wird der Ausgangspunkt ersetzt und die Signalleitung

auf den gewünschten logischen Pegel gebracht. Es ist dann möglich, die Wirkung des Signals statisch zu verfolgen. (Anmerkung: Das Element der zeitabhängigen Signale wird mit der SST ausgeschaltet.)

Mit digitalen Standard-Hilfsmitteln und der SST können Sie die Hardware eines ganzen Mikroprozessor-Systems durchtesten. Unabhängig von der Komplexität der System-Hardware können Problembereiche viel leichter gefunden werden, wenn die dynamische Situation in eine statische Situation überführt wird.

Eine zusätzliche Eigenschaft der SST ist die, daß sie auch an Systemen mit speziellen LSI-Bausteinen, wie PIOs und SIOs, eingesetzt werden kann. Das beruht darauf, daß sie unter der Prämisse arbeitet, daß die gesamte Mikroprozessor-Kommunikation innerhalb des Systems statisch abläuft.

Sie können natürlich mit dieser Prämisse brechen, wenn Sie sagen, daß dynamische RAMs keine statischen Bausteine sind. Und weil das tatsächlich wahr ist, müssen Einschränkungen getroffen werden. Der einzige Teil der dynamischen RAMs, der wirklich dynamisch ist, sind die Speicherzellen. Alle Adreßeingänge sowie $\overline{RAS}$, $\overline{CAS}$ und MUX können statisch arbeiten. Es ist leider wahr, daß die Speicherzellen selber nicht statisch arbeiten können, doch können Sie mit statischen Mitteln sämtliche Peripheriebauteile wie auch die Hardware des Systemspeichers debuggen.

Um den Einsatz der SST zu illustrieren, wollen wir ein kleines Beispiel besprechen. In diesem Beispiel wollen wir die Adreßeingänge des System-ROM prüfen. (Anmerkung: In diesem Text können wir keine detaillierte Bedienungsanleitung für die SST geben. Es soll Ihnen lediglich das Konzept zum Einsatz der SST bei einem realen Problem gezeigt werden.)

Abb. 12.1 zeigt das Blockschaltbild eines Mikroprozessor-Systems mit ROM. Es zeigt auch die Punkte für die Quelle und das Ziel der Adreßleitungen. Wir wollen die ordentliche Hardware-Funktion dieser Adreßleitungen mit der SST überprüfen. Wir machen dabei folgende Schritte:

1. Zuerst entfernen wir den Mikroprozessor aus dem System und installieren ein Kabel zur SST-Schalterkonsole wie in Abb. 12.2.

2. Wir nehmen an, wir haben einen Schalter für jede der Adreßleitungen A0–A15. Jeder Schalter kann eine Leitung auf logisch 1 oder logisch 0 setzen. Die 16 Schalter können vom Bediener in eine Position der 64K möglichen Kombinationen gebracht werden.

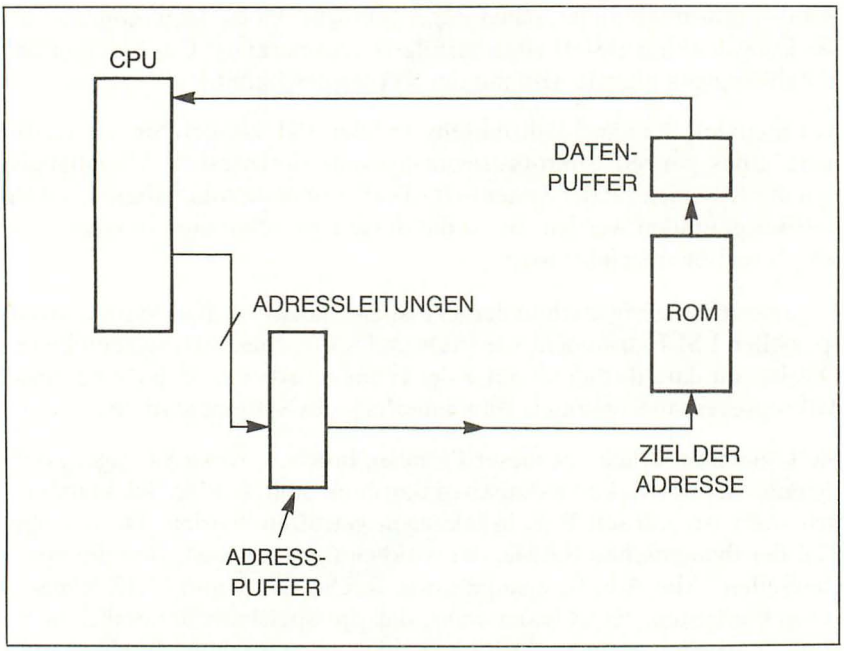


Abb. 12.1: *Blockschaltbild eines Systems mit physikalischer Verbindung von CPU und ROM.*

3. Die gesetzte Schalterkombination bleibt angelegt (statisch). Wir können nun die Eingänge zu den Adreßpuffern mit einfachen Gleichstrom-Meßmethoden ansehen. Wir können sehen, ob alle Punkte entlang dieser Leitungen denselben (richtigen) logischen Zustand haben wie die entsprechenden Schalter. Das gleiche wird mit den Ausgängen der Adreß-Puffer gemacht.
4. Jetzt messen wir die logischen Spannungspegel an den Eingangsleitungen des ROM. Hierfür können wir einen Oszillographen, einen Logik-Prüfstab, ein Digital-Voltmeter oder irgendein anderes Gleichstrom-Meßgerät benutzen.
5. Die gesamte Adreßdecodierung kann in gleicher Weise statisch überprüft werden.

In diesem allgemeinen Beispiel haben wir nur die Adreßleitungen überprüft. Die SST macht es möglich, diese Leitungen separat zu prüfen, da sie unabhängig von den anderen Leitungen arbeitet. Beachten Sie ferner, daß wir nichts über absolute Zeiten gesagt haben. Das liegt daran, daß die

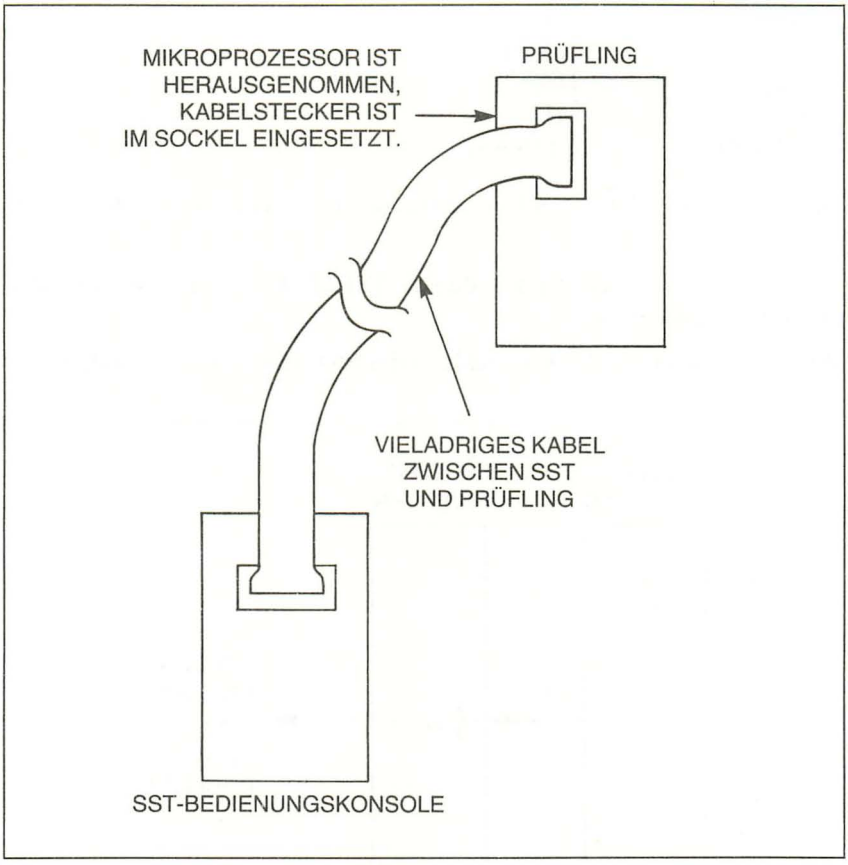


Abb. 12.2: *Verbindung der SST-Schalterkonsole mit dem zu testenden Mikroprozessor-System. Beachten Sie, daß der Mikroprozessor entfernt wurde und das SST-Kabel im jetzt freien Sockel steckt.*

SST den Signalablauf auseinander zieht und wir somit, solange wir wollen, ein Signal vom Ursprung bis zum Ziel verfolgen können.

Hardware der SST

Die SST benötigt nur sehr einfache Hardware. Sie ist ebenso gut geeignet für Ausbildungszwecke wie zum industriellen Gebrauch. Die Idee hinter diesem Instrument ist, daß der Bediener eine statische Kontrolle über alle wichtigen vom Mikroprozessor ausgehenden Signale hat. Beim Z80 sind das folgende Leitungen:

A0–A15	$\overline{\text{IORQ}}$
D0–D7	$\overline{\text{RFSH}}$
$\overline{\text{RD}}$	$\overline{\text{HALT}}$
$\overline{\text{WR}}$	$\overline{\text{M1}}$
$\overline{\text{MREQ}}$	$\overline{\text{BUSAK}}$

Die restlichen Z80-Bauteil-Pins sind Eingänge, die wir hier nicht berücksichtigen.

Als letztes sei noch angemerkt, daß die SST alle Rückkopplungsschleifen im System unterbricht.

Wir wollen uns jetzt der Entwicklung der SST-Hardware zuwenden.

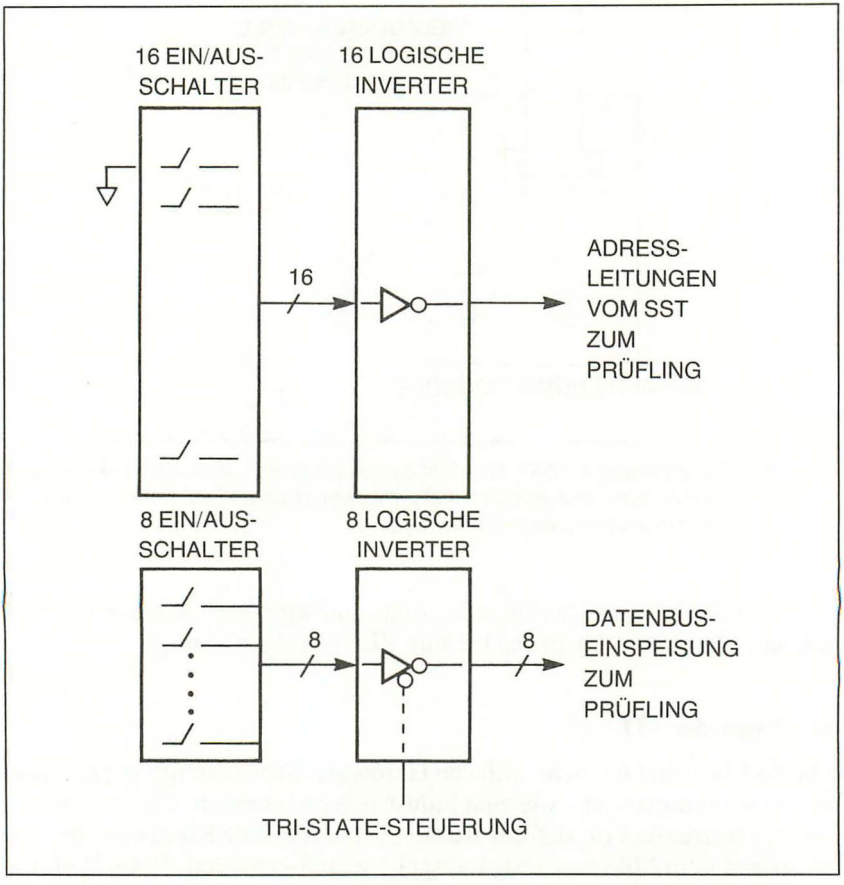


Abb. 12.3: *Blockschaltbild der Adreß- und Dateneinspeisung mittels DIP-Schaltern.*

Adreß- und Datenausgangsleitungen bei der SST

Abb. 12.3 zeigt das Blockschaltbild der Hardware für die Adressen- und Dateneinspeisung der SST. Der logische Zustand der Ausgangsleitungen wird durch die Position der DIP-Schalter bestimmt. Der Ausgang jedes Schalters wird logisch invertiert und gepuffert. Die Puffer für die Dateneinspeisung sind tri-state-fähig. Dadurch läßt sich der Datenbus in beide Richtungen betreiben.

Die mit D0–D7 bezeichneten Ausgänge der Puffer werden zur Generierung der Systemdatenleitungen benutzt. Wird die SST zur Fehlersuche eingesetzt, bestimmen die DIP-Schalter genauso den Zustand des Datenbusses, wie es sonst der Z80 tut. Abb. 12.4 und 12.5 zeigen die Schaltpläne zur Realisation von Adreß- und Dateneinspeisung mit SST.

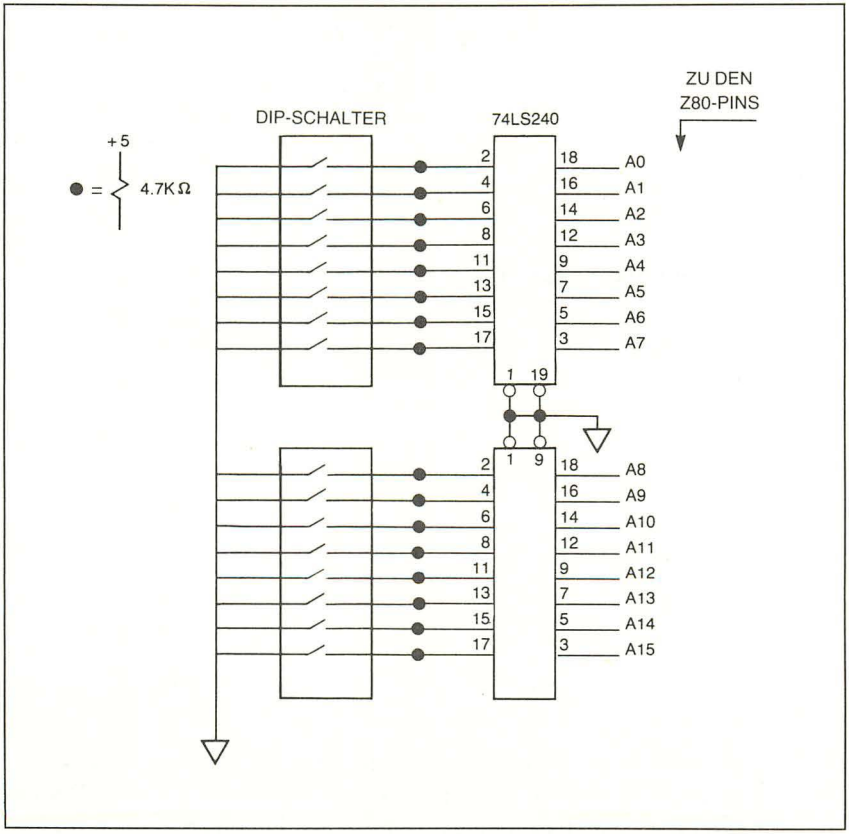


Abb. 12.4: *Schaltung der Adreßeinspeisung.*

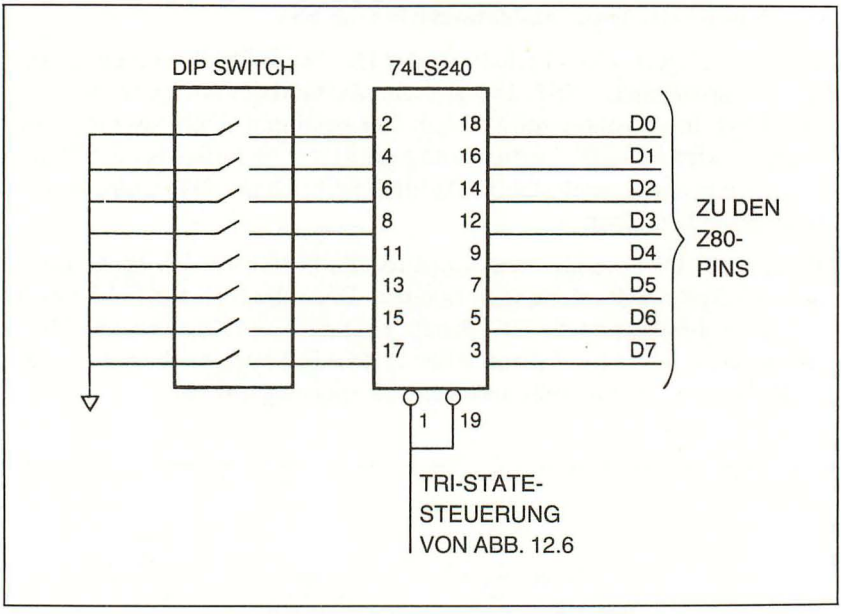


Abb. 12.5: *Schaltung der Dateneinspeisung.*

Einspeisung der Steuersignale

Wir wollen nun sehen, wie die folgenden Steuersignale bei der SST erzeugt werden:

$\overline{M1}$
$\overline{MREQ}$
$\overline{IORQ}$
$\overline{RD}$
$\overline{WR}$
$\overline{RFSH}$
$\overline{HALT}$
$\overline{BUSAK}$

Wir behandeln all diese Signale im gleichen Abschnitt, da die Hardware für alle identisch ist.

Die aufgelisteten Signale stellen alle vom Z80 generierte Steuerbits dar. Die meisten davon sind für den Start eines Speicherzyklus ohne Bedeutung und bleiben es auch für den ganzen Zyklus. Um die Signale zu erzeugen, werden einpolige Umschalter verwendet. Die Schalter liegen an den

Eingängen von Entprell-Schaltungen, deren Ausgänge dem zu testenden System zugeführt werden. Abb. 12.6 zeigt die Schaltung zur Erzeugung der Signale.

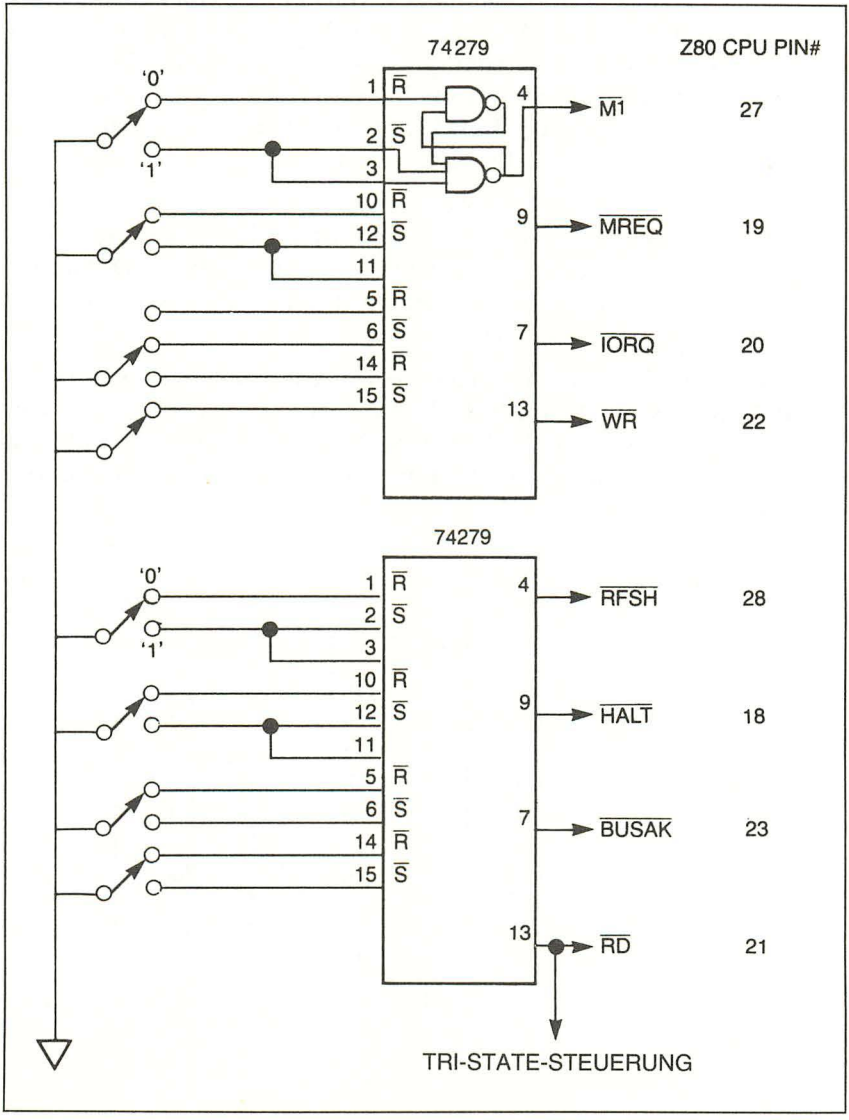


Abb. 12.6: *Schaltung zur Entprellung der wichtigsten Steuersignale. Das 74LS279 besteht aus vier R-S-Flipflops, die zur Entprellung benutzt werden. Als Schalter dienen einpolige Umschalter.*

LED-Anzeige für den Datenbus

Wir haben jetzt alle Schaltungen für die SST besprochen, mit Ausnahme der Schaltung zum Ansehen des logischen Zustands des Datenbusses. Hierzu benutzen wir acht Leuchtdioden, von der jede den Zustand einer Datenbusleitung anzeigt. Abb. 12.7 zeigt die Realisierung solch einer Anzeige. Der Datenbus des Mikroprozessors (oder der SST-Hardware) wird an die Eingänge eines Inverters 74LS240 geschlossen. Die Inverter-Ausgänge treiben die LEDs. Eine logische 0 auf den Eingängen schaltet die LEDs aus, eine logische 1 schaltet sie ein.

Die LEDs zeigen den logischen Zustand der Mikroprozessor-Anschlüsse D0-D7. Das ermöglicht es, den aktuellen Zustand der vom Mikroprozessor gelesenen oder geschriebenen Daten zu sehen. Mit der LED-Anzeige ist schnell festzustellen, ob zwei Leitungen kurzgeschlossen sind oder Daten den Mikroprozessor nicht erreichen.

Bei einer Schreiboperation können Sie leicht feststellen, welche Daten der Mikroprozessor oder die SST-Hardware zum System ausgibt. Beachten Sie, daß die Anzeige die Daten direkt an den Bausteinanschlüssen wiedergibt. Das ist für die Überprüfung der System-Datenbus-Leitungen sehr nützlich.

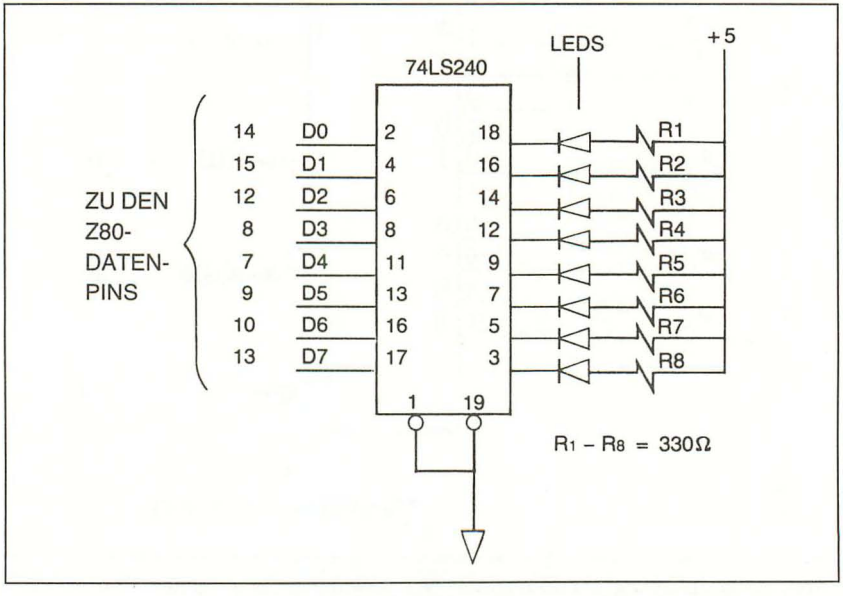


Abb. 12.7: *Schaltplan einer LED-Anzeige für den logischen Zustand des System-Datenbusses in einem Z80-Mikroprozessor-System.*

Zusammenfassung

In diesem Kapitel haben wir die erforderliche Hardware für ein SST-System für den Z80 kennengelernt. Diese Technik stellt ein leistungsfähiges Werkzeug zur Prüfung eines Mikroprozessor-Systems dar. Es kann sowohl zum Debuggen von Prototypen wie auch zum Debuggen fehlerhafter Geräte eingesetzt werden. Die SST kann von jedem benutzt werden. Es erfordert nicht einen erfahrenen Mikroprozessor-System-Entwickler, Software-Spezialisten oder Hardware-Spezialisten, um vorzügliche Resultate zu erzielen.

Als letzte Anmerkung sei noch gesagt, daß Sie, falls Sie noch keinen statischen Einspeisungs-Tester besitzen, aber einen benötigen, zwei Möglichkeiten haben. Sie können entweder mit Hilfe der angegebenen Schaltbilder selber einen konstruieren oder ihn von einem industriellen Hersteller beziehen.*

* Zwei Z80-SST-Versionen sind von folgender Firma lieferbar: Creative Microprocessor Systems, Inc., P.O.Box 1538, Los Gatos, CA.95030.

Interne Register-Beschreibungen des Z80-SIO

WRITE REGISTERS

The Z80-SIO contains eight registers (WR0-WR7) in each channel that are programmed separately by the system program to configure the functional personality of the channels. With the exception of WR0, programming the write registers requires two bytes. The first byte contains three bits (D_0-D_2) that point to the selected register; the second byte is the actual control word that is written into the register to configure the Z80-SIO.

Note that the programmer has complete freedom, after pointing to the selected register, of either reading to test the read register or writing to initialize the write register. By designing software to initialize the Z80-SIO in a modular and structured fashion, the programmer can use powerful block I/O instructions.

WR0 is a special case in that all the basic commands (CMD_0-CMD_2) can be accessed with a single byte. Reset (internal or external) initializes the pointer bits (D_0-D_2) to point to WR0.

The basic commands (CMD_0-CMD_2) and the CRC controls (CRC_0, CRC_1) are contained in the first byte of any write register access. This maintains maximum flexibility and system control. Each channel contains the following control registers. These registers are addressed as commands (not data).

WRITE REGISTER 0

WR0 is the command register; however, it is also used for CRC reset codes and to point to the other registers.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CRC Reset Code 1	CRC Reset Code 0	CMD 2	CMD 1	CMD 0	PTR 2	PTR 1	PTR 0

Pointer Bits (D_0-D_2). Bits D_0-D_2 are pointer bits that determine which other write register the next byte is to be written into or which read register the next byte is to be read from. The first byte written into each channel after a reset (either by a Reset command or by the external reset input) goes into WR0. Following a read or write to any register (except WR0), the pointer will point to WR0.

Command Bits (D_3-D_5). Three bits, D_3-D_5, are encoded to issue the seven basic Z80-SIO commands.

COMMAND	CMD_2	CMD_1	CMD_0	
0	0	0	0	Null Command (no effect)
1	0	0	1	Send Abort (SDLC Mode)
2	0	1	0	Reset External/Status Interrupts
3	0	1	1	Channel Reset
4	1	0	0	Enable Interrupt on next Rx Character
5	1	0	1	Reset Transmitter Interrupt Pending
6	1	1	0	Error Reset (latches)
7	1	1	1	Return from Interrupt (Channel A)

Command 0 (Null). The Null command has no effect. Its normal use is to cause the Z80-SIO to do nothing while the pointers are set for the following byte.

Command 1 (Send Abort). This command is used only with the SDLC mode to generate a sequence of eight to thirteen 1's.

Command 2 (Reset External/Status Interrupts). After an External/Status interrupt (a change on a modem line or a break condition, for example), the status bits of RR0 are latched. This command re-enables them and allows interrupts to occur again. Latching the status bits captures short pulses until the CPU has time to read the change.

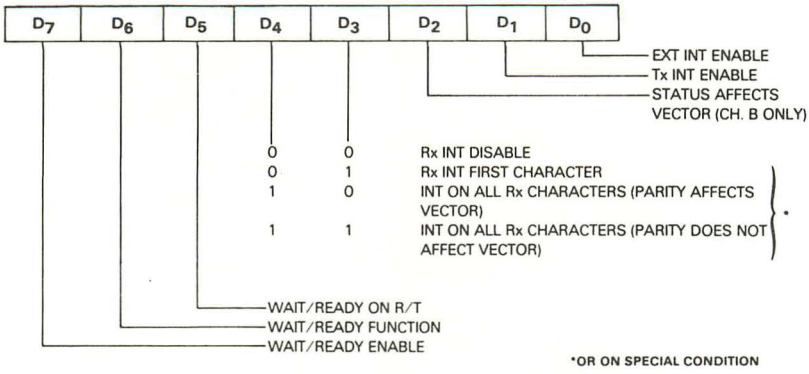
Command 3 (Channel Reset). This command performs the same function as an External Reset, but only on a single channel. Channel A Reset also resets the interrupt prioritization logic. All control registers for the channel must be rewritten after a Channel Reset command.

WRITE REGISTER BIT FUNCTIONS

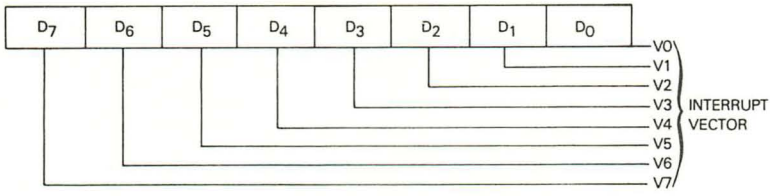
WRITE REGISTER 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
					0	0	0	REGISTER 0
					0	0	1	REGISTER 1
					0	1	0	REGISTER 2
					0	1	1	REGISTER 3
					1	0	0	REGISTER 4
					1	0	1	REGISTER 5
					1	1	0	REGISTER 6
					1	1	1	REGISTER 7
		0	0	0	NULL CODE			
		0	0	1	SEND ABORT (SDLC)			
		0	1	0	RESET EXT/STATUS INTERRUPTS			
		0	1	1	CHANNEL RESET			
		1	0	0	ENABLE INT ON NEXT Rx CHARACTER			
		1	0	1	RESET Tx INT PENDING			
		1	1	0	ERROR RESET			
		1	1	1	RETURN FROM INT (CH-A ONLY)			
0	0	NULL CODE						
0	1	RESET Rx CRC CHECKER						
1	0	RESET Tx CRC GENERATOR						
1	1	RESET Tx UNDERRUN/EOM LATCH						

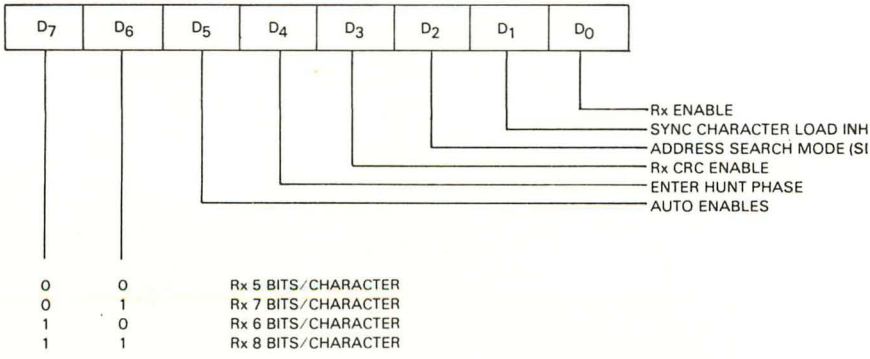
WRITE REGISTER 1



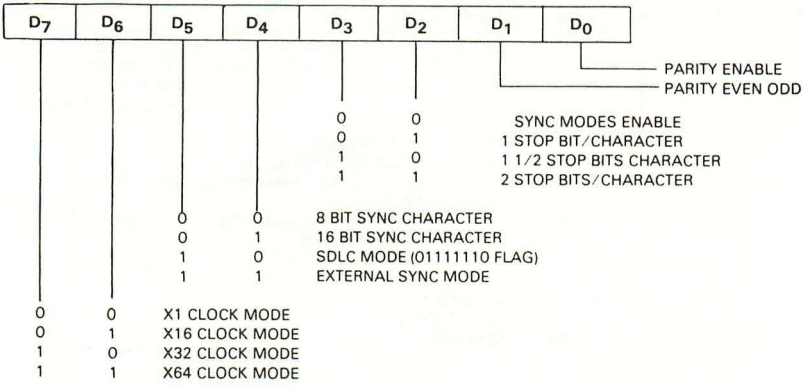
WRITE REGISTER 2 (CHANNEL B ONLY)



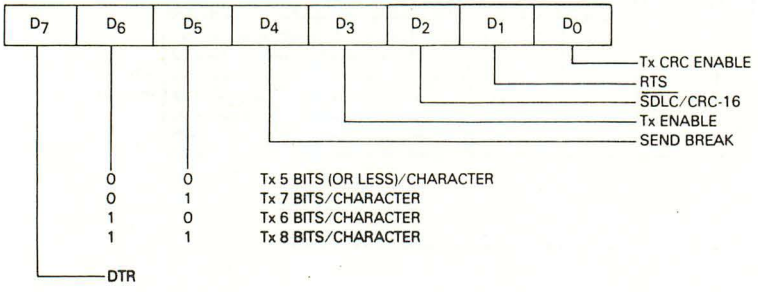
WRITE REGISTER 3



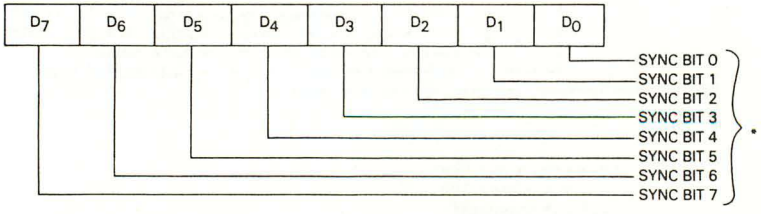
WRITE REGISTER 4



WRITE REGISTER 5

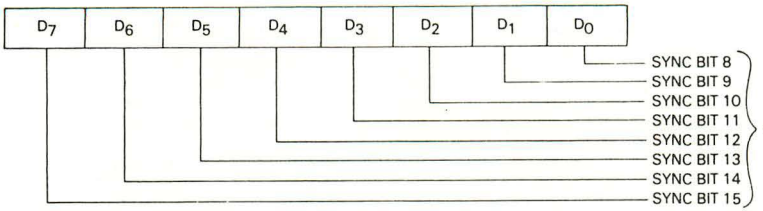


WRITE REGISTER 6



*ALSO SDLC ADDRESS FIELD

WRITE REGISTER 7



*FOR SDLC, IT MUST BE PROGRAMMED TO "01111110" FOR FLAG RECOGNITION

After a Channel Reset, four extra system clock cycles should be allowed for Z80-SIO reset time before any additional commands or controls are written into that channel. This can normally be the time used by the CPU to fetch the next op code.

Command 4 (Enable Interrupt On Next Character). If the Interrupt On First Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the Z80-SIO for the next message.

Command 5 (Reset Transmitter Interrupt Pending). The transmitter interrupts when the transmit buffer becomes empty if the Transmit Interrupt Enable mode is selected. In those cases where there are no more characters to be sent (at the end of message, for example), issuing this command prevents further transmitter interrupts until after the next character has been loaded into the transmit buffer or until CRC has been completely sent.

Command 6 (Error Reset). This command resets the error latches. Parity and Overrun errors are latched in RR1 until they are reset with this command. With this scheme, parity errors occurring in block transfers can be examined at the end of the block.

Command 7 (Return From Interrupt). This command must be issued in Channel A and is interpreted by the Z80-SIO in exactly the same way it would interpret a RETI command on the data bus. It resets the interrupt-under-service latch of the highest-priority internal device under service and thus allows lower priority devices to interrupt via the daisy chain. This command allows use of the internal daisy chain even in systems with no external daisy chain or RETI command.

CRC Reset Codes 0 and 1 (D_6 and D_7). Together, these bits select one of the three following reset commands:

CRC Reset Code 1	CRC Reset Code 0	
0	0	Null Code (no effect)
0	1	Reset Receive CRC Checker
1	0	Reset Transmit CRC Generator
1	1	Reset Tx Underrun/End Of Message Latch

The Reset Transmit CRC Generator command normally initializes the CRC generator to all 0's. If the SDLC mode is selected, this command initializes the CRC generator to all 1's. The Receive CRC checker is also initialized to all 1's for the SDLC mode.

WRITE REGISTER 1

WR1 contains the control bits for the various interrupt and Wait/Ready modes.

D_7 Wait/Ready Enable	D_6 Wait Or Ready Function	D_5 Wait/Ready On Receive/Transmit	D_4 Receive Interrupt Mode 1
D_3 Receive Interrupt Mode 0	D_2 Status Affects Vector	D_1 Transmit Interrupt Enable	D_0 External Interrupts Enable

External/Status Interrupt Enable (D_0). The External/Status Interrupt Enable allows interrupts to occur as a result of transitions on the DCD, CTS or SYNC inputs, as a result of a Break/Abort detection and termination, or at the beginning of CRC or sync character transmission when the Transmit Underrun/EOM latch becomes set.

Transmitter Interrupt Enable (D_1). If enabled, the interrupts occur whenever the transmitter buffer becomes empty.

Status Affects Vector (D_2). This bit is active in Channel B only. If this bit is not set, the fixed vector programmed in WR2 is returned from an interrupt acknowledge sequence. If this bit is set, the vector returned from an interrupt acknowledge is variable according to the following interrupt conditions:

	V_3	V_2	V_1	
Ch B	0	0	0	Ch B Transmit Buffer Empty
	0	0	1	Ch B External/Status Change
	0	1	0	Ch B Receive Character Available
	0	1	1	Ch B Special Receive Condition*
Ch A	1	0	0	Ch A Transmit Buffer Empty
	1	0	1	Ch A External/Status Change
	1	1	0	Ch A Receive Character Available
	1	1	1	Ch A Special Receive Condition*

*Special Receive Conditions: Parity Error, Rx Overrun Error, Framing Error, End Of Frame (SDLC).

Receive Interrupt Modes 0 and 1 (D_3 and D_4). Together, these two bits specify the various character-available conditions. In Receive Interrupt modes 1, 2 and 3, a Special Receive Condition can cause an interrupt and modify the interrupt vector.

D_4 Receive Interrupt Mode 1	D_3 Receive Interrupt Mode 0	
0	0	0. Receive Interrupts Disabled
0	1	1. Receive Interrupt On First Character Only
1	0	2. Interrupt On All Receive Characters—parity error is a Special Receive condition
1	1	3. Interrupt On All Receive Characters—parity error is not a Special Receive condition

Wait/Ready Function Selection (D_5-D_7). The Wait and Ready functions are selected by controlling D_5, D_6 and D_7. Wait/Ready function is enabled by setting Wait/Ready Enable (WR1, D_7) to 1. The Ready Function is selected by setting D_6 (Wait/Ready function) to 1. If this bit is 1, the $\overline{WAIT}/\overline{READY}$ output switches from High to Low when the Z80-SIO is ready to transfer data. The Wait function is selected by setting D_6 to 0. If this bit is 0, the $\overline{WAIT}/\overline{READY}$ output is in the open-drain state and goes Low when active.

Both the Wait and Ready functions can be used in either the Transmit or Receive modes, but not both simultaneously. If D_5 (Wait/Ready or Receive/Transmit) is set to 1, the Wait/Ready function responds to the condition of the receive buffer (empty or full). If D_5 is set to 0, the Wait/Ready function responds to the condition of the transmit buffer (empty or full).

The logic states of the $\overline{WAIT}/\overline{READY}$ output when active or inactive depend on the combination of modes selected. Following is a summary of these combinations:

And D_6 = 1		If D_7 = 0	And D_6 = 0
$\overline{READY}$ is High			$\overline{WAIT}$ is floating
And D_5 = 0		If D_7 = 1	And D_5 = 1
$\overline{READY}$	Is High when transmit buffer is full.	$\overline{READY}$	Is High when receive buffer is empty.
$\overline{WAIT}$	Is Low when transmit buffer is full and an SIO data port is selected.	$\overline{WAIT}$	Is Low when receive buffer is empty and an SIO data port is selected.
$\overline{READY}$	Is Low when transmit buffer is empty.	$\overline{READY}$	Is Low when receive buffer is full.
$\overline{WAIT}$	Is floating when transmit buffer is empty.	$\overline{WAIT}$	Is Floating when receive buffer is full.

The $\overline{WAIT}$ output High-to-Low transition occurs when the delay time $t_DIC(WR)$ after the I/O request. The Low-to-High transition occurs with the delay $t_DH\Phi(WR)$ from the falling edge of Φ. The $\overline{READY}$ output High-to-Low transition occurs with the delay $t_DL\Phi(WR)$ from the rising edge of Φ. The $\overline{READY}$ output Low-to-High transition occurs with the delay $t_DIC(WR)$ after $\overline{IORQ}$ falls.

The Ready function can occur any time the Z80-SIO is not selected. When the $\overline{READY}$ output becomes active (Low), the DMA controller issues $\overline{IORQ}$ and the corresponding B/$\overline{A}$ and C/$\overline{D}$ inputs to the Z80-SIO to transfer data. The $\overline{READY}$ output becomes inactive as soon as $\overline{IORQ}$ and $\overline{CS}$ become active. Since the Ready function can occur internally in the Z80-SIO whether it is addressed or not, the $\overline{READY}$ output becomes inactive when any CPU data or command transfer takes place. This does not cause problems because the DMA controller is not enabled when the CPU transfer takes place.

The Wait function—on the other hand—is active only if the CPU attempts to read Z80-SIO data that has not yet been received, which occurs frequently when block transfer instructions are used. The Wait function can also become active (under program control) if the CPU tries to write data while the transmit buffer is still full. The fact that the $\overline{WAIT}$ output for either channel can become active when the opposite channel is addressed (because the Z80-SIO is addressed) does not affect operation of software loops or block move instructions.

WRITE REGISTER 2

WR2 is the interrupt vector register; it exists in Channel B only. V_4-V_7 and V_0 are always returned exactly as written; V_1-V_3 are returned as written if the Status Affects Vector (WR1, D_2) control bit is 0. If this bit is 1, they are modified as explained in the previous section.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0

WRITE REGISTER 3

WR3 contains receiver logic control bits and parameters.

D_7	D_6	D_5	D_4
Receiver Bits/ Char 1	Receiver Bits/ Char 0	Auto Enables	Enter Hunt Phase
D_3	D_2	D_1	D_0
Receiver CRC Enable	Address Search Mode	Sync Char Load Inhibit	Receiver Enable

Receiver Enable (D_0). A 1 programmed into this bit allows receive operations to begin. This bit should be set only after all other receive parameters are set and receiver is completely initialized.

Sync Character Load Inhibit (D_1). Sync characters preceding the message (leading sync characters) are not loaded into the receive buffers if this option is selected. Because CRC calculations are not stopped by sync character stripping, this feature should be enabled only at the beginning of the message.

Address Search Mode (D_2). If SDLC is selected, setting this mode causes messages with addresses not matching the programmed address in WR6 or the global (11111111) address to be rejected. In other words, no receive interrupts can occur in the Address Search mode unless there is an address match.

Receiver CRC Enable (D_3). If this bit is set, CRC calculation starts (or restarts) at the beginning of the last character transferred from the receive shift register to the buffer stack, regardless of the number of characters in the stack. See "SDLC Receive CRC Checking" (SDLC Receive section) and "CRC Error Checking" (Synchronous Receive section) for details regarding when this bit should be set.

Enter Hunt Phase (D_4). The Z80-SIO automatically enters the Hunt phase after a reset; however, it can be re-entered if character synchronization is lost for any reason (Synchronous mode) or if the contents of an incoming message are not needed (SDLC mode). The Hunt phase is re-entered by writing a 1 into bit D_4. This sets the Sync/Hunt bit (D_4) in RR0.

Auto Enables (D_5). If this mode is selected, $\overline{DCD}$ and $\overline{CTS}$ become the receiver and transmitter enables, respectively. If this bit is not set, $\overline{DCD}$ and $\overline{CTS}$ are simply inputs to their corresponding status bits in RR0.

Receiver Bits/Character 1 and 0 (D_7 and D_6). Together, these bits determine the number of serial receive bits assembled to form a character. Both bits may be changed during the time that a character is being assembled, but they must be changed before the number of bits currently programmed is reached.

D_7	D_6	Bits/Character
0	0	5
0	1	7
1	0	6
1	1	8

WRITE REGISTER 4

WR4 contains the control bits that affect both the receiver and transmitter. In the transmit and receive initialization routine, these bits should be set before issuing WR1, WR3, WR5, WR6, and WR7.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Clock Rate	Clock Rate	Sync Modes	Sync Modes	Stop Bits	Stop Bits	Parity Even/$\overline{\text{Odd}}$	Parity
1	0	1	0	1	0		

Parity (D_0). If this bit is set, an additional bit position (in addition to those specified in the bits/character control) is added to transmitted data and is expected in receive data. In the Receive mode, the parity bit received is transferred to the CPU as part of the character, unless 8 bits/character is selected.

Parity Even $\overline{\text{Odd}}$ (D_1). If parity is specified, this bit determines whether it is sent and checked as even or odd (1 = even).

Stop Bits 0 and 1 (D_2 and D_3). These bits determine the number of stop bits added to each asynchronous character sent. The receiver always checks for one stop bit. A special mode (00) signifies that a synchronous mode is to be selected.

D_3 Stop Bits 1	D_2 Stop Bits 0	
0	0	Sync modes
0	1	1 stop bit per character
1	0	1½ stop bits per character
1	1	2 stop bits per character

Sync Modes 0 and 1 (D_4 and D_5). These bits select the various options for character synchronization.

Sync Mode 1	Sync Mode 0	
0	0	8-bit programmed sync
0	1	16-bit programmed sync
1	0	SDLC mode (01111110 flag pattern)
1	1	External Sync mode

Clock Rate 0 and 1 (D_6 and D_7). These bits specify the multiplier between the clock ($\overline{\text{TxC}}$ and $\overline{\text{RxC}}$) and data rates. For synchronous modes, the x1 clock rate must be specified. Any rate may be specified for asynchronous modes; however, the same rate must be used for both the receiver and transmitter. The system clock in all modes must be at least 5 times the data rate. If the x1 clock rate is selected, bit synchronization must be accomplished externally.

Clock Rate 1	Clock Rate 0	
0	0	Data Rate x1 = Clock Rate
0	1	Data Rate x16 = Clock Rate
1	0	Data Rate x32 = Clock Rate
1	1	Data Rate x64 = Clock Rate

WRITE REGISTER 5

WR5 contains control bits that affect the operation of transmitter, with the exception of D2, which affects the transmitter and receiver.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
DTR	Tx Bits/ Char 1	Tx Bits/ Char 0	Send Break	Tx Enable	CRC-16/ SDLC	RTS	Tx CRC Enable

Transmit CRC Enable (D_0). This bit determines if CRC is calculated on a particular transmit character. If it is set at the time the character is loaded from the transmit buffer into the transmit shift register, CRC is calculated on the character. CRC is not automatically sent unless this bit is set when the Transmit Underrun condition exists.

Request To Send (D_1). This is the control bit for the $\overline{RTS}$ pin. When the $\overline{RTS}$ bit is set, the $\overline{RTS}$ pin goes Low; when reset, $\overline{RTS}$ goes High. In the Asynchronous mode, $\overline{RTS}$ goes High only after all the bits of the character are transmitted and the transmitter buffer is empty. In Synchronous modes, the pin directly follows the state of the bit.

CRC-16/$\overline{SDLC}$ (D_2). This bit selects the CRC polynomial used by both the transmitter and receiver. When set, the CRC-16 polynomial ($X^{16} + X^{15} + X^2 + 1$) is used; when reset, the SDLC polynomial ($X^{16} + X^{12} + X^5 + 1$) is used. If the SDLC mode is selected, the CRC generator and checker are preset to all 1's and a special check sequence is used. The SDLC CRC polynomial must be selected when the SDLC mode is selected. If the SDLC mode is not selected, the CRC generator and checker are present to all 0's (for both polynomials).

Transmit Enable (D_3). Data is not transmitted until this bit is set and the Transmit Data output is held marking. Data or sync characters in the process of being transmitted are completely sent if this bit is reset after transmission has started. If the transmitter is disabled during the transmission of a CRC character, sync or flag characters are sent instead of CRC.

Send Break (D_4). When set, this bit immediately forces the Transmit Data output to the spacing condition, regardless of any data being transmitted. When reset, TxD returns to marking.

Transmit Bits/Character 0 and 1 (D_5 and D_6). Together, D_6 and D_5 control the number of bits in each byte transferred to the transmit buffer.

D_6 Transmit Bits/ Character 1	D_5 Transmit Bits/ Character 0	Bits/Character
0	0	Five or less
0	1	7
1	0	6
1	1	8

Bits to be sent must be right justified, least-significant bits first. The Five Or Less mode allows transmission of one to five bits per character; however, the CPU should format the data character as shown in the following table.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	1	1	1	0	0	0	D	Sends one data bit
1	1	1	0	0	0	D	D	Sends two data bits
1	1	0	0	0	D	D	D	Sends three data bits
1	0	0	0	D	D	D	D	Sends four data bits
0	0	0	D	D	D	D	D	Sends five data bits

Data Terminal Ready (D_7). This is the control bit for the $\overline{DTR}$ pin. When set, $\overline{DTR}$ is active (Low); when reset, $\overline{DTR}$ is inactive (High).

WRITE REGISTER 6

This register is programmed to contain the transmit sync character in the Monosync mode, the first eight bits of a 16-bit sync character in the Bisync mode or a transmit sync character in the External Sync mode. In the SDLC mode, it is programmed to contain the secondary address field used to compare against the address field of the SDLC frame.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Sync 7	Sync 6	Sync 5	Sync 4	Sync 3	Sync 2	Sync 1	Sync 0

WRITE REGISTER 7

This register is programmed to contain the receive sync character in the Monosync mode, a second byte (last eight bits) of a 16-bit sync character in the Bisync mode and a flag character (01111110) in the SDLC mode. WR7 is not used in the External Sync mode.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Sync 15	Sync 14	Sync 13	Sync 12	Sync 11	Sync 10	Sync 9	Sync 8

READ REGISTERS INTRODUCTION

The Z80-SIO contains three registers, RR0-RR2 (Figure 7-1), that can be read to obtain the status information for each channel (except for RR2-Channel B only). The status information includes error conditions, interrupt vector and standard communications-interface signals.

To read the contents of a selected read register other than RR0, the system program must first write the pointer byte to WR0 in exactly the same way as a write register operation. Then, by executing an input instruction, the contents of the addressed read register can be read by the CPU.

The status bits of RR0 and RR1 are carefully grouped to simplify status monitoring. For example, when the interrupt vector indicates that a Special Receive Condition interrupt has occurred, all the appropriate error bits can be read from a single register (RR1).

READ REGISTER 0

This register contains the status of the receive and transmit buffers, the $\overline{DCD}$, $\overline{CTS}$ and $\overline{SYNC}$ inputs, the Transmit Underrun/EOM latch; and the Break/Abort latch.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Break Abort	Transmit Underrun/ EOM	$\overline{CTS}$	Sync/ Hunt	$\overline{DCD}$	Transmit Buffer Empty	Interrupt Pending (Ch. A only)	Receive Character Available

Receive Character Available (D_0). This bit is set when at least one character is available in the receive buffer; it is reset when the receive FIFO is completely empty.

Interrupt Pending (D_1). Any interrupting condition in the Z80-SIO causes this bit to be set; however, it is readable only in Channel A. This bit is mainly used in applications that do not have vectored interrupts available. During the interrupt service routine in these applications, this bit indicates if any interrupt conditions are present in all Z80-SIO. This eliminates the need for analyzing all the bits of RR0 in both Channels A and B. Bit D_1 is reset when all the interrupting conditions are satisfied. This bit is always 0 in Channel B.

Transmit Buffer Empty (D_2). This bit is set whenever the transmit buffer becomes empty, except when a CRC character is being sent in a synchronous or SDLC mode. The bit is reset when a character is loaded into the transmit buffer. This bit is in the set condition after a reset.

Data Carrier Detect (D_3). The DCD bit shows the inverted state of the $\overline{DCD}$ input at the time of the last change of any of the five External/Status bits (DCD, $\overline{CTS}$, Sync/Hunt, Break/Abort or Transmit Underrun/EOM). Any transition of the $\overline{DCD}$ input causes the DCD bit to be latched and causes an External/Status interrupt. To read the current state of the DCD bit, this bit must be read immediately following a Reset External/Status Interrupt command.

Sync/Hunt (D_4). Since this bit is controlled differently in the Asynchronous, Synchronous and SDLC modes, its operation is somewhat more complex than that of the other bits and, therefore, requires more explanation.

In Asynchronous modes, the operation of this bit is similar to the DCD status bit, except that Sync/Hunt shows the state of the $\overline{SYNC}$ input. Any High-to-Low transition on the $\overline{SYNC}$ pin sets this bit and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of $\overline{SYNC}$ pin at the time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the $\overline{SYNC}$ input.

In the External Sync mode, the Sync/Hunt bit operates in a fashion similar to the Asynchronous mode, except the Enter Hunt Mode control bit enables the external sync detection logic. When the External

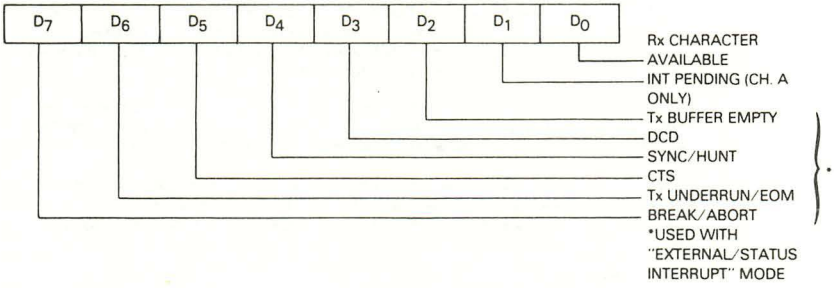
Sync Mode and Enter Hunt Mode bits are set (for example, when the receiver is enabled following a reset), the $\overline{SYNC}$ input must be held High by the external logic until external character synchronization is achieved. A High at the $\overline{SYNC}$ input holds the Sync/Hunt status bit in the reset condition.

When external synchronization is achieved, $\overline{SYNC}$ must be driven Low on the second rising edge of $\overline{RxC}$ on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive clock cycles to activate the $\overline{SYNC}$ input. Once $\overline{SYNC}$ is forced Low, it is a good practice to keep it Low until the CPU informs the external sync logic that synchronization has been lost or a new message is about to start. Refer to Figure 8.6 for timing details. The High-to-Low transition of the $\overline{SYNC}$ input sets the Sync/Hunt bit, which—in turn—sets the External/Status interrupt. The CPU must clear the interrupt by issuing the Reset External/Status Interrupt command.

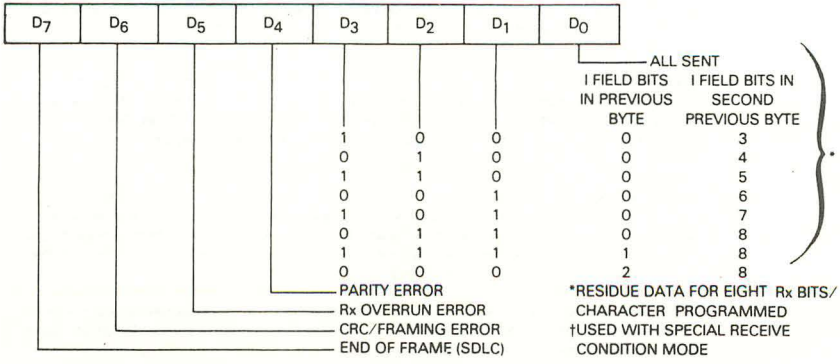
When the $\overline{SYNC}$ input goes High again, another External/Status interrupt is generated that must also be cleared. The Enter Hunt Mode control bit is set whenever character synchronization is lost or the end of message is detected. In this case, the Z80-SIO again looks for a High-to-Low transition on the $\overline{SYNC}$ input and the operation repeats as explained previously. This implies the CPU should also inform the external logic that character synchronization has been lost and that the Z80-SIO is waiting for $\overline{SYNC}$ to become active.

READ REGISTER BIT FUNCTIONS

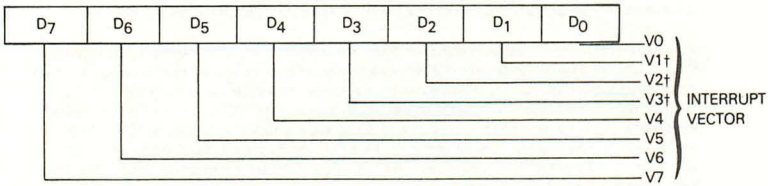
READ REGISTER 0



READ REGISTER 1†



READ REGISTER 2



†VARIABLE IF "STATUS AFFECTS VECTOR" IS PROGRAMMED

In the Monosync and Bisync Receive modes, the Sync/Hunt status bit is initially set to 1 by the Enter Hunt Mode bit. The Sync/Hunt bit is reset when the Z80-SIO establishes character synchronization. The High-to-Low transition of the Sync/Hunt bit causes an External/Status interrupt that must be cleared by the CPU issuing the Reset External/Status Interrupt command. This enables the Z80-SIO to detect the next transition of other External/Status bits.

When the CPU detects the end of message of that character and synchronization is lost, it sets the Enter Hunt Mode control bit, which—in turn—sets the Sync/Hunt bit to 1. The Low-to-High transition of the Sync/Hunt bit sets the External/Status interrupt, which must also be cleared by the Reset External/Status Interrupt command. Note that the SYNC pin acts as an output in this mode and goes Low every time a sync pattern is detected in the data stream.

In the SDLC mode, the Sync/Hunt bit is initially set by the Enter Hunt mode bit or when the receiver is disabled. In any case, it is reset to 0 when the opening flag of the first frame is detected by the Z80-SIO. The External/Status interrupt is also generated and should be handled as discussed previously.

Unlike the Monosync and Bisync modes, once the Sync/Hunt bit is reset in the SDLC mode, it does not need to be set when the end of message is detected. The Z80-SIO automatically maintains synchronization. The only way the Sync/Hunt bit can be set again is by the Enter Hunt Mode bit or by disabling the receiver.

Clear to Send (D_5). This bit is similar to the DCD bit, except that it shows the inverted state of the $\overline{CTS}$ pin.

Transmit Underrun/End of Message (D_6). This bit is in a set condition following a reset (internal or external). The only command that can reset this bit is the Reset Transmit Underrun/EOM Latch command (WR0, D_6 and D_7). When the Transmit Underrun condition occurs, this bit is set; its becoming set causes the External/Status interrupt, which must be reset by issuing the Reset External/Status Interrupt command bits (WR0). This status bit plays an important role in conjunction with other control bits in controlling a transmit operation. Refer to "Bisync Transmit Underrun" and "SDLC Transmit Underrun" for additional details.

Break/Abort (D_7). In the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when Break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, CMD_2) to the break detection logic so the Break sequence termination can be recognized.

The Break/Abort bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.

In the SDLC Receive mode, this status bit is set by the detection of an Abort sequence (seven or more 1's). The External/Status Interrupt is handled the same way as in the case of a Break. The Break/Abort bit is not used in the Synchronous Receive mode.

READ REGISTER 1

This register contains the Special Receive condition status bits and Residue codes for the I-field in the SDLC Receive Mode.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
End of Frame (SDLC)	CRC/ Framing Error	Receiver Overrun Error	Parity Error	Residue Code 2	Residue Code 1	Residue Code 0	All Sent

All Sent (D_0). In Asynchronous modes, this bit is set when all the characters have completely cleared the transmitter. Transitions of this bit do not cause interrupts. The bit is always set in Synchronous modes.

Residue Codes 0, 1, and 2 (D_1-D_3). In those cases of the SDLC receive mode where the I-field is not an integral multiple of the character length, these three bits indicate the length of the I-field. These codes are meaningful only for the transfer in which the End Of Frame bit is set (SDLC). For a receive character length of eight bits per character, the codes signify the following:

Residue Code 2	Residue Code 1	Residue Code 0	I-Field Bits In Previous Byte	I-Field Bits In Second Previous Byte
1	0	0	0	3
0	1	0	0	4
1	1	0	0	5
0	0	1	0	6
1	0	1	0	7
0	1	1	0	8
1	1	1	1	8
0	0	0	2	8

I-Field bits are right-justified in all cases

If a receive character length different from eight bits is used for the I-field, a table similar to the previous one may be constructed for each different character length. For no residue (that is, the last character boundary coincides with the boundary of the I-field and CRC field), the Residue codes are:

Bits per Character	Residue Code 2	Residue Code 1	Residue Code 0
8 Bits per Character	0	1	1
7 Bits per Character	0	0	0
6 Bits per Character	0	1	0
5 Bits per Character	0	0	1

Parity Error (D_4). When parity is enabled, this bit is set for those characters whose parity does not match the programmed sense (even/odd). The bit is latched, so once an error occurs, it remains set until the Error Reset command (WR0) is given.

Receive Overrun Error (D_5). This bit indicates that more than three characters have been received without a read from the CPU. Only the character that has been written over is flagged with this error, but when this character is read, the error condition is latched until reset by the Error Reset command. If Status Affects Vector is enabled, the character that has been overrun interrupts with a Special Receive Condition vector.

CRC/Framing Error (D_6). If a Framing Error occurs (asynchronous modes), this bit is set (and not latched) for the receive character in which the Framing error occurred. Detection of a Framing Error adds an additional one-half of a bit time to the character time so the Framing Error is not interpreted as a new start bit. In Synchronous and SDLC modes, this bit indicates the result of comparing the CRC checker to the appropriate check value. This bit is reset by issuing an Error Reset command. The bit is

not latched, so it is always updated when the next character is received. When used for CRC error and status in Synchronous modes, it is usually set since most bit combinations result in a non-zero CRC, except for a correctly completed message.

End of Frame (D_7). This bit is used only with the SDLC mode and indicates that a valid ending flag has been received and that the CRC Error and Residue codes are also valid. This bit can be reset by issuing the Error Reset command. It is also updated by the first character of the following frame.

READ REGISTER 2 (Ch. B Only)

This register contains the interrupt vector written into WR2 if the Status Affects Vector control bit is not set. If the control bit is set, it contains the modified vector shown in the Status Affects Vector paragraph of the Write Register 1 section. When this register is read, the vector returned is modified by the highest priority interrupting condition at the time of the read. If no interrupts are pending, the vector is modified with $V_3=0$, $V_2=1$, and $V_1=1$. This register may be read only through Channel B.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0
				Variable if Status Affects Vector is enabled			

Stichwortverzeichnis

von Rodnay Zaks

das Standardwerk über CP/M, das meistgebrauchte Betriebssystem für Mikrocomputer. Für Anfänger eine verständliche Einführung, für Fortgeschrittene ein umfassendes Nachschlagewerk über die CP/M-Versionen 2.2, 3.0 und CCP/M-86 sowie MP/M., 2. überarbeitete Ausgabe.

356 Seiten, 56 Abbildungen, Best.-Nr.: **3053** (1984)

von J. W. Coffron

vermittelt alle nötigen Anweisungen, um Peripherie-Bausteine mit dem Z80 zu steuern und individuelle Hardware-Lösungen zu realisieren.

296 Seiten, 204 Abbildungen, Best.-Nr.: **3037** (1984)

Die SYBEX-Bibliothek

Einführende Literatur

CHIP UND SYSTEM: Einführung in die Mikroprozessoren-Technik
von Rodnay Zaks – eine sehr gut lesbare Einführung in die faszinierende Welt der Computer, vom Microprozessor bis hin zum vollständigen System. 2., überarbeitete und aktualisierte Ausgabe. 568 Seiten, 325 Abbildungen, Best.-Nr.: **3601** (1985)

Pascal

EINFÜHRUNG IN PASCAL UND UCSD/PASCAL
von Rodnay Zaks – das Buch für jeden, der die Programmiersprache PASCAL lernen möchte. Vorkenntnisse in Computerprogrammierung werden nicht vorausgesetzt. Eine schrittweise Einführung mit vielen Übungen und Beispielen. 535 Seiten, 130 Abbildungen, Best.-Nr.: **3004** (1982)

PASCAL PROGRAMME – MATHEMATIK, STATISTIK, INFORMATIK
von Alan Miller – eine Sammlung von 60 der wichtigsten wissenschaftlichen Algorithmen samt Programmauflistung und Musterdurchlauf. Ein wichtiges Hilfsmittel für Pascal-Benutzer mit technischen Anwendungen. 398 Seiten, 120 Abbildungen, Format 23 x 18 cm, Best.-Nr.: **3007** (1982)

GRUNDKURS IN PASCAL Bd. 1
von K.-H. Rollke – der sichere Einstieg in Pascal, speziell für Schule und Fortbildung (Reihe SYBEX Informatik). 224 Seiten, mit Abb., Format 17,5x25 cm, Best.-Nr. **3046** (1984), Lehrerbegleitheft Best.-Nr. **3059**

GRUNDKURS IN PASCAL BAND 2
von K.H. Rollke – Mit diesem Buch wird der Pascal-Grundkurs aus der Reihe SYBEX Informatik abgerundet. Für Lehrer, Schüler, Teilnehmer an Pascal-Kursen, Studenten und Autodidakten. 224 Seiten, mit Abb., Best.-Nr. **3061** (1985), Lehrerbegleitheft Best.-Nr. **3090**

DAS TURBO PASCAL BUCH
von Karl-Hermann Rollke – Sie lernen die Arbeitsweise des Turbo-Editors und des Systems kennen und werden mit Programmierkonzepten, Daten- und Kontrollstrukturen für den Programmfluß vertraut gemacht. 288 Seiten, mit Abbildungen, Best.-Nr.: **3608** (1985)

DAS PASCAL HANDBUCH
von Jacques Tiberghien – ein Wörterbuch mit jeder Pascal-Anweisung und jedem Symbol, reservierten Wort, Bezeichner und Operator, für beinahe alle bekannten Pascal-Versionen incl. Turbo Pascal. 520 Seiten, 270 Abbildungen, Format 23 x 18 cm, Best.-Nr.: **3614** (1986)

DAS ARBEITSBUCH ZU TURBO PASCAL
von Karl-Udo Bromm u.a. – Der Autor, ein erfahrener Pädagoge, vermittelt dem mit Turbo Pascal arbeitenden Leser eine Fülle wertvoller Routinen zu den unterschiedlichsten Themenbereichen für Schule, Hobby und Beruf. Ca. 320 Seiten, ca. 60 Abb., Best.-Nr. **3629** (1986)

Andere Programmiersprachen

ERFOLGREICH PROGRAMMIEREN MIT C
von J. A. Illik – ein unentbehrliches Handbuch für jeden, der mit der universellen Sprache C erfolgreich programmieren will. Aussagekräftige Beispiele, auf verschiedenen Mini- und Mikrocomputern getestet. 408 Seiten, Best.-Nr.: **3055** (1984)

C – EINE EINFÜHRUNG
von Bruce H. Hunter – Das ideale Buch für den Einsteiger in die Programmiersprache C, speziell für Anwender, die von BASIC auf den leistungsfähigen Compiler umsteigen wollen. Ca. 256 Seiten, ca. 12 Abb., Best.-Nr. **3632** (1986)

Anwendungssoftware

EINFÜHRUNG IN WORDSTAR
von Arthur Naiman – eine klar gegliederte Einführung, die aufzeigt, wie das Textbearbeitungsprogramm WORDSTAR funktioniert, was man damit tun kann und wie es eingesetzt wird. 240 Seiten, 36 Abbildungen, Best.-Nr.: **3019** (1983)

PRAKTISCHE WORDSTAR-ANWENDUNGEN
von J. A. Arca – das Buch für Einsteiger, um nach kurzer Zeit praktische Textverarbeitungs-Probleme zu lösen, eine programmierte Unterweisung zur Leistungsoptimierung mit WORDSTAR. 368 Seiten, 69 Abbildungen, Best.-Nr.: **3057** (1985)

ERFOLG MIT VisiCalc
von D. Hergert – umfassende Einführung in VisiCalc und seine Anwendung. Zeigt Ihnen u. a.: Aufstellung eines Verteilungsbogens, Benutzung von VisiCalc-Formeln, Verwendung der DIF-Datei-Funktion. 224 Seiten, 58 Abbildungen, Best.-Nr.: **3030** (1983)

ERFOLG MIT MULTIPLAN
von Th. Ritter – das Tabellenkalkulations-Programm Multiplan hilft Ihnen bei der Lösung kommerzieller, wissenschaftlicher und allgemeiner Probleme. Lernen Sie die Möglichkeiten kennen, Ihre Software optimal zu nutzen! 208 Seiten, 68 Abbildungen, Best.-Nr.: **3043** (1984)

ARBEITEN MIT dBASE II
von A. Simpson – Grundlagen und Programmiertechniken für die Datenbank-Verwaltung mit dBASE II. Zahlreiche praktische Tips. 264 Seiten, 50 Abb., Best.-Nr. **3070** (1984)

dBASE II PROFIBUCH
von Alan Simpson – eine schrittweise Unterweisung in fortgeschrittenen Programmiertechniken; vom Erkennen des Problems bis zu Programmsystemen für geschäftliche Anwendungen. 432 Seiten, zahlr. Abbildungen, Best.-Nr. **3610** (1986)

SYBEX RATGEBER dBASE II
von Gerhard Renner – Das sichere Nachschlagewerk mit kurzem Zugriff für PC-Anwender, die nicht die Zeit haben, Befehle und sonstige Detail-Informationen langwierig zu suchen. Mit vielen Querverweisen und Suchhilfen. 360 Seiten, Best.-Nr. **3305** (1986)

MIKROPROZESSOR INTERFACE TECHNIKEN (3. überarbeitete Ausgabe)
von Rodnay Zaks/Austin Lesea – Hardware- und Software-Verbindungstechniken samt Digital/Analog-Wandler, Peripheriegeräte, Standard-Busse und Fehlersuchtechniken. 432 Seiten, 400 Abbildungen, Format DIN A5, Best.-Nr.: **3012** (1982)

Kommunikation

DAS MODEMBUCH ZUR DFÜ
von Bruno Hurth und Manfred Hurth – Jetzt steht Ihnen ein Nachschlagewerk mit einer Fülle unentbehrlicher Informationen zur Verfügung, auf das Sie immer wieder zurückgreifen werden: Möglichkeiten der DFÜ im Bereich der DBP, Mailboxen, Akustikkoppler, Btx u.v.m. 224 Seiten, Best.-Nr.: **3619** (1985)

SYBEX MAILBOX FÜHRER
Alles über den Zugang zu elektronischen Briefkästen in Deutschland: Voraussetzungen für die Teilnahme an der Datenkommunikation; Telefon-Nummern, Eingangs- und Untermenüs wichtiger Mailboxen und was diese Ihnen nutzen können, u.v.m. 272 Seiten, Best.-Nr.: **3663** (überarbeitete Ausgabe 1986)

V24/RS-232 KOMMUNIKATION
von J. Campbell – zeigt Ihnen, wie Sie mit den Schnittstellen V24 und RS-232 notwendiges Zubehör an Ihren Rechner anschließen; mit praktischen Fallbeispielen. 224 Seiten, 97 Abb., Best.-Nr. **3075** (1984)

ONLINE DATENBANKEN
Zugang zum Wissen der Welt mit Mikrocomputern von **Steffen Schubert** – Manager aller Unternehmensgruppen, Wissenschaftler und Ingenieure erhalten einen Überblick über alle Datenbank-Typen, bekommen wichtige Anbieter in Europa und Übersee genannt und lernen, Datenbanken effektiv und kostengünstig zu nutzen. 200 Seiten, Best.-Nr. **3621** (1986)

Fordern Sie ein Gesamtverzeichnis unserer Verlagsproduktion an:

SYBEX-VERLAG GmbH
Vogelsanger Weg 111
4000 Düsseldorf 30
Tel.: (02 11) 61 80 2-0
Telex: 8 588 163

SYBEX INC.
2021 Challenger drive, NBR 100
Alameda, CA 94501, USA
Tel.: (4 15) 523-8233
Telex: 287 639 SYBEX UR

SYBEX
6–8, Impasse du Curé
75018 Paris
Tel.: 1/203-95-95
Telex: 211.801 f

SYBEX

Z80 Anwendungen

Entwickeln Sie Ihre eigenen Anwendungen mit dem Z80 Mikroprozessor. Dieses Buch – leicht verständlich geschrieben und klar illustriert – hilft Ihnen dabei. Sie lernen alle notwendigen Anweisungen kennen, um Peripherie-Bausteine mit dem Z80 zu steuern und damit individuelle Hardware-Lösungen zu realisieren.

Zahlreiche Schaltbilder und Beispiele geben Ihnen anschauliche Informationen über den Gebrauch von:

- ROMs und statischen RAMs für den Z80.
- Bausteinen für Ein- und Ausgabe.
- Dynamischen RAMs.
- Interrupts.
- Peripherie-Bausteinen einschließlich Z80-SIO, PIO und CTC.

Lernen Sie, wie leicht Sie ein eigenes System entwickeln können, das auf Ihre spezifischen Bedürfnisse zugeschnitten ist.

Über den Autor:
James W. Coffron ist Ingenieur für Computersysteme und hat sich auf die Entwicklung und das Testen von Mikrocomputern spezialisiert. Er leitet Seminare im akademischen und industriellen Bereich und ist Autor verschiedener Bücher über Mikroprozessoren.

ISBN 3-88745-037-X

DM 48,–
sfr 44,20
öS 374,–

9 783887 450373