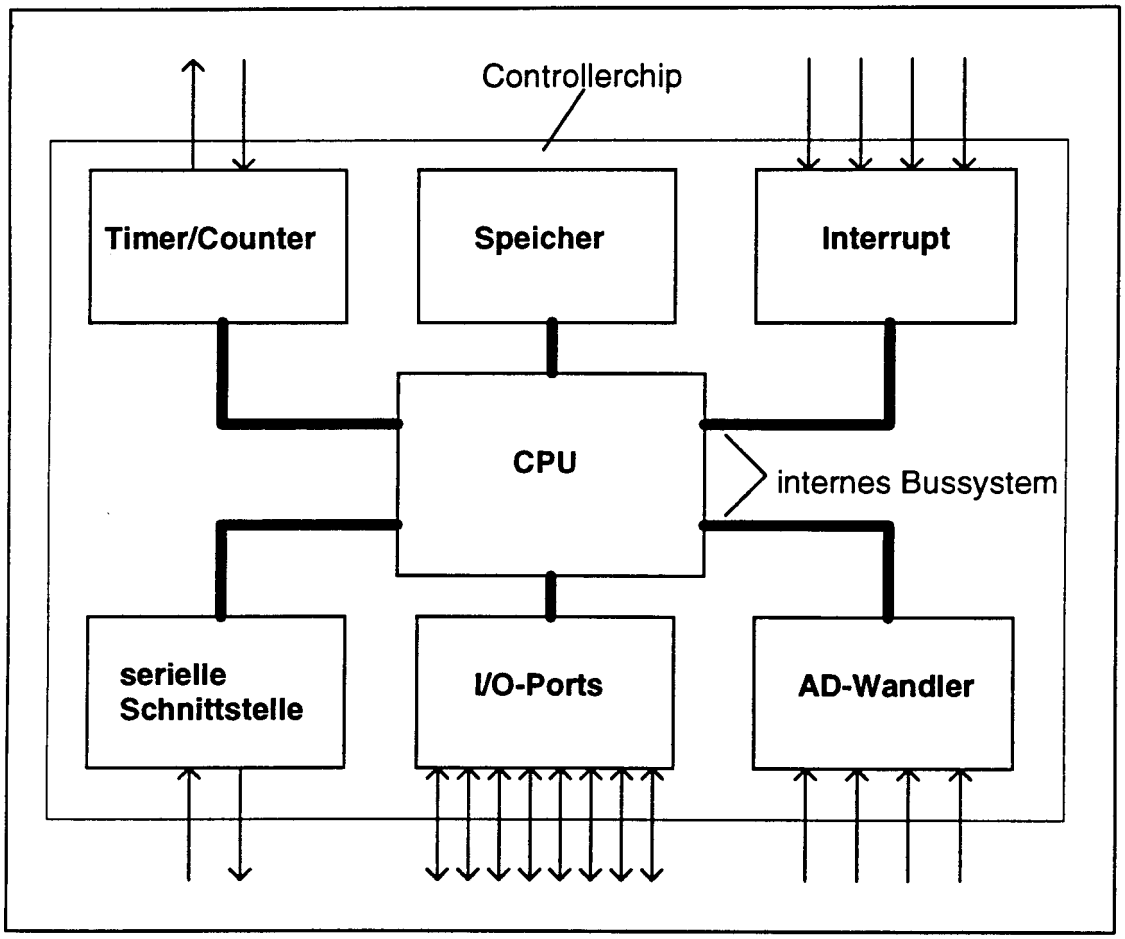
Einführung in das Mikrocontroller-System 80(C)515/80(C)535



Inhalt Seite

Quellenangaben:

Microcomputer Components
SAB 80515 / SAB 80C515
SAB 80C515 / SAB 80C535
SAB 80515 / SAB 80525
8-Bit Single-Chip Microcontroller Family
Siemens Datenbuch 08.95
Siemens AG, München

Mikrocomputertechnik I
Alexander Götz, Peter von Viebahn
Vorlesungsmanuskript der Berufsakademie Stuttgart
Fachbereich Elektrotechnik

MC-TOOLS 5
Handbuch des 80C517 und 80C517A
R. Johannis, N. Papadopoulos
Feger + Co. Hard- und Software Verlags OHG, Traunreut

Eine Bemerkung des Autors

Die vorliegende Ausarbeitung zu den Mikrocontrollern 80(C)515/-535 entstand aus einer Notlage heraus. Mir war es trotz intensiver Suche nicht möglich, eine kompakte deutschsprachige Beschreibung der Controller zu finden.

Da nach meiner Erfahrung und aus pädagogischen Gründen eine Arbeitsunterlage - und so will ich das vorliegende Manuskript auch verstanden wissen - zur Ausbildung von Studierenden der Berufsakademie und in der gewerblichen Ausbildung nur in Ausnahmefällen fremdsprachlich sein darf, habe ich mich entschlossen, dieses Manuskript zusammenzustellen. Zu vermittelnder Inhalt ist der Controller, seine Möglichkeiten und die Funktionsweise, und erst nachgeordnet das Lesen und Verstehen fremdsprachlicher Texte. Es ist nicht als Ersatz für das Datenblatt oder das Handbuch gedacht. Für mich bilden diese drei eine Einheit.

im Juni 1998

1 Einführung

Der SAB 80C515/80C535 ist ein Mitglied der SAB 8051-Familie der Acht-Bit-Mikrocontroller. Der 80C515/80C535 ist in ACMOS-Technik aufgebaut und voll funktionskompatibel mit dem in MYMOS-Technik aufgebauten 80515/80535.

In vielen Veröffentlichungen werden die Technologienamen synonym für die Controller verwendet. Dabei bedeutet „ACMOS-Version“ den 80C515/80C535 und „MYMOS-Version“ steht für den 80515/80535. Beides sind Einchipcontroller die auf dem 8051 aufbauen. Der einzige Unterschied zwischen den sonst identischen Controllern 80(C)535 und 80(C)515 ist, daß der 80(C)515 ein 8Kx8-ROM enthält.

Also:

- 80C515: ACMOS-Technologie mit 8-Kbyte maskenprogrammierbarem ROM
- 80C535: ACMOS-Technologie ohne ROM
- 80515: MYMOS-Technologie mit 8-Kbyte maskenprogrammierbarem ROM
- 80535: MYMOS-Technologie ohne ROM

Leistungsmerkmale:

- 8-Kbyte ROM eingebaut (nur -515)
- 256 Byte eingebauter RAM-Speicher
- sechs 8-Bit Ein-/Ausgabe Ports
- ein Eingangsport für digitale Signale (bei den ACMOS-Versionen)
- Serieller Port, Vollduplex mit vier Betriebsarten, mit festen oder variablen Baudraten
- drei 16-Bit Zähler/Timer
- Capture and Compare-Einheit mit 16-Bit Registern
- A/D-Umsetzer mit programmierbaren Referenzspannungen
- acht gemultiplexte Analogeingänge
- 16-Bit Watchdog-Timer
- Power-down-Versorgung für 40 Bytes im RAM
- Bool’sches Rechenwerk
- 256 direkt adressierbare Bits
- 12 getrennte Interruptquellen (7 externe, 5 interne), 4 Prioritätsebenen
- Stackpointer mit bis zu 256 Byte Tiefe

- 1 µs Befehlszykluszeit bei 12 MHz Taktversorgung
- Externer Programm- und Datenspeicher mit bis zu 64 KByte jeweils
- Kompatibel mit 8080/8085-Peripherie- und Speicherbausteinen

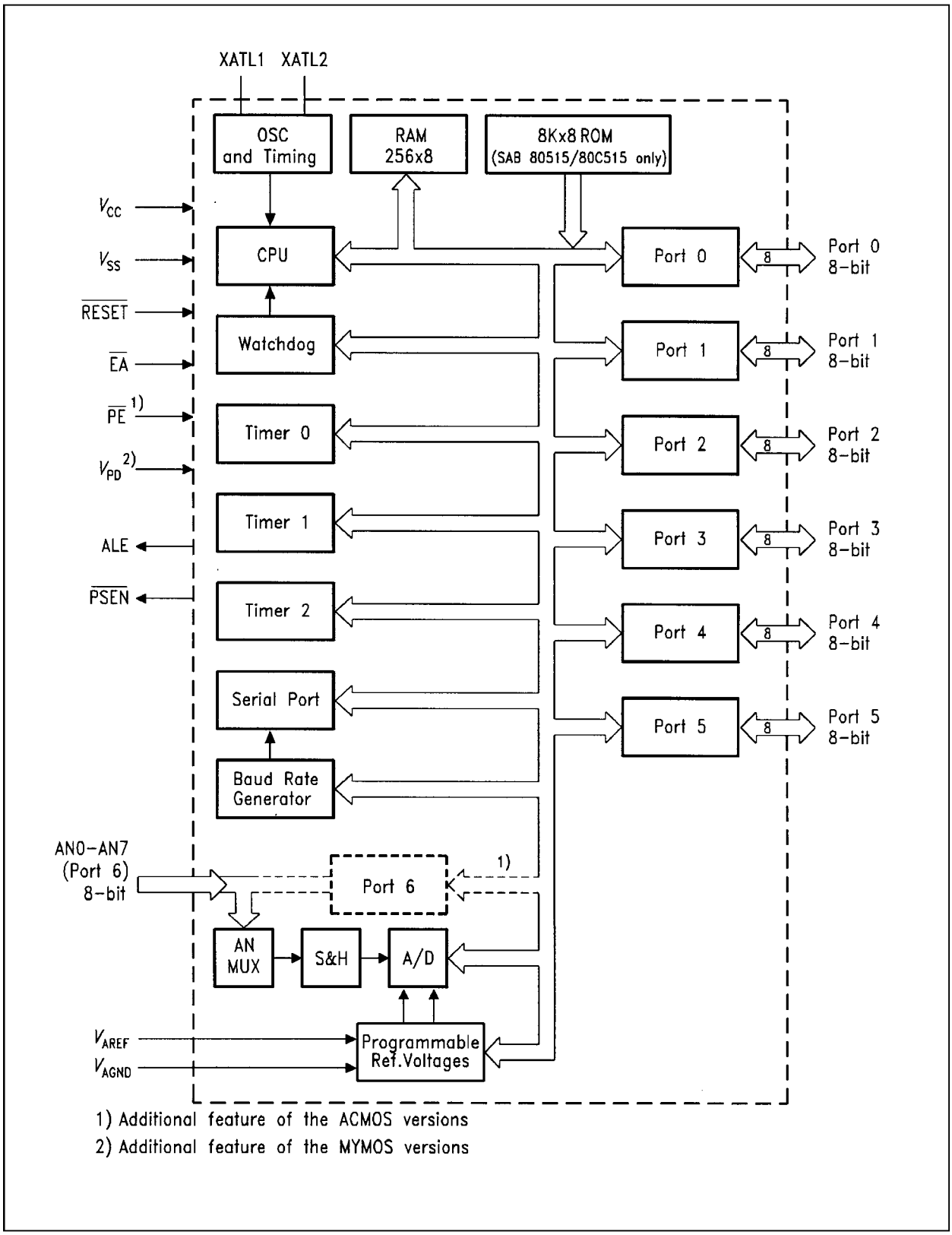


Bild 1.1: Blockschaltbild des 80C515/80C535

2 Grundzüge der Architektur

Der 80(C)515/-535 ist ein zum 8051 vollständig kompatibler Mikrokontroller, obwohl seine zusätzlichen Leistungsmerkmale im Vergleich zu diesem deutlich angestiegen sind. Einige der zahlreichen Zusatzmerkmale wurden integriert, um die steigenden Anforderungen an die Leistungsfähigkeit des 8-Bit-Kerns zu erfüllen, ohne jedoch die Kompatibilität zum 8051 zu verlieren. Darüber hinaus enthält der 80(C)515/-535 beispielsweise einen 8-Bit Analog-Digital-Umsetzer, zwei Zeitgeber, ROM und RAM wie vom 8051 bekannt und einen weiteren Zeitgeber mit Capture/Compare/Reload-Möglichkeiten für digitale Signalverarbeitung. Der 80C515/-535 vereint die Architektur eines Industrie-Standardcontrollers mit den Vorteilen der ACMOS-Technologie.

2.1 Die CPU

Der Rechenkern (CPU) kann Bits und Bytes verarbeiten. Die Befehle, die aus bis zu drei Bytes bestehen, benötigen einen, zwei oder vier Maschinenzyklen, wobei ein Maschinenzyklus zwölf Taktperioden des Oszillators lang ist. Der Befehlssatz des 80(C)515/-535 bietet hinreichend Möglichkeiten zum Datentransfer, zu logischen oder arithmetischen Operationen; auch das boolsche Rechenwerk hat eigene Befehle im Befehlssatz. Der 80(C)515/-535 verfügt über fünf Adressierungsarten:

- Direkte Adessierung (direct)
- Registeradressierung (register)
- Unmittelbare Adressierung (immediate)
- Indirekte Registeradressierung (register indirect)
- Indirekte indizierte Registeradressierung (base register plus index register indirect)

2.2 Die Speicherorganisation

Der 80(C)515 enthält 8 KByte ROM. Der Programmspeicher kann extern bis 64 KByte erweitert werden. Das interne RAM umfaßt 256 Byte. Innerhalb dieses Adressenbereiches liegen 128 bitadressierbare Speicherzellen und vier getrennte Registerbänke mit je acht Universalregistern. Zusätzlich zu diesem RAM gibt es einen Speicherbereich mit 128 sogenannten „special function registers“ (SFR). Auf Grund seiner Harvard-Struktur unterscheidet der 80(C)515/-535 zwischen externem Programm- und externem Datenspeicher. Beide können bis 64 KByte tief sein.

2.3 Die externe Buserweiterung

Die Buserweiterung des 80(C)515/-535 verfügt über einen 8-Bit Datenbus (Port 0), eine 16-Bit Adressenbus (Port 0 und Port 2) sowie fünf Steuerleitungen. Das Signal *ALE* (address latch

enable) dient zum Demultiplexen des Adreß- und Datenbusses. Der externe Programmspeicher wird durch das Signal $\overline{PSEN}$ (programm store enable) zweimal pro Maschinenzyklus angesprochen. Der Anschluß $\overline{EA}$ (external access) informiert den Microcontroller darüber, ob er außerhalb des internen ROM's arbeiten soll oder innerhalb. Die Anschlüsse $\overline{RD}$ (read) oder $\overline{WR}$ (write) dienen zum Ansprechen des externen Datenspeichers.

2.4 Die integrierten Peripheriekomponenten

Alle integrierten Peripheriekomponenten wie Ein-/Ausgabeports, serielle Schnittstelle, A/D-Umsetzer usw. werden über die zugehörigen Special Function Register bedient. Dieses Zugriffskonzept erlaubt eine hohe Flexibilität auch für weitere Erweiterungen.

2.5 Das Interruptsystem

Eine besondere Stellung nimmt das Interruptsystem des 80(C)515/-535 ein. Es ist unterteilt in zwei Gruppen von Anforderungen: sieben externe und fünf interne stehen dem Anwender zur Verfügung. Darüber hinaus können die Interrupts in vier Prioritätsstufen eingeordnet werden.

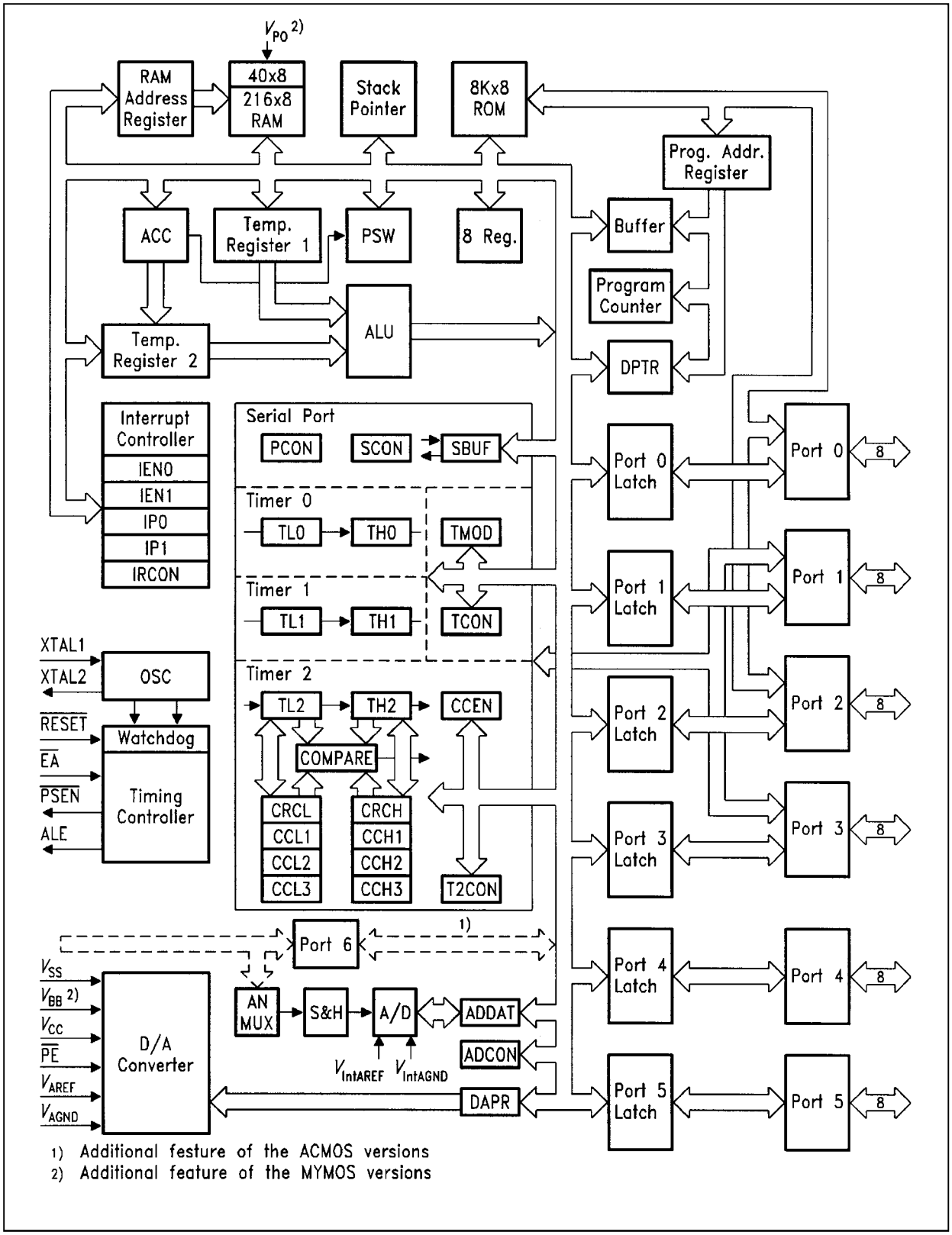


Bild 2.1: Komponenten im 80C515/80C535

3 Die CPU - Central Processing Unit

3.1 Funktion der CPU

Die CPU des 80(C)515/-535 ist das Herz des ganzen Mikrocontrollersystems. Sie ist, wie alle anderen Komponenten des Systems auf dem Chip integriert und besteht aus drei Teilen:

- dem Befehlsdekoder
- dem Rechenwerk und
- der Ablaufsteuerung

3.1.1 Der Befehlsdekoder

Hier werden die Befehle, die vom internen oder externen Programmspeicher eingelesen wurden, dekodiert und die zur Ausführung nötigen Schritte unternommen. Dies sind verschiedene Maßnahmen, wie z.B. weitere Daten aus dem Speicher zu holen, das Rechenwerk anzuweisen, bestimmte Verknüpfungen durchzuführen oder die Ablaufsteuerung zu veranlassen, an eine andere Programmstelle zu springen. Der Befehlsdekoder gibt seine Anweisungen an die verschiedenen CPU-Teile durch interne Steuersignale.

Der Befehlsdekoder dekodiert einen bestimmten Befehlssatz, das sind die binären Maschinenbefehle der CPU. Beim 80(C)515/-535 ist der Befehlssatz identisch mit dem des 8051.

3.1.2 Das Rechenwerk

Dieser Teil der CPU wird auch oft ALU (arithmetic-logic-unit) genannt, was nicht ganz korrekt ist; die ALU ist quasi der mathematische Kern. Sie führt die arithmetischen und logischen Verknüpfungen durch, die der Befehlsdekoder zur Abarbeitung eines bestimmten Befehls anfordert. Das Rechenwerk des 80(C)515/-535 besteht aus der ALU, den beiden Registern A und B und dem Register für das Programm-Status-Wort (PSW). Die ALU bildet auf Anweisung des Befehlsdekoders aus einer oder zwei Eingangsinformationen eine neue 8-Bit Ausgangsinformation. Dazu benutzt sie die vier Grundrechenarten, Inkrement oder Dekrement bilden, Schiebeoperationen, logische Operationen wie AND, OR, EXOR, Komplementbildung, Vergleichsoperationen usw. Ebenso sind Einzelbit-Operationen wie Bit-Setzen oder Bit-Zurücksetzen, oder Komplementieren und bedingte Sprünge möglich.

An erster Stelle der oben angesprochenen Register, die als Ergänzung zur ALU gesehen werden können, steht der Akkumulator, kurz Akku oder Register A, der an jeder Operation des Rechenwerks beteiligt ist. Hier steht nach einem arithmetischen oder logischen Befehl das Ergebnis. Ein weiteres Register ist das PSW, das Programm-Status-Wort. Hier befinden sich zahlreiche Status-Flags, die in Abhängigkeit von den Ergebnissen der Operationen gesetzt oder zurückgesetzt wer-

den. Zuletzt ist noch das Register B zu erwähnen, das für Multiplikation oder Division neben dem Akku benötigt wird.

3.1.3 Die Ablaufsteuerung

Die Ablaufsteuerung bestimmt den Ablauf des Programms. Normalerweise ist dieser Ablauf sequentiell, d.h. auf einen Befehl folgt der im Programmspeicher an der darauffolgenden Stelle stehende. Dieser Ablauf wird von einem 16-Bit-Register, dem Programmzähler (PC = program counter) sichergestellt, das auf den nächsten auszuführenden Befehl im Programmspeicher zeigt. Für den sequentiellen Ablauf wird dieser Programmzähler einfach hochgezählt. Es gibt jedoch die Möglichkeit zu bedingten oder unbedingten Sprüngen, die vom sequentiellen Ablauf abweichen. In diesen Fällen wird der PC durch einen neuen Wert, das Sprungziel, überschrieben. Bei bedingten Sprüngen geschieht das nur, wenn eine logische Bedingung erfüllt ist. Bei unbedingten Sprüngen wird der PC immer überschrieben.

3.2 Das CPU-Timing

Die CPU des 80(C)515/-535 hat ein relativ einfaches Timingschema mit einem festen Raster. Danach laufen alle Befehle in einem oder zwei Maschinenzyklen ab. Nur die Multiplikation und die Division benötigen vier Zyklen. Jeder Maschinenzyklus hat die gleiche Länge und wird direkt aus dem Oszillatortakt durch Herunterteilen gewonnen.

Befehle bestehen immer aus einem, zwei oder drei Bytes. Das erste ist immer der sogenannte Opcode; er legt fest, um welchen Befehlstyp es sich überhaupt handelt. Die CPU kann am Opcode erkennen, ob der ganze Befehl noch aus weiteren Bytes besteht und diese dann einlesen. Ein Beispiel für einen Ein-Byte-Befehl (er besteht nur aus einem Byte) ist RET; dieser Befehl führt einen Rücksprung aus einem Unterprogramm durch. Wenn die CPU einen solchen Befehl erkennt, „weiß“ sie vollständig, was zu tun ist.

Ein Beispiel für einen Zwei-Byte-Befehl ist MOV A,direct. Der Opcode dieses Befehls weist die CPU an, einen Operanden mittels direkter Adressierung aus dem Speicher zu holen und in den Akku zu bringen. Wenn die CPU diesen Opcode gelesen hat, „weiß“ sie, daß das nächste Byte noch zu diesem Befehl gehört; sie liest daher dieses Byte ebenfalls ein, benutzt es als direkte Adresse und kann damit den Befehl ausführen.

Ganz analog verhält es sich mit Drei-Byte-Befehlen. Hier legt der Opcode fest, daß noch zwei weitere Bytes zum Befehl gehören. Ein Beispiel dafür ist MOV direct,#data (bringe in eine direkt adressierte Speicherzelle einen 8-Bit-Wert). Die Zieladresse ist das zweite Byte des Befehls, im dritten steht der Wert, der eingeschrieben werden soll.

Nun zurück zum Timing. In Bild 3.1 ist das Timing-Schema abgebildet, mit dem die CPU die Befehle aus dem Programmspeicher holt und sie abarbeitet. Ganz oben ist der zugrunde liegende Takt zu sehen, das Signal OSC. Es bezieht sich auf den Takt am Anschluß *XTAL2* des Contollers. Ob dieses Signal durch einen Oszillator, einen Quarz oder eine externe Signalquelle erzeugt wird, spielt für den internen Ablauf keine Rolle. Der Takt wird durch Zähler und Teiler

in diverse interne Taktsignale zerlegt. Diese Signale werden dann überall im 80(C)515/-535 verwendet und bilden synchrones Korsett für alle Abläufe.

Die immer gleich langen Maschinenzyklen werden aus zwölf Oszillatortaktperioden gebildet. Je zwei Oszillatortaktperioden werden zu einem sogenannten State zusammengefaßt. Damit besteht ein Maschinenzyklus aus sechs States; diese sind im Bild 3.1 mit S1 bis S6 bezeichnet. Die States ihrerseits werden wieder in zwei Phasen unterteilt; im Bild P1 und P2 genannt. Die kleinste intern verwendete Zeiteinheit ist damit der Phase-Takt, der aus einer Oszillatortaktperiode besteht. Mit diesem System läßt sich jeder Zeitpunkt innerhalb eines Maschinenzykluses einfach durch die Angabe von State und Phase beschreiben: z.B. S4P2.

Da man die internen States und Phases nicht von außen sehen kann, ist das Signal *ALE* (Address Latch Enable) gezeigt. Es zeigt an, daß die CPU gültige Adressen ausgibt. *ALE* ist zweimal während eines Maschinenzykluses aktiv (auf High-Pegel): während S1P2 und S2P1 und noch mal während S4P2 und S5P1. Da dieses Schema starr ist und jeder Befehl in ein, zwei oder vier Maschinenzyklen abläuft, hat jeder Befehl eine feste und berechenbare Ausführungszeit.

Ein Ein-Zyklus-Befehl läuft folgendermaßen ab: Der Maschinenzyklus beginnt definitionsgemäß mit S1P1. Während S1P2 wird das Opcodebyte vom Speicher in den Befehlsdekoder übernommen und mit der Ausführung begonnen. Wie oben erklärt, kann die CPU am Opcode erkennen, ob noch weitere Bytes zum Befehl gehören. Wenn ja, wird die CPU noch im laufenden Maschinenzyklus ein weiteres Byte einlesen; dies geschieht während S4P2. Aus diesem Grund ist es möglich, daß es Zwei-*Byte*-Befehle gibt, die trotzdem Ein-*Zyklus*-Befehle sind. Voraussetzung ist aber, daß die CPU die Ausführung noch im ersten Maschinenzyklus beenden kann.

Was geschieht nun, wenn die CPU einen reinen Ein-Byte-Befehl abarbeitet? In diesem Fall müßte die CPU während S4P2 nichts einlesen. Das ist richtig; die CPU liest das Byte (den Opcode des nächsten Befehls) aber trotzdem. Da sie es aber zu diesem Zeitpunkt nicht braucht, wird es verworfen. Dasselbe Byte wird dann während des nächsten Maschinenzyklus zum Zeitpunkt S1P2 nochmals eingelesen und dann verwendet. Das gleiche Schema gilt analog auch für Zwei- und Vier-Zyklus-Befehle.

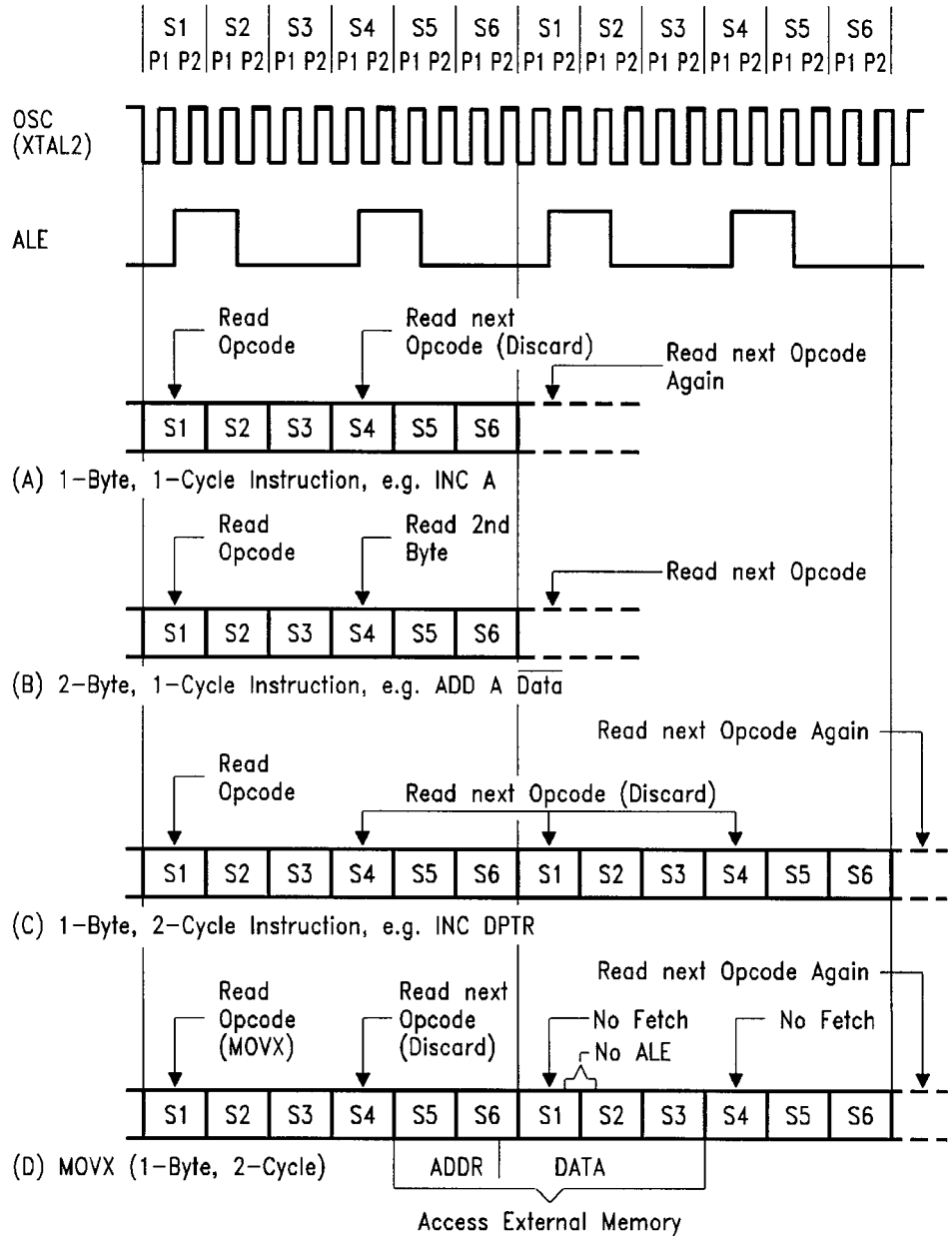


Bild 3.1: Fetch/Execute Sequenz

4 Speicherorganisation

Der Speicherraum des 80(C)515/-535 ist in vier verschiedene Bereiche unterteilt. Siehe auch Bild 4.1.

- bis zu 64 KByte Programmspeicher
- bis zu 64 KByte externer Datenspeicher
- 256 Byte interner Datenspeicher
- ein 128 Byte großer Bereich für Special-Function-Registers (SFR)
- 8 KByte interner Progammspeicher (nur 80(C)515)

Mit der strengen Unterscheidung zwischen Programm- und Datenspeicher, die mit unterschiedlichen Befehlen erreicht werden, weist der 80(C)515/-535 eine sogenannte Harvard-Struktur auf im Gegensatz zu der Von-Neumann-Struktur, bei der Daten- und Programmspeicher im selben logischen Speicherbereich liegen. Außerdem enthält der Befehlssatz keinen Befehl, um in den Programmspeicher schreiben zu können; der Programmspeicher ist ein ROM (Nurlesespeicher). Die Konsequenz daraus ist, daß das Programm nicht dynamisch, also während der Laufzeit, verändert werden kann. Vorteile sind eine bessere Übersichtlichkeit der Programme und eine erhöhte Sicherheit.

4.1 Der Programmspeicher

Wie oben schon gesagt, enthält der 80(C)515 einen eingebauten 8 KByte Programmspeicher als optionalen Teil. Der ROM-lose 80(C)535 arbeitet dagegen nur mit externem Speicher. In allen Fällen kann der externe Speicher bis zu 64 KByte groß sein. Liegt der Eingangspin $\overline{EA}$ (external access) auf H-Pegel, arbeiten die ROM-Versionen aus dem internen ROM, solange der Programmzähler Adressen unterhalb 2000H enthält. Speicherplätze mit Adressen darüber bis 0FFFFH weisen immer auf den externen Programmspeicher. Da der 80(C)535 keinen internen Programmspeicher besitzt, muß der Anschluß $\overline{EA}$ hier generell auf L-Pegel gelegt werden. In jedem Fall erfolgt die Adressierung beim Programmzugriff über den sechzehn Bit breiten Programmzähler (PC), unabhängig, ob mit internem oder internem Speicher gearbeitet wird.

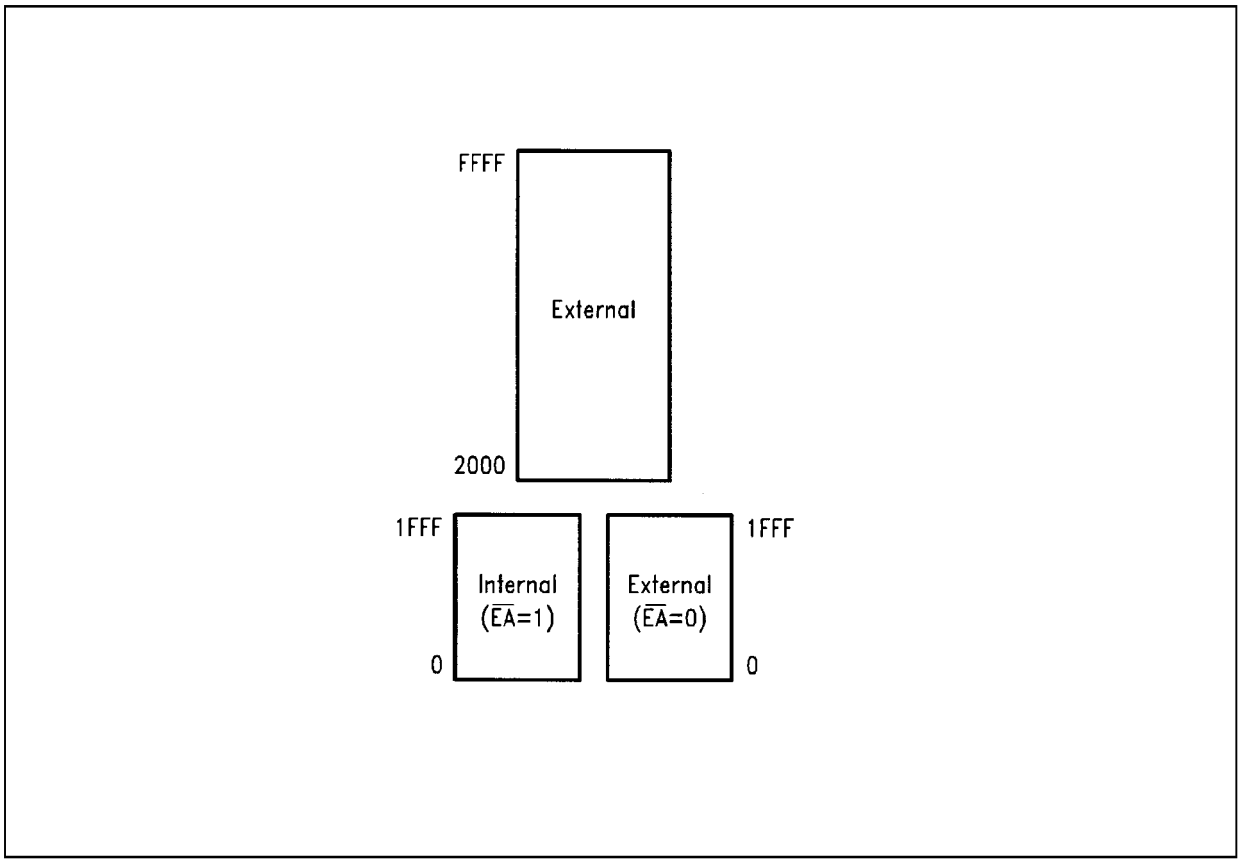


Bild 4.1: Adreßbereich des Programmspeichers

Im Bereich von 03H bis 93H des Programmspeichers liegen die Einsprungsadressen der Interrupt-Routinen. Wenn einzelne Interrupts nicht benutzt werden, kann der zugehörige Bereich für normales Programm verwendet werden.

4.2 Der Datenspeicher

Wie Bild 4.2 zeigt, läßt sich auch der Datenspeicher in einen internen und einen externen Bereich aufteilen.

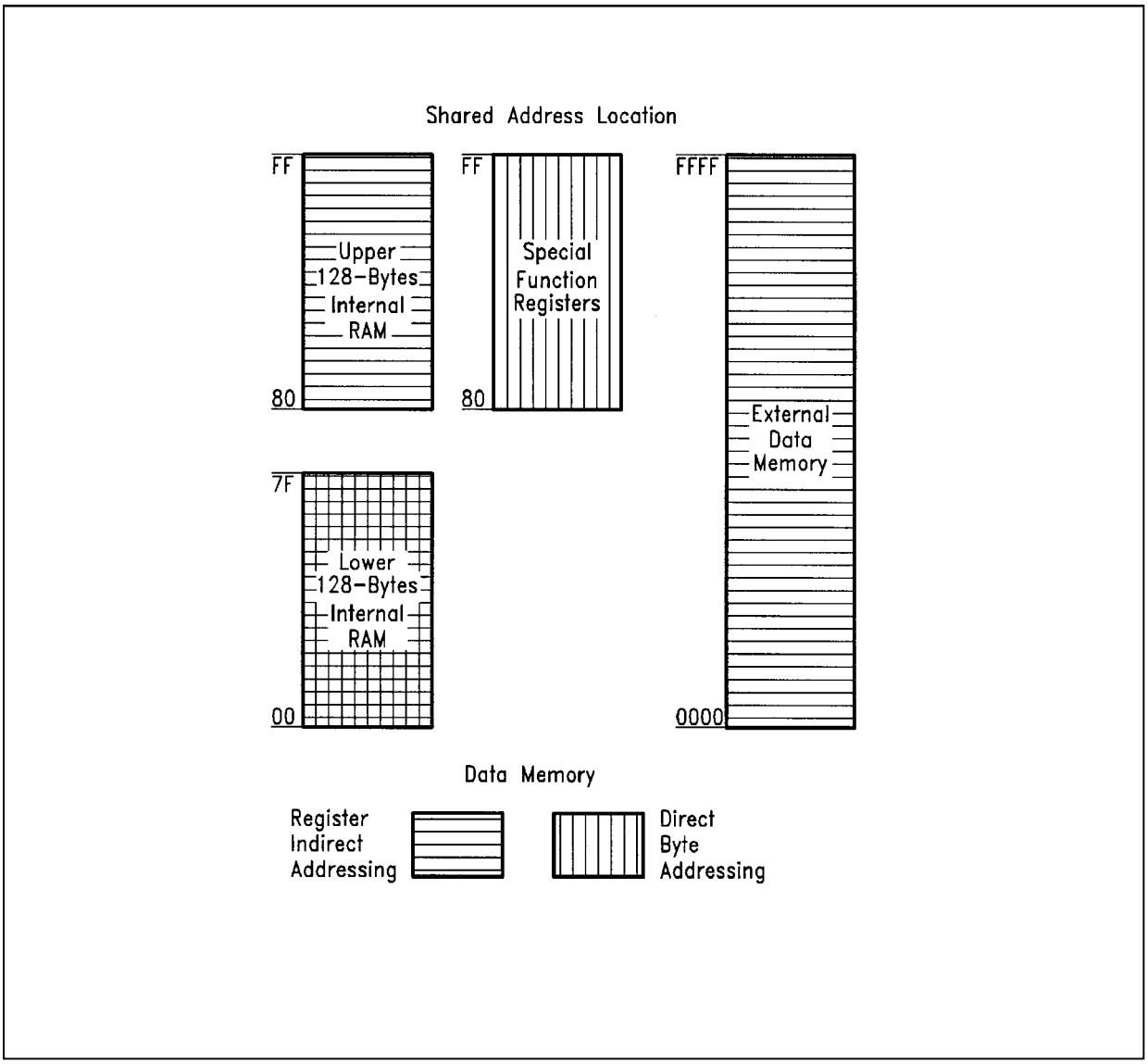


Bild 4.2: Adreßraum für Datenspeicher und Special Function Register

Der interne Datenspeicher, der sich beschreiben und lesen läßt (RAM), besteht aus drei getrennten Blöcken: den unteren 128 Byte RAM, den oberen 128 Byte RAM und dem 128 Byte großen Bereich der Special-Function-Registers (SFR). Bild 4.2 veranschaulicht diese Aufteilung. Weil die beiden letztgenannten Bereiche logisch den gleichen Adressraum belegen, erfolgt die Auswahl des jeweiligen Blocks durch die Adressierungsart. Die unteren 128 Byte können sowohl mit direkter als auch indirekter Adessierung angesprochen werden. Die oberen 128 Byte des internen RAM nur durch Befehle, die indirekte Adressierung verwenden. Die SFR müssen daher ausschließlich direkt angesprochen werden.

Speicherbereich	Adresse	Adressierungsart
Untere 128 Byte RAM	00H bis 7FH	direkt oder indirekt
Obere 128 Byte RAM	80H bis 0FFH	indirekt
SFR	80H bis 0FFH	direkt

Tabelle 4.1: Speicherbereiche und Adressierungsarten

Die unteren 128 Byte des internen RAM lassen sich weiter in drei Gruppen aufteilen:

- Auf den Adressen 00H bis 1FH sind allgemein verwendbare Register untergebracht.

- Die anschließenden 16 Byte (Adresse 20H bis 2FH) enthalten 128 direkt adressierbare Bits Diese Bits lassen sich auf zwei verschiedene Weisen ansprechen. Die eine Möglichkeit ist, ihre Bitadressen zu verwenden, d.h. Bit 00H bis 07H. Die andere ist, sich auf die Bitstelle im jeweiligen Byte 20H bis 2FH zu beziehen. Die Bits 0 bis 7 können also auch als Bits 20.0 bis 20.7 adressiert werden. Die Bits 8 bis 0FH sind die selben wie 21.0 bis 21.7. Selbstverständlich können alle 16 (bitadressierbaren) Byte auch als solche verwendet werden.

- Die Adressen 30H bis 7FH stehen allgemein als Datenbytes zur Verfügung.

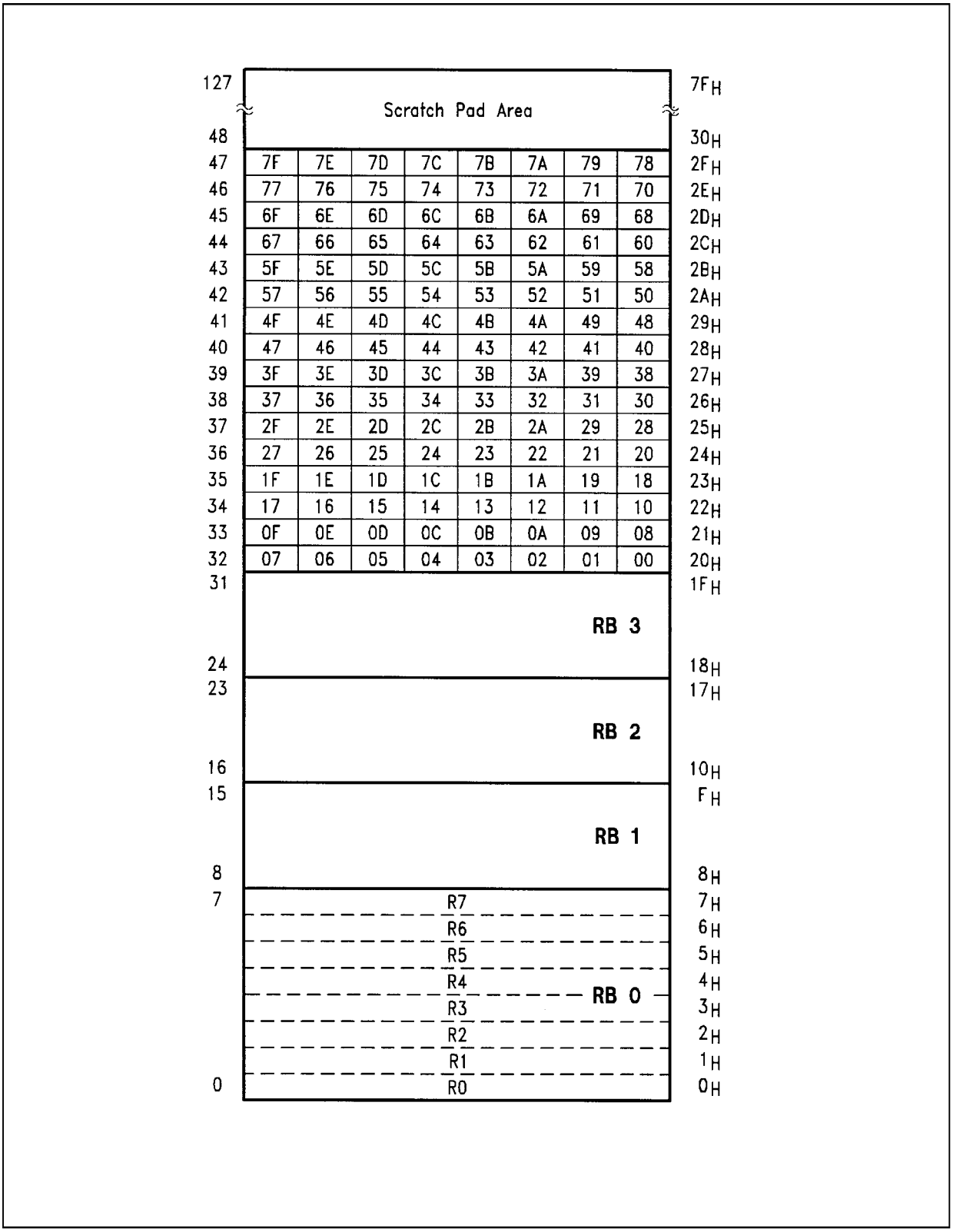


Bild 4.3: Der untere interne Datenspeicher

Mit dem Stack Pointer (SP) oder Stackzeiger - einem SFR - läßt sich der Stack überall innerhalb des internen RAM legen (auch in den indirekt adressierbaren Bereich). Die Stacktiefe ist nur durch die Größe des internen Datenspeichers begrenzt. Der Anwender muß lediglich darauf achten, daß der Stack nicht die anderen Daten überschreibt und umgekehrt.

Die Bilder 4.2 und 4.3 geben eine Übersicht der Aufteilung zwischen internem und externem Datenspeicher wieder. Um den Datenspeicher außerhalb des Mikrocontrollers zu erreichen, stellt der Befehlssatz die Instruktion MOVX (MOV eXternally) bereit, die in Kombination mit dem 16 Bit breiten DPTR (Datapointer) oder einem 8 Bit Universalregister verwendet wird. Insgesamt können maximal 64 KByte externer Datenspeicher adressiert werden.

4.3 Die allgemein nutzbaren Register - Universalregister

Den unteren 32 Byte des internen RAM sind vier Registerbänke mit je acht allgemein nutzbaren Registern zugeordnet. Diese Register stehen dem Anwender zur freien Verfügung. Allerdings kann zu einem Zeitpunkt immer nur eine Bank aktiv sein. Jeder Befehl, der eines der Register benutzt, bezieht sich stets auf die aktuelle Registerbank. So kann das Register R0 je nach aktiver Bank das Datenbyte 00H, 08H, 10H oder 18H sein (siehe Bild 4.3). Die Bankauswahl erfolgt über die Bits PSW.3 und PSW.4 des SFR PSW (Program Status Word, 0D0H), das im Zusammenhang mit den Special Function Registers besprochen wird. Wird keine Registerbank explizit angegeben, ist automatisch die Bank 0 selektiert. Die Möglichkeit, verschiedene Registerbänke zur Verfügung zu haben, erlaubt sogenanntes „Fast Context Switching“, d.h. ein schnelles Umschalten von einer Prozedur (z.B. Hauptprogramm) zu einer anderen (z.B. Interrupt-Routine), ohne erst wichtige Registerinhalte auf den Stack retten zu müssen.

Die acht Register können sowohl als solche als auch mit Hilfe der absoluten Adressen im internen RAM angesprochen werden. Der Vorteil der Registeradressierung ist, daß manche Befehle damit kürzer und schneller abgearbeitet werden oder gar nur für Registeradressierung vorgesehen sind. R0 und R1 können bei indirekten Adressmodi als Zeiger oder Indexregister verwendet werden, um auf internen oder externen Speicher zuzugreifen (z.B. MOV @R0,A).

Es muß beachtet werden, daß der Stack Pointer (SP) nach dem Einschalten oder nach einem Reset auf die Adresse 07H zeigt und damit den Stack ab Adresse 08H anlegt. Dies ist gleichzeitig R0 der Registerbank 1. Wenn also ein Anwender mehr als nur die untere Registerbank benutzen möchte, muß er den Stack Pointer auf einen Wert setzen, der auf einen freien Bereich zeigt.

4.4 Die Special Funciton Register - SFR

Der Bereich der SFR hat zwei wesentliche Aufgaben. Zum einen liegen in diesem Bereich alle CPU-Register außer dem Programmzähler PC und den vier Registerbänken. SFR sind in erster Linie die Arithmetikregister A, B und PSW sowie die Zeiger SP, DPH und DPL (Data Pointer higher Byte und lower Byte). Zum anderen stellen eine Anzahl von SFR die Schnittstelle zwischen CPU und den integrierten Peripherieeinheiten dar. Letzteres heißt, daß alle

Kontrollfunktionen und Datentransfers zu oder von Peripherieeinheiten ausschließlich über SFR abgewickelt werden.

Der Bereich der SFR liegt überlappend mit der oberen Hälfte des internen RAM auf den Adressen 80H bis 0FFH. Die Adressierungsart entscheidet, welcher Bereich angesprochen wird (siehe Tabelle 4.1). Bei direkter Adressierung werden die SFR gewählt. Sechzehn der Register, die auf durch acht teilbaren Adressen liegen, sind sowohl bit- als auch byteadressierbar. Damit ergeben sich 128 Bitadressen im SFR-Speicher.

Der Zugriff auf die SFR erfolgt genauso einfach wie auf die unteren 128 Byte des internen RAM und ist mit den meisten Befehlen möglich. Einige der SFR können, wenn der zugehörige Peripherieteil nicht benutzt wird, als allgemeine Register dienen. Allerdings sollte man sich vergewissern, daß durch das Schreiben in eines dieser SFR nicht doch eine Funktion ausgelöst wird. Im Zweifelsfall sollte auf SFR als Datenspeicher generell verzichtet werden.

Symbol		Name	Hex-Adresse
P0	*	Port 0	80H
SP		Stack Pointer	81H
DPL		Data Pointer, lower Byte	82H
DPH		Data Pointer, higher Byte	83H
PCON		Power Control Register	87H
TCON	*	Timer Control Register	88H
TMOD		Timer Mode Register	89H
TL0		Timer 0, lower Byte	8AH
TL1		Timer 1, lower Byte	8BH
TH0		Timer 0, higher Byte	8CH
TH1		Timer 1, higher Byte	8DH
P1	*	Port 1	90H
SCON	*	Serial Channel Control Register	98H
SBUF		Serial Channel Buffer Register	99H
P2	*	Port 2	0A0H
IEN0	*	Interrupt Enable Register 0	0A8H
IP0		Interrupt Priority Register	0A9H
P3	*	Port 3	0B0H
IEN1	*	Interrupt Enable Register 1	0B8H
IP1		Interrupt Priority Register 1	0B9H
IRCON	*	Interrupt Request Control Register	0C0H
CCEN		Compare/Capture Enable Register	0C1H
CCL1		Compare/Capture Register 1, lower Byte	0C2H
CCH1		Compare/Capture Register 1, higher Byte	0C3H
CCL2		Compare/Capture Register 2, lower Byte	0C4H
CCH2		Compare/Capture Register 2, higher Byte	0C5H
CCL3		Compare/Capture Register 3, lower Byte	0C6H
CCH3		Compare/Capture Register 3, higher Byte	0C7H
T2CON	*	Timer 2 Control Register	0C8H
CRCL		Compare/Relaod/Capture Register, lower Byte	0CAH
CRCH		Compare/Relaod/Capture Register, higher Byte	0CBH
TL2		Timer 2, lower Byte	0CCH
TH2		Timer 2, higher Byte	0CDH
PSW	*	Program Satus Word Register	0D0H
ADCON	*	A/D Converter Control Register	0D8H
ADDAT	*	A/D Converter Data Register	0D9H
DAPR		D/A Converter Program Register	0DAH
P6		Port 6	0DBH[1)]
ACC	*	Accumulator	0E0H
P4	*	Port 4	0E8H
B	*	B Register	0F0H
P5	*	Port 5	0F8H

[1)] Zusatz bei den ACMOS-Typen

Die mit einem Sternchen (*) in der zweiten Spalte gekennzeichneten SFR sind sowohl bit- als auch byteadressierbar

Tabelle 4.2: Die Special Function Register im 80C515/80C535

Beim aufmerksamen Lesen der Liste mit den SFR läßt sich feststellen, daß nicht alle Adressen von Registern belegt sind. Können diese dann zu Datenspeicherzwecken benutzt werden? Die eindeutige Antwort lautet: „Nein!“ Zur Erklärung: Das Konzept des 8051 ist, unter Wahrung der Software- und Funktionskompatibilität, relativ einfach neue Derivate (Ableger, Abkömmlinge)

entwickeln zu können, die neue Anforderungen abdecken. Der 80(C)515/-535 ist ein solches Derivat. Dazu muß die On-Chip-Peripherie verbessert und erweitert werden. Die SFR sind die einzigste Möglichkeit zur Kommunikation zwischen CPU und Peripherie, daher müssen neue SFR in den Adressraum aufgenommen werden. Die bekommen dann eben noch nicht belegte Adressen zugewiesen, die bei den Vorgängern freigehalten wurden. Wird eine solche Adresse beschrieben, in der (noch) kein SFR liegt, bedeutet das nicht, daß die zur späteren Erweiterung gedachte Funktion nicht bereits implementiert (aber noch dokumentiert) ist. Ein Schreibzugriff kann dann zu ungeahnten und nicht nachvollziehbaren Effekten führen.

Die Tabelle 4.2 zeigt die einzelnen SFR nach Adressen aufsteigend geordnet. Das Sternchen (*) in der zweiten Spalte bedeutet, daß dieses SFR sowohl bit- als auch byteadressierbar ist.

Nachfolgend werden einige SFR, die allgemeinen Zwecken dienen, näher beschrieben. Für die übrigen sei auf gesonderte Kapitel verwiesen.

4.4.1 Der Akkumulator - ACC, SFR-Adresse 0E0H

ACC ist der symbolische Name für das Akkumulator-Register, auch kurz Akku genannt. Wird dieses Symbol innerhalb eines mnemonischen Assemblerbefehls als direkte Adresse verwendet, ersetzt der Assembler ACC durch die Adresse 0E0H. Damit wird dann aus dem Befehl MOV R0,ACC der Maschinencode A8H, E0H. In vielen Befehlen hingegen kann der Operand Akkumulator als A implizit verwendet werden, wobei die Befehlslänge und die Ausführungszeit kürzer sind. Im vorhergehenden Beispiel würde damit aus MOV R0,A der Ein-Byte-Befehl F8H. Viele arithmetische oder logische Befehle können generell nur mit dem impliziten Akku arbeiten. So würde der Befehl ADD ACC,R0 zu einer Fehlermeldung führen (weil ACC ja eine direkte Adresse ist, für die dieser Befehl nicht existiert) während ADD A,R0 ein korrekter Befehl ist.

ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0
E7H (MSB)	E6H	E5H	E4H	E3H	E2H	E1H	E0H (LSB)

Bild 4.4: Special Function Register ACC (0E0H)

4.4.2 Das Programm Satus Wort - PSW, SFR-Adresse 0D0H

Dieses Register enthält Informationen über den aktuellen Programmstatus.

CY	AC	F0	RS1	RS0	OV	F1	P
D7H (MSB)	D6H	D5H	D4H	D3H	D2H	D1H	D0H (LSB)

Bit	Funktion
CY	Übertragsflag (Carry-Flag). Es wird gesetzt, wenn ein Überlauf aus der höchstwertigen Stelle des Akkus erfolgt.
AC	Hilfs-Überlaufflag (Auxillary-Carry) für BCD-Befehle. Es wird gesetzt, wenn ein Überlauf aus der Bitstelle 3 des Akkus erfolgt.
F0	Benutzerflag 0, zur freien Verfügung
RS1 RS0 0 0 0 1 1 0 1 1	Auswahlbits für Registerbank Bank 0 selektiert (Adresse 00H - 07H) Bank 1 selektiert (Adresse 08H - 0FH) Bank 2 selektiert (Adresse 10H - 17H) Bank 3 selektiert (Adresse 18H - 1FH)
OV	Überlaufbit (Overflow). Es wird gesetzt, wenn ein Übertrag von der Bitstelle 6 auf Bitstelle 7 erfolgt
F1	Benutzerflag 1, zur freien Verfügung
P	Paritybit. Es zeigt nach jedem Befehl die Parität des Akkus an. Es ist gesetzt, wenn die Anzahl der auf 1 gesetzten Bits ungerade ist.

Bild 4.5: Special Function Register PSW (0D0H)

4.4.3 Das B-Register - SFR-Adresse F0H

Das B-Register wird für Multiplikation und Division verwendet, die der Befehlssatz des 80(C)515/-535 zur Verfügung stellt. B dient zur Bereitstellung sowohl der Operanden als auch des Ergebnisses.

B.7	B.6	B.5	B.4	B.3	B.2	B.1	B.0
F7H (MSB)	F6H	F5H	F4H	F3H	F2H	F1H	F0H (LSB)

Bild 4.6: Special Function Register B (0F0H)

4.4.4 Der Stack Pointer (SP) - SFR-Adresse 81H

Der Stack Pointer ist acht Bit breit. Er wird inkrementiert, bevor ein Datenbyte während der Befehle PUSH und CALL auf dem Stack (Stapel) abgelegt wird, und dekrementiert, nachdem ein Datenbyte vom Stack während eines POP- oder RET- bzw. RETI-Befehls geholt wird. Es wird also nicht unterschieden, ob es sich bei den abgelegten Daten um Daten im engeren Sinn

oder um Adressen handelt. Der Stack Pointer zeigt also stets auf das zuletzt abgespeicherte Byte (Top of Stack). Der SP kann überall innerhalb des internen Datenspeichers angelegt werden.

Nach dem Reset ist der SP mit 07H initialisiert, so daß der Stack ab Adresse 08H beginnt (also direkt über der Registerbank 0). Im übrigen kann der SP durch das Programm gelesen und dynamisch verändert werden. Dabei ist jedoch einiges an Erfahrung und Disziplin notwendig.

SP.7 (MSB)	SP.6	SP.5	SP.4	SP.3	SP.2	SP.1	SP.0 (LSB)

Bild 4.7: Special Function Register SP (81H)

4.4.5 Der Data Pointer (DTPR) - SFR-Adresse 82H und 83H

Der Data Pointer ist, wie in allen 8051-Abkömmlingen, ein 16-Bit-Register. Er dient als Zeiger (pointer) zur Adressierung sowohl von externem Datenspeicher als auch externem (und internem) Programmspeicher. Die entsprechenden Zugriffe erfolgen über die Befehle MOVX und MOVC. In diesen Befehlen wird der DTPR immer als 16-Bit-Register verwendet. Der Data Pointer ist im aber auch im SFR-Bereich direkt zugänglich. Hier ist er in zwei 8-Bit-Register zerlegt, die jeweils eine eigene Adresse einnehmen. Die oberen acht Bit befinden in DPH (Adresse 83H), die unteren acht Bit im Register DPL (Adresse 82H). Das folgende Bild zeigt die beiden SFR.

DPL.7	DPL.6	DPL.5	DPL.4	DPL.3	DPL.2	DPL.1	DPL.0
DPH.7 MSB	DPH.6	DPH.5	DPH.4	DPH.3	DPH.2	DPH.1	DPH.0 LSB

Bild 4.8: Special Function Register DTPR (DPH und DPL; 83H und 82H)

4.4.6 Die Ports 0 bis 5

Mit den SFR P0 bis P5 werden die Ein-/Ausgabeports 0 bis 5 angesprochen. Jeder dieser Ports ist über sein SFR sowohl bit- als auch byteadressierbar und umfaßt acht Bit. Somit stehen für allgemeine Ein- und Ausgabezwecke insgesamt 48 I/O-Anschlüsse zur Verfügung. Nähere Einzelheiten werden in Kapitel 7 behandelt.

4.4.7 Der Port 6 - (AN0 bis AN7)

Bei den MYMOS-Versionen können die acht analogen Eingänge AN0 bis AN7 ausschließlich als Eingänge für den integierten A/D-Umsetzer genutzt werden. Bei den ACMOS-Versionen darüber hinaus auch als digitale Eingänge. In diesem Falle werden sie als zusätzlicher (Nur-) Eingangsport (Port 6) mit Hilfe des SFR P6 (0DBH) behandelt. Im Gegensatz zu den übrigen Ports besitzt P6 keine internen Flipflops, so daß der Inhalt, der über SFR P6 gelesen wird, nur von dem anstehenden Pegel am Anschlußpin abhängt.

4.4.8 Die Register der übrigen Peripheriekomponenten

Die meisten SFR werden als Steuerregister für die integrieten Peripheriekomponenten bzw. als deren Daten- und Statusregister benötigt. Die verbleibenden SFR werden in den folgenden Kapiteln, zu denen sie thematisch gehören, behandelt.

5 Der externe Bus

Der 80(C)515/-535 erlaubt, externe Speicher sowohl für Programme als auch für Daten anzuschließen. Obwohl dies im strengen Sinn dem Ein-Chip-Grundgedanken widerspricht, da dann das System ähnlich dem klassischen Mikroprozessorsystem aus mehreren Bausteinen aufgebaut wird, gibt es viele gute Gründe für die externe Speichererweiterung. Oft wird externer Speicher deswegen eingesetzt, weil der On-Chip-ROM beim Hersteller maskenprogrammiert wird und somit nur große Stückzahlen (üblicherweise mehrere Tausend Stück) mit spezieller ROM-Programmierung geliefert werden können. Da dann auch nachträgliche Programmänderungen nicht mehr möglich sind, ist für Anwendungen, die in kleinen Stückzahlen laufen, oder die häufige Änderungen erfahren, der externe Programmspeicher die vernünftigere Wahl. Hier kommt dann die ROM-lose Version, der 80(C)535, zum Einsatz, bzw. der 80(C)515 wird als „ROM-los“ betrieben.

5.1 Zugriffe auf externen Programm- und Datenspeicher

Im Kapitel 4 wurde bereits die Architektur des 80(C)515/-535 besprochen, die hinsichtlich der Speicherorganisation von der klassischen Von-Neumann-Struktur abweicht. Wie dargestellt hat der 80(C)515/-535 keinen einheitlichen Adreßraum für alle Arten des Speichers. Vielmehr gibt es logisch und physikalisch getrennte Adreßräume für Programmspeicher, 256 Byte internes RAM, die SFR und den externen Datenspeicher. Je nach Bedarfsfall wird der entsprechende Speicher aktiviert.

Der externe Bus erlaubt nun den Anschluß von Programmspeicher und externem Datenspeicher. Der externe Programmspeicher verhält sich dabei identisch wie der interne; die CPU sieht keinen Unterschied, ob sie auf den internen oder externen Programmspeicher zugreift. Dies bedeutet auch, daß die Ausführungszeiten der Befehle in beiden Fällen gleich sind. Auch die Software muß nicht unterschiedlich sein; es ist lediglich zu beachten, daß bei Verwendung des externen Busses verschiedene Ports nicht mehr zur Verfügung stehen, da sie für den Busbetrieb benötigt werden.

Die Umschaltung zwischen internem und externem Programmspeicher ist eine reine Hardwarefunktion, wobei es zwei Möglichkeiten gibt, die Zugriffe zu lenken: Zum einen schaltet der 80(C)515 vom internen auf den externen Programmspeicher um, wenn der Programmzählerstand über die Grenze des internen Programmspeichers (1FFFH) hinausläuft, zum anderen wird durch die Aktivierung des Pins $\overline{EA}$ der 80(C)515 generell gezwungen, externen Programmspeicher zu verwenden. Bei den ROM-losen Versionen muß dieser Pin immer aktiviert (low) sein.

Grundsätzlich anders ist die Situation beim externen Datenspeicher. Wie im Kapitel 4 schon erläutert, handelt es sich hier um einen getrennten Adreßbereich, der mit eigenen Befehlen (MOVX, ...) adressiert wird. Es ist also nicht möglich, den externen Datenspeicher mit den gleichen vielseitigen Befehlen wie das interne RAM zu bedienen; vielmehr ist der Befehlstyp MOVX auf Datentransfers zwischen Akku und externem Datenspeicher beschränkt. Außer ihm steht nur die indirekte Adressierung über den 16-Bit-Data-Pointer oder die Bankregister R0 und R1 zur Verfügung.

Der externe Bus stellt, wie jeder Mikroprozessorbus, Adressen, Daten und Steuersignale zur Verfügung. Adressen- und Datenbus werden durch die Ports P0 und P2 gebildet: Port P0 bedient im Multiplex sowohl die unteren acht Adressenbits als auch die acht Datenbits. Port P2 ist nur für die oberen acht Adressenbits zuständig. Der so gebildete externe Adreß- und Datenbus wird für Zugriffe auf Programm- und Datenspeicher verwendet. Die Unterscheidung, welcher dieser beiden Speicherbereiche tatsächlich angesprochen wird, wird durch die Aktivierung unterschiedlicher Steuersignale getroffen. Bei Zugriffen auf den Programmspeicher wird das Signal $\overline{PSEN}$ aktiviert. Es handelt sich dabei um ein Lesesignal, da definitionsgemäß aus dem Programmspeicher nur gelesen werden kann. Für den Zugriff auf den externen Datenspeicher gibt es sowohl ein Lesesignal, $\overline{RD}$ (Alternativfunktion von *P3.7*), als auch ein Schreibsignal, $\overline{WR}$ (Alternativfunktion von *P3.6*). Je eines dieser Signale wird während der Ausführung der MOVX-Befehle aktiviert.

5.2 Signale des externen Busses

Bei Zugriffen auf externe Speicher dient Port P0 sowohl zur Ausgabe des unteren Adressenbytes als auch zur Übertragung des Datenbytes. Dabei arbeitet P0 im Zeitmultiplex, d.h. während bestimmter Zeiten im Maschinenzyklus wird die Adresse (lower Byte) ausgegeben, während zu anderen Zeiten Daten gelesen oder geschrieben werden. Bei der Verwendung als externer Busteil wird Port P0 von seinem internen Latch abgetrennt und die Steuerung der Ausgangstreiber wird direkt von den internen Bussignalen übernommen. Somit ist P0 kein Open-Drain-Port wie während seiner Ein-/Ausgabe-Funktion. Analoges gilt für den Port P2: Auch er steht bei der Verwendung als Busteil (higher Byte der Adresse) nicht mehr als Ein-/Ausgabeport im eigentlichen Sinn zur Verfügung. Eine Programmierung von Port P0 oder P2 ist wirkungslos ohne aber dem Baustein zu schaden.

5.2.2 Das Signal ALE - Adress Latch Enable

Die Funktion des Signales *ALE* ist es, die im Zeitmultiplex verschachtelten Signale auf Port P0 zu trennen, wenn dieser als Busteil arbeitet. Die abfallende Flanke von *ALE* zeigt an, daß zu diesem Zeitpunkt auf Port P0 das untere Adreßbyte gültig ist. Dies gilt bei Zugriffen sowohl auf den externen Daten- als auch Programmspeicher.

ALE wird üblicherweise an den Strobe-Eingang eines acht Bit breiten Latch angeschlossen. Damit speichert dieses Latch bei der fallenden Flanke von *ALE* die anliegenden Adressen und stellt sie dem Bussystem bis zur nächsten Aktivierung von *ALE* zur Verfügung, während Port P0 mit dem Datenbyte arbeitet. Obwohl es zur Funktion nicht nötig wäre, wird *ALE* auch aktiviert, wenn der Zugriff auf internen Speicher stattfindet.

5.2.3 Das Signal $\overline{PSEN}$ - Program Store Enable

Das Signal $\overline{PSEN}$ ist das Lesesignal für den externen Programmspeicher. Der 80(C)515 gibt es nicht aus, wenn auf den internen Programmspeicher zugreift. Ob der Zugriff ein echter Lesevor-

gang oder nur ein Prefetch, der dann intern wieder verworfen wird, ist dabei unerheblich (zum Prefetch siehe Abschnitt 3.2). Das bedeutet, daß bei externem Zugriff zweimal pro Maschinenzyklus das $\overline{PSEN}$-Signal erzeugt wird, solange kein MOVX-Befehl ausgeführt wird. In diesem Fall entfallen die $\overline{PSEN}$-Signale zugunsten der Signale $\overline{RD}$ oder $\overline{WR}$.

5.2.4 Das Timing der Bussignale

In diesem Abschnitt wird das grundsätzliche Zeitverhalten der Bussignale diskutiert, d.h. in welchem Raster der Taktsignale der Bus reagiert. Hierzu wird eine idealisierte Betrachtung angestellt, und die Laufzeiten der Signale vernachlässigt; es wird einfach mit den Sollzeitpunkten gearbeitet.

Das Timing des externen Busses ist in Bild 5.1 dargestellt. Der obere Teil, Teil a), zeigt den Zugriff auf den internen Programmspeicher, der untere Teil, Teil b), den Zugriff auf den externen Datenspeicher. Das Bustiming folgt exakt dem CPU-Timing, wie es in Abschnitt 3.2 schon angesprochen wurde. Die dort beschriebene Unterteilung eines Maschinenzyklus in sechs States (S1 bis S6) und die weitere Unterteilung in die sognannten Phases (P1, P2) wird auch hier angewendet. Wie bei allen Mitgliedern der 8051-Familie folgt auch beim 80(C)515/-535 der externe Bus exakt den Abläufen in der CPU.

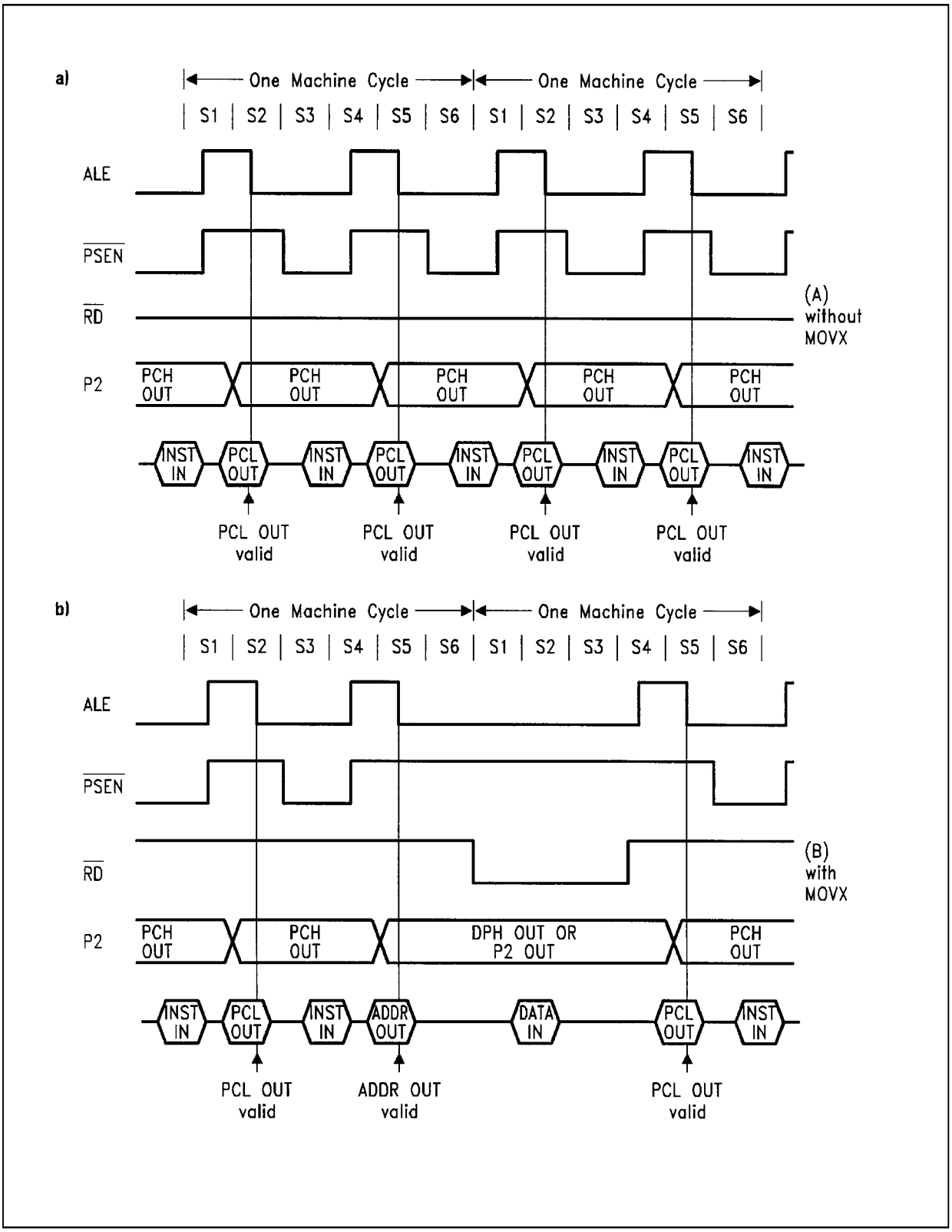


Bild 5.1: Timing beim Zugriff auf externen Speicher

Wie schon mehrfach erwähnt, führt der 80(C)515/-535 in jedem Maschinenzyklus zwei Programmspeicherzugriffe durch. Wegen der direkten Kopplung des externen Busses an die CPU ist die Ausführungszeit und der Ablauf bei internem oder externem Zugriff identisch. Ein solcher Zugriff läuft folgendermaßen ab: Zum Zeitpunkt S2P1 beginnt der 80(C)515/-535 mit der Adreßausgabe auf Port P0 und Port P2. Auf Port P0 wird das lower Byte des PC, auf Port P2 das higher Byte des PC ausgegeben. Nach einer Phase (S2P2) wird *ALE* inaktiv (abfallende Flanke); damit wird normalerweise die dann auf Port P0 anliegende untere Adresse in einem externen Latch gespeichert. Nach Ablauf einer weiteren Phase (S3P1) wird das Lesesignal für den externen Programmspeicher ($\overline{PSEN}$) aktiviert (low); gleichzeitig wird Port P0 hochohmig geschaltet, um das folgende Einlesen des Datenbytes aus dem externen Speicher zu ermöglichen. Spätestens bis zum Zeitpunkt S4P2 muß der Speicher das gültige Datenbyte auf den Bus (Port P0) angelegt haben, da der 80(C)515/-535 in diesem Moment das Byte einliest. Zu diesem Zeitpunkt wird auch das Lesesignal $\overline{PSEN}$ wieder inaktiv. Außerdem nimmt *ALE* wieder den aktiven Zustand (high) ein. Nach einer weiteren Phase (S5P1) schaltet der Port P2 von der bisher ausgegebenen Adresse auf die nächstfolgende Adresse (das obere Byte) um. Hier beginnt dann der zweite Programmspeicherzugriff innerhalb dieses Maschinenzyklus, der exakt so abläuft wie der eben beschriebene.

Man erkennt in Bild 5.1, daß die Signale *ALE* und $\overline{PSEN}$ gleichmäßig erzeugt werden und in bestimmten Fällen sogar als Taktsignal für andere Schaltungsteile verwendet werden können. Lediglich bei Zugriffen auf den externen Datenspeicher fallen die beiden Signale aus. Wie bereits oben erwähnt, wird das *ALE*-Signal sogar dann ausgegeben, wenn interner Programmspeicher angesprochen wird.

Bild 5.1b) zeigt die Vorgänge beim Zugriff auf den externen Datenspeicher; diese Art von Zugriff wird immer bei der Ausführung eines MOVX-Befehls erzeugt. Ob zur Adressierung der 16-Bit-Data-Pointer oder eines der Bankregister R0 oder R1 verwendet wird, hat auf den Ablauf keinen Einfluß; als einziger Unterschied wird beim 16-Bit-Zugriff Port P2 zur Ausgabe des higher Bytes der Adresse verwendet, während beim 8-Bit-Zugriff an Port P2 der Inhalt des Portlatches ausgegeben wird.

Die MOVX-Befehle sind immer Zwei-Zyklus-Befehle. Während des ersten Zyklus wird wie beim Programmspeicherzugriff zu S1P2 der entsprechende Opcode eingelesen; es folgt ein Prefetch (S4P2), der intern verworfen wird. Während S5P1 und S5P2 wird dann auf Port P0 die Adresse für den externen Datenspeicher ausgegeben. Auch hier wird *ALE* zur Steuerung des externen Latch verwendet. Zum Zeitpunkt S1P1 des zweiten Maschinenzyklus wird das Lese- oder Schreibsignal ($\overline{RD}$ oder $\overline{WR}$) aktiviert, je nachdem ob es sich um einen lesenden (z.B. MOVX A, @DPTR) oder schreibenden (z.B. MOVX @DPTR, A) Befehl handelt. Beim Lesen wird Port P0 hochohmig, um die Daten aus dem Speicher zu übernehmen, während beim Schreiben das Datenbyte auf Port P0 gelegt wird.

Das Bild 5.1 macht auch deutlich, daß die Anforderungen an die Zugriffzeiten beim externen Datenspeicher wesentlich weniger scharf als beim externen Programmspeicher sind. Dies ist auch aus den entsprechenden Parametern im Datenblatt ersichtlich. Damit ist es möglich, auch langsamere externe Peripheriebausteine an den externen Bus anzuschließen. Eine Möglichkeit, den Zugriff auf die externen Speicher anderweitig zu verzögern, z.B. Wait-States einzufügen, gibt es bei Mikrocontrollern der 8051-Familie nicht.

5.2.5 Überlappung von externem Programm- und Datenspeicher

In manchen Anwendungsfällen kann es nötig sein, denselben physikalischen Speicher in die logisch getrennten Bereiche Programm- und Datenspeicher einzublenden. Dies wird dann gemacht, wenn man während des Programmlaufes Code in den Programmspeicher hineinschreiben will; da dies definitionsgemäß bei 80(C)515/-535 nicht möglich ist, hilft man sich so, daß man den gewünschten Code in den externen Datenspeicher schreibt; durch das Zusammenlegen der beiden Bereiche wird dann der in den Datenbereich geschriebene Code auch aus dem Programmspeicher gelesen und ausgeführt.

Die beschriebene Verknüpfung kann man einfach dadurch realisieren, daß die getrennten Lesesignale $\overline{PSEN}$ und $\overline{RD}$ durch ein UND-Gatter zu einem einzigen Lesesignal verknüpft werden. Dieses Signal wird dann an den Speicher angeschlossen. Das Schreibsignal $\overline{WR}$ kann dann wie sonst verwendet werden. Es ist lediglich zu beachten, daß der Speicher nun auch die schärferen Anforderungen an die Zugriffszeit des Programmspeichers erfüllen muß.

5.2.6 Zugriff durch den Data Pointer

Die 8051-Architektur sieht lediglich einen Data Pointer vor, der mit Hilfe von MOVX- und MOVC-Befehlen zur indirekten Adressierung bei Zugriffen auf den Programm- und externen Datenspeicher dient. Dazu kommt, daß der Data Pointer selbst, der ein 16-Bit-Register ist, bis auf zwei Ausnahmen nur 8-Bit-Operationen bearbeiten kann. Die Ausnahmen sind das Laden des Data Pointers mit einem 16-Bit-Wert (MOV DPTR,#data16), sowie das Inkrementieren (INC DPTR). Was aber sehr oft benötigt wird, ist das Sichern und Zurückladen des Data Pointer (z.B. auf den Stack), wie man es in Interrupt-Subroutinen und Unterprogrammen verwendet. Sobald man außerdem mit mehr als einem Datenbereich gleichzeitig arbeiten will, z.B. um ein Datenfeld vom ROM in den externen Datenspeicher zu kopieren, muß der Data Pointer ständig umgeladen und der alte Inhalt gesichert werden. Dies ist wie gesagt dadurch bedingt, daß diese Operationen nur mit Hilfe des Data Pointer durchgeführt werden können. So wird in den Programmen der Data Pointer oft zum Flaschenhals bei der Programmierung.

6 Reset

Auf den ersten Blick scheint die Reset-Funktion nichts Besonderes zu sein. So wird oft bei Mikroprozessoren dieses Thema nur kurz beschrieben und die voreingestellten Werte (Default-Werte) angegeben. In den meisten Fällen wird dies auch genügen. Bei komplexeren Controlleranwendungen kommen aber noch weitere Fragen ins Spiel. Wie ist der genaue Ablauf der Reset-Funktion, wie verhalten sich die Portpins auch nach dem Wiedereinschalten? Hier muß der Entwickler eines Controllersystems genau wissen, was während des Resets und nach dem Wiederanlauf geschieht, besonders, wenn zeitkritische Anwendungen gesteuert werden.

6.1 Hardware-Reset

6.1.1 Funktion und Beschaltung

Die Hardware-Resetfunktion im 80(C)515/-535 wird benötigt, um den Baustein nach dem Einschalten in einen definierten Zustand zu bringen. Aber auch während des Betriebs kann über die Reset-Funktion ein Neustart des Bausteins erzwungen werden. Ein besonderer Anwendungsfall für den Hardware-Reset ist die Beendigung eines Software-Power-Down-Modus (siehe Kapitel 12), aus dem kein anderer Weg herausführt.

Der Begriff Hardware-Reset beschreibt hier das Anlegen eines Reset-Signals an den Pin $\overline{RESET}$ (Pin 10). Über diesen Hardware-Reset hinaus kennt der 80(C)515/-535 noch zwei weitere Reset-Quellen: den internen Hardware-Reset und den Reset, den der Watchdog-Timer erzeugen kann. Hier soll zunächst nur der Hardware-Reset besprochen werden; er ist dann Grundlage für die beiden anderen.

Der Pin $\overline{RESET}$ ist ein Eingang, der durch Low-Pegel aktivert wird. Der 80(C)515/-535 hat intern einen Schmitt-Trigger mit einer Hysterese von einigen 100mV. Dies dient dazu, daß der Baustein beim langsamen Anstieg der Spannung am Reset-Eingang das interne Reset-Signal definiert abschaltet. Der erwähnte langsame Anstieg tritt vorallem beim Anlegen einer Spannung auf, die mit Schaltungen nach Bild 6.1 erzeugt wurden.

Im Fall von Bild 6.1 a) wird ein externer Kondensator durch einen internen Pullup-Widerstand langsam geladen. Diese Spannung ist anfänglich, auch nach Betätigen des Tasters in Bild 6.1 c) 0 Volt. Damit ist die Reset-Funktion aktiviert. Allmählich steigt die Spannung am Reset-Eingang bis die Schaltschwelle durchlaufen ist und der Reset beendet ist. In jedem Fall muß dafür Sorge getragen werden, daß die Betriebsspannung nach dem Einschalten stabil ansteht und keine Einbrüche aufweist.

Ist die vorgenannte Bedingung nicht garantiert, sollte eine Betriebsspannungsüberwachung beim Einbruch oder Absinken einen definierten Reset-Impuls erzeugen. Der Ausgang einer solchen Schaltung kann z.B. nach Bild 6.1 b) angeschlossen werden.

Für ein korrektes Verständnis des internen Reset-Ablaufs ist es wichtig zu wissen, daß der 80(C)515/-535 den Reset synchron behandelt. Das bedeutet, daß der Pegel am Reset-Eingang einmal pro Maschinenzyklus abgetastet wird. Falls er aktiv (low) ist, führt die CPU eine interne Reset-Sequenz durch, die zwei Maschinenzyklen in Anspruch nimmt, vergleichbar einem Zwei-Zyklus-Befehl. Damit der zweite Zyklus noch korrekt abläuft, ist es notwendig, daß der 80(C)515/-535 beim zweiten Abtasten des Reset-Eingangs nach dem Resetbeginn immer noch Low-Pegel vorfindet. Andernfalls wird der Reset-Vorgang abgebrochen und die CPU befindet sich in einem undefinierten Zustand. Der Reset-Eingang muß also mindestens zwei Maschinenzyklen aktiviert sein, damit der Reset auch sicher ausgeführt wird; ist er länger aktiv, bleibt der 80(C)515/-535 solange im Reset-Zustand, bis der Eingang High-Pegel führt.

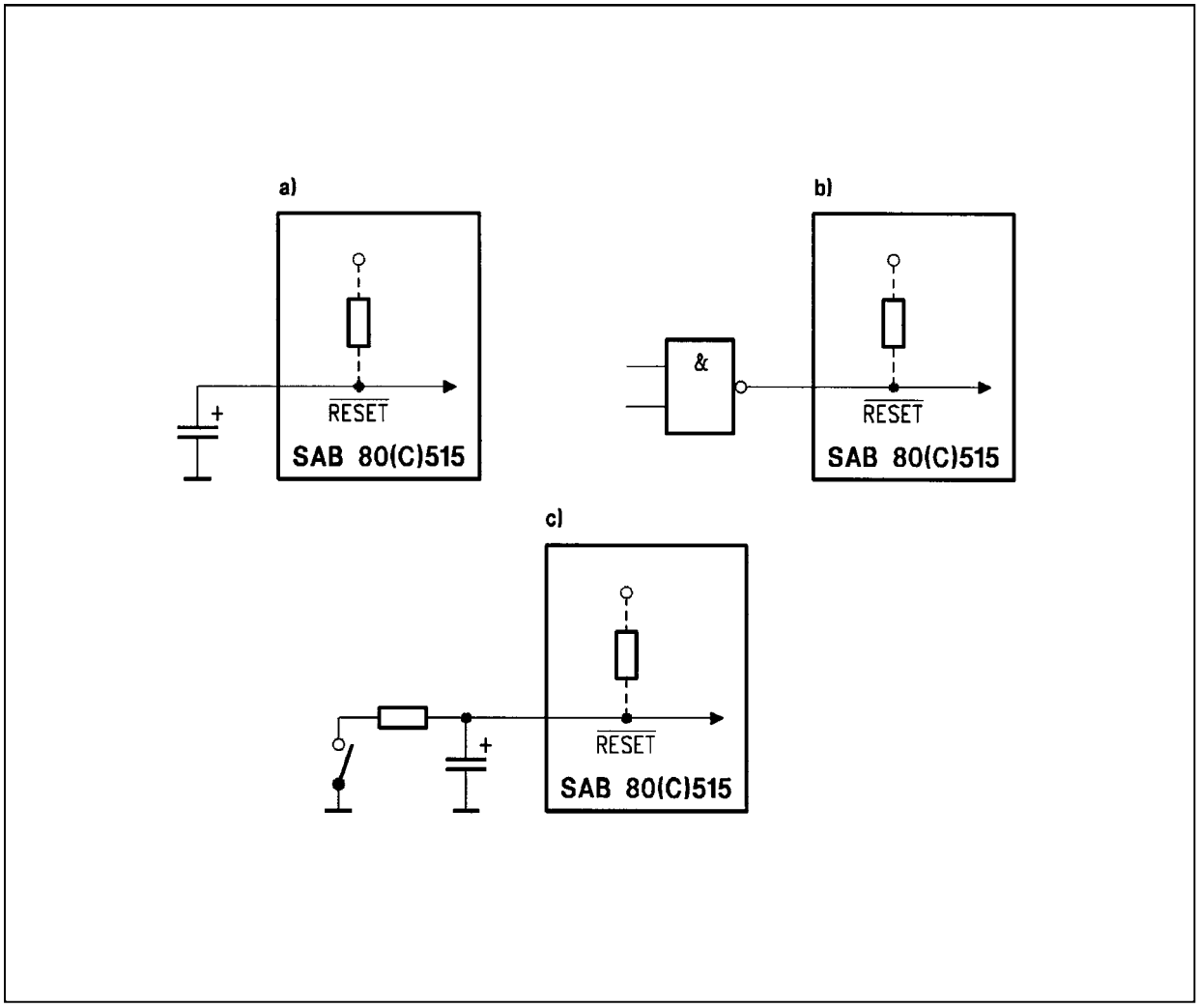


Bild 6.1: Schaltungen für Reset

Während des Reset-Vorgangs muß der Oszillator des 80(C)515/-535 laufen. Bei Quarzoszillatoren dauert es erfahrungsgemäß mehrere Millisekunden nach dem Einschalten bis der Oszillator angeschwungen und stabil ist. Das Reset-Signal muß diese Zeitspanne in jedem Fall überbrücken und zusätzlich die beiden Maschinenzyklen für den Reset-Vorgang garantieren. Der Hersteller

empfiehlt eine Mindest-Resetzeit beim Einschalten von 10 bis 20 ms. Bei den Schaltungen nach Bild 6.1 erfüllt ein Kondensator von 10 µF diese Forderung.

Falls bei einem bereits arbeitendem Baustein das Reset-Signal aktiviert wird, genügt eine viel kürzere Reset-Zeit von nur zwei Maschinenzyklen, da der Oszillator ja schon stabil schwingt. Beim Sonderfall des Aufweckens aus dem Power-Down ist allerdings zu beachten, daß der der Oszillator während des Power-Down ebenfalls abgeschaltet war und jetzt neu anlaufen muß. Damit ist die längere Zeit zu berücksichtigen.

Während des Reset-Vorgangs nimmt der 80(C)515/-535 einen definierten Ausgangszustand an. Der Programmzähler und die SFR werden auf ihre Default-Werte gesetzt, so, daß der 80(C)515/-535 neu mit der Programmausführung beginnt. Für die Ports bedeutet das, daß alle auf FFH gesetzt werden. Speziell für Port P0 heißt dies aber, daß alle seine Pins floaten, da es sich um eine Open-Drain-Konfiguration handelt. Bei den MYMOS-Versionen können die Anschlüsse AN0 bis AN7 (analoger Eingangsport) nur als analoge Eingänge benutzt werden; bei den ACMOS-Versionen auch als digitale Eingänge.

Register	Inhalt	Register	Inhalt
P0 - P5	FFH	SP	07H
DPTR	0000H	PCON	000X 0000B
TCON	00H	TMOD	00H
TL0, TH0	00H	TL1, TH1	00H
TL2, TH2	00H	SCON	00H
IEN0, IEN1	00H	SBUF	XXXX XXXXB
IRCON	00H	IP0	X000 0000B
		IP1	X000 0000B
CCL1, CCH1	00H	CCEN	00H
CCL3, CCH3	00H	CCL2, CCH2	00H
T2CON	00H	CRCL, CRCH	00H
ADCON	00X0 0000B	PSW	00H
DAPR	00H	ADDAT	00H
B	00H	ACC	00H
PC	0000H	Watchdog	0000H

Tabelle 6.1: Default-Werte der SFR

Es ist wichtig zu wissen, daß der Inhalt des internen RAM vom Reset-Vorgang nicht berührt wird. Das bedeutet zum einen, daß der RAM-Inhalt nach dem Poweron-Reset vollständig undefiniert ist, daß aber nach einem Reset bei bereits laufendem Controller nichts am vorherigen RAM-Inhalt verändert wurde. Dadurch ist es möglich, nach einem Power-Down-Zustand, bei dem ja die Versorgungsspannung vorhanden war, wieder mit den Variablenwerten im RAM und den Registern R0 bis R7 weiterzuarbeiten.

6.1.2 Timing des Hardware-Reset

Der genaue Zeitablauf beim und nach dem Reset-Vorgang dürfte den Anwender nur in besonderen Fällen interssieren, dort wo z.B. Schaltungsteile synchronisiert werden sollen.

Das Bild 6.2 zeigt den zeitlichen Ablauf beim Beenden einer Reset-Sequenz und den Anlauf des Programms. Die Abtastung des Reset-Eingangs erfolgt immer zum Zeitpunkt S5P2 in jedem Maschinenzyklus. Das Bild setzt den korrekten Reset-Ablauf voraus, d.h. das Reset-Signal muß mindestens zwei Maschinenzyklen aktiv (low) gewesen sein. Ändert sich der Pegel an $\overline{RESET}$ irgendwann zwischen den Abtastpunkten, wird das Reset-Signal erst beim nächsten Abtasten inaktiv vorgefunden. Daraufhin wird die interne Reset-Sequenz beendet; im darauffolgenden Maschinenzyklus zu S5P2 wird erstmals das Signale *ALE* auf low gelegt. Falls externer Programmspeicher benutzt wird (Pin $\overline{EA}$ ist low), wurde bereits zu S5P1 die Adresse 0000H auf Port P0 und Port P2 ausgegeben. Somit ist zu erkennen, daß nach dem Deaktivieren des Reset-Eingangs bis zum ersten *ALE*-Signal eine Zeit von mindestens einem, aber höchstens zwei Maschinenzyklen vergeht.

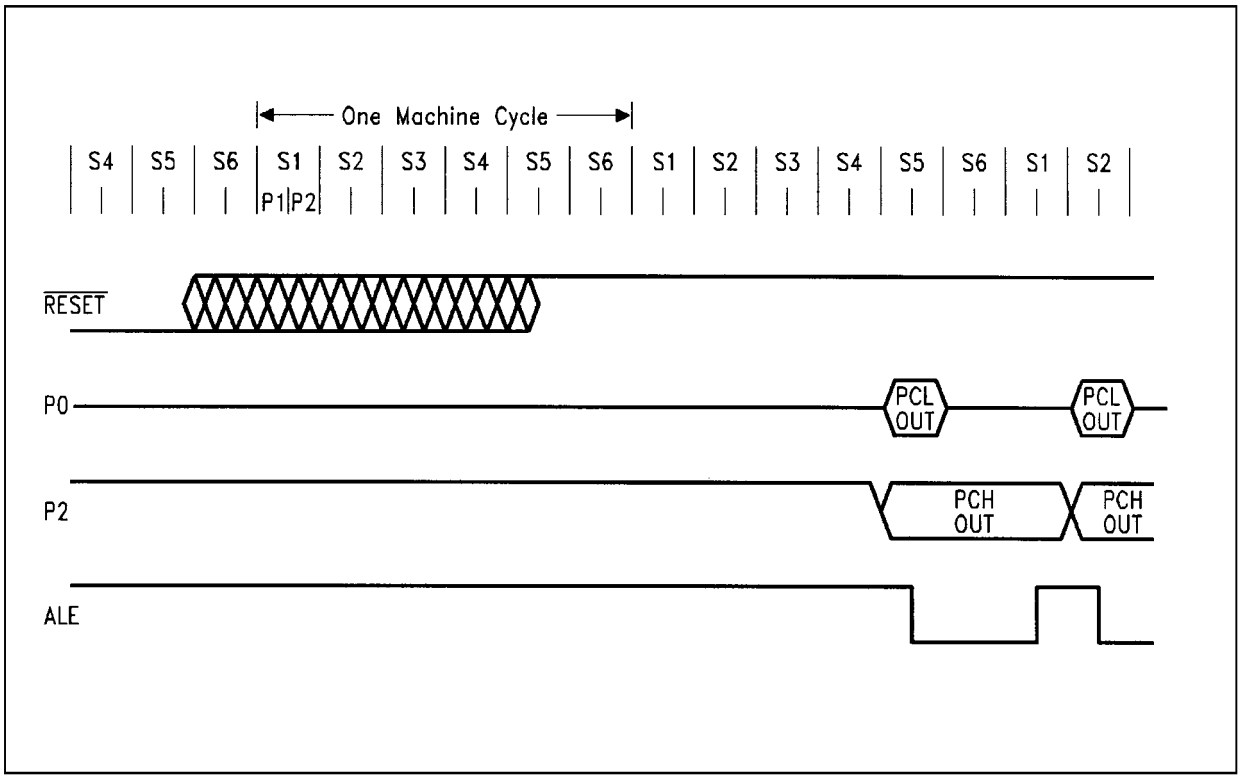


Bild 6.2: CPU-Timing nach einem Reset

7 Parallele Eingabe-Ausgabe-Ports

In diesem und in den folgenden Kapitel wird näher auf die im 80(C)515/-535 integrierten Peripheriekomponenten eingegangen. Beim 80(C)515/-535 ist vieles auf dem Baustein bereits vorhanden, was bei klassischen Mikroprozessorsystemen durch verschiedene Bausteine realisiert werden mußte. Die erste Peripheriekomponente sind die Ein-/Ausgabeports (kurz I/O-Ports genannt). Hierbei handelt es sich allgemein um byteweise zusammengefaßte Gruppen von Leitungen, an denen entweder logische Zustände von der Außenwelt in den Controller eingelesen oder logische Zustände an die Umwelt abgegeben werden können. I/O-Ports sind damit die grundlegende Möglichkeit für den Mikroprozessor oder -controller, mit der Umwelt in Verbindung zu treten; schließlich soll er ja durch sein Programm auf verschiedene Eingangssignale in gewünschter Weise reagieren, was wiederum Ausgabemöglichkeiten erfordert.

Der 80(C)515/-535 besitzt 48 bidirektionelle (in beiden Richtungen arbeitende) Ein-/Ausgabe-Anschlüsse zusammengefaßt in sechs I/O-Ports. Neben diesen grundlegenden Portfunktionen werden die Ports allerdings bei Bedarf auch von verschiedenen anderen Komponenten auf dem Baustein zu speziellen Ein- bzw. Ausgabezwecken benutzt, wenn die spezifischen Funktionen aktiviert worden sind; sie heißen dann Alternativfunktionen. In ähnlicher Weise werden auch die Ports P0 und P2 verwendet, wenn die externe Busschnittstelle benutzt wird.

Auf die Besonderheit des analog/digitalen Ports AN0 bis AN7 wird im Punkt 7.1.4 näher eingegangen.

7.1 Die Port-Strukturen

7.1.1 Digitale Ein-/Ausgabe-Ports

Der 80(C)515/-535 hat 48 einzelne digitale Portleitungen, die zu sechs je acht Bit breiten Ports zusammengefaßt sind. All diese Anschlüsse, die an einzelnen Portpins am Baustein zur Verfügung stehen, sind wahlweise als Eingang oder Ausgang für digitale Signale zu betreiben. Jeder Pin hat sein eigenes Latch (internes Speicherflipflop), sowie einen Ausgangstreiber. Die Schreib- und Lesezugriffe auf diese Ports werden über die SFR P0 bis P5 durchgeführt. Die Ein-/Ausgabe-Ports sind damit genauso wie die anderen SFR ansprechbar; besondere I/O-Befehle (oder einen eigenen I/O-Adressraum) wie von Mikroprogzessoren bekannt, gibt es nicht. Vielmehr steht hier der volle Umfang des Befehlssatzes auch für Port-Operationen zur Verfügung. Besonders bemerkenswert in diesem Zusammenhang sind die zahlreichen Bitbefehle, die einen einfachen Umgang auch mit einzelnen Portpins ermöglichen. Besonders bei Steuerungsaufgaben ist dies wichtig, da man hier oft mit einzelnen Pins, z.B zur Abfrage von Schaltern, zur Steuerung von Stellgliedern usw. umgehen muß.

MSB							LSB
P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
87H	86H	85H	84H	83H	82H	81H	80H
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
97H	96H	95H	94H	93H	92H	91H	90H
P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
0A7H	0A6H	0A5H	0A4H	0A3H	0A2H	0A1H	0A0H
P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
0B7H	0B6H	0B5H	0B4H	0B3H	0B2H	0B1H	0B0H
P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
0EFH	0EEH	0EDH	0ECH	0EBH	0EAH	0E9H	0E8H
P5.7	P5.6	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0
0FFH	0FEH	0FDH	0FCH	0FBH	0FAH	0F9H	0F8H

Tabelle 7.1: SFR der I/O-Ports

Eine bereits erwähnte Sonderfunktion der Ports P0 und P2 des 80(C)515/-535 bei der Verwendung eines externen Busses wurde bereits in Kapitel 5 ausführlich beschrieben. Im Abschnitt 7.2 werden die elektrischen Eigenschaften dieser Ports bei Busbetrieb näher erläutert.

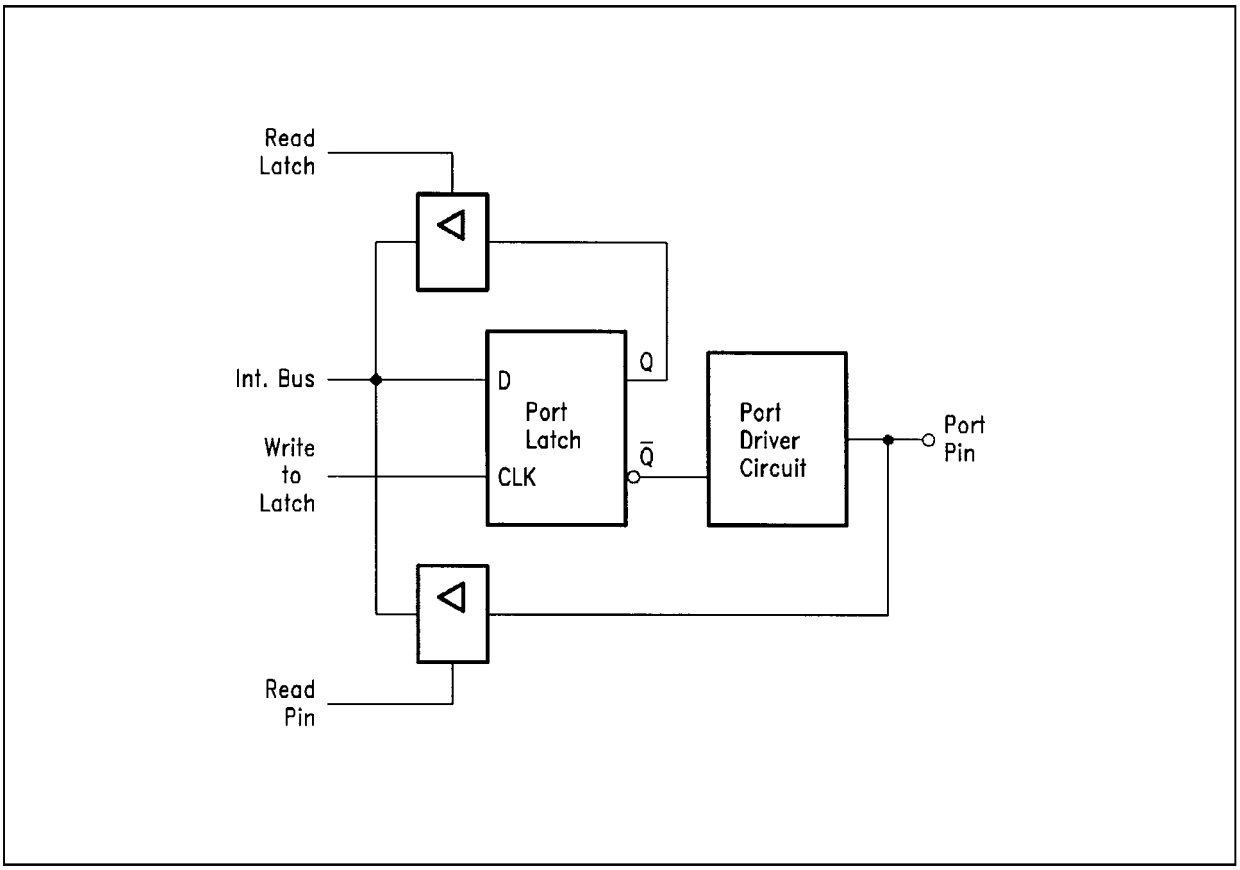


Bild 7.1: Grundschaltung eines Ports

Im Bild 7.1 ist eine Blockschaltbild zur allgemeinen Struktur aller I/O-Ports abgebildet. Es wird nur ein einzelnes Bit eines Ports gezeigt; diese Struktur ist also pro Port achtmal vorhanden. Der Kern ist das D-Port-Latch. Bei einem Schreibbefehl in das entsprechende Portregister P0 bis P5 (z.B: MOV P3, #55H) übernimmt jedes der D-Flipflops den ihm zugewiesenen Wert durch Betätigen der internen CLK-Leitung. Der Wert bleibt bis zum nächsten Schreibbefehl erhalten. Der Wert des nicht invertierten Ausgangs des Flipflops kann über den Read-Latch-Treiber wieder zurückgelesen werden. Der invertierte Ausgang dient zur Ansteuerung des Ausgangstreibers für den Portpin. Der Wert am Portpin kann ebenfalls durch eine eigene Treiberschaltung (Read Pin) gelesen werden.

Achtung: Verschiedene Befehle lesen den Portpin direkt, andere nur das Port-Latch. Eine Tabelle der sogenannten „Read-Modify-Write-Befehle“ (sie lesen immer das Latch) ist in Punkt 7.4.3 zu finden.

Im Bild 7.1 ist Porttreiberschaltung sehr allgemein dargestellt. Das Bild 7.2 zeigt nun Details für die Ports P1 bis P5. Allen gemeinsam ist, daß sie sowohl als Eingabe- oder als Ausgabeport betrieben werden können, sogar jedes einzelne Bit.

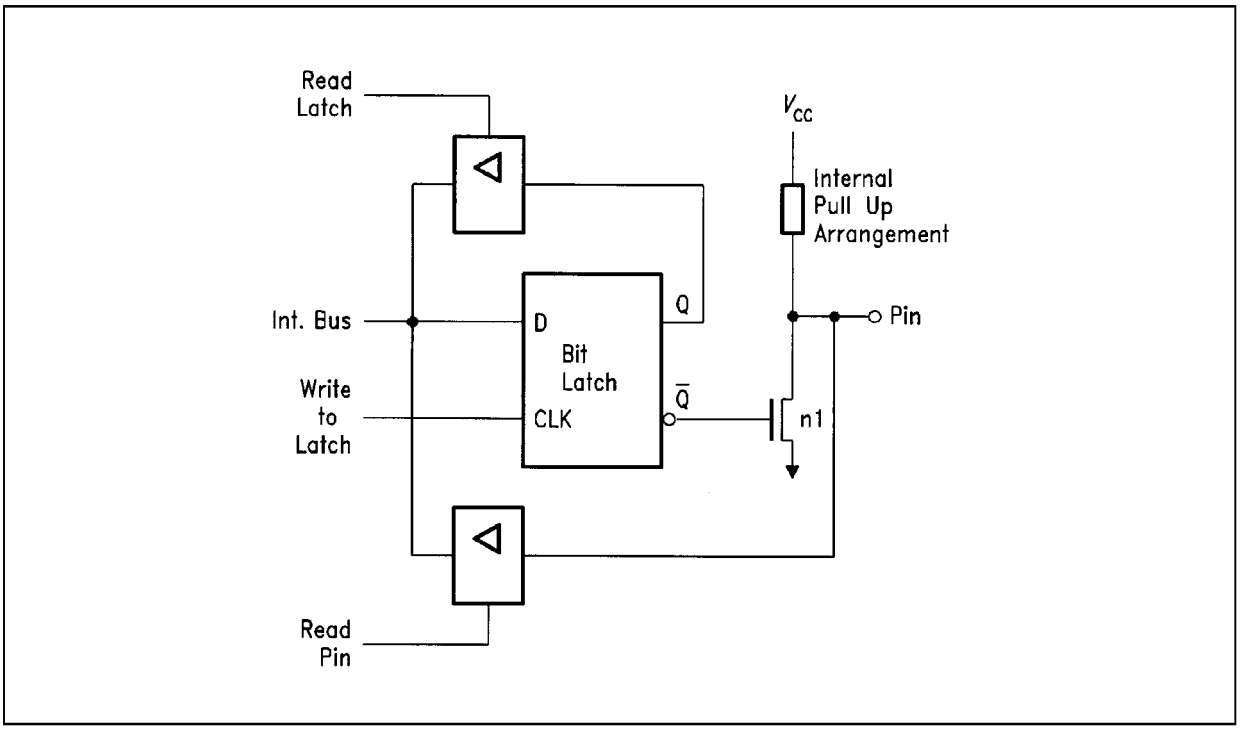


Bild 7.2: Grundschaltung der Ausgangstreiber der Ports P1 bis P5

Die Struktur im Ausgangstreiber benutzt einen starken N-Kanal-Feldeffekttransistor (FET) n_1 zur Erzeugung des Low-Pegels. Der High-Pegel wird durch einen aus mehreren Transistoren bestehendes Gebilde, das wie ein Pull-Up-Widerstand wirkt, erzeugt. Wenn sich im Port-Latch eine 0 befindet, ist der Ausgang $\overline{Q}$ auf 1; damit wird der FET n_1 leitend und zieht den Portpin auf Low-Pegel. Wenn sich im Port-Latch eine 1 befindet, sperrt der FET und der Pull-Up-Widerstand zieht den Portpin auf eine „schwache“ 1 (High-Pegel). Somit ist die Ausgabe einer logischen 0 oder 1 am Pin gewährleistet.

Durch die „schwache“ Eins ist es aber möglich, den Portpin von außen her auf Low-Pegel zu ziehen, während im Port-Latch eine 1 steht. Genau dies wird getan, wenn der Pin als Eingang konfiguriert ist. Die Pull-Up-Schaltung kann bei externer Beschaltung problemlos auf Low-Pegel gezogen werden. Jetzt unterscheidet sich Inhalt des Port-Latch vom tatsächlichen Pinzustand. Deswegen kann sowohl direkt vom Pin als auch aus dem Latch gelesen werden.

Diese Konstellation erlaubt es beim 80(C)515/-535 jedes Portbit wahlweise als Ein- oder Ausgang zu definieren (quasi-bidirektionelles Verhalten). Hierzu muß nur eine 1 bei den Pins im Latch stehen, die als Eingang verwendet werden sollen. Somit ist es möglich, Ein- und Ausgänge beliebig zu mischen (auch innerhalb eines Ports).

Die bisher gobal beschriebene Funktion der Ausgangstreiberschaltung soll jetzt aufgeschlüsselt und beschrieben werden. Hierzu ist eine Unterscheidung zwischen den MYMOS- und ACMOS-Versionen zu treffen. Das Bild 7.3 zeigt im oberen Teil die Treiberschaltung eines MYMOS-Typs, im unteren Teil die eines ACMOS-Typen.

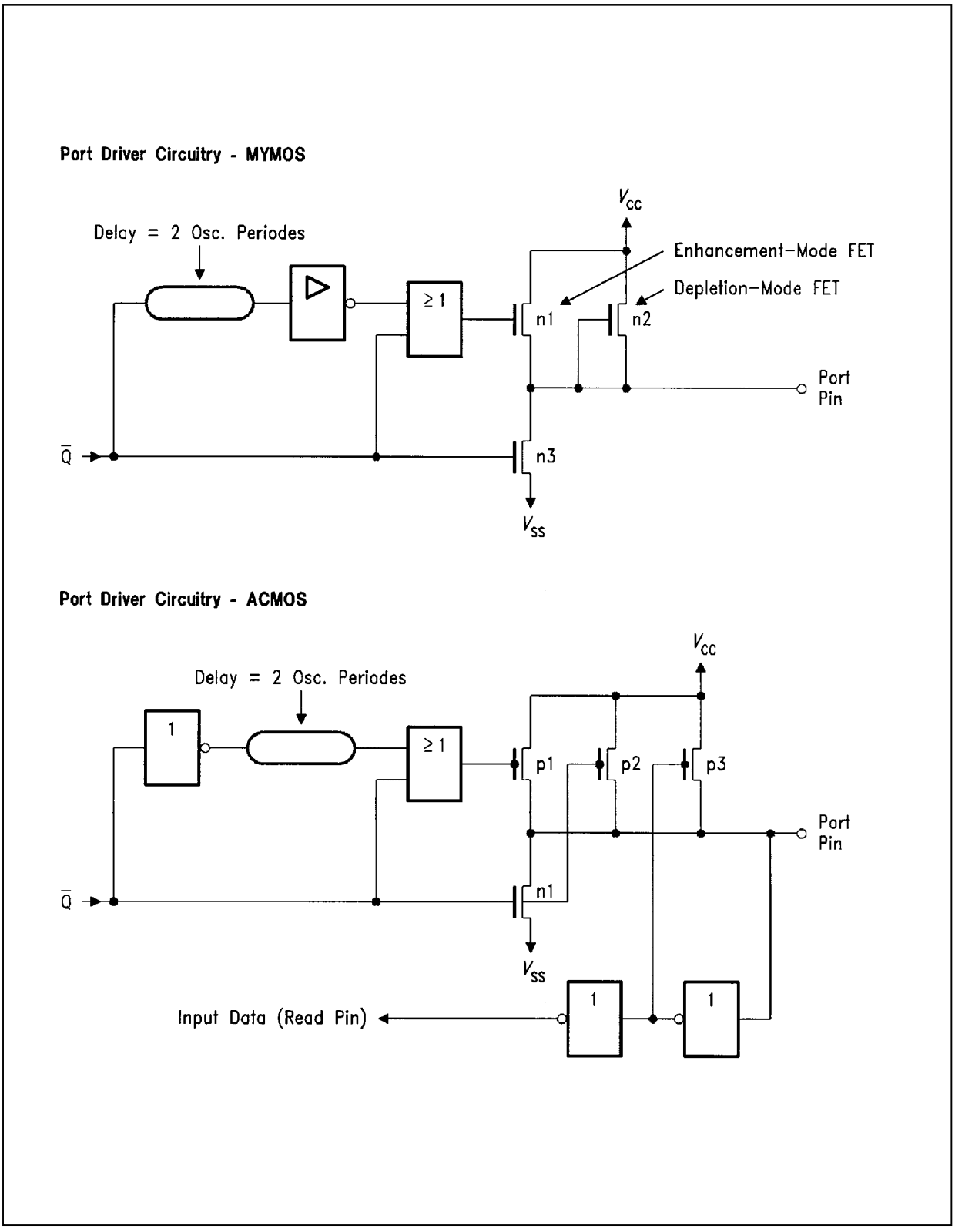


Bild 7.3: Ausgangstreiber der Ports P1 bis P5

7.1.2 Ausgangstreiber bei MYMOS-Typen

Der Ausgangstreiber der MYMOS-Version besteht aus den beiden FET's n_1 und n_2, die als Pull-Up-Schaltung arbeiten und dem Pull-Down-FET n_3. Dabei ist der Transistor n_1 ein Anreicherungstyp, der nur für zwei Oszillatorperioden leitend wird, wenn ein 0-nach-1-Übergang für diese Portbit auftritt. Der FET n_1 kann mit großen Strömen beaufschlagt werden um schnell auf den Wechsel reagieren zu können.

Der Transistor n_2 ist bedeutend schwächer ausgelegt (Verarmungstyp) und permanent leitend. Wenn der Portpin auf Low-Pegel gezogen wird (z.B. bei der Verwendung als Eingang) liefert n_2 einen schwachen Strom. Den Wert dieses Stromes, der für die Dimensionierung externer Schaltungen benötigt wird, kann der Anwender im Datenblatt als Parameter I_{IL} nachlesen.

Der Transistor n_3 ist ein kräftiger Pull-Down-Transistor, der leitend ist, wenn eine 0 im Port-Latch programmiert wurde. Der Transistor kann sehr hohe Ströme aufnehmen; siehe im Datenblatt den Parameter I_{OL}. Ein Kurzschluß zur Betriebsspannung muß aber vermieden werden, da wegen des hohen Stromes der Transistor zerstört würde.

7.1.3 Ausgangstreiber bei ACMOS-Typen

Hier setzt sich die Pull-Up-Schaltung aus drei FET's (p_1 bis p_3) zusammen. Für die Erzeugung des Low-Pegels ist der FET n_1 zuständig.

Der FET n_1 ist ein N-Kanal-Anreicherungstransistor. Er ist leitend, wenn an Ausgang $\overline{Q}$ des Latch's High-Pegel ansteht. Er ist so dimensioniert, daß er hohe Ströme schalten kann. Er wird nur leitend, wenn eine 0 im zugehörigen Latch gespeichert ist. In diesem Zustand ist ein Kurzschluß zur Betriebsspannung zu vermeiden, da der fließende Strom den Transisitor zerstören würde.

Der FET p_1 ist ein P-Kanal-Anreicherungstransistor. Er verhält sich polaritätsmäßig genau umgekehrt wie der gerade beschriebene FET n_1. Dieser Transistor wird nur für die Dauer von zwei Oszillatorperioden aktiviert, wenn ein 0-nach-1-Übergang am Portpin erzeugt werden soll. Wird eine 1 auf schon vorhandene 1 geschrieben, wird p_1 nicht aktiviert. Die Treiberleistung des Transistors ist ähnlich hoch wie die des n_1 und dient dazu, dem Portausgang kurze Anstiegszeiten zu ermöglichen, besonders wenn kapazitive Lasten angeschlossen sind.

Der FET p_2 ist ebenfalls ein P-Kanal-FET. Im Gegensatz zu p_1 ist er allerdings schwach dimensioniert. Er ist immer dann aktiviert, wenn sich eine 1 im Port-Latch befindet. Es ist problemlos, diesen Transistor dauernd nach Masse kurzzuschließen; genau das wird gemacht, wenn der Portpin als Eingang benutzt wird.

Der FET p_3 ist ebenfalls ein P-Kanal-Typ. Er ist dann leitend, wenn die Spannung am Portpin höher als etwa 1,0 bis 1,5 V ist. Damit unterstützt er den Transistor p_2 in seiner Pull-Up-Wirkung, solange die Spannung am Pin die oben erwähnte Schwelle nicht unterschreitet. Der Strom, der über beide Transistoren gemeinsam zur Masse abfließen kann, ist deutlich größer als der, den p_2 alleine aufbringen kann. Der Sinn dieser Schaltung liegt darin, daß beim Betrieb des

Ports als Ausgang auch für den High-Pegel eine etwas höhere Treiberleistung zur Verfügung steht.

Die obige Beschreibung trifft wie gesagt nur auf die I/O-Ports P1 bis P5 zu. Port P0 weist einige Besonderheiten auf, wie Bild 7.4 a) zeigt.

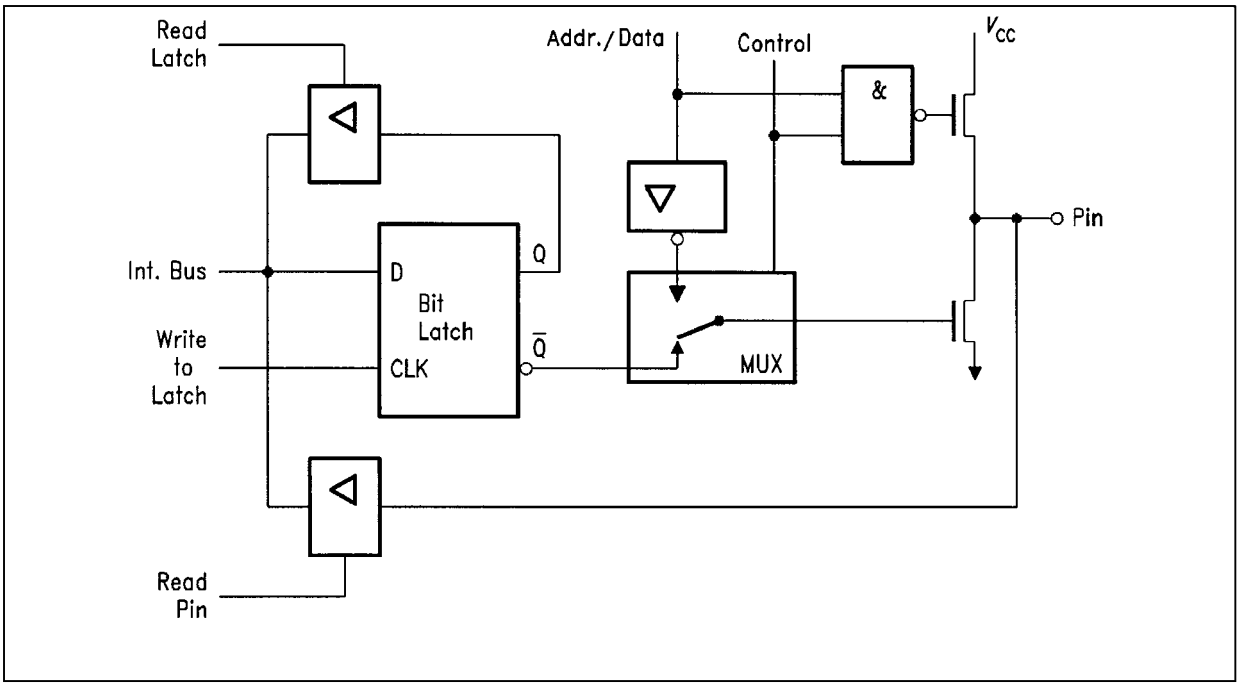


Bild 7.4 a): Schaltung für Port P0

Im Gegensatz zu den anderen Ports wird hier streng zwischen I/O-Betrieb und Betrieb als Daten- und Adreßbus unterschieden. Der eingezeichnete Multiplexer (MUX) wird dazu von einem internen Steuersignal umgeschaltet, das immer aktiv wird, wenn ein externer Buszugriff erfolgen soll. Ist dieses Signal nicht aktiv (somit I/O-Betrieb) ist keinerlei Pull-Up aktiv. Damit liegt ein echter bidirektionaler Port vor, da er als Eingang (1 im Latch) völlig hochohmig ist. Soll Port P0 dagegen als Ausgang benutzt werden, muß der High-Pegel durch externe Pull-Up-Widerstände erzeugt werden. Auch der Port P2 weist bei der Benutzung als Adreßbus Besonderheiten auf, die im Abschnitt 7.2 angesprochen werden.

7.1.4 Analog/Digitaler Eingangsport

Bei den MYMOS-Versionen können die Anschlüsse AN0 bis AN7 ausschließlich als analoge Eingänge benutzt werden. Im Gegensatz dazu besteht bei den ACMOS-Typen 80C515/-535 auch die Möglichkeit, sie als digitale Eingänge zu verwenden. In diesem Fall werden sie als zusätzlicher Port P6 über das SFR P6 (0DBH) angesprochen. Da dieser Port P6 keine internen Latchs besitzt, ist ein Schreibvorgang auf den Port ohne Wirkung.

Primär sind die Anschlüsse AN0 bis AN7 als Eingänge für den integrierten A/D-Wandler vorgesehen. Wenn eine analoge Eingangsspannung gemessen werden soll, muß der entsprechende Eingangskanal durch Programmierung des Registers ADCON selektiert werden.

Hierbei ist zu beachten, daß der Port selbst zwar hochohmig ist, der angeschlossene A/D-Wandler aber während der Samplingzeit eine kapazitive Last darstellt.

7.2 Port P0 und Port P2 als Daten- und Adreßbus

Die Abbildungen 7.4 a) und 7.4 b) zeigen die Strukturen der Ports P0 und P2. Die in der Struktur vorhandenen Multiplexer (MUX) werden durch interne Steuersignale umgeschaltet, wenn die Ports für externe Buszugriffe benutzt werden sollen. Dies geschieht, wie im Kapitel 5 beschrieben, in Abhängigkeit vom Pin $\overline{EA}$ und der PC-Adresse; außerdem aktivieren die MOVX-Befehle den Bus. Bei Verwendung als externer Bus sind mit den Ports P0 und P2 keine normalen I/O-Operationen möglich; dies trifft auch auf Port P2 zu, selbst dann wenn nicht alle Pins als Adreßleitungen gebraucht werden.

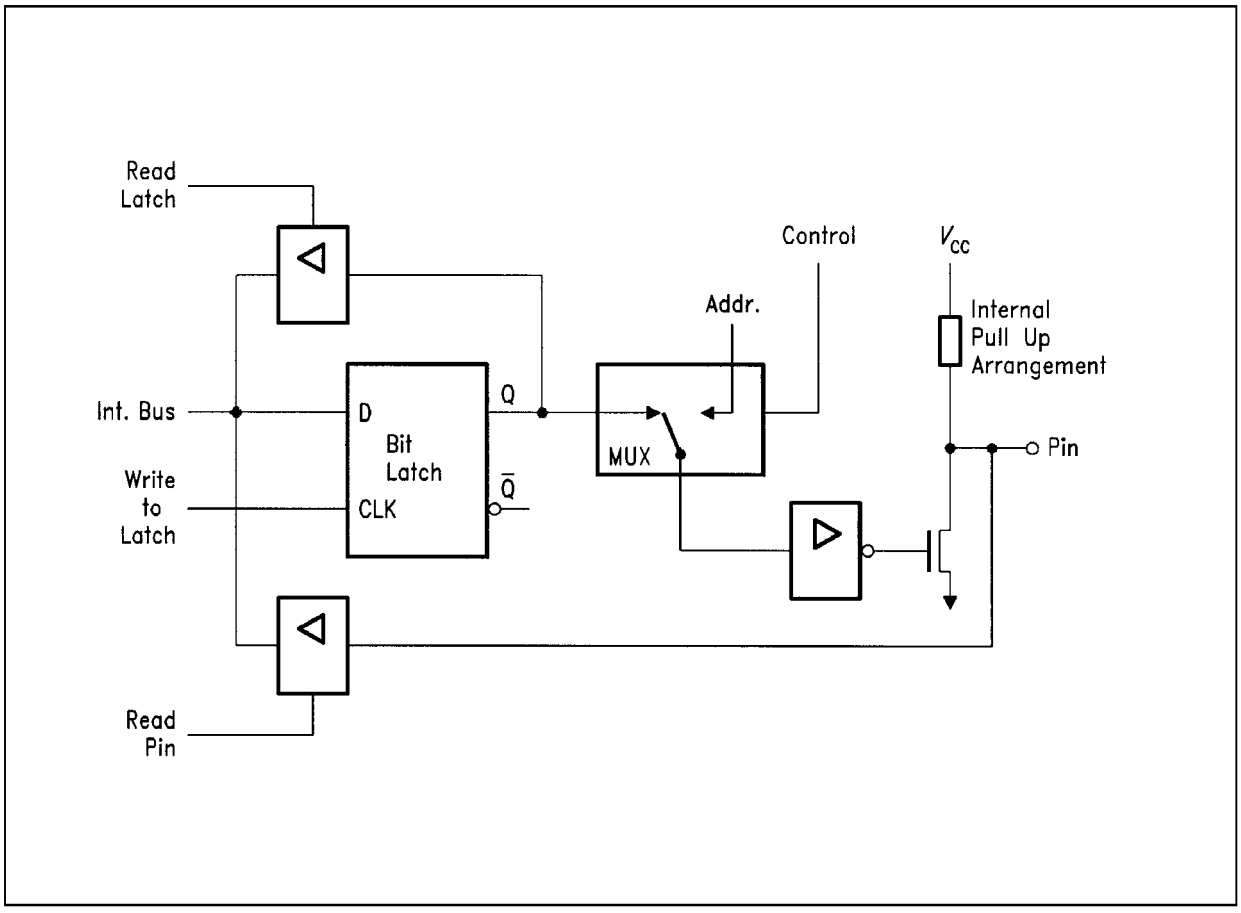


Bild 7.4 b): Schaltung für Port 2

Beim Umschalten auf den Busbetrieb haben die internen Latches keinen Einfluß mehr auf die Treiberschaltung. Diese werden stattdessen durch die Adreß- und Datensignale direkt angesteuert oder (bei Port P0) zum Einlesen der Daten hochohmig gesteuert. Beim Port P0 steht in dieser Betriebsart ein starker Pull-Up-FET zur Verfügung. Somit sind bei Busbetrieb keine externen Pull-Up-Widerstände am Port P0 nötig. Im Port P2 wird im Busbetrieb generell der

Pull-Up-Transistor p_1 aktiviert, wenn High-Pegel benötigt wird. Somit steht auch hier genügend Treiberleistung zur Verfügung.

Die Port-Latches werden im Port P0 durch jeden externen Buszugriff auf FFH gesetzt. Im Port P2 bleibt der Latch-Inhalt unverändert, wenn ein Buszugriff durchgeführt wird.

7.3 Alternative Port-Funktionen

Alle Pins der Ports P1 und P3 haben über die normale I/O-Funktion eine Erweiterung, um bei Bedarf für die auf dem Chip integrierten Peripheriekomponenten ebenfalls Ein- oder Ausgabefunktionen zur Verfügung zu stellen. Man nennt dies alternative Funktionen (alternate function). Welcher Pin für welche Alternativfunktion verwendet wird, listet folgende Tabelle auf.

Port	Pin	Alternativfunktion
P1.0	$\overline{INT3}$/CC0	Ext. Interrupt 3 Eingang, Compare 0 Ausgang, Capture 0 Eingang
P1.1	INT4/CC1	Ext. Interrupt 4 Eingang, Compare 1 Ausgang, Capture 1 Eingang
P1.2	INT5/CC2	Ext. Interrupt 5 Eingang, Compare 2 Ausgang, Capture 2 Eingang
P1.2	INT6/CC3	Ext. Interrupt 6 Eingang, Compare 3 Ausgang, Capture 3 Eingang
P1.4	$\overline{INT2}$	Ext. Interrupt 2 Eingang
P1.5	T2EX	Timer 2 external reload trigger input
P1.6	CLKOUT	System clock output
P1.7	T2	Timer 2 external reload trigger input
P3.0	RxD	Serieller Dateneingang (asynchron) oder Datenein/-ausgang (synchron)
P3.1	TxD	Serieller Datenausgang (asynchron) oder Taktausgang (synchron)
P3.2	$\overline{INT0}$	Ext. Interrupt 0 Eingang, Timer 0 Steuereingang
P3.3	$\overline{INT1}$	Ext. Interrupt 1 Eingang, Timer1 Steuereingang
P3.4	T0	Ext. Eingang für Timer 0
P3.5	T1	Ext. Eingang für Timer 1
P3.6	$\overline{WR}$	Bussteuersignal „Lesen“ bei externem Datenspeicher
P3.7	$\overline{RD}$	Bussteuersignal „Schreiben“ bei externem Datenspeicher

Tabelle 7.2: Alternativfunktionen der Ports P1 und P3

Den Aufbau der betroffenen Ports zeigt Bild 7.5. Es handelt sich hier nur um eine Erweiterung der oben vorgestellten Portstruktur; es wird lediglich ein NAND-Gatter in den Ansteuerpfad für den Ausgangstreiber eingefügt. Der eine Eingang des Gatters ist wie gewohnt der Ausgang des Port-Latches. Der andere Eingang wird aus dem Ausgangssignal der jeweiligen Peripheriekomponente gespeist. Benötigt die Peropheriekomponente ein Eingangssignal, wird dies direkt hinter der Eingangsschaltung abgegriffen.

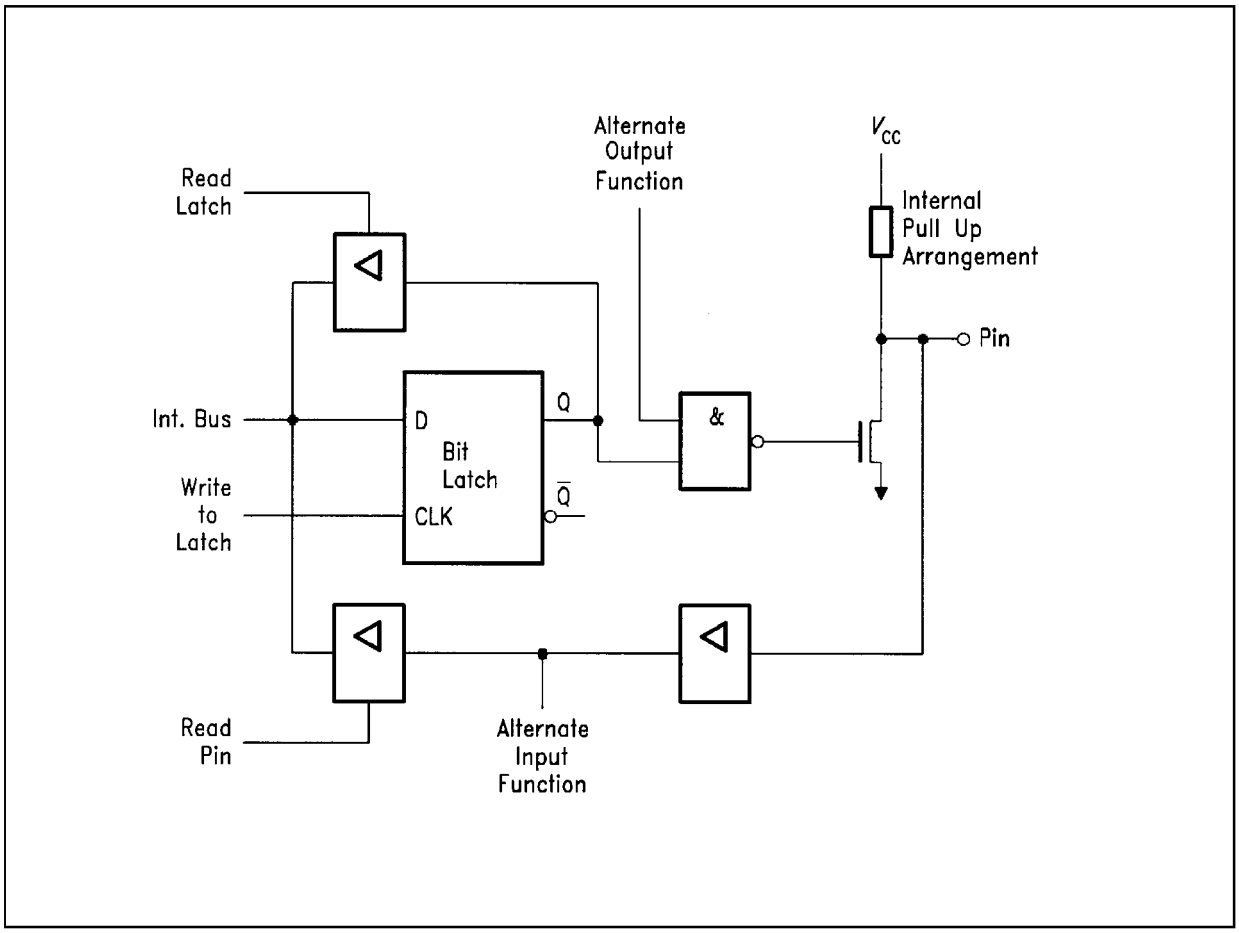


Bild 7.5: Schaltung der Ports P1 und P3

Solange die Peripheriekomponente nicht aktiv ist, ist ihr Ausgangssignal auf High-Pegel, somit steuert das Port-Latch wie gehabt die Ausgangsstruktur. Wird durch entsprechende Programmierung die Peripheriekomponente eingeschaltet, wird sie ihre Ausgangssignale ebenfalls über das NAND-Gatter an die Treiber schicken können, solange das Port-Latch eine 1 enthält. Somit wird klar, daß keine gesonderte Unterprogrammierung des Ports für die Alternativfunktion nötig ist. Es muß nur eine 1 im Port-Latch sein, wenn die betreffende Peripheriekomponente aktiviert wird, andernfalls bleibt der Pin dauernd auf 0. Die 1 im Port-Latch ist übrigens auch der Zustand nach dem Reset. Damit ist die Alternativfunktion also ohne weitere Porgrammierung des Ports möglich. Eine Besonderheit weisen sie Alternativfunktionen der Compare-Ausgangspins auf. Hier muß nicht zwingend eine 1 im Latch sein. Diese Peripheriefunktionen programmieren das Latch selbst um.

Keine Kollision ist beim Eingangssignal möglich. Hier können sowohl die Peripheriekomponente als auch die normale Eingangsfunktion bedient werden; sogar gleichzeitig.

7.4 Verwendung der Ports

7.4.1 Zeitverhalten

Wenn ein Befehl abläuft, der den Wert im Port-Latch ändert (z.B. MOV P1,#27H), so wird der neue Wert zum Zeitpunkt S6P2 des letzten Zyklus des Befehls in das Latch eingeschrieben. Die Übernahme in die Teiberschaltung erfolgt dann im nächsten P1-Takt. Somit wird der neue Wert am Pin zu S1P1 des nächsten Befehls erscheinen. Diese Zeitfunktion beschreibt nur die durch normale Ausgabefunktionen erzeugten Signale. Im Falle der alternativen Funktionen von Peripheriefunktionen sind die Verhältnisse anders. Dies wird in den entsprechenden Kapiteln behandelt.

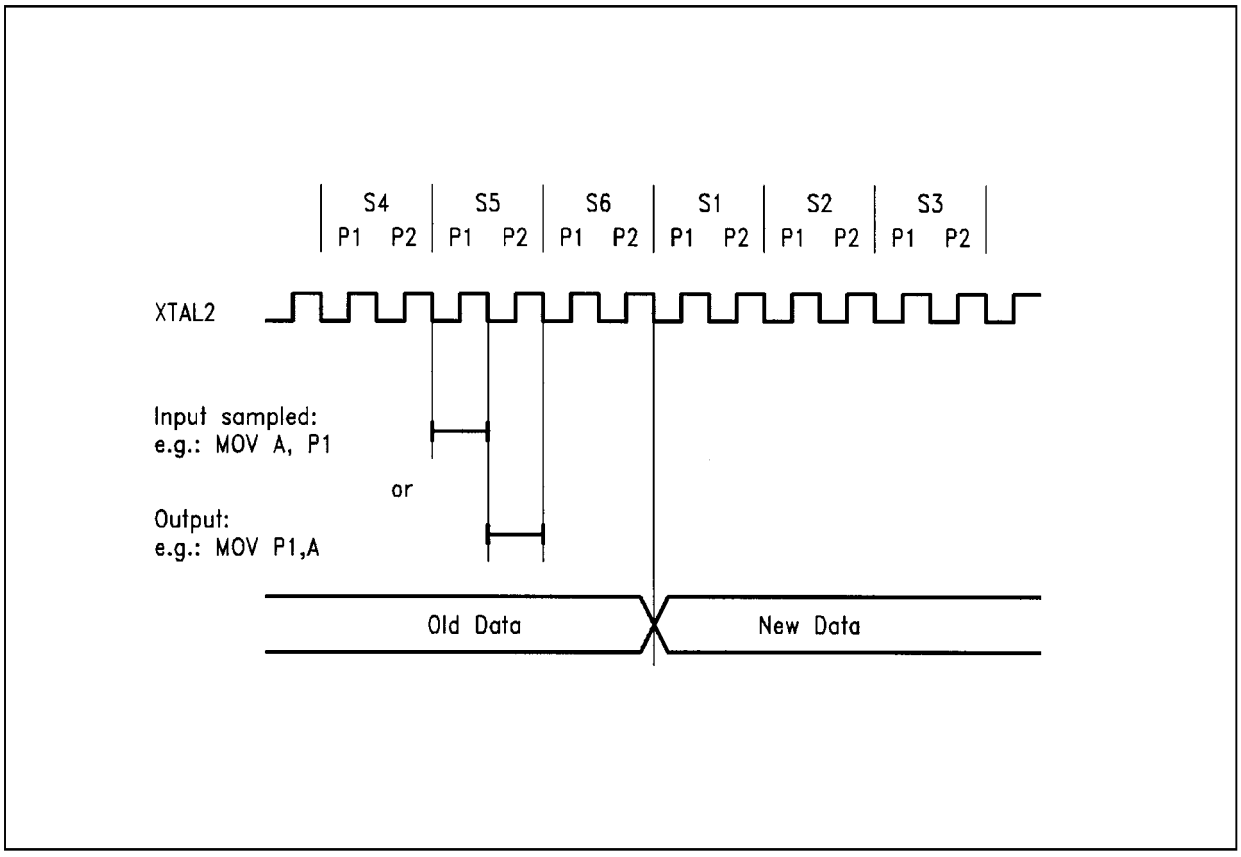


Bild 7.6: Port-Timing

Wenn ein Befehl abläuft, der einen Wert von einem Port einliest (z.B. MOV A,P1), wird der Portpin einmal pro Maschinenzyklus abgetastet. Je nach Portpin geschieht dies zum Zeitpunkt S5P1 oder S5P2. Die einmalige Abtastung pro Maschinenzyklus wird übrigens generell verwendet, um Flanken an den Pins zu erkennen. Eine Flanke gilt dann als erkannt, wenn zwei aufeinderfolgende Abtastungen unterschiedliche Werte geliefert haben. Daran erkennt man auch den Grund für die Forderung, daß z.B. Eingangsimpulse, die als Alternativfunktion für einen Counter benötigt werden, nie kürzer als ein Maschinenzyklus sein dürfen. Andernfalls ist es nicht sicher, daß der Impuls mindestens einmal abgetastet und damit erkannt wird.

7.4.2 Elektrisches Verhalten

Die Ein- und Ausgangspegel der Ports sind so spezifiziert, daß sowohl TTL- als auch CMOS-Bausteine angeschlossen werden können. Werden die Ports zur Ausgabe verwendet, beschreiben die Parameter V_{OL} und V_{OH} mit den zugehörigen Testbedingungen die Pegel und die dabei zur Verfügung stehenden Ströme. Wie bereits erwähnt, hat der Port P0 im I/O-Betrieb keine internen Pull-Up-Schaltungen. Somit müssen hier die High-Pegel durch externe Widerstände erzeugt werden. Dies gilt nicht für den Busbetrieb.

Wenn die Ports als Eingänge benutzt werden, müssen bei den Ports P1 bis P5 durch die externen Ansteuerschaltungen die Ströme I_{TL} und I_{IL} aufgenommen werden, falls Low-Pegel anliegt. Der Grund sind die in diesen Ports integrierten Pull-Up-Schaltungen. Die exterenen Schaltungen müssen dementsprechend dimensioniert sein, daß sie auch wirklich in der Lage sind, den Low-Pegel zu erzwingen. Der Port P0 und der Nur-Eingangsport P6 der ACMOS-Versionen sind dagegen hochohmig und stellen somit keine besonderen Ansprüche an die Treiberleistungen der externen Schaltungen. Die jeweils gültigen Werte und die Parameter sind im Datenblatt des Bausteins zu finden.

7.4.3 Read-Modify-Write-Funktion der Ports P0 bis P5

Dieser Punkt behandelt eine auf den ersten Blick etwas eigenartige Unterscheidung in der Funktion, die bei Lesevorgängen aus den Portregistern vorgenommen wird. Bei allen I/O-Ports besteht die Möglichkeit, entweder der aktuellen Pegel am zugehörigen Pin oder aber den Inhalt des Port-Latches zu lesen (siehe Bild 7.1). Wie bereits beschrieben, besteht die Möglichkeit, daß verschiedene Werte am Pin und im Latch anzutreffen sind. Dies ist z.B. dann der Fall, wenn der Pin als Eingang benutzt werden soll. Dann muß eine 1 im Latch stehen, aber extern kann Low-Pegel anliegen.

Die Unterscheidung zwischen dem Lesen am Pin oder aus dem Latch wird nun folgendermaßen getroffen: Die sogenannten Read-Modify-Write-Befehle in Verbindung mit einer Portadresse lesen immer aus dem Port-Latch, alle anderen Befehle lesen immer vom Pin. In der folgenden Tabelle sind alle Read-Modify-Write-Befehle aufgeführt. Innerhalb des Befehlsablaufs wird zunächst der Wert des adressierten Registers eingelesen (Read), dann in befehlsspezifischer Weise verändert (Modify) und zuletzt wieder in dasselbe Register zurückgeschrieben (Write). Ein typischer Vertreter dieser Art ist ANL P1,A; hier wird der Inhalt der acht Port-Latches des Port P1 eingelesen, dieses Byte wird anschließend mit dem Inhalt des Akku bitweise logisch UND-verknüpft und das Ergebnis wieder in die Latches des Ports P1 eingeschrieben. Ein Befehl, der direkt vom Pin liest, ist z.B. MOV A,P3; hier werden die acht Pins vom Port P3 gelesen und der Wert in den Akku gebracht.

Befehl	Funktion	Beispiel
ANL	Logische UND-Verknüpfung	ANL P1,A
ORL	Logische ODER-Verknüpfung	ORL P4,#0FCH
XRL	Logische EXOR-Verknüpfung	XRL P2,@R1
JBC	Verzweigung, wenn das adressiert Bit gesetzt ist, dann Löschen des Bits	JBC P5.3,LABEL_2
CPL	Komplementieren des adressierten Bits	CPL P1.3
INC	Inkrementieren eines Bytes	INC P4
DEC	Dekrementieren eines Bytes	DEC P4
DJNZ	Dekrementieren und Verzweigen, wenn ≠ 0	DJNZ P0,LABEL_1
MOV Px.y,C	Transfer des Carry-Bits zu Bit y in Port x	MOV P3.5,C
CLR Px.y	Löschen von Bit y in Port x	CLR P0.7
SETB Px.y	Setzen von Bit y in Port x	SETB P2.4

Tabelle 7.3: Read-Modify-Write-Befehle

Der Grund für diese unterschiedliche Behandlung der Lesevorgänge liegt in der (quasi-) bedirektionellen Struktur der Ports. Wie bereits gesagt, können Pinpegel und Latchinhalt unterschiedlich sein. Dies ist der Fall, wenn ein Pin als Eingang arbeitet und eine 0 anliegt. Angenommen, andere Pins des Ports sollen als Ausgang verwendet werden und auf bestimmte Werte programmiert sein; dies z.B. mit ANL- oder ORL-Befehlen. Würde nun ein solcher Befehl alle acht Bits des Ports von den Pins lesen, würden auch die Eingänge als 0 gelesen werden. Diese 0 würde dann nach der logischen Verknüpfung, die an diesen Pins nichts ändern soll, als 0 in das Latch zurückgeschrieben. Damit wäre dann die Eingangsfunktion für diesen Portpin nicht mehr möglich. Dagegen liest der Befehl aus dem Latch, findet dort die 1 für den Eingangsbetrieb und schreibt sie wieder zurück.

Aus genau dem selben Grund gehören auch die letzten drei Befehle aus der Tabelle zu den Read-Modify-Write-Befehlen. Diese Befehle nehmen eine bitweise Manipulation vor. Da intern aber auch hier alle acht Bits des entsprechenden Bytes eingelesen werden, dann das eigentlich adressierte Bit behandelt und schließlich das ganze Byte zurückgeschrieben wird, liegt hier derselbe Vorgang wie oben beschrieben zu Grunde.

8 Die serielle Schnittstelle

Der 80(C)515/-535 besitzt eine serielle Schnittstelle, die den Datenaustausch mit anderen Mikrocontrollern oder allgemein mit seiner Umwelt erlaubt. Die Schnittstelle ist voll-duplexfähig, was heißt, sie kann gleichzeitig senden und empfangen. Ferner ist sie empfangsgepuffert, d.h. es kann mit dem Empfang eines weiteren Bytes begonnen werden, während das vohergehende noch auf die Abholung durch die CPU wartet. Allerdings ist das zweite Byte verloren, wenn das erste noch nicht aus dem Empfangsregister gelesen wurde, bis das zweite komplett empfangen ist. Die serielle Schnittstelle des 80(C)515/-535 ist vollständig kompatibel mit der des 80(C)51.

8.1 Die Betriebsarten

Die serielle Schnittstelle arbeitet in vier Betriebsarten: Ein synchroner Modus (Modus 0) und drei asynchrone Modi (Modus 1 bis 3). Die Baudrate wird entweder direkt vom Oszillatortakt abgeleitet (Modus 0 und 2), über den Timer 1 oder aus einem speziellen Baudratengenerator gewonnen (Modus 1 oder 3).

Im Anschluß folgt eine grobe Beschreibung der verschiedenen Betriebsarten. Eine detaillierte Diskussion der Modi erfolgt in Abschnitt 4.

Die serielle Schnittstelle hat zwei Special Function Register, über die die Bedienung erfolgt. Die Programmierung und Steuerung, sowie Statusanzeigen befinden sich in dem Steuerregister SCON (98H). Das Register für die Sende- und Empfangsdaten ist SBUF (99H); dieses SFR bedient in der Hardware zwei unterschiedliche Register: Das Sende- und das Empfangsregister. Die Unterscheidung erfolgt durch Lesen oder Schreiben auf SBUF: Lesen adressiert das Empfangsregister, während Schreiben das Senderegister betrifft. Außerdem wird beim Beschreiben dieses Registers die Aussendung des eingeschriebenen Bytes (im vorher gewählten Modus) gestartet. Der Empfangsvorgang wird abhängig vom Modus unterschiedlich ausgelöst. In den asynchronen Modi beginnt der Empfang automatisch, wenn das Startbit erkannt wird. Das neunte Bit bei den Modi 2 und 3 befindet sich im Register SCON: Das neunte Sendebit ist TB8, das neunte Empfangsbit ist RB8.

Die Modi werden durch durch Bits SM0 und SM1 im SFR SCON gewählt. Das zusätzliche Steuerbit SM2 dient zur Aktivierung einer besonderen Eigenschaft, nämlich der Multiprozessor-Kommunikation.

Die Schnittstelle stellt auch Interrupt-Anforderungen bereit, wenn entweder ein Byte empfangen wurde oder eine Aussendung beendet wurde. Die Interruptrequest-Flags sind RI für den Empfang und TI für das Aussenden. Diese Flags können auch durch das Programm im Pollingverfahren abgefragt werden, wenn dieser Interrupt aus irgendwelchen Gründen nicht verwendet werden soll.

8.1.1 Modus 0 - Schieberegister-Modus (synchroner Modus)

In diesem Modus stellt die Schnittstelle den Takt für die Übertragung an einem getrennten Pin zur Verfügung. Es werden acht Datenbits entweder ausgegeben oder eingelesen, wobei das Least Significant Bit (LSB) zuerst übertragen wird. Die Daten werden am Pin *RxD* ausgegeben oder empfangen. Der zugehörige Schiebetakt erscheint am Pin *TxD*. Die verwendete Baudrate ist immer $f_{osz}/12$. Dieser Modus eignet sich gut dazu, ein Schieberegister zu füllen bzw. Daten aus einem solchen zu lesen. Es ist aber zu beachten, daß es unmöglich ist, den Takt an der seriellen Schnittstelle von außen vorzugeben; der 80(C)515/-535 muß den Takt immer selbst erzeugen.

8.1.2 Modus 1 - Asynchrone 8-Bit-Datenübertragung, varialbel Baudrate

In den asynchronen Betriebsarten ist der Takt der seriellen Übertragung extern nicht verfügbar; alle Teilnehmer an der Übertragung müssen daher auf dieselbe Baudrate eingestellt sein. Um die Nutzdaten zu erkennen, sind sie bei den asynchronen Modi in Start- und Stoppbits eingebettet. Im Modus 1 werden insgesamt zehn Bit übertragen: ein Startbit (0), acht Datenbits (das LSB zuerst) und schließlich ein Stoppbit (1). Die Baudrate in Modus 1 kann in einem weiten Bereich gewählt werden, indem entweder Timer 1 oder ein spezieller Baudratengenerator programmiert wird.

8.1.3. Modus 2 - Asynchrone 9-Bit-Übertragung, feste Baudrate

Im Modus 2, der ebenfalls asynchron ist, werden insgesamt elf Bit übertragen: ein Startbit (0), neuen Datenbits und ein Stoppbit (1). Das neunte Datenbit kann für verschiedene Zwecke verwendet werden; z.B. kann darin ein Parity-Bit untergebracht werden, um die Übertragung zu sichern. Eine weitere Möglichkeit ist es, das neunte Bit immer auf 1 zu setzen; dann erscheint es als zweites Stopppbit, da es als höchstwertigstes Bit (MSB) als letztes übertragen wird, direkt vor dem Stoppbit. Die Baudrate im Modus 2 ergibt sich direkt aus der Oszillatorfrequenz des Bausteins; es ist lediglich möglich, eine Teilung durch den Faktor 2 einzuschalten. Sonst sind keine Alternativen möglich.

8.1.4. Modus 3 - Asynchrone 9-Bit-Übertragung, variable Baudrate

In Modus 3, der ebenfalls asynchron ist, werden insgesamt elf Bit übertragen: ein Startbit (0), neun Datenbits und abschließend ein Stoppbit (1). Das Format der Datenübertragung und die Anwendungsmöglichkeiten in Modus 3 entsprechen denen im Modus 2. Der Unterschied liegt lediglich in der flexibelen Erzeugung der Baudraten, die dem Modus 1 entspricht. Wie auch dort kann die Baudrate in einem weiten Bereich gewählt werden, indem entweder Timer 1 oder ein Baudratengenerator programmiert wird.

8.1.5 Die Register SCON (98H) und SBUF (99H)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
9FH (MSB)	9EH	9DH	9CH	9BH	9AH	99H	98H (LSB)

Bit	Funktion
SM0 SM1 0 0 0 1 1 0 1 1	Serieller Modus Modus 0: Schieberegister-Modus; Baudrate: $f_{OSZ}/12$ Modus 1: 8-Bit-UART, flexible Baudrate Modus 2: 9-Bit-UART, feste Baudrate; Baudrate: $f_{OSZ}/12$ oder $f_{OSZ}/64$ Modus 3: 9-Bit-UART, flexible Baudrate
SM2	Aktiviert den Multiprozessor-Kommunilationsbetrieb in Modus 2 und 3. Mit SM2=1 in Modus 2 und 3 wird RI nicht gesetzt, wenn das 9. empfangene Datenbit 0 ist. Mit SM2=1 in Modus 1 wird RI nicht gesetzt, wenn kein gültiges Stoppbit erkannt wurde. Im Modus 0 muß SM2 immer 0 sein.
REN	Receive Enable = Empfängeraktivierung. Wenn REN=1 ist, ist serieller Empfang möglich, mit REN=0 ist serieller Empfang nicht möglich. REN muß durch Software gesetzt/zurückgesetzt werden.
TB8	Tranmitter Bit 8 (Sendebit 8). Dies ist das 9. Datenbit, das in Modus 2 und 3 ausgesendet wird. Muß durch Software gesetzt/zurückgesetzt werden.
RB8	Receiver Bit 8 (Empfangsbit 8). Dies ist das 9. Datenbit, das in Modus 2 und 3 empfangen wird. In Modus 1 (mit SM2=0) ist RB8 das empfangene Stoppbit. Im Modus 0 wird RB8 nicht benutzt.
TI	Transmitter Interrupt (Sender-Interrupt). Dies ist das Interrupt-Request-Flag für den Sender. TI wird von der Hardware im Modus 0 am Ende des 8. Datenbits gesetzt, in den anderen Modi bei Beginn des Stoppbits. TI muß durch Software zurückgesetzt werden.
RI	Receiver Interrupt (Empfänger-Interrupt). Dies ist das Interrupt-Request-Flag für den Empfänger. RI wird von der Hardware im Modus 0 am Ende des 8. Datenbits gesetzt, in den anderen Modi während des Stoppbits. RI muß durch Software zurückgesetzt werden.

Bild 8.1: Special Function Register SCON (98H)

SBUF ist das gemeinsame Empfangs- und Sendepuffer der seriellen Schnittstelle. Ein Schreibbefehl auf SBUF lädt den Sendepuffer und startet eine Übertragung (nur in den Modi 1 bis 3). Ein Lesebefehl greift auf das physikalisch getrennte Empfangsregister SBUF zu.

SBUF7 (MSB)	SBUF6	SBUF5	SBUF4	SBUF3	SBUF2	SBUF1	SBUF0 (LSB)

Bild 8.2: Special Function Register SBUF (99H)

8.2 Multiprozessor-Kommunikation

Die serielle Schnittstelle hat die Möglichkeit, das neunte übertragene Datenbit in den Modi 2 und 3 für einen besonderen Anwendungsfall zu verwenden. Es ist möglich, daß ein Interrupt bei Empfang eines neunten Datenbits nur dann einen Empfangsinterrupt wirklich auslöst, wenn das Bit RB8 eine 1 ist. Bei RB8=0 geschieht nichts. Dieses Verhalten wird aktiv, wenn das Steuerbit SM2 gleich 1 ist.

Mit diesem Leistungsmerkmal ist es relativ einfach, eine Multiprozessorkommunikation zwischen Prozessoren der 8051-Familie aufzubauen. Dabei erfolgt die Kopplung über sie serielle Schnittstelle. Einer der Prozessoren ist der Master. Wenn dieser dann an einen der anderen (Slaves) Daten übertragen will, kann das folgendermaßen geschehen: Das neunte Datenbit wird zur Unterscheidung zwischen einer Adressierung eines der Slaves (TB8=1) und der eigentlichen Datenübertragung (TB8=0) verwendet. Zunächst wird der Master ein Byte mit gesetztem neunten Bit senden, um den gewünschten Slave auf eine Datenübertragung vorzubereiten. Obwohl alle Slaves das Bit SM2 gesetzt haben, werden sie durch diese Übertragung unterbrochen, da der Empfang wegen des gesetzten neunten Bits immer möglich ist. Alle prüfen nun das empfangene Byte, das sie vereinbarungsgemäß als Adresse ansehen. Nur der Slave, der vom Master adressiert wurde, setzt dann sein Modus-Bit SM2 auf 0 zurück. Dann kann der Master mit der eigentlichen Datenübertragung beginnen, wobei er das neunte Bit zu 0 überträgt (TB8=0). Diese Übertragungen werden jetzt nur noch von dem gewünschten Slave wahrgenommen, während alle anderen Slaves wegen SM2=1 diese Übertragungen ignorieren und somit nicht in ihrem Programmablauf gestört werden.

Ist die gesamte Übertragung beendet, setzt der betroffene Slave sein Bit SM2 wieder auf 1, das Spiel kann von vorn beginnen. Zur Klarstellung soll ausdrücklich darauf hingewiesen werden, daß der beschriebene Kommunikationsablauf durch Programme in allen beteiligten Controllern realisiert werden muß. Damit ist auch klar, daß das beschriebene Schema modifiziert und an besondere Aufgabenstellungen angepaßt werden kann. Die einzige durch die Hardware unterstützte Funktion ist die Unterdrückung des Empfangs in Abhängigkeit vom neunten Bit.

Wie bereits gesagt, ist der Einsatz des Bits SM2 lediglich in den Modi 2 und 3 sinnvoll. Bei Betrieb der Schnittstelle im Modus 0 muß SM2 immer auf 0 bleiben. Wird SM2 im Modus 1 auf 1 gesetzt, so erfolgt ein Empfangsinterrupt (RI=1) nur dann, wenn ein gültiges Stoppbit erkannt wurde; andernfalls wird das Byte ignoriert.

8.3 Baudraten

Wie bereits angedeutet, existieren verschiedene Möglichkeiten zur Baudratenerzeugung in Abhängigkeit von der gewählten Betriebsart der seriellen Schnittstelle.

Im folgenden muß zwischen zwei Begriffen, die in der Beschreibung und in den Abbildungen auftauchen, genau unterschieden werden: Die Baudrate ist die tatsächlich von der Schnittstelle an ihren Ein- oder Ausgängen verwendete Datenrate, d.h. die Anzahl der empfangenen bzw. gesendeten Bits pro Sekunde. Intern wird zur Takterzeugung und zum Empfang der seriellen Daten aber ein um den Faktor 16 schnellerer Takt benötigt; dieser Takt wird Baudraten-Takt genannt. Er wird in dieser Geschwindigkeit benötigt, um z.B. mehrere Abtastungen innerhalb eines Bits vornehmen zu können. Der Baudraten-Takt (baud rate clock) wird von diversen Quellen zur Verfügung gestellt. Er ist generell 16-mal höher als die damit erzeugte Baudrate (baud rate).

Wenn in den Tabellen und Formeln Programmierwerte für bestimmte Baudraten angegeben sind, beziehen sich diese allerdings immer schon auf die endgültige Baudrate und müssen deshalb nicht mehr durch 16 geteilt werden.

8.3.1 Baudrate in Modus 0

Die Baudrate im Modus 0 ist fest.

$$\text{Baudrate für Modus 0} = \frac{f_{osz}}{12}$$

8.3.2 Baudrate in Modus 2

Die Baudrate im Modus 2 ist ebenfalls direkt von der Oszillatorfrequenz abgeleitet. Das Steuerbit SMOD im SFR PCON (87H) kann allerdings eine zusätzliche Teilung durch 2 einschalten. Mit SMOD=0 (Voreinstellung) ist die Baudrate 1/64 der Oszillatorfrequenz, mit SMOD=1 ist sie 1/32 der Oszillatorfrequenz.

$$\text{Baudrate für Modus 2} = \frac{2^{SMOD}}{2} * \frac{1}{16} * \frac{f_{osz}}{2}$$

SMOD	PDS	IDLS	-	GF!	GF0	PDE	IDLE
(MSB)							(LSB)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

Bit	Funktion
SMOD	Bei SMOD=1 verdoppelt sich die Baudrate der seriellen Schnittstelle in den Modi 1, 2 und 3

Bild 8.3: Special Function Register PCON (87H)

8.3.3 Baudrate in Modus 1 und 3

In diesen beiden Modi ist die Baudrate wählbar und wird durch die Überlaufrate entweder eines speziellen Baudratengenerators oder des Timers 1 bestimmt.

In den Betriebsarten 1 und 3 steht im 80(C)515/-535 ein spezieller Baudratengenerator zur Erzeugung des Baudratentakts für die serielle Schnittstelle zur Verfügung. Um ihn einzuschalten, muß das Steuerbit BD im Special Function Register ADCON (0D8H) auf 1 gesetzt werden. Der Baudratengenerator bildet den Baudratentakt durch eine Teilung des Oszillatortakts durch 2500. Der entstandene Takt kann dann genauso wie im Modus 2 nochmals durch 2 geteilt werden, indem das Steuerbit SMOD programmiert wird. SMOD=0 schaltet die zusätzliche Teilung durch 2 ein. Somit ergeben sich für die übliche Quarzfrequenz vom 12 MHz als Baudraten 4800 Bd (SMOD=0) und 9600 Bd (SMOD=1). Bei anderen Quarzfrequenzen verändern sich die Baudraten entsprechend. Folgende Formel gibt den Zusammenhang wieder:

$$\text{Baudrate für Modus 1, 3} = \frac{2^{SMOD}}{2} * \frac{f_{osz}}{1250}.$$

BD	CLK	ADEX	BSY	ADM	MX2	MX1	MX0
DFH (MSB)	DEH	DDH	DCH	DBH	DAH	D9H	D8H (LSB)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

Bit	Funktion
BD	Freigabebit für den Baudratengenerator der seriellen Schnittstelle. Wenn es gesetzt ist, wird die Baudrate in den Modi 1 und 3 vom Baudratengenerator abgeleitet. Damit können die Standardbaudraten 4800 und 9600 Bd bei 12 MHz Oszillatorfrequenz erzeugt werden.

Bild 8.4: Special Function Register ADCON (0D8H)

In Modus 1 und 3 kann die Baudrate auch durch den Timer 1 erzeugt werden. Wenn das Steuerbit DB auf 0 gesetzt ist, wird die Timer 1-Überlaufrate als Baudratentakt verwendet. Folgende Formel zeigt den Zusammenhang:

$$\text{Baudrate in Modus 1, 3} = \frac{2^{SMOD} * (Timer1 - Überlaufrate)}{32}$$

Unter der Überlaufrate ist die Anzahl der Überläufe des Timer pro Sekunde zu verstehen. Durch welche Betriebsart des Timer 1 die Überlaufrate zustande kommt, ist nicht von Bedeutung. Man wird hier die Betriebsart wählen, die den geringsten Softwareaufwand für die gewünschte Baudrate erfordert. Für die gängigen Baudraten ist es sinnvoll, Timer 1 im Modus 2 zu programmieren; dieser betreibt den Timer als 8-Bit-Timer mit Reload bei Überlauf aus Register TH1. Die folgende Formel zeigt die entstehenden Baudraten in Abhängigkeit vom gewählten Reloadwert (mit Timer 1 in Modus 2):

$$\text{Baudrate in Modus 1, 3} = \frac{2^{SMOD}}{2} * \frac{1}{16} * \frac{f_{osz}}{12 * (256 - (TH1))}$$

Sehr niedrige Baudraten lassen sich auf diese Weise allerdings nicht erzeugen. Wird dies aber gewünscht, ist es zweckmäßig, den Timer 1 im Modus 1 zu betreiben (16-Bit-Timer-Modus).

Den notwendigen Reload muß man dann aber durch Software unter Verwendung des Timer-1-Interrupts vornehmen.

In der folgenden Tabelle sind einige repräsentative Betriebsfälle bei der Baudratenerzeugung mit Timer 1 herausgegriffen:

Baudraten in Modus 1 und 3	f_{OSZ}/MHz	SMOD	Timer 1		
			$C/\overline{T}$	Modus	Reload
62,5 kBd	12,000	1	0	2	FFH
19,2 kBd	11,059	1	0	2	FDH
9,6 kBd	11,059	0	0	2	FDH
4,8 kBd	11,059	0	0	2	FAH
2,4 kBd	11,059	0	0	2	F4H
1,2 kBd	11,059	0	0	2	E8H
110 Bd	6,000	0	0	2	72H
110 Bd	12,000	0	0	1	FEEBH

Tabelle 8.1: Baudratenerzeugung mit Timer 1

Das Blockschaltbild 8.5 zeigt zusammenfassend die Baudratenabhängigkeit des Baudratentakts von den Steuerbits.

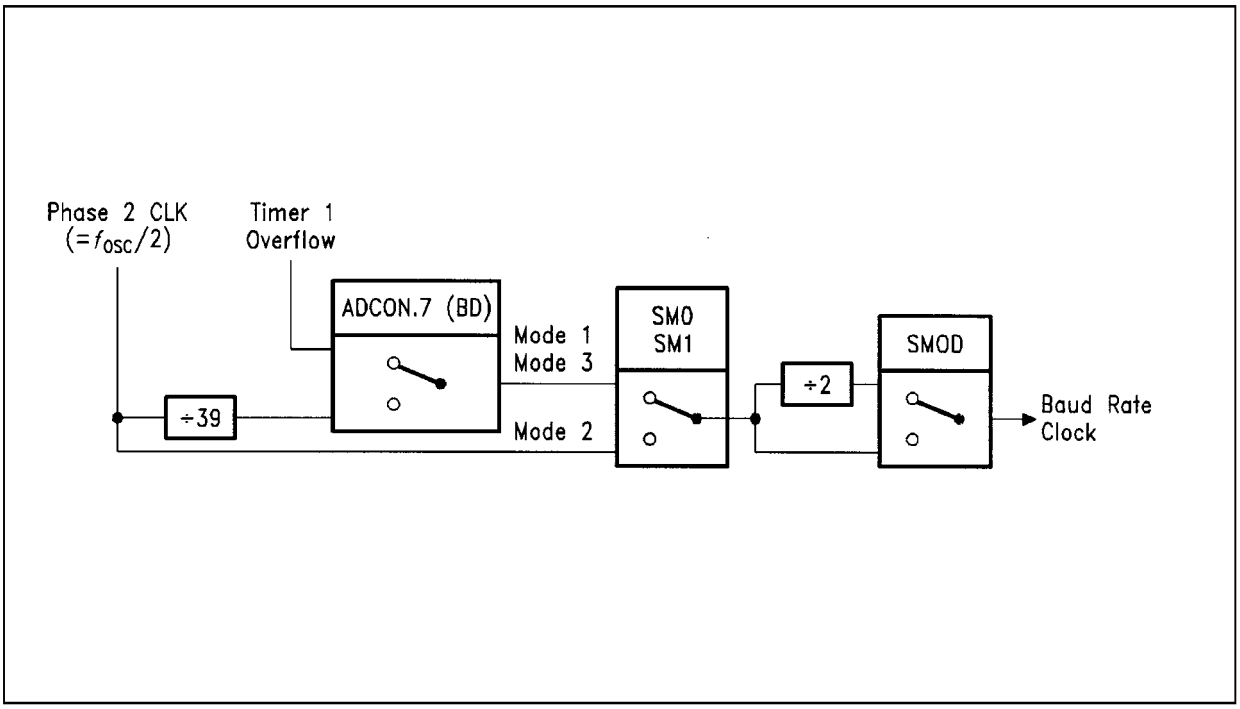


Bild 8.5: Baudratenerzeugung für die serielle Schnittstelle

Baudrate abgeleitet von	Modus	Baudrate
Timer 1 in Modus 1	1, 3	$\frac{2^{SMOD} * (Timer1 - Überlaufrate)}{32}$
Timer 1 in Modus 2	1, 3	$\frac{2^{SMOD}}{2} * \frac{1}{16} * \frac{f_{osz}}{12 * (256 - (TH1))}$
Oszillator	2	$\frac{2^{SMOD}}{2} * \frac{1}{16} * \frac{f_{osz}}{2}$
Baudratengenerator	1, 3	$\frac{2^{SMOD}}{2} * \frac{f_{osz}}{1250}$

Tabelle 8.2: Formeln für Baudraten der seriellen Schnittstelle

8.4. Die Betriebsarten im Detail

8.4.1 Modus 0 - Synchroner Betrieb

Die Daten werden gesendet bzw. empfangen über Pin *RxD*. *TxD* gibt den Schiebetakt aus. Es werden acht Bit - das LSB zuerst - übertragen. Die Baudrate ist mit $f_{OSZ}/12$ fest. Die Abbildungen 8.6 a) und b) zeigen ein funktionales Schaltbild in Modus 0, sowie das zugehörige Timing.

Jeder Schreibbefehl nach SBUF startet die Aussendung eines Bytes. Das entsprechende interne Schreibsignal Write-to-SBUF wird zum Zeitpunkt S6P2 ausgelöst und lädt eine 1 in das Sende-Schieberegister und veranlaßt die Ablaufsteuerung (TX Control bloc), mit der Aussendung zu beginnen. Aufgrund des internen Timings vergeht mindestens ein Maschinenzyklus zwischen Write-to-SBUF und der Aktivierung des Signals Send.

Das Signal Send schaltet den Ausgang des Sende-Schieberegisters auf die Alternativfunktion von Portpin *P3.0* und schaltet den Schiebetakt (Shift Clock) auf die Alternativfunktion von Portpin *P3.1*. Der Schiebetakt ist auf Low-Pegel während S3, S4 und S5 jedes Maschinenzyklus und auf High-Pegel während S6, S1 und S2, solange die Übertragung läuft. Vorher und nachher ist der Shift Clock high. Zum Zeitpunkt S6P2 während der Übertragung (Send ist aktiv) wird der Inhalt des Sende-Schieberegisters um eine Stelle nach rechts geschoben.Während die Datenbits nach rechts hinausgeschoben werden, kommen von links Nullen herein. Ist das MSB des Datenbytes am Ausgang des Sende-Schieberegisters angekommen, befindet sich die 1, die ursprünglich in die neunte Position geladen worden war, an der Position links vom MSB. Alle Positionen links davon enthalten Nullen. Genau diese Bedingung veranlaßt die Ablaufsteuerung, einen letzten Schiebeschritt auszuführen und Send zu deaktivieren, sowie TI zu setzen. Die letzten beiden Aktionen werden zum Zeitpunkt S1P1 im zehnten Maschinenzyklus nach Write-to-SBUF ausgeführt.

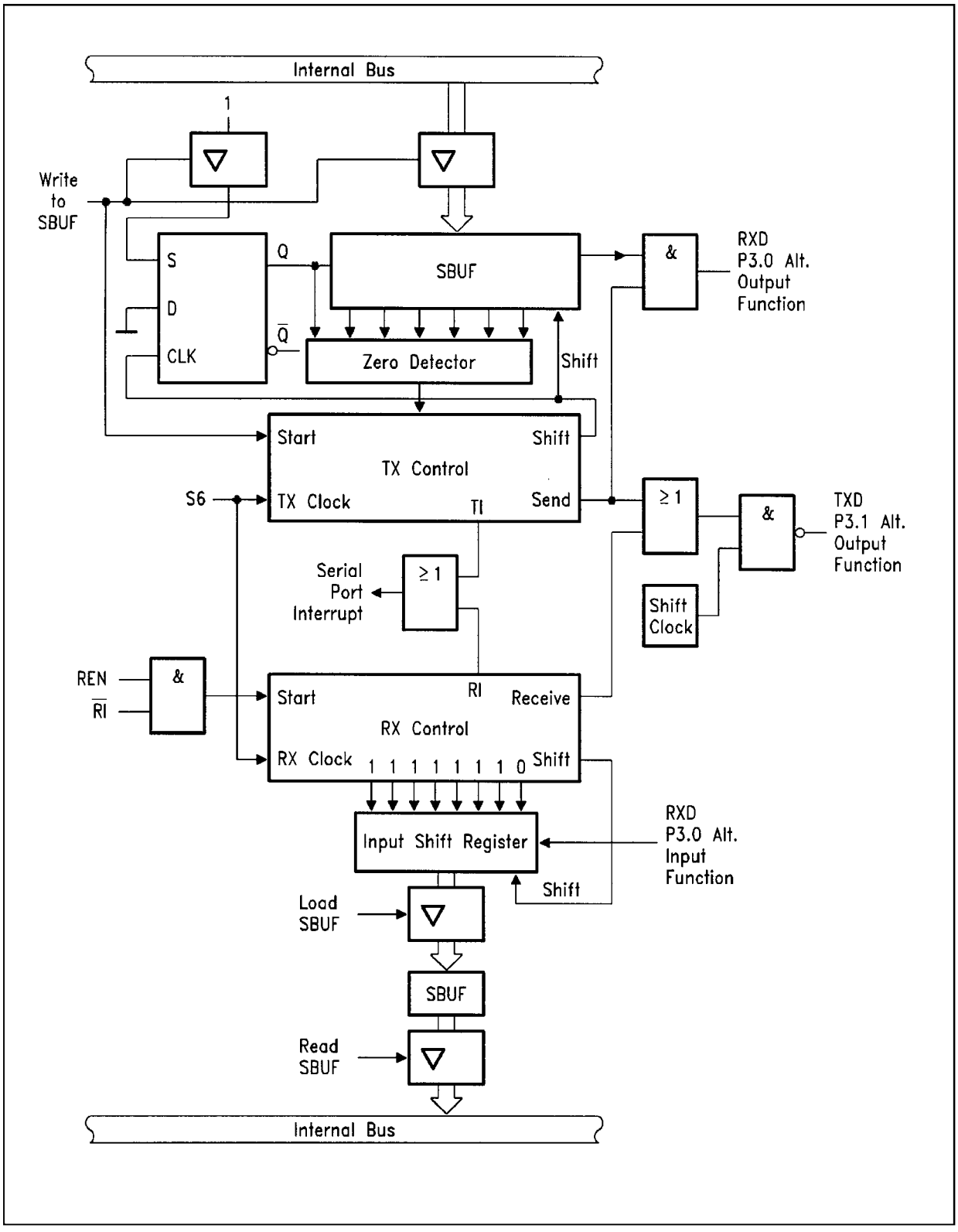


Bild 8.6a): Funktionsdiagramm der seriellen Schnittstelle im Mode 0

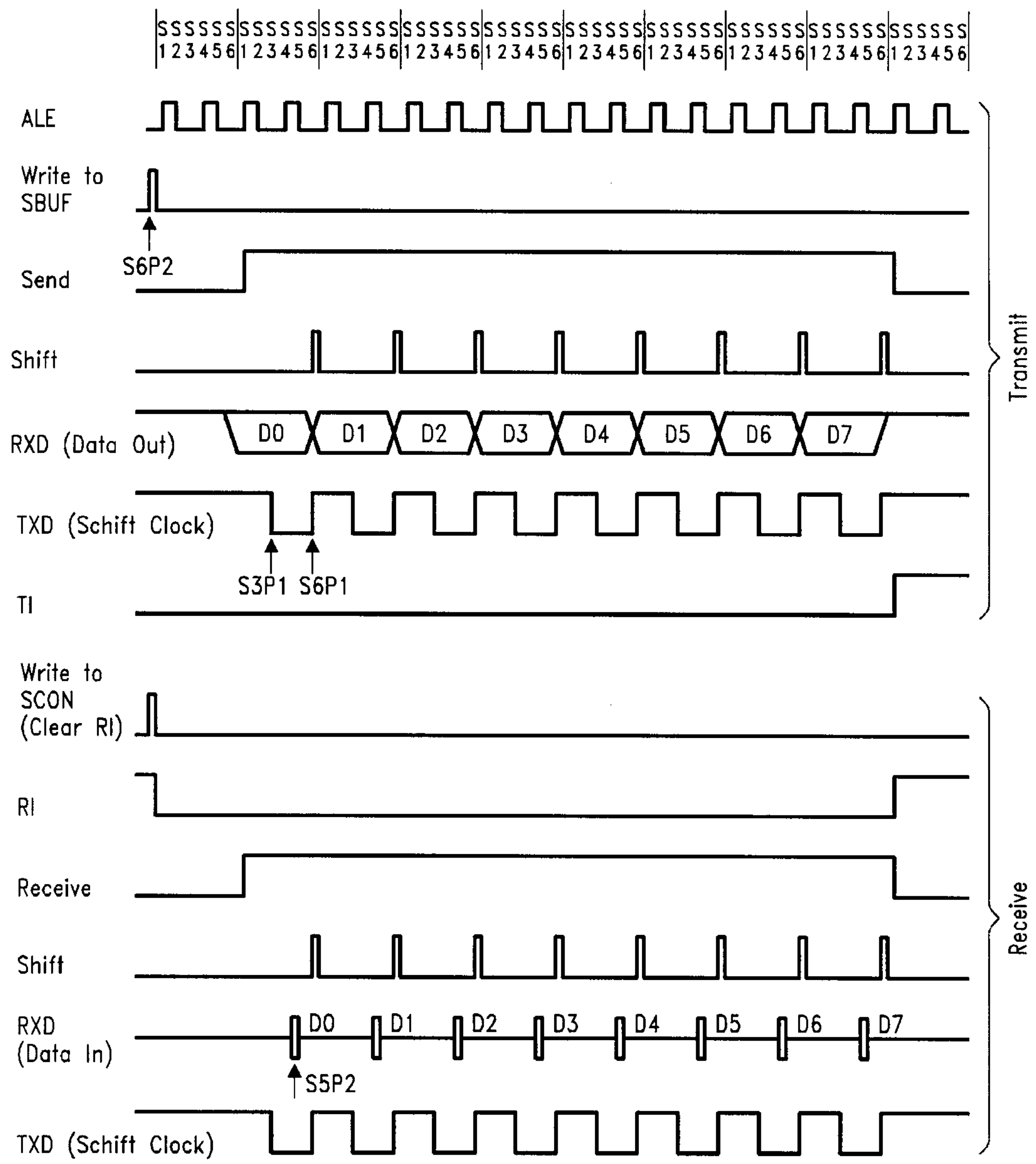


Bild 8.6b): Timingdiagramm der seriellen Schnittstelle im Mode 0

Der Empfang eines Bytes wird gestartet, sobald die Bedingung REN=1 und RI=0 vorliegt. Dies muß durch einen Schreibbefehl in das Register SCHON geschehen, der die beiden Bits auf diese

Werte setzt. Zum Zeitpunkt S6P2 des nächsten Maschinenzyklus schreibt die Empfangs-Ablaufsteuerung (RX Control unit) die Bits 1111 1110 in das Empfangs-Schieberegister; in der darauffolgenden Taktphase wird das Signal Receive aktiviert.

Das Signal Receive schaltet den Schiebetakt auf die Alternativfunktion von Portpin *P3.1*. Der Schiebetakt hat einen Pegelwechsel zum Zeitpunkt S3P1 und S6P1 in jedem Maschinenzyklus, während Receive aktiv ist. Zu jedem Zeitpunkt S6P2 (Receive aktiv) wird der Inhalt des Empfangs-Schieberegisters um eine Position nach links geschoben. Von rechts wird jeweils das Bit hereingeschoben, das jeweils zum Zeitpunkt S5P2 am Portpin *P3.0* abgetastet wird.

Während die empfangenen Datenbits von rechts hereinkommen, werden die anfangs geladenen Einsen nach links herausgeschoben. Wenn die 0, die ursprünglich ganz rechts im Schieberegister geladen worden war, an der ganz linken Position ankommt, erkennt die Ablaufsteuerung, daß ein letzter Schiebeschritt ausgeführt werden muß und dann SBUF mit dem empfangen Byte beschrieben wird. Zum Zeitpunkt S1P1 im zehnten Maschinenzyklus nach dem Schreibvorgang nach SCON, der den Vorgang startete, wird das Signal Receive wieder zurückgesetzt und das Flag RI auf 1 gesetzt.

8.4.2 Modus 1 - 8-Bit-UART

Zehn Bits werden ausgesendet (über den Pin *TxD*) oder empfangen (über den Pin *RxD*): Ein Startbit (0), acht Datenbits mit dem LSB zuerst, ein Stoppbit (1). Bei Abschluß des Empfangs wird der Wert des Stoppbits (immer 1, außer im Fehlerfall), in das Bit RB8 des Special Function Register SCON geschrieben. Die Baudrate ist variabel. Sie wird entweder durch den Baudratengenerator oder durch die Überlaufrate von Timer 1 bestimmt.

Die Abbildungen 8.7 a) und b) zeigen ein funktionales Schaltbild der Schnittstelle im Modus 1, sowie das zugehörige Timing. Die Baudratengenerierung wurde bereits besprochen.

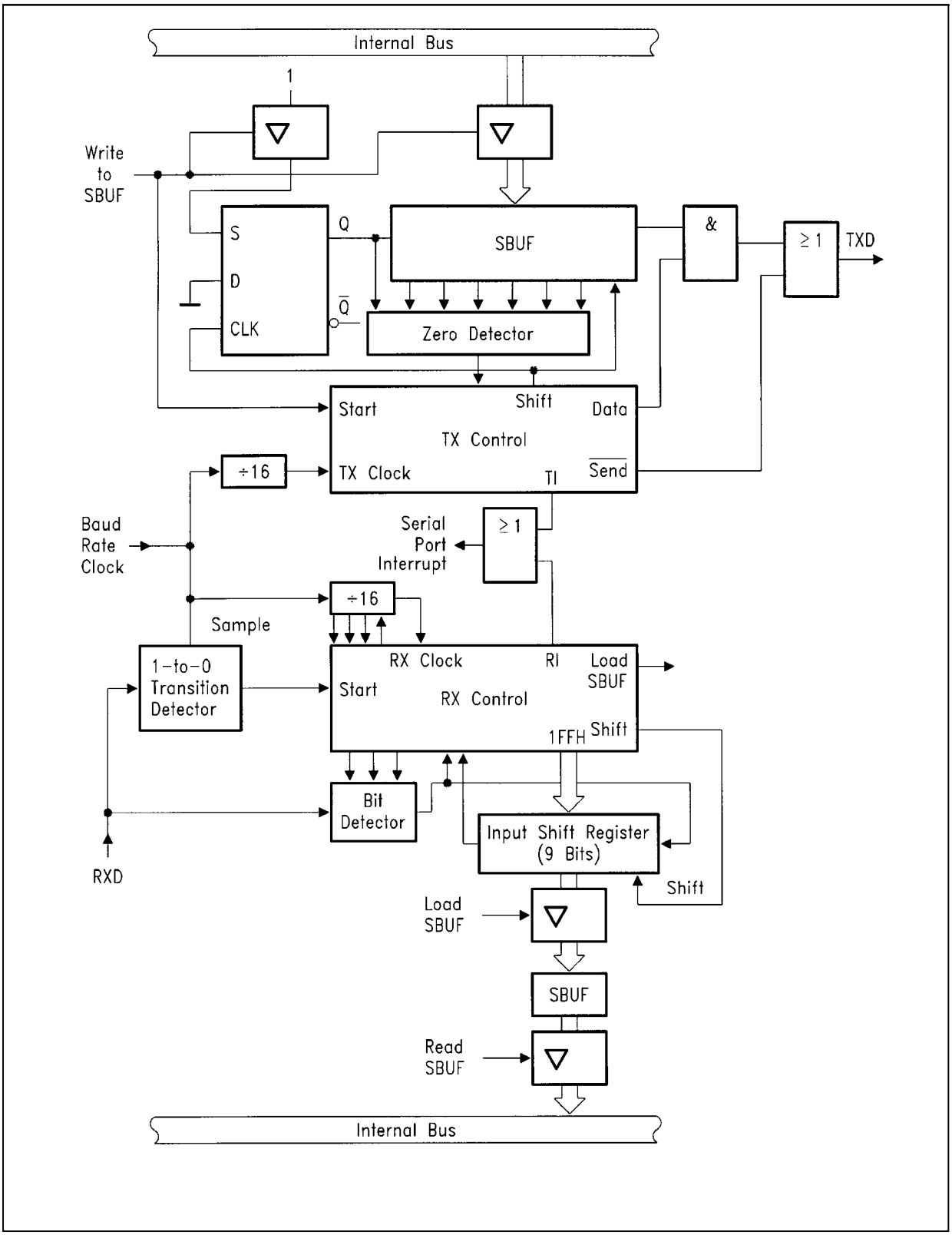


Bild 8.7a): Funktionsdiagramm der seriellen Schnittstelle im Mode 1

Die Aussendung wird durch jeden Befehl gestartet, der SBUF beschreibt. Das entsprechende interne Schreibsignal Write-to-SBUF lädt eine 1 in die neunte Position des Sende-Schieberegisters und veranlaßt die Ablaufsteuerung (TX Control bloc), die Aussendung freizugeben. Der Sendevorgang beginnt dann zum Zeitpunkt S1P1 des Maschinenzyklus, der dem nächsten Überlauf der Teilung durch 16 folgt. Damit werden die Bittakte auf den Teiler durch 16 synchronisiert und nicht auf das Write-to-SBUF-Signal.

Die Aussendung startet dann mit der Aktivierung des Steuersignals $\overline{SEND}$, das zunächst das Startbit an den Pin *TxD* bringt. Einen Bittakt später wird das Signal Data aktiviert. Damit wird der Ausgang des Sende-Schieberegisters auf den Ausgangspin *TxD* geschaltet. Der erste Schiebepuls folgt dann einen weiteren Bittakt später.

Während nun die Datenbits nach rechts hinausgeschoben werden, kommen von links Nullen herein. Wenn das MSB des Datenbytes am Ausgang des Schieberegisters angekommen ist, ist die 1, die ursprünglich an die neunte Bitposition geladen worden war, an der Position links vom MSB angekommen. Alle Positionen links davon enthalten Nullen. Genau diese Bedingung veranlaßt die Ablaufsteuerung, einen letzten Schiebeschritt auszuführen und $\overline{SEND}$ zu deaktivieren, sowie TI zu setzen. Diese Aktionen werden beim zehnten Überlauf des Teilers durch 16 nach Write-to-SBUF ausgeführt.

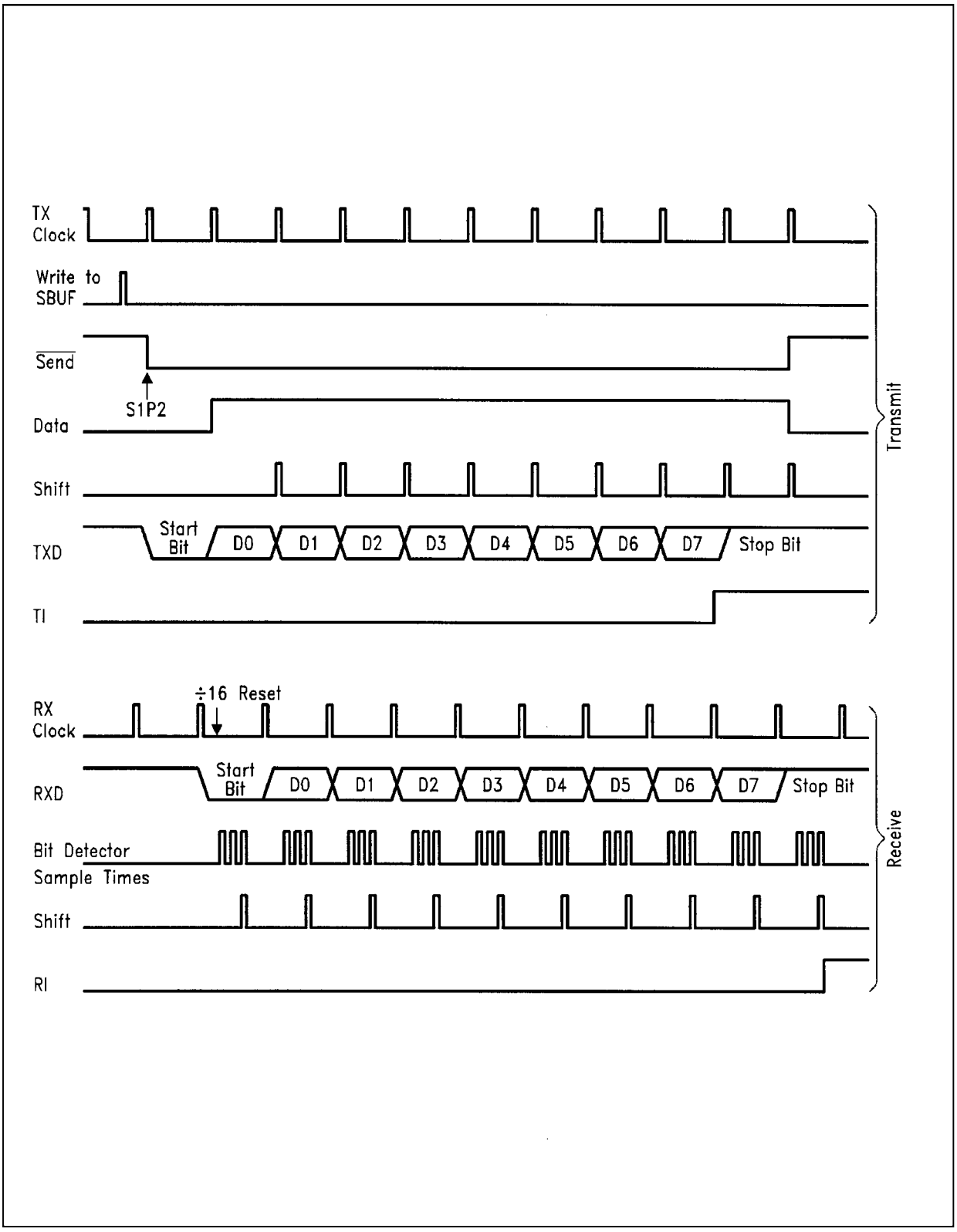


Bild 8.7b): Timingdiagramm der seriellen Schnittstelle im Mode 1

Der Empfang eines Bytes wird gestartet, sobald ein 1-0-Pegelwechsel am Pin *RxD* festgestellt wird. Zu diesem Zweck wird *RxD* mit der 16-fachen Rate der eingestellten Baudraten abgetastet (dies ist der Baudratentakt, der aus diesem Grund 16mal höhere Takt als die eigentliche Baudrate). Sobald der Pegelwechsel erkannt wurde, wird der Teiler durch 16 zurückgesetzt und der Wert 1FFH wird in das Empfangs-Schieberegister eingeschrieben. Dadurch werden die Überläufe des Teilers durch 16 mit dem empfangenen Bittakt synchronisiert.

Die 16 Zählzustände des Teilers durch 16 unterteilen jeden Bittakt in 16 Untertakte. Zum 7., 8. und 9. Untertakt jedes Bittaktes wird der Wert am Pin *RxD* abgetastet. Der endgültige Wert diesen Bittakt wird dann als Mehrheitsentscheidung gebildet, d.h. derjenige Wert wird angenommen, der in mindestens zwei der drei Abtastungen angetroffen wurde. Dies ergibt eine gute Störunterdrückung. Dieser Mechanismus wird auch bereits auf das erste Bit, das Startbit, angewendet. Ergibt die Abtastung keine 0, so wird der Empfang abgebrochen, die Empfangseinheit wird zurückgesetzt und ein neues Startbit wird gesucht. Damit werden Störungen zurückgewiesen, und verhindert, daß der Empfang eines gar nicht vorhandenen Bytes abläuft. Wird das Startbit als korrekt erkannt, wird es in das Empfangs-Schieberegister gebracht und der Empfang läuft normal weiter.

Während die empfangenen Bits von rechts hereinkommen, werden die anfangs geladenen Einsen nach links aus dem Schieberegister (es ist in diesem Modus 9 Bit lang) herausgeschoben. Wenn das Startbit an der Position ganz links ankommt, erkennt die Ablaufsteuerung diese Bedingung. Sie veranlaßt einen letzten Schiebeschritt. Das Signal, das SBUF mit dem empfangen Wert beschreibt und das das Stoppbit in RB8 des Special Function Register SCON einschreibt, sowie RI setzt wird, wird nur unter der Bedingung erzeugt daß:

1. RI = 0 und

2. entweder SM2 = 0 oder das empfangene Stoppbit = 1

Wenn eine der beiden Bedingungen nicht erfüllt ist, werden die empfangenen Daten unwiderruflich verworfen. Sind die Bedingungen aber erfüllt, werden die acht Datenbits in SBUF und das Stoppbit in RB8 von SCON eingeschrieben und das Interrupt-Requestflag RI aktiviert. Danach geht der Empfangsteil wieder in den Anfangszustand, wo er auf den 1-0-Pegelwechsel eines Startbits wartet.

8.4.3 Modus 2 - 9-Bit-UART

Die Betriebsart 2 ist funktionell völlig identisch mit der Betriebsart 3. Die einzige Ausnahme liegt in der Erzeugung der Baudrate. Im Modus 2 kann die Baudrate auf zwei feste Werte programmiert werden: $f_{OSZ}/32$ oder $f_{OSZ}/64$. Im Modus 3 wird die Baudrate aus dem Timer 1 gewonnen, der mit einem Zwölftel der Oszillatorfrequenz oder dem internen Baudratengenerator getaktet wird.

8.4.4 Modus 3 - 9-Bit-UART

Elf Bits werden ausgesendet (über den Pin *TxD*) oder empfangen (über den Pin *RxD*): Ein Startbit (0), acht Datenbits mit dem LSB zuerst, ein programmierbares neuntes Datenbit und ein Stoppbit (1).

Beim Senden wird der (vorher programmierte) Inhalt des Bits TB8 als neuntes Datenbit gesendet. Bei Abschluß des Empfangs wird der Wert des empfangen neunten Bits in das Bit RB8 des Special Function Register SCON geschrieben. Die Baudrate ist variabel. Sie wird wie im Modus 1 erzeugt, entweder durch den Baudratengenerator oder durch die Überlaufrate von Timer 1 bestimmt.

Die Abbildungen 8.8 a) und b) zeigen ein funktionales Schaltbild der Schnittstelle im Modus 2, sowie das zugehörige Timing. Die Baudratengenerierung wurde bereits besprochen.

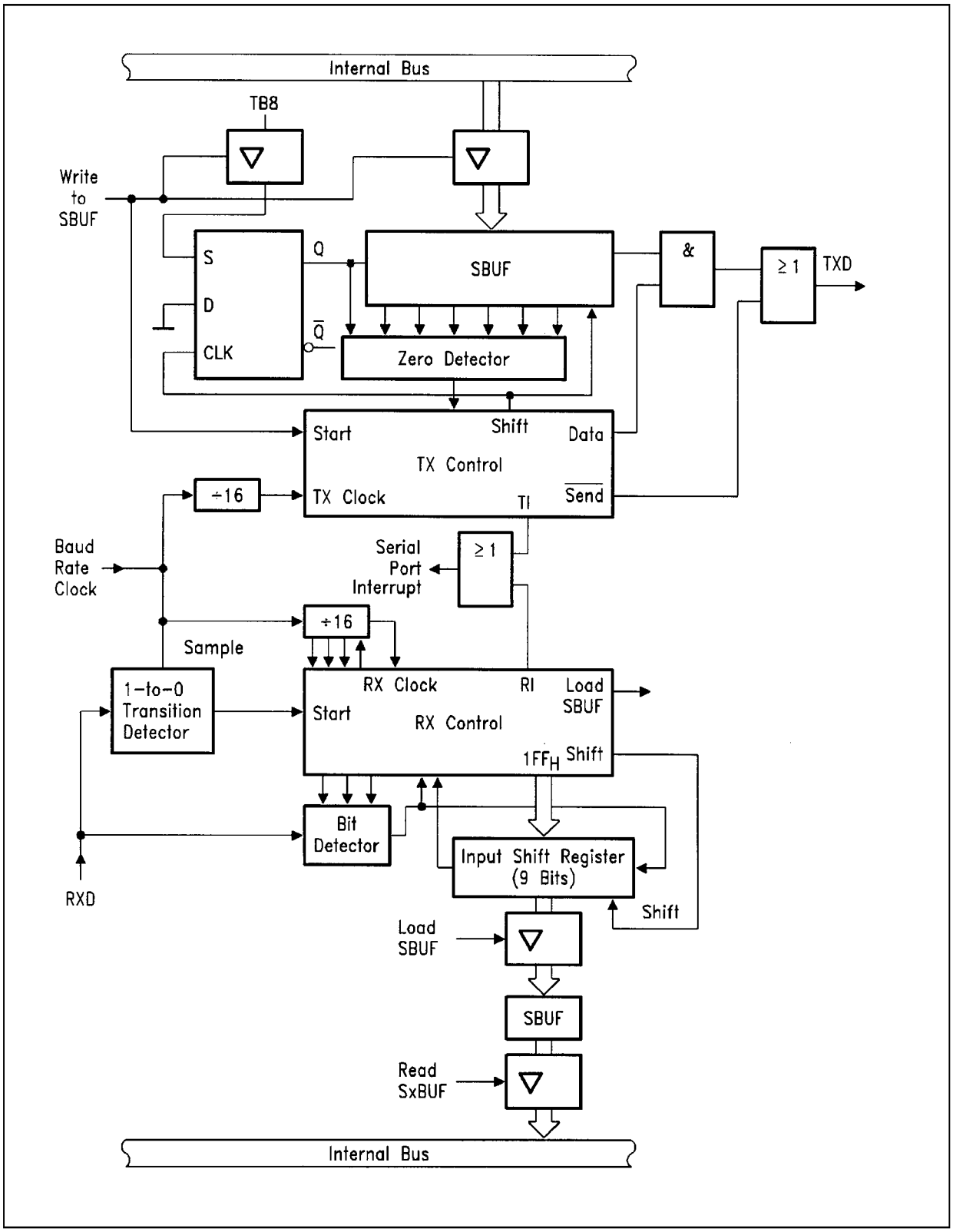


Bild 8.8a): Funktionsdiagramm der seriellen Schnittstelle in den Modi 2 und 3

Die Aussendung wird durch jeden Befehl gestartet, der SBUF beschreibt. Das entsprechende interne Schreibsignal Write-to-SBUF lädt den Inhalt von TB8 in die neunte Position des Sende-Schieberegisters und veranlaßt die Ablaufsteuerung (TX Control bloc), die Aussendung freizugeben. Der Sendevorgang beginnt dann zum Zeitpunkt S1P1 des Maschinenzyklus, der dem nächsten Überlauf der Teilung durch 16 folgt. Damit werden die Bittakte auf den Teiler durch 16 synchronisiert und nicht auf das Write-to-SBUF-Signal.

Die Aussendung startet dann mit der Aktivierung des Steuersignals $\overline{SEND}$, das zunächst das Startbit an den Pin *TxD* bringt. Einen Bittakt später wird das Signal Data aktiviert. Damit wird der Ausgang des Sende-Schieberegisrters auf den Ausgangspin *TxD* geschaltet. Der erste Schiebepuls folgt dann einen weiteren Bittakt später. Dabei wird eine 1 (das spätere Stoppbit) in die neunte Position des Schieberegisters gebracht. Danach werden aber nur noch Nullen nachgezogen.

Während nun die Datenbits nach rechts hinausgeschoben werden, kommen von links Nullen herein. Wenn der Inhalt von RB8 am Ausgang des Schieberegisters angekommen ist, es ist die 1, die beim ersten Schiebetakt in die neunte Bitposition geladen worden war, an der Position links neben TB8 angekommen. Alle Positionen links davon enthalten Nullen. Genau diese Bedingung veranlaßt die Ablaufsteuerung, einen letzten Sachiebeschritt auszuführen und $\overline{SEND}$ zu deaktivieren, sowie TI zu setzen. Diese Aktionen werden beim elften Überlauf des Teilers durch 16 nach Write-to-SBUF ausgeführt.

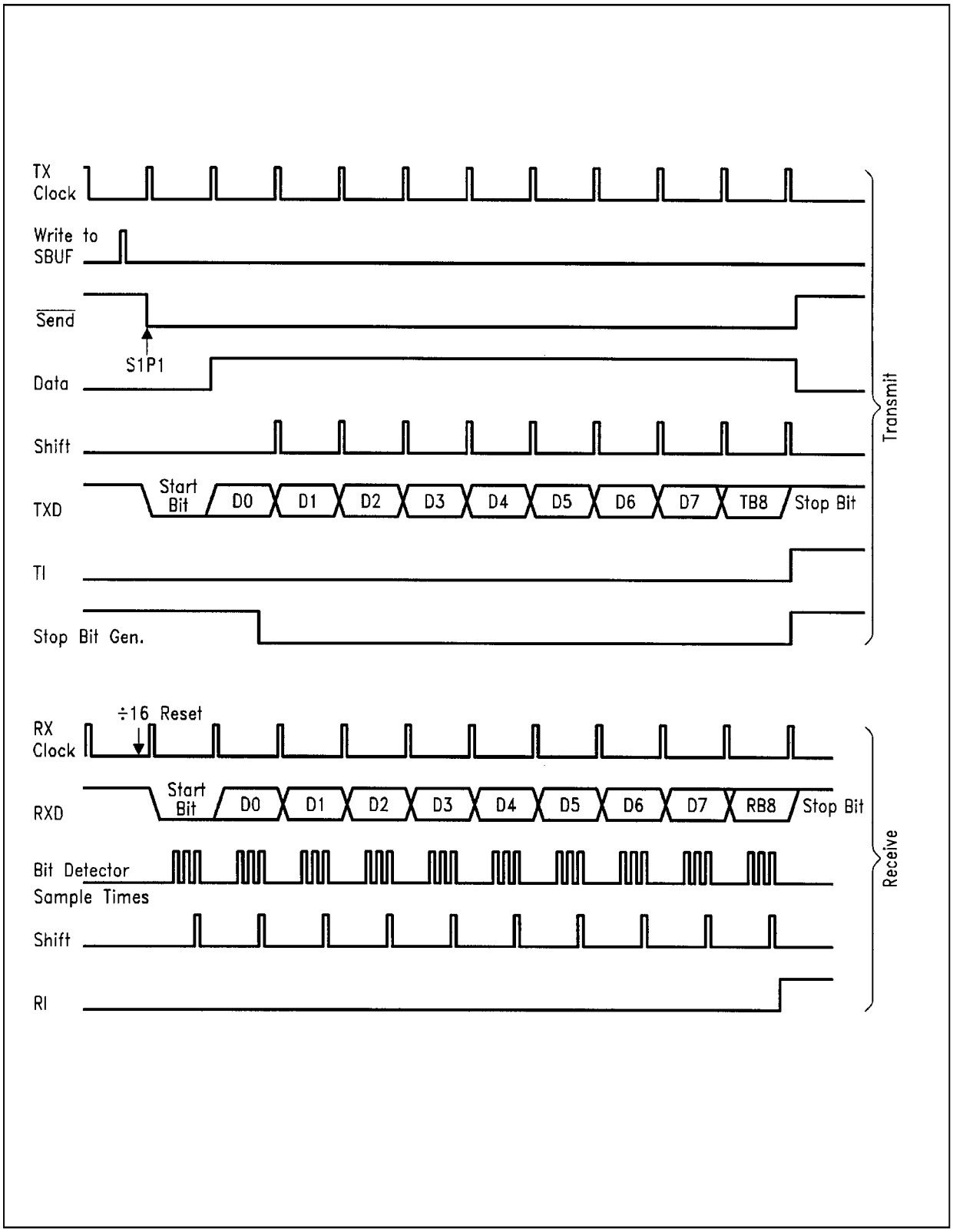


Bild 8.8b): Timingdiagramm der seriellen Schnittstelle in den Modi 2 und 3

Der Empfang eines Bytes wird gestartet, sobald ein 1-0-Pegelwechsel am Pin *RxD* festgestellt wird. Zu diesem Zweck wird *RxD* mit der 16-fachen Rate der eingestellten Baudraten abgetastet (dies ist der Baudratentakt, der aus diesem Grund 16mal höhere Takt als die eigentliche Baudrate). Sobald der Pegelwechsel erkannt wurde, wird der Teiler durch 16 zurückgesetzt und der Wert 1FFH wird in das Empfangs-Schieberegister eingeschrieben. Dadurch werden die Überläufe des Teilers durch 16 mit dem empfanfgenen Bittakt synchronisiert.

Die 16 Zählzustände des Teilers durch 16 unterteilen jeden Bittakt in 16 Untertakte. Zum 7., 8. und 9. Untertakt jedes Bittaktes wird der Wert am Pin *RxD* abgetastet. Der endgültige Wert diesen Bittakt wird dann als Mehrheitsentscheidung gebildet, d.h. derjenige Wert wird angenommen, der in mindestens zwei der drei Abtastungen angetroffen wurde. Dies ergibt eine gute Störunterdrückung. Dieser Mechanismus wird auch bereits auf das erste Bit, das Startbit, angewendet. Ergibt die Abtastung keine 0, so wird der Empfang abgebrochen, die Empfangseinheit wird zurückgesetzt und ein neues Startbit wird gesucht. Damit werden Störungen zurückgewiesen, und verhindert, daß der Empfang eines gar nicht vorhandenen Bytes abläuft. Wird das Startbit als korrekt erkannt, wird es in das Empfangs-Schieberegister gebracht und der Empfang läuft normal weiter.

Während die empfangenen Bits von rechts hereinkommen, werden die anfangs geladenen Einsen nach links aus dem Schieberegister (es ist in diesem Modus 9 Bit lang) herausgeschoben. Wenn das Startbit an der Position ganz links ankommt, erkennt die Ablaufsteuerung diese Bedingung. Sie veranlaßt einen letzten Schiebeschritt. Das Signal, das SBUF mit dem empfangen Wert beschreibt und das das Stoppbit in RB8 des Special Function Register SCON einschreibt, sowie RI setzt wird, wird nur unter der Bedingung erzeugt daß:

1. RI = 0 und

2. entweder SM2 = 0 oder das empfangene neunte Bit = 1

Wenn eine der beiden Bedingungen nicht erfüllt ist, werden die empfangenen Daten unwiderruflich verworfen. Sind die Bedingungen aber erfüllt, werden die acht Datenbits in SBUF und das Stoppbit in RB8 von SCON eingeschrieben und das Interrupt-Requestflag RI aktiviert. Danach geht der Empfangsteil wieder in den Anfangszustand, wo er auf den 1-0-Pegelwechsel eines Startbits wartet.

Achtung: Im Modus 3 wird der Wert des Stoppbits nicht berücksichtigt.

9 Timer 0 und Timer 1

Der 80(C)515/-535 hat insgesamt drei 16-Bit-Timer: die Timer 0 und 1 sind voll kompatibel mit dem des 80(C)51 und können daher mit derselben Programmierung in denselben Betriebsarten genutzt werden, der Timer 2 wird in Kapitel 11 behandelt bei der Besprechung der Capture/Compare/Reload-Funktion.

Die beiden Timer 0 und 1 können grundsätzlich in allen Betriebsarten entweder als Timer oder als Counter benutzt werden. Die englischen Begriffe sind etwas verwirrend, da sie einmal als Bezeichnung für die Funktion, ein anderes mal als Bezeichnung für die Komponente verwendet werden.

- Timer-Funktion bedeutet, daß das Zählregister vom entsprechend geteilten Oszillatortakt regelmäßig inkrementiert wird. Bei den Timern 0 und 1 beträgt der Teilerwert 12. Damit erfolgt praktisch eine Zählung der Maschinenzyklen, da die Zählrate 1/12 des Oszillatortaktes ist. Ein Sonderfall ist die Gated-Timer-Funktion; hier arbeitet die Schaltung wie ein Timer, allerdings nur, wenn durch die Aktivierung eines externen Pins ein Gatter freigeschaltet wird, das sonst das Zählen unterdrücken kann.

- Counter-Funktion bedeutet, daß das Zählregister immer dann inkrementiert wird, wenn ein 1-0-Pegelwechsel (abfallende Flanke) an den entsprechenden Zähleingängen (T0-Alternativfunktion von Pin *P3.4* bzw. T1-Alternativfunktion von Pin *P3.5*) erkannt wurde. Zur Flankenerkennung wird der Pin jeweils zum Zeitpunkt S5P2 abgetastet. Wenn die Abtastungen einen High-Pegel und im darauffolgenden Zyklus einen Low-Pegel ergeben, wird der Zählerstand inkrementiert. Der neue Wert erscheint dann zu S3P1 des folgenden Maschinenzyklus im Zählregister. Es erfordert mindestens zwei Maschienzyklen (24 Oszillatorperioden) um einen 1-0-Wechsel zu erkennen. Daher ist die höchstmögliche Eingangsfrequenz für die Counter-Funktion 1/24 des Oszillatorfrequenz. Andernfalls gehen Zählsignale verloren. Das Tastverhältnis spielt keine Rolle, solange jeder Pegel mindestens einen Maschinenzyklus anliegt, damit er sicher erkannt wird. Ein Sonderfall ähnlich wie die schon erwähnte Gated-Timer-Funktion ist die sehr selten genutzte Gated-Counter-Funktion; hier arbeitet die Schaltung wie ein Counter, allerdings nur, während durch die Aktivierung eines externen Pins ein Gatter freigeschaltet wird, das sonst das Zählen unterdrücken kann.

Beide Timer können in vier Betriebsarten (Modi) benutzt werden. Jeder Timer besteht aus zwei 8-Bit-Registern: TL0 (Timer 0 Low; 8AH) und TH0 (Timer 0 High; 8CH) für den Timer 0 und TL1 (Timer 1 Low; 8BH) und TH1 (Timer 1 High; 8DH) für Timer 1. Diese Register werden abhängig von der Betriebsart unterschiedlich verwendet. Die Modusauswahl und die Steuerung beider Timer erfolgt über die Special-Function-Rregister TCON (Timer Control, 88H) und TMOD (Timer Modus, 89H), die in den Bildern 9.1 und 9.2 dargestellt sind.

Da die Modi 0, 1 und 2 für beide Timer identisch sind, werden sie in der folgenden Beschreibung nur für Timer 0 explizit dargestellt und beschrieben.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
8FH (MSB)	8EH	8DH	8CH	8BH	8AH	89H	88H (LSB)

Bit	Funktion
TF1	Timer-1-Überlaufflag. Es wird von der Hardware bei einem Überlauf des Timer 1 gesetzt. Beim Einsprung in die zugehörige Interrupt-Adresse wird es automatisch gelöscht.
TR1	Freigabebit für Timer 1 (Timer 1 Run Flag). Es muß durch Software gesetzt oder gelöscht werden, um den Timer 1 zu starten oder anzuhalten.
TF0	Timer-0-Überlaufflag. Es wird von der Hardware bei einem Überlauf des Timer 0 gesetzt. Beim Einsprung in die zugehörige Interrupt-Adresse wird es automatisch gelöscht.
TR0	Freigabebit für Timer 0 (Timer 0 Run Flag). Es muß durch Software gesetzt oder gelöscht werden, um den Timer 0 zu starten oder anzuhalten.

Bild 9.1: Special Function Register TCON (88H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

Timer 1				Timer 0			
Gate	C/$\overline{T}$	M1	M0	Gate	C/$\overline{T}$	M1	M0
(MSB)							(LSB)

Bit	Funktion
Gate	Wenn dieses Bit gesetzt ist, läuft der Timer nur dann, wenn er mit Trx freigegeben ist und gleichzeitig der Portpin P3.2 (Timer 0) bzw. der Portpin P3.3 (Timer 1) auf High-Pegel ist.
C/$\overline{T}$	Dieses Bit legt fest, ob Timer- oder Counter-Modus verwendet wird. C/$\overline{T}$ = 0 wählt Timer- und C/$\overline{T}$ = 1 wählt Counter-Modus.
M1 M0	Arbeitsmodus
0 0	THx dient als 8-Bit-Timer, TLx bildet einen 5-Bit-Vorteiler
0 1	THx und TLx bilden einen 16-Bit-Timer
1 0	Auto-Reload-Timer (8 Bit). Der Inhalt von THx wird beim Timerüberlauf nach TLx kopiert. THx selbst bleibt unverändert.
1 1	Timer 0: Beide Register TH0 und TL0 arbeiten als eigenständige 8-Bit-Timer. TL0 wird durch die Steuerbits von Timer 0, TH0 durch die Steuerbits von Timer 1 eingestellt. Timer 1: Timer 1 stoppt in dieser Betriebsart.

Bild 9.2: Special Function Register TMOD (89H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

9.1 Modus 0

Wählt man den Modus 0 für Timer 0 oder Timer 1, so wird er als 8-Bit breiter Timer mit einem Vorteiler 1/32 betrieben. In Bild 9.3 ist dies in einem Funktionsschaltbild dargestellt. Der Grund, warum dieser etwas eigenartige Modus implementiert ist, liegt in der von den Entwicklern des 80(C)515/-535 gewünschten Kompatibilität zur Vorgängerfamilie, der 8048-Familie.

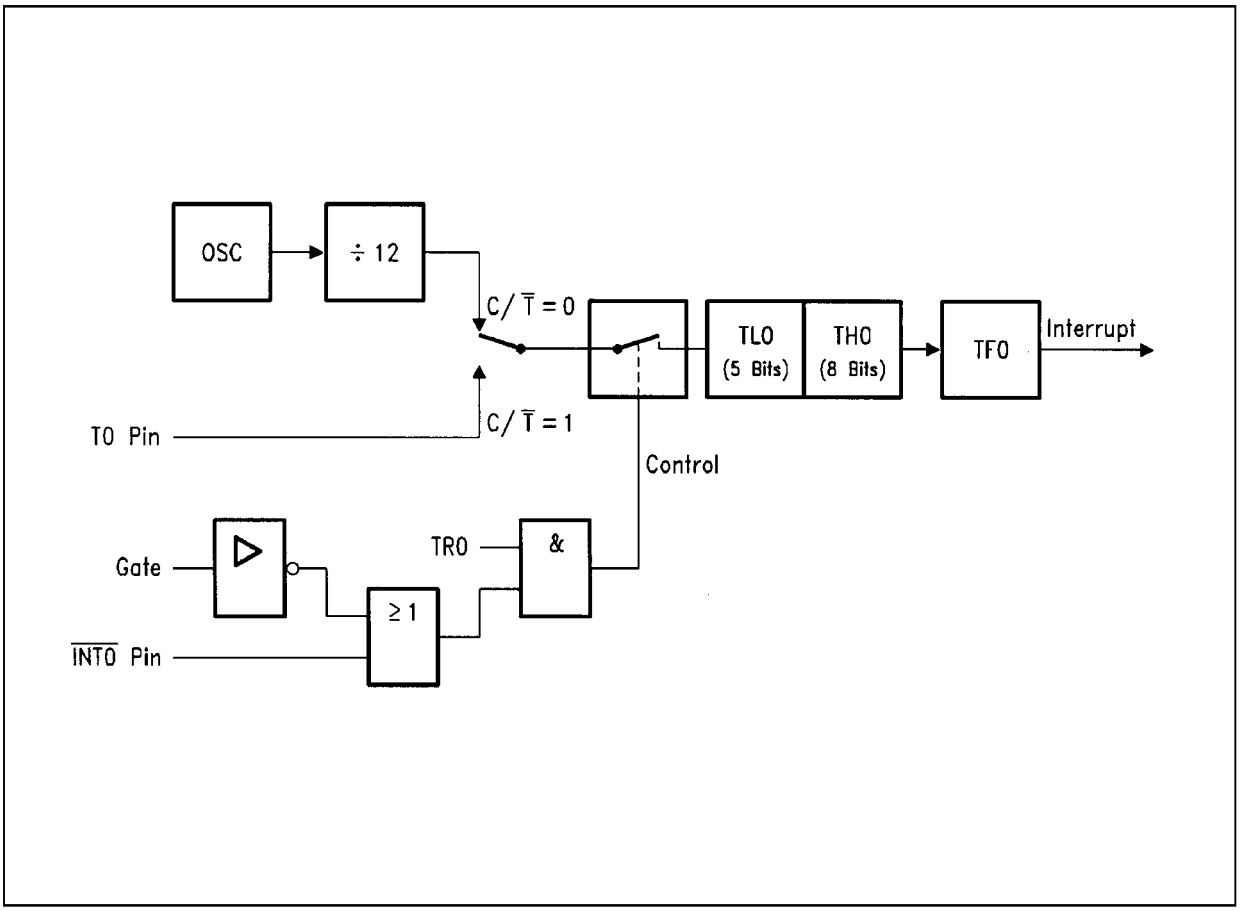


Bild 9.3: Timer/Counter 0 und Timer/Counter 1, Modus 0; 13-Bit-Timer/Counter

Die Timerregister werden zu einem 13-Bit breiten Register kombiniert. Dazu werden alle acht Bit von TH0 und die unteren fünf Bit von TL0 verwendet. Die oberen drei Bit von TL0 sind in diesem Modus undefiniert und sollten nicht durch den Programmierer verwendet werden.

Die Unterscheidung zwischen Timer- und Counterfunktion wird durch das Steuerbit C/$\overline{T}$ (im Register TMOD) getroffen; eine 1 in C/$\overline{T}$ wählt den Counter aus, während eine 0 die Timerfunktion selektiert. Der Timer kann in beiden Funktionen zählen, wenn das Timer-Run-Bit TR0 (im Register TCON) auf 1 gesetzt ist; das Setzten von TR0 gibt den Timer frei, löscht aber nicht das Zählregister. Damit läßt sich mit diesem Bit der Zählvorgang beliebig unterbrechen und wieder aufnehmen.

Die Gated-Timer/Counter-Funktion wird durch das Steuerbit GATE im Register TMOD aktiviert. Ist GATE = 1, so kann nur gezählt werden, wenn der externe Pin *P3.2* auf 1 gehalten wird (Alternativfunktion von *P3.2* ist $\overline{INT0}$).

Wenn der Zähler von seinem Maximalwert (alle Bits sind 1) auf 00H überläuft, setzt der Timer das Überlaufflag TF0. Dieses Flag wird auch von der Interruptlogik als Interrupt-Request-Flag verwendet.

Wie oben schon erwähnt, ist die Btriebsart 0 auch für den Timer 1 in gleicher Weise verfügbar. Es müssen natürlich die entsprechenden Steuer- und Statsbits von Timer 1 verwendet werden (TR1, TF1, TL1, TH1, *P3.3* sowie C/$\overline{T}$ und Gate für Timer 1).

9.2 Modus 1

In Modus 1 wird der Timer genauso betrieben wie im eben beschriebenen Modus 0. Der einzige Unterschied liegt in der vollen Ausnutzung der beiden Timerregister TL0 und TH0, die zu einem einzigen 16-Bit-Zählregister zusammengeschaltet werden. Damit steht dann ein 16-Bit-Timer zur Verfügung. Abbildung 9.4 zeigt das Funktionsschaltbild.

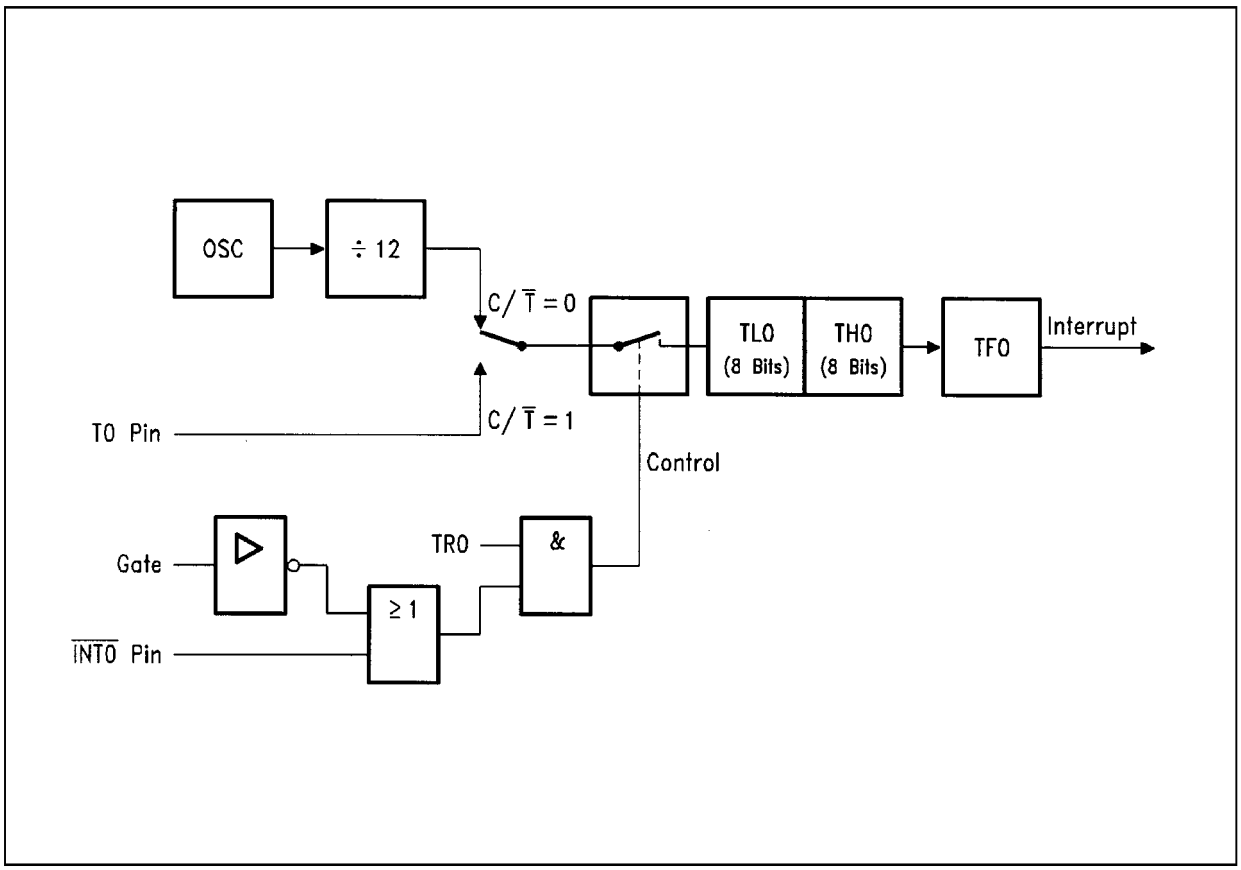


Bild 9.4: Timer/Counter 0 und 1, Mode 1; 16-Bit-Timer/Counter

Es ist übrigens auch möglich, die Register TL0 und TH0 auch während des Betriebs zu schreiben und zu lesen (in allen Modi). Man muß dann allerdings genau auf verschiedene Nebeneffekte achten, um auch das gewünschte Ergebnis zu erzielen: das Problem liegt darin, daß man nur mit 8-Bit-Schreib- oder Leseoperationen auf das 16 Bit breite Timerregister zugreifen kann. Ein Beispiel soll den Vorgang verdeutlichen: Man will den 16-Bit-Wert aus dem laufenden Timer 0 lesen. Zuerst liest man das TH0-Register und legt den Inhalt irgendwo ab. Dann erfolgt die gleiche Prozedur für TL0. Dabei kann der Timer aber zwischen den beiden Lesevorgängen einen Übertrag vom Low- ins High-Register gehabt haben; damit gehört der gelesene High-Wert gar nicht mehr zum später gelesenen Low-Wert. Ähnliche Probleme treten auch beim Schreiben in

den laufenden Timer auf, so daß man diesen Schwierigkeiten am besten dadurch aus dem Weg geht, daß man den Zähler vorher stoppt (wenn es möglich ist). Sonst muß man sich durch Kniffe behelfen. Für den oben betrachteten Fall wäre es möglich, das TH0-Register ein zweites mal zu lesen und seinen aktuellen Inhalt mit dem vorherigen zu vergleichen. Sind die Werte unterschiedlich, ist genau der angenommene Fehler aufgetreten; der gesamte Vorgang muß wiederholt werden, in der Hoffnung, daß jetzt keine Übertrag stattgefunden hat.

Die Betriebsart 1 ist analog auch für den Timer 1 mit dessen Steuer- und Statusflags vorhanden.

9.3 Modus 2

In Modus 2 werden die Register TH0 und TL0 nicht als ein Zählregister zusammengeschaltet. Vielmehr wird nur TL0 als 8-Bit-Zählregister verwendet. TH0 dient als Speicher für einen 8-Bit-Reloadwert. Beim Überlauf von TL0 wird der dort liegende Wert wieder nach TL0 eingeschrieben; damit läßt sich der Zeitraum zwischen den Überläufen entsprechend verkürzen. Die sonstigen Funktionen des Timers (Timer/Counter-Funktion, Gate, Interrupt-Request, etc.) entsprechen den Modi 0 und 1. Üblicherweise wird der Modus 2 dazu verwendet, regelmäßig einen Interrupt zu erzeugen, um darin Aktionen auszuführen, die in gleichmäßigen zeitlichen Abständen benötigt werden. Die geeignete Wahl des Reloadwertes erlaubt es dann, den Abstand zwischen solchen Aktionen festzulegen.

Abbildung 9.5 zeigt das Funktionsschaltbild. Die Betriebsart 2 ist analog auch für den Timer 1 mit dessen Steuer- und Statusflags vorhanden.

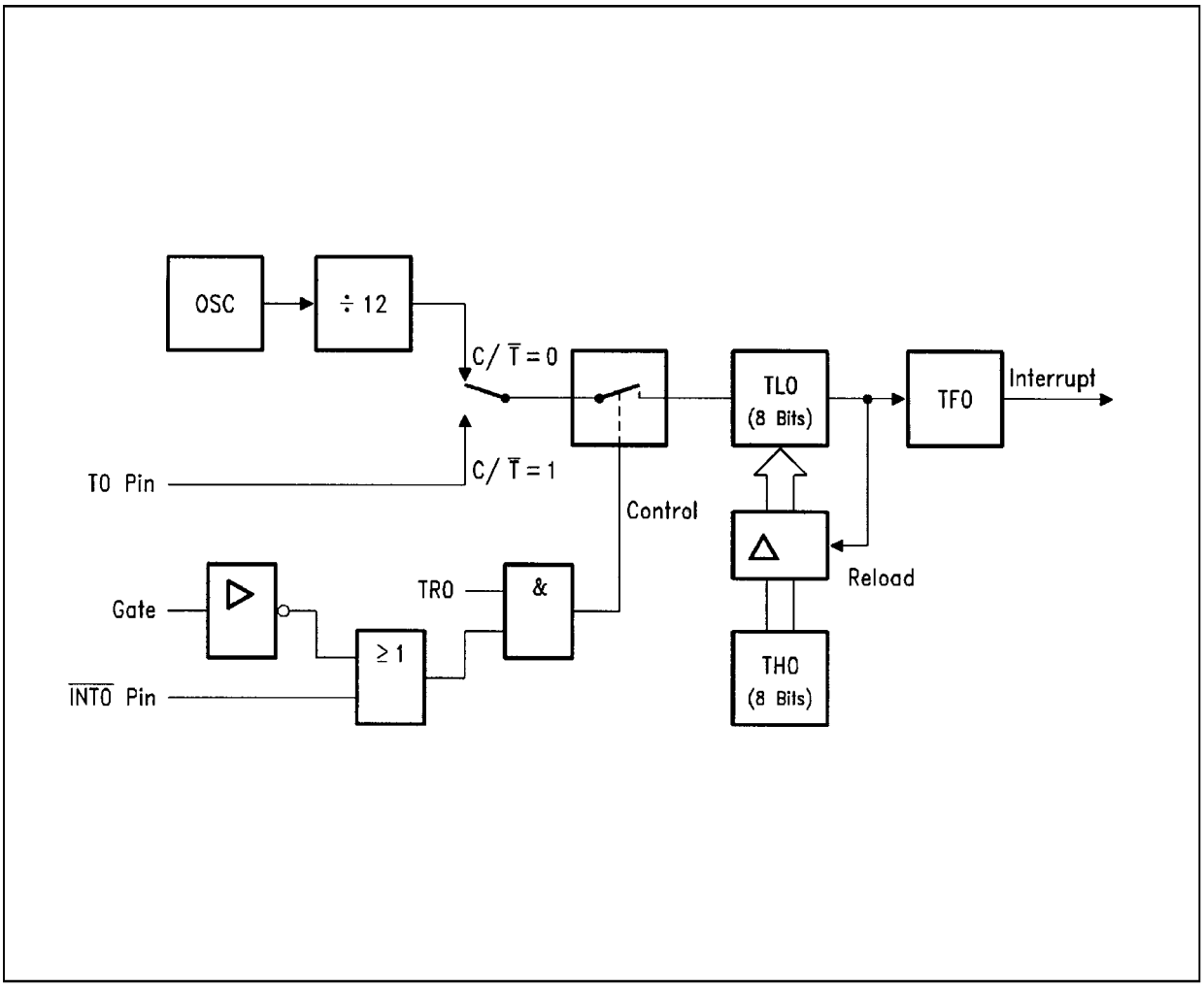


Bild 9.5: Timer/Counter 0 und 1, Mode 2; 8-Bit-Timer/Counter mit Reload

9.4 Modus 3

Im Modus 3 verhalten sich Timer 0 und Timer1 unterschiedlich. Bild 9.6 zeigt das Funktionsschaltbild.

9.4.1 Modus 3 für Timer 0

Im Modus 3 wird der Timer 0 in zwei unabhängige 8-Bit-Timer zerlegt. Das Register TL0 arbeitet mit den aus Modus 0, 1 und 2 bekannten Möglichkeiten (Timer/Counter-Funktion, Gate, Interrupt-Request-Flag TF0). Das Register TH0 wird fest in Timer-Funktion (Zählen von internen Maschinenzyklen) betrieben; als Steuerflag wird TR1 verwendet, das sonst dem Timer 1 zugeordnet ist. Als Interrupt-Request-Flag wird TF1 benutzt. Damit kann TH0, wenn auch eingeschränkt, wie ein acht Bit breiter Timer 1 betrieben werden.

9.4.2 Modus 3 für Timer 1

Timer 1 stoppt, wenn er in Modus 3 versetzt wird. Die Wirkung ist dieselbe, als wenn TR1 auf 0 zurückgesetzt würde.

Die Betriebsart 3 wurde erfunden, als zu Zeiten des 80(C)51 Timer knapp waren. Man gewinnt mit der Betriebsart 3 einen dritten Timer aus dem Timer 0/1-Block. Allerdings hat dann Timer 1 selbst keinen Interrupt mehr, aber wenn man ihn als Baudratengenerator verwendet, benötigt er sowieso keinen.

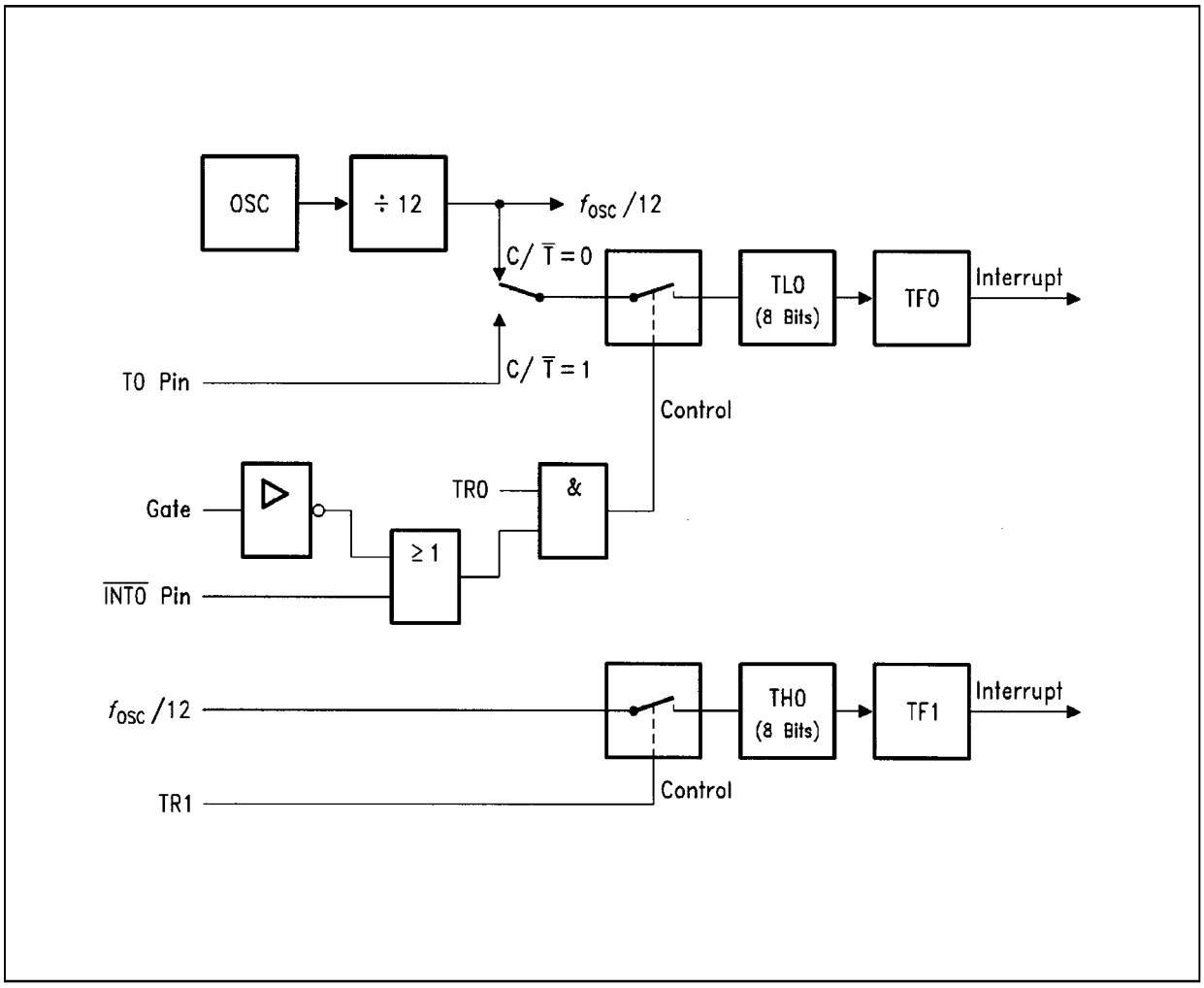


Bild 9.6: Timer 0, Modus 3; Zwei 8-Bit Timer/Counter

10 Analog/Digital-Wandler

Der 80(C)515/-535 hat einen auf dem Chip integrierten Analog/Digital-Wandler mit folgenden Funktionen:

- Acht gemultiplexte Analogeingänge
- Die Möglichkeit, diese Eingänge auch als digitale Eingänge zu verwenden (nur ACMOS-Version)
- Programmierbare interne Referenzspannungsquellen (Teilung in 16 Stufen)
- 8-Bit Auflösung innerhalb des gewählten Bereiches der Referenzspannungen
- Wandlungszeit 13 Maschinenzyklen einschließlich der Samplezeit (ACMOS-Version)
- Wandlungszeit 15 Maschinenzyklen einschließlich der Samplezeit (MYMOS-Version)
- Start der Wandlung durch Software
- Interruptrequest nach jeder Wandlung

Der Wandler arbeitet nach dem Prinzip der sukzessiven Approximation, d.h. für jedes der acht Ergebnisbits, beginnend mit dem höchstwertigen, muß ein Einstell- und Vergleichsschritt durchgeführt werden. Dieses Verfahren ist ein guter Kompromiß zwischen den Anforderungen an die Genauigkeit und die Geschwindigkeit. Im Unterschied zu verschiedenen anderen A/D-Wandlern, die das gleiche Verfahren anwenden, dient hier kein R-2R-Netzwerk zur Erzeugung der benötigten internen Vergleichsspannungen, sondern diese Funktion wird durch ein binär gewichtetes Kondensatorfeld übernommen. Als Konsequenz ergibt sich, daß der gewählte analoge Eingang des Wandlers für die zu messende Spannungsquelle eine kapazitive Last darstellt.

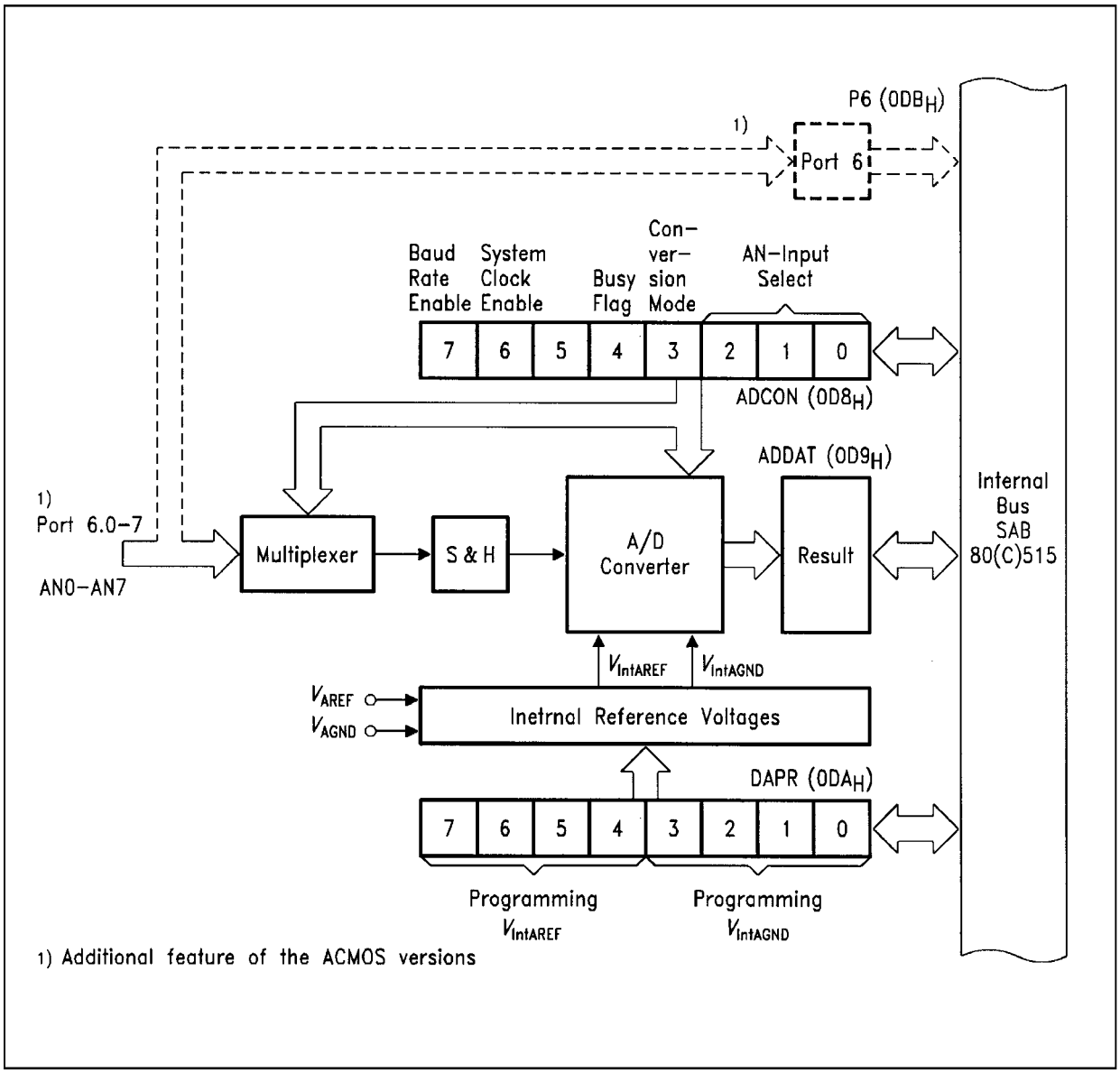


Bild 10.1: Funktionsschaltbild des A/D-Wandlers im 80(C)515/-535

Eine weitere Besonderheit ist die Programmierbarkeit der internen Referenzspannungsquellen. Zunächst werden an die zwei Pins *VAREF* (Voltage Analoque Reference) und *VAGND* (Voltage Analoque Ground) die für die Wandlungen erforderlichen Referenzspannungen angelegt. Damit wird dem Wandler mitgeteilt, welcher analoge Eingangsspannungsbereich das Ergebnis von Null bis zum Maximalwert durchläuft. Der 80(C)515/-535 hat nun die zusätzliche Möglichkeit, sowohl für die obere als auch für die untere Referenzspannung aus den angelegten Spannungen heruntergeteilte Spannungen als Referenzen zu verwenden. Damit erfolgt eine Anpassung an den gewünschten Eingangsspannungsbereich.

Zur Steuerung des Wandlers stehen die Special Funktion Register ADCON (0D8H) ADDAT (0D9H) und DAPR (0DAH) zur Verfügung. Dabei dient ADCON zur Steuerung und zur Statusanzeige, ADDAT enthält nach der Wandlung das Ergebnis und DAPR ist zur

Programmierung der Referenzspannungen vorgesehen. Als Eingang dienen die Portpins *P6.0* bis *P6.7* bzw. *AN0* bis *AN7*.

10.1 Grundfunktion

Das Special Funktion Register ADCON dient zur Einstellung der Betriebsart des Wandlers; zusätzlich enthält es Statusflags und erlaubt die Auswahl eines der acht Eingangskanäle.

BD	CLK	-	BSY	ADN	MX2	MX1	MX0
DFH (MSB)	DEH	DDH	DCH	DBH	DAH	D9H	D8H (LSB)

Bit	Funktion
BSY	Busy-Flag. Es ist während der Wandlung gesetzt und wird von der Hardware gesteuert.
ADM	Betriebsart des Wandlers (A/D Converter Mode). Es legt fest, ob eine einmalige Wandlung (ADM=0) oder Dauerwandlung (ADM=1) durchgeführt wird.
MX2, MX1, MX0	Auswahl des analogen Eingangskanals (siehe Tabelle 10.1)

Bild 10.2: Special Function Register ADCON (0D8H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

Mit den Bit ADM aus dem SFR ADCON können zwei Betriebsarten des Wandlers gewählt werden. Ist ADM auf 0 zurückgesetzt, ist eine Einzelwandlung programmiert. Dabei wird nach dem Wandlungsstart eine einzige Wandlung durchgeführt, danach stoppt der Wandler bis zum nächsten Start. Ist ADM auf 1 gesetzt, wird der Wandler nach Wandlungsende sofort neu gestartet.

Das Busy-Flag zeigt eine gerade ablaufende Wandlung an. Es wird mit Wandlungsstart gesetzt und am Ende durch die Hardware zurückgesetzt. Die Abfrage durch Software (z.B. mit dem Befehl JB BSY,$) erlaubt das Warten auf das Wandlungsende, wenn der entsprechende Interrupt nicht verwendet werden soll. Außerdem ermöglicht es diese Abfrage sicherzustellen, daß nicht während einer laufenden Wandlung in die Steueregister geschrieben wird, was undefinierte Ergebnisse erzeugen kann.

MX2	MX1	MX0	Eingangskanal	Pin	
				MYMOS	ACMOS
0	0	0	Eingang 0	*AN0*	*P6.0*
0	0	1	Eingang 1	*AN1*	*P6.1*
0	1	0	Eingang 2	*AN2*	*P6.2*
0	1	1	Eingang 3	*AN3*	*P6.3*
1	0	0	Eingang 4	*AN4*	*P6.4*
1	0	1	Eingang 5	*AN5*	*P6.5*
1	1	0	Eingang 6	*AN6*	*P6.6*
1	1	1	Eingang 7	*AN7*	*P6.7*

Tabelle 10.1: Auswahl der A/D-Wandler Eingangskanäle

Bei den ACMOS-Versionen kann der Eingangsport P6 sowohl als digitaler Eingang (wenn die Spannungen den Logikpegeln für High und Low entsprechen) oder als Eingang für den A/D-Wandler verwendet werden, dabei spielen die absoluten Pegel keine Rolle, solange sie nicht die Grenzwerte überschreiten.

Das Special Function Register ADDAT wird am Wandlungsende mit dem Ergebnis der Wandlung beschrieben. Außerdem kann ADDAT auch durch Software beschrieben werden, so daß es bei Nichtbenutzung des Wandlers möglich ist, das Register als normale Speicherzelle zu benutzen.

ADDAT7 (MSB)	ADDAT6	ADDAT5	ADDAT4	ADDAT3	ADDAT2	ADDAT1	ADDAT0 (LSB)

Bild 10.3: Special Function Register ADDAT (0D9H)

Der A/D-Wandler kann weiterhin am Ende der Wandlung einen Interrupt anfordern. Er setzt dazu, wie alle Peripheriekomponenten, ein Interrupt-Request-Flag, das hier den Namen IADC trägt. Es liegt im Register IRCON (IRCON.0).

10.2 Referenzspannungen

Der 80(C)515/-535 hat als Besonderheit die durch Software programmierbaren Referenzspannungen, d.h. die Programmierbarkeit erlaubt, verschiedene Spannungen, die aus den extern angelegten Referenzspannungen durch Spannungsteilung abgeleitet worden sind, als Referenzen für den Wandler zu benutzen.

Bekanntlich benötigt jeder A/D-Wandler eine Vorgabe über den analogen Spannungsbereich, innerhalb dessen alle digitalen Ergebniswerte durchlaufen werden (beim 8-Bit-Wandler die Werte 00H bis FFH). Die Wandler haben dazu zwei externe Anschlüsse, die untere und obere Referenz. So hat auch der 80(C)515/-535 beide Anschlüsse. Sie tragen die Namen *VAGND* für die untere Referenz und *VAREF* für die obere Referenz. An die Genauigkeit der angeschlossenen Spannungen sind hohe Anforderungen gestellt, da das Ergebnis nicht genauer sein kann, als die zur Verfügung stehenden Referenzen.

Anders als bei diskreten Wandlern ist es beim 80(C)515/-535 nicht möglich, die extern angelegten Referenzen beliebig zu variieren. Vielmehr steht hier eine relativ enge Bindung an die digitalen Versorgungsspannungen des Bausteins. So darf VAGND nicht mehr als 0,2 V vom Massepegel V_{SS} abweichen und VAREF muß sich in einem Bereich von ± 5% um die Versorgungsspannung V_{CC} bewegen. Außerdem sollte die differentielle Impedanz der Referenzspannungsquelle möglichst niedrig sein und höchstens 5 kΩ betragen. Sichere Ergebnisse werden erzielt, wenn eine gesonderte analoge Referenzspannungsquelle mit geringer Ausgangsimpedanz verwendet wird.

Als Ersatz für die externe Variabilität der Referenzspannungen hat der 80(C)515/-535 die wesentlich flexiblere Lösung der internen Programmierbarkeit. Dazu wird die extern angelegte Referenzspannung durch einen Spannungsteiler in 16 gleichgroße Stufen geteilt. Jede der so gewonnenen Spannungen kann sowohl für die untere als auch für die obere Referenz des Wandlers benutzt werden. Da diese Spannungen nur intern verfügbar sind, tragen sie die Bezeichnungen $V_{IntAGND}$ und $V_{IntAREF}$. Die Auswahl der Spannungen erfolgt durch Programmierung des Special Function Register DAPR. Dabei wird das untere Nibble (die unteren 4 Bit) für die untere interne Referenz und das obere Nibble (die oberen 4 Bit) für obere interne Referenz verwendet. Obwohl es theoretisch möglich wäre, für die untere Referenz eine höhere Spannung als für die obere einzustellen, ist dies natürlich nicht sinnvoll. Der Bausteinhersteller fordert daher, daß die obere interne Referenz immer um mindestens 1 V höher eingestellt ist als die untere, da sonst Fehlmessungen auftreten können.

$V_{IntAREF}$				$V_{IntAGND}$			
DAPR.7 (MSB)	DAPR.6	DAPR.5	DAPR.4	DAPR.3	DAPR.2	DAPR.2	DAPR.0 (LSB)

Bild 10.4: Special Function Register DAPR (0DAH)

Das Register DAPR dient zur Erzeugung der internen Referenzspannungen. Ein Schreibzugriff startet eine A/D-Wandlung. Soll die interne Programmierung nicht genutzt werden, und statt dessen mit den externen Referenzen gearbeitet werden, so muß der Wert 00H in DAPR programmiert werden.

Die Referenzspannungen können, wenn nötig, vor jeder Wandlung neu programmiert werden. Zum Wandlungsstart muß sowieso ein Wert nach DAPR geschrieben werden. Die folgende Tabelle zeigt die sich ergebenden internen Referenzspannungen für den Fall, daß als externe Referenzen 0 V und 5,0 V gewählt wurden.

Stufe	DAPR (.3-.0) DAPR (.7-.4)	$V_{IntAGND}$/V	$V_{IntAREF}$/V
0	0000	0,0	5,0
1	0001	0,3125	-
2	0010	0,625	-
3	0011	0,9375	-
4	0100	1,25	1,25
5	0101	1,5625	1,5625
6	0110	1,875	1,875
7	0111	2,1875	2,1875
8	1000	2,50	2,50
9	1001	2,8125	2,8125
10	1010	3,125	3,125
11	1011	3,4375	3,4375
12	1100	3,75	3,75
13	1101	-	4,0625
14	1110	-	4,375
15	1111	-	4,6875

Tabelle 10.2: Beispiele für interne Referenzspannungen

Die mathematischen Zusammenhänge geben die folgenden Formeln wieder:

$$V_{IntAGND} = V_{AGND} + \frac{DAPR(.3–.0)}{16} * \left(V_{AREF} - V_{AGND}\right) \text{ wobei DAPR(.3-.0) < DH}$$

$$V_{IntAREF} = V_{AGND} + \frac{DAPR(.7-.4)}{16} * \left(V_{AREF} - V_{AGND}\right) \text{ wobei DAPR(.7-.4) > 3H}$$

Allgemein gilt für die Auflösung U_{LSB}, das ist der Spannungswert, der einer Änderung des niederwertigsten Bits entspricht:

$$U_{LSB} = \frac{obere\text{Re}\,ferenz - untere\text{Re}\,ferenz}{2^n - 1} \text{ wobei n die Anzahl der Bits ist}$$

Die Programmierbarkeit der Referenzen erlaubt zwei Anwendungen. So kann der Spannungsbereich, der an einzelnen Kanälen gewandelt wird, an die dort gelieferten Spannungen angeglichen werden. Viele Schaltungen mit Sensoren liefern nicht den gesamten möglichen Bereich von 0 V bis 5 V, sondern nur einen kleineren Bereich. Eine Einstellung der Referenzen auf den notwendigen Bereich ergibt dann direkt ein Ergebnis höherer Auflösung (siehe Bild 10.5).

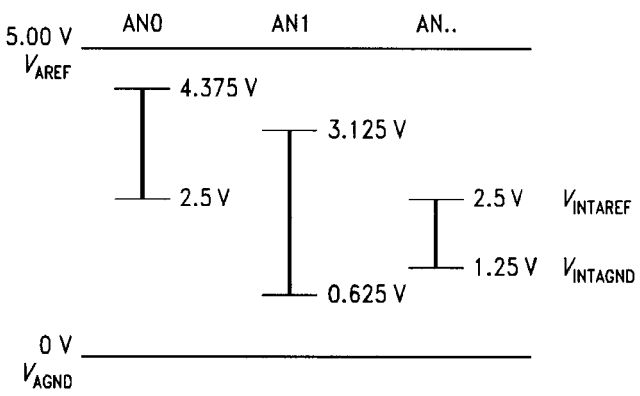


Bild 10.5: Anpassung der internen Referenzspannungen an externe Spannungsbereiche

Als zweite Anwendungsmöglichkeit ist eine erhöhte Auflösung des Ergebnisses möglich, indem zuerst eine Messung mit dem vollen Bereich durchgeführt wird, um eine grobe Lage der Eingangsspannung zu finden. Eine zweite Wandlung des gleichen Eingangswertes mit einem passend gewählten kleineren Referenzspannungsbereich ergibt dann ein Ergebnis höherer Auflösung (siehe Bild 10.6). Hierbei ist allerdings zu beachten, daß durch die Einschränkung des Referenzbereichs keine Genauigkeitsverbesserung erfolgt, sondern lediglich die Auflösung verbessert werden kann.

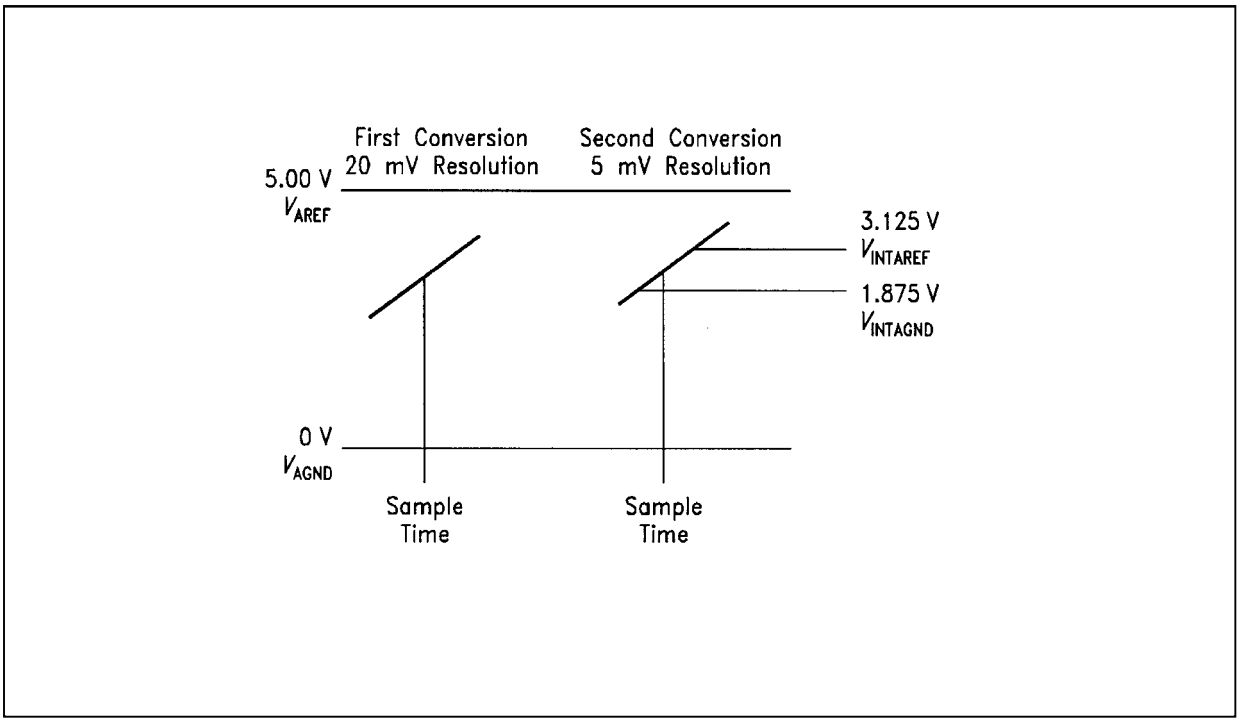


Bild 10.6: Erhöhung der Auflösung durch eine zweite Wandlung

10.3 Timing

Die Wandlung wird angestoßen durch Beschreiben des Registers DAPR, gleichgültig ob eine bereits laufende Wandlung abgeschlossen ist oder nicht (BSY-Flag). Die neue Wandlung beginnt dann im folgenden Maschinenzyklus und das BSY-Flag wird gesetzt. Der Wandlungsablauf wird durch drei Zeiten charakterisiert, die im folgenden beschrieben werden (Bild 10.7).

10.3.1 Load Time T_L

Während dieser Zeit muß die analoge Spannungsquelle die Eingangskapazität C_L des Sample-and-Hold-Glieds aufladen (siehe Bild 10.1). Der Spannungswert, der am Ende der Load Time erreicht ist, wird dann in einen Digitalwert umgesetzt. Aus diesem Grund muß die Eingangsspannungsquelle niederohmig genug sein, um den Aufladevorgang in der geforderten Zeit zustandezubringen. In typischen Anwendungen ist ihre Impedanz kleiner als 5 kΩ.

10.3.2 Sample Time T_S

Während der Sample Time ist das interne Kondensatornetz mit dem ausgewählten analogen Eingang verbunden. Die oben beschriebene Load Time ist ein Teil der Sample Time. Im Anschluß an das Aufladen werden während des Restes der Sample Time intern die Kondensatoren abgeglichen; währenddessen sollte sich die Eingangsspannung nicht ändern, damit

Ergebnisverfälschungen verhindert werden. Eine Veränderung der Eingangsspannung von weniger als 200 bis 300 mV bleibt jedoch ohne Auswirkungen.

10.3.3 Conversion Time T_C

Dies ist die gesamte Wandlungszeit. Sie beinhaltet die oben beschriebene Load Time und die Sample Time. Nach Abschluß der Sample Time beginnt die eigentliche Wandlung durch Vergleich gegen binär gewichtete Teile der Referenzspannung. Im letzten Maschinenzyklus der Wandlung wird das Ergebnis nach ADDAT geschrieben und das BSY-Flag zurückgesetzt. Das Interrupt-Request-Flag IADC wird bereits vier Maschinenzyklen vor dem Rücksetzten von BSY gesetzt. Dies dient dazu, eine schnellstmögliche Interrupt-Bearbeitung des Ergebnisses zu ermöglichen, da das Timing von IADC so gewählt wurde, daß beim frühestmöglichen Einsprung in die Interrupt-Routine gerade das Ergebnis zur Verfügung steht.

Alle Zeiten werden direkt aus dem Oszillatortakt abgeleitet und werden somit in Maschinenzyklen angegeben. Die Zeiten ergeben sich deshalb direkt aus der Oszillatorfrequenz. Bei Dauerwandlung (ADM=1) schließt die folgende Wandlung lückenlos an die vorhergehende an. Das Flag BSY ist dann nur einen Maschinenzyklus zurückgesetzt. Dies ist beim Polling mit dem Befehl JB BSY,$ zu beachten, der als Zwei-Zyklus-Befehl u.U. das kurze Inaktivwerden von BSY übersieht, wenn es im falschen der zwei Zyklen geschieht.

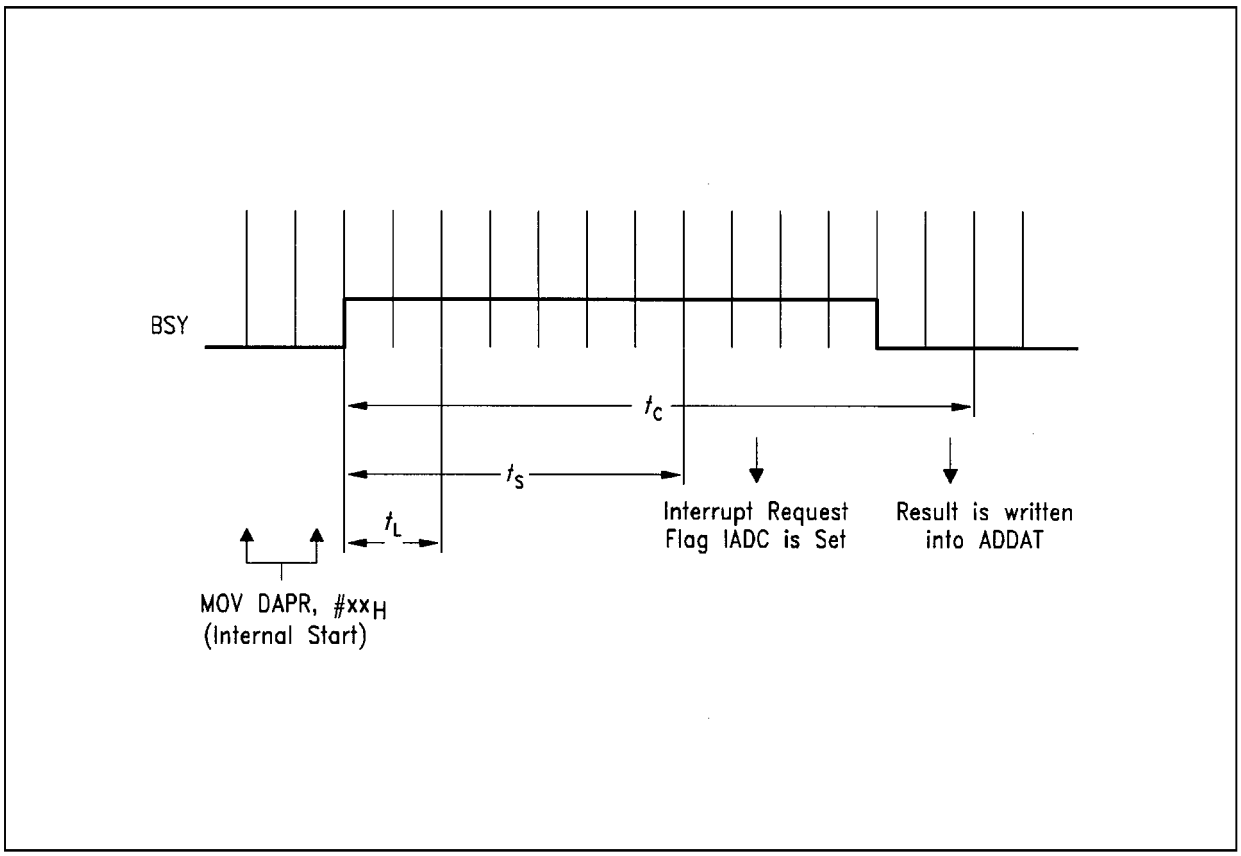


Bild 10.7: Timingdiagramm des A/D-Wandlers im 80(C)515/-535

11 Timer 2 in Verbindung mit Capture/Compare/Reload

Der Timer 2 mit seinen zusätzlichen Capture/Compare/Reload-Möglichkeiten ist eine der mächtigsten Peripherieteile im 80(C)515/-535. Diese Gruppe, sie wird an anderer Stelle auch als Capture/Compare Unit (CCU) bezeichnet, ist hervorragend geeignet zum Einsatz in Anwendungen, die die Erzeugung bzw. Erfassung komplexer Signalverläufe erfordern, wie z.B. Impulsgenerierung, Pulsweitenmodulation, Impulsbreitenmessung usw. Aus diesem Grund kommt die CCU in verschiedenen Automobilapplikationen ebenso zum Einsatz wie im industriellen Bereich (Gleich- Wechselstromsteuerungen, Schrittmotorsteuerung, Frequenzerzeugung, ...).

11.1 Begriffsdefinitionen

Je nach Konfiguration bietet der Timer 2 mit den angeschlossenen Registern unterschiedliche Funktionen an. Die grundlegenden Begriffe werden kurz erläutert.

11.1.1 Compare

Der Inhalt des Timers wird ständig mit dem Inhalt des Compare-Registers verglichen. Bei Übereinstimmung werden ein oder mehrere Ereignisse (je nach Konfiguration) ausgelöst. Damit kann durch geeignete Wahl des Compare-Wertes der Zeitpunkt für den Eintritt eines Ereignisses festgelegt werden. Beim 80(C)515/-535 können bis zu vier pulsdauermodulierte Signale mit maximal 65.535 Schritten und einer zeitlichen Auflösung von 1 µs realisiert werden.

11.1.2 Reload

Nach einem Überlauf läuft der Timer nicht mit dem Wert 0000H weiter, sondern mit dem im Reload-Register abgelegten Nachladewert. Dadurch kann die Timer-Periode (die Zeit von Überlauf zu Überlauf) über den Reload-Wert variiert werden.

11.1.3 Capture

Auf ein bestimmtes Ereignis hin wird der Inhalt des laufenden Timers in das Capture-Register übernommen. Dadurch läßt sich der Zeitpunkt des Ereigniseintritts festhalten. Im 80(C)515/-535 stehen bis zu vier High-Speed-Eingänge mit einer Auflösung von 1 µs zur Verfügung.

11.2 Konfiguration

Das Funktionschaltbild 11.1 zeigt die allgemeine Konfiguration von Timer 2 mit seinen Compare/Capture/Reload-Registern.

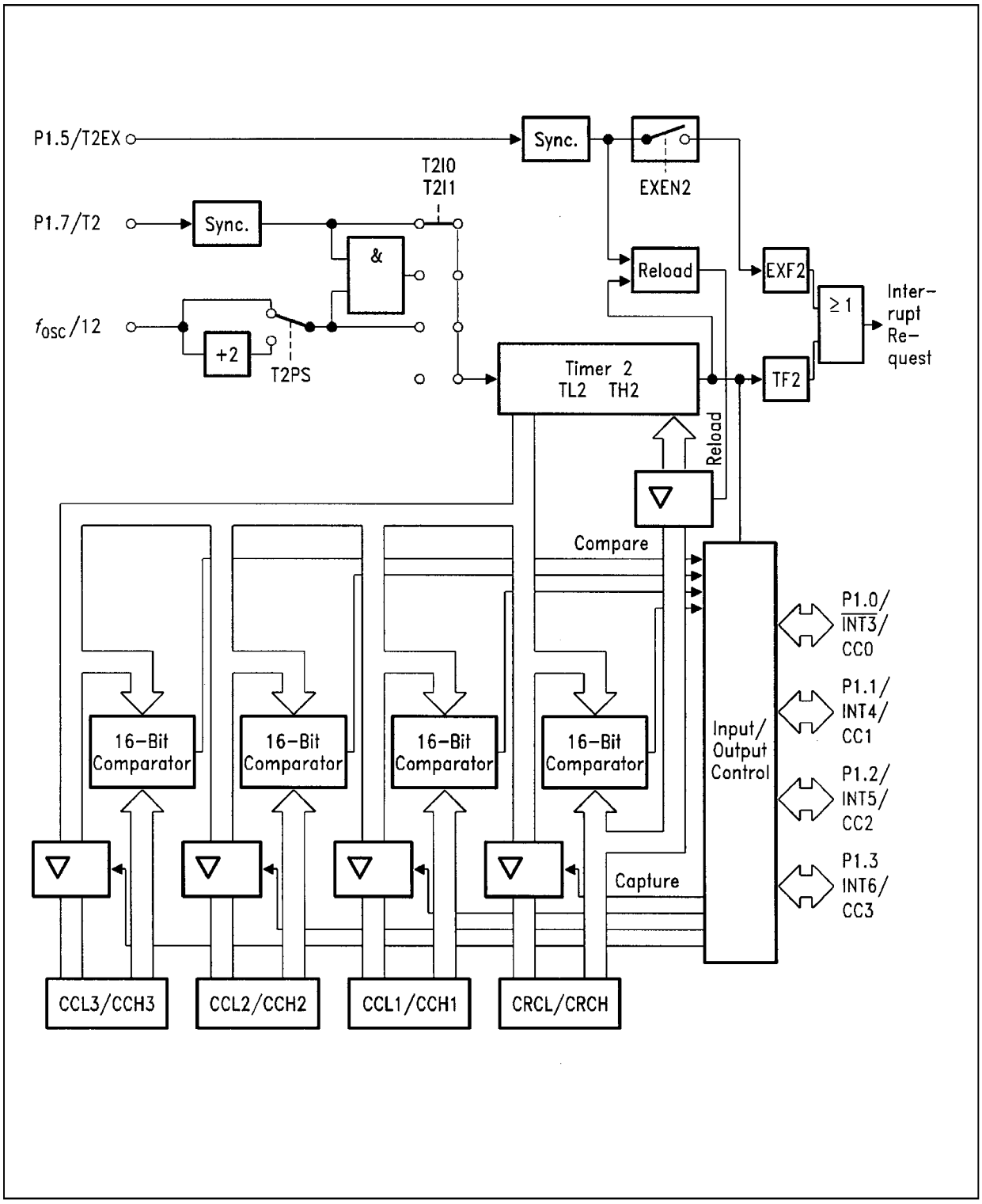


Bild 11.1: Blockschaltbild von Timer 2 mit Compare/Capture/Reload-Registern

Im Zusammenhang mit der CCU hat Port 1 alternative Funktionen. Normalerweise steuert jedes Compare-Register jeweils einen Portpin. Die folgende Tabelle listet diese Alternativfunktionen auf.

Pinsymbol	Eingang (I) Ausgang (O)	Funktion
P1.0/ $\overline{INT3}$*/CC0*	I/O	Ext. Interrupt 3, Comp. Output/Capt. Input für CRC-Register
P1.1/INT4/CC1	I/O	Ext. Interrupt 4, Comp. Output/Capt. Input für CC1
P1.2/INT5/CC2	I/O	Ext. Interrupt 5, Comp. Output/Capt. Input für CC2
P1.3/INT6/CC3	I/O	Ext. Interrupt 6, Comp. Output/Capt. Input für CC3
P1.5/T2EX	I/O	Ext. Reload-Trigger für Timer 2
P1.7/T2	I/O	Ext. Eingang für Timer 2

Tabelle 11.1: Alternative Funktionen der Portpins *P1.x* in Verbindung mit Timer 2

Wie aus Bild 11.1 ersichtlich, gruppieren sich um den Timer 2 eine Reihe von Special Function Register, die in der folgenden Tabelle kurz vorgestellt werden sollen.

Symbol	Beschreibung	Adesse
CCEN	Compare/Capture-Freigaberegister	0C1H
CCH1	Compare/Capture-Register 1, higher Byte	0C3H
CCL1	Compare/Capture-Register 1, lower Byte	0C2H
CCH2	Compare/Capture-Register 2, higher Byte	0C5H
CCL2	Compare/Capture-Register 2, lower Byte	0C4H
CCH3	Compare/Capture-Register 3, higher Byte	0C7H
CCL3	Compare/Capture-Register 3, lower Byte	0C6H
CRCH	Compare/Capture/Reload-Register, higher Byte	0CBH
CRCL	Compare/Capture/Reload-Register, lower Byte	0CAH
IRCON	Interrupt Control Register	0C0H
TH2	Timer 2, higher Byte	0CDH
TL2	Timer 2, lower Byte	0CCH
T2CON	Timer 2 Control Register	0C8H

Tabelle 11.2: Die Special Function Register um Timer 2

11.3 Timer 2

Die Betriebsarten des Timer 2 sind Timer-Modus, Event-Counter oder Gated-Timer-Modus. Die Auswahl der Betriebsart und des Zähltaktes erfolgt durch entsprechendes Setzen der Bits im SFR T2CON.

T2PS	I3FR	I2FR	T2R1	T2R0	T2CM	T2I1	T2I0
CFH (MSB)	CEH	CDH	CCH	CBH	CAH	C9H	C8H (LSB)

Bit	Funktion
T2I1 T2I0	Auswahlbits für die Arbeitsweise des Timers; Takteinspeisung:
0 0	keine Taktquelle freigegeben, Timer 2 hält an
0 1	Timer 2 wird vom internen Oszillator gespeist, die Frequenz hängt vom Vorteiler ab (T2PS)
1 0	Timer 2 arbeitet im Zählmodus und wird vom externen Signal am Pin *P1.7/T2* getaktet
1 1	Timer 2 arbeitet als Gated-Timer, der intern gewählte Takt wird durchgelassen, solange am Pin *P1.7/T2* High-Pegel anliegt
T2PS	Timer 2 Vorteilerbit (Prescaler): wenn T2PS=1 arbeitet der Timer 2 im Modus Timer oder Gated-Timer mit 1/24 der Oszillatorfrequenz, ist T2PS=0, arbeitet Timer 2 mit 1/12 der Oszillatorfrequenz. T2PS muß 0 sein, wenn Timer 2 als Counter arbeiten soll.
T2R1 T2R0	Bits zur Auswahl des Nachlademodus:
0 X	Nachladen abgeschaltet; der Timer 2 beginnt nach jedem Überlauf mit dem Startwert 0000H
1 0	Modus 0, Autoreload nach jedem Überlauf
1 1	Modus 1, Nachladen bei der fallenden Flanke am Pin *P1.5/T2EX*
I3FR	Flankenwahl für Interrupt 3 (steigende oder fallende Flanke). Damit wird auch die Flanke des internen Capt./Comp-Pulses für das Register CRC bestimmt. Ist I3FR=1 erscheint am Portpin *P1.0/CC0* eine ansteigende Fanke; bei I3FR=0 eine abfallende
T2CM	Auswahlbit des Compare-Modus für die Register CRC, CC1-CC3. Wenn T2CM=1 ist der Compare-Modus 1 gewählt. Bei T2CM=0 der Compare-Modus 0

Bild 11.2 Special Function Register T2CON (0C8H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

11.3.1 Timer-Modus für Timer 2

Das ist die klassische Betriebsart des Timers. In diesem Modus wird der Timer 2 mit der gewählten Zähltaktrate inkrementiert. Nach Erreichen des maximalen Wertes (FFFFH) läuft der Zähler über und beginnt wieder von vorne (bei 0000H) oder beim Reload-Wert aus dem Register CRC.

Der Zähltakt wird von der Oszillatorfrequenz abgeleitet. Er läßt sich mit dem Vorteilerbit T2PS aus dem Special Function Register T2CON einstellen. Ist T2PS zurückgesetzt, arbeitet der Timer 2 mit 1/12 der Oszillatorfrequenz, bei gesetztem T2PS mit 1/24 der Oszillatorfrequenz.

11.3.2 Gated-Timer-Modus für Timer 2

Dies ist ein Sonderfall des normalen Timer-Modes. Der Timer 2 wird nur inkrementiert, d.h. der Takt wird nur dann durchgelassen, solange am Pin *P1.7* High-Pegel anliegt. Bei Low-Pegel an *P1.7* hält Timer 2 an. Der Pin wirkt also wie eine Aktivierung, innerhalb deren der Timer den Takt sehen bekommt. Auf diese Weise können Impulsweiten einfach durch Ablesen des Timer-2-Inhalts gemessen werden. Der Pegel am Pin wird einmal in jedem Maschinenzyklus abgetastet.

11.3.3 Event-Counter-Modus für Timer 2

In dieser Betriebsart ist der Timer 2 vom internen Eingangstakt abgetrennt. Statt dessen wird nur bei der fallenden Flanke am Pin *P1.7/T2* inkrementiert. Timer 2 arbeitet damit als Ereigniszähler.

Auch hier wird der Eingangspin einmal pro Zyklus abgetastet. Ein Pegelwechsel wird dadurch erkannt, daß in einem Maschinenzyklus High- im nächsten Low-Pegel erkannt wird. Da also nicht die Flanke ausgewertet wird, muß bei einem High-Low-Übergang sowohl der vorausgehende High-Pegel als auch der nachfolgende Low-Pegel mindestens einen Zyklus lang anliegen, damit der Übergang bemerkt wird. Es sind also mindestens zwei Zyklen (24 Oszillatorperioden) notwendig, um Timer 2 einmal inkrementieren zu lassen (im einen wird High-Pegel, im nächsten Low-Pegel abgetastet). Damit ist die maximale Frequenz für abzutastende Signale:

$$f_{max} = f_{osz}/24.$$

Beachte: Um die korrekte Funktion des Zählermodus zu gewährleisten, muß der Timer-2-Vorteiler abgeschaltet sein, d.h. T2PS muß 0 sein.

11.3.4 Nachladen des Timers 2 - Reload

Die Nachladeschaltung des Timer 2 wird durch die Bits T2R0 und T2R1 im Special Function Register T2CON programmiert. Es stehen zwei Modi zur Auswahl:

Im Modus 0 erfolgt das Nachladen nach einem Überlauf von FFFFH. Dabei wird nicht nur das Interrupt-Request-Flag TF2 im Special Function Register IRCON gesetzt, sondern auch der 16 Bit breite Inhalt des CRC-Registers (der vorher mit dem gewünschten Reload-Wert beschrieben worden ist) in den Timer 2 geladen. Das Nachladen geschieht im gleichen Maschinenzyklus in dem TF2 gesetzt wird. Dadurch wird der eben erreichte Zählerstand 0000H überschrieben.

Diese Möglichkeit der Timerkonfiguration unterstüzt die Erzeugung genauer pulsweitenmodulierter Signale, da es mit diesem Modus leicht möglich ist, durch Verändern des Nachladewertes die Überlauffrequenz bzw. die Zeit von Überlauf zu Überlauf des Timer 2 zu variieren.

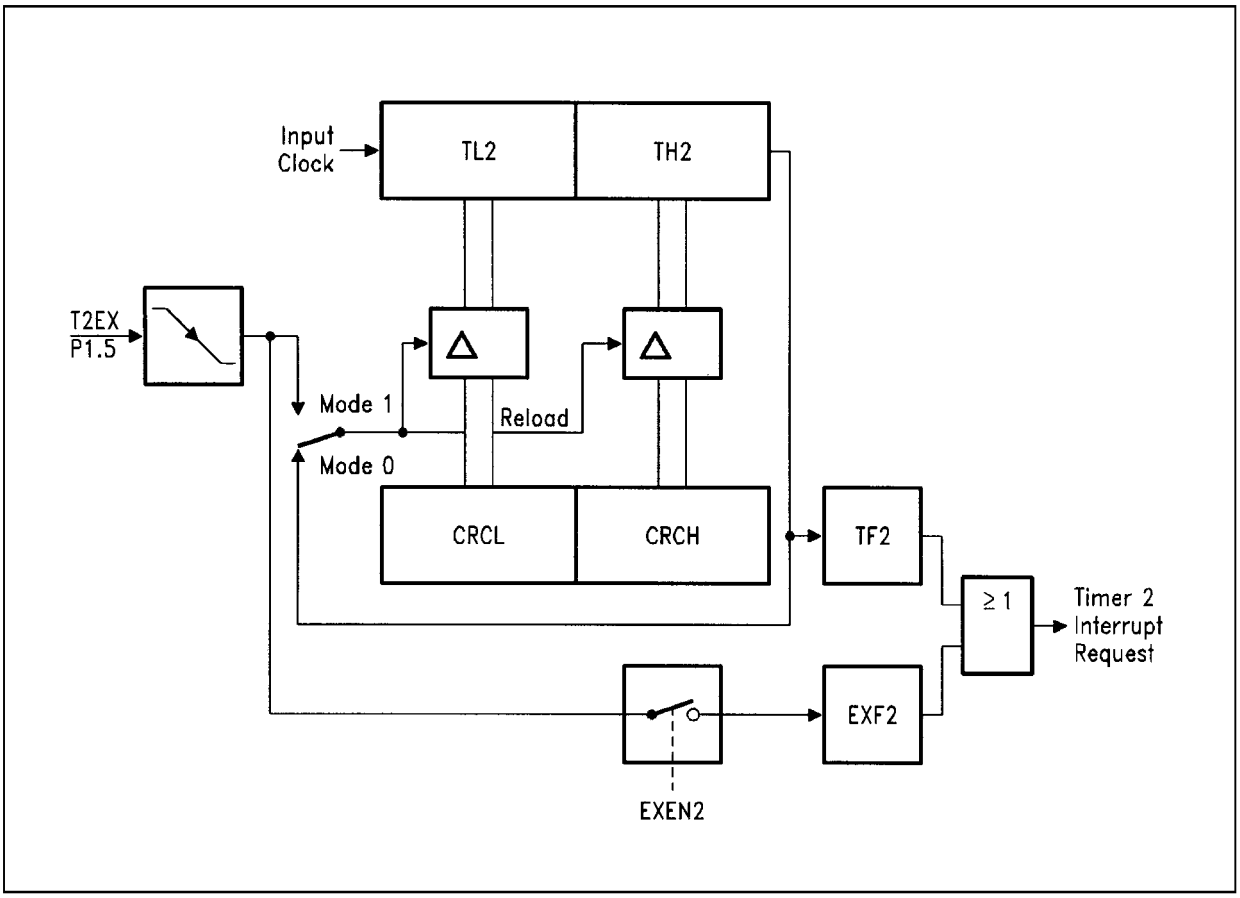


Bild 11.3: Timer 2 im Reload-Modus

Im Modus 1 löst eine fallende Flanke am Eingangspin *P1.5/T2EX* das Nachladen aus. Auch hier bedeutet die fallende Flanke, daß in einem Maschinenzyklus am Pin High-Pegel, im nächsten Low-Pegel anliegen muß, damit ein Pegelübergang erkannt werden kann. Der Pinzustand wird einmal pro Maschinenzyklus abgefragt. Der 16 Bit breite Wert von Register CRC wird dann im Zyklus nach Erkennen des Pegelübergangs in den Timer 2 übernommen. Zusätzlich wird dabei das Flag EXF2 im Special Function Register IRCON gesetzt, falls das Freigabebit EXEN2 im SFR IEN1 gesetzt ist. Darüberhinaus löst ein Setzten von EXF2 den Überlauf-Interrupt des Timer 2 aus, falls er freigegeben ist. Übrigens kann der Pin *P1.5/T2EX* auch ohne den Reload-Modus 1 als ganz normaler externer Interrupt verwendet werden.

11.4 Compare-Funktionen

11.4.1 Prinzip der Compare Funktion

Eine der effekektivsten und einfachsten Möglichkeiten, präzise Signale und Impulsfolgen zu erzeugen, ist die bereits mehrfach erwähnte Compare-Funktion. Dabei vergleicht ein Komparator eine stetig hochzählende Zeitbasis, hier den Timer 2, ständig mit einem der 16 Bit breiten Compare-Register. Während sich der Inhalt des Timer mit jedem Zeitinkrement um eins erhöht, ist

der Wert im Compare-Register zunächst einmal fest. Sobald die Inhalte von Timer und Compare-Register gleich sind, wird ein Ereignis (z.B. Pegelwechsel am Portpin oder Anfordern eines Interrupts) ausgelöst. Im Prinzip entspricht dies der Funktion eines Weckers, wobei der Timer den normalen Uhrzeigern und der Inhalt des Compare-Registers dem Weckzeiger entspricht. Das Klingeln des Weckers wäre dann das Compare-Ereignis.

Es ist jedoch zu beachten, daß - wie beim Wecker - ein Gleich-Vergleich stattfindet, nicht etwa ein Größer/Gleich-Vergleich. Das bedeutet, daß kein Ereignis stattfindet, wenn das Compare-Register mit einem Wert beschrieben wird, den der Timer 2 bereits durchlaufen hat. Erst beim nächsten Überlauf, wenn der Timer wieder an diesem Wert vorbeikommt, wird das gewünschte Ereignis stattfinden. Das bedeutet aber auch, daß nie ein Ereignis ausgelöst wird, wenn der Timer 2, z.B. aufgrund eines ungeeigneten Nachladewertes, nie den Wert des Compare-Registers erreicht.

Beispiel: Das Compare-Register CC1 enthält den Wert 0001H, während der Timer 2 mit dem Wert 0002H vom Register CRC nachgeladen wird. Da der Timer 2 nach dem Überlauf von FFFFH sofort mit 0002H belegt wird, nimmt er nie den Wert 0001H an. Somit kann auch kein Compare-Ereignis stattfinden.

Der Inhalt eines Compare-Registers kann also als ein fester Zeitpunkt verstanden werden, zu dem ein vorher festgelegtes Ereignis stattfindet. Eine Variierung des Inhalts verändert den Zeitpunkt des Ereignisses. Dies kann, abhängig vom Compare-Modus, zur Erzeugung von PWM-Signalen - durch Verschiebung der Einschaltflanke innerhalb einer festen Periodendauer - ebenso verwendet werden wie zur Generierung von beliebig gestalteten Rechtecksignalen.

11.4.2 Compare-Modus 0 (PWM-Modus)

Im Modus 0 wechselt der Pinpegel von Low auf High, sobald der Timerinhalt mit dem Inhalt des Compare-Registers übereinstimmt. Beim Überlauf des Timers geht der Pegel automatisch auf Low-Pegel, unabhängig von seinem bisherigen Zustand. Solange der Compare-Modus 0 gewählt ist, wird der Pin ausschließlich von der CCU geseteuert, nicht vom Benutzerprogramm. Ein Schreibbefehl in das betreffende Port-Latch bleibt wirkungslos. Bild 11.4 zeigt das Funktions-schaltbild der Portstruktur im Compare-Modus 0. Das Port-Latch wird direkt von den Signalen Compare und Timer-2-Überlauf gesteuert. Der interne Datenbus und die Schreibleitung sind vom Latch-Eingang abgetrennt.

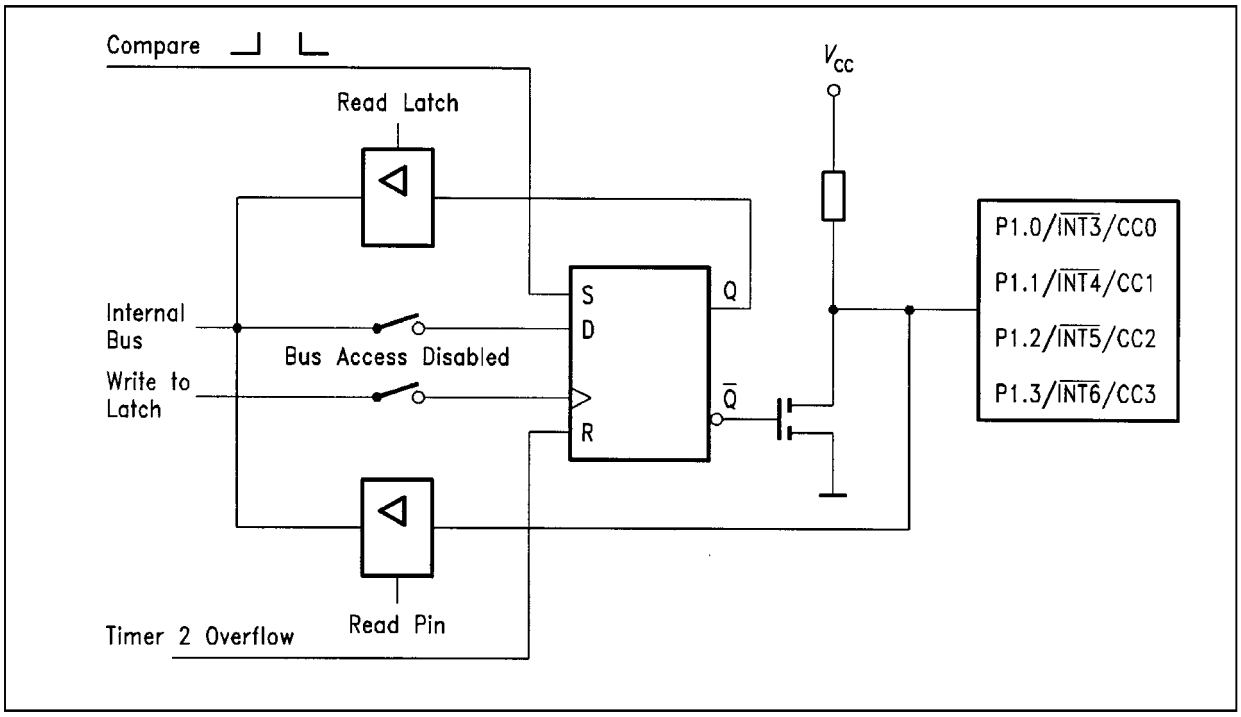


Bild 11.4: Port-Latch im Compare-Modus 0

Der Compare-Modus 0 eignet sich gut zur Erzeugung pulsweitenmodulierter Signale (daher PWM-Modus), weil mit der fallenden Flanke am Ende jedes Timerzyklus eine feste Grundfrequenz realisiert wird (einstellbar durch Verändern des Nachladewertes), während durch die Wahl des Compare-Wertes die Lage der steigenden Flanke innerhalb der Timerperiode und damit die Pulsdauer festgelegt wird. Anwendungsfälle des Modus 0 können die Steuerung von Gleich- oder Drehstrommotoren oder die Digital-Analog-Wandlung mit externem Tiefpaß sein.

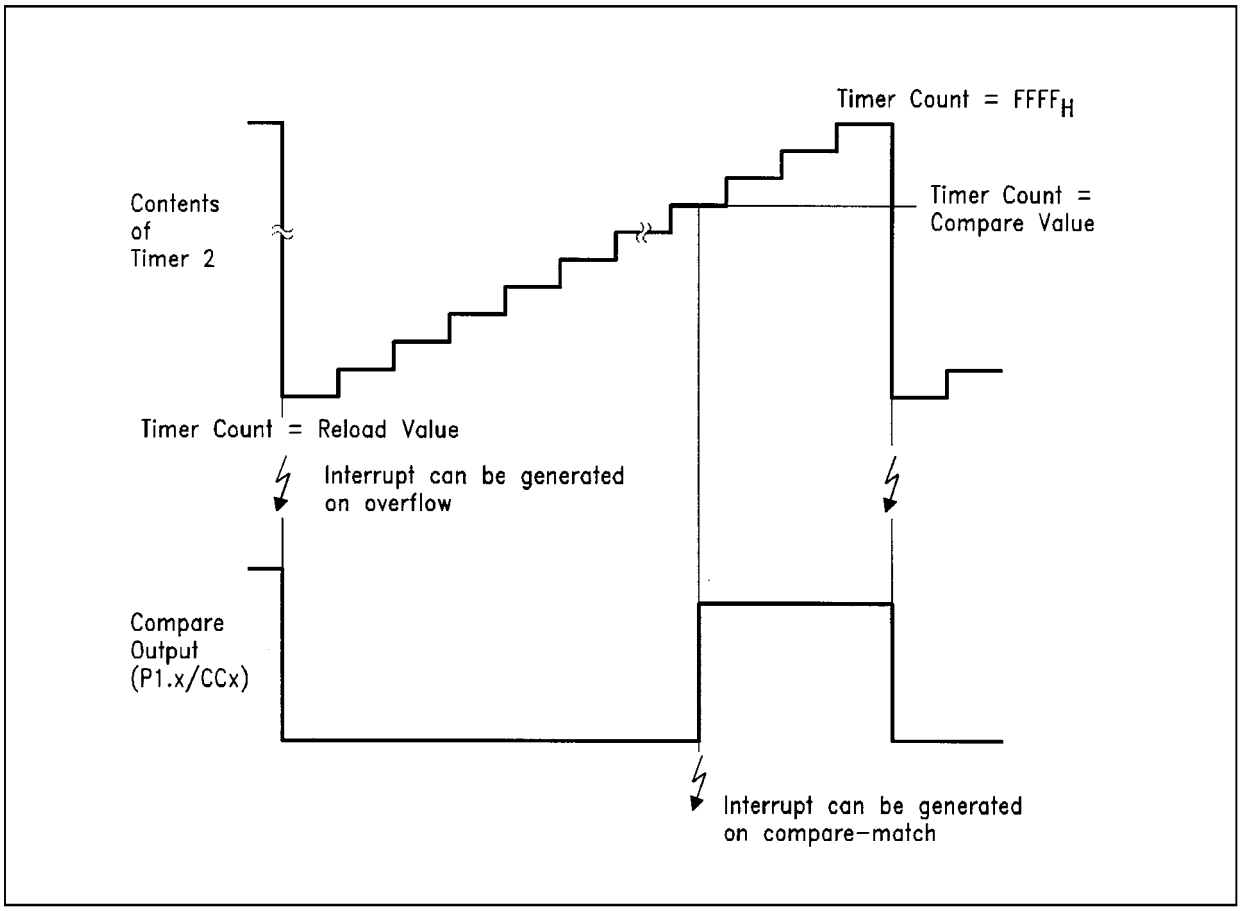


Bild 11.5: Funktion des Compare-Modus 0

Dieser Modus benötigt keine CPU-Zeit, sobald er einmal initialisiert ist. Lediglich bei Änderungen der PWM-Periode (Ändern des Nachladewertes) oder Pulsdauer (Ändern des Compare-Wertes) muß die CPU eingreifen.

11.4.3 Compare-Modus 0 mit den Registern CRC und CC1 bis CC3

Wie aus Bild 11.1 ersichtlich, sind die Compare-Register CRC und CC1 bis CC3 fest dem Timer 2 zugeordnet und arbeiten auf die Portpins *P1.0* (CRC) bis *P1.3* (CC3). Alle vier Register (eigentlich sind es ja Registerpaare, die aus je zwei 8-Bit-Registern bestehen) sind multifunktional, d.h. außer dem Compare-Modus 0 können sie auch im Compare-Modus 1, im Capture-Modus, oder - nur CRC - im Auto-Reload-Modus arbeiten. Natürlich kann jedes Register nur jeweils eine Funktion erfüllen, d.h. daß z.B. CRC nicht als Nachladeregister für Timer 2 und gleichzeitig als Compare-Register dienen kann.

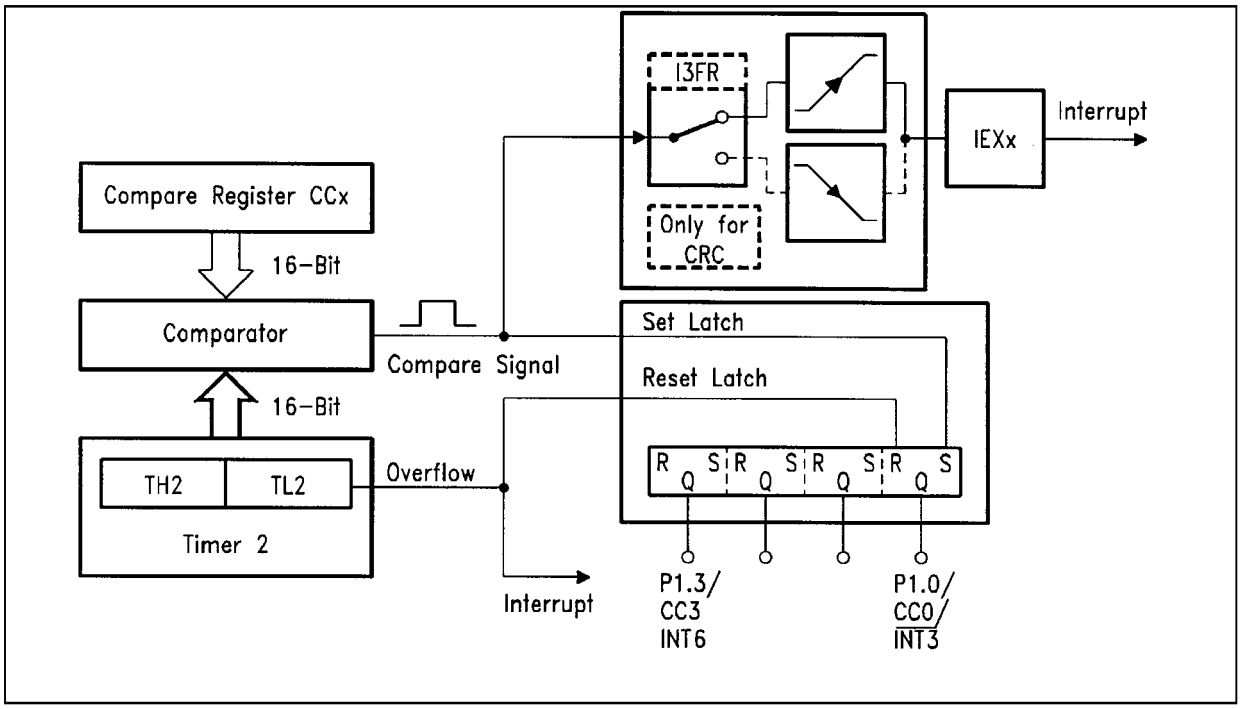


Bild 11.6: Timer 2 mit seinen Register CRC und CC1 bis CC3 im Compare-Modus 0

11.4.4 Initialisierung von CRC und CC1 bis CC3

Die generelle Einstellung auf einen Compare-Modus erfolgt individuell im Special Function Register CCEN. Darüberhinaus muß der gewünschte Compare-Modus mit dem Bit T2CM im SFR T2CON festgelegt werden. Die Register CRC und CC1 bis CC3 können zwar einzeln als Compare- oder Capture-Register konfiguriert werden, allerdings können die als Compare-Register initialisierten Register nur gemeinsam auf einen Compare-Modus festgelegt werden.

COCAH3 (MSB)	COCAL3	COCAH2	COCAL2	COCAH1	COCAL1	COCAH0	COCAL0 (LSB)

Bit		Funktion
COCAH3		Compare/Capture-Modus für CC3
COCAL3		Compare/Capture gesperrt
0	0	Capture bei steigender Flanke an Pin *P1.3/CC3*
0	1	Compare-Funktion eingeschaltet (Modus über T2CM festgelegt)
1	0	Capture bei Schreibzugriff auf CCL3 (Low-Byte von CC3)
1	1	
COCAH2		Compare/Capture-Modus für CC2
COCAL2		Compare/Capture gesperrt
0	0	Capture bei steigender Flanke an Pin *P1.2/CC2*
0	1	Compare-Funktion eingeschaltet (Modus über T2CM festgelegt)
1	0	Capture bei Schreibzugriff auf CCL2 (Low-Byte von CC2)
1	1	
COCAH1		Compare/Capture-Modus für CC1
COCAL1		Compare/Capture gesperrt
0	0	Capture bei steigender Flanke an Pin *P1.1/CC1*
0	1	Compare-Funktion eingeschaltet (Modus über T2CM festgelegt)
1	0	Capture bei Schreibzugriff auf CCL1 (Low-Byte von CC1)
1	1	
COCAH0		Compare/Capture-Modus für CRC
COCAL0		Compare/Capture gesperrt
0	0	Capture bei steigender Flanke an Pin *P1.0/CC0*
0	1	Compare-Funktion eingeschaltet (Modus über T2CM festgelegt)
1	0	Capture bei Schreibzugriff auf CRCL (Low-Byte von CRC)
1	1	

Bild 11.7: Special Function Register CCEN (0C1H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

11.4.5 Interrupts mit CRC und CC1 bis CC3 im Compare-Modus 0

Bei jeder Compare-Übereinstimmung wird auch ein Interrupt angefordert. Abhängig davon, ob der Interrupt freigegeben bzw. ob ein anderer Interrupt gleicher oder höherer Priorität in Bearbeitung ist, wird er auch bedient. Die Freigabe der Compare-Interrupts muß durch Setzen des jeweiligen Freigabebits im Special Function Register IEN1 geschehen.

Interrupts kann man nutzen, um zu erkennen, wann ein Compare-Wert vom Timer 2 bereits erreicht wurde. Die CPU kann daraufhin möglicherweise einen neuen Wert in das Compare-Register laden. Dadurch wird gewährleistet, daß nicht versehentlich der neue Compare-Wert vorzeitig geladen wird.

Als Compare-Ausgänge dienen die Portpins *P1.0* bis *P1.3* in einer ihrer alternativen Funktion. Eine weitere alternative Funktion ist die Verwendung als externe Interrupt-Eingänge. Sobald jedoch die Pins als Compare-Ausgänge benutzt werden, sind die Portleitungen vom Interrupt-System abgekoppelt (obwohl sie durch Software eingelesen werden können). Damit kann ein Pegelwechsel am Pin keinen Interrupt anfordern. Statt dessen wird der Interrupt von einem internen Compare-Signal ausgelöst. Das interne Compare-Signal (gemeint ist nicht das Signal am Portpin, siehe auch Bild 11.6) ist ein Signal, das vom Komparator ausgegeben wird und im Compare-Modus 0 letztlich das Port-Latch und den Interrupt-Request steuert. Diese Leitung wird aktiv (steigende Flanke), sobald der Timer- und der Registerinhalt übereinstimmen, und wieder inaktiv, sobald die Inhalte nicht mehr gleich sind. In Fällen, in denen der Vorteiler oder der externe Eingang zur Taktspeisung von Timer 2 verwendet wird, kann dieser interne Compare-Impuls relativ lang sein. Für das Compare-Register CRC kann die Flanke des internen Compare-Pulses ausgewählt werden, die den Interrupt anfordert. Dies geschieht mit dem Bit I3FR im SFR T2CON. Bei gesetzten Bit I3FR gilt die fallende Flanke. Für die Register CC1 bis CC3 wird der Interrupt bei der steigenden Flanke des internen Pulses angefordert. In jedem Fall bleibt der Zeitpunkt der steigenden Flanke des externen Signals am Portpin davon unberührt (die erscheint nämlich bei der steigenden Flanke des internen Compare-Signals).

Eine Besonderheit von Interrupts mit abfallender aktiver Flanke (hier nur bei CRC möglich) bedarf im Zusammenhang mit der Compare-Funktion noch der Erwähnung (gilt übrigens für alle Compare-Modi). Wenn am Portpin *P1.0* ein High-Pegel liegt, wird der Interrupt IEX3 sofort gesetzt, sobald ein Compare-Modus für das Register CRC selektiert wird. Der Grund ist, daß zuvor der Interrupt-Eingang direkt vom Portpin gesteuert wird. Somit sieht der Interrupt-Controller einen High-Pegel. Im Compare-Modus wird der Interrupt-Eingang ausschließlich vom Komparator gesteuert. Da beim Umschalten auf einen Compare-Modus im Normalfall die Inhalte von Timer 2 und CRC nicht übereinstimmen, ist das interne Compare-Signal auf Low-Pegel. Damit erkennt der Interrupt-Controller einen 1-0-Übergang, also eine negative Flanke, und setzt daher das Interrupt-Request-Flag IEX3.

Das Auslösen eines Interrupts beim Initialisieren des Compare-Registers kann aber auf zweierlei Weise verhindert werden:

- Nach Auswahl des Compare-Modus muß das Interrupt-Request-Flag gelöscht werden, bevor die Interrupts freigegeben werden.
- Der Compare-Modus wird selektiert, bevor für die Interrupts die fallende Flanke als die aktive Flanke gewählt wird.

11.4.6 Aussteuergrenzen für PWM

Allgemein kann man sagen, daß bei der Verwendung von n Bit breiten Registern für eine PWM-Erzeugung 2^n Stufen zur Einstellung der Pulsweite möglich sind. Nimmt man als erste Stufe eine Dauer-Low-Pegel an (die Pulsbreite is 0% der Periodendauer), berechnet sich die maximal mögliche Pulsdauer TH_{max} zu

$$TH_{\max} = \frac{2^n - 1}{2^n} * 100\% .$$

Dies bedeutet, daß eine Aussteuerung von 0% bis 100% nicht möglich ist, wenn Compare-Register und Timer die gleich Bitbreite haben. Es entsteht immer eine Impulsspitze (Spike) von der Länge eines Timer-Taktes (Bild 11.8).

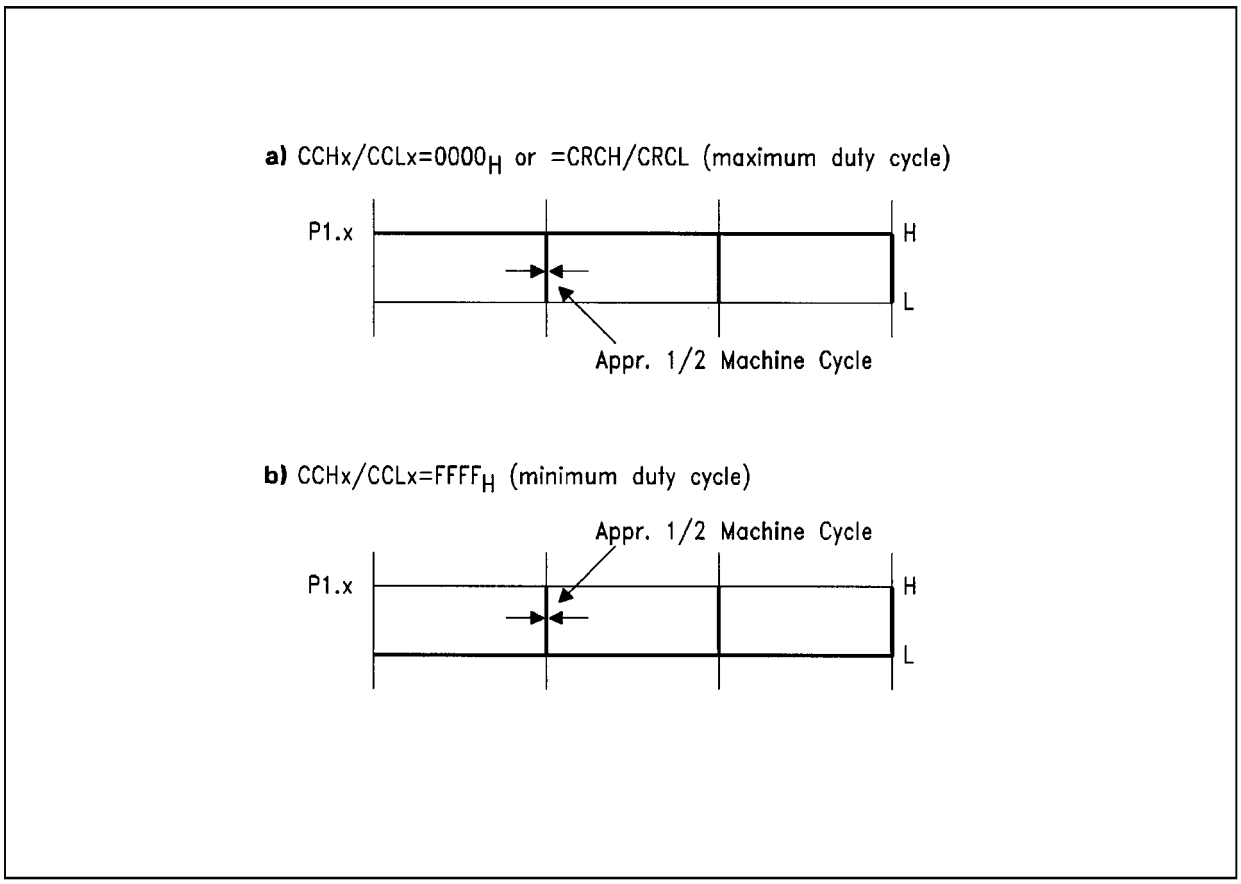


Bild 11.8: PWM-Aussteuerbereich mit Timer 2 im Compare-Modus 0

Im folgenden Beispiel wird eine Auflösung von 8 Bit angenommen (d.h. der Reload-Wert ist FF00H). TH_{max} ist dann:

$$TH_{\max} = \left(1 - \frac{1}{2^8}\right) * 100\% = \left(1 - \frac{1}{256}\right) * 100\% = 99{,}61\%$$

Der Aussteuerbereich mit dem Timer 2 und Registern CRC und CC1 bis CC3 ist in zwei Hälften geteilt. Ein Spike mit der halben Länge des Timer-Taktes entsteht zu Beginn der PWM-Periode, wenn der Compare-Wert gleich dem Timer-2-Nachaldewert ist, der zweite Spike am Ende der PWM-Periode, falls der Compare-Wert gleich dem maximalen Timer-Wert (FFFFH) ist.

Um keine Mißverständnisse zu lassen: Im normalen PWM-Betrieb entstehen keine Impulsspitzen! Lediglich für den Fall, daß ein Dauer-Low-Pegel (Compare-Wert = Reload-Wert) oder ein Dauer-High-Pegel (Compare-Wert = FFFFH) erzeugt werden soll, ergibt sich zu Beginn bzw. am Ende der PWM-Periode eine kurze Impulsspitze von je der halben Dauer eines Timer-Inkrements. Bei 12 MHz und dem Vorteiler 1/12 für den Timer 2 ist dieser Puls jeweils 500 ns lang.

11.4.7 Compare-Modus 1

Im Compare-Modus 1 hat die Software die Kontrolle über den Pegel am Portpin. Er wird üblicherweise angewendet, wenn Ausgangssignale nicht an eine konstante Periode gebunden sind (wie bei der PWM-Erzeugung), sondern präzise Signalimpulse mit eine sehr hohen Auflösung benötigt werden. Jeder Pinzustand wird von der Software bestimmt, der Zeitpunkt des Wechsels wird durch das Compare-Signal festgelegt. Die Compare-Ausgänge können als schnelle Signalausgänge (High Speed Outputs) betrachtet werden, wobei der Ausgabezeitpunkt unabhängig von der CPU ist.

Normalerweise gibt ein Portpin den logischen Wert aus, der in seinem Ausgangsregister (Latch) enthalten ist. Anders im Compare-Modus 1: Ist er gewählt, und wird im Programmlauf ein logische Bitwert (0 oder 1) in das zugehörige Port-Latch geschrieben, erscheint dieser Wert erst dann am Pin, wenn der Timerinhalt und der Inhalt des Compare-Registers übereinstimmen. Durch den Compare-Wert kann man also festlegen, zu welchem Zeitpunkt ein bestimmter Signalpegel (high oder low, je nachdem, welcher Wert zuvor in das Port-Latch geschrieben wurde) an den betreffenden Portpin weitergegeben wird.

Bild 11.9 zeigt ein Funktionsschaltbild des Timer/Compare-Register-Ports im Compare-Modus 1. In diesem Fall besteht das Port-Latch aus zwei Teilen. Das obere fungiert als Schatten-Latch (Shadow-Latch), das vom Datenbus aus zugänglich ist und im Programmlauf als Port-Latch beschrieben werden kann. Das untere Latch (Output-Latch), das direkt auf den Ausgangspin wirkt, übernimmt den zuvor (ins Shadow-Latch) eingschriebenen Bitwert gewissermaßen als Antwort auf eine Compare-Übereinstimmung.

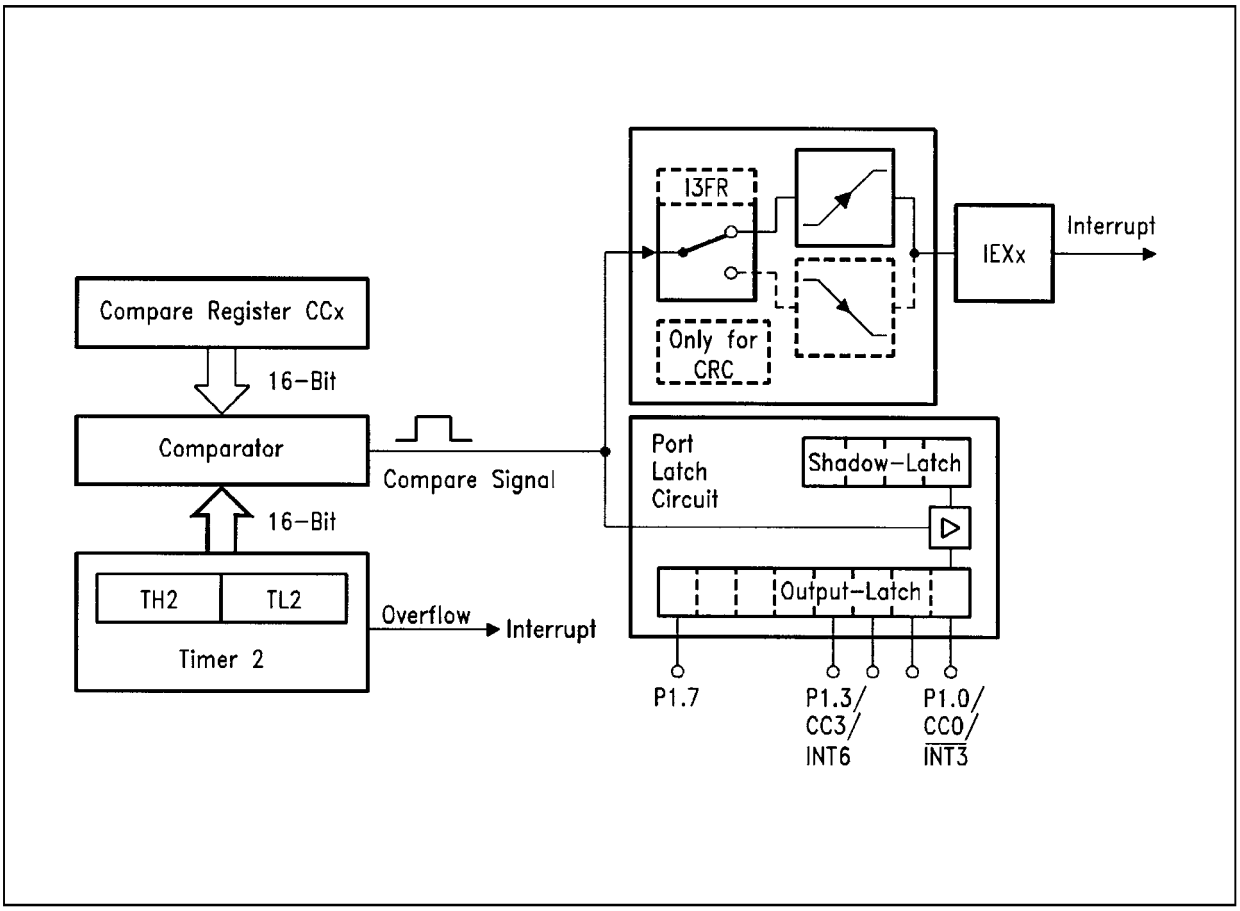


Bild 11.9: Timer 2 im Compare-Modus 1 mit Registern CRC und CC1 bis CC3

Es ist zu beachten, daß die doppelte Latch-Struktur transparent ist, solange das interne Compare-Signal aktiv ist. Während dieser Zeit wirkt jede Schreiboperation auch direkt am Pin. Das kann von Bedeutung sein, wenn der Timer 2 mit einem sehr langsamen Takt gespeist wird. In einem solchen Fall, in dem das interne Compare-Signal mehrere Maschinenzyklen lang aktiv ist, könnte die CPU den Inhalt des Ausgabe-Latch und damit den Pinpegel unbeabsichtigt ändern.

Dazu noch ein Hinweis: Ein Read-Modify-Write-Befehl liest das dem Benutzer zugängliche Schatten-Latch ein und schreibt den geänderten Wert auch dahin zurück. Hingegen liest ein normaler Lesebefehl wie gewohnt direkt den Pin des entsprechenden Compare-Ausgangs.

Das Register CRC arbeitet gewöhnlich als Reload-Register für den Timer 2, es kann auch als Compare-Register verwendet werden.

11.4.8 Initialisierung von CRC und CC1 bis CC3 im Compare-Modus 1

Wie bereits in Abschnitt 11.4.4 erwähnt, muß für jedes der Register CRC und CC1 bis CC3 die Funktion im Special Function Register CCEN separat gewählt werden. Allerdings kann für diese Register die Auswahl des Compare-Modus nur gemeinsam durch das Bit T2CM im SFR T2CON getroffen werden.

11.4.9 Interrupts im Compare-Modus 1

Bei einem Compare-Match, also bei Übereinstimmung des Timer 2 mit dem Inhalt des Compare-Registers, wird auch das zugehörige Interrupt-Request-Flag gesetzt und der Interrupt bedient, wenn er nicht gesperrt oder gerade durch einen anderen blockiert ist.

Der Interrupt des Registers CRC weist, wie bereits beim PWM-Modus erwähnt, die Besonderheit auf, daß das Interrupt-Flag je nach Wahl entweder bei der steigenden oder fallenden Flanke des internen Compare-Signals ausgelöst werden kann. Zur Erinnerung: Das interne Compare-Signal wird aktiv (steigende Flanke), sobald der Timer 2 den Compare-Wert erreicht, und wieder inaktiv, wenn der Timer 2 weiterzählt.

Ebenso wie im Compare-Modus 0 gilt auch im Compare-Modus 1 die Besonderheit bei der Initialisierung der Compare-Funktion im Zusammenhang mit Register CRC, daß nämlich unter bestimmten Voraussetzungen am Compare-Ausgang eine fallende Signalflanke erscheint (siehe auch Abschnitt 11.4.5).

11.5 Capture Funktionen

Wie bereits zu Beginn dieses Kapitels erwähnt, dient die Capture-Funktion dazu, auf ein Ereignis hin den aktuellen Stand des Timer 2 in einem der Capture-Register zu sichern, um dadurch den Zeitpunkt des Ereignisses festhalten zu können. Mit dieser Funktion läßt sich der Zeitabstand zwischen zwei Ereignissen (z.B. Signalflanken) genau messen.

Im 80(C)515/-535 sind die vier Compare/Capture-Register CRC und CC1 bis CC3 für die Capture-Funktion vorgesehen. Sie arbeiten generell mit dem Timer 2 und können dessen 16 Bit breiten Inhalt auf Anforderung hin übernehmen.

Zwei Capture-Modi können für jedes Register eingestellt werden: Im Modus 0 (Hardware-Capture) übernimmt das Capture-Register auf ein externes Signal am zugehörigen Pin den Inhalt des Timer 2. Im Modus 1 (Software-Capture) wird der aktuelle Stand des Timers 2 in das Capture-Register übertragen, wenn ein Schreibbefehl auf das niederwertige Byte des Capture-Registers erfolgt. Dieser Modus erlaubt, den Inhalt von Timer 2 während des Timerlaufs („on the fly“) festzustellen. Bei einem byteweisen Auslesen durch MOV-Befehle bestünde ansonsten die Gefahr, daß zwischen den beiden Ausleseoperationen ein Übertrag in das höherwertige Byte oder ein Überlauf des Timer 2 erfolgt, was dann einen falschen Timerinhalt vorspiegeln würde.

Im Capture-Modus 0 ist das externe Hardware-Ereignis:

- für die Register CC1 bis CC3 eine ansteigende Flanke an zugehörigen Pin des Ports 1 (*P1.1/CC1* bis *P1.3/CC3*)

- für das Register CRC je nach Programmierung eine steigende oder fallende Flanke am Pin *P1.0/CC0.* Die auslösende Flanke läßt sich über das Bit I3FR im SFR T2CON wählen. Bei gesetztem I2FR löst eine steigende Flanke, bei zurückgesetztem I3FR eine fallende Flanke am Pin *P1.0/CC0* einen Capture-Vorgang aus.

In beiden Fällen muß der zum Capture-Register gehörige Pin als Eingang konfiguriert sein, das Port-Latch muß also eine 1 enthalten. Der Eingangspin wird einmal pro Maschinenzyklus abgetastet. Falls der 80(C)515/-535 in einem Zyklus einen Low- und im darauffolgenden einen High-Pegel feststellt, wird eine steigende Flanke erkannt. Auf analoge Weise wird beim CRC-Register eine fallende Flanke erkannt. In jedem Fall muß also ein Pegelwechsel für mindestens einen Maschinenzyklus gültig bleiben, um sicherzustellen, daß der Mikrocontroller ihn auch erkennen kann. Der Inhalt von Timer 2 wird dann im auf die Flankenerkennung folgenden Maschinenzyklus in das betreffende Capture-Register übernommen.

Darüber hinaus setzt er Hardware-Capture-Modus auch das Anforderungsbit für den entsprechenden Interrupt (IEX3 bis IEX6). Ist dieser freigegeben, verzweigt der Mikrocontroller in die jeweilige Interrupt-Routine.

Im Capture-Modus 1 (Software-Capture) erfolgt eine Übernahme des Inhalts von Timer 2 bei einem Schreibbefehl in das niederwertige Byte des gewünschten Capture-Registers. Welcher Wert dabei eingtragen wird, ist gleichgültig; er wird vom Inhalt des Timers 2 im darauffolgenden Maschinenzyklus sowieso überschrieben. In diesem Modus wird kein Interrupt gefordert.

Die beiden Modi lassen sich für jedes der Capture-Register über zwei Bits im Special Function Register CCEN auswählen. Anders also als bei der Compare-Funktion, bei der sich Modus für die Register nur gemeinsam festlegen läßt, ist der Capture-Modus für jedes Register getrennt einstellbar. Damit kann z.B. für CC1 der Hardware-Capture-Modus selektiert werden, während CC2 im Software-Capture-Funktion arbeitet. Das Bild 11.10 gibt das Prinzipschaltbild der Capture-Funktion des Registers CRC wieder, während Bild 11.11 die Capture-Funktion der Register CC1 bis CC3 zeigt.

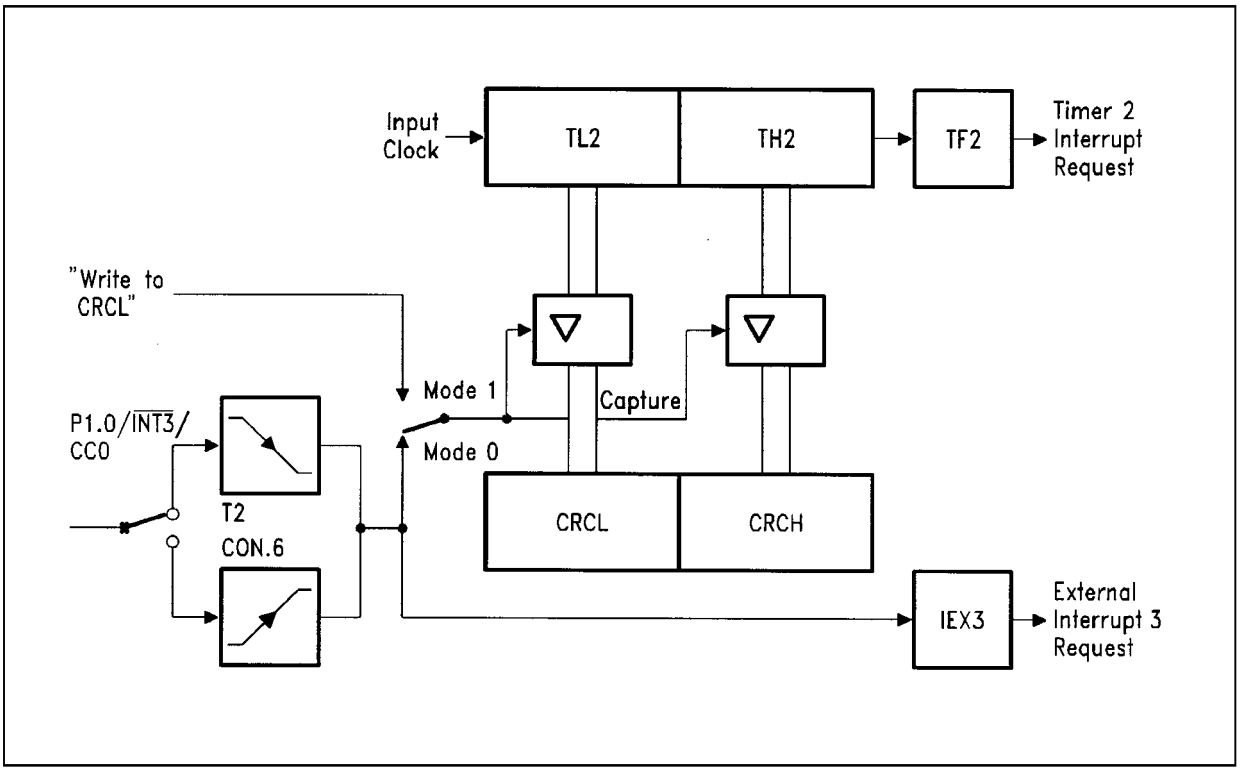


Bild 11.10: Capture-Funktion mit dem Register CRC

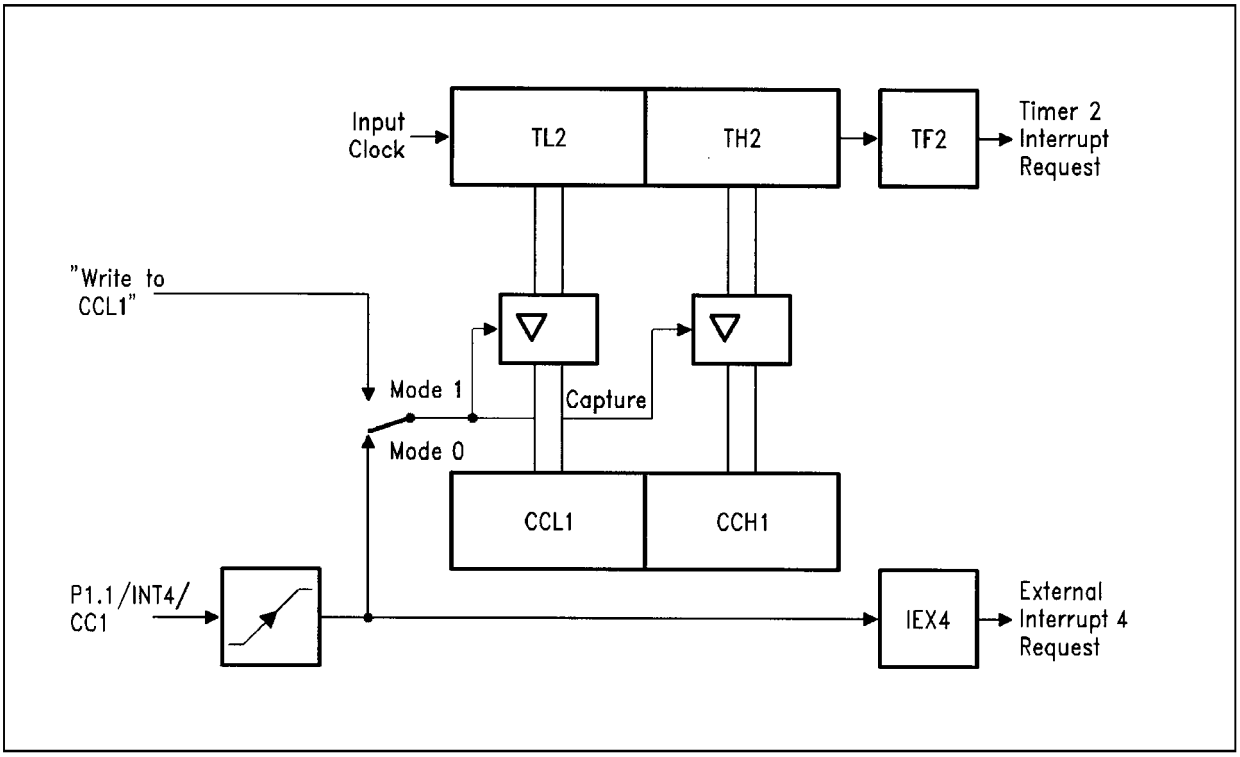


Bild 11.11: Capture-Funktion mit den Registern CC1 bis CC3

12 Reduzierung der Stromaufnahme

Um eine deutliche Reduzierung der Leistungsaufnahme zu ermöglichen, bietet der 80(C)515/-535 zwei Betriebsarten an, in denen die Stromaufnahme gegenüber dem normalen Betrieb herabgesetzt ist.

- Der Power-Down-Modus
  Der Oszillator ist abgeschaltet, alle Aktivitäten im 80(C)515/-535 werden vollständig angehalten (auch der Watchdog-Timer). Der Inhalt des internen Datenspeichers bleibt erhalten, solange die Versorgungsspannung nicht abgeklemmt wird oder unterhalb des spezifizierten Mindestwertes fällt. Dieser Modus braucht nur einen geringen Haltestrom und wird per Programm eingeschaltet. Er kann mit einem Hardware-Reset wieder aufgehoben werden.

- Der Idle-Modus (nur beim 80C515/-535)
  Die CPU ist vom Oszillator abgekoppelt. Dagegen werden alle Peripherieeinheiten weiterhin mit dem Takt versorgt und arbeiten weiter. Diese Betriebsart wird per Software initiiert und kann durch jeden freigegebenen Interrupt oder durch Reset wieder verlassen werden.

Im 80(C)515/-535 sind mehrere Maßnahmen ergriffen worden, um ihn auch bei rauhen Umgebungsbedingungen oder in Systemen mit hohen Sicherheitsanforderungen einsatzfähig zu machen. In derartigen Anwendungen muß ein ungewolltes Aktivieren der Stromreduktionsmodi unbedingt vermieden werden. Eine reduzierte Verfügbarkeit oder ein vollkommener Stillstand hätte fatale Folgen. Zudem nutzen oft gerade kritische Anwendungen den Watchdog-Timer zur Überwachung des Programmlaufs. Da dieser während der Stromreduktionsmodi abgeschaltet ist, wäre ein rein zufälliges Einschalten dieser Modi mit unabsehbaren Folgen verbunden.

Für die Software gesteuerten Stromreduktionsmodi sind Steuerbits im Special Function Register PCON untergebracht. Der hardwaregesteuerte Sparmodus benötigt keine Steuerbits.

SMOD (MSB)	PDS	IDLS	-	GF1	GF0	PDE	IDLE (LSB)

Bit	Funktion
PDS	Bit zur Einleitung des Power-Down-Modus (Power-Down Mode Start). Ein Schreibbefehl, der dieses Bit nach zuvor erfolgter Freigabe setzt, ist der letzte Befehl vor Inkrafttreten des Power-Down-Modus.
IDLS	Bit zur Einleitung des Idle-Modus (Idle Mode Start). Ein Schreibbefehl, der dieses Bit nach zuvor erfolgter Freigabe setzt, ist der letzte Befehl vor Inkrafttreten des Idle-Modus.
GF1, GF0	Flags zur allgemeine Verwendung (Generel Purpose Flag)
PDE	Bit zur Freigabe des Power-Down-Modus (Power Down Mode Enable). Setzten dieses Bits gibt den Power-Down-Modus frei.
IDLE	Bit zur Freigabe des Idle-Modus (Idle Mode Enable). Setzen dieses Bits gibt den Idle-Modus frei.

Bild 12.1: Special Function Register PCON (87H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

Die Stromreduktionsmodi der ACMOS- und MYMOS-Versionen unterscheiden sich. Während die MYMOS-Versionen (80515/-535) nur über einen Software-Power-Down-Modus verfügen, bieten die ACMOS-Versionen (80C515/-535) darüberhinaus einen Hardware-Power-Down-Modus und den Idle-Modus an.

12.1 Stromreduktionsmodus des 80515/-535

Der Software-Power-Down-Modus (SWPD) erlaubt in den Controllern 80515/-535 die Betriebsspannung am Pin V_{CC} abzuklemmen, während 40 Bytes des internen Datenspeichers gesichert werden. Voraussetzung ist, daß am Pin V_{PD} eine Notspannungsversorgung angeschlossen ist.

Solange die Spannung am Pin V_{CC} größer ist als die an V_{PD}, wird der gesamte Baustein und damit auch der interne Datenspeicher aus dieser Quelle versorgt. Im Störfall ist die Spannung am Pin V_{PD} größer als die an V_{CC}. Jetzt muß die Notversorgung sicherstellen, daß Inhalt des Datenspeicherbereichs von 58H bis 7FH erhalten bleibt. Der Strom, den die Notversorgung dann liefern muß, liegt bei etwa 1 mA bis maximal 3 mA.

Wird eine Störung in der Betriebsstromversorgung erkannt, so muß durch Software sichergestellt sein, daß relevante Daten in den gepufferten Bereich des internen Datenspeichers transferiert werden und die Notversorung aktiviert wird. Bevor die Spannung an V_{CC} unter den zulässigen Minimalwert abfällt soll durch einen Interrupt ein interner Reset ausgelöst und abgeschlossen sein und der Power-Down-Modus eingeleitet werden. Nach Wegfall der Störung sollte ein „Power-On-Reset“ durchgeführt werden und dafür Sorge getragen werden, daß die Notversorgung solange aktiviert bleibt, bis die Betriebsspannungsversorgung wieder normal arbeitet.

Zur Rettung der oben angesprochenen relevanten Daten eignet sich eine Interrupt-Routine mit sehr hoher Priorität.

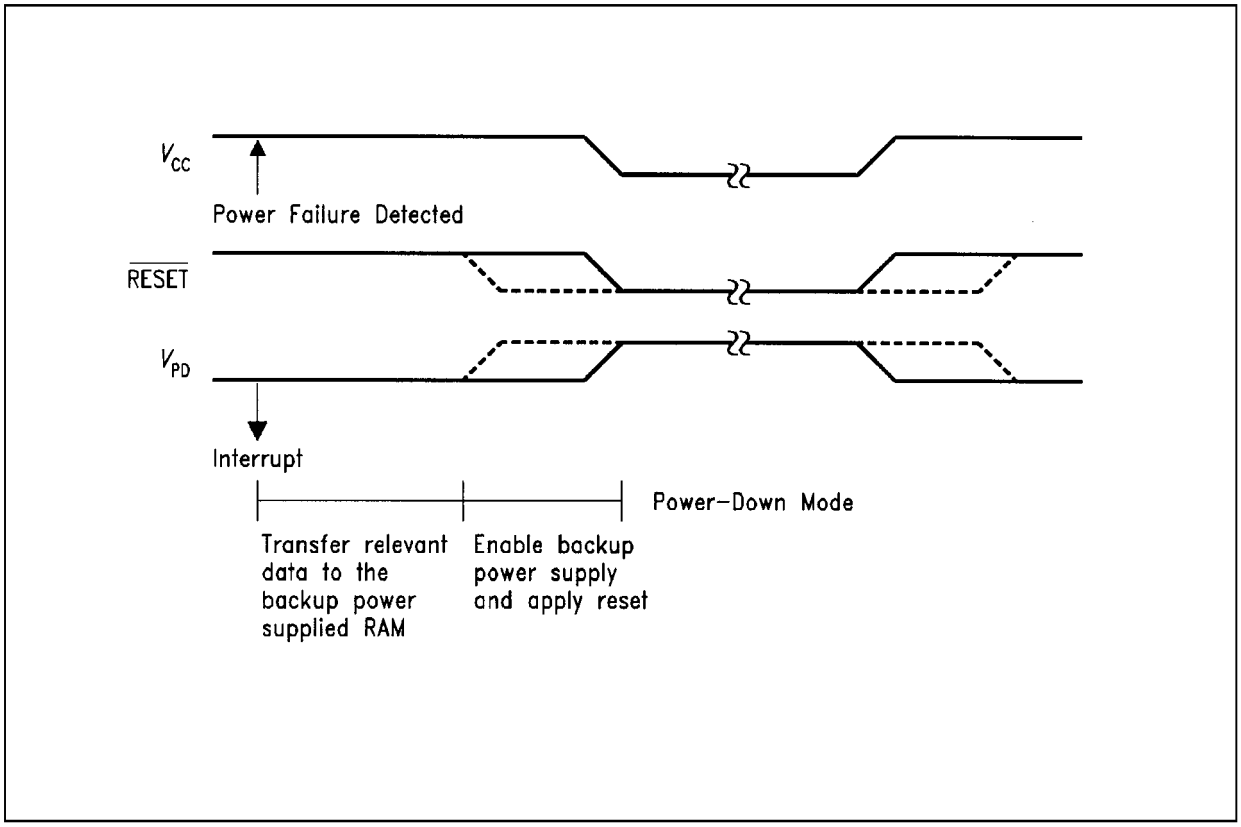


Bild 12.2: Reset und RAM-Backup Timing des 80515/-535

12.2 Stromreduktionsmodi des 80C515/-535

Neben dem oben beschrieben Software-Power-Down-Modus bieten Mikrocontroller der ACMOS-Versionen 80C515/-535, die nur über den Pin V_{CC} versorgt werden, Hardware-Power-Down-Modus und den Idle-Modus zur Stromreduktion.

12.2.1 Hardware-Freigabe der softwaregesteuerten Stromreduktionsmodi

Um einen möglichst wirkungsvollen Schutz gegen unbeabsichtigte Aktivierung der softwaregesteuerten Stromreduktionsmodi zu gewährleisten, ist beim 80C515/-535 ein gesonderter Pin vorgesehen, der deren Einschaltung blockiert. Dieser Pin $\overline{PE}$ finktioniert folgendermaßen:

- $\overline{PE}$ ist auf High-Pegel: Die Aktivierung der Stromreduktion ist nicht möglich. Die Befehlssequenzen, die diese Betriebsarten einschalten würden haben keine Wirkung.
- $\overline{PE}$ ist auf Low-Pegel: Alle Stromreduktionsmodi können wie in den folgenden Abschnitten beschrieben aktiviert werden.

Wenn der Pin $\overline{PE}$ extern offengelassen wird, sorgt ein schwacher Pullup-Widerstand für einen definierten High-Pegel. Damit ist ein eindeutiger Zustand hergestellt.

Während des Programmlaufs kann der Pegel an $\overline{PE}$ beliebig zwischen low und high wechseln, um die Stromreduktionsmodi zeitweise freizugeben oder zu blockieren. Ein Schmitt-Tigger am Eingang sorgt für zusätzliche Sicherheit gegen Störungen am Pin.

Zusätzlich zu den hardwareseitigen Vorkehrungen gibt es Schutzmaßnahmen auf der Softwareseite. So sind jeweils zwei Befehle in fester Folge notwendig, um den Power-Down- oder Idle-Modus einzuschalten. Diese Kombination von Vorsichtsmaßnahmen gewährleistet eine hohe Systemsicherheit. Dabei muß noch einmal betont werden, daß all dies nur für Modi gilt, die durch Software eingeschaltet werden können.

12.2.2. Anwendungsbeispiel für den Einsatz des Pins $\overline{PE}$

Ein Applikationsbeispiel für den Einsatz des Pins $\overline{PE}$ verdeutlicht die Funktion. In vielen Anwendungsfällen werden externe Komponenten (z.B. Spannungsregler) verwendet, die dem Controller einen Ausfall oder das Abschalten der Betriebsspannungsversorgung durch ein spezielles Signal an einem gewählten Pin anzeigen. Dieser Pin könnte zum einen mit dem Pin $\overline{PE}$, zum anderen mit einem externen Interrupt-Eingang verbinden werden. Folgende Schritte müßte der 80C515/-535 zur Einleitung des softwaregesteuerten Power-Down-Modus unternehmen, um die Datenhaltung durch eine Notversorgung zu ermöglichen:

- Bei Ausfall der Spannungsversorgung löst das Alarmsignal im Controller einen hochpriorisierten Interrupt aus. Die Interrupt-Routine rettet den aktuellen Programmstatus. Gleichzeitig wird $\overline{PE}$ auf Low-Pegel gezogen.

- Anschließend bringt sich der 80C515/-535 in den Power-Down-Modus. Über die Notversorgung wird der Datenerhalt im internen Datenspeicher gewährleistet.

12.2.3 Power-Down-Modus beim 80C515/-535

Im Power-Down-Modus wird der Oszillator abgeschaltet. Das heißt, daß sämtliche Aktivitäten im Baustein aufhören, lediglich die Inhalte des internen Datenspeichers und die Special Function Register bleiben erhalten. Die Portpins behalten ihren Zustand, den sie zum Zeitpunkt der Aktivierung des Power-Down-Modus haben, gleichgültig, ob sie als normale Ein-/Ausgabe-Pins arbeiten oder eine alternative Funktion erfüllen. Die Steuersignale für den externen Bus, *ALE* und $\overline{PSEN}$, haben im Power-Down-Modus Low-Pegel. Das Verhalten der Ports 0 und 2 hängt davon ab, ob aus internem oder externem Programmspeicher gearbeitet wurde. Eine Zusammenfassung über die Pegel an den Ports gibt Tabelle 12.1. Dabei bedeutet Portdaten/letzte Ausgabe, daß der Pin entweder den Inhalt seines Port-Latchs oder den letzten Ausgabewert ausgibt. Der Taktausgang (*P1.6/CLKOUT*) gibt, wenn er aktiviert wurde, einen Low-Pegel aus. Es ist zu beachten, daß Signal-Spannungen, die extern an die Pins angelegt werden, u.U. den Stromverbrauch im Power-Down-Modus erhöhen können (durch Querströme in den

Eingangsstufen). Dies kann dadurch vermieden werden, daß nur die Pegel, die im Datenblatt für die Testbedingungen beim Power-Down-Modus genannt sind, anliegen.

Wenn der Power-Down-Modus benutzt werden soll, muß er zuvor durch einen Low-Pegel am Pin $\overline{PE}$ freigegeben werden. Dies kann auch dynamisch kurz vor dem Auslösen des Power-Down-Modus geschehen. Wie später auch beim Idle-Modus sind auch hier zwei Schreibbefehle unmittelbar hintereinander notwendig, um den Power-Down-Modus auszulösen. Der erste muß das Bit PDE (PCON.1) und darf nicht das Bit PDS (PCON.6) setzten, der zweite muß PDS und darf nicht PDE setzten. Auch hier ist gewährleistet, daß bei gleichzeitigem Schreiben in beide Bitpositionen der Controller nicht in den Power-Down-Modus geht. Ebenso müssen die beiden Aktivierungsbefehle unmittelbar hintereinander ausgeführt werden. Dies Schaltung bewahrt den 80C515/-535 davor, bei einem eventuellen Programmabsturz unbeabsichtigt stehen zu bleiben. PDE und PDS werden automatisch nach dem Setzen wieder gelöscht, so daß beim Lesen beide Bits stets 0 enthalten.

Da das Special Function Register PCON nicht bitadressierbar ist, müssen die Bits über Bytebefehle gesetzt werden. Mit folgenden Programmzeilen kann der Power-Down-Modus aufgerufen werden:

```
ORL     PCON,#00000010B     ;Bit PDE setzen, PDS nicht setzen

ORL     PCON,#01000000B     ;Bit PDS setzen, PDE nicht setzen
```

Der Befehl, der das Bit PDS setzt, wird als letzter abgearbeitet, bevor der Power-Down-Modus aktiv wird. Werden Idle-Modus und Power-Down-Modus gleichzeitig (innerhalb derselben Befehle) aufgerufen, hat der Power-Down-Modus Vorrang.

Die einzige Möglichkeit, den Power-Down-Modus wieder zu verlassen, ist ein externer Hardware-Reset. Dabei werden alle SFR mit ihrem Reset-Wert neu belegt, aber der gesamte interne Datenspeicher bleibt unberührt.

Zur weiteren Stromeinsparung kann während des Power-Down-Modus die Versorgungsspannung herabgesetzt werden (siehe DC Characteristics im Datenblatt). Der Anwender muß allerdings dafür sorgen, daß die Spannung nicht vor Eintritt des Power-Down-Modus reduziert wird, bzw. daß sie vor Verlassen des Modus (d.h. vor der Aktivierung des Reset) wieder auf den normalen Wert eingestellt wird. Da das Reset-Signal neben der Beendigung des Power-Down-Modus auch das Anschwingen des Oszillators bewirkt, muß es ähnlich wie beim normalen Einschalten so lange aktiv sein, bis der Oszillator sich stabilisiert hat.

12.2.4 Idle-Modus

In dieser Betriebsart läuft der Oszillator des 80C515/-535 weiter, wobei die CPU von der Taktversorgung abgetrennt ist. Dagegen wird die interne Peripherie weiterhin mit dem normalen Takt gespeist. Das Interrupt-System, die serielle Schnittstelle, der A/D-Wandler sowie sämtliche Timer (außer dem Watchdog-Timer) bleiben voll funktionsfähig. Der Status der CPU bleibt während des Idle-Modus erhalten: Die Registerinhalte von Stack-Pointer, Programmzähler,

Programmstatuswort, Akku und der gesamten CPU-Register bleiben unverändert. Das gleich gilt selbstverständlich auch für den gesamten internen Datenspeicher.

Wie stark der Stromverbrauch in diesem Modus gesenkt werden kann, hängt davon ab, wie viele Peripheriekomponenten aktiv sind. Die größte Stromersparnis wird erreicht, wenn die Timer, die serielle Schnittstelle und der A/D-Wandler abgeschaltet sind. Das ist auch die Testbedingung für die Messung des Stromverbrauchs (siehe DC-Characteristics im Datenblatt). Der Anwender hat also dafür Sorge zu tragen, daß Peripherieteile, die nicht benötigt werden, während des Idle-Modus abgeschaltet sind, um eine möglichst große Stromreduktion zu erreichen.

Normalerweise behalten die Portpins ihren Zustand, wie er bei Aktivieren des Idle-Modus gewesen ist. Falls Pins in ihrer alternativen Funktion programmiert sind, sind sie weiterhin aktiv, wenn die zugehörige Funktion während des Idle-Modus eingeschaltet ist. Das gilt für die Compare-Ausgänge ebenso wie für den Systemtaktausgang und die serielle Schnittstelle. Die Steuersignale für den externen Bus, *ALE* und $\overline{PSEN}$ haben im Idle-Modus High-Pegel. Das Verhalten von Port 0 und Port 2 hängt davon ab, ob aus internem oder externem Programmspeicher gearbeitet wurde. Eine Zusammenfassung über die Pegel an den Pins gibt Tabelle 12.1. Dabei bedeutet Portdaten/letzte Ausgabe, daß der Pin entweder den Inhalt seines Port-Latchs oder den letzten Ausgabewert ausgibt.

Wie bei normalem Betrieb, können auch im Idle-Modus Portpins als Eingänge benutzt werden. So kann weiterhin ein Capture, ein Nachladen des Timer 2 oder eine A/D-Wandlung angestoßen werden, die Timer können zum Zählen externer Flankenwechsel verwendet werden, und externe Interrupts können ausgelöst werden.

Der Watchdog-Timer ist die einzige Peripherieeinheit, die während des Idle-Modus angehalten wird. Somit ist es auch bei eingeschaltetem Watchdog-Timer möglich, den CPU-Status bis zum Eintritt eines externen Ereignisses einzufrieren.

Zur Freigabe der Stromreduktionsmodi muß der Pin $\overline{PE}$ auf Low-Pegel liegen. Der Modus selbst wird durch zwei unmittelbar aufeinanderfolgende Schreibbefehle aktiviert. Der erste Befehl muß das Bit IDLE (PCON.0) setzen und darf nicht das Bit IDLS (PCON.5) setzen. Im darauffolgenden Befehl muß IDLS und darf IDLE nicht gesetzt werden. Treten diese beiden Befehle nicht unmittelbar hintereinander auf, sondern unterbrochen durch einen anderen Befehl, wird der Idle-Modus nicht aktiviert. Es ist durch eine interne Schaltung sichergestellt, daß auch gleichzeitiges Setzen von IDLE und IDLS den Idle-Modus nicht aktiviert. Dadurch, daß zwei Befehle notwendig sind, verringert sich die Wahrscheinlichkeit einer versehentlichen Aktivierung drastisch. Andernfalls wäre es möglich, daß bei einem Programmabsturz, ausgelöst z.B. durch äußere Störeinflüsse, der Mikrocontroller versehentlich in den Idle-Modus versetzt wird, was für die Sicherheit des Systems untragbar wäre, da ja der Watchdig-Timer als Überwachungseinheit dabei ausfallen würde.

Nach dem Setzen werden die beiden Bits automatisch wieder zurückgesetzt, so daß eine Leseoperation der Bits stets 0 anzeigt. Es ist zu beachten, daß das SFR PCON nicht bitadressierbar ist und daher über Maskenoperationen angesprochen werden muß. Mit folgenden Programmzeilen kann der Idle-Modus aufgerufen werden:

```
ORL     PCON,#00000001B     ;Bit IDLE setzen, ILDS nicht setzen
```

```
ORL     PCON,#00100000B     ;Bit IDLS setzen, IDLE nicht setzen
```

Der Befehl, der das Bit IDLS setzt, wird als letzter abgearbeitet, bevor der Idle-Modus aktiv wird.

Der Idle-Modus wird durch jeden freigegebenen Interrupt beendet. Die CPU nimmt ihre Tätigkeit wieder auf und verzweigt in die entsprechende Interrupt-Routine. Der erste Befehl, der nach Ende der Interrupt-Bearbeitung ausgeführt wird, ist der hinter dem Befehl, der das Bit IDLS gesetzt hat. Die andere Möglichkeit, den Idle-Modus zu verlassen, ist ein Hardware-Reset. Da der Oszillator noch läuft, genügt ein Reset-Impuls von zwei Maschinenzyklen Dauer.

Ausgänge	Letzter Befehl aus internem Speicher		Letzter Befehl aus externem Speicher	
	Idle-Modus	Power-Down-M.	Idle-Modus	Power-Down-M.
ALE	High-Pegel	Low-Pegel	High-Pegel	Low-Pegel
$\overline{PSEN}$	High-Pegel	Low-Pegel	High-Pegel	Low-Pegel
Port 0	Portdaten	Portdaten	Hochohmig	Hochohmig
Port 1	Portdaten/ Alternativfunktion	Portdaten/ letzte Ausgabe	Portdaten/ Alternativfunktion	Portdaten/ letzte Ausgabe
Port 2	Portdaten	Portdaten	Adresse	Portdaten
Port 3	Portdaten/ Alternativfunktion	Portdaten/ letzte Ausgabe	Portdaten/ Alternativfunktion	Portdaten/ letzte Ausgabe
Port4	Portdaten/ Alternativfunktion	Portdaten/ letzte Ausgabe	Portdaten/ Alternativfunktion	Portdaten/ letzte Ausgabe
Port 5	Portdaten/ Alternativfunktion	Portdaten/ letzte Ausgabe	Portdaten/ Alternativfunktion	Portdaten/ letzte Ausgabe

Tabelle 12.1: Zustand der externen Pins während Idle- und Power-Down-Modus

13 Der Watchdog-Timer

Der 80(C)515/-535 bietet als Überwachungseinheit, die bei Fehlfunktionen der Hard- oder Software einen definierten Zustand des Bausteins herstellen und ein „Versagen nach der sicheren Seite“ gewährleistet, den Watchdog-Timer (WDT) als 16 Bit breiten Zähler an.

Die Aufgabe des WDT ist, prinzipiell sicherzustellen, daß das ablaufende Programm korrekt arbeitet. Insbesondere soll er erkennen, wenn es aufgrund von Störungen aus seiner Bahn geworfen wird. Beispielsweise könnten elektromagnetische Einstreuungen auf den externen Bus einen falschen Programmcode erzeugen und die CPU so veranlassen, in einen nicht definierten Programmspeicherbereich zu springen. Nun ist es dem WDT nicht möglich, zwischen gewollten (sinnvollen) und falschen (unsinnigen) Befehlen zu unterscheiden, da die CPU in beiden Fällen einen Programmcode abarbeitet. Daher muß man sich die Arbeitsweise des Watchdog-Timers wie folgt vergegenwärtigen:

Wenn der WDT einmal gestartet ist, kann man ihn nicht mehr anhalten (außer durch einen externen Reset). Sobald er überläuft, löst er einen Hardware-Reset aus, der den Controller wieder definiert beginnen läßt. Diese Reset unterscheidet sich vom externen nur dadurch, daß der WDT weiterläuft und sein Statusflag setzt. Um den WDT vom Überlaufen abzuhalten, muß ihn die CPU immer wieder zurücksetzen und von vorne zählen lassen. Der Softwareentwickler muß also sicherstellen, daß im normalen Programmablauf der WDT immer wieder zurückgesetzt wird. Falls dann die CPU außer Kontrolle geraten sollte, wird der WDT nicht mehr zurückgesetzt und löst nach seiner Ablaufzeit (Zählerstand FFFCH) einen Reset aus. Der Mikrocontroller beginnt danach wieder von einem definierten Zustand an zu arbeiten. Dabei ist zu beachten, daß das Rücksetzten des WDT nicht in einer immer wiederkehrenden Interrupt-Routine erfolgt, da diese ja stets ausgeführt wird, unabhängig davon, wo sich die CPU gerade befindet. Somit würde der WDT zwar im Falle des Programmabsturzes immer wieder zurückgesetzt, das Programm würde anschließend aber weiterhin Unsinn machen. Vielmehr ist zu empfehlen, den WDT im Hauptprogramm an Stellen zurückzusetzen, die im normalen Betrieb zyklisch durchlaufen werden.

13.1 Starten und Rücksetzen des Watchdog-Timers

Durch Setzen des Bits SWDT im Special Function Register IEN1 wird der Watchdog-Timer gesetartet.

Einmal angestoßen, läßt sich der Watchdog-Timer nicht mehr durch Software anhalten, sondern er kann nur noch zyklisch zurückgesetzt werden. Dazu müssen in zwei aufeinanderfolgenden Befehlen zuerst das Bit WDT im Special Function Register IEN0 und dann das Bit SWDT im Special Function Register IEN1 gesetzt werden. Das Bit WDT wird im dritten Maschinenzyklus nach seinem Setzbefehl durch die Hardware wieder gelöscht. Bei der Resetsequenz müssen die beiden Befehle direkt nacheinander programmiert sein. Liegt z.B. auch nur ein NOP-Befehl zwischen den beiden, erfolgt kein Rücksetzen des Watchdog-Timers. Diese Doppelbefehlssequenz wurde implementiert, um im Falle des Programmabsturzes die Wahrscheinlichkeit herabzusetzten, daß der WDT durch einen Befehl zufällig gelöscht wird.

EXEN2	SWDT	EX6	EX5	EX4	EX3	EX2	EADC
BFH (MSB)	BEH	BDH	BCH	BBH	BAH	B9H	B8H (LSB)

Bit	Funktion
SWDT	Startbit des Watchdog-Timers. Wenn es gesetzt wird, startet der Watchdog-Timer. Wird SWDT bei laufendem Watchdog-Timer unmittelbar nach dem Bit WDT (im SFR IEN0) gesetzt, erfolgt ein Rücksetzen des Watchdog-Timers

Bild 13.1: Special Function Register IEN1 (0B8H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

EAL	WDT	ET2	ES	ET1	EX1	ET0	EX0
AFH (MSB)	AEH	ADH	ACH	ABH	AAH	A9H	A8H (LSB)

Bit	Funktion
WDT	Rücksetzbit des Watchdog-Timers (Watchdog Timer Flag). Es leitet den Rücksetzvorgang des Watchdog-Timers ein und muß unmittelbar vor dem Setzen des Bits SWDT (im SFR IEN1) gesetzt werden.

Bild 13.2: Special Function Register IEN0 (0A8H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

13.2 Watchdog-Timer-Status

Falls die Software aus irgendwelchen Gründen versäumt, den Watchdog-Timer zurückzusetzen, löst dieser beim Erreichen des Wertes FFFCH intern einen Reset aus, der vier Maschinenzyklen lang ist. Dieser interne Reset unterscheidet sich von einem externen Hardware-Reset dadurch, daß der WDT das Bit WDTS im Special Function Register IP0 setzt. WDTS wird nach einem externen Reset gelöscht, kann aber auch durch Software gelöscht werden. Es erlaubt dem Programm, direkt nach einem Reset zu überprüfen, welche Quelle letzlich den Reset ausgelöst hat.

- (MSB)	WDTS	IP0.5	IP0.4	IP0.3	IP0.2	IP0.1	IP0.0 (LSB)

Bit	Funktion
WDTS	Status-Flag des Watchdog-Timers. Es wird durch die Hardware gesetzt, wenn ein Reset durch den Watchdog-Timer ausgelöst wurde. Es kann durch Software gesetzt oder gelöscht werden.

Bild 13.3: Special Function Register IP0 (0A9H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

Bei dem üblicherweise verwendeten Oszillator mit 12 MHz dauert es etwa 65 ms bis der WDT seinen Überlaufwert von FFFCH erreicht hat, da er pro Maschinenzyklus einmal inkrementiert wird.

14 Oszillator und Taktversorgung

Die beiden Pins *XTAL1* und *XTAL2* sind der Eingang bzw. der Ausgang eines einstufigen, integrierten Inverters, der mit externen Bauteilen als Pierce-Oszllilator konfiguriert werden kann. Dieser Oszillator treibt den internen Taktgenerator, der die Frequenz für die interne Taktversorgung durch zwei teilt. Der interne Systemtakt des 80(C)515/-535 beträgt also $f_{OSZ}/2$. Damit werden die Phases 1 und 2, die States 1 bis 6 und die Maschinenzyklen festgelegt. Siehe Kapitel 3.

14.1 Quarz-Oszillator-Betrieb

Bild 14.1 zeigt ein Schaltungsbeispiel bei Verwendung eines Quarzes. In dieser Anwendung wird der integierte Inverter als quarzgesteuerter Oszillator mit einer positiven Reaktanz beschaltet.

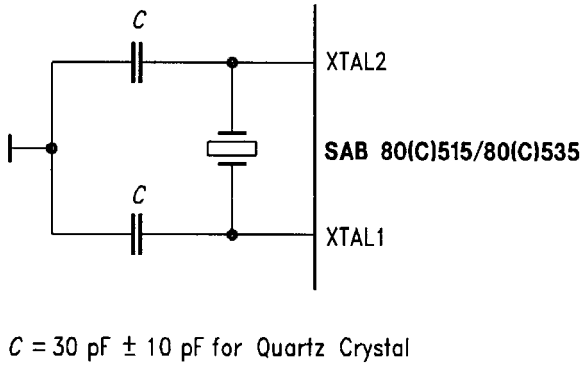


Bild 14.1: Empfohlene Oszillatorbeschaltung für den 80(C)515/-535

Kapazitäten zwischen 20 pF und 40 pF zusammen mit einem für Mikroprozessorsysteme empfohlenen Quarz haben sich bewährt. Unter Umständen kann auch in kostenkritischen Anwendungen ein keramischer Schwinger anstelle des Quarzes eingesetzt werden. Dann sollten die Werte der Kondensatoren C_1 und C_2 etwas höher (und unterschiedlich) gewählt werden. Genaue Empfehlungen gibt der Hersteller des Quarzes oder Schwingers.

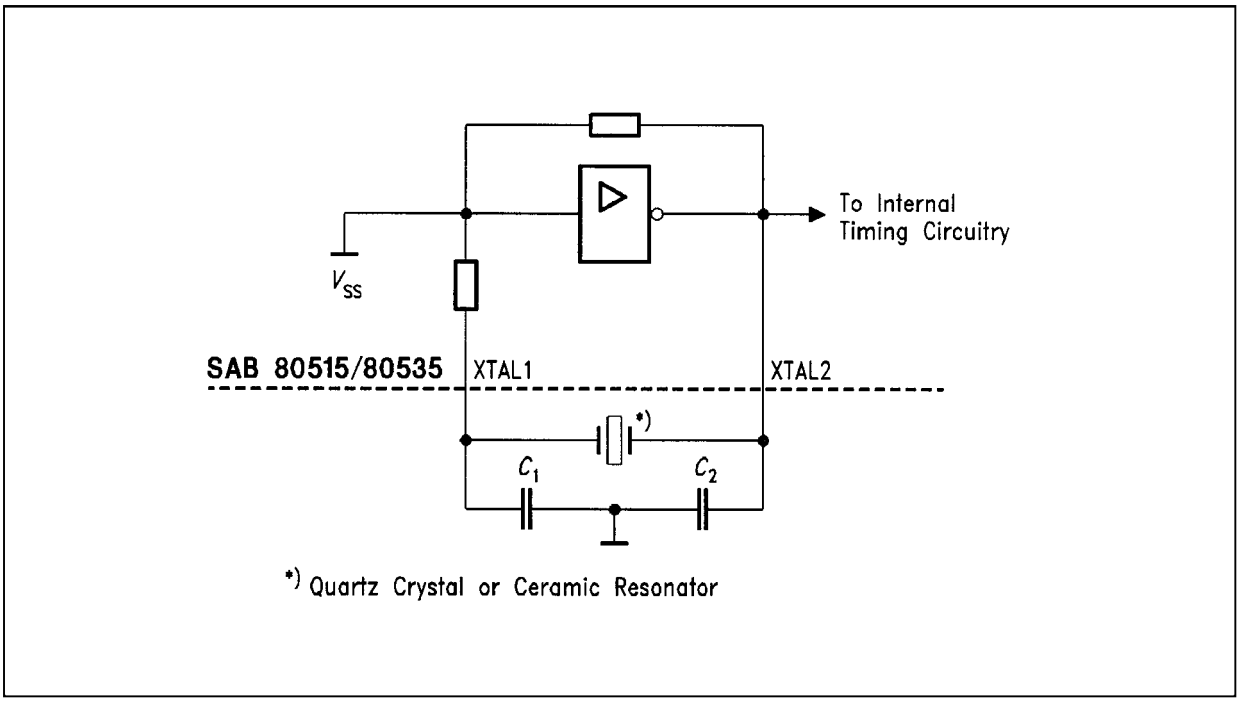


Bild 14.2: Detailliertes Schaltbild des On-Chip-Oszillators beim 80515/-535

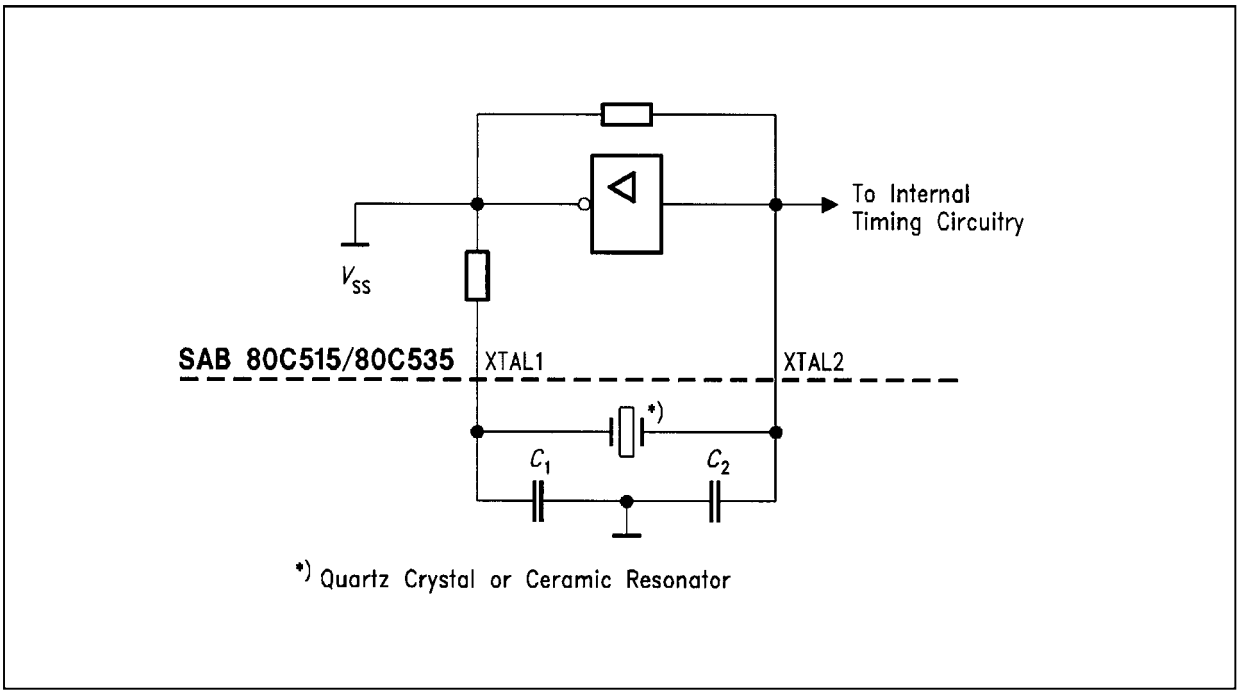


Bild 14.3:Detailliertes Schaltbild des On-Chip-Oszillators beim 80C515/-535

14.2 Externe Taktquellen

Soll der Takt extern eingspeist werden, wird der interne Oszillator nicht verwendet. In diesem Fall muß der Takt an *XTAL2* angelegt werden (Eingang des Oszillatorinverters und des Bausteintakts), während der Ausgang des Inverters *XTAL1* ...

- ... beim 80515/-535 auf Betriebsmasse gelegt werden muß
- ... beim 80C515/-535 offen bleibt.

Um die Rauschempfindlichkeit zu mindern, wird dabei ein externer Pullup-Widerstand an *XTAL2* empfohlen. Er ist aber nicht notwendig, wenn die Spannungspegel des Takts den Spezifikationen V_{IL} und V_{IH2} von *XTAL2* entsprechen (siehe Datenblatt).

15 Systemtakt-Ausgang

Für den Fall, daß externe Peripheriekomponenten einen zum Baustein synchronen Takt benötigen, stellt der 80(C)515/-535 ein Taktsignal am Pin *P1.6/CLKOUT* zur Verfügung, das aus dem internen Systemtakt abgeleitet wird. Wenn das Bit CLK im Special Function Register ADCON gesetzt ist, wird am Pin *P1.6/CLKOUT* ein Taktsignal mit 1/12 der Oszillatorfrequenz ausgegeben. Um diese Eigenschaft zu nutzen, muß das Latch des Pin auf 1 gesetzt sein (dies ist die Voreinstellung nach einem Reset).

BD	CLK	-	BSY	ADM	MX2	MX1	MX0
DFH (MSB)	DEH	DDH	DCH	DBH	DAH	D9H	D8H (LSB)

Bit	Funktion
CLK	Taktausgangsfreigabe (Clockout Enable). Wenn es gesetzt ist, wird am Pin *P1.6/CLKOUT* der Systemtakt (f_{OSZ}/12) ausgegeben.

Bild 15.1: Special Function Register ADCON (0D8H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

Der ausgegebene Takt liegt auf High-Pegel während S3P1 und S3P2 in jedem Maschinenzyklus, während der restlichen Zeit ist er auf Low-Pegel. Die Impulsbreite beträgt somit 1/6 der Taktperiode. Bei Ausführung eines Zugriffs auf den externen Datenspeicher (MOVX-Befehle) fallen das Ende des Taktpulses und des $\overline{RD}$- oder $\overline{WR}$-Signals im letzten State (S3) zusammen. Bild 15.2 gibt das Timingdiagramm wieder.

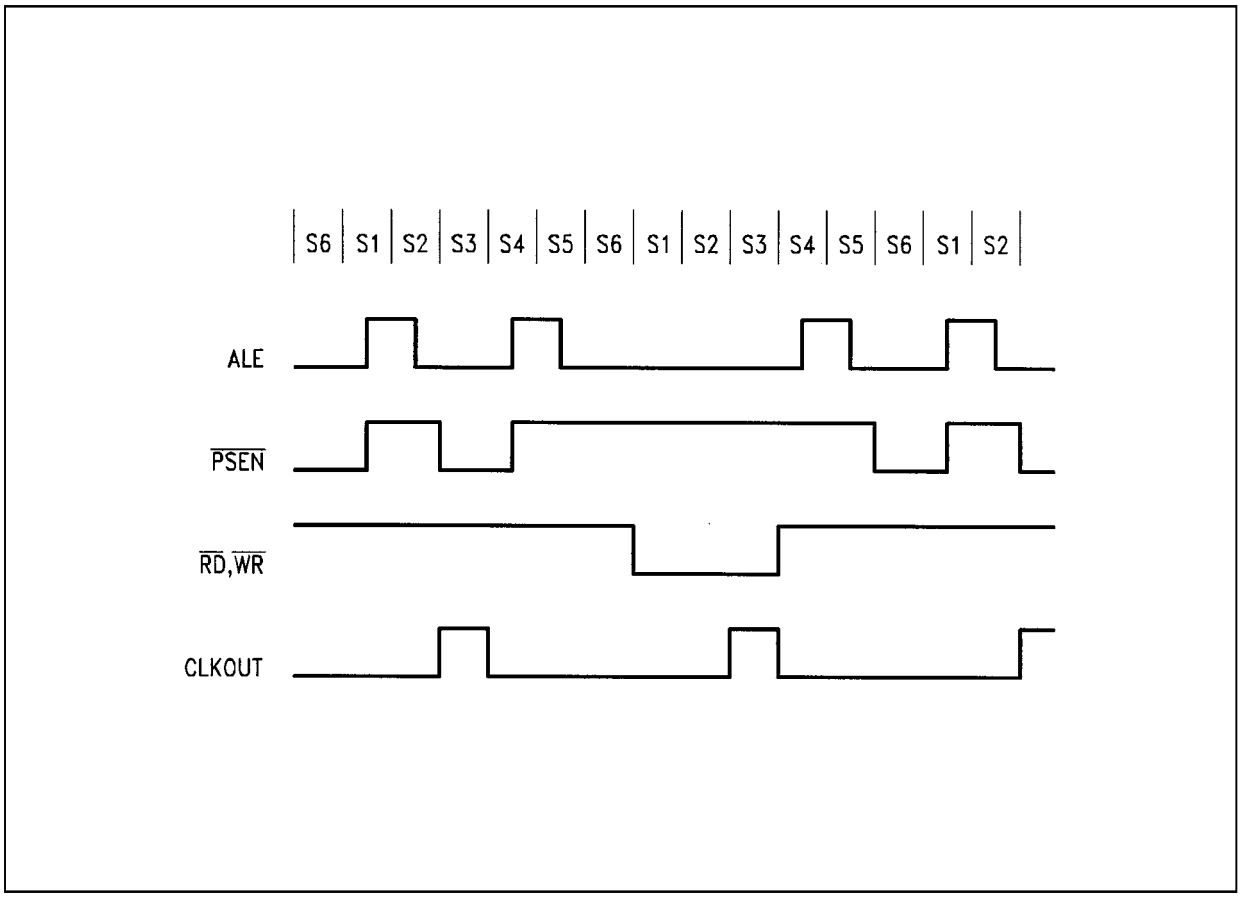


Bild 15.2: Timingdiagramm des Ausgangstakts

16 Interrupt-System

Der 80(C)515/-535 hat ein leistungsfähiges Interrupt-System, das insgesamt zwölf Interrupt-Requests aus den internen Peripheriekomponenten (fünf Interrupts) und von externen Quellen (sieben Interrupts) sehr selbständig bearbeiten kann. Dadurch wird die Belastung der CPU auf das wirklich notwendige Maß beschränkt, da nach einmaliger Prioritätsvergabe und Freigabe ein automatisches Verwalten und Schachteln der Anforderungen erfolgt.

16.1 Struktur und Prinzip des Interrupt-Systems

Interrupts werden in allen Mikrocontroller-Systemen verwendet, um sehr gezielt auf fast immer unvorhersehbare oder asynchrone Ereignisse reagieren zu können. Das gerade laufende Programm soll dabei ohne aufwendige Vorbereitungen unterbrochen und ein bestimmtes Programmstück, die Interrupt-Routine, angesprungen und abgearbeitet werden, und zwar unabhängig davon, was sonst gerade abläuft. Dazu müssen auch mehrere Interrupt-Requests, die aufeinanderfolgen oder parallel eintreffen, priorisiert und geschachtelt werden können. Im 80(C)515/-535 übernimmt eine gesonderte Komponente, der Interrupt-Controller, die Aufgabe, zyklisch die internen Peripheriekomponenten und die externen Interruptquellen nach Anmeldungen abzufragen. Diese Anmeldung geschieht nicht durch irgendwelche verborgenen Schaltungsteile, sondern transparent mit verschiedenen Special Function Registern und die dort befindlichen Interrupt-Request-Flags. Ein Beispiel ist das Flag TF0, das beim Überlauf von Timer 0 von der Hardware gesetzt wird. Manche Interrupts haben auch mehrere Anforderungs-Flags; sie haben dann mehrere Quellen. Der Interrupt wird durch eines dieser Flags angefordert und die Interrupt-Software muß entscheiden, welche Aktionen durchgeführt werden.

Normalerweise werden Interrupt-Flags von der Hardware (interne Komponenten oder externe Bausteine) gesetzt, wenn eine vorgebbare Bedingung vorliegt. Die Flags sind aber ganz normale Bits in den verschiedenen SFR. Das bedeutet, daß die Request-Flags auch direkt durch das Programm gesetzt oder gelöscht werden können. Damit kann ein Interrupt-Request, der noch nicht bedient wurde, durch Software gelöscht werden, indem das zugehörige Request-Flag zurückgesetzt wird. Auch für den Fall, daß Interrupts gar nicht benutzt werden sollen, kann man auf das Anforderungs-Flag zurückgreifen: Durch Abfragen in einer Warteschleife kann auf das Eintreffen einer Anforderung gewartet werden (Polling). Den zu erwartenden Interrupt kann man in diesem Fall unterdrücken, wenn das zugehörige Freigabe-Flag zurückgesetzt bleibt. Damit wird auch klar, was geschieht, wenn einer erneuter Interrupt-Request erzeugt wird, bevor der vorhergehende bedient wurde: das entsprechende Flag wird nochmals gesetzt, obwohl es schon 1 ist. Die Konsequenz ist, daß Interrupts derselben Quelle „verlorengehen“. Andererseits wartet ein gesetztes Interrupt-Flag solange, bis es irgendwann (oder auch nie) bedient wird.

Der Interrupt-Controller fragt nun ständig alle vorhandenen Request-Flags ab. Findet er eines oder mehrere davon gesetzt, so trifft er folgende Entscheidungen:

- Ist der entsprechende Interrupt überhaupt freigegeben? Wenn nein, wird das Flag nicht berücksichtigt.

- Welche Prioritätsstufe ist für den angeforderten Interrupt programmiert? Ist die Stufe höher als eine gerade schon bearbeitetes Interrupt-Programm? Nur wenn dies der Fall ist, d.h. es läuft nur das normale Programm oder ein niedriger priorisiertes Interrupt-Programm, wird der neu angeforderte Interrupt angenommen und bearbeitet. Läuft dagegen ein gleich oder höher priorisiertes Interrupt-Programm, wird der neue Interrupt noch nicht angenommen. Er muß warten, bis das laufende Interrupt-Programm fertig ist.

- Wurde der Request letztlich akzeptiert, so erfolgt die Verzweigung in das zugehörige Interrupt-Programm (Interrupt-Service-Routine, Interrupt-Routine). Bei einigen Interrupts werden bei diesem Sprung gleich die Request-Flags von der Hardware gelöscht, was andernfalls durch Software in der Interrupt-Routine erledigt werden muß. Bei Interrupts, die durch mehrere Request-Flags ausgelöst werden können, erfolgt dieses automatische Rücksetzen generell nicht, damit die Interrupt-Routine durch Abfragen der Flags entscheiden kann, welches das auslösende Flag war. Dann muß es durch Software zurückgesetzt werden, bevor die Interrup-Routine wieder verlassen wird.

Der Sprung in eine Interrupt-Routine sieht genauso aus, als ob ein normales Unterprogramm mit einem CALL-Befehl aufgerufen würde. Allerdings wird der CALL-Befehl bei der Interrupt-Routine vom Interrupt-Controller selbst erzeugt, im Programmcode taucht er nicht auf. Je nach Interrupt wird an eine andere Stelle gesprungen, an die „Vektoradresse“. Diese Adressen sind den einzelnen Interrupts fest zugeordnet und können nicht variiert werden. Der Programmierer kann an dieser Adresse dann einen Sprung an jede beliebige Stelle vorsehen. Zusätzlich zur Erzeugung des internen CALL-Befehls, speichert der Interrupt-Controller auch die Prioritätsebene des Interrupts. Wie oben beschrieben, wird diese Information benötigt, falls ein weiterer Interrupt gefordert wird, während aktuell ein anderer bedient wird.

Natürlich wird bei der Durchführung des selbsterzeugten CALL-Befehls die Rückkehradresse wie gewohnt auf dem Stack gespeichert. Dementsprechend endet das Interrupt-Programm auch mit einem Return-Befehl, allerdings in der Sonderform RETI (statt RET). Der Unterschied liegt darin, daß RETI die Prioritätssteuerung informiert, daß die gerade laufende Prioritätsebene verlassen wird. Ein normaler RET-Befehl macht dies nicht.

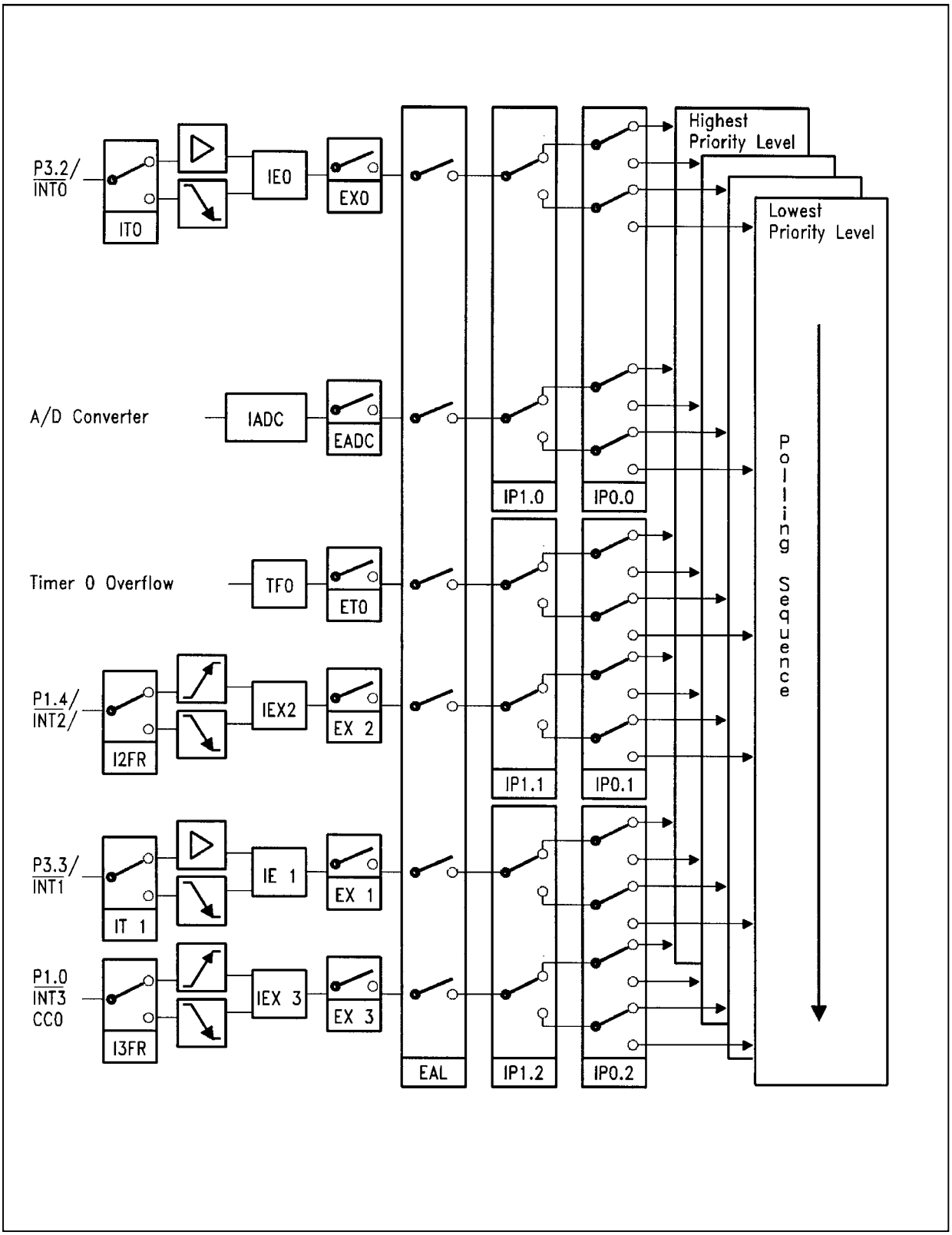


Bild 16.1a: Interruptstruktur des 80(C)515/-535

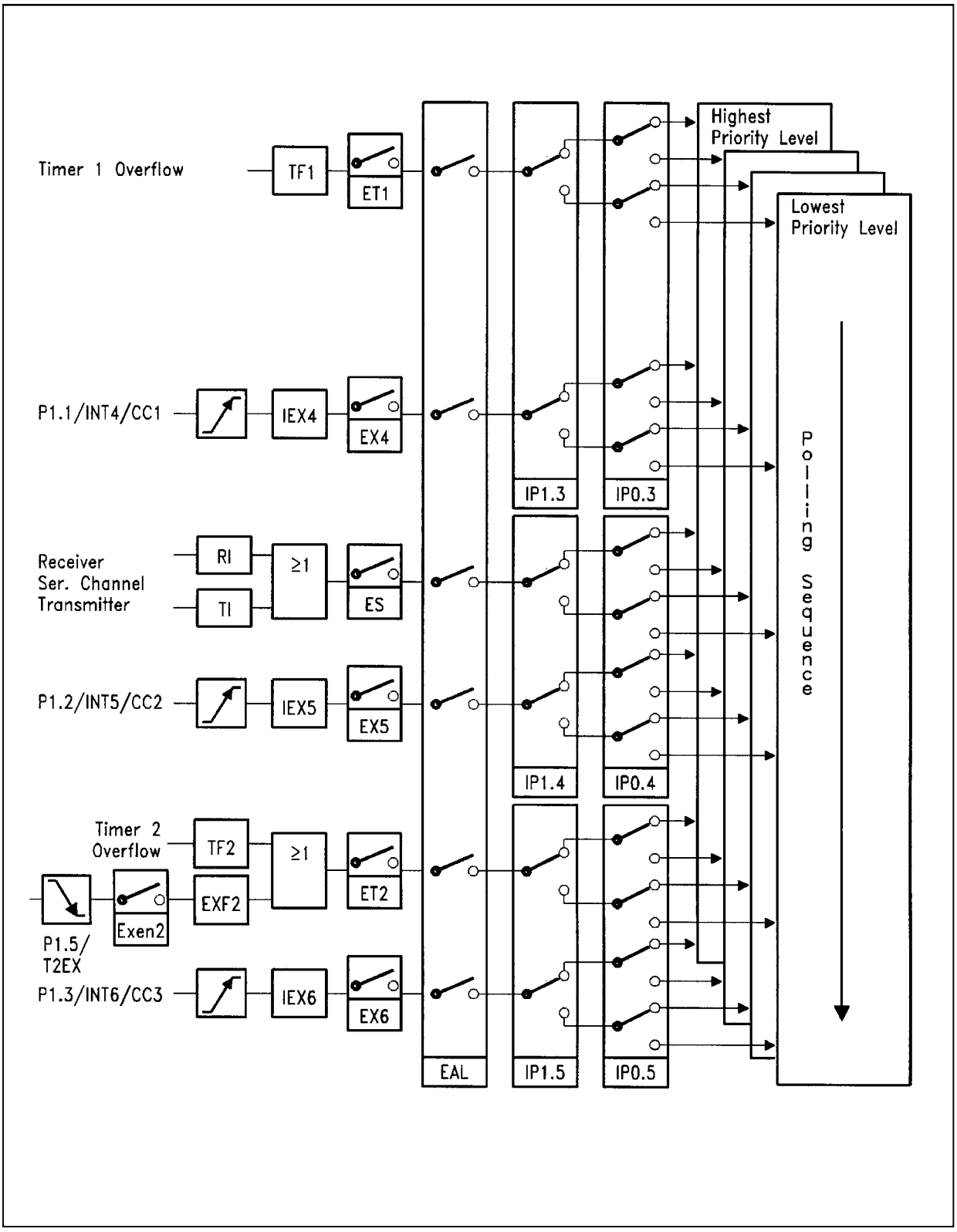


Bild 16.1b: Interruptstruktur des 80(C)515/-535 (Fortsetzung)

16.2 Special Function Register der Interrupts

Die Freigabe-Flags für alle Interrupts befinden sich in den Special Function Register IEN0 und IEN1. Dort befindet sich für jeden Interrupt ein eigenes Freigabe-Bit. Nur wenn dies gesetzt ist, wird der Interrupt-Controller auf die zugehörigen Anforderungs-Flags überhaupt reagieren. Das Setzen der Anforderungs-Flags selbst ist dagegen, wie oben dargestellt, von der Freigabe unabhängig. Zusätzlich gibt es ein globales Freigabe-Flag EAL (Bit IEN0.7), das alle Interrupts ausschaltet, wenn es zurückgesetzt ist, unabhängig davon, welche Einzel-Freigabebits gesetzt sind. Alle individuellen Freigabebits betreffen immer einen Interrupt(-Vektor), unabhängig, aus wie vielen Quellen (Interrupt-Flags) er angefordert wird. Es ist daher nicht möglich, mit Hilfe des Freigabebits bei Interrupts mit mehreren Request-Flags nur bei z.B. einem dieser Flags einen Interrupt zu erzeugen.

EAL	WDT	ET2	ES	ET1	EX1	ET0	EX0
AFH (MSB)	AEH	ADH	ACH	ABH	AAH	A9H	A8H (LSB)

Bit	Funktion
EAL	Generelles Freigabebit aller Interrupts. Bei EAL=0 wird kein angeforderter Interrupt bedient. Bei EAL=1 gilt das individuelle Freigabebit des jeweiligen Interrupts.
ET2	Freigabebit des Timers-2-Interrupt. Es gibt den Interrupt frei (ET2=1) oder sperrt ihn (ET2=0).
ES	Freigabebit des Interrupts für die serielle Schnittstelle. Es gibt den Interrupt frei (ES=1) oder sperrt ihn (ES=0).
ET1	Freigabebit des Timer-1-Interrupt. Es gibt den Interrupt frei (ET1=1) oder sperrt ihn (ET1=0).
EX1	Freigabebit des externen Interrupts 1. Es gibt den Interrupt frei (EX1=1) oder sperrt ihn (EX1=0).
ET0	Freigabebit des Timer-0-Interrupts. Es gibt den Interrupt frei (ET0=1) oder sperrt ihn (ET0=0).
EX0	Freigabebit des externen Interrupts 0. Es gibt den Interrupt frei (EX0=1) oder sperrt ihn (EX0=0).

Bild 16.2: Special Function Register IEN0 (0A8H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

EXEN2	SWDT	EX6	EX5	EX4	EX3	EX2	EADC
BFH (MSB)	BEH	BDH	BCH	BBH	BAH	B9H	B8H (LSB)

Bit	Funktion
EXEN2	Freigabebit des extern gesteuerten Nachlade-Interrupts des Timer 2 (External Reload of Timer2 Enable). Es gibt den Interrupt frei (EXEN2=1) oder sperrt ihn (EXEN2=0). Die Nachladefunktion selbst wird von EXEN2 nicht beeinflußt.
EX6	Freigabebit des externen Interrupt 6 bzw. Compare/Capture-Interrupt 3 am Pin *P1.3*. Es gibt den Interrupt frei (EX6=1) oder sperrt ihn (EX6=0). Die Compare/Capture-Funktion selbst wird von EX6 nicht beeinflußt
EX5	Freigabebit des externen Interrupt 5 bzw. Compare/Capture-Interrupt 2 am Pin *P1.2*. Es gibt den Interrupt frei (EX5=1) oder sperrt ihn (EX5=0). Die Compare/Capture-Funktion selbst wird von EX5 nicht beeinflußt
EX4	Freigabebit des externen Interrupt 4 bzw. Compare/Capture-Interrupt 1 am Pin *P1.1*. Es gibt den Interrupt frei (EX4=1) oder sperrt ihn (EX4=0). Die Compare/Capture-Funktion selbst wird von EX4 nicht beeinflußt
EX3	Freigabebit des externen Interrupt 3 bzw. Compare/Capture-Interrupt 0 am Pin *P1.0*. Es gibt den Interrupt frei (EX3=1) oder sperrt ihn (EX3=0). Die Compare/Capture-Funktion selbst wird von EX3 nicht beeinflußt
EX2	Freigabebit des externen Interrupt 2 bzw. Compare/Capture-Interrupt 4 am Pin *P1.4*. Es gibt den Interrupt frei (EX4=1) oder sperrt ihn (EX2=0). Die Compare/Capture-Funktion selbst wird von EX2 nicht beeinflußt
EADC	Freigabebit des A/D-Wandler-Interrupts. Es gibt den Interrupt frei (EADC=1) oder sperrt ihn (EADC=0)

Bild 16.3: Special Function Register IEN1 (0B8H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

Im folgenden werden die einzelnen Interrupt-Quellen anhand ihrer Request-Flags diskutiert. Die betroffenen Special Function Register sind TCON, T2CON und IRCON. Dabei ist zu beachten, daß in diesen Registern auch Status- und Steuerbits für andere Peripheriekomponenten enthalten sind, die hier nicht beschrieben werden.

Die externen Interrupts 0 und 1 haben die Interrupt-Request-Flags IE0 und IE1 im Register TCON. Bei diesen beiden Interrupts besteht die Wahlmöglichkeit, ob sie durch einen Low-Pegel oder eine fallende Flanke am zugehörigen Interrupt-Eingang (*P3.2*/$\overline{INT0}$ und P3.3/$\overline{INT1}$) ausgelöst werden. Diese Auswahl geschieht durch die zwei Steuerbits IT0 und IT1, die sich ebenfalls im SFR TCON befinden. Bei ITx=1 ist Flankentriggerung gewählt, sonst wird der Interrupt-

Request immer bei Low-Pegel aktiv. Wenn der Interrupt-Controller einen solchen Interrupt annimmt und in die entsprechende Service-Routine einspringt, wird gleichzeitig das Request-Flag zurückgesetzt. Dies aber nur dann, wenn Flankentriggerung gewählt ist. Wird dagegen mit Pegelaktivierung gearbeitet, ist der Zustand des Request-Flags nur vom am Interrupt-Eingang anliegenden Pegel abhängig: bei Low-Pegel ist das Flag IEx gesetzt, andernfalls ist es gelöscht.

Die Interrupts von Timer 0 und Timer 1 werden durch die Request-Flags TF0 und TF1 ausgelöst. Sie liegen ebenfalls im SFR TCON. Die Flags werden durch den Überlauf des betreffenden Timers gesetzt. Beim Sprung in die Interrupt-Routine werden TF0 und TF1 automatisch von der Hardware gelöscht.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
8FH (MSB)	8EH	8DH	8CH	8BH	8AH	89H	88H (LSB)

Bit	Funktion
TF1	Timer-1-Überlaufflag. Es wird von der Hardware bei einem Überlauf des Timer 1 gesetzt. Beim Einsprung in die zugehörige Interrupt-Adresse wird es automatisch gelöscht.
TF0	Timer-0-Überlaufflag. Es wird von der Hardware bei einem Überlauf des Timer 0 gesetzt. Beim Einsprung in die zugehörige Interrupt-Adresse wird es automatisch gelöscht.
IE1	Interrupt-Request-Flag des externen Interrupts 1 (*P3.3*/$\overline{INT1}$). Es wird gesetzt, wenn eine fallende Flanke am Pin *P3.3* entdeckt wird (Voraussetzung: mit IT1 wurde die fallende Flanke selektiert). Im Fall der Low-Pegel-Aktivierung bleibt IE1 gesetzt, solange der Pegel anliegt, und kann nicht durch Software gelöscht werden.
IT1	Selektionsbit für den Interrupt 1. Es legt fest, ob der Interrupt durch eine fallende Flanke (IT1=1) oder durch Low-Pegel (IT1=0) ausgelöst wird.
IE0	Interrupt-Request-Flag des externen Interrupts 0 (*P3.2*/$\overline{INT0}$). Es wird gesetzt, wenn eine fallende Flanke am Pin *P3.2* entdeckt wird (Voraussetzung: mit IT0 wurde die fallende Flanke selektiert). Im Fall der Low-Pegel-Aktivierung bleibt IE0 gesetzt, solange der Pegel anliegt, und kann nicht durch Software gelöscht werden.
IT0	Selektionsbit für den Interrupt 0. Es legt fest, ob der Interrupt durch eine fallende Flanke (IT0=1) oder durch Low-Pegel (IT0=0) ausgelöst wird.

Bild 16.4: Special Function Register TCON

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

Die serielle Schnittstelle des Bausteins hat zwei Request-Flags. Für den Empfang ist dies das Flag RI, für die Aussendung das Flag TI aus dem SFR SCON. Der Interrupt wird durch eine ODER-Verknüpfung von RI und TI (RI$\vee$TI) erzeugt, d.h. eines der Flags oder beide gleichzeitig

lösen den Interrupt aus. Beim Sprung in die Interrupt-Routine werden diese Flags nicht zurückgesetzt; dadurch kann die Software innerhalb der Routine abfragen, welches Flag der Auslöser war und dann reagieren. Das Rücksetzen muß durch Software erfolgen.

SM0	SM1	SM2	REN	TB8	RB8	TI	RI
9FH (MSB)	9EH	9DH	9CH	9BH	9AH	99H	98H (LSB)

Bit	Funktion
TI	Transmitter Interrupt (Sender-Interrupt). Dies ist das Interrupt-Request-Flag für den Sender. TI wird von der Hardware im Modus 0 am Ende des 8. Datenbits gesetzt, in den anderen Modi bei Beginn des Stoppbits. TI muß durch Software zurückgesetzt werden.
RI	Receiver Interrupt (Empfänger-Interrupt). Dies ist das Interrupt-Request-Flag für den Empfänger. RI wird von der Hardware im Modus 0 am Ende des 8. Datenbits gesetzt, in den anderen Modi während des Stoppbits. RI muß durch Software zurückgesetzt werden.

Bild 16.5: Special Function Register SCON (98H)

Der Interrupt des Timer 2 wird ebenfalls durch eine ODER-Verknüpfung zweier Request-Flags erzeugt: es handelt sich hier um das Überlaufflag TF2 und das Reload-Flag EXF2, das bei externem Reload gesetzt wird. Beide Flags befinden sich im SFR IRCON. Keines dieser Flags wird beim Einsprung in die Interrupt-Routine automatisch zurückgesetzt, dies muß durch Software erfolgen.

Der Interrupt des Analog/Digital-Wandlers wird durch das Flag IADC angefordert. Es befindet sich im SFR IRCON. IADC wird nicht durch die Hardware beim Einsprung in die Interrupt-Routine zurückgesetzt, dies muß durch Software geschehen.

T2PS	I3FR	I2FR	T2R1	T2R0	T2CM	T2I1	T2I0
CFH (MSB)	CEH	CDH	CCH	CBH	CAH	C9H	C8H (LSB)

Bit	Funktion
I3FR	Flankenwahl für den externen Interrupt 3 (steigende oder fallende Flanke). Es legt fest, ob das Request-Flag IEX3 gesetzt wird, wenn eine fallende (I3FR=0) oder eine steigende (I3FR=1) Flanke am Pin *P1.0*/$\overline{INT3}$ entdeckt wird.
I2FR	Flankenwahl für den externen Interrupt 2 (steigende oder fallende Flanke). Es legt fest, ob das Request-Flag IEX2 gesetzt wird, wenn eine fallende (I2FR=0) oder eine steigende (I2FR=1) Flanke am Pin *P1.4*/$\overline{INT2}$ entdeckt wird.

Bild 16.6 Special Function Register T2CON (0C8H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

Der externe Interrupt 2 benutzt das Bit IEX2 im SFR IRCON als Request-Flag. Es besteht die Wahlmöglichkeit, ob der Interrupt-Eingang *P1.4*/$\overline{INT2}$ auf positive oder negative Flanken reagiert. Dies wird durch das Steuerbit I2FR im Register T2CON eingestellt. IEX2 wird beim Sprung in die Interrupt-Routine automatisch zurückgesetzt.

Der externe Interrupt 3 verhält sich völlig analog zum Interrupt 2. Das Request-Flag ist IEX2 im SFR IRCON. Druch das Bit I3FR im SFR T2CON wird festgelegt, ob der Interrupt-Eingang *P1.0*/$\overline{INT3}$/*CC0* auf positive oder negative Flanken reagiert. Zusätzlich reagiert das Request-Flag auch auf Compare-Ereignisse im angeschlossenen Compare-Register CC0. Dies geschieht bei allen Compare-Modi und unabhängig davon, ob und welcher Flankenwechsel durch den Compare am Pin erzeugt wird. IEX3 wird beim Sprung in die Interrupt-Routine automatisch zurückgesetzt.

EXF2	TF2	IEX6	IEX5	IEX4	IEX3	IEX2	IADC
C7H (MSB)	C6H	C5H	C4H	C3H	C2H	C1H	C0H (LSB)

Bit	Funktion
EXF2	Interrupt-Request-Flag des externen Timer-2-Nachlademodus. Es wird bei einer fallenden Flanke am Pin P1.5/T2EX gesetzt, wenn der externe Reload-Modus für den Timer 2 gewählt ist. Ist der Nachlade-Interrupt des Timer 2 freigegeben (EXEN2=1), wird in die Interrupt-Routine verzweigt. EXF2 muß durch Software gelöscht werden.
TF2	Interrupt-Request-Flag des Überlaufs von Timer 2. Bei einem Überlauf des Timer 2 wird es von der Hardware gesetzt. Es muß durch Software gelöscht werden.
IEX6	Interrupt-Request-Flag des externen Interrupts 6. Es wird gesetzt, wenn eine steigende Flanke bzw. eine Compare-Flanke am Pin *P1.3/INT6/CC3* auftritt. IEX6 wird beim Einsprung in die Interrupt-Routine durch die Hardware gelöscht, es kann aber auch durch Software gelöscht werden.
IEX5	Interrupt-Request-Flag des externen Interrupts 5. Es wird gesetzt, wenn eine steigende Flanke bzw. eine Compare-Flanke am Pin *P1.2/INT5/CC2* auftritt. IEX5 wird beim Einsprung in die Interrupt-Routine durch die Hardware gelöscht, es kann aber auch durch Software gelöscht werden.
IEX4	Interrupt-Request-Flag des externen Interrupts 4. Es wird gesetzt, wenn eine steigende Flanke bzw. eine Compare-Flanke am Pin *P1.1/INT4/CC1* auftritt. IEX4 wird beim Einsprung in die Interrupt-Routine durch die Hardware gelöscht, es kann aber auch durch Software gelöscht werden.
IEX3	Interrupt-Request-Flag des externen Interrupts 3. Es wird gesetzt, wenn eine steigende/fallende (abhängig von I3FR im SFR T2CON) Flanke bzw. eine Compare-Flanke am Pin *P1.0/$\overline{INT3}$/CC0* auftritt. IEX3 wird beim Einsprung in die Interrupt-Routine durch die Hardware gelöscht, es kann aber auch durch Software gelöscht werden.
IEX2	Interrupt-Request-Flag des externen Interrupts 2. Es wird gesetzt, wenn eine steigende/fallende (abhängig von I2FR im SFR T2CON) Flanke bzw. eine Compare-Flanke am Pin *P1.4/$\overline{INT2}$/CC0* auftritt. IEX2 wird beim Einsprung in die Interrupt-Routine durch die Hardware gelöscht, es kann aber auch durch Software gelöscht werden.
IADC	Interrupt-Request-Flag des A/D-Wandlers. Es wird am Ende eine A/D-Wandlung von der Hardware gesetzt. Es muß durch Software gelöscht werden.

Bild 16.7: Special Function Register IRCON (0C0H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

Die externen Interrupts 4 (INT4), 5 (INT5) und 6 (INT6) benutzen die Interrupt-Flags IEX4, IEX5 und IEX6. Diese drei Interrupts reagieren immer auf eine positive Flanke an ihren Eingängen *P1.1/INT4/CC1*, *P1.2/INT5/CC2* und *P1.3/INT6/CC3*. Genauso wie bei den Interrupts 2 und 3 reagieren die IEX4, IEX5 und IEX6 auf Compare-Ereignisse in ihren angeschlossenen Compare-Registern CC1, CC2 und CC3; dies geschieht bei allen Compare-Modi und unabhängig davon, ob und welcher Flankenwechsel durch diesen Compare am Pin erzeugt wird. Die Interrupt-Request-Flags IEX4, IEX5 und IEX6 werden beim Sprung in die entsprechende Interrupt-Routine automatisch zurückgesetzt.

16.3 Prioritätssteuerung

Bereits am Anfang des Kapitels wurde das Prioritätensystem des Interrupt-Controllers kurz angesprochen. Wie erwähnt, ist es möglich, die Interrupts paarweise auf eine von vier Prioritätsebenen zu programmieren. Damit wird festgelegt, ob ein Interrupt einen anderen bereits laufenden Interrupt unterbrechen darf. Dies erfolgt dann, wenn der neue Interrupt eine höhere Prioritätsstufe als der gerade laufende hat. Ist die Prioritätsstufe gleich oder gar niedriger, so erfolgt keine Unterbrechung; in diesem Fall muß der neue Interrupt warten, bis der laufende fertig bedient ist (d.h. der Befehl RETI wurde ausgeführt).

Die vorhandenen vier Stufen erlauben somit, daß sich bis zu vier Interrupt-Routinen schachteln können. Dies geschieht ganz automatisch und der Interrupt-Controller stellt sicher, daß diese Schachtelung auch wieder korrekt aufgelöst wird. Programmtechnisch läuft dabei nichts anderes als eine mehrfache Verschachtelung von Unterprogrammen ab. Deshalb ist zu beachten, daß entsprechend der Schachteltiefe Platz auf dem Stack benötigt wird.

Die Interrupts können nur paarweise, also nicht beliebig, auf die vier Prioritätsstufen verteilt werden. Es besteht eine feste Zusammenfassung von je zwei Interrupts zu einer Gruppe. Diese Gruppen können dann auf eine der vier Stufen gelegt werden. In der folgenden Abbildung sind die Gruppen dargestellt. Jede Zeile bildet eine Gruppe.

Externer Interrupt 0 (IE0)	A/D-Wandler-Interrupt (IADC)
Timer-0-Interrupt (TF0)	Externer Interrupt 2 (EX2)
Externer Interrupt (IE1)	Externer Interrupt 3 (EX3)
Timer-1-Interrupt (TF1)	Externer Interrupt 4 (EX4)
Interrupt der seriellen Schnittstelle (RI, TI)	Externer Interrupt 5 (EX5)
Timer-2-Interrupt (TF2, EXF2)	Externer Interrupt 6 (EX6)

Tabelle 16.1: Prioritätsgruppen der Interrupts mit den Interrupt-Flags

Die Festlegung der Prioritäten für die einzelnen Gruppen erfolgt durch Steuerbits in den Special Function Registern IP0 und IP1. Je ein Bit aus IP0 und IP1 an derselben Bitposition dient paarweise zur Einstellung der vier Stufen.

-	WDTS	IP0.5	IP0.4	IP0.3	IP0.2	IP0.1	IP0.0
- (MSB)	-	IP1.5	IP1.4	IP1.3	IP1.2	IP1.1	IP1.0 (LSB)

Bit	Funktion
IP1.x IP0.x	Prioritätsprogrammierung der Interruptgruppen
0 0	Prioritätsstufe 0 - niedrigste Priorität
0 1	Prioritätsstufe 1
1 0	Prioritätsstufe 2
1 1	Prioritätsstufe 3 - höchste Priorität
Zuordnung:	Interruptgruppe:
IP1.0/IP0.0	Externer Interrupt 0 / A/D-Wandler
IP1.1/IP0.1	Timer 0 / Externer Interrupt 2
IP1.2/IP0.2	Externer Interrupt 1 / Externer Interrupt 3
IP1.3/IP0.3	Timer 1 / Externer Interrupt 4
IP1.4/IP0.4	Serielle Schnittstelle / Externer Interrupt 5
IP1.5/IP0.5	Timer 2 / Externer Interrupt 6

Bild 16.8: Special Function Register IP0 (0A9H) und IP1 (0B9H)

Dieses Bit/Diese Bits wird/werden für die aktuelle Einstellung nicht gebraucht.

Über die Prioritätssteuerung hinaus, die festlegt, wer wen unterbrechen darf, muß auch festgelegt werden, was passiert, wenn die Interrupt-Request-Flags von zwei oder mehr Interrupts auf derselben Prioritätsebene gleichzeitig anliegen. Dann muß ja einer zuerst bedient werden, während die anderen bis zu dessen Ende warten müssen. Diese „interne Rangfolge“ zeigt Tabelle 16.2. Die Interrupts der ersten Zeile werden zuerst bedient, dann die in der zweiten Zeile usw. Innerhalb der Zeilen wird von links nach rechts bedient. Es sei nochmals betont, daß dies keine Unterbrechung von bereits laufenden Interrupts beschreibt, sondern lediglich angibt, wer beim gleichzeitigen Anfordern auf derselben Prioritätsstufe zuerst zum Zuge kommt.

Hoch	Niedrig	Priorität
IE0	IADC	Hoch
TF0	IEX2	
IE1	IEX3	
TF1	IEX4	
RI, TI	IEX5	
TF2, EXF2	IEX6	Niedrig

Tabelle 16.2: Rangfolge innerhalb einer Prioritätsstufe

16.4 Bedienung der Interrupts

16.4.1 Timing der Interrupts

In diesem Abschnitt wird das Timing der Interruptbehandlung näher diskutiert. Wie nicht anders zu erwarten, sind auch diese Vorgänge fest an das bekannte Taktschema des Bausteins gebunden. Das Abtasten aller Interrupt-Request-Flags erfolgt zum Zeitpunkt S5P2 in jedem Maschinenzyklus. Das Auswerten der abgetasteten Werte erfolgt dann durch den Interrupt-Controller im darauffolgenden Maschinenzyklus. Wurde eines oder mehrere Flags aktiv angetroffen, so erkennt dies die Interrupt-Logik und erzeugt dann und nur dann einen internen LCALL-Befehl, wenn nicht eine der folgenden Bedingungen den Interrupt blockiert:

- Es läuft bereits ein Interrupt mit gleicher oder höherer Priorität
- Der gerade laufende Maschinenzyklus ist nicht der letzte des Befehls, der gerade abgearbeitet wird
- Der Befehl, der gerade abläuft, ist RETI oder ein Schreibzugriff auf eines der Register IEN0, IEN1, IEN2 oder IP0 bzw. IP1.

Sobald auch nur eine dieser Bedingungen vorliegt, wird der LCALL-Befehl noch nicht erzeugt. Die erste Bedingung stellt dabei das oben beschriebene Verhalten der Prioritätssteuerung sicher. Die zweite Bedingung sorgt dafür, daß ein Interrupt nicht einen gerade laufenden Befehl unterbricht, dies ist nur zwischen komplett abgeschlossenen Befehlen möglich. Die dritte Bedingung schließlich garantiert, daß eventuelle Änderungen am Interrupt-Status (z.B. durch neue Steuerbits in den erwähnten Registern) erst wirksam werden können, bevor die Bewertung des Interrupt-Request erfolgt. Der Befehl RETI beendet eine Interrupt-Routine. Deshalb bleibt auch er ununterbrochen, um unnötige Schachteltiefen zu vermeiden.

Dieser Entscheidungsvorgang findet in jedem Maschinenzyklus statt und zwar immer auf der Basis der im vorherigen Zyklus abgetasteten Werte der Request-Flags. Dadurch wird z.B. ein Interrupt-Request, der aus irgendwelchen Gründen nicht bedient worden ist und durch Software gelöscht wurde, nicht mehr berücksichtigt. Eine Speicherung der einstmals anliegenden Anforderung erfolgt somit nicht; es interessiert generell nur der aktuelle Zustand der Request-Flags.

Im Bild 16.9 ist der zeitliche Ablauf dargestellt. Ein Sonderfall ist, wenn während des mit C3 bezeichneten Zyklus ein höher priorisierter Interrupt akzeptiert wurde (ausgelöst durch ein Request-Flag, das zu S5P2 im Zyklus C2 gesetzt war). In diesem Spezialfall wird sofort im Anschluß an den laufenden LCALL-Befehl ein weiterer LCALL zu der neuen Sprungadresse erzeugt, ohne daß ein Befehl von der ersten Interrupt-Routine abgearbeitet wurde.

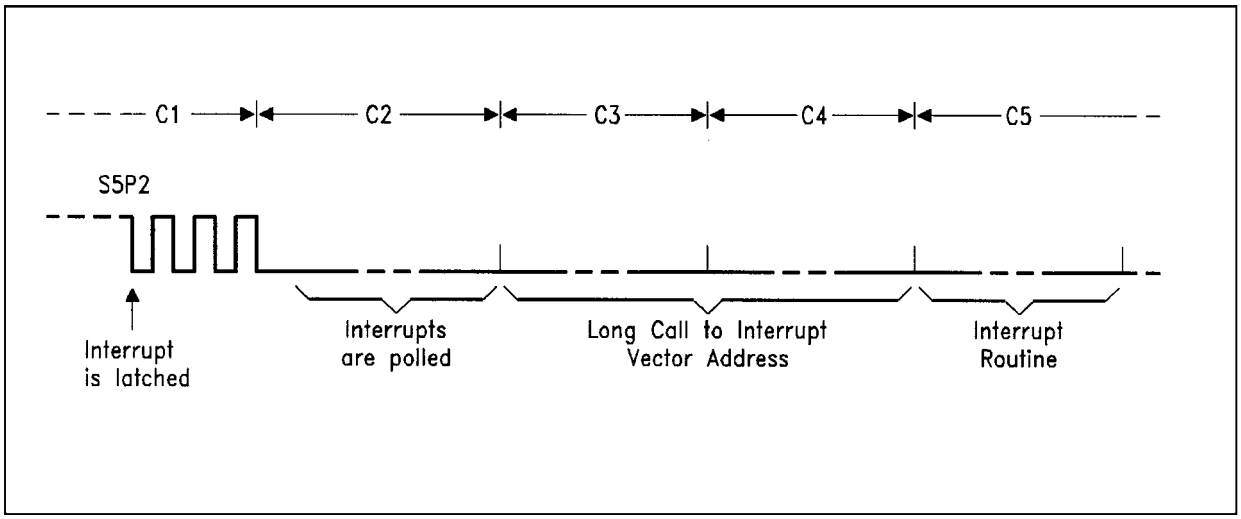


Bild 16.9: Interrupt-Antwortverhalten

16.4.2 Interrupt-Vektoren

Man erkennt im Bild 16.9, daß der LCALL-Befehl wie auch sonst zwei Maschinenzyklen Ausführungszeit benötigt. Er speichert dabei die Rücksprungadresse auf dem Stack. Bei bestimmten Interrupts werden während des LCALL auch die auslösenden Request-Flags gelöscht. Außerdem wird die Prioritäts-Logik auf die programmierte Ebene eingestellt, damit andere Interrupt-Requests korrekt bewertet werden können. Das Sprungziel des LCALL-Befehls, die Vektoradresse, ist für jeden Interrupt unterschiedlich. Die Programmausführung in der Interrupt-Routine beginnt an den in der folgenden Tabelle angegebenen Adressen. Am Ende der Routine steht dann der RETI-Befehl, der alle Funktionen des normalen RET-Befehls hat, wie z.B. Rücksprung auf die vom Stack geholte Rückkehradresse. Zusätzlich informiert der RETI-Befehl die Prioritäts-Logik, daß die laufende Prioritätsebene jetzt verlassen wird. Aus diesem Grund wird im Fall, daß die Interrupt-Routine mit dem einfachen RET verlassen wird, kein weiterer Interrupt derselben oder niedrigerer Priorität mehr angenommen.

Interrupt-Quelle	Request-Flags	Vektoradresse
Externer Interrupt 0	IE0	0003H
Timer-0-Interrupt (Overflow)	TF0	000BH
Externer Interrupt 1	IE1	0013H
Timer-1-Interrupt (Overflow)	TF1	001BH
Interrupt der seriellen Schnittstelle	RI, TI	0023H
Timer-2-Interrupt (Overflow/ext. Reload)	TF2/EXF2	002BH
A/D-Wandler	IADC	0043H
Externer Interrupt 2	IEX2	004BH
Externer Interrupt 3	IEX3	0053H
Externer Interrupt 4	IEX4	005BH
Externer Interrupt 5	IEX5	0063H
Externer Interrupt 6	IEX6	006BH

Tabelle 16.3: Interrupt-Vektoren

16.5. Externe Interrupts

Dieser Abschnitt geht auf die Besonderheiten bei der Benutzung der externen Interrupts ein und diskutiert auch das Antwort-Zeitverhalten. Die externen Interrupts 0 und 1 können entweder auf eine fallende Flanke oder auf Low-Pegel am externen Interrupt-Eingang reagieren (*P3.2*/$\overline{INT0}$ bzw. *P3.3*/$\overline{INT1}$). Dies wird eingestellt durch die Steuerbits IT0 und IT1 im SFR TCON. Ist Bit ITx auf 0 zurückgesetzt, so ist das entsprechende Interrupt-Request-Flag IEx immer dann aktiv, wenn am Eingang Low-Pegel liegt. Das Request-Flag wird damit direkt durch den Pegelzustand gesteuert, d.h. bei High-Pegel ist IEx=0, bei Low-Pegel ist IEx=1. Die Request-Flags können in diesem Modus auch durch den Sprung in die Interrupt-Routine nicht zurückgesetzt werden, der Low-Pegel am Pin verhindert das. In der Anwendung bedeutet das, daß die externe Anforderung erstens solange anstehen muß, bis sie bedient wird, und zweitens, daß sie dann während der Interrupt-Routine extern zurückgenommen werden muß, da sonst der Interrupt gleich wieder gefordert werden würde.

Wird durch ITx=1 die Flankenaktivierung eingeschaltet, so verhalten sich die Request-Flags anders (nämlich so, wie es alle anderen Request-Flags tun). Wird in diesem Modus eine fallende Flanke am Interrupt-Eingang erkannt, so wird durch die Hardware das entsprechende Request-Flag IE0 bzw. IE1 gesetzt. Dies führt dann zur Auslösung des Interrupts, wenn die restlichen Bedingungen erfüllt sind. Erfolgt der Einsprung in die Interrupt-Routine (durch einen internen LCALL-Befehl), wird IEx automatisch zurückgesetzt. Außerdem ist es in dieser Betriebsart auch möglich, die Request-Flags direkt durch die Software zu setzen oder zu löschen. Damit kann dann z.B. ein externer Interrupt angefordert werden (durch direktes Setzen des Request-Flags), ohne daß am Pin überhaupt etwas passiert ist. Andererseits ist es durch Löschen des gesetzten, aber noch nicht bedienten Request-Flags möglich, die Anforderung wieder verschwinden zu lassen.

Die Erkennung der Signalflanken erfolgt nach demselben Prinzip, wie der Baustein auch sonst Flanken an den Ports erkennt: es wird nicht die eigentliche Flanke detektiert, sondern nur regelmäßig einmal pro Maschinenzyklus abgetastet; haben nun zwei aufeinanderfolgende

Abtastungen ein unterschiedliches Ergebnis, so erkennt die Logik daran, daß ein Pegelwechsel, also eine Flanke, dagewesen sein muß. An den Pins $\overline{INT0}$ und $\overline{INT1}$ wird immer zum Zeitpunkt S5P2 abgetastet; daher muß auch jeder Pegel immer für die Dauer von mindestens einem Maschinenzyklus anliegen, damit der Controller ihn auch sicher erkennt. Somit wird dann zum Zeitpunkt S5P2 in dem Zyklus, wo zum erstenmal Low-Pegel erkannt wird, das Request-Flag gesetzt.

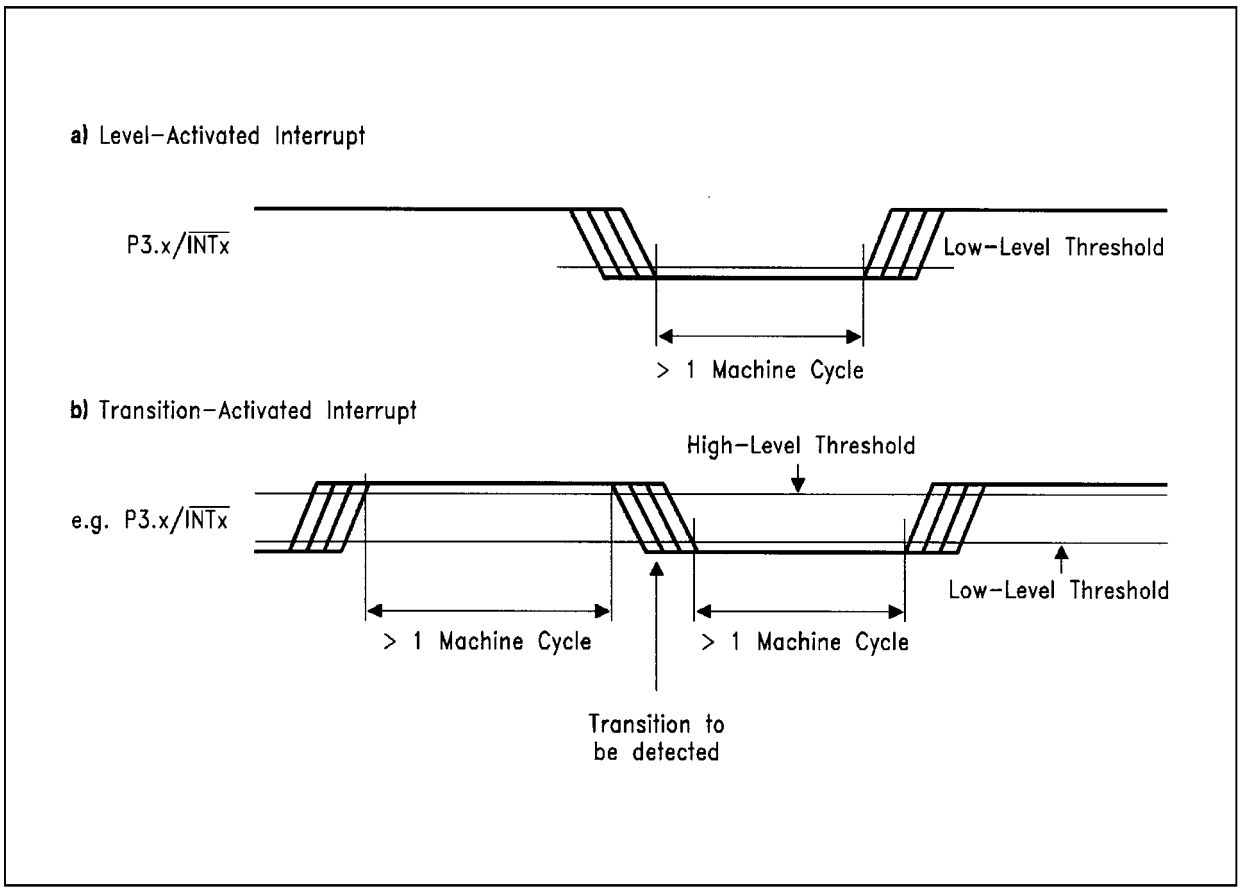


Bild 16.10: Erkennung externer Interrupts

Die externen Interrupts 2 und 3 mit ihren Request-Flags IEX2 und IEX3 bedienen die Eingänge *P1.4*/$\overline{INT2}$ und *P1.0*/$\overline{INT3}$/CC0. Diese beiden Interrupts können wahlweise durch die fallende oder steigende Signalflanke am Eingang aktiviert werden, was dann die Request-Flags setzt. Die Steuerbits I2FR und I3FR im SFR T2CON legen die Art der Flankenerkennung fest: I2FR/I3FR=0 heißt fallende Flanke, bei I2FR/I3FR=1 wird auf die ansteigende Flanke reagiert. Die Request-Flags verhalten sich sonst wie gewohnt, d.h. sie können auch durch Software manipuliert werden. Beim Sprung in die Interrupt-Routine werden sie automatisch zurückgesetzt.

Ganz analog dazu verhalten sich die Interrupts 4, 5 und 6. Die Request-Flags IEX4, IEX5 und IEX6 reagieren auf Ereignisse an den externen Eingängen *P1.1/INT4/CC1*, *P1.2/INT5/CC2* und *P1.3/INT6/CC3*. Allerdings gibt es bei diesen drei Interrupts keine Wahlmöglichkeit bei der Aktivierung zwischen fallender und steigender Flanke, vielmehr werden sie immer durch

steigende Signalflanken ausgelöst. Die Request-Flags werden beim Einsprung in die Interrupt-Routine zurückgesetzt.

Zum Abschluß des Kapitels soll noch eine Betrachtung zur Antwortzeit der externen Interrupts erfolgen: das ist die Zeit, die zwischen dem Signal am Interrupt-Eingang und der Ausführung des ersten Befehls der Interrupt-Routine vergeht. Die Abtastung des Pins und das Erkennen der Aktivierung des Signals erfolgt, wie bereits beschrieben, zum Zeitpunkt S5P2 im jedem Maschinenzyklus; zu diesem Zeitpunkt wird dann auch das zugehörige Request-Flag gesetzt. Die Auswertung der Zustände dieser Request-Flags führt der Interrupt-Controller während des folgenden Zyklus durch. Sind alle Bedingungen für den Interrupt schon erfüllt, so folgt bereits in den nächsten zwei Maschinenzyklen der selbsterzeugte LCALL-Befehl. Anschließend erfolgt dann der erste Befehl der Interrupt-Routine selbst. Somit ergibt sich in diesem Fall eine Antwortzeit von etwas mehr als drei Maschinenzyklen; dies ist die kürztmögliche Zeit, da alle Bedingungen erfüllt waren.

Die Antwortzeit verlängert sich, wenn eine der drei Bedingungen aus Abschnitt 16.4.1 nicht erfüllt war. Im Falle, daß noch ein anderer Interrupt gleicher oder höherer Priorität läuft, liegt es in der Natur dieses Interrupts, wie lange er noch zur kompletten Abarbeitung benötigt. Erst im Anschluß daran kommt ja der niedriger priorisierte Interrupt zum Zuge, und auch nur dann, wenn nicht noch höher priorisierte Interrupts ebenfalls warten. Die zweite Bedingung fordert, daß der laufende Befehl sich im letzten Zyklus seiner Ausführung befindet. Aus dieser Bedingung kann sich eine zusätzliche Verzögerung von maximal drei Zyklen ergeben. Die letzte Bedingung fordert, daß der Befehl, der gerade abläuft, keinen Schreibzugriff auf eines der Interrupt-Steuerregister durchführt oder ein RETI-Befehl ist. Sonst wird der nächste Befehl ausgeführt, bevor der Interrupt bedient wird. Daher kann durch diese Bedingung eine Verzögerung entstehen, die bis zu fünf Zyklen sein kann. Ein Zyklus für das Fertigstellen der gesperrten Befehle (die alle nur maximal zwei Zyklen haben), und bis zu vier Zyklen für den noch auszuführenden nachfolgenden Befehl. Somit kann in einem System, wo nur ein Interrupt benutzt wird, oder der fragliche Interrupt allein auf die höchste Prioritätsstufe programmiert ist, die Antwortzeit zwischen etwas mehr als drei und weniger als neun Maschinenzyklen betragen.

17 Der Befehlssatz

Der Befehlssatz des 80(C)515/-535 beinhaltet 111 Befehle, von denen 49 aus einem Byte, 45 aus zwei Bytes und 17 aus drei Bytes bestehen. Das allgemeine Befehlsformat setzt sich immer zusammen aus einer sog. Mnemonic-Anweisung, eventuell gefolgt von einem oder mehreren Operantenfeldern.

Wie alle Mitglieder der 8051-Familie benutzt auch der 80(C)515/-535 denselben Befehlssatz wie der Urvater. Deswegen ist der 80(C)515/-535 vollkommen softwarekompatibel zum 8051 und kann mit dem Standard-Assembler oder einer Hochsprache (z.B. C) programmiert werden. Vorhandene Bibliotheken können weiterhin verwendet werden.

17.1. Adressierung

Der 80(C)515/-535 verfügt über fünf verschiedene Adressierungsarten, die nachfolgend beschrieben sind:

- Registeradressierung (register)
- Direkte Adressierung (direct)
- Unmittelbare Adressieung (immediate)
- Indirekte Registeradressierung (register indirect)
- Indirekte indizierte Registeradressierung (base register plus index-register indirect)

Adressierungart	Erreichbare Speicherbereiche
Registeradressierung	Register R0 bis R7 der aktiven Registerbank, Akku, B,CY (Bit), DPTR
Direkte Adressierung	Untere 128 Byte des internen RAM, SFR
Unmittelbare Adressierung	Programmspeicher
Indirekte Registeradressierung	Gesamter interner RAM (@R0, @R1, SP), externer Datenspeicher (@R0, @R1, @DPTR)
Indirekte indizierte Registeradressierung	Programmspeicher (@DPTR+A, @PC+A)

Tabelle 17.1: Adressierungsarten und Speicherbereiche

17.1.1 Registeradressierung (Register Addressing)

Bei Registeradressierung wird auf die acht Arbeitsregister (R0 bis R7) der gerade selektierten Registerbank zugegriffen. Die niederwertigsten drei Bit des Opcodes kennzeichnen, welches Register verwendet wird. Registeradressierung hat zwei Vorteile: Zum einen ist ein Operand, nämlich das Register, bereits im Opcode enthalten (Einsparen eines Bytes, gilt für R0 bis R7).

Zum anderen beträgt die Ausführungszeit eines Registeradressierungsbefehls nur einen Maschinenzyklus. A, B, DPTR und das Flag C werden in manchen Befehlen ebenfalls wie Register adressiert, da sie in vielen Opcodes implizit enthalten sind.

17.1.2 Direkte Adressierung (Direct Addressing)

Bei der direkten Adressierung steht der zu verarbeitende Wert in der im Befehlscode angesprochenen Datenzelle. Diese Adressierungsart ist die einzige, mit der die Special Function Register (SFR) erreicht werden können. Auch die unteren 128 Byte des internen RAM lassen sich direkt adressieren.

17.1.3 Unmittelbare Adressierung (Immediate Addressing)

Dabei steht der Wert, der übertragen oder mit dem eine Berechnung durchgeführt werden soll, nicht in einer Datenzelle, sondern unmittelbar als Operand im Befehlscode. Da der Befehlscode (normalerweise) in einem dynamisch (während des Programmlaufs) nicht veränderbaren Speicher (ROM) steht, handelt es sich bei diesem Wert um eine Konstante.

17.1.4 Indirekte Registeradressierung (Register Indirect Addressing)

Allgemein bedeutet indirekte Adressierung, daß die im Befehl angesprochene Datenzelle die Adresse einer anderen Datenzelle enthält, in der letztlich der zu verarbeitende Wert liegt. Speziell beim 80(C)515/-535 heißt indirekte Registeradressierung, daß eines der beiden Register R0 oder R1 (der gewählten Registerbank) oder der 16 Bit breite Data-Pointer als Zeiger auf eine Datenzelle arbeitet, in der der gewünschte Wert steht. Dabei ist diese Speicherzelle entweder im 256 Byte großen internen RAM (MOV-Befehl) oder im externen Datenspeicher (MOVX-Befehl). Es muß darauf geachtet werden, daß die oberen 128 Byte des internen RAM nur durch indirekte Adressierung zu erreichen sind, wohingegen die SFR, die ja im gleichen Adressenbereich liegen, nur direkt adressierbar sind. Im übrigen werden die Stack-Operationen PUSH und POP auch mit indirekter Adressierung (über den Stack-Pointer) durchgeführt.

17.1.5 Indirekte indizierte Registeradressierung (Base Register plus Index-Register Addressing)

Dies ist eine Abwandlung der indirekten Registeradressierung. Auf den Inhalt eines Basisregisters (DPTR oder PC) wird der Inhalt eines anderen Registers (beim 80(C)515/-535 ist es der Akku) aufaddiert. Das Ergebnis ist die Zieladresse.

17.2 Flags

Die Flags verhalten sich bei arithmetischen Operationen wie folgt (Sonderfälle sind in der Befehlsbeschreibung vermerkt):

- CY (Carry, Übertrag): wird bei einem Übertrag von der höchstwertigen Stelle des Ergebnisses einer Operation (z.B. bei Addition) gesetzt. Das gleiche gilt, wenn z.B. bei einer Subtraktion das Ergebnis kleiner als Null wird. Auf diese Weise ist es möglich, Addition und Subtraktion mit Operanden beliebiger Länge durchzuführen, indem das Carry-Flag als Übertrag benutzt wird. Tritt kein Übertrag auf, wird CY gelöscht.

- AC (Auxiliaury Carry, Hilfsübertrag): verhält sich wie ein Carry für das untere Nibble (die unteren vier Bit) des Ergebnisses. Das AC ist wichtig bei der BCD-Arithmetik.

- OV (Overflow, Überlauf): wird dann gesetzt, wenn ein Übertrag in das höchstwertige Bit, aber nicht vom höchstwertigen Bit ins CY stattfindet oder umgekehrt. Ansonsten wird OV gelöscht. Damit ist Arithmrtik im Zweierkomplement möglich, weil OV=1 ist, sobald das Ergebnis nicht mehr in acht Bit einschließlich Vorzeichen dargestellt werden kann.

- P (Parity, Parität): zeigt die Summe der einzelnen Bits im Akku modulo 2 (ungerade Parität). Mit anderen Worten ist P=1, wenn eine ungerade Anzahl von Bits im Akku 1 ist. Ansonsten wird P gelöscht. Bei einer Schreiboperation nach PSW wird das Paritäts-Flag nicht beeinflußt, da es nur vom Zustand des Akkus abhängt. P kann also nicht direkt durch Software verändert werden.

17.3 Multiplikation und Division

Der Befehl MUL multipliziert die beiden in den Register A und B enthaltenen vorzeichenlosen Zahlen und liefert als Ergebnis eine 16-Bit-Zahl. Deren höchstwertiger Anteil steht im Register B, der niederwertige Anteil in A. Ist der Anteil im Register B Null, wird OV nicht gesetzt. OV wird gesetzt, wenn das Ergebnis größer als 255 war. CY ist immer gelöscht und AC bleibt unbeeinflußt.

DIV dividiert eine im Register A enthaltene vorzeichenlose 8-Bit-Zahl durch die vorzeichenlose 8-Bit-Zahl im Register B. Als Ergebnis steht der Quotient in A und der Divisonsrest in B. Die Flags CY und OV werden gelöscht. Bei einer Division durch Null sind die Inhalte von A und B undefiniert und OV wird gesetzt.

17.4 Bitverarbeitung

Die CPU der gesamten 8051-Familie enthält einen Bool'schen Prozessor zur Bitverarbeitung. Er hat einen eigenen Befehlssatz, der im Gesamtbefehlssatz enthalten ist. Mit ihm sind folgende Verarbeitungsbefehle erlaubt:

- Bit setzen

- Bit löschen

- Bit invertieren

- einen bedingten Sprung ausführen, wenn ein bestimmtes Bit gesetzt ist
- einen bedingten Sprung ausführen, wenn ein bestimmtes Bit gelöscht ist
- einen bedingten Sprung ausführen, wenn ein bestimmtes Bit gesetzt ist und das Bit anschließend löschen
- Bittransfer vom und zum Carry-Bit
- Verknüpfungen von adressierbaren Bits oder deren Komplement mit dem Carry-Bit (logische UND- bzw. ODER-Verknüpfung)

17.5 Definitionen zur Befehlsbeschreibung

Die 111 Befehle des 80(C)515/-535 können auf 54 Basisbefehle zurückgeführt werden, die im folgenden alphabetisch geordnet im Detail beschrieben sind. Meistens ist ein kurzes Anwendungsbeispiel angegeben. Die Anzahl der Befehlsbytes und die zur Abarbeitung benötigten Maschinenzyklen sind ebenfalls aufgeführt.

Der jeweilige Einfluß auf die Flags im Programm-Status-Wort (PSW) ist bei jedem Befehl vermerkt. Es gilt zu beachten, daß nur der Einfluß auf das Carry-, das Hilfscarry- und das Überlaufflag (CY, AC, OV) besprochen ist. Das Paritätsbit wird automatisch nach Abschluß der Operation neu berechnet. Bei allen Befehlen, die SFR beschreiben können ist zu beachten, daß sie auch damit die Status-Flags unmittelbar verändern. Ebenso ist es möglich, Status-Flags durch Bitbefehle zu manipulieren.

Innerhalb der Befehlsbeschreibungen werden spezifische Abkürzungen verwendet. Die folgende Tabelle und die Bemerkung im Anschluß erläutern diese.

Rn	Eines der Register R0 bis R7 der aktuellen Registerbank. Bei der Codierung wird die Auswahl des Registers durch „r r r“ vertreten (rrr=000 bis rrr=111).
Direct address	Eine Speicherzelle mit der Adresse „direct address“ im unteren internen RAM oder ein Special Function Register
@Ri	Eine durch R0 oder R1 indirekt adressierte Speicherzelle im internen oder externen RAM. Bei der Codierung wird die Auswahl des Registers durch „i“ vertreten. (i=0 oder i=1)
#data	8-Bit-Konstante, die Bestandteil des Befehls ist
#data16	16-Bit-Konstante, die die Bytes 2 und 3 des Befehls bildet.
Bit	Die 128 Bitadressen im internen RAM oder jede bitadressierbare Zelle in den SFR
A	Akkumulator, Akku
adr16	16-Bit-Zieladresse für LCALL- oder LJMP-Befehle; kann überall innerhalb des 64 k-Programmspeichers liegen
adr11	11-Bit-Zieladresse für ACALL- oder AJMP-Befehle; kann überall innerhalb derselben 2k-Seite liegen wie das erste Byte des auffordernden Befehls (siehe Anmerkung)
rel	SJMP und alle bedingten Sprungbefehle enthalten eine relative 8-Bit-Sprungadresse. Der Sprungbereich ist +127/-128 Bytes relativ zum ersten Byte des darauffolgenden Befehls

Tabelle 17.2: Begriffe und Abkürzungen im Befehlssatz

17.5.1 Anmerkung zu den Befehlen ACALL und AJMP

Der 80(C)515/-535 kann 64-kByte Programm-Speicher adressieren. Dazu benötigt er 16-Bit-Adessen. Eine Seite von 2-kByte ist bereits mit elf Adreßbits festgelegt. Daher wird bei den Befehlen ACALL und AJMP nur eine elfstellige Adresse angegeben. Die oberen fünf Adressenbits bleiben daher im Programm-Zähler bei der Verwendung eines dieser Befehle erhalten. Somit ergeben sich wegen der Codierung der Befehle jeweils acht sog. „pages“, für die die Befehle einen anderen Binär- oder Hexcode besitzen. Dabei ist eine „page“ der Adreßbereich von 00H bis FFH des niederwertigen Byte einer 16-Bit-Adresse.

Prinzipiell lassen sich die Binärcodes des Befehle so darstellen:

F F F F F P2 P1 P0 0 0 0 0 1

für den Befehl AJMP und

F F F F F P2 P1 P0 1 0 0 0 1

für den Befehl ACALL. Dabei sind „F“ die fünf festen Adreßbits, die im PC erhalten bleiben.

Mit P2 bis P0 wird die Seite festgelegt und die übrigen Bits sind befehlsspezifisch.

17.6 Liste der Befehle und deren Beschreibung

In den Beschreibungen werden Abkürzungen oder Symbole verwendet, die vorab vorgestellt werden sollen:

(X)	Heißt „der Inhalt von X“. Dabei kann X ein Flag, ein Register, ein adressierbares Bit oder eine Speicherzelle sein.
((X))	Heißt „der Inhalt dessen Adresse in X steht“. Hier wird nicht der Inhalt von selbst verändert, sondern der des Bits, der Speicherzelle dessen/deren Adresse in X steht. X wirkt als 8-Bit-Zeiger.
Y ← Wert	Zuweisung: der Variablen Y wird der Wert „Wert“ zugewiesen. Mit anderen Worten: der Ausdruck, der rechts vom Pfeil steht, wird dem links stehenden übergeben
Y ↔ X	Die Variablen X und Y tauschen ihre Inhalte.
¬ X	„NICHT X“. Logische NICHT-Verknüpfung.
X ∧ Y	„X UND Y“. Logische UND-Verknüpfung.
X ∨ Y	„X ODER Y“. Logische ODER-Verknüpfung.
X ⊕ Y	„X EXOR Y“. Logische EXKLUSIV-ODER-Verknüpfung.

ACALL adr11

springe zu Unterprogramm innerhalb der aktuellen 2-kByte-Seite

Beschreibung:

Absoluter Sprung zu einem Unterprogramm. Der Befehl inkrementiert den PC um 2 auf die Adresse des dem ACALL-Befehl folgenden Befehls. Dann wird dieses Ergebnis im Stack gesichert (zuerst das niederwertige Adreßbyte) und der SP um zwei inkrementiert. Die 11 niederwertigen Bits des PC werden duch die Zieladresse des ACALL ersetzt. Die 5 höherwertigen bleiben im PC erhalten. Deshalb muß das Unterprogramm sich in derselben 2-kByte-Seite wie der ACALL-Befehl. Liegt der ACALL in den letzten beiden Bytes der 2-kByte-Seite, geht der Sprung in die folgende Seite, weil der PC vor der Befehlsausführung inkrementiert wurde.

Beispiel:

Der SP wurde mit 07H initialisiert. Das Label „SUBRTN“ ist der Anfang eines Unterprogramms mit der Adresse 0345H. Nach der Ausführung des Befehls
ACALL SUBRTN
von der Speicheradresse 0123H und 0124H enthält der SP 09H, die interne RAM-Adresse (Stack) 08H und 09H enthalten 25H und 01H und der PC enthält 0345H

Einzeloperationen:

(PC) ← (PC)+2
(SP) ← (SP)+1
((SP)) ← (PC.7-0)
(SP) ← (SP)+1
((SP)) ← (PC.15-8)
(PC.10-0) ← page address (Adressenbits a10 bis a0)

Codierung des Befehls:

a10 a9 a8 1 0 0 0 1	a7 a6 a5 a4 a3 a2 a1 a0

Beeinflußte Flags im PSW:

Keine

Bytes:

2

Zyklen:

2

ADD A,<Quellbyte>

addiere

Beschreibung:	Der Befehl addiert zum Inhalt des Akkumulators den Inhalt des Operanden „Quellbyte“ und liefert das Ergebnis im Akkumulator zurück. Das CY und das AC im PSW werden bei einem Übertrag von Bit 7 bzw. 3 gesetzt, sonst gelöscht. OV wird gesetzt, wenn es einen Übertrag von Bit 7, nicht aber von Bit 6, oder von Bit 6, aber nicht von Bit 7 gibt. Andernfalls wird OV gelöscht. Bei der Addition vorzeichenbehafteter Zahlen ist dadurch zu erkennen, ob die Addition zweier positiver Zahlen ein negatives oder die Addition zweier negativer Zahlen ein positives Ergebnis ergab (d.h. der Wertebereich der Vorzeichenbehafteten Zahlen wurde überschritten). Als „Quellbyte“ stehen vier mögliche Quellen zur Verfügung: die Register (Rn) der aktiven Registerbank, eine direkt adressierte Speicherzelle (direct), eine durch die Register R0 und R1 adressierte Speicherzelle (register indirect) oder eine 8-Bit Konstante (immediate).
Beispiel:	Der Akku enthält den Wert C3H (1100 0011B) und R0 AAH (1010 1010B). Nach dem Befehl ADD A,R0 enthält A den Wert 6DH (0110 1101B), AC ist gelöscht. CY, P und OV sind gesetzt.
Einzeloperationen:	(A) ← (A)+(Quellbyte)
Beeinflußte Flags im PSW:	CY, P, A, OV

ADD A,Rn

Einzeloperationen:	(A) ← (A)+(Rn)
Codierung des Befehls:	0 0 1 0 1 r r r
Beeinflußte Flags im PSW:	CY, P, A, OV
Bytes:	1
Zyklen:	1

ADD A,direct

Einzeloperationen:	(A) ← (A)+(direct)	
Codierung des Befehls:	0 0 1 0 0 1 0 1	direct address
Beeinflußte Flags im PSW:	CY, P, A, OV	
Bytes:	2	
Zyklen:	1	

ADD A,@Ri

Einzeloperationen:	(A) ← (A)+((Ri))
Codierung des Befehls:	0 0 1 0 0 1 1 i
Beeinflußte Flags im PSW:	CY, P, A, OV
Bytes:	1
Zyklen:	1

ADD A,#data

Einzeloperationen	(A) ← (A)+#data
Codierung des Befehls:	0 0 1 0 0 1 0 0 | immediate data
Beeinflußte Flags im PSW:	CY, P, A, OV
Bytes:	2
Zyklen:	1

ADDC A, <Quellbyte>

addiere Quellbyte + Carry zu A

Beschreibung:	Der Befehl addiert den Inhalt des Operanden „Quellbyte“ und den Inhalt des Carry-Flags zum Inhalt des Akku. Das CY und AC werden bei einem Übertrag von Bit 7 bzw. Bit 3 gesetzt, sonst gelöscht. OV wird gesetzt, wenn es einen Übertrag von Bit 7, nicht aber von Bit 6, oder von Bit 6, aber nicht von Bit 7 gibt. Andernfalls wird OV gelöscht. Bei der Addition vorzeichenbehafteter Zahlen ist dadurch zu erkennen, ob die Addition zweier positiver Zahlen ein negatives oder die Addition zweier negativer Zahlen ein positives Ergebnis ergab (d.h. der Wertebereich der Vorzeichenbehafteten Zahlen wurde überschritten). Als „Quellbyte“ stehen vier mögliche Quellen zur Verfügung: die Register (Rn) der aktiven Registerbank, eine direkt adressierte Speicherzelle (direct), eine durch die Register R0 und R1 adressierte Speicherzelle (register indirect) oder eine 8-Bit Konstante (immediate).
Beispiel:	Der Akku enthält den Wert C3H (1100 0011B) und R0 AAH (1010 1010B) und CY ist gesetzt. Nach dem Befehl ADDC A,R0 enthält A den Wert 6EH (0110 1110B), AC ist gelöscht. CY, P und OV sind gesetzt.
Einzeloperationen:	(A) ← (A) + (C) + (Quellbyte)
Beeinflußte Flags im PSW:	CY, P, A, OV

ADDC A,Rn

Einzeloperationen:	(A) ← (A)+(C)+(Rn)
Codierung des Befehls:	0 0 1 1 1 r r r
Beeinflußte Flags im PSW:	CY, P, A, OV
Bytes:	1
Zyklen:	1

ADDC A,direct

Einzeloperationen:	(A) ← (A)+(C)+(direct)	
Codierung des Befehls:	0 0 1 1 0 1 0 1	direct address
Beeinflußte Flags im PSW:	CY, P, A, OV	
Bytes:	2	
Zyklen:	1	

ADD A,@Ri

Einzeloperationen:	(A) ← (A)+(C)+((Ri))
Codierung des Befehls:	0 0 1 1 0 1 1 i
Beeinflußte Flags im PSW:	CY, P, A, OV
Bytes:	1
Zyklen:	1

ADDC A,#data

Einzeloperationen:	(A) ← (A)+(C)+#data
Codierung des Befehls:	0 0 1 1 0 1 0 0 | immediate data
Beeinflußte Flags im PSW:	CY, P, A, OV
Bytes:	2
Zyklen:	1

AJMP adr11

springe unbedingt innerhalb der aktuellen 2-kByte-Seite

Beschreibung:	Absoltuer Sprung innerhalb einer 2-kByte-Seite. Der Befehl transferiert Bit 0 bis Bit 10 seines Adreßteils in den PC. Die höherwertigen 5 Bits bleiben im PC erhalten. Deshalb muß sich die Zieladresse immer in derselben 2-kByte-Seite des Programmspeichers befinden. Liegt der AJMP-Befehl in den letzten beiden Bytes der Seite, geht der Sprung in die folgende Seite, weil der PC vor der Befehlsausführung inkrementiert wird.
Beispiel:	Das Label „JMPADR“ liegt im Programmspeicher an der Stelle 0123H. Der Befehl AJMP JMPADR an der 0345H ersetzt den Inhalt des PC durch 0123H
Einzeloperationen:	(PC) ← (PC)+2 (PC.10-0) ← page address (Adressenbits a10 bis a0)
Codierung des Befehls:	a10 a9 a8 0 0 0 0 1 | a7 a6 a5 a4 a3 a2 a1 a0
Beeinflußte Flags im PSW:	Keine
Bytes:	2
Zyklen:	2

ANL <Zielbyte>,<Quellbyte>

verknüpfe logisch UND

Beschreibung:	ANL verknüpft Variablen und Speicherinhalte bitweise logisch UND. Die Flags im PSW bleiben unbeeinflußt (außer P, wenn das Ziel der der Akku war). Die beiden Operanden erlauben 6 Adressierungsarten. Ist A der Zieloperand, kann die Quelle ein Registerinhalt, eine direkte oder indirekte Adresse oder eine 8-Bit-Konstante sein. Ist der Zieloperand eine direkte Adresse, kann die Quelle A oder ein Immediate-Byte sein. Ist der Quelloperand ein Ausgangs-Port, ist die Quelle das Port-Register, nicht die Eingangspins.
Beispiel:	A enthält C3H (1100 0011B) und R0 enthält AAH (1010 1010B). Nach dem Befehl ANL A,R0 enthält der Akku 81H (1000 0001B), P=0. Ist der Zieloperand ein direkt adressierbares Byte, kann man beliebige Bitkombinationen in einer RAM-Zelle oder einem SFR löschen. Im „Maskenbyte“ bestimmen die 0-Bits die zu löschenden Stellen. der Befehl ANL P1,#0111 0011B löscht bie Bits 2, 3 und 7 im Port P1.
Einzeloperationen:	(Zielbyte) ← (Zielbyte)∧(Quellbyte)
Beeinflußte Flags im PSW:	P (wenn A das Zielbyte ist)

ANL A,Rn

Einzeloperationen:	(A) ← (A)∧(Rn)
Codierung des Befehls:	0 1 0 1 1 r r r
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	1

ANL A,direct

Einzeloperationen:	(A) ← (A)∧(direct)
Codierung des Befehls:	0 1 0 1 0 1 0 1 | direct address
Beeinflußte Flags im PSW:	P
Bytes:	2
Zyklen:	1

ANL A,@Ri

Einzeloperationen:	(A) ← (A)∧((Ri))
Codierung des Befehls:	0 1 0 1 0 1 1 i
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	1

ANL A,#data

Einzeloperationen:	(A) ← (A)∧#data
Codierung des Befehls:	0 1 0 1 0 1 0 0 | immediate data
Beeinflußte Flags im PSW:	P
Bytes:	2
Zyklen:	1

ANL direct,A

Einzeloperationen:	(direct) ← (direct)∧(A)
Codierung des Befehls:	0 1 0 1 0 1 0 1 | direct address
Beeinflußte Flags im PSW:	Keine
Bytes:	2
Zyklen:	1

ANL direct,#data

Einzeloperationen:	(direct) ← (direct)∧#data
Codierung des Befehls:	0 1 0 1 0 0 1 1 | direct address | immediate data
Beeinflußte Flags im PSW:	Keine
Bytes:	3
Zyklen:	2

ANL C,<Quellbit>

verknüpfe logisch UND mit Bitvariable

Beschreibung:	ANL C verknüpft zwei Bits logisch UND. Ist der Boolsche Wert der Bitquelle 0, wird C gelöscht. Andernfalls behält er seinen ursprünglichen Wert. Der „/“-Operator veranlaßt die CPU, das Quellbit im Speicher invertiert zu interpertieren. Das Quellbit selbst bleibt unverändert.
Beispiel:	Setzt C nur, wenn P1.0=1, ACC.7=1 und OV=0. MOV C,P1.0 ; lade CY mit P1.0 ANL C,ACC.7 ; UND-Verknüpfung CY mit Akkubit 7 ANL C,/OV ; UND-Verknüpfung mit $\overline{OV}$
Einzeloperationen:	(C) ← (C)∧(Quellbit)
Beeinflußte Flags im PSW:	CY

ANL C,bit

Einzeloperationen:	(C) ← (C) ∧ (bit)
Codierung des Befehls:	1 0 0 0 0 0 1 0 | bit address
Beeinflußte Flags im PSW:	CY
Bytes:	2
Zyklen:	2

ANL C,/bit

Einzeloperationen:	(C) ← (C) ∧ ¬ (bit)
Codierung des Befehls:	1 0 1 1 0 0 0 0 | bit address
Beeinflußte Flags im PSW:	CY
Bytes:	2
Zyklen:	2

CALL

spring zu Unterprogramm; siehe bei ACALL, LCALL

CJNE <Zielbte>,<Quellbyte>,rel

vergleiche und springe, wenn nicht gleich

Beschreibung:

Der Befehl CJNE vergleicht den Inhalt zweier Operanden und verzweigt, wenn ihre Werte nicht gleich sind. Das Sprungziel wird im PC durch die Summe des PC-Inhalts plus 3 (auf den Vergleichsbefehl folgende Adresse) und der vorzeichenbehafteten relativen Adreßentfernung (letztes Byte des Befehls) gebildet. Die maximale Sprungweite beträgt -128 bzw. +127 Byte. CY wird gesetzt, wenn der vorzeichenlose Wert des Quellbyts größer als der des Zielbyts ist; andernfalls wird CY gelöscht.

Beispiel:

A enthält 34H und R7 enthält 56H. Der erste Befehl des Porammteils

```
            CJNE    R7,#60H,NOT_EQ    ; (R7)=60
NOT_EQ:  JC         REG_LOW           ; wenn (R7) < 60H
                                      ; (R7) > 60H
```

setzt CY und springt zur Marke NOT_EQ. Der Test des CY bestimmt, ob der Inhalt von R7 größer oder kleiner als 60H ist.

Die Daten an Port 1 sollen auf den Wert 34H in A geprüft werden:

```
WAIT:    CJNE    A,P1,WAIT
```

Der Befehl löscht C und fährt mit dem nächsten Befehl fort, wenn die Werte in A und P1 übereinstimmen. Falls in P1 ein anderer Wert ist, befindet sich das Programm in einer Warteschleife.

Einzeloperationen:

(PC) ← (PC) + 3
wenn (Zielbyte) <> (Quellbyte)
dann (PC) ← (PC) + rel. Address
wenn (Zielbyte) < (Quellbyte)
dann (C) ← 1
sonst (C) ← 0

Beeinflußte Flags im PSW:

CY

CJNE A,direct,rel

Einzeloperationen:
(PC) ← (PC) + 3
wenn (A) <> (direct)
dann (PC) ← (PC) + rel. Offset
wenn (A) < (direct)
dann (C) ← 1
sonst (C) ← 0

Codierung des Befehls:

1 0 1 1 0 1 0 1	direct address	rel. Address

Beeinflußte Flags im PSW: CY

Bytes: 3

Zyklen: 2

CJNE A,#data,rel

Einzeloperationen:
(PC) ← (PC) + 3
wenn (A) <> data
dann (PC) ← (PC) + rel. Offset
wenn (A) < data
dann (C) ← 1
sonst (C) ← 0

Codierung des Befehls:

1 0 1 1 0 1 0 0	immediate data	rel. Address

Beeinflußte Flags im PSW: CY

Bytes: 3

Zyklen: 2

CJNE Rn,#data,rel

Einzeloperationen:
(PC) ← (PC) + 3
wenn (Rn) <> (data)
dann (PC) ← (PC) + rel. Offset
wenn (Rn) < data
dann (C) ← 1
sonst (C) ← 0

Codierung des Befehls:	1 0 1 1 1 r r r	immediate data	rel. Address

Beeinflußte Flags im PSW: CY

Bytes: 3

Zyklen: 2

CJNE @Ri,data,rel

Einzeloperationen:
(PC) ← (PC) + 3
wenn (Ri) <> data
dann (PC) ← (PC) + rel. Offset
wenn (Ri) < data
dann (C) ← 1
sonst (C) ← 0

Codierung des Befehls:	1 0 1 1 0 1 1 i	immediate data	rel. Address

Beeinflußte Flags im PSW: CY

Bytes: 3

Zyklen: 2

CLR A
lösche Akkumulator

Beschreibung:	A wird gelöscht, d.h. alle Bit im Akkumulator werden 0 gesetzt. Außer P wird keines der PSW-Flags beeinflußt
Beispiel:	CLR A
Einzeloperationen:	(A) ← 0
Codierung des Befehls:	1 1 1 0 0 1 0 0
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	1

CLR <Bit>
lösche Bit

Beschreibung:	Das adressiert Bit wird gelöscht.
Beispiel:	Port 1 enthält 5BH (0101 1011B). Nach dem Befehl CLR P1.1 steht am Port-Ausgang der Wert 59H (0101 1001B)
Einzeloperationen:	(Bit) ← 0
Beeinflußte Flags im PSW:	Siehe unten
Bytes:	1
Zyklen:	1

CLR C

Einzeloperationen:	(C) ← 0
Codierung des Befehls:	1 1 0 0 0 0 1 1
Beeinflußte Flags im PSW:	P, C
Bytes:	1
Zyklen:	1

CLR bit

Einzeloperationen:	(bit) ← 0	
Codierung des Befehls:	1 1 0 0 0 0 1 0	bit address
Beeinflußte Flags im PSW:	P, falls ein Bit aus dem Akku betroffen war	
Bytes:	2	
Zyklen:	1	

CPL A

negiere Akkumulator (Einerkomplement)

Beschreibung:	Bildet die bitweise Negierung des Akku-Inhalts
Beispiel:	A enthält 5C (0101 1100B). Nach dem Befehl CPL A enthält A A3H (1010 0011B)
Einzeloperationen:	(A) ← ¬ (A)
Codierung des Befehls:	1 1 1 1 0 1 0 0
Beeinflußte Flags im PSW:	Keines
Bytes:	1
Zyklen:	1

CPL <Bit>

komplmentiere Bit

Beschreibung:	Das adressierte Bit wird logisch invertiert. Aus 0 wird 1 und aus 1 wird 0. Adressiert der Befehl einen Port-Pin, wird nicht der Pin, sondern das Port-Latch-Bit eingelesen und komplementiert.
Beispiel:	Port 1 enthält 5DH (0101 1101B). Nach den Befehlen CPL P1.1 CPL P1.2 enthält Port 1 5BH (0101 1011B)
Einzeloperationen:	(bit) ← ¬ (bit)
Beeinflußte Flags im PSW:	keine

CPL C

Einzeloperationen:	(C) ← ¬ (C)
Codierung des Befehls:	1 0 1 1 0 0 1 1
Beeinflußte Flags im PSW:	C
Bytes	1
Zyklen	1

CPL bit

Einzeloperationen:	(bit) ← ¬ (bit)	
Codierung des Befehls:	1 0 1 1 0 0 1 0	bit address
Beeinflußte Flags im PSW:	P	
Bytes	2	
Zyklen	1	

DA A

Dezimalkorrektur des Akkumulators für Addition

Beschreibung:

In A steht ein 8-Bit-Wert als Summe nach einer vorhergehenden Addition. DA A korrigiert diesen Wert zu zwei BCD-Ziffern (zusammen in einem gepackten Format). Das Ergebnis sind zwei 4-Bit-Ziffern im Dezimalformat.
Falls der Wert der Bits 3-0 in A größer als 9 ist (1010 bis 1111) oder falls AC gesetzt ist, wird 6 zu A addiert. Diese Addition setzt C, wenn bei dieser Korrekturaddition ein Übertag in höcherwertige Nibble erfolgt, andernfalls wird es jedoch nicht gelöscht. Ist jetzt C gesetzt, oder das höherwertige Nibble größer als 9, wird 6 zum höherwertigen Nibble addiert, um eine exakte BCD-Zahl zu erzeugen. Wieder wird bei einem Übertrag C gesetzt, andernfalls jedoch nicht gelöscht.
Ist C gesetzt, war die Summe der Addition größer als 99. Es sind damit Additionen von größeren Dezimalzahlen möglich. Diese Operationen laufen während eines Maschinenzyklus ab.
DA A sollte nicht nach einer Subtraktion angewendet werden. Der Befehl leistet keine Binär-Dezimal-Konvertierung (außer für Werte zwische 0 und 15 falls AC=0).

Beispiel:

A enthält 56H (0101 0110B), die gepackte Dezimalzahl 56. R3 enthält mit 67H (0110 0111B) die gepackte Dezimalzahl 67. CY ist gesetzt. Die Befehlen

```
ADDC    A,R3
DA      A
```

bewirken zuerst eine Zweierkomplement-Binär-Addition. Diese ergibt BEH (1011 1110B) in A. CY und AC sind gelöscht.
DA A ändert den Inhalt von A in 24. CY wird gesetzt, weil die Dezimalzahl größer als 99 (56+67+1=124) ist.

BCD-Variablen können durch die Addition von 99H oder 1H dekrementiert oder inkrementiert werden. Enthält A 30H (als Dezimalzahl), so ergibt sich aus der Befehlsfolge

```
ADD     A,#99H
DA      A
```

29 in A. CY ist gesetzt, weil 30+99=129 ist. Das niederwertige Byte der Summe kann als 30-1=29 interpretiert werden.

Einzeloperationen:

A enthält eine BCD-Zahl:
wenn [[(A3-0) > 9] oder [(AC)=1]],
dann (A3-0) ← (A3-0))+6
und wenn [[(A7-4) > 9] oder [(C)=1]],
dann (A7-4) ← (A7-4)+6

Codierung des Befehls:	1 1 0 1 0 1 0 0
Beeinflußte Flags im PSW:	P, CY, AC
Bytes:	1
Zyklen:	1

DEC <Byte>

dekrementiere Byte

Beschreibung:	Die Variable „Byte“ wird um 1 vermindert, aus 00H wird FFH. PSW wird nicht beeinflußt (außer bei DEC A das Flag P)
Beispiel:	R0 enthält 7FH. Die Bytes des internen RAM 7EH und 7FH enthalten 0H und 40H. Nach den Befehlen DEC @R0 DEC R0 DEC @R0 enthält R0 den Wert 7EH; in 7EH steht FFH und in 7FH steht 3FH.
Einzeloperationen:	(Byte) ← (Byte) - 1
Beeinflußte Flags im PSW:	P, falls Byte=A

DEC A

Einzeloperationen:	(A) ← (A) - 1
Codierung des Befehls:	0 0 0 1 0 1 0 0
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	1

DEC Rn

Einzeloperationen:	(Rn) ← (Rn) - 1
Codierung des Befehls:	0 0 0 1 1 r r r
Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	1

DEC direct

Einzeloperationen:	(direct) ← (direct) - 1	
Codierung des Befehls:	0 0 0 1 0 1 0 1	direct address
Beeinflußte Flags im PSW:	P, falls direct=A	
Bytes:	2	
Zyklen:	1	

DEC @Ri

Einzeloperationen:	((Ri)) ← ((Ri)) - 1
Codierung des Befehls:	0 0 0 1 0 1 1 i
Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	1

DIV AB

dividiere den Akkumulator durch B

Beschreibung:	Der befehl DIV AB dividiert eine vorzeichenlose 8-Bit-Zahl in A durch eine vorzeichenlose 8-Bit-Zahl in B. Nach dem Befehl steht der Quotient (als ganzzahliger Anteil) in A und der Divisionsrest in B. CY und OV sind gelöscht
Beispiel:	A enthält 251=FBH (1111 1011B) und B 18=12H (0001 0010B). Bach dem Befehls DIV AB ist in A 13=0DH (0000 1101B) und in B 17=11H (0001 0001B), weil 251=13*18+17 ist. CY und OV sind gelöscht, P=1.
Einzeloperationen:	(A) ← Int[(A) / (B)] (B) ← (B) * [(A) / (B) - Int[(A) / (B)]]
Codierung des Befehls:	1 0 0 0 0 1 0 0
Beeinflußte Flags im PSW:	CY, P, OV
Bytes:	1
Zyklen:	4

DJNZ <Byte>,<rel>

dekrementiere und springe wenn <>0

Beschreibung: DJNZ dekrementiert das adressierte Byte um 1 und verzweigt, wenn des Ergebnis nicht 0 ist. Ist der Inhalt des Bytes vor dem Befehl 00H, läuft es auf FFH. Das PSW wird nicht beeinflußt. Das Sprungziel errechnet sich aus der Summe des vorzeichenbehafteten, relativen 8-Bit-Werts im letzten Befehlsbyte und der Adresse des auf den DJNZ folgenden Befehls (-128 bis +127). Dekrementiert der Befehl einen Port, so bezieht er sich auf das Port-Latch und nicht auf die Port-Pins.

Beispiel: Mit DJNZ lassen sich leicht Zeitschleifen (Zählschleifen) realisieren. Die Befehlsfolge

```
          MOV   R2,#8H
TOGGLE:   CPL   P1.7
          DJNZ  R2,TOGGLE
```

schaltet P1.7 achtmal um, wodurch vier Impulse entstehen. Jeder Impuls benötigt drei Maschinenzyklen (zwei für DJNZ und einen für das Komplementieren des P1.7)

Einzeloperationen:
(PC) ← (PC) + 2
(byte) ← (byte) - 1, wenn (byte) <>0
dann (PC) ← (PC) + rel. offset

DJNZ Rn,rel

Einzeloperationen:
(PC) ← (PC) + 2
(Rn) ← (Rn) - 1, wenn (Rn) <>0
dann (PC) ← (PC) + rel. offset

Codierung des Befehls:	1 1 0 1 1 r r r	rel. address

Beeinflußte Flags im PSW: keine

Bytes: 2

Zyklen: 2

DJNZ direct,rel

Einzeloperationen: (PC) ← (PC) + 2
(direct) ← (direct) - 1, wenn (direct) <>0
dann (PC) ← (PC) + rel. offset

Codierung des Befehls:

1 1 0 1 0 1 0 1	direct address	rel. address

Beeinflußte Flags im PSW: P, falls direct=A

Bytes: 3

Zyklen: 2

INC <Byte>

inkrementiere Byte

Beschreibung: Die Variable „Byte“ wird um 1 inkrementiert. Bei FFH läuft sie durch den Befehl auf 00H über. Die PSW-Flags werden nicht beeinflußt, außer bei INC A das P-Flag. Wird ein Port inkrementiert, wird zum Port-Latch-Inhalt eine 1 addiert und dann das Ergebnis ausgegeben.

Einzeloperationen: (byte) ← (byte) + 1

Beeinflußte Flags im PSW: siehe unten

INC A

Einzeloperationen:	(A) ← (A) + 1
Codierung des Befehls:	0 0 0 0 0 1 0 0
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	1

INC Rn

Einzeloperationen:	(Rn) ← (Rn) + 1
Codierung des Befehls:	0 0 0 0 1 r r r
Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	1

INC direct

Einzeloperationen:	(direct) ← (direct) + 1	
Codierung des Befehls:	0 0 0 0 0 1 0 1	direct address
Beeinflußte Flags im PSW:	P, falls direct=A	
Bytes:	2	
Zyklen:	1	

INC @Ri

Einzeloperationen:	((Ri)) ← ((Ri)) + 1
Codierung des Befehls:	0 0 0 0 0 1 1 i
Beeinflußte Flags im PSW:	P, falls (Ri)=A
Bytes:	1
Zyklen:	1

INC DPTR

inkrementiere Data-Pointer

Beschreibung:	Der Befehl INC DPTR inkrementiert den 16 Bit breiten Data-Pointer. DPTR ist das einzige 16-Bit-Register, das inkrementiert werden kann. Ein Dekrementieren des DPTR ist nicht möglich.
Beispiel:	In Teilregistern DPH und DPL stehen die Werte 12H und FEH. die Befehle INC DPTR INC DPTR INC DPTR ändern DPH in 13H und DPL in 01H.
Einzeloperationen:	(DPTR) ← (DPTR) + 1
Codierung des Befehls:	1 0 1 0 0 0 1 1
Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	2

JB bit,rel

springe, wenn Bit gesetzt

Beschreibung: Ist das mit „bit“ adressiert Bit=1, wird zur angegebenen Adresse verzweigt. Andernfall wird das Programm mit dem nächsten Befehl fortgesetzt. Die Sprungweite ist maximal -128 bis +127, bezogen auf die Adresse des auf JB folgenden Befehls.

Beispiel: P1 enthält 1100 1010B. Nach dem Befehl
JB P1.1,Marke_2
verzweigt das Programm zur Marke_2, da P1.1=1 war.

Einzeloperationen: (PC) ← (PC) + 3
wenn (bit)=1
dann (PC) ← (PC) + rel

Codierung des Befehls:

0 0 1 0 0 0 0 0	bit address	rel. address

Beeinflußte Flags im PSW: keine

Bytes: 3

Zyklen: 2

JBC bit,rel

springe, wenn Bit gesetzt ist und lösche Bit

Beschreibung:	Ist das adressierte Bit 1, wird zur angegebenen Adresse verzweigt und das Bit gelöscht. Andernfalls wird das Programm mit dem nächsten Befehl fortgesetzt. Die Sprungweite ist von maximal -128 bis +127 bezogen auf die Adresse des auf JBC folgenden Befehls. Ist ein Port adressiert, bezieht sich der Befehl auf das Port-Latch und nicht auf die Port-Pins.
Beispiel:	A enhält 56H (0101 0110B). Die Befehle JBC ACC.3,LABEL1 JBC ACC.2,LABEL2 lassen das Programm zu der Stelle LABEL2 springen und der Akku enthält danach 52H (0101 0010B).
Einzeloperationen:	(PC) ← (PC) + 3 wenn (bit) = 1 dann (bit) ← 0 (PC) ← (PC) + rel

Codierung des Befehls:			
	0 0 0 1 0 0 0 0	bit address	rel. address

Beeinflußte Flags im PSW:	keine
Bytes:	3
Zyklen:	2

JC rel

springe, wenn C gesetzt

Beschreibung:	Ist der Inhalt von C=1, wird zur angegebenen Adresse verzweigt. Andernfalls wird das Programm mit dem nächsten Befehl fortgesetzt. Die Sprungweite ist von maximal -128 bis +127 bezogen auf die Adresse des auf JC folgenden Befehls.
Beispiel:	C ist gelöscht. Nach den Befehlen JC Marke_1 CPL C JC Marke_2 verzweigt das Programm an die Stelle Marke_2.
Einzeloperationen:	(PC) ← (PC) + 2 wenn (C) = 1 dann (PC) ← (PC) + rel

Codierung des Befehls:		
	0 1 0 0 0 0 0 0	rel. address

Beeinflußte Flags im PSW:	keines
Bytes:	2
Zyklen:	2

JMP

spring unbedingt; siehe bei AJMP, LJMP oder SJMP

JMP @A+DPTR

springe direkt

Beschreibung:	Die Summe aus dem vorzeichenlosen Inhalt von A und dem Inhalt des 16-Bit-Data-Pointer ergibt die Adresse des nächsten Befehls. A und DPTR bleiben unverändert.
Beispiel:	In A steht einer der Werte 0, 2, 4 oder 6. Die folgenden Befehle verzweigen auf einen der vier AJMP-Befehle in der Sprungtabelle: MOV DPTR.#JMP_TBL JMP @A+DPTR JMP_TBL: AJMP MARKE_0 AJMP MARKE_1 AJMP MARKE_2 AJMP MARKE_3 Enthält der Akku 4H, verzweigt das Programm zu MARKE_2. Ungerade Zahlen führen wegen der 2-Byte-AJMP-Befehle zu undefinierten Programmzuständen und sind deshalb vorher abzufragen.
Einzeloperationen:	(PC) ← (A) + (DPTR)
Codierung des Befehls:	0 1 1 1 0 0 1 1
Beeinflußte Flags im PSW:	keines
Bytes:	1
Zyklen:	2

JNB bit,rel

springe, wenn Bit nicht gesetzt

Beschreibung: Ist das adressierte Bit 0, wird zur angegebenen Adress verzweigt. Andernfall wird das Programm mit dem nächsten Befehl fortgesetzt. Die Sprungweite ist von maximal -128 bis +127 bezogen auf die Adresse des auf JNB folgenden Befehls.

Beispiel: P1 enthält 1100 1010B, A enfhält 56H (0101 0110B). Nach den Befehlen

```
JNB     P1.3,LABEL_1
JNB     ACC.3,LABEL_2
```

wird das Programm an der Stelle LABEL_2 fortgesetzt.

Einzeloperationen:

(PC) ← (PC) + 3
wenn (bit) = 0
dann (PC) ← (PC) + rel

Codierung des Befehls:

0 0 1 1 0 0 0 0	bit address	rel. address

Beeinflußte Flags im PSW: keine

Bytes: 3

Zyklen: 2

JNC rel

springe, wenn C nicht gesetzt

Beschreibung:	Ist C=0, wird zur angegebenen Adresse verzweigt. Andernfall wird das Programm mit dem nächsten Befehl fortgesetzt. Die Sprungweite ist von maximal -128 bis +127 bezogen auf die Adresse des auf JNC folgenden Befehls.
Beispiel:	C sei gesetzt. Nach den Befehlen JNC MARKE_1 CPL C JNC MARKE_2 wird das Programm an Stelle MARKE_2 fortgesetzt.
Einzeloperationen:	(PC) ← (PC) + 2 wenn (C) = 0 dann (PC) ← (PC) + rel

Codierung des Befehls:		
	0 1 0 1 0 0 0 0	rel. address

Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	2

JNZ rel

springe, wenn A <> 0

Beschreibung:	Ist der Inhalt von A nicht 0, wird zur angegebenen Adresse verzweigt. Andernfall wird das Programm mit dem nächsten Befehl fortgesetzt. Die Sprungweite ist von maximal -128 bis +127 bezogen auf die Adresse des auf JNZ folgenden Befehls.
Beispiel:	A enthält 00H. Nach den Befehlen JNZ MARKE_1 INC A JNZ MARKE_2 wird das Programm an Stelle MARKE_2 fortgesetzt und in A steht 01H.
Einzeloperationen:	(PC) ← (PC) + 2 wenn (A) ≠ 0 dann (PC) ← (PC) + rel
Codierung des Befehls:	0 1 1 1 0 0 0 0 | rel. address
Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	2

JZ rel

springe, wenn A = 0

Beschreibung:	Ist der Akkumulator 0, wird zur angegebenen Adresse verzweigt. Andernfall wird das Programm mit dem nächsten Befehl fortgesetzt. Die Sprungweite ist von maximal -128 bis +127 bezogen auf die Adresse des auf JZ folgenden Befehls.
Beispiel:	A enthält 01H. Nach den Befehlen JZ MARKE_1 DEC A JZ MARKE_2 wird das Programm an Stelle MARKE_2 fortgesetzt und A enthält 00H.
Einzeloperationen:	(PC) ← (PC) + 2 wenn (A) = 0 dann (PC) ← (PC) + rel
Codierung des Befehls:	0 1 1 0 0 0 0 0 | rel. address
Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	2

LCALL adr16

springe unbedingt in ein Unterprogramm an der angegebenen 16-Bit-Adresse (Long Call)

Beschreibung:

LCALL verfügt über eine 16-Bit-Adresse und erreicht deshalb jeden Programm-Speicherplatz. Der Befehl addiert 3 zum PC und rettet das Ergebnis als Rücksprungadresse auf den Stack (zuerst das Low-Byte). Dabei wird der SP um 2 inkrementiert. Der PC wird dann mit der LCALL-Adresse (zweites und drittes Byte) geladen, und das Programm fährt an dieser Stelle fort.

Beispiel:

SP war mit 07H initialisiert. Das Label „SUBRTN“ ist mit der Adresse 1234H verbunden. Nach dem Befehl
LCALL SUBRTN
bei der Adresse 0123H enthält der SP 09H, die RAM-Speicherzellen 08H und 09H enthalten 26H und 01H; der PC enthält 1234H.

Einzeloperationen:

(PC) ← (PC) + 3
(SP) ← (SP)+1
((SP)) ← (PC.7-0)
(SP) ← (SP)+1
((SP)) ← (PC.15-8)
(PC) ← adr15-0 (Adressenbits 15 bis 0)

Codierung des Befehls:

0 0 0 1 0 0 1 0	adr15 ... adr8	adr7 ... adr0

Beeinflußte Flags im PSW:

keine

Bytes:

3

Zyklen:

2

LJMP adr16

springe unbedingt zur angegebenen 16-Bit-Adresse

Beschreibung:	Da auf den LJMP-Befehl eine 16-Bit-Adresse folgt, kann jede Stelle im 64-kByte-Programmspeicher erreicht werden. Das niederwertige und das höherwertige Adreßbyte im zweiten und dritten Befehlsbyte werden in den PC geladen.
Beispiel:	Das Label „JMPADR“ ist mit der Stelle 1234H im Programmspeicher verbunden. Mit dem Befehl LJMP JMPADR an der Stelle 0123H wird der Wert 1234H in den PC geladen und das Programm dort fortgesetzt.
Einzeloperationen:	(PC) ← adr15-0 (Adressenbits 15 bis 0)

Codierung des Befehls:			
	0 0 0 0 0 0 1 0	adr15 ... adr8	adr7 ... adr0

Beeinflußte Flags im PSW:	keine
Bytes:	3
Zyklen:	2

MOV <Zielbyte>,<Quellbyte>

transportiere ein Byte aus „Quellbyte“ nach „Zielbyte“

Beschreibung: Der Inhalt des zweiten Operanden (Quellbyte) wird nach „Zielbyte“ (erster Operand) kopiert. Das Quellbyte bleibt unverändert.
Insgesamt gibt es 15 verschiedene MOV-Kombinationen.

Beispiel: Der Speicherplatz 30H des internen RAM enthält 40H, im RAM-Speicherplatz 40H steht der Wert 10H. An Port P1 steht CAH.
Die Befehlsfolge

```
MOV   R0,#30H      ; R0 mit 30H laden
MOV   A,@R0        ; bringt den Wert 40H nach A
MOV   R1,A         ; und kopiert ihn nach R1
MOV   B,@R1        ; bringt den Inhalt von 40H nach B
MOV   @R1,P1       ; kopiert CAH nach Stelle 40H im RAM
MOV   P2,P1        ; und eine Kopie davon nach Port P2
```

bringt 30H nach R0, 40H sowohl in den Akku als auch in das Register R1, 10H in das Register B und CAH sowohl in die RAM-Zelle 40H als auch nach Port P2.

Einzeloperationen: (Zielbyte) ← (Quellbyte)

MOV A,Rn

Beschreibung: Kopiert den Inhalt vom Register Rn (R0 bis R7 der aktiven Registerbank) in den Akku.

Einzeloperationen: (A) ← (Rn)

Codierung des Befehls: 1 1 1 0 1 r r r

Beeinflußte Flags im PSW: P

Bytes: 1

Zyklen: 1

MOV A,direct

Beschreibung:	Kopiert den Inhalt des mit „direct“ adressierten Bytes in den Akku. MOV A,ACC ist dabei unzulässig!
Einzeloperationen:	(A) ← (direct)
Codierung des Befehls:	1 1 1 0 0 1 0 1 | direct address
Beeinflußte Flags im PSW:	P
Bytes:	2
Zyklen:	1

MOV A,@Ri

Beschreibung:	Kopiert den Inhalt des durch Ri (R0 oder R1 der aktiven Registerbank) adressierten Bytes in den Akku.
Einzeloperationen:	(A) ← ((Ri))
Codierung des Befehls:	1 1 1 0 0 1 1 i
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	1

MOV A,#data

Beschreibung:	Lädt den Akku mit einer 8-Bit-Zahl
Einzeloperationen:	(A) ← data
Codierung des Befehls:	0 1 1 1 0 1 0 0 | immediate data
Beeinflußte Flags im PSW:	P
Bytes:	2
Zyklen:	1

MOV Rn,A

Beschreibung:	Kopiert den Inhalt des Akkumulators in das mit Rn bezeichnete Register (R0 bis R7 der aktiven Registerbank)
Einzeloperationen:	(Rn) ← (A)
Codierung des Befehls:	1 1 1 1 1 r r r
Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	1

MOV Rn,direct

Beschreibung:	Kopiert den Inhalt des mit „direct“ adressierten Bytes in eines der Register R0 bis R7 der aktiven Registerbank.
Einzeloperationen:	(Rn) ← (direct)
Codierung des Befehls:	1 0 1 0 1 r r r | direct address
Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	2

MOV Rn,#data

Beschreibung:	Lädt eine 8-Bit-zahl in das Register R0 bis R7 der aktiven Registerbank.
Einzeloperationen:	(Rn) ← data
Codierung des Befehls:	0 1 1 1 1 r r r | immediate data
Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	1

MOV direct,A

Beschreibung:	Kopiert den Inhalt des Akku in ein mit „direct“ adressierten Byte.
Einzeloperationen:	(direct) ← (A)
Codierung des Befehls:	1 1 1 1 0 1 0 1 | direct address
Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	1

MOV direct,Rn

Beschreibung:	Kopiert den Inhalt des mit Rn bezeichneten Registers in das mit „direct“ adressierte Byte
Einzeloperationen:	(direct) ← (Rn)
Codierung des Befehls:	1 0 0 0 1 r r r | direct address
Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	2

MOV direct,direct

Beschreibung:	Kopiert den Inhalt des mit „direct“ adressierten Bytes in ein weiteres mit „direct“ adressiertes Byte.
Einzeloperationen:	(direct) ← (direct)

Codierung des Befehls:	1 0 0 0 01 0 1	direct address (SRC)	direct address (dest)

Beeinflußte Flags im PSW:	keine
Bytes:	3
Zyklen:	2

MOV direct,@Ri

Beschreibung:	Kopiert den Inhalt der mit Ri adressierten Zelle in das mit „direct“ adressierte Byte.
Einzeloperationen:	(direct) ← ((Ri))

Codierung des Befehls:	1 0 0 0 0 1 1 i	direct address

Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	2

MOV direct,#data

Beschreibung:	Lädt das mit „direct“ adressierte Byte mit einer 8-Bit-Zahl.
Einzeloperationen:	(direct) ← data

Codierung des Befehls:	0 1 1 1 0 1 0 1	direct address	immediate data

Beeinflußte Flags im PSW:	keine
Bytes:	3
Zyklen:	2

MOV @Ri,A

Beschreibung:	Kopiert den Inhalt des Akku in das durch R0 oder R1 der aktiven Registerbank adressierte Byte.
Einzeloperationen:	((Ri)) ← (A)

Codierung des Befehls:	1 1 1 1 0 1 1 i

Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	1

MOV @Ri,direct

Beschreibung:	Kopiert den Inhalt des mit „direct“ adressierten Bytes in das durch Ri adressierte Byte.
Einzeloperationen:	((Ri)) ← (direct)
Codierung des Befehls:	1 0 1 0 0 1 1 i | direct address
Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	2

MOV @Ri,data

Beschreibung:	Lädt das durch R1 oder R0 der aktiven Registerbank adressierte Byte mit einer 8-Bit-Zahl.
Einzeloperationen:	((Ri)) ← data
Codierung des Befehls:	0 1 1 1 0 1 1 i | immediate data
Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	1

MOV <Zielbit>,<Quellbit>

transportiere Bit aus „Quellbit“ nach „Zielbit“

Beschreibung: Das Bit des Qulloperanden wird in den Zieloperanden übertragen. Einer der beiden Operanden ist immer das CY-Flag, der andere kann jedes direkt adressierbare Bit sein.

Beispiel: Das Carry-Flag ist gesetzt. Am Eingangs-Port P3 steht 1100 0101B. Zuvor wurde in den Ausgangs-Port P1 35H (0011 0101B) geschrieben. Die Befehle

```
MOV   P1.3,C
MOV   C,P3.3
MOV   P1.2,C
```

löschen das Carry-Flag und ändern den Port P1 zu 39H (0011 1001B).

Einzeloperationen: (Zielbit) ← (Quellbit)

MOV C,bit

Beschreibung: Kopiert den Wert des adressierbaren Bits ins Carry-Flag

Einzeloperationen: (C) ← (bit)

Codierung des Befehls:

1 0 1 0 0 0 1 0	bit address

Beeinflußte Flags im PSW: C

Bytes: 2

Zyklen: 1

MOV bit,C

Beschreibung:	Kopiert den Wert des Carry-Flags in ein adressierbares Bit
Einzeloperationen:	(bit) ← (C)
Codierung des Befehls:	1 0 0 1 0 0 1 0 | bit address
Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	2

MOV DPTR,#data16

lade den Data-Pointer mit einer 16-Bit-Konstanten

Beschreibung:	Der Data-Pointer wird mit einer 16-Bit-Konstanten geladen. Der Wert im zweiten Byte des Befehls wird nach DPH geschrieben (höherwertiges Byte) der aus dem dritten Byte nach DPL.
Beispiel:	Der Befehl MOV DPTR,#1234H lädt DPH mit 12H und DPL mit 34H.
Einzeloperationen:	(DPTR) ← data15-0 dabei (DPH) ← data15-8 und (DPL) ← data7-0
Codierung des Befehls:	1 0 0 1 0 0 0 0 | immediate data15-8 | immediate data7-0
Beeinflußte Flags im PSW:	keine
Bytes:	3
Zyklen:	2

MOVC A,@A+<16-Bit-Register>

lade den Akkumulator indirekt mit einem Wert aus dem Code-Speicher

Beschreibung:

MOVC lädt A mit einem Byte aus dem Programmspeicher. Die Quelladresse errechnet sich aus der Summe der vorzeichenlosen Zahl im Akku und dem Inhalt des 16-Bit-Registers (PC oder DPTR). Ist der PC das Basis-Register, wird der Inhalt des PC auf die Adresse des folgenden Befehls erhöht, bevor die Addition mit A durchgeführt wird. Durch die Addition werden A, PC bzw. DPTR nich beeinflußt.
Eine Anwendung für diesen Befehl ist die Parameterübergabe beim Rücksprung aus einem Unterprogramm bzw. aus einer Interrupt-Routine.

Beispiel:

In A steht ein Wert zwischen 0 und 3. Die folgenden Befehle bringen eine von vier Konstanten nach A. Diese Konstanten seien durch DB-Anweisungen (Define Byte) im Programm definiert.

```
REL_PC:  INC      A
         MOVC     A,@A+PC
         RET
         DB    66H
         DB    77H
         DB    99H
```

Falls beim Einsprung ins Unterprogramm eine 1 in A steht, wird auf 77H zugegriffen. Dieser Wert dann vom Unterprogramm nach A gebracht (MOVC) und an das aufrufende Programm übergeben. Der INC-Befehl am Anfang ist erforderlich, um den RET-Befehl zu überbrücken.

Einzeloperationen:

(PC) ← (PC) + 1
(A) ← ((A) + (PC))
oder
(A) ← ((A) + (DPTR))

MOVC A,@A+DPTR

Einzeloperationen:	(A) ← ((A) + (DPTR))
Codierung des Befehls:	1 0 0 1 0 0 1 1
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	2

MOVC A,@A+PC

Einzeloperationen:	(PC) ← (PC) + 1 (A) ← ((A) + (PC))
Codierung des Befehls:	1 0 0 0 0 0 1 1
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	2

MOVX <Zielbyte>,<Quellbyte>

transportiere ein Byte vom/zum externen RAM

Beschreibung:	Die MOVX-Befehle transferieren Daten zwischen dem Akku und dem externen Datenspeicher. Es gibt MOVX-Befehle mit 8-Bit- oder 16-Bit-Adresse. Beim 8-Bit-MOVX-Befehl liegt die indirekte Adresse in R0 oder R1 der aktiven Registerbank. Adreß- und Datenbyte gehen im Zeitmultiplex über Port P0. Der 16-Bit-MOVX-Befehl arbeitet über den Data-Pointer. Das höherwertige Adreßbyte geht über Port P2, das niederwertige über Port P0 im Zeitmultiplex mit dem Datenbyte.

Beispiel:	An Port P0 ist ein 256-Byte-RAM angeschlossen. R0 und R1 enthalten 12H bzw. 34H. Im externen RAM liegen unter der Adresse 34H der Wert 56H. Nach den Befehlen
	MOVX A,@R1
	MOVX @R0,A
	befindet sich die Zahl 56H im Akku und in der externen RAM-Adresse 12H.

MOVX A,@Ri

Einzeloperationen:	(A) ← ((Ri))
Codierung des Befehls:	1 1 1 0 0 0 1 i
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	2

MOVX A,@DPTR

Einzeloperationen:	(A) ← ((DPTR))
Codierung des Befehls:	1 1 1 0 0 0 0 0
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	2

MOVX @Ri,A

Einzeloperationen:	((Ri)) ← (A)
Codierung des Befehls:	1 1 1 1 0 0 1 i
Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	2

MOVX @DPTR,A

Einzeloperationen:	((DPTR)) ← (A)
Codierung des Befehls:	1 1 1 1 0 0 0 0
Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	2

MUL AB

multipliziere Inhalt von A mit Inhalt von B

Beschreibung:	MUL AB multipliziert die in den Registern A und B enthaltenen vorzeichenlosen 8-Bit-Zahlen. Nach dem Befehl steht das niederwertige Byte des Produkts in A, das högerwertige in B. Ist das Produkt größer als 255 (B>00H), wird OV gesetzt. C ist immer gelöscht.
Beispiel:	A enthält 80 (50H) und B 160 (A0H). Durch MUL AB ergibt sich 12800 (3200H). in A steht 00H, in B 32H und OC=1; C=0.
Einzeloperationen:	(A) ← (A) * (B), Bit7-0 (B) ← (A) * (B), Bit15-8
Codierung des Befehls:	1 0 1 0 0 1 0 0
Beeinflußte Flags im PSW:	P, OV, C
Bytes:	1
Zyklen:	4

NOP

Leerbefehl

Beschreibung:	Der Befehl inkrementiert den PC um 1 und fährt mit dem nächsten Befehl fort. Er hat sonst keine Auswirkungen.
Beispiel:	An P2.7 soll ein Low-Impuls ausgegeben werden, der exakt fünf Maschinenzyklen lang ist. Dazu die Befehle: CLR P2.7 NOP NOP NOP NOP SETB P2.7
Einzeloperationen:	(PC) ← (PC) + 1
Codierung des Befehls:	0 0 0 0 0 0 0 0
Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	1

ORL <Zielbyte>,<Quellbyte>

verknüpfe die beiden Bytes „Zielbyte“ und „Quellbyte“ logisch ODER

Beschreibung:	Der Befehl ORL verknüpft Variablen oder Speicherinhalte logisch mit ODER. Die PSW-Flags bleiben unbeeinflußt (außer P, wenn A das Ziel der Operation ist). Der Befehl ist mit 6 Adressierungsarten ausführbar. Ist A der Zieloperand, kann die Quelle ein Registerinhalt, ein indirekt adressierter Speicherinhalt oder eine 8-Bit-Konstante sein. Ist der Zieloperand eine direkte Adresse, kann die Quelle A oder eine 8-Bit-Konstante sein. Ist die Quelle ein Ausgangs-Port, wird immer das Port-Latch, nicht die Pins gelesen.
Beispiel:	A enthält C3H (1100 0011B) und R0 enthält AAH (1010 1010B). Nach dem Befehl ORL A,R0 enthält A EBH (1110 1011B). Ist der Zieloperand ein direkt adressierbares Byte, kann man beliebige Bitkombinationen in einem SFR oder im RAM setzen. Im „Maskenbyte“ bestimmen die 1-Bits die zu setzenden Stellen. ORL P1,#01110011B setzt die Bits 0, 1, 4, 5 und 6 in P1.
Einzeloperationen:	(Zielbyte) ← (Zielbyte) ∨ (Quellbyte)

ORL A,Rn

Einzeloperationen:	(A) ← (A) ∨ (Rn)
Codierung des Befehls:	0 1 0 0 1 r r r
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	1

ORL A,direct

Einzeloperationen:	(A) ← (A) ∨ (direct)
Codierung des Befehls:	0 1 0 0 0 1 0 1 | direct address
Beeinflußte Flags im PSW:	P
Bytes:	2
Zyklen:	1

ORL A,@Ri

Einzeloperationen:	(A) ← (A) ∨ ((Ri))
Codierung des Befehls:	0 1 0 0 0 1 1 i
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	1

ORL A,#data

Einzeloperationen:	(A) ← (A) ∨ data
Codierung des Befehls:	0 1 0 0 0 1 0 0 | immediate data
Beeinflußte Flags im PSW:	P
Bytes:	2
Zyklen:	1

ORL direct,A

Einzeloperationen: (direct) ← (direct) ∨ (A)

Codierung des Befehls:	0 1 0 0 0 0 1 0	direct address

Beeinflußte Flags im PSW: keine

Bytes: 2

Zyklen: 1

ORL direct,#data

Einzeloperationen: (direct) ← (direct) ∨ data

Codierung des Befehls:	0 1 0 0 0 0 1 1	direct address	immediate data

Beeinflußte Flags im PSW:

Bytes: 3

Zyklen: 2

ORL C,bit

verknüpfe ein Bit logisch mit C

Beschreibung: Der Befehl ORL C verknüpft zwei Bits logisch mit ODER. Ist der boolsche Wert von „bit“ 1, wird CY gesetzt, andernfalls behält es seinen Wert. Der „/“-Operator veranlaßt die CPU, das Quellbit „bit“ negiert zu interpretieren. Das Quellbit selbst bleibt unverändert.

Beispiel: Setzte C, wenn P1.0=1 oder ACC.7=1 oder OV=0

```
MOV    C,P1.0      ; zum Vergleich C mit P1.0 laden
ORL    C,ACC.7     ; wenn P1.0 oder ACC.7 = 1 waren ist C=1
ORL    C,/OV       ; wenn OV 0 war, wird C=1
```

Einzeloperationen: $(C) \leftarrow (C) \vee (bit)$

ORL C,bit

Einzeloperationen: $(C) \leftarrow (C) \vee (bit)$

Codierung des Befehls:

0 1 1 1 0 0 1 0	bit address

Beeinflußte Flags im PSW: C

Bytes: 2

Zyklen: 2

ORL C,/bit

Einzeloperationen:	(C) ← (C) ∨ ¬ (bit)
Codierung des Befehls:	0 1 1 1 0 0 0 0 | bit address
Beeinflußte Flags im PSW:	C
Bytes:	2
Zyklen:	2

POP direct

hole vom Stack

Beschreibung:	Der Inhalt des durch den SP adressierten internen Speicherplatzes wird gelesen und im direkt adressierten Speicherplatz „direct“ abgelegt. Der SP wird anschließend dekrementiert.
Beispiel:	Der SP enthält 32H und im internen RAM liegen bei den Adressen 30H bis 32H die Zahlen 20H, 23H und 01H. Die Befehle: POP DPH POP DPL setzen den Data-Pointer auf 0123H und den SP auf 30H.
Einzeloperationen:	(direct) ← ((SP)) (SP) ← (SP) - 1
Codierung des Befehls:	1 1 0 1 0 0 0 0 | direct address
Beeinflußte Flags im PSW:	P bei POP ACC
Bytes:	2
Zyklen:	2

PUSH direct

bringe zum Stack

Beschreibung:	Der SP wird inkrementiert. Der Inhalt des durch den SP jetzt adressierten internen Speicherplatzes wird mit dem Inhalt des direkt adressierten Speicherplatzes „direct“ überschrieben.
Beispiel:	Beim Sprung in ein Unterporgramm enthält der SP 09H und der DPTR 0123H. Nach den Befehlen PUSH DPL PUSH DPH enthält der SP 0BH und die RAM-Adressen 0AH und 0BH enthalten 23H bzw. 01H.
Einzeloperationen:	(SP) ← (SP) + 1 ((SP)) ← (direct)
Codierung des Befehls:	1 1 0 0 0 0 0 0 | direct address
Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	2

RET

springe aus dem Unterprogramm zurück

Beschreibung:	Der Befehl RET bringt das höherwertige und das niederwertige Byte aus dem über den SP adressierten internen RAM in den PC zurück (Rücksprungadresse). Das Programm fährt dann an der Stelle fort, die dem vorausgegangenen LCALL- oder ACALL-Befehl unmittelbar folgt.
Beispiel:	Im SP steht 0BH, und die RAM-adressen 0AH und 0BH enthalten die Werte 23H und 01H. Nach RET steht im PC 0123H, der SP enthält 09H
Einzeloperationen:	(PC15-8) ← ((SP)) (SP) ← (SP) - 1 (PC7-0) ← ((SP)) (SP) ← (SP) - 1
Codierung des Befehls:	0 0 1 0 0 0 1 0
Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	2

RETI

springe aus der Interrupt-Routine zurück

Beschreibung:	Der Befehl RETI bringt das höherwertige und das niederwertige Byte aus dem über den SP adressierten internen RAM in den PC zurück (Rücksprungadresse) und informiert die Interrupt-Logik, daß die eben bearbeitete Interrupt-Priorität wieder freigegeben werden kann. Das Programm fährt dann an der Stelle fort, die dem vorausgegangenen Interrupt-Routinen-Aufruf unmittelbar folgt.
Beispiel:	Im SP steht 0BH, und die RAM-adressen 0AH und 0BH enthalten die Werte 23H und 01H. Ein Interrupt wurde aktiv, während des Ausführung eines Befehls, der auf der Adresse 0122H endet.Nach RETI steht im PC 0123H, der SP enthält 09H
Einzeloperationen:	(PC15-8) ← ((SP)) (SP) ← (SP) - 1 (PC7-0) ← ((SP)) (SP) ← (SP) - 1
Codierung des Befehls:	0 0 1 1 0 0 1 0
Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	2

RL A

rotiere den Akkumulator bitweise nach links

Beschreibung:	Der Inhalt von A wird um eine Bitstelle nach links rotiert. Das links aus der höchstwertigen Stelle herausgeschobene Bit gelangt in Bit 0 von A. Viermal A zu rotieren ist mit dem Befehl SWAP schneller, als viermal RL A hintereinander.
Beispiel:	A enthält C5H (1100 0101B). Nach dem Befehl RL A steht in A 8BH (1000 1011B).
Einzeloperationen:	(A.n+1) ← (A.n), für n=0-6 (A.0) ← (A.7)
Codierung des Befehls:	0 0 1 0 0 0 1 1
Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	1

RLC A

rotiere den Akkumulator und C bitweise nach links

Beschreibung:	Der Inhalt von A wird um eine Bitstelle nach links rotiert. Das links aus der höchstwertigen Stelle herausgeschobene Bit gelangt nach C und der bisherige Inhalt von CY geht in Bit 0 von A.
Beispiel:	A enthält C5H (1100 0101B) und C ist gelöscht. Nach dem Befehl RLC A steht in A 8AH (1000 1010B) und C ist gesetzt.
Einzeloperationen:	(A.n+1) ← (A.n), für n=0-6 (A.0) ← (C) (C) ← (A.7)
Codierung des Befehls:	0 0 1 1 0 0 1 1
Beeinflußte Flags im PSW:	C, P
Bytes:	1
Zyklen:	1

RR A

rotiere den Akkumulator bitweise nach rechts

Beschreibung:	Der Inhalt von A wird um eine Bitstelle nach rechts rotiert. Das rechts aus der niederwertigsten Stelle herausgeschobene Bit gelangt in Bit 7 von A. Das PSW bleibt unverändert.
Beispiel:	A enthält C5H (1100 0101B). Nach dem Befehl RR A steht in A E2H (1110 0010B).
Einzeloperationen:	(A.n) ← (A.n+1), für n=0-6 (A.7) ← (A.0)
Codierung des Befehls:	0 0 0 0 0 0 1 1
Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	1

RRC A

rotiere den Akkumulator und C bitweise nach rechts

Beschreibung:	Der Inhalt von A wird um eine Bitstelle nach rechts rotiert. Das rechts aus der niederwertigsten Stelle herausgeschobene Bit gelangt nach C und der bisherige Inhalt von C geht in Bit 7 von A.
Beispiel:	A enthält C5H (1100 0101B) und C ist gelöscht. Nach dem Befehl RRC A steht in A 62H (0110 0010B) und C ist gesetzt.
Einzeloperationen:	(A.n) ← (A.n+1), für n=0-6 (A.7) ← (C) (C) ← (A.0)
Codierung des Befehls:	0 0 0 1 0 0 1 1
Beeinflußte Flags im PSW:	C, P
Bytes:	1
Zyklen:	1

SETB <bit>

setzte Bit

Beschreibung:	SETB setzt das adressierte Bit „bit“ auf 1. Es kann C oder jedes andere adressierbare Bit gesetzt werden.
Beispiel:	Das Carry-Flag ist gelöscht. Am Ausgangs-Port 1 steht der Wert 34H (0011 0100B). Mit den Befehlen SETB C SETB P1.0 wird C gesetzt und am Ausgangs-Port 1 steht 34H (0011 0101B).
Einzeloperationen:	(bit) ← 1

SETB C

Beschreibung:	Setzt C auf 1
Einzeloperationen:	(C) ← 1
Codierung des Befehls:	1 1 0 1 0 0 1 1
Beeinflußte Flags im PSW:	C
Bytes:	1
Zyklen:	1

SETB bit

Beschreibung:	Setzt das adressierbare Bit „bit“ auf 1	
Einzeloperationen:	(bit) ← 1	
Codierung des Befehls:	1 1 0 1 0 0 1 0	bit address
Beeinflußte Flags im PSW:		
Bytes:	2	
Zyklen:	1	

SJMP rel

springe relativ (Short Jump)

Beschreibung:	SJMP ist ein unbedingter Verzweigungsbefehl. Die Zieladresse errechnet sich aus der Summe des um 2 erhöhten PC-Inhalts und dem vorzeichenbefafteten relativen Anteil „rel“ des Befehls. Der Sprungbereich reicht von -128 bis +127.
Beispiel:	Das Label „REL_ADR“ ist dem Speicherplatz 0123H zugeordnet. Der Befehl SJMP REL_ADR steht bei Adresse 0100H. Nach dem Befehl fährt das Programm an der Adresse 0123H fort.
Einzeloperationen:	(PC) ← (PC)+2 (PC) ← (PC) + rel

Codierung des Befehls:

1 0 0 0 0 0 0 0	rel. address

Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	2

SUBB A,<Quellbyte>

subtrahiere mit Borrow (Borgebit C)

Beschreibung:	SUBB subtrahiert das Quellbyte und den Inhalt des Carry-Flags vom Inhalt des Akku. Das Ergebnis steht anschließend in A. SUBB setzt bei einem Unterlauf das C (borrow). Sonst wird es gelöscht. C wird ebenfalls von A subtrahiert, dies ermöglicht Mehrbyte-Rechnungen. OV wird gesetzt, wenn ein Unterlauf von Bit 7, aber nicht von Bit 6 auftritt, oder umgekehrt. Bei der Subtraktion vorzeichenbehafteter Zahlen ist dadurch zu erkennen, ob sich eine negative Zahl ergab, wenn eine negative von einer positiven Zahl abgezogen wurde, oder ob sich eine positive Zahl ergab, wenn eine positive von einer negativen abgezogen wurde (d.h. der Wertebereich der vorzeichenbehafteten Zahlen wurde überschritten). AC wird bei einem Unterlauf von Bit 4 nach Bit 3 gesetzt, sonst gelöscht. Eine Subtraktion einer Konstanten ohne C ist mit ADD A,#data möglich. C ist dann entgegengesetzt dem von SUBB. Vier mögliche Quellbytes stehen zur Verfügung.
Beispiel:	A enthält C9H und R2 enthält 54H, das C ist gesetzt. Nach SUBB A,R2 enthält A 74H, OV ist gesetzt, C und AC gelöscht. Beachte: C9H-54H=75H. Die Differenz zum obigen Ergebnis ist das zuvor gesetzte C (borrow). Ist der Zustand vor der Subtraktion nicht bekannt oder undefiniert, muß C gelöscht werden (CLR C).
Einzeloperationen:	(A) ← (A) - (C) - (Quellbyte)

SUBB A,Rn

Einzeloperationen:	(A) ← (A) - (C) - (Rn)
Codierung des Befehls:	1 0 0 1 1 r r r
Beeinflußte Flags im PSW:	P, C, AC, OV
Bytes:	1
Zyklen:	1

SUBB A,direct

Einzeloperationen:	(A) ← (A) - (C) - (direct)	
Codierung des Befehls:	1 0 0 1 0 1 0 1	direct address
Beeinflußte Flags im PSW:	P, C, AC, OV	
Bytes:	2	
Zyklen:	1	

SUBB A,@Ri

Einzeloperationen:	(A) ← (A) - (C) - ((Ri))
Codierung des Befehls:	1 0 0 1 0 1 1 i
Beeinflußte Flags im PSW:	P, C, AC, OV
Bytes:	1
Zyklen:	1

SUBB A,#data

Einzeloperationen:	(A) ← (A) - (C) - data
Codierung des Befehls:	1 0 0 1 0 1 0 0 | immediate data
Beeinflußte Flags im PSW:	P, C, AC, OV
Bytes:	2
Zyklen:	1

SWAP A

tausche Nibbles

Beschreibung:	SWAP tauscht das höherwertige Nibble mit dem niederwertigen im Akku. Der Befehl kann auch als 4-Bit-Shift-Befehl (Rotation) betrachtet werden.
Beispiel:	In A steht C5H (1100 0101B). Nach dem Befehl SWAP A enthält A 5CH (0101 1100B).
Einzeloperationen:	(A.3-0) ↔ (A.7-4)
Codierung des Befehls:	1 1 0 0 0 1 0 0
Beeinflußte Flags im PSW:	keine
Bytes:	1
Zyklen:	1

XCH A,<Quellbyte>

tausche Akkumulator mit adressiertem Byte (Quellbyte)

Beschreibung:	XCH tauscht den Inhalt des Akku mit dem eines adressierten Bytes.
Beispiel:	R0 enthält 20H, A den Wert 3FH. Die interne RAM-Speicherzelle mit Adresse 20H enthält den Wert 75H. Der Befehl XCH A,@R0 liefert den Wert 3FH in die Speicherzelle und den Wert 75H nach A.
Einzeloperationen:	(A) ↔ (Quellbyte)

XCH A,Rn

Einzeloperationen:	(A) ↔ (Rn)
Codierung des Befehls:	1 1 1 0 0 1 r r r
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	1

XCH A,direct

Einzeloperationen:	(A) ↔ (direct)	
Codierung des Befehls:	1 1 0 0 0 1 0 1	direct address
Beeinflußte Flags im PSW:	P	
Bytes:	2	
Zyklen:	1	

XCH A,@Ri

Einzeloperationen:	(A) ↔ ((Ri))
Codierung des Befehls:	1 1 0 0 1 1 1 i
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	1

XCHD A,@Ri

tauche Low-Nibble

Beschreibung:	XCHD tauscht die niederwertigen vier Bits (Low-Nibble) von A mit denen eines durch die Register R0 oder R1 adressierten Bytes aus. Das höherwertige Nibble bleibt unverändert.
Beispiel:	In R0 steht 20H. In A steht 36H (0011 0110B), in der internen RAM-Zelle 20H steht 75H (0111 0101B). Nach XCHD A,@R0 steht in A 35H (0011 0101B) und in der RAM-Zelle 76H (0111 0110B).
Einzeloperationen:	(A.3-0) ↔ ((Ri)3-0)
Codierung des Befehls:	1 1 0 1 0 1 1 i
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	1

XRL <Zielbyte>,<Quellbyte>

verknüpfe die beiden Bytes „Zielbyte“ und „Quellbyte“ logisch EXKLUSIV-ODER

Beschreibung:	Der Befehl ORL verknüpft Variablen oder Speicherinhalte logisch mit EXKLUSIV-ODER. Die PSW-Flags bleiben unbeeinflußt (außer P, wenn A das Ziel der Operation ist). Der Befehl ist mit 6 Adressierungsarten ausführbar. Ist A der Zieloperand, kann die Quelle ein Registerinhalt, ein indirekt adressierter Speicherinhalt oder eine 8-Bit-Konstante sein. Ist der Zieloperand eine direkte Adresse, kann die Quelle A oder eine 8-Bit-Konstante sein. Ist die Quelle ein Ausgangs-Port, wird immer das Port-Latch, nicht die Pins gelesen.
Beispiel:	A enthält C3H (1100 0011B) und R0 enthält AAH (1010 1010B). Nach dem Befehl XRL A,R0 enthält A 69H (0110 1001B). Ist der Zieloperand ein direkt adressierbares Byte, kann man beliebige Bitkombinationen in einem SFR oder im RAM komplementieren. Der Befehl XRL P1,#00110001B komplmentiert die Bits 0, 4, und 5 in P1.
Einzeloperationen:	(Zielbyte) ← (Zielbyte) ⊕ (Quellbyte)

XRL A,Rn

Einzeloperationen:	(A) ← (A) ⊕ (Rn)
Codierung des Befehls:	0 1 1 0 1 r r r
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	1

XRL A,direct

Einzeloperationen:	(A) ← (A) ⊕ (direct)
Codierung des Befehls:	0 1 1 0 0 1 0 1 | direct address
Beeinflußte Flags im PSW:	P
Bytes:	2
Zyklen:	1

XRL A,@Ri

Einzeloperationen:	(A) ← (A) ⊕ ((Ri))
Codierung des Befehls:	0 1 1 0 0 1 1 i
Beeinflußte Flags im PSW:	P
Bytes:	1
Zyklen:	1

XRL A,#data

Einzeloperationen:	(A) ← (A) ⊕ data
Codierung des Befehls:	0 1 1 0 0 1 0 0 | immediate data
Beeinflußte Flags im PSW:	P
Bytes:	2
Zyklen:	1

XRL direct,A

Einzeloperationen:	(direct) ← (direct) ⊕ (A)
Codierung des Befehls:	0 1 1 0 0 0 1 0 | direct address
Beeinflußte Flags im PSW:	keine
Bytes:	2
Zyklen:	1

XRL direct,#data

Einzeloperationen:	(direct) ← (direct) ⊕ data
Codierung des Befehls:	0 1 1 0 0 0 1 1 | direct address | immediate data
Beeinflußte Flags im PSW:	P, wenn direct=A
Bytes:	3
Zyklen:	2

18 Anhang

18.1 Die Special Function Register in Funktionsgruppen

18.1.1 CPU:

ACC									E0H(*)
B									F0H(*)
PSW	CY	AC	F0	RS1	RS0	OV	F1	P	D0H(*)
SP									81H
DPL									83H
DPH									83H
ADCON	BD	CLK	ADEX	BSY	ADM	MX2	MX1	MX0	D8H(*)

18.1.2 Timer 0, Timer1

TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	88H(*)
TMOD	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0	89H
TL0									8AH
TH0									8CH
TL1									8BH

TH1									8DH

18.1.3 Timer 2 und Capture and Compare

T2CON	T2PS	I3FR		T2R1	T2R0	T2CM	T2I1	T2I0	C8H(*)
TL2									CCH
TH2									CDH
CCEN	COCAH3	COCAL3	COCAH2	COCAL2	COCAH1	COCAL1	COCAH0	COCAL0	C1H
CCL1									C2H
CCH1									C3H
CCL2									C4H
CCH2									C5H
CCl3									C6H
CCH3									C7H
CRCL									CAH
CRCH									CBH
IRCON	EXF2	TF2							C0H(*)

18.1.4 Ports

P0									80H(*)
P1	T2	CLKOUT	T2EX	¬INT2	INT6	INT5	INT4	¬INT3	90H(*)
P2									A0H(*)
P3	¬RD	¬WR	T1	T0	¬INT1	¬INT0	TXD	RXD	B0H(*)
P4									E8H(*)
P5									F8H(*)
P6									DBH

18.1.5 Serielle Schnittstelle

SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	98H(*)
SBUF									99H
PCON	SMOD								87H
ADCON	BD								D8H(*)

18.1.6 A/D-Wandler

ADCON				BSY	ADM	MX2	Mx1	MX0	D8H(*)
ADDAT									D8H

DAPR	$V_{intAREF}$	$V_{intAGND}$	DAH

18.1.7 Interruptsystem

SCON							TI	RI	98H(*)
IEN0	EAL		ET2	ES	ET1	EX1	ET0	EX0	A8H(*)
IEN1	EXEN2		EX6	EX5	EX4	EX3	EX2	EADC	B8H(*)
IP0	-	WDTS	IP0.5	IP0.4	IP0.3	IP0.2	IP0.1	IP0.0	A9H
IP1	-	-	IP1.5	IP1.4	IP1.3	IP1.2	IP1.1	IP1.0	B9H
TCON	TF1		TF0		IE1	IT1	IE0	IT0	88H(*)
T2CON		I3FR	I2FR						C8H(*)
IRCON	EXF2	TF2	IEX6	IEX5	IEX4	IEX3	IEX2	IADC	C0H(*)

18.1.8 Energiesparmodi

PCON	SMOD	PDS	IDLS	-	GF1	GF0	PDE	IDLE	87H

18.1.9 Watchdog-Timer

IEN1		SWDT							B8H(*)
IEN0		WDT							A8H(*)

IP0		WDTS							A9H

18.2 Die Special Function Register

Symbol		Name	Hex-Adresse
P0	*	Port 0	80H
SP		Stack Pointer	81H
DPL		DataPointer, lower Byte	82H
DPH		Data Pointer, higher Byte	83H
PCON		Power Control Register	87H
TCON	*	Timer Control Register	88H
TMOD		Timer Mode Register	89H
TL0		Timer 0, lower Byte	8AH
TL1		Timer 1, lower Byte	8BH
TH0		Timer 0, higher Byte	8CH
TH1		Timer 1, higher Byte	8DH
P1	*	Port 1	90H
SCON	*	Serial Channel Control Register	98H
SBUF		Serial Channel Buffer Register	99H
P2	*	Port 2	0A0H
IEN0	*	Interrupt Enable Register 0	0A8H
IP0		Interrupt Priority Register	0A9H
P3	*	Port 3	0B0H
IEN1	*	Interrupt Enable Register 1	0B8H
IP1		Interrupt Priority Register 1	0B9H
IRCON	*	Interrupt Request Control Register	0C0H
CCEN		Compare/Capture Enable Register	0C1H
CCL1		Compare/Capture Register 1, lower Byte	0C2H
CCH1		Compare/Capture Register 1, higher Byte	0C3H
CCL2		Compare/Capture Register 2, lower Byte	0C4H
CCH2		Compare/Capture Register 2, higher Byte	0C5H
CCL3		Compare/Capture Register 3, lower Byte	0C6H
CCH3		Compare/Capture Register 3, higher Byte	0C7H
T2CON	*	Timer 2 Control Register	0C8H
CRCL		Compare/Relaod/Capture Register, lower Byte	0CAH
CRCH		Compare/Relaod/Capture Register, higher Byte	0CBH
TL2		Timer 2, lower Byte	0CCH
TH2		Timer 2, higher Byte	0CDH
PSW	*	Program Satus Word Register	0D0H
ADCON	*	A/D Converter Control Register	0D8H
ADDAT	*	A/D Converter Data Register	0D9H
DAPR		D/A Converter Program Register	0DAH
P6		Port 6	0DBH[1)]
ACC	*	Accumulator	0E0H
P4	*	Port 4	0E8H
B	*	B Register	0F0H
P5	*	Port 5	0F8H

[1)] Zusatz bei den ACMOS-Typen

Die mit einem Sternchen (*) in der zweiten Spalte gekennzeichneten SFR sind sowohl bit- als auch byteadressierbar

18.3 Default-Werte der Special Function Register

Register	Inhalt	Register	Inhalt
P0 - P5	FFH	SP	07H
DPTR	0000H	PCON	000X 0000B
TCON	00H	TMOD	00H
TL0, TH0	00H	TL1, TH1	00H
TL2, TH2	00H	SCON	00H
IEN0, IEN1	00H	SBUF	XXXX XXXXB
IRCON	00H	IP0	X000 0000B
		IP1	X000 0000B
CCL1, CCH1	00H	CCEN	00H
CCL3, CCH3	00H	CCL2, CCH2	00H
T2CON	00H	CRCL, CRCH	00H
ADCON	00X0 0000B	PSW	00H
DAPR	00H	ADDAT	00H
B	00H	ACC	00H
PC	0000H	Watchdog	0000H

18.4 Alternativfunktionen der Ports P1 und P3

Port	Pin	Alternativfunktion
P1.0	$\overline{INT3}$/CC0	Ext. Interrupt 3 Eingang, Compare 0 Ausgang, Capture 0 Eingang
P1.1	INT4/CC1	Ext. Interrupt 4 Eingang, Compare 1 Ausgang, Capture 1 Eingang
P1.2	INT5/CC2	Ext. Interrupt 5 Eingang, Compare 2 Ausgang, Capture 2 Eingang
P1.2	INT6/CC3	Ext. Interrupt 6 Eingang, Compare 3 Ausgang, Capture 3 Eingang
P1.4	$\overline{INT2}$	Ext. Interrupt 2 Eingang
P1.5	T2EX	Timer 2 external reload trigger input
P1.6	CLKOUT	System clock output
P1.7	T2	Timer 2 external reload trigger input
P3.0	RxD	Serieller Dateneingang (asynchron) oder Datenein/-ausgang (synchron)
P3.1	TxD	Serieller Datenausgang (asynchron) oder Taktausgang (synchron)
P3.2	$\overline{INT0}$	Ext. Interrupt 0 Eingang, Timer 0 Steuereingang
P3.3	$\overline{INT1}$	Ext. Interrupt 1 Eingang, Timer1 Steuereingang
P3.4	T0	Ext. Eingang für Timer 0
P3.5	T1	Ext. Eingang für Timer 1
P3.6	$\overline{WR}$	Bussteuersignal „Lesen“ bei externem Datenspeicher
P3.7	$\overline{RD}$	Bussteuersignal „Schreiben“ bei externem Datenspeicher

18.5 Read-Modify-Write-Befehle

Befehl	Funktion	Beispiel
ANL	Logische UND-Verknüpfung	ANL P1,A
ORL	Logische ODER-Verknüpfung	ORL P4,#0FCH
XRL	Logische EXOR-Verknüpfung	XRL P2,@R1
JBC	Verzweigung, wenn das adressiert Bit gesetzt ist, dann Löschen des Bits	JBC P5.3,LABEL_2
CPL	Komplementieren des adressierten Bits	CPL P1.3
INC	Inkrementieren eines Bytes	INC P4
DEC	Dekrementieren eines Bytes	DEC P4
DJNZ	Dekrementieren und Verzweigen, wenn ≠ 0	DJNZ P0,LABEL_1
MOV Px.y,C	Transfer des Carry-Bits zu Bit y in Port x	MOV P3.5,C
CLR Px.y	Löschen von Bit y in Port x	CLR P0.7
SETB Px.y	Setzen von Bit y in Port x	SETB P2.4