7.5 MC68230
PI/T Parallel Interface/Timer

Ein Baustein, der in den wenigsten Applikationen fehlen darf, ist eine parallele Schnittstelle. Mit dem MC68230 stehen den M68000-Systemen zwei sehr gefragte Betriebsmittel zur Verfügung, nämlich Allzweck-Parallelinterface-Ports und ein betriebssystemorientierter Zeitgeber.
Der PI/T besteht aus drei im wesentlichen unabhängigen Abschnitten: den Ports, dem Zeitgeber und einem asynchronen Bus- (68000-) Interface (Bild 7-12). Der PI/T bietet drei 8bit-Paralleldatenports, von denen zwei miteinander verkettet werden können, um ein 16bit-Port zu bilden, falls dies gewünscht wird. Diese Ports arbeiten in vier Hauptbetriebsarten, die programmiert werden können, unidirektional und bidirektional, jeweils mit einer Breite von 8 oder 16 Bit. In jedem dieser Modi sind sowohl doppelt gepufferte Datenwege für Ein- oder Ausgabeübertragungen als auch einfach gepufferte I/O-Konfigurationen programmierbar (sogenannte Submodi, siehe Tabellen 7-5a und b). Doppelt gepuffert bedeutet, daß der PI/T zwei Datenworte direkt hintereinander einlesen bzw. ausgeben kann (FIFO). Das dritte Port ist mit Doppelfunktionen belegt, wobei jeder Anschluß als Port oder als andere Funktion eingesetzt werden kann, die entweder mit dem Zeitgeber oder den beiden anderen Ports verbunden ist. Außerdem sind vier Mehrzweck-Handshake-Anschlüsse vorgesehen. Sie steuern die Übertragungen in die bzw. aus den doppelt gepufferten Ports, können jedoch auch für andere Zwecke benutzt werden. Somit stehen also insgesamt 28 vom Benutzer programmierbare Leitungen zur Verfügung.

Beim Aufbau des Zeitgebers wurde in erster Linie darauf geachtet, daß Systemfunktionen geschaffen werden, die von vielen, für den 68000-Mikroprozessor konzipierten Betriebssystemen benötigt werden. Er kann jedoch andere Allzweck-Zeitgeber, wie z. B. den M6840 nicht vollständig ersetzen. Der PI/T kann so programmiert werden, daß eine beträchtliche Flexibilität ermöglicht wird, wobei er fünf allgemeine Anwendungsgebiete abdeckt:

- periodische Interrupts
- Rechteckgenerator
- Einzelinterrupt nach vorprogrammierter Zeiteinheit
- Messen einer abgelaufenen Zeiteinheit
- Geräteüberwachung ("Watchdog Timer")

Der PI/T ist mit einem asynchronen 8bit-Datenbusinterface versehen. Er ist mit Bus Mastern, wie dem 68000 und dem DMA-Controller 68450, kompatibel. Sämtliche Funktionen des M68000 im Zusammenhang mit der Steuerung der Peripherie können ausgenutzt werden. **Er ist mit der Non-Autovektor-Interruptstruktur des M68000 kompatibel und liefert Interruptvektoren für alle On-Chip-Quellen. Zur Benutzung während eines 16bit-Datentransfers und in der Initialisierungsphase werden interne Register im Hinblick auf Kompatibilität mit dem MOVEP-Befehl des M68000 angesprochen.** Dadurch werden 16- oder 32bit-Datentransfers zwischen einem Datenregister des Prozessors und Peripheriebausteinen mit 8bit-Datenbus, wie dem PI/T, ohne Hardwareaufwand ermöglicht. Der DMAC 68450 kann so programmiert werden, daß er mit demselben Datenformat wie der PI/T arbeitet. Auf diese Weise können Daten zwischen dem PI/T und einer anderen Peripherie bzw. Speicher übertragen werden. Dabei können alle vier Hauptmodi des PI/T Anwendung finden.
Der Baustein wird in der HMOS-Technologie gefertigt und ist in einem 48-Pin-Gehäuse untergebracht (Bild 7-13).

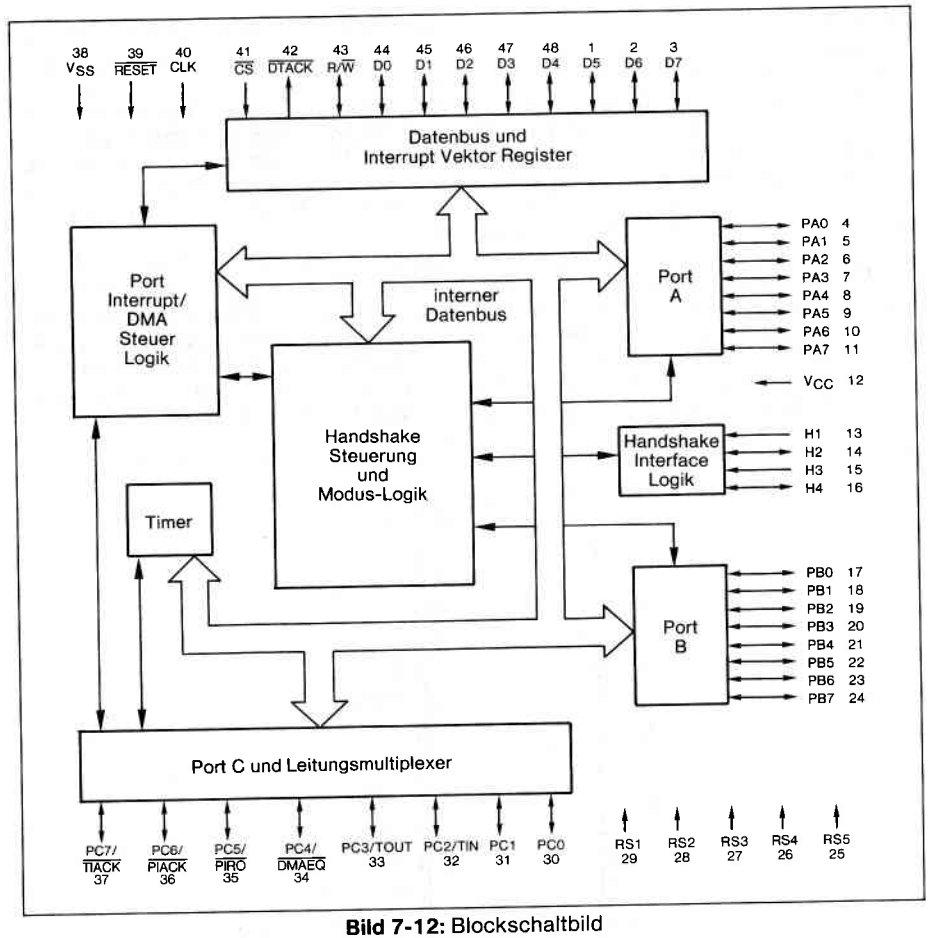


Bild 7-12: Blockschaltbild

7.5.1 Benutzer-Interface und Programmiermodell

Wie aus Bild 7-12 ersichtlich, sind die Ports A und B Mehrzweck-Parallelports, die je nach Bedarf des Benutzers in unidirektionale oder bidirektionale 8- oder 16bit-Betriebsarten konfigurierbar sind (siehe Kapitel 7.5.2). Die Handshake-Signale H1, H2, H3 und H4 ermöglichen, je nach Betriebsart, die Steuerung der Datentransfers von und zu den Ports A und B. Sie dienen auch als Eingänge für die Status/Interruptgenerierung oder als Allzweck E/A-Anschlüsse. Der dritte Port, Port C, enthält verschiedene Doppelfunktionsanschlüsse.
Je nach Konfiguration werden bis zu drei Anschlüsse vom Zeitgeber belegt. PC2/TIN kann als externer Takteingang für den Zeitgeber oder als "Start/Stop"-Eingang des Watchdog-Timers eingesetzt werden. PC3/TOUT dient als Interruptanforderung des Zeitgebers, als Rechteckspannungsausgang oder als Watchdog-Timer-Ausgang. PC7/TIACK wird nur für die Interruptquittierung des Zeitgebers in vektorgesteuerten Interruptsystemen verwendet.
Eine allgemeine Eigenschaft des PI/T besteht darin, daß in Anwendungen, in denen ein E/A-Anschluß nicht für Aufgaben in einem bestimmten Protokoll benötigt wird, dieser An-

schluß für einen anderen Zweck zur Verfügung steht. Dies gilt insbesondere für jeden der Doppelfunktionsanschlüsse von Port C und die vier Handshakeanschlüsse. Damit ist eine ausgezeichnete Anschlußausnutzung gewährleistet.

PC5/$\overline{\text{PIRQ}}$ und PC6/$\overline{\text{PIACK}}$ stellen die Interrupt-Anforderung und -Quittierung für die Ports dar, vergleichsweise wie die Pins TOUT und $\overline{\text{TIACK}}$ für den Zeitgeber. Dies erlaubt die Anhebung der Portfunktionen auf eine andere Interruptebene als beispielsweise der Zeitgeber. Ganz allgemein gesagt, sind die Port- und Zeitgeberfunktionen vollständig unabhängig voneinander und verhalten sich, mit Ausnahme der Doppelfunktionsanschlüsse, so, als wären sie auf zwei verschiedenen Chips.

Das Interface für den Systembus erfordert Chipselect- (CS), Lese-/Schreib- (R/$\overline{\text{W}}$) und fünf Registerauswahl-Eingänge (RS1 – RS5), um normale Lese- und Schreibzyklen einzuleiten. Interruptquittierungszyklen von Port oder Zeitgeber können durch $\overline{\text{PIACK}}$ oder $\overline{\text{TIACK}}$ eingeleitet werden, wenn diese Anschlüsse als solche programmiert sind. Für jeden Typ eines Buszyklusses antwortet der PI/T mit Ausgabe des $\overline{\text{DTACK}}$-Signales, wenn die entsprechende Zugriffszeit abgelaufen ist. Dies kann von einem asynchronen Busmaster wie dem 68000 oder 68450 zur Beendigung des Zyklus benutzt werden.

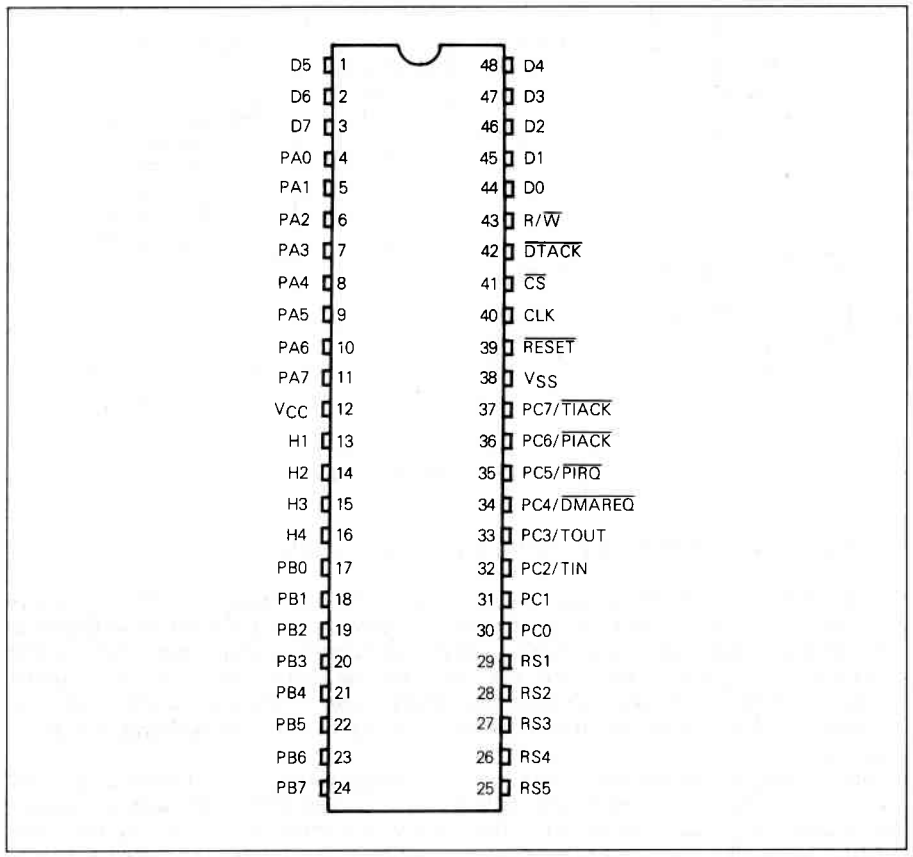


Bild 7-13: Anschlußbelegung PI/T

Register-Auswahl-Leitungen RS 5 4 3 2 1	Register-Bit 7	6	5	4	3	2	1	0	Wert nach RESET (Hex)	Register	Abkürzung
0 0 0 0 0	Port Modus Steuerung		H 3/4 Freigabe	H 1/2 Freigabe	H4 akt. Pegel	H3 akt. Pegel	H2 akt. Pegel	H1 akt. Pegel	00	Port General Control Register	PGCR
0 0 0 0 1	*	SVCRQ Auswahl		IPF Auswahl		Port Interrupt Prioritätssteuerung			00	Port Service Request Register	PSRR
0 0 0 1 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	00	Port A Data Direction Register	PADDR
0 0 0 1 1	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	00	Port B Data Direction Register	PBDDR
0 0 1 0 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	00	Port C Data Direction Register	PCDDR
0 0 1 0 1	Interrupt Vektor Nummer						Bit 1	Bit 0	0F	Prot Interrupt Vektor Register	PIVR
0 0 1 1 0	Port A Submodus		H2 Steuerung			H2 Int Freigabe	H1 SVCRQ Freigabe	H1 Stat Str.	00	Port A Control Register	PACR
0 0 1 1 1	Port B Submodus		H4 Steuerung			H4 Int Freigabe	H3 SVCRQ Freigabe	H3 Stat Str.	00	Port B Control Register	PBCR
0 1 0 0 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	**	Port A Data Register	PADR
0 1 0 0 1	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	**	Port B Data Register	PBDR
0 1 0 1 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	***	Port A Alternate Register	PAAR
0 1 0 1 1	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	***	Port B Alternate Register	PBAR
0 1 1 0 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	****	Port C Data Register	PCDR
0 1 1 0 1	H4 Pegel	H3 Pegel	H2 Pegel	H1 Pegel	H4S	H3S	H2S	H1S	****	Port Status Register	PSR
0 1 1 1 0	*	*	*	*	*	*	*	*	00	(Null)	–
0 1 1 1 1	*	*	*	*	*	*	*	*	00	(Null)	–

* Unbenutzt, wird als Null gelesen
** Wert vor $\overline{\text{RESET}}$
*** Aktueller Pegel der Pins
**** Unbestimmter Wert

Bild 7-14a: Register (Ports)

Register-Auswahl-Leitungen RS 5 4 3 2 1	Register-Bit 7	6	5	4	3	2	1	0	Wert nach RESET (Hex)	Register	Abkürzung
1 0 0 0 0	TOUT/TIACK Steuerung			ZD Str.	*	Takt Steuerung		Timer Freigabe	00	Timer Control Register	TCR
1 0 0 0 1	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	0F	Timer Interrupt Vector Register	TIVR
1 0 0 1 0	*	*	*	*	*	*	*	*	00	(Null)	–
1 0 0 1 1	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16	**	Counter Preload Register (High)	CPRH
1 0 1 0 0	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	**	Counter Preload Register (Mid)	CPRM
1 0 1 0 1	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	**	Counter Preload Register (Low)	CPRL
1 0 1 1 0	*	*	*	*	*	*	*	*	00	(Null)	–
1 0 1 1 1	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16	**	Count Register (High)	CNTRH
1 1 0 0 0	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	**	Count Register (Mid)	CNTRM
1 1 0 0 1	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	**	Count Register (Low)	CNTRL
1 1 0 1 0	*	*	*	*	*	*	*	ZDS	00	Timer Status Register	TSR
1 1 0 1 1	*	*	*	*	*	*	*	*	00	(Null)	–
1 1 1 0 0	*	*	*	*	*	*	*	*	00	(Null)	–
1 1 1 0 1	*	*	*	*	*	*	*	*	00	(Null)	–
1 1 1 1 0	*	*	*	*	*	*	*	*	00	(Null)	–
1 1 1 1 1	*	*	*	*	*	*	*	*	00	(Null)	–

* nicht benutzt, wird als Null gelesen
** Wert vor RESET

Bild 7-14b: Register (Timer)

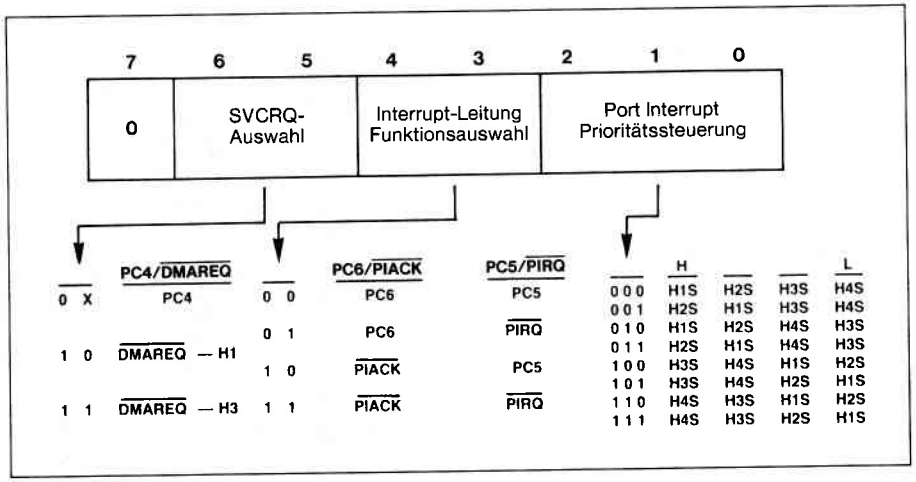


Bild 7-14c: Port Service Request-Register (PSRR)

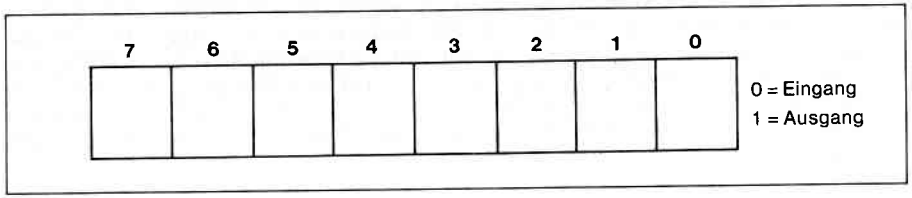


Bild 7-14d: Daten-Richtungsregister (PADDR, PBDDR, PCDDR)

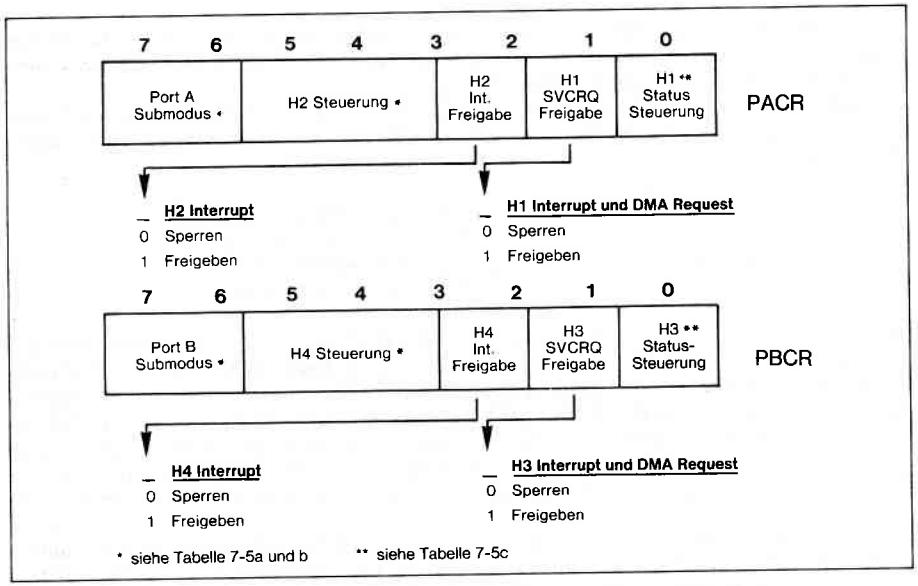


Bild 7-14e: Port-Steuerregister (PACR, PBCR)

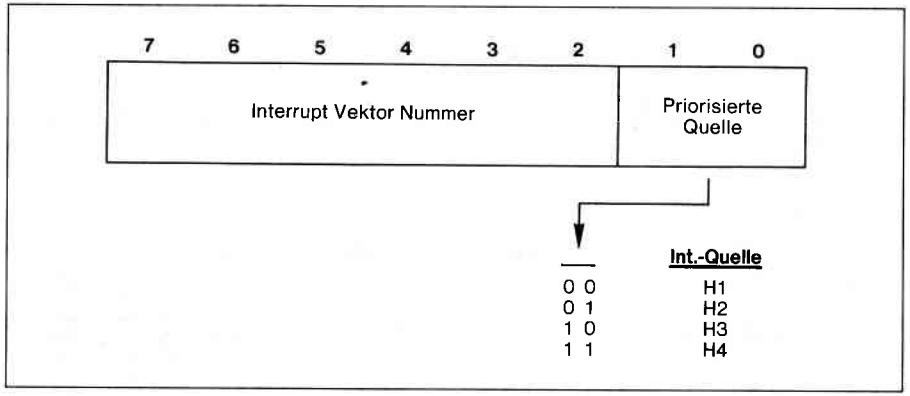


Bild 7-14f: Port Interrupt-Vektorregister (PIVR)

Mit den insgesamt fünf Registerauswahl-Eingängen RS1 – RS5 sind maximal 32 Register adressierbar, jedes Register mit einer Breite von 8 Bit. 25 Register sind erforderlich, um alle Steuer-, Daten- und Statusfunktionen zu realisieren. Sie sind in Bild 7-14 dargestellt. Zwei mit "nicht verwendet" gekennzeichnete Register sind so angeordnet, daß auf Gruppen, die Zeitgeberregister betreffen, bequem mit Hilfe des MOVEP-Befehles (Langwort) zugegriffen werden kann (siehe Kap. 7.5.4). Mit einer Gruppe werden damit beispielsweise auf einmal alle 24 Bit des Zeitgebers geladen, mit einer anderen Gruppe werden sie gelesen. So lassen sich diese Register vom Anwender mit einem Einzel-Befehl manipulieren.

7.5.2 Die Modi und die Parallelports

Viele Anwendungen konzentrieren sich im wesentlichen auf die Ports A und B, die vier Handshake-Anschlüsse, die Port-Interruptanschlüsse und den DMA-Anforderungsanschluß. Sie werden folgendermaßen gesteuert:
Das allgemeine Steuerregister der Ports (PGCR, Registeradresse 0) enthält ein 2bit-Feld (Bit 6 und 7), mit denen die vier Hauptbetriebsarten programmiert werden (siehe Tabellen 7-5a und b):

- Modus 0 (00) Unidirektionaler 8bit-Modus
- Modus 1 (01) Unidirektionaler 16bit-Modus
- Modus 2 (10) Bidirektionaler 8bit-Modus
- Modus 3 (11) Bidirektionaler 16bit-Modus

Diese Betriebsarten steuern den Gesamtbetrieb der Ports A und B und bestimmen ihre Beziehung zueinander. Darüber hinaus erfordern einige Modi spezifische Informationen für jeden Port. Diese Informationen werden durch 2bit-Submodus-Felder in den Steuerregistern von Port A und B (PACR, PBCR, Registeradresse 6 und 7) geliefert (siehe Tabellen 7-5a und b). Jede Modus-/Portsubmodus-Kombination gibt einen Satz von programmierbaren Eigenschaften an, die das Verhalten dieses Ports und zwei der Handshake-Leitungen vollständig definieren. Diese Struktur ist in der Tabelle 7-5 zusammengefaßt.

In Modus 0 bis 1 werden die Anschlüsse von Port A und B als Eingänge oder Ausgänge gesteuert, indem die Datenrichtungsregister (Data Direction Register, PADDR, PBDDR) von Port A bzw. B programmiert werden. Eine 0 bedeutet Eingang, eine 1 dagegen Ausgang (Bild 7-14d) der entsprechenden Leitung. Hier handelt es sich um eine statische De-

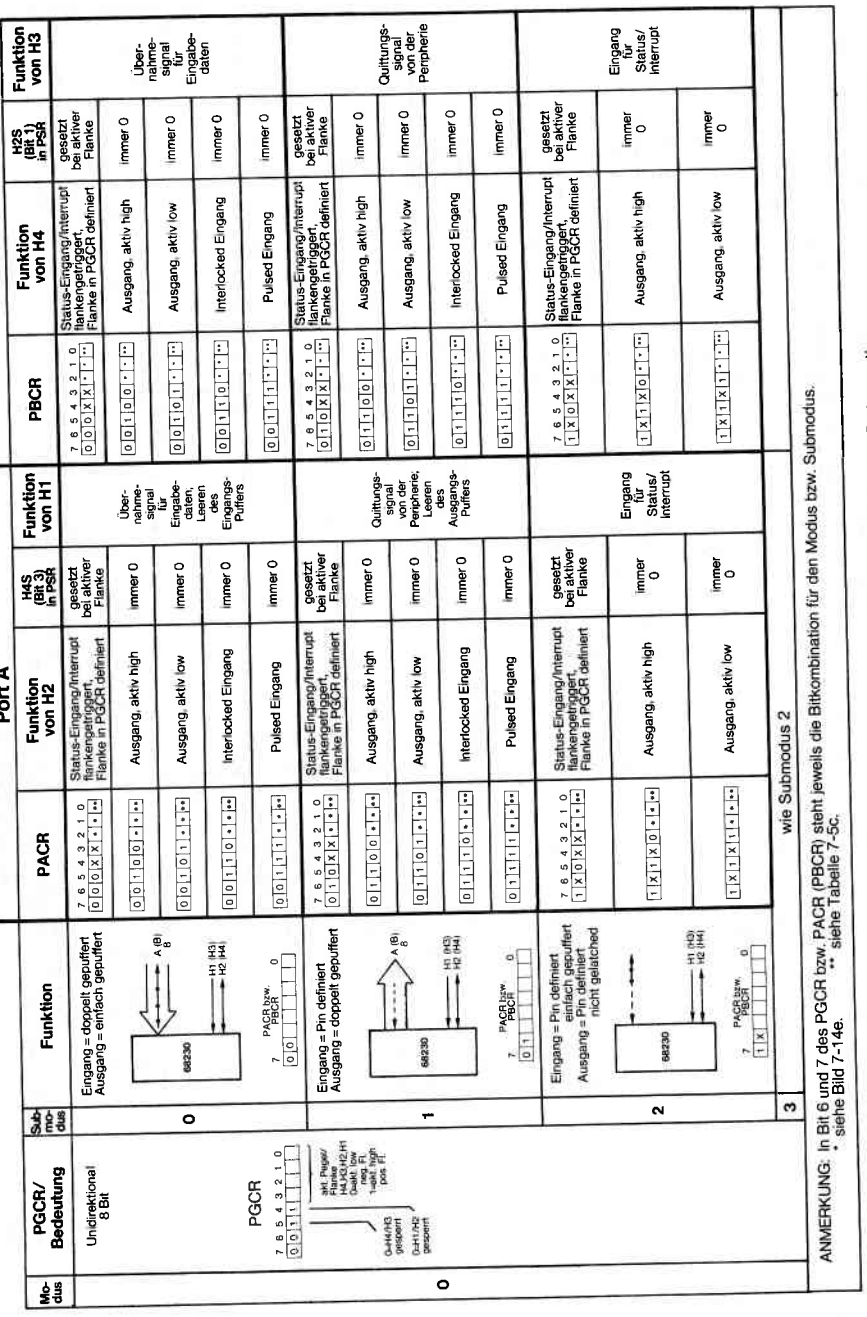


Tabelle 7-5a: Zusammenfassung des Modus 0 und Submodi

Modus	PGCR/ Bedeutung	Submodus	Funktion	Port A: PACR	Port A: Funktion von H2	Port A: H2S (Bit 1) in PSR	Port A: Funktion von H1	Port B: PBCR	Port B: Funktion von H4	Port B: H4S (Bit 3) in PSR	Port B: Funktion von H3
1	Unidirektional 16 Bit PGCR 7 6 5 4 3 2 1 0 0 1 siehe Tab. 7-5a	0	Eingang = doppelt gepuffert Ausgang = einfach gepuffert	X 0 0 X X * * **	Status-Eingang/Interrupt flankengetriggert, Flanke in PGCR definiert	gesetzt bei aktiver Flanke	Status- oder Interrupt-Eingang siehe Tab. 7-5c	X 0 0 X X * * **	Status-Eingang/Interrupt flankengetriggert, Flanke in PGCR definiert	gesetzt bei aktiver Flanke	Übernahmesignal der 16bit-Daten
				X 0 1 X 0 * * **	Ausgang, aktiv high	immer 0		X 0 1 0 0 * * **	Ausgang, aktiv high	immer 0	
				X 0 1 X 1 * * **	Ausgang, aktiv low	immer 0		X 0 1 1 0 * * **	interlocked (Eingang)	immer 0	
								X 0 1 1 1 * * **	pulsed (Eingang)	immer 0	
		1	Eingang = Bit I/O nicht gelatched Ausgang = doppelt gepuffert	X 1 0 X X * * **	Status-Eingang/Interrupt flankengetriggert, Flanke in PGCR definiert	gesetzt bei aktiver Flanke	Status- oder Interrupt-Eingang siehe Tab. 7-5c	X 1 0 X X * * **	Status-Eingang/Interrupt flankengetriggert, Flanke in PGCR definiert	gesetzt bei aktiver Flanke	Quittungssignal von der Peripherie
				X 1 1 X 0 * * **	Ausgang, aktiv high	immer 0		X 1 1 0 0 * * **	Ausgang, aktiv high	immer 0	
				X 1 1 X 1 * * **	Ausgang, aktiv low	immer 0		X 1 1 0 1 * * **	interlocked (Eingang)	immer 0	
								X 1 1 1 1 * * **	pulsed (Eingang)	immer 0	
		2	wie Submodus 0								
		3	wie Submodus 1								
2	Bidirektional 8 Bit PGCR 7 6 5 4 3 2 1 0 1 0 siehe Tab. 7-5a	–	Eingang = Bit I/O Ausgang = doppelt gepuffert	X X X X X 0 * * **	interlocked (Ausgang)	immer 0	Gibt die Richtung der Daten von Port B an	X X X X X 0 * * **	interlocked (Ausgang)	immer 0	Übernahmesignal für Port B
				X X X X 1 * * **	pulsed (Ausgang)	immer 0		X X X X 1 * * **	pulsed (Ausgang)	immer 0	
3	Bidirektional 16 Bit PGCR 7 6 5 4 3 2 1 0 1 1 siehe Tab. 7-5a	–	Eingang = doppelt gepuffert Ausgang = doppelt gepuffert	X X X X 0 * * **	interlocked (Eingang)	immer 0	Gibt die Richtung der Daten von Port A und B an	X X X X 0 * * **	interlocked (Eingang)	immer 0	Übernahmesignal der 16bit-Daten
				X X X X 1 * * **	pulsed (Eingang)	immer 0		X X X X 1 * * **	pulsed (Eingang)	immer 0	

ANMERKUNG: In Bit 6 und 7 des PGCR bzw. PACR (PBCR) steht jeweils die Bitkombination für den Modus bzw. Submodus.
* siehe Bild 7-14e. ** siehe Tabelle 7-5c.

Tabelle 7-5b: Zusammenfassung der Modi 1, 2 und 3 und Submodi

Modus*	Submodus	PACR	Bedeutung von H1S: gesetzt (1)	Bedeutung von H1S: gelöscht (0)	PBCR	Bedeutung von H3S: gesetzt (1)	Bedeutung von H3S: gelöscht (0)
0 UNI 8	00	7 6 5 4 3 2 1 0 0 0 X X X X X	Datenregister A nicht gelesen	Datenregister A gelesen	7 6 5 4 3 2 1 0 0 0 X X X X X	Datenregister B nicht gelesen	Datenregister B gelesen
	01	0 1 X X X X X 0	Datenregister A nicht voll	Datenregister A voll	0 1 X X X X X 0	Datenregister B nicht voll	Datenregister B voll
		0 1 X X X X X 1	Datenregister A leer	Datenregister A nicht leer	0 1 X X X X X 1	Datenregister B leer	Datenregister B nicht leer
	1X	1 X X X X X X X	bei aktiver Flanke	durch Schreiben in Bit 0, PACR	1 X X X X X X X	bei aktiver Flanke	durch Schreiben in Bit 0, PBCR
1 UNI 16	X0	X 0 X X X X X X	Datenregister A nicht gelesen	Datenregister A gelesen	X 0 X X X X X X	Datenregister B nicht gelesen	Datenregister B gelesen
	X1	X 1 X X X X X X	bei aktiver Flanke	durch Schreiben in Bit 0, PACR	0 X 1 X X X X 0	Datenregister B nicht voll	Datenregister B voll
					1 X 1 X X X X 1	Datenregister B leer	Datenregister B nicht leer
2 BI 8	XX	X X X X X X X 0	Datenregister B nicht voll	Datenregister B voll	X X X X X X X X	Datenregister B nicht gelesen	Datenregister B gelesen
		X X X X X X X 1	Datenregister B leer	Datenregister B nicht leer			
3 BI 16	XX	X X X X X X X 0	Datenregister A und B nicht voll	Datenregister A und B voll	X X X X X X X X	Datenregister A und B nicht gelesen	Datenregister A und B gelesen
		X X X X X X X 1	Datenregister A und B leer	Datenregister A und B nicht leer			

* in PGCR definiert, wie Tabelle 7-5a und b.

Tabelle 7-5c: Bedeutung von H1S und H3S

finition, die nur durch Rücksetzen des Chips oder durch Neuprogrammierung des Datenrichtungsregisters geändert werden kann. Modus 0 und 1 werden demzufolge als unidirektionale Modi bezeichnet. Mit Ausnahme des Bit-E/A-Submodus erlauben sie doppelt gepufferte Datentransfers in der Richtung, die durch die Definition von Modus und Submodus bestimmt wird (siehe Tabellen 7-5a und b). Diese Richtung ist als Primärrichtung bekannt. Datentransfers in der Primärrichtung werden durch die Handshake-Anschlüsse gesteuert. Datentransfers, die nicht in die Primärrichtung gehen, stehen im allgemeinen in keiner Beziehung zueinander. Für sie werden einzelne oder ungepufferte Datenwege zur Verfügung gestellt. **In Modus 2 und 3 besteht das Konzept der Primärrichtung nicht, und die Datenrichtungsregister haben keine Wirkung (mit Ausnahme von Bit E/A in Modus 2). Diese Betriebsarten werden als bidirektional bezeichnet, da die Richtung jedes Transfers dynamisch durch den Status der Handshake-Anschlüsse bestimmt wird und unter der vollen Kontrolle der externen Schaltungsanordnung des Benutzers steht.** So können beispielsweise Daten aus den Ports heraus übertragen werden, und kurz darauf folgt eine Übertragung in dieselben Portanschlüsse. Die Transfers in beide Richtungen sind unabhängig und können in jeder beliebigen Folge auftreten. Der Handshake-Anschluß H3 kann immer dazu benutzt werden, Eingabedaten einzuspeichern, während der Anschluß H1 bestimmt, ob es sich bei den Portausgabepuffern um Tri-State-Puffer handelt oder ob sie den Bus treiben. H2 gibt an, daß gültige Ausgabedaten in den internen Signalspeichern des Ports vorhanden sind. Der Handshake-Anschluß H4 gibt an, wann die Eingangsspeicher (Latches) des Ports zur Aufnahme neuer Daten bereit sind.

Allgemein gesprochen, handelt es sich bei dem PI/T um einen doppelt gepufferten Baustein (FIFO). In jeder Modus- oder Portsubmodus-Kombination, in der eine doppelte Pufferung vorgesehen ist, werden die Handshake-Anschlüsse dazu benutzt, die Datentransfers zu steuern. Die Benutzung der doppelten Pufferung ist in Systemen von Vorteil, in denen ein Peripheriegerät und ein Computer in der Lage sind, Daten mit nahezu derselben Geschwindigkeit zu übertragen. In derartigen Situationen können sich die Abrufoperationen des Gerätes, das die Daten sendet, und die Speicheroperation des Empfangsgerätes überschneiden. Somit wird der in Bytes oder Worten pro Sekunde gemessene Durchsatz vergrößert. Bestehen zwischen Computer und Peripheriegerät große Unterschiede in der Transferkapazität, so kann nur wenig oder kein Vorteil erzielt werden.

Die Steuerung des Datenweges zwischen den PI/T Ports und angeschlossenen Peripheriegerät(en) kann jedes der vier Haupt-Handshakeprotokolle annehmen. Die beiden verriegelten (interlocked) und impulsgesteuerten Eingabeprotokolle ermöglichen einen "synchronisierten" Datentransfer von einem Peripheriegerät zu dem Port. Dabei zeigen die Handshake-Signale an, ob die Puffer voll bzw. leer sind. Das Port stellt doppelt gepufferte Eingabedatenwege zur Verfügung. Die Handshake-Anschlüsse werden zur Implementierung des Protokolls benutzt. **Das benutzte Handshake-Anschlußpaar hängt vom gewählten Modus und Submodus ab, wie in den Tabellen 7-5a und b dargelegt ist.** Außerdem stehen zwei Ausgabeprotokolle zur Verfügung. Auch sie sind verriegelt und impulsgesteuert. Sie übertragen Daten von dem/den PI/T-Port(s) an das Peripheriegerät, wobei ein unabhängig gesteuerter Satz intern doppelt gepufferter Datenwege benutzt wird. In den Tabellen 7-5a und b wird angegeben, welche Handshake-Anschlüsse für jeden Modus benutzt werden. Ein Impulsdiagramm für jedes Ausgabeprotokoll wird in Bild 7-15 dargestellt.

16bit-Daten können mit Hilfe der PI/T-Ports sehr wirksam zu oder von dem System übertragen werden. Bei Ausgabetransfers werden die Daten Byte für Byte vom Busmaster zuerst in das Datenregister von Port A und dann in das Datenregister von Port B geschrieben. Damit sie mit dem Datenformat des Move Peripheral-Befehls (MOVEP) des M68000 übereinstimmen, gehen die nächsten 8bit-Daten über Port A. Die Daten von Port A werden intern gehalten, bis das Datenregister von Port B geschrieben ist. Dann werden sämtliche Daten zu den Ausgangsanschlüssen der Ports A und B übertragen. So scheint das Interface zu dem Peripheriegerät eine Breite von 16 Bit zu haben. Der Betrieb der Handshake-Anschlüsse, das Löschen der Statusbits und die Generierung des

DMA-Anforderungsimpulses sind mit dem Schreiben des Datenregisters von Port B verknüpft.

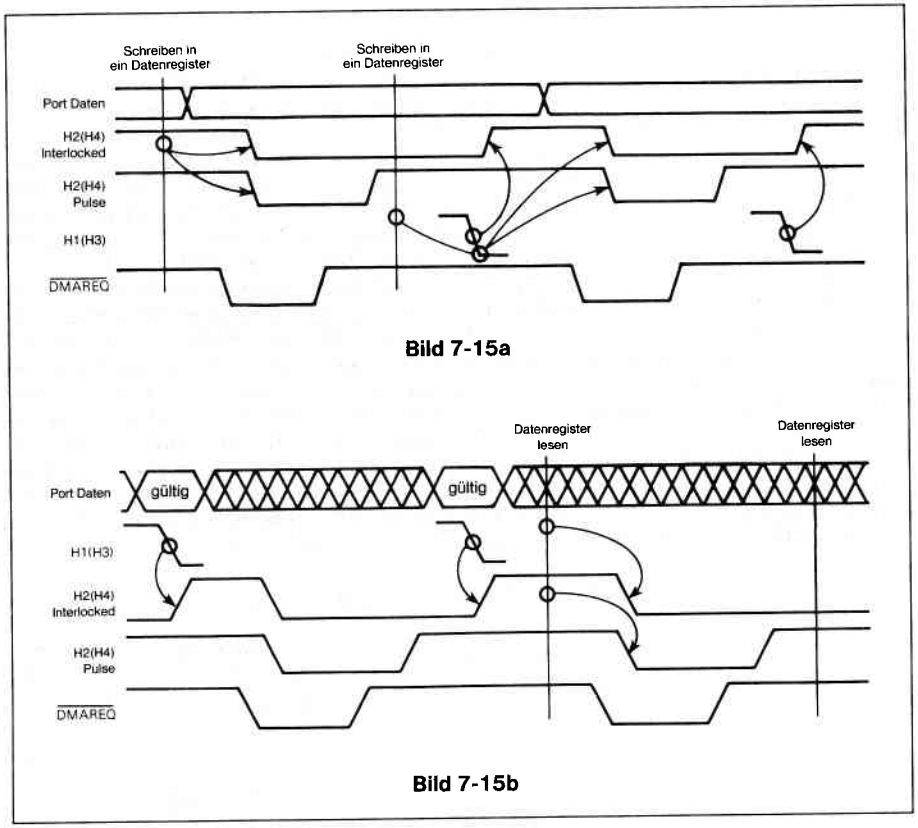


Bild 7-15a

Bild 7-15b

Bild 7-15:
Interlocked und Pulsed Transfer für doppelt gepufferte Eingangs- (a) und Ausgangsdaten (b)

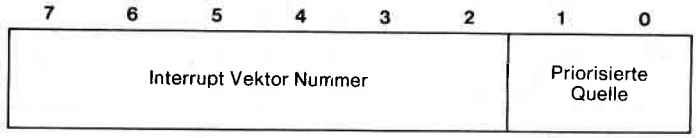
16bit-Eingangsdaten werden ähnlich gehandhabt. Sie werden asynchron mit der angesprochenen Flanke des Handshake-Anschlusses H3 verriegelt. Danach liest der Busmaster das Datenregister von Port A (MSB), worauf dann das Datenregister von Port B gelesen wird. Auch diese Daten stimmen wieder mit dem Datenformat des MOVEP-Befehls überein. Das Handshaking H4 mit dem Peripheriegerät, das Löschen des Statusbit H3 und die Generierung eines DMA-Anforderungsimpulses hängen alle vom Lesen des Datenregisters von Port B ab.

7.5.3 Service vom Busmaster

Es gibt drei Methoden, mit denen der PI/T einen Busmaster wie den M68000 oder DMAC davon unterrichten kann, daß ein Service erforderlich ist: Setzen von lesbaren Bit in ein Statusregister, Unterbrechen des Prozessors und Generierung von DMA-Anforderungen. Ein Polling des Statusregisters ist am wenigsten wirksam. Der Prozessor muß das Port-Statusregister periodisch überprüfen um zu sehen, ob irgendeine Portfunktion Service

braucht. Benötigt mehr als eine Funktion Service, so muß ein Software-Prioritätsschema (Polling) benutzt werden, um die Reihenfolge zu bestimmen, in der die Funktion behandelt werden soll.

Bei den meisten Anwendungen ist ein Polling und eine Software-Prioritätsfestlegung nicht erforderlich, wenn vektorgesteuerte Interrupts benutzt werden. Der $\overline{\text{PIRQ}}$-Anschluß (Port Interrupt Request) ist das logische ODER von aktiven Interrupts von allen vier Portquellen (H1, H2, H3 und H4). Jede Quelle muß durch Service-Anforderungs- oder Interruptaktivierungs-Bit in den Steuerregistern von Port A und B (PACR bzw. PBCR, Bild 7-14e) aktiviert werden. (Auch darf im Falle von H1 und H3 dieser Handshake-Anschluß nicht anderweitig mit DMA verknüpft werden.) Tatsächlich sammelt $\overline{\text{PIRQ}}$ jede der aktiven Interruptquellen und gibt an das System bei entsprechender Programmierung (PSRR) ein Quittierungssignal ($\overline{\text{PIACK}}$) ab (ähnliche Anschlüsse stehen bei dem Zeitgeber zur Verfügung).
Die interne Logik zur Festlegung von Interruptprioritäten ermöglicht es dem Programmierer, die Prioritäten der Port-Interruptquellen statisch festzulegen. Unter acht festgelegten Reihenfolgen kann eine ausgewählt werden (Bild 7-14c). Wird der $\overline{\text{PIACK}}$-Eingang aktiviert (Port Interrupt Acknowledge), so wird ein 8bit-Vektor auf den Datenbus ausgegeben. Der M68000 behandelt diesen wie jeden anderen Interruptvektor und geht direkt zu der Interrupt-Serviceroutine. Ein Status-Polling ist nicht erforderlich. Der Vektor wird berechnet, indem die aktivierte Interruptquelle mit der höchsten Priorität genommen wird, die auf keine andere Weise mit der DMA-Anforderungsfunktion verknüpft ist. Das Interruptvektorregister des Ports (PIVR) liefert die sechs höchstwertigen Bit des Interruptvektors, während die interne Logik zur Prioritätsfestlegung die beiden niederwertigen Bit liefert. So wird eine Tabelle mit vier Eingaben gebildet, wie folgende Zusammenstellung zeigt:



	Int.-Quelle
0 0	H1
0 1	H2
1 0	H3
1 1	H4

Es muß unbedingt beachtet werden, daß die einzige sich aus den Interrupt-Quittierungszyklen für den PI/T ergebende Auswirkung darin besteht, daß der Vektor auf den Datenbus ausgegeben wird. Insbesondere sind keine Register, Daten, Status oder andere interne Zustände des Bausteins von dem Zyklus betroffen.
Wird der PI/T in einem Autovektorsystem eingesetzt, so ist die On-Chip-Logik zur Prioritätsfestlegung nicht von Vorteil. Jedoch können die $\overline{\text{PIRQ}}$-Anschlußfunktionen, wie oben beschrieben, und der PC6/$\overline{\text{PIACK}}$-Anschluß als Ein/Ausgabeleitung von Port C benutzt werden. Ein Polling des Statusregisters kann in Autovektorsystemen erforderlich sein, die mehr als eine Interruptquelle von den PI/T-Ports benutzen. Eine andere Methode der Service-Anforderung geht über den $\overline{\text{DMAREQ}}$-Anschluß. Dieser kann in jedem Modus mit doppelt gepufferten Ein- oder Ausgabetransfers verknüpft werden. Ist er mit einem DMA-Controller verknüpft, so kann er nahezu dieselbe Wirkung auf das System haben, wie eine Interruptanforderung: bei Eingabetransfers, bei denen Daten in den doppelt gepufferten Signalspeichern stehen, die nicht von DMAC gelesen wurden, wird ein $\overline{\text{DMAREQ}}$-Impuls generiert, damit sie gelesen werden. Bei Ausgabetransfers wird, wenn einer der beiden doppelt gepufferten Signalspeicher leer ist, ein $\overline{\text{DMAREQ}}$-Impuls generiert, damit

dieser Signalspeicher aufgefüllt wird. Unter keinen Umständen generiert jedoch der $\overline{\text{DMAREQ}}$-Anschluß eine Anforderung, die bei dem DMA-Controller einen Datenverlust verursachen würde. Dieser Anschluß ist in jeder Beziehung mit dem MC68450-DMAC kompatibel.

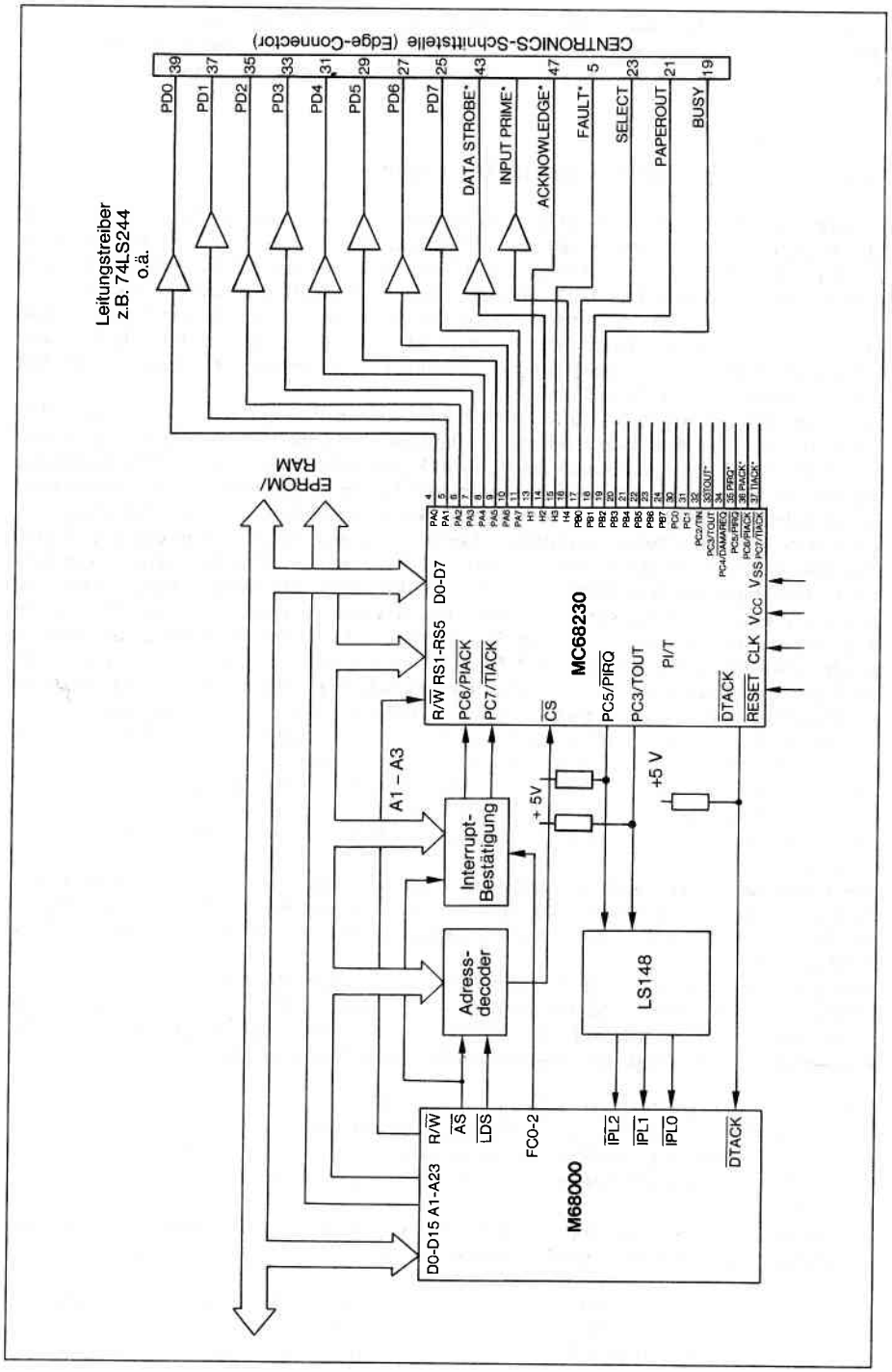
7.5.4 E/A-Programmierung und Beispiele

Es sind eine Reihe von anderen Systemeinrichtungen vorgesehen, die den PI/T noch nützlicher machen und die E/A-Programmierung vereinfachen. Erstens enthält jeder Handshake-Anschluß interne Exklusiv-ODER-Ports, mit denen der Programmierer definieren kann, ob dieser Anschluß bei Low- oder High-Pegel angesprochen wird. Diese Möglichkeit steht unabhängig vom Verwendungszweck zur Verfügung, gleichgültig ob der Anschluß als Eingangs- oder Ausgangsanschluß benutzt wird. **Diese aktive Ebene wird von den Prüfbit (aktiver Pegel) im Allgemeinen Port-Steuerregister (PGCR) gesteuert (Bild 7-14a bzw. Tab. 7-5a und b).**
Zweitens kann im Gegensatz zu vielen früheren Parallelinterface-Chips der augenblickliche Status der vier Handshake-Anschlüsse und der Ports von der Software gelesen werden (PSR, Bild 7-14a). Wird also ein Handshake-Anschluß im System nicht anderweitig genutzt, so kann er als Allzweck-Eingangsanschluß unabhängig von anderen Operationen benutzt werden (der mit ihm verknüpfte Interrupt sollte jedoch gesperrt werden).
Eine dritte nützliche Systemeinrichtung besteht in den Alternativdatenregistern (PAAR bzw. PBAR). Die Alternativregister von Port A und B können nur für das Lesen der augenblicklichen Pegel der Anschlüsse von Port A und B verwendet werden. Im Gegensatz zu den Zugriffen zu Datenregistern von Port A und B wird kein weiteres Teil des PI/T durch das Lesen oder Schreiben in diesen Registern betroffen. Dies ist in vielen Situationen nützlich. Müssen beispielsweise 7bit-ASCII-Daten von einer Tastatur oder einem anderen System empfangen werden, so kann das höchstwertige Bit des PI/T-Ports entweder als Eingangs- oder als Ausgangsanschluß benutzt werden. Wird es als Ausgangsanschluß benutzt, so stellen sich keine Probleme ein; setzen Sie einfach das Bit des Datenrichtungsregisters auf 1 (Ausgang). Fordert die Anwendung jedoch einen weiteren Eingangsanschluß und muß die Eingabe unabhängig von der Betätigung einer Taste durch den Benutzer gelesen werden, so kann die Software sie aus einem Alternativregister lesen, ohne daß das Handshaking oder die doppelte Pufferung dieses Ports dadurch beeinträchtigt werden.
Der Anwender wünscht sich oft einige Anwendungsbeispiele. Um diesem Fall Rechnung zu tragen, werden nachfolgend vier Beispiele gezeigt, die den MC68230 in verschiedenen Submodi demonstriert. Es sind Programme, die sich alle auf die Schaltung auf Seite 7-37 beziehen.
Dies stellt eine sogenannte CENTRONICS Parallel-Druckerschnittstelle dar. Der Port A ist ein Datenausgang, von Port B werden die Leitungen PB0 – PB2 zusammen mit den Handshakeleitungen H1, H2, H3 und H4 als zusätzliche Status/Steuerleitungen verwendet. **Die Programme sind Anwendungen des Modus 0 mit den Unterschieden:**

- EIN/AUSGABE ohne Interrupts, einfach gepuffert
- EIN/AUSGABE ohne Interrupts, doppelt gepuffert
- EIN/AUSGABE mit Interrupts, einfach gepuffert
- EIN/AUSGABE mit Interrupts, doppelt gepuffert

Des weiteren sollen diese Programme zeigen, wie bei der Initialisierung des Bausteines MC68230 vorzugehen ist. Empfehlenswert sind die Schritte in der Reihenfolge:

- Laden des PGCR (allgemeines Port Steuerregister)
- Laden des PSSR (Port Service Anforderungsregister)
- Laden des PADDR bzw. PBDDR (Datenrichtungsregister)

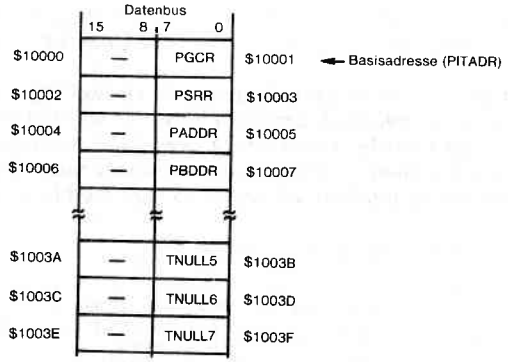


– Laden des PACR bzw. PBCR (Port Steuerregister)
und – Laden des PIVR (Port Interrupt Vektorregister)

Um das Lesen oder das Schreiben eines Registers in einem Programm zu vereinfachen bzw. sich zu merken, sollte man sich folgenden Hinweis, der generell für alle Bausteine gelten kann, merken. Die Register werden nach ihrer Reihenfolge (s. Tab. 7-15a und b) mit den jeweiligen Abkürzungen als Symbole in ein sogenanntes Offset-File (MOTOROLA-Assembler) ab Adresse 0 abgelegt. Für den MC68230 ergibt sich diese Reihenfolge:

```
*       Register-Offset Tabelle fuer den MC68230  PI/T
*
        OFFSET   0        nachfolgende Marken werden relativ
PGCR    DS.W     1        Allgemeines Port Steuerregister
PSRR    DS.W     1        Port Service-Anforderungsregister
PADDR   DS.W     1        Port A  Richtungsregister
PBDDR   DS.W     1        Port B  Richtungsregister
PCDDR   DS.W     1        Port C  Richtungsregister
PIVR    DS.W     1        Port Interrupt Vektorregister
PACR    DS.W     1        Port A  Steuerregister
PBCR    DS.W     1        Port B  Steuerregister
PADR    DS.W     1        Port A  Datenregister
PBDR    DS.W     1        Port B  Datenregister
PAAR    DS.W     1        Port A  Alternativregister
PBAR    DS.W     1        Port B  Alternativregister
PCDR    DS.W     1        Port C  Datenregister
PSR     DS.W     1        Port Statusregister
PNULL1  DS.W     1        nicht belegt
PNULL2  DS.W     1        nicht belegt
TCR     DS.W     1        Timer Steuerregister
TIVR    DS.W     1        Timer Interrupt Vektorregister
TNULL1  DS.W     1        nicht belegt
CPRH    DS.W     1        Zaehler Voreinstellungsregister (hoechstwert.)
CPRM    DS.W     1        Zaehler Voreinstellungsregister (mittleres)
CPRL    DS.W     1        Zaehler Voreinstellungsregister (niederwert.)
TNULL2  DS.W     1        nicht belegt
CNTRH   DS.W     1        Zaehler (hoechstwertiges Byte)
CNTRM   DS.W     1        Zaehler (mittleres Byte)
CNTRL   DS.W     1        Zaehler (niederwertiges Byte)
TSR     DS.W     1        Timer Statusregister
TNULL3  DS.W     1        nicht belegt
TNULL4  DS.W     1        nicht belegt
TNULL5  DS.W     1        nicht belegt
TNULL6  DS.W     1        nicht belegt
TNULL7  DS.W     1        nicht belegt
```

Abhängig von der Hardware liegen die Register entweder in den niederwertigen (ungerade Adressen) oder höherwertigen Byte eines Wortes (gerade Adressen). Bei geraden Adressen müßte UDS mit dem Chip-Select $\overline{CS}$ des PIT verknüpft sein, und die Datenleitungen des PIT müßten mit D8 – D15 des M68000 verbunden sein (Vorsicht bei Non-Autovektor-Interrupt!). Bei ungeraden Adressen ist demnach sinngemäß $\overline{LDS}$ verknüpft bzw. mit den Datenleitungen D0 – D7 verbunden. Letzteren Fall haben wir in unserem Beispiel. Die Speicherbelegung des PIT ergibt damit folgendes Bild:



Auf die verschiedenen Register hat man nun Zugriff, indem die Adressierungsart Adressregister indirekt mit Adressdistanz (Kap. 3.5.4, Bd. I) angewandt wird. Ein beliebiges Adressregister wird zuvor mit der Basisadresse des PI/T, die von der Hardware festgelegt wurde, geladen. Will man z. B. das Datenregister der Seite A lesen, so könnte dies lauten:

```
MOVE.B   PADR(A0),D1
```

Nach diesem Schema sind die Programmbeispiele realisiert. Ein weiterer Hinweis: Der MOVEP-Befehl findet häufig Anwendung, da er bei Bausteinen, die byteorientiert sind, wie der PI/T, Lesen oder Schreiben jeder zweiten Speicherstelle (gerade oder ungerade) ermöglicht. Beim MC68008 (Kap. 8.1) kann ein MOVE.L verwendet werden, wenn die Registerauswahlleitungen mit RS1 mit A0, RS2 mit A1, usw. verbunden sind!
Die strukturierten Assembleranweisungen (z. B. REPEAT .. UNTIL) ermöglichen eine klare Übersicht der Programme. Der MOTOROLA-Assembler löst diese Strukturen mit Branch- bzw. Compare-Befehlen auf.
Ein Beispiel:

```
aus                 REPEAT
                    BTST      #2,PBDR(A0)
                    UNTIL     <EQ>

wird      Z_L1.000  BTST      #2,PBDR(A0)
                    BNE       Z_L1.000
```

Es folgen die vier Softwarebeispiele. Die Programme sind im Quellcode dargestellt.

a) Ein/Ausgabe ohne Interrupts, einfach gepuffert

```
*
PITADR   EQU       $10001     PI/T-Basisadresse
*
         INCLUDE   OFF.SA              File OFF.SA soll mituebersetzt werden
         SECTION   11                  Verschiebbarer Abschnitt
START    LEA       $2000,A7            Stackpointer initialisieren
         LEA       PITADR,A0           PI/T-Basisadresse laden
         MOVE.L    #$0000FF00,D0       PGCR,PSRR=0; Port A=Ausg.,B=Eing.
         MOVEP.L   D0,PGCR(A0)         Register laden
         MOVE.W    #$A0A0,D0           Subm. 2, Int. gesperrt,H2(H4) aktiv low
         MOVEP.W   D0,PACR(A0)         PACR und PBCR laden
         MOVE.W    #$3000,D0           Modus 0; H12,H34 freigegeben, aktiv low
         MOVEP.W   D0,PGCR(A0)         PGCR und PSRR laden
         BSET      #3,PBCR(A0)         H4 = 0    Input Prime
         BCLR      #3,PBCR(A0)         H4 = 1
         REPEAT
          BTST      #2,PBDR(A0)        Warten, bis Drucker nicht mehr beschaeftigt
         UNTIL   <EQ>
         LEA       TEXT1,A5            A5 zeigt auf TEXT1
         JSR       DRUCK               Drucken des ersten Textes
         LEA       LF,A5               Drucken Zeilenvorschub,Wagenruecklauf
         LEA       TEXT2,A5            A5 zeigt auf TEXT2
         JSR       DRUCK               Drucken des zweiten Textes
         LEA       LF,A5               Drucken Zeilenvorschub,Wagenruecklauf
         TRAP      #15                 Programmabschluss, Ruecksprung ins
         DC.W      0                   Monitorprogramm, koennte auch anderer
*                                      Abschluss sein
DRUCK    REPEAT                        Druck bis Wagenruecklauf ($D)
          REPEAT
           BTST      #0,PSR(A0)        Warten auf Acknowledge
          UNTIL   <NE>
          MOVE.B    #1,PSR(A0)        loeschen H1S
          MOVE.B    (A5)+,PADR(A0)    naechstes Zeichen ins Datenregister A
          BSET      #3,PACR(A0)        H2 = 0     Datenstrobe
          BCLR      #3,PACR(A0)        H2 = 1
         UNTIL     -1(A5) <EQ> #$D
         RTS                           Ruecksprung
TEXT1    DC.B      'M68000-FAMILIE',$D
TEXT2    DC.B      'TEIL 2',$D
LF       DC.B      $A,$D
         END       START
```

b) Ein/Ausgabe ohne Interrupts, doppelt gepuffert

```
         INCLUDE   OFF.SA              File OFF.SA soll mituebersetzt werden
         SECTION   11                  Verschiebbarer Abschnitt
START    LEA       $2000,A7            Stackpointer initialisieren
         LEA       PITADR,A0           PI/T-Basisadresse laden
         MOVE.L    #$0000FF00,D0       PGCR,PSRR=0; Port A=Ausg.,B=Eing.
         MOVEP.L   D0,PGCR(A0)         Register laden
         MOVE.W    #$60A0,D0           Subm. 1, Int. gesperrt,H2(H4) aktiv low
         MOVEP.W   D0,PACR(A0)         PACR und PBCR laden
         MOVE.W    #$3000,D0           Modus 0; H12,H34 freigegeben, aktiv low
         MOVEP.W   D0,PGCR(A0)         PGCR und PSRR laden
         BSET      #3,PBCR(A0)         H4 = 0    Input Prime
         BCLR      #3,PBCR(A0)         H4 = 1
         REPEAT
          BTST      #2,PBDR(A0)        Warten, bis Drucker nicht mehr beschaeftigt
         UNTIL   <EQ>
         LEA       TEXT1,A5            A5 zeigt auf TEXT1
         JSR       DRUCK               Drucken des ersten Textes
         LEA       LF,A5               Drucken Zeilenvorschub,Wagenruecklauf
         LEA       TEXT2,A5            A5 zeigt auf TEXT2
         JSR       DRUCK               Drucken des zweiten Textes
         LEA       LF,A5               Drucken Zeilenvorschub,Wagenruecklauf
         TRAP      #15                 Programmabschluss, Ruecksprung ins
         DC.W      0                   Monitorprogramm, koennte auch anderer
*                                      Abschluss sein
DRUCK    REPEAT                        Druck bis Wagenruecklauf ($D)
          REPEAT
           BTST      #0,PSR(A0)        Warten auf Acknowledge
          UNTIL   <NE>
          MOVE.B    (A5)+,PADR(A0)    naechstes Zeichen ins Datenregister A
          BTST      #0,PSR(A0)        Warten auf Acknowledge  (H1S)
          IF <NE> AND.B -(A5) <NE> #$D THEN.S
            MOVE.B   (A5)+,PADR(A0)    Wenn Acknowledge, dann naechstes Zeichen
          ENDI
          BSET      #3,PACR(A0)        H2 = 0     Datenstrobe
          BCLR      #3,PACR(A0)        H2 = 1
         UNTIL     -1(A5) <EQ> #$D
         RTS                           Ruecksprung
```

```
TEXT1   DC.B    'M68000-FAMILIE',$D
TEXT2   DC.B    'TEIL 2',$D
LF      DC.B    $A,$D
        END     START
```

c) Ein/Ausgabe mit Interrupts, einfach gepuffert

```
        INCLUDE  OFF.SA                 File OFF.SA soll mituebersetzt werden
PRINT   MACRO
        LEA      /1,A5                  A5 zeigt auf den Textpuffer
        BSET     #1,PACR(A0)            Freigabe des H1-Interrupts
        REPEAT
         BTST     #1,PACR(A0)           Warten auf Interrupt
        UNTIL <EQ>
        ENDM
*
        ORG      $110                   Port-Interrupt Vektor Adresse
        DC.L     DRUCK                  (Vektor Nr. $44)

        SECTION  11                     Verschiebbarer Abschnitt
START   LEA      $2000,A7               Stackpointer initialisieren
        LEA      PITADR,A0              PI/T-Basisadresse laden
        MOVE.L   #$0000FF00,D0          PGCR,PSRR=0; Port A=Ausg.,B=Eing.
        MOVEP.L  D0,PGCR(A0)            Register laden
        MOVE.W   #$A0A0,D0              Subm. 2, Int. gesperrt,H2(H4) aktiv low
        MOVEP.W  D0,PACR(A0)            PACR und PBCR laden
        MOVE.W   #$3018,D0              Modus 0; H12,H34 freigegeben, aktiv low
                                        PC5=PIRQ; PC6=PIACK; H1S,H2S,H3S,H4S Pr.
        MOVEP.W  D0,PGCR(A0)            PGCR und PSRR laden
        BSET     #3,PBCR(A0)            H4 = 0    Input Prime
        BCLR     #3,PBCR(A0)            H4 = 1
        REPEAT
         BTST     #2,PBDR(A0)       Warten, bis Drucker nicht mehr beschaeftigt
        UNTIL  <EQ>
        MOVE.B   #$44,PIVR(A0)          Port Interrupt - Vektor Nr. $44
        ANDI     #$F8FF,SR              Freigabe der Interrupts (CPU)
        PRINT    TEXT1                  Drucken des ersten Textes
        PRINT    LF                     Drucken Zeilenvorschub,Wagenruecklauf
        PRINT    TEXT2                  Drucken des zweiten Textes
        PRINT    LF                     Drucken Zeilenvorschub,Wagenruecklauf
        TRAP     #15                    Programmabschluss, Ruecksprung ins
        DC.W     0                      Monitorprogramm, koennte auch anderer
*                                       Abschluss sein
DRUCK   MOVE.B   #1,PSR(A0)             Loeschen H1S
        MOVE.B   (A5)+,PADR(A0)        naechstes Zeichen ins Datenregister A
        BSET     #3,PACR(A0)            H2 = 0     Datenstrobe
        IF.B   -1(A5) <NE> #$D THEN.S
         BCLR     #3,PACR(A0)           H2 = 1
        ELSE.S
         MOVE.B   #$A0,PACR(A0)         H2 = 1 und sperren H1-Interrupts
        ENDI
        RTE                            Ruecksprung aus Interrupt-Exception
TEXT1   DC.B    'M68000-FAMILIE',$D
TEXT2   DC.B    'TEIL 2',$D
LF      DC.B    $A,$D
        END     START
```

d) Ein/Ausgabe mit Interrupts, doppelt gepuffert

```
        INCLUDE  OFF.SA                 File OFF.SA soll mituebersetzt werden
PRINT   MACRO
        LEA      /1,A5                  A5 zeigt auf den Textpuffer
        BSET     #1,PACR(A0)            Freigabe des H1-Interrupts
        REPEAT
         BTST     #1,PACR(A0)           Warten auf Interrupt
        UNTIL <EQ>
        ENDM
*
        ORG      $110                   Port-Interrupt Vektor Adresse
        DC.L     DRUCK                  (Vektor Nr. $44)
```



```
        SECTION  11                     Verschiebbarer Abschnitt
START   LEA      $2000,A7               Stackpointer initialisieren
        LEA      PITADR,A0              PI/T-Basisadresse laden
        MOVE.L   #$0000FF00,D0          PGCR,PSRR=0; Port A=Ausg.,B=Eing.
        MOVEP.L  D0,PGCR(A0)            Register laden
        MOVE.W   #$60A0,D0              Subm. 1, Int. gesperrt,H2(H4) aktiv low
        MOVEP.W  D0,PACR(A0)            PACR und PBCR laden
        MOVE.W   #$3018,D0              Modus 0; H12,H34 freigegeben, aktiv low
                                        PC5=PIRQ; PC6=PIACK; H1S,H2S,H3S,H4S Pr.
        MOVEP.W  D0,PGCR(A0)            PGCR und PSRR laden
        BSET     #3,PBCR(A0)            H4 = 0    Input Prime
        BCLR     #3,PBCR(A0)            H4 = 1
        REPEAT
         BTST     #2,PBDR(A0)       Warten, bis Drucker nicht mehr beschaeftigt
        UNTIL  <EQ>
        MOVE.B   #$44,PIVR(A0)          Port Interrupt - Vektor Nr. $44
        ANDI     #$F8FF,SR              Freigabe der Interrupts (CPU)
        PRINT    TEXT1                  Drucken des ersten Textes
        PRINT    LF                     Drucken Zeilenvorschub,Wagenruecklauf
        PRINT    TEXT2                  Drucken des zweiten Textes
        PRINT    LF                     Drucken Zeilenvorschub,Wagenruecklauf
        TRAP     #15                    Programmabschluss, Ruecksprung ins
        DC.W     0                      Monitorprogramm, koennte auch anderer
*                                       Abschluss sein
DRUCK   MOVE.B   (A5)+,PADR(A0)        naechstes Zeichen ins Datenregister A
        BSET     #0,PSR(A0)            Setze und Teste H1S
        IF <NE> AND.B -1(A5) <NE> #$D THEN.S
         MOVE.B   (A5)+,PADR(A0)       naechstes Zeichen ins Datenregister A
        ENDI
         BSET     #3,PACR(A0)           H2 = 0
        IF.B -1(A5) <NE> #$D  THEN.S
         BCLR     #3,PACR(A0)           H2 = 1
        ELSE.S
         MOVE.B   #$60,PACR(A0)         H2 = 1 und sperren H1-Interrupts
        ENDI
        RTE                             Ruecksprung aus Interrupt-Exception
TEXT1   DC.B    'M68000-FAMILIE',$D
TEXT2   DC.B    'TEIL 2',$D
LF      DC.B    $A,$D
        END     START
```

Diese Programme können nur einen Teil der Möglichkeiten aufzeigen, die dieser Baustein bietet. Dennoch sollen die Beispiele ein Schema vorgeben, das z. B. auch in anderen Modi angewandt werden kann. Die Initialisierungsroutinen sind bei allen Bausteinen sehr ähnlich.

7.5.5 Betriebssystemorientierter Zeitgeber

Der PI/T-Zeitgeber wurde in Hinblick auf verschiedene Eigenschaften konstruiert, die in Betriebssystemen von Hochleistungs-Mikroprozessoren und -Minicomputern häufig benötigt werden. Aufgrund eines periodischen Interrupts kann der Prozessor eine Uhr aufbauen, Tasks umschalten oder andere gebräuchliche Funktionen ausführen. Bei Anwendungen, in denen ein Betriebssystem auf ein Teil der E/A-Hardware wartet, wie es an die PI/T-Ports angeschlossen werden kann, kann der Zeitgeber so programmiert werden, daß er einen Interrupt generiert, wenn ein bestimmtes Ereignis nicht innerhalb einer programmierten Zeitspanne eingetreten ist (die Port- und Zeitgeberfunktionen sind jedoch vollständig unabhängig voneinander). Der Zeitgeber kann eine Rechteckspannung generieren, die für einen Allzweck-Taktausgang, wie z. B. einen Baudrate-Generator, von Nutzen ist. Prozeßsteuerungsanwendungen erfordern oft genaue Messungen der abgelaufenen Zeit. Dies wird erreicht, indem der Zeitgeber bei einem bekannten Wert gestartet und dann abgelesen wird, wenn das Ereignis eingetreten ist. Die Differenz zwischen dem ursprünglichen und dem endgültigen Wert ergibt die abgelaufene Zeit. Viele Hochleistungssysteme benötigen ein Gerät oder einen Busmonitor zur Anzeige eines nicht programmierten Stops bei einem Interface. Der Zeitgeber kann eine obere Grenze

in dem Zeitintervall feststellen, während der TIN-Eingang auf high liegt. Jede dieser Anwendungen wird nachstehend erklärt, nachdem wir eine kurze Einführung in den Betrieb des Zeitgebers gegeben haben.

Der Zeitgeber enthält einen synchronen 24bit-Abwärtszähler. Er wird von drei 8bit-Zähler-Preloadregistern (CPR) initialisiert und kann, wenn er sich im Ruhezustand befindet, vom Prozessor von drei Nur-Lese-Zählregistern aus gelesen werden. Ein optioneller 5bit- (Dividiere-durch-32-) Vorteiler ist ebenfalls vorhanden. Der Zeitgeber kann auf verschiedene Arten getaktet werden. Wird der Vorteiler benutzt, so kann der externe Zeitgebereingang TIN oder der Anschluß CLK von PI/T dazu benutzt werden, den Vorteiler zu takten. Der Vorteiler wiederum taktet den 24bit-Zähler. Wird der Vorteiler nicht benutzt, so wäre die Frequenz an dem CLK-Anschluß so hoch (typischerweise 6 bis 10 MHz), daß der 24bit-Zähler nicht betriebsfähig wäre. Somit muß der TIN-Eingangsanschluß benutzt werden, um den Zeitgeber auf eine niedrigere Frequenz zu takten, wenn der Vorteiler nicht benutzt wird.

In allen Anwendungen signalisiert der Zeitgeber das Eintreten eines Ereignisses, wenn er entdeckt, daß der Zähler auf Null dekrementiert ist. Dies wird im Zeitgeber-Statusregister und über den Zeitgeber-Ausgangsanschluß TOUT reflektiert. Das Zeitgeber-Statusregister enthält ein einziges Bit, das ZDS-Bit (Zero Detect Status-Bit). Es besteht aus einem Flip-Flop, das gesetzt wird, wenn der Zähler auf Null dekrementiert wird. Dieses Bit kann immer von dem Prozessor gelesen oder benutzt werden, um eine Interruptanforderung des Zeitgebers zu generieren.

Der Zeitgeber hat nur einen Ausgang, TOUT. Er wird als Interruptanforderung des Zeitgebers, als Rechteckausgang oder als Zeitüberwachungsausgang benutzt. **In der Interruptanforderungs- oder der Zeitüberwachungsfunktion ist TOUT ein Open-Drain-Ausgang, der gesetzt wird (logisch Null), wenn das ZDS-Statusbit gleich 1 und die Interrupt- oder Zeitüberwachungsfunktion aktiviert ist.** Bei der Generierung einer Rechteckspannung ist TOUT ein Push-Pull-Ausgang, der immer dann umgeschaltet wird, wenn der Zähler den Zählerstand Null erreicht.

Der $\overline{\text{TIACK}}$-Anschluß wird immer nur für den Interrupt-Quittierungseingang des Zeitgebers benutzt. Wird der $\overline{\text{TIACK}}$ aufgerufen, so antwortet der PI/T, indem er den Inhalt des Zeitgeber-Interruptvektorregisters auf den Datenbus ausgibt und $\overline{\text{DTACK}}$ (Data Transfer Acknowledge) aktiviert. Kein anderer Teil des PI/T wird von dem Interrupt-Quittierungs-Buszyklus des Zeitgebers betroffen. Es muß unbedingt beachtet werden, daß der $\overline{\text{TIACK}}$-Anschluß nur benötigt wird, wenn der TOUT-Anschluß für eine Interrupt-Anforderungsoperation in einem System programmiert ist, das vektorgesteuerte Interrupts unterstützt. Ansonsten kann er als anderer Allzweckanschluß von Port C verwendet werden.

Der PI/T-Zeitgeber wird vollständig durch ein einziges 8bit-Timer-Control-Register (TCR) konfiguriert und gesteuert (siehe Bild 7-14b und 7-16a). Es enthält Bit, die den Zweck jedes Zeitgeberanschlusses definieren, des weiteren bestimmen, welche Operation stattfindet, wenn der Zähler den Stand Null erreicht, die Frequenzquelle des Zählers wählen und den Zeitgeber aktivieren oder den TIN-Anschluß mit dieser Aktivierung beauftragen. Insbesondere wird jede Zeitgeberanwendung durch die Programmierung der einzelnen Bit des Zeitgeber-Steuerregisters definiert (siehe Bild 7-16a).

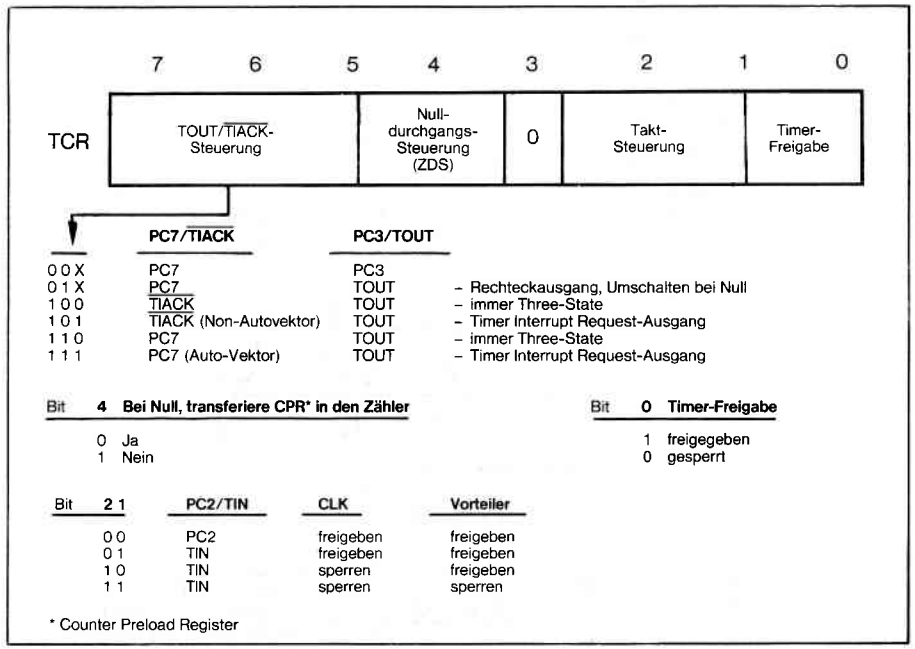
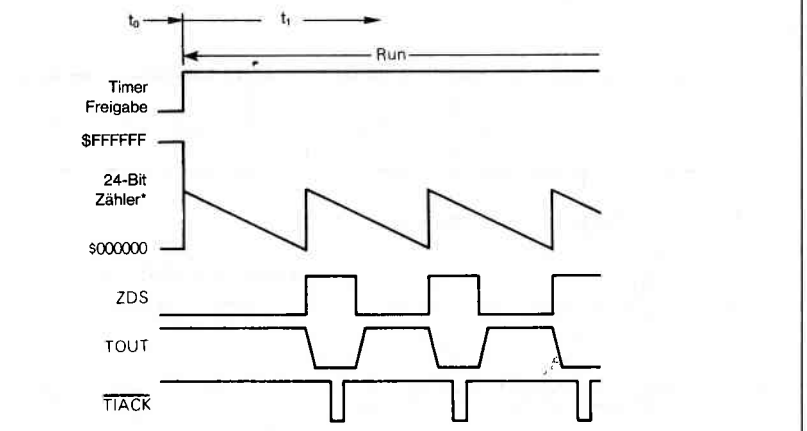


Bild 7-16a: Timer-Steuerregister (TCR)

Der Betrieb des Zeitgebers kann in Form von Run- (Betriebs-) und Haltzuständen beschrieben werden. Befindet sich der Zeitgeber im Betriebszustand, so wird der Zähler durch eine programmierte Frequenzquelle getaktet. Wenn der 24bit-Zähler auf Null dekrementiert, wird das ZDS-Statusbit auf 1 gesetzt. Bei vielen Anwendungen braucht der Prozessor den Zähler nicht zu lesen, während er im Betriebszustand ist. So sind für diesen Zweck keine Signalspeicher vorgesehen. Im Haltezustand wird der Zähler eingefroren. Er kann über die drei Nur-Lese-Zählregister gelesen werden. Als Vorbereitung für den kommenden Betriebszustand werden alle Bit des Vorteilers auf Eins gesetzt, und das ZDS-Statusbit wird gelöscht.

Am besten versteht man die Funktionsweise des Zeitgebers, wenn man einige typische Betriebssystem-Anwendungen betrachtet. Periodische Interrupts sind in nahezu jedem Betriebssystem erforderlich. Der Zeitgeber kann vektorgesteuerte oder autovektorgesteuerte Interrupts generieren. Der TOUT-Anschluß wird als Interruptanforderung des Zeitgebers in der Interruptstruktur des Systems angeschlossen. Der $\overline{\text{TIACK}}$-Anschluß dient als Interruptquittierung des Zeitgebers in vektorgesteuerten Interruptsystemen, wenn Bit 6 im TCR = 0 ist. Der TIN-Anschluß ist wahlweise und kann als externer Takteingang benutzt werden, wenn die Taktfrequenz des Systems nicht paßt oder nicht ausreichend stabil ist. Der gewünschte Teiler wird in den Zähler-Preloadregistern programmiert (24 Bit). **Ein Impulsdiagramm, (b) und die Programmierung des Timer Control-Registers TCR (c), werden in Bild 7-16 dargestellt.** Der Timer ist in diesem Beispiel als Generator für periodische Interrupts programmiert.

Wird der Zeitgeber aktiviert, so wird der Inhalt der Zähler-Preloadregister in den 24bit-Zähler übertragen. Der Zähler dekrementiert mit jedem Taktimpuls, der von dem Vorteiler oder dem TIN-Anschluß stammen kann. Die Aktion, die eintreten soll, wenn der Zähler den Zählerstand Null erreicht, kann programmiert werden: Der Inhalt der Zähler-Preloadregister kann wieder in den Zähler geladen werden, oder der Zähler kann durch Null laufen und mit dem Zählen fortfahren.



*Analoge Darstellung des Zählers

Bild 7-16b

	7	6	5	4	3	2	1	0
TCR	TOUT/TIACK Steuerung			Z.D. Steuerung	*	Clock Steuerung		Timer Freigabe
	1	X	1	0	0	00 oder 1X		geändert

Bit 6 = "1" Auto-Vektor
Bit 6 = "0" Non-Autovektor

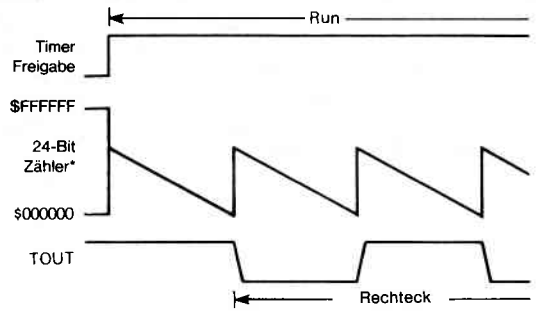
während t_0 = "0"
während t_1 = "1"

Bild 7-16c

Bild 7-16b, c: Generator für periodische Interrupts (b) und Programmierung des Timer Control-Registers TCR (c)

In dieser Anordnung soll der Zähler neu geladen werden. Ist ein Zählerstand Null erreicht, so wird das ZDS-Statusbit gesetzt. Wird der Zeitgeber-Interrupt aktiviert, so wird der Interruptanforderungsausgang (TOUT) angesprochen. Das Statusbit kann jederzeit vom Prozessor gelesen werden. In vektorgesteuerten Interruptsystemen ist dies jedoch nicht erforderlich. Während des Interruptquittierungszyklus liefert der PI/T einen programmierbaren Vektor, der den Prozessor direkt zu der entsprechenden Interrupt-Serviceroutine schickt. In dieser Anwendung muß also das Zeitgeber-Statusbit nie vom Prozessor gelesen werden. Wie bei anderen M68000-Peripheriebausteinen (mit wenigen Ausnahmen) muß der Prozessor die Ursache des laufenden Interrupts löschen, damit die Interruptanforderung gelöscht wird. Bei dem PI/T-Zeitgeber wird dies erreicht, indem eine "1" in das Zeitgeber-Statusregister geschrieben wird, die das ZDS-Bit löscht.

Auf ähnliche Weise kann der Zeitgeber eine Rechteckspannung ausgeben. Anstatt daß der TOUT-Anschluß angesprochen wird, wenn der Zeitgeberinterrupt aktiviert wird und das ZDS-Bit gleich "1" ist, wird nun der TOUT-Anschluß zu einem Push-Pull-Ausgang, der immer dann umgeschaltet wird, wenn Null erkannt wird, wodurch eine Allzweck-Rechteckspannung generiert wird. Wie vorher wird der Zeitgeber von den Zähler-Preloadregistern neu geladen und fährt mit dem Zählen fort. Der TIACK-Anschluß wird nicht benutzt, und der TIN-Anschluß kann als Takteingang benutzt werden. Der Vorteiler ist wahlweise. **Ein Impulsdiagramm (a) und die Programmierung des TCR (b) werden in Bild 7-17 dargestellt.**



*Analoge Darstellung des Zählers

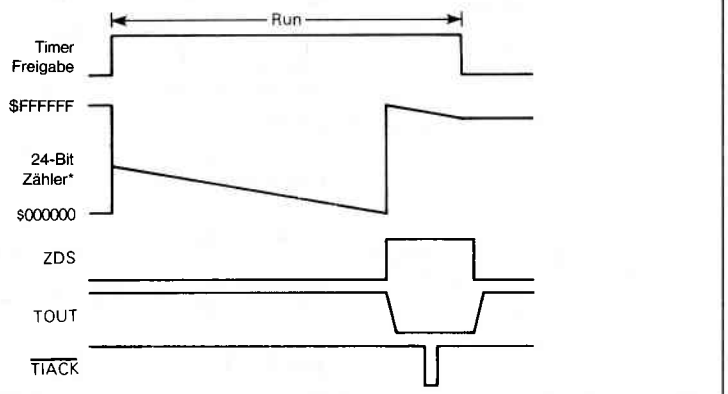
Bild 7-17a

	7	6	5	4	3	2	1	0
TCR	TOUT/TIACK Steuerung			Z.D. Steuerung	*	Clock Steuerung		Timer Freigabe
	0	X	1	0	0	00 oder 1X		geändert

Bild 7-17b

Bild 7-17: Impulsdiagramm (a) und Programmierung (b) des TCR

Der Zeitgeber kann auch einen "einzigen" Interrupt nach einer programmierten Zeitspanne liefern. Diese Betriebsart ist identisch mit dem periodischen Interruptgenerator; nur wenn der Zähler Null erreicht, werden alle Bit auf Eins gesetzt, und der Zähler fährt mit dem Zählen fort. Wenn der Prozessor den Interrupt abarbeitet, löscht er das Zeitgeber-Freigabebit, hält den Zeitgeber an und friert den Zähler ein. Er kann dann die Zählregister lesen, um die Interruptwartezeit zu bestimmen, falls dies in der Anwendung wichtig ist. **Ein Impulsdiagramm wird in Bild 7-18 dargestellt.**



*Analoge Darstellung des Zählers

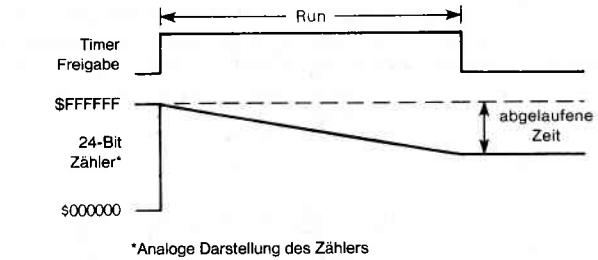
Bild 7-18a

	7	6	5	4	3	2	1	0
TCR	TOUT/$\overline{\text{TIACK}}$ Steuerung			Z.D. Steuerung	*	Clock Steuerung		Timer Freigabe
	1	X	1	1	0	00 oder 1X		geändert

Bild 7-18b

Bild 7-18: Einzelner Interrupt (a) und TCR (b)

Das Messen der abgelaufenen Zeit ist einfach und erfordert gegebenenfalls nur den externen Takteingang TIN. Ein bekannter Wert, wie beispielsweise alle Bit auf Eins, wird in die Zähler-Preloadregister geschrieben, und der Zähler wird so programmiert, daß er umspringt, wenn Null erreicht ist. (Dies dauert bei Benutzung des Vorteilers 67 Sekunden bei 8 MHz.) Der Zähler beginnt mit dem Dekrementieren, wenn der Zeitgeber aktiviert wird. Wenn der Zähler gestoppt werden muß, löscht die Software das Zeitgeber-Freigabebit in dem Zeitgeber-Steuerregister. Der Zähler kann dann über die Zählerregister gelesen werden, indem gegebenenfalls der MOVEP.L-Befehl benutzt wird, wodurch die abgelaufene Zeit angegeben wird. Ein Impulsdiagramm (a) und die dazu programmierte Bitkombination werden in Bild 7-19 des TCR (b) dargestellt.



*Analoge Darstellung des Zählers

Bild 7-19a

	7	6	5	4	3	2	1	0
TCR	TOUT/$\overline{\text{TIACK}}$ Steuerung			Z.D. Steuerung	*	Clock Steuerung		Timer Freigabe
	0	0	X	1	0	0	0	geändert

Bild 7-19b

Bild 7-19: Messung der abgelaufenen Zeit (a) und TCR (b)

Der letzte große Anwendungsbereich des Zeitgebers liegt in der Überwachung von "Hang-ups" (Watchdog) bei Geräten. Anstatt den TIN-Anschluß als Takteingang zu benutzen, wird er an das betreffende Gerät angeschlossen. Ein "Timeout" (z. B. Generierung des $\overline{BERR}$-Signales) wird dann erzeugt, wenn vom Voreinstellwert heruntergezählt wurde, ohne daß das TIN-Signal in dieser Zeit inaktiviert (log. "0") worden ist. Der TOUT-Anschluß wird ähnlich wie bei den Interruptanforderungs-Anwendungen benutzt, kann jedoch an andere Hardwareelemente angeschlossen werden (z. B. $\overline{BERR}$). Er gibt an, wenn der TIN-Anschluß zu lange auf "1" gehalten wurde. Befindet sich der TIN-Anschluß auf "1", so ist der Zeitgeber im Betriebsstatus und dekrementiert, bis der Zähler den Zählerstand Null erreicht hat. Wird er jedoch "0" gehalten, bevor der Zähler Null erreicht, so hält der Zeitgeber an, der TOUT bleibt negiert. Ansonsten wird TOUT aktiviert. Es ist keine Reinitialisierung seitens des Prozessors notwendig, da dies der Timer mit der ansteigenden Flanke von TIN selbst erledigt. Wird TIN auf "1" gesetzt, so wird der Zeitgeber gesperrt, das ZDS-Statusbit wird auf Null gesetzt und TOUT wird automatisch negiert. **Ein Impulsdiagramm (a) und die programmierte Bitkombination des TCR (b) werden in Bild 7-20 dargestellt.**

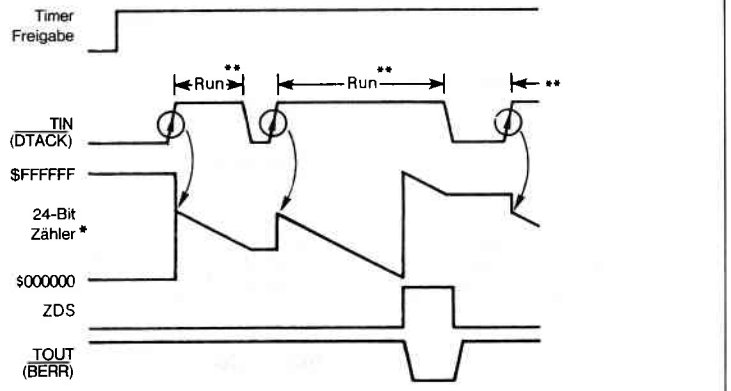


Bild 7-20a

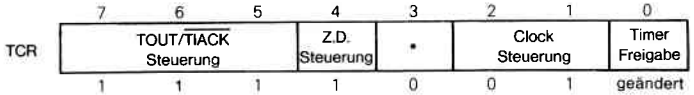


Bild 7-20b

Bild 7-20: Impulsdiagramm (a) und programmierte Bitkombination (b) des TCR für einen "Watchdog"-Timer.

7.6 DMA-Bausteine

DMA ist eine Abkürzung für Direct Memory Access, das heißt: direkter Speicherzugriff. Im folgenden wird das Kürzel DMA noch häufiger verwendet. Was ist damit gemeint? Kenner der Materie und damit auch viele Leser dieses Buches wissen die Antwort auf diese Frage. Es gibt dennoch eine Reihe von Anwendern, die noch keine Erfahrung mit dieser Technik haben oder sich nach wie vor scheuen, aus welchen Gründen auch immer, das DMA-Konzept einzuführen. Und dies, obwohl sie mit dem Umgang von Mikroprozessorsystemen vertraut sind. Für diesen Leserkreis soll hier eine Einführung in das DMA-Konzept gegeben werden.
Wozu dient ein direkter Speicherzugriff? Allgemein gesagt, ist ein DMA-Controller eine Prozesseinheit, die für eine spezielle Klasse von Applikationen optimiert ist, nämlich für Datenverschiebungen zwischen Speicher und Peripheriebausteinen. An die Peripheriebausteine können die verschiedenartigsten Geräte, meist sehr datenintensiv, angeschlossen sein. Beispiele hierfür sind: Disk-Speicher, Magnetbänder, Video-Schnittstellen, Bildschirme, Drucker und andere Kommunikationsmittel wie lokale Netzwerke, IEC-Bus, UARTs, usw.
Wie ein Mikroprozessor ist auch ein DMA-Baustein in der Lage, den Adress- und Steuerbus zu liefern. Der Unterschied liegt darin, daß keine Daten zwischengespeichert werden müssen, wodurch wesentliche höhere Übertragungsraten erreicht werden als bei einer CPU.

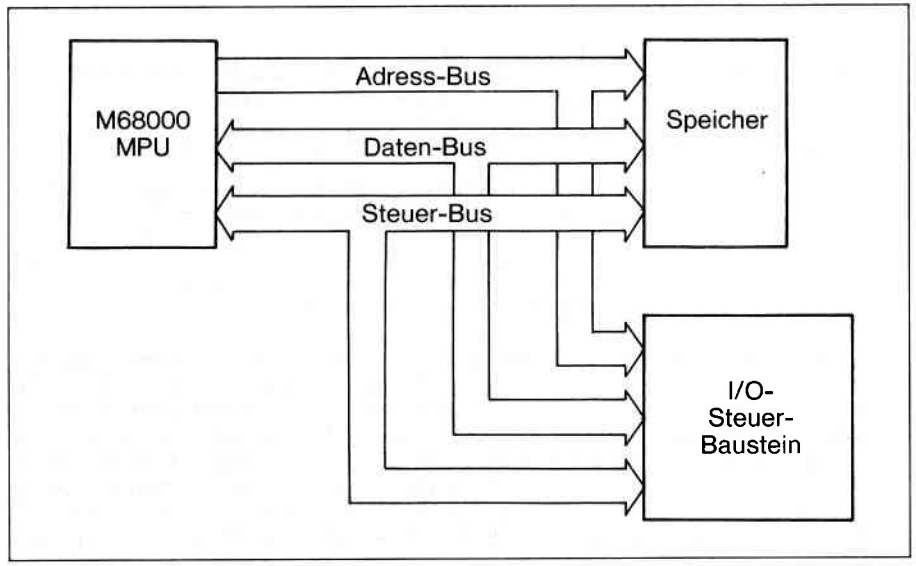


Bild 7-21: Standard-Mikroprozessorsystem

Betrachten wir ein Standard-Mikroprozessorsystem, wie es in Bild 7-21 dargestellt ist. Daten von der Peripherie zum Speicher oder umgekehrt müssen grundsätzlich den (Um-) Weg über den Mikroprozessor machen. D. h. Daten werden von der Peripherie mit einem normalen Lesezyklus in ein Register der MPU geladen und anschließend mit einem Schreibzyklus in den Speicher geschrieben. Sind große Datenmengen zu transferieren, wie z. B. von einem Plattenspeicher, dann multipliziert sich die Übertragungszeit sofort um ein Vielfaches. Man hat ein Prinzip gefunden, um den Mikroprozessor bei einer Datenüber-