Zusammenfassung MC WS0607

1 Kapitel 2 – Wichtige Grundlagen

1.1 Operanden als Hexadezimal- oder Oktalzahlen

1.1.1 Hexadezimalsystem

Die hexadezimale Schreibeweise eignet sich hervorragend Adressbereiche eines RAMs zu adressieren. Mit wenigen Zeichen kann ein grosser Bereich abgedeckt werden. Die Kennzeichnung erfolget meist entweder durch die Präfixe „$“ und „0x“ oder durch den Buchstaben „H“ als Suffix.

Zum Umformen der Binärzahlen in hexadezimale Zahlen werden jeweils von rechts nach links Nibbles (4er Bit-Gruppe) gebildet. Danach werden die Nibbles in Hex-Werte umgeformt. Bsp.:

1011101b = 101 1101 = 5 D = $5D

Um sich die Pseudotetraden A bis F besser merken zu können, kann folgende Eselsbrücke verwendet werden:

A = 10	D = 13 (**D**reizehn)
B = 11	E = 14
C = 12 (**C**wölf)	F = 15 (**F**ünfzehn)

1.1.2 Zahlenformate ohne und mit Vorzeichen

1.1.2.1 Vorzeichenlose Dualzahlen (Unsigned Binary Numbers)

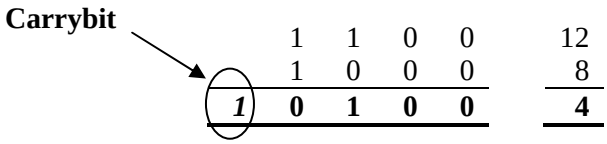
Im Dualsystem stehen nur die Ziffern 0 und 1 zur Verfügung. Elektronische Schaltungen verwenden dieses Zahlensystem, da diese nur die Zustände Strom oder kein Strom unterscheiden können.

Eine beliebige Zahl ohne Vorzeichen kann in einem Dualsystem wie folgt dargestellt werden:

$$Z = a_0 \cdot 2^0 + a_1 \cdot 2^1 + a_2 \cdot 2^2 + \ldots + a_n \cdot 2^n$$

- $23 = \mathbf{1} \cdot 2^4 + \mathbf{0} \cdot 2^3 + \mathbf{1} \cdot 2^2 + \mathbf{1} \cdot 2^1 + \mathbf{1} \cdot 2^0$

Bereichsüberschreitung werden durch Setzen des Übertragsbits (**Carrybit**) im Statusregister signalisiert. In einem solchen Fall entsteht bei einer Addition aus zwei Zahlen ein Ergebnis, welches zwar im innerhalb der Bereichsgrenze liegt, dessen aber der Wert nicht korrekt ist. Bsp.:



1.1.2.2 Dualzahlen mit Vorzeichen (Signed Binary Numbers)

Es gibt drei verschiedene Arten ganze Zahlen mit Vorzeichen darzustellen:

- mit Betrag und Vorzeichen
- im Einerkomplement
- im Zweierkomplement

Das Zweierkomplement ist die am meisten verbreitete Art der Zahlendarstellung von ganzen Zahlen.

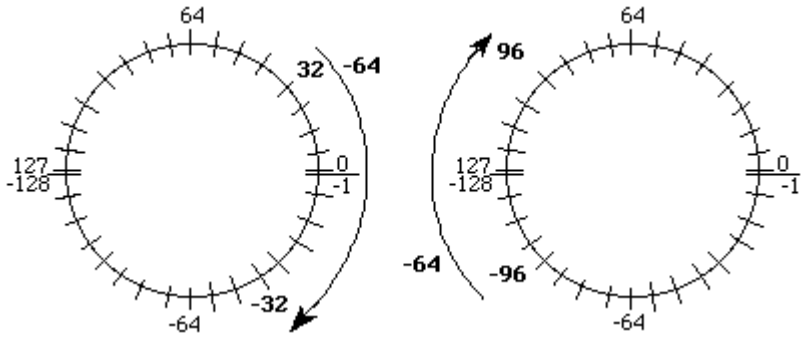
Das Zweierkomplement erhält man, wenn man zuerst das Einerkomplement bildet und 1 dazu addiert. Das Einerkomplement wiederum ist die Bildung des Inversen einer Zahl.

Beispiel einer Rechnung im 2er-Komplement:

19		= 00010011
Inversion		11101100
Addition		+1
2er-Komplement		11101101
23 – 19	=>	00010111
		+ 11101101
		100000100 => 23-19 = 4

Bereichsüberschreitungen können bei ganzen Zahlen mit Vorzeichen in zwei Fällen auftretten:

- Überschreitung des Wertebereichs. Tritt auf, wenn eine Zahl zu gross ist für den zur Verfügung stehenden Wertebereich.
  Folge: ***Carry-Bit*** *wird gesetzt*
- Überschreitung der Vorzeichengrenze: Tritt auf, wenn eine Zahl ein falsches Vorzeichen im MSB, nach einer Addition oder Subtraktion (im 2er-Komplement), signalisiert.
  Folge: ***Overflow-Bit*** *wird gesetzt*



Beachte, dass der Wertebereich einer Zahl halbiert wird, sobald diese als signed (Vorzeichenorientiert) deklariert wird.

Unsigned	2^8 => 0…256	
Signed	2^8 => -128…127	(Das MSB wird für das Vorzeichen verwendet)

1.2 Kodierung der Kommastellen

1.2.1 Codierung von Festkomma-Zahlen

Unter Festkommazahlen werden Zahlen-Ziffernfolgen verstanden, bei welchen die Position des Kommas festgelegt ist und auch dort bleibt.

Da das Komma immer an der gleichen Stelle liegt, ist dies nur ein Problem der Darstellung. Im Rechenwerk selber wird das Komma nicht dargestellt und die Zahl wird mit führenden Nullen aufgefüllt. Bsp.:

1.05 + 2.3 => 00105 + 00230 = 00335 => = 3.35

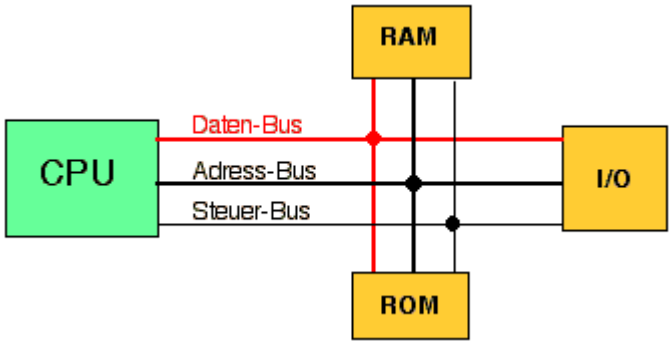
Festkommazahlen sind zwar einfach in der Darstellung, sie können aber bei betragsmässig kleinen Zahlen leicht zu Ungenauigkeiten führen.

1.2.2 Codierung von Gleitkommazahlen

Wird nicht geprüft ☺

1.3 Buskonzepte

Ein Bus besteht aus einer beliebigen Anzahl von parallelen Leitungen, welche mehrere Signalquellen (Sender von binären Wörtern) und mehrere Signalsenken (Empfänger dieser Wörter) miteinander verbindet.
In einem gegebenen Zeitabschnitt darf nur ein Sender Daten auf den Bus geben. Nur betroffene Empfänger sollen auf die Daten reagieren.



Ein Busbetrieb ist gekennzeichnet durch:

- Informationsaustausch über derselben Leitung
- Mehrere Teilnehmer
- Sender-Empfänger-Prinzip

Wichtig dabei sind:

- Kollisionsgefahr berücksichtigen
- Geschwindigkeit, WCTA (Worst Case Timing Analyse)
- Pegel
- Protokolle
- Der Empfänger muss Daten speicher können, damit sie permanent zur Verfügung stehen können

1.3.1 Strategie 1: Hierarchische Struktur

Ein einziger Teilnehmer dominiert den Bus und erlaubt den anderen Teilnehmern, den Bus zu benützen. Diese Methode wird bei Mikrocomputern angewendet, wo das Steuerwerk des Prozessors den Datensender und den Datenempfänger bestimmt.

1.3.2 Strategie 2: Gleichberechtigte Struktur

Jeder Busteilnehmer darf von sich aus Daten auf den Bus geben und den Empfänger der Nachricht bestimmen. Diese Methode braucht jedoch mehr und bessere „Spielregeln“ (Protokolle) und verlangt deshalb relativ „intelligente“ Busteilnehmer.

1.4 Von der Digitaltechnik zum Mikrocomputer

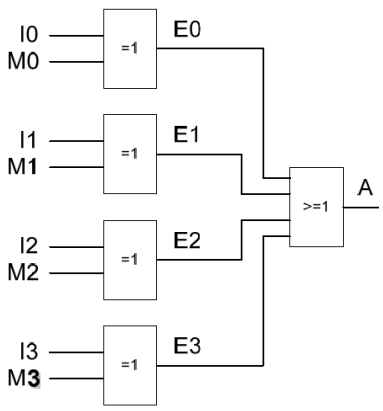
1.4.1 Ein einfacher Mikrocomputer

Nachfolgend wird gezeigt, wie von einer zuerst primitiven Schaltung ein ausgeklügeltes Schaltungssystem entsteht.

1.4.1.1 Aufgabenstellung

Es sollen vier digitale Eingänge (Input I0..I3) mit vier gespeicherten Werten (Memory M0..M3) verglichen werden. Bei Ungleichheit soll das Ausgangssignal A gleich 1, andernfalls gleich 0 sein.

1.4.1.2 Lösung 1: Kombinatorischer Logik

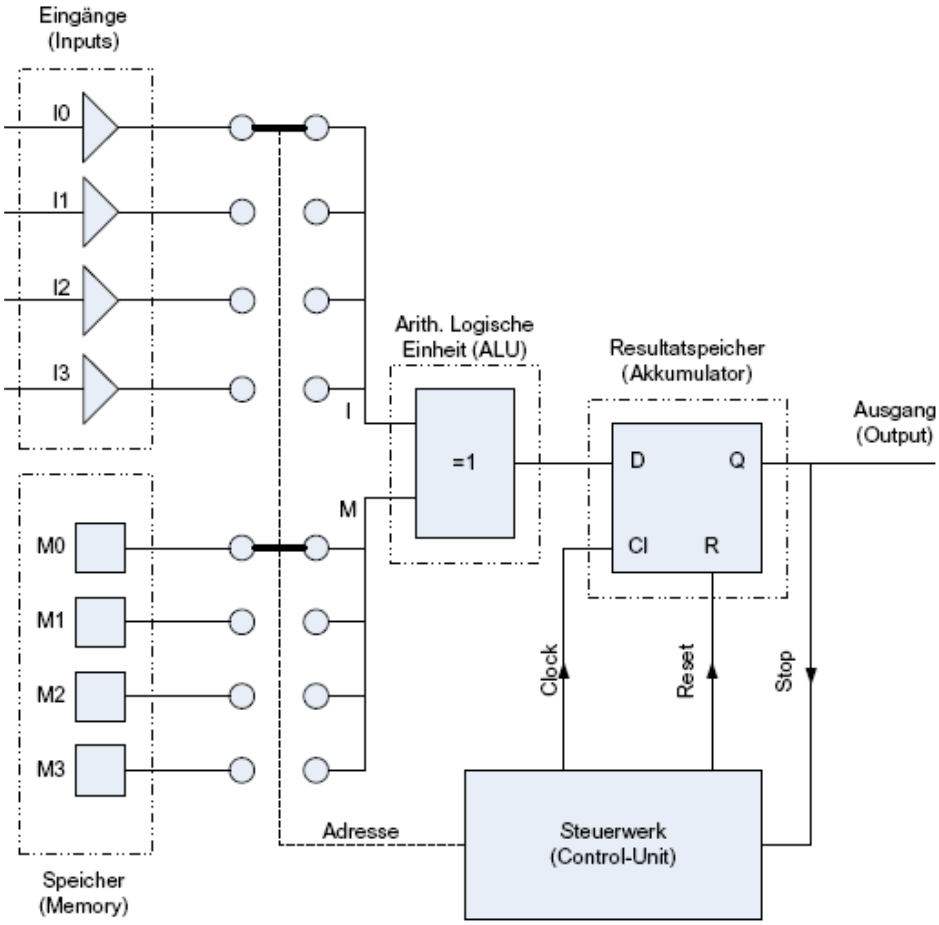


Die Eingänge und die Vergleichswerte werden mit „Exklusiv Oder-Gattern“ (EXOR) paarweise verglichen.

Die Vier Ausgänge der Vergleichsstufen werden mit einem „Oder-Gatter“ (OR) zusammengefasst.

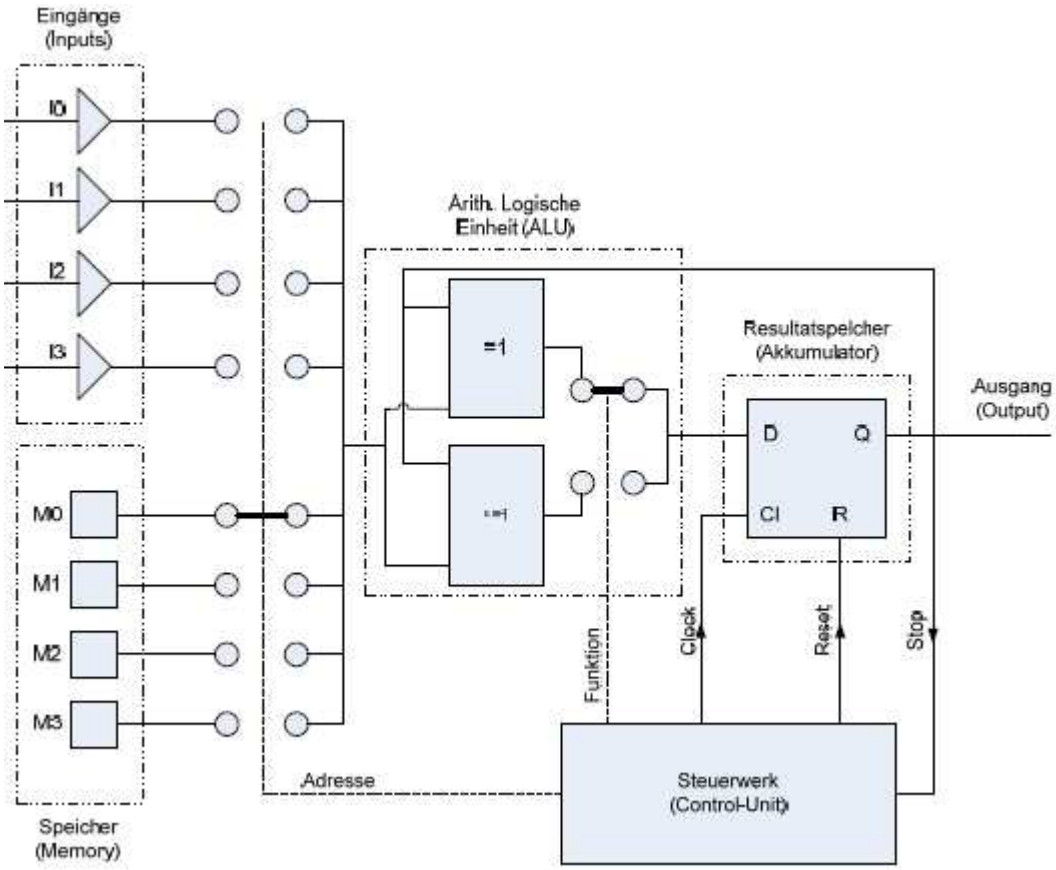
1.4.1.3 Lösung 2: Programmsteuerung

- Die Verarbeitung der Daten I und M erfolgt schrittweise. Zur Ablaufsteuerung sind bestimmte Programmschritte notwendig.
- Die Schalterstellung (Adresse) wird vom Steuerwerk (Control Unit) bestimmt.
- Der Vergleichswert ist im Speicher (Memory) abgelegt.
- Die Arithmetisch-Logische Einheit (ALU), in unserem Fall das EXOR-Gatter, verarbeitet die Daten.
- Das Ergebnis der Verarbeitung wird im Resultatspeicher (Akkumulator) abgespeichert.



1.4.1.4 Lösung 3: Die Einadress-Schaltung

In dieser Lösung sind die zu vergleichenden Daten nicht mehr durch zwei gekoppelte Schalter voneinander abhängig. Der Eingang der ALU wird mit dem Ausgang des Akkumulators verbunden. Die ALU wird gleichzeitig um eine OR-Funktion erweitert.



Ein Verarbeitungsbefehl besteht aus einer Anweisung an die ALU (OR, EXOR) und einer Anweisung an den Schalter (Adresse).

Ein Befehl enthält die Informationen:

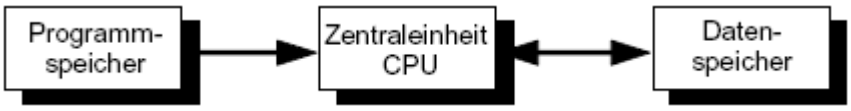
Operation: Art der Verarbeitung
Operand: Adresse der zu verarbeitenden Daten

In einem weiteren Beispiel wird die ALU noch um die Gatter AND und NOT erweitert. Aber dieses soll nicht weiter verfolgt werden.

1.4.2 Erweiterung des Verständnisses mit der Von Neumann Architektur

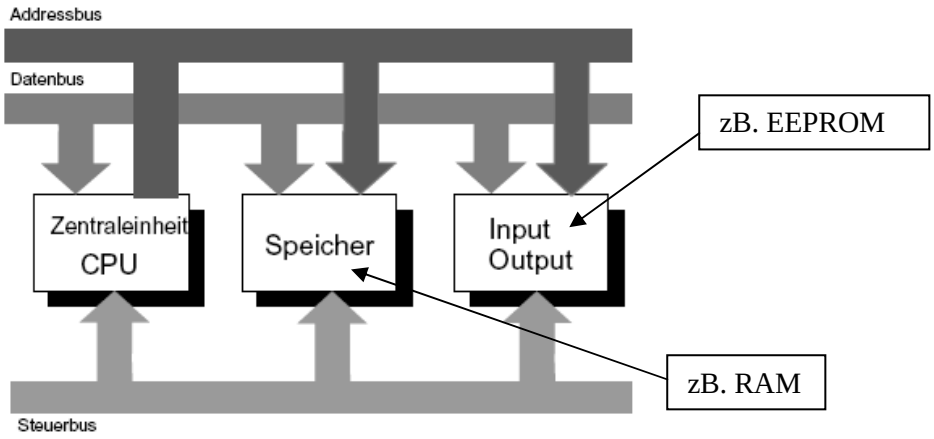
1.4.2.1 Die Harvard Architektur

Die Harvard-Architektur besitzt eine strikte Trennung zwischen Programm- und Datenspeicher. Die beiden Speicher werden über getrennte Buse mit der CPU verbunden. Sie ist älter als die Von Neumann Architektur.



1.4.2.2 Die Von Neumann Architektur

Bei der Von Neumann Architektur sind die Programme und Daten im gleichen Speicher untergebracht. Die CPU holt Daten und Programminstruktionen über den gleichen Bus. **Die Von Neumann Architektur besteht aus CPU, Speicher und I/O-Teil.**



Die CPU bilden das Kernstück. Sie steuert den zeitlichen Ablauf des gesamten Systems sowie den Bus und führt die Operationen aus.

Die Verbindung zur Aussenwelt (Tastatur, Bildschirm) wird durch Ein- und Ausgabeeinheiten (I/O) hergestellt.

Es wird zwischen drei Bussysteme unterschieden:

- **Datenbus:** Verbindungsleitung für Daten. Als Daten werden Befehle, Operanden und Resultate verstanden
- **Adressbus:** Verbindungsleitung für Adressen. Adressen dienen zur Auswahl der Speicherzellen im Memory oder einer bestimmten Ein- und Ausgabeeinheit.
- **Steuerbus:** Verbindungsleitung für Steuersignale. Signale zum Informationsaustausch zwischen den Komponenten, zum Beispiel Angabe der Datenrichtung.

1.5 Antworten zu den Kurzfragen

1. Geben Sie in eigenen Worten Definitionen für „Wort“ und „MSB“ an.

   Wort = 2 Bytes
   MSB (Most significed Bit) = 1 Bit im Wort

2. Welche Vorteile haben Zweierkomplementzahlen bei der Darstellung von negativen Zahlen?

   Das Vorzeichen wird durch das erste Bit definiert. Vorteil: Subtraktion durch Addition!

3. Was zeigen die beiden Flags „Negativ“ und „Overflow“ an?

   Negativ: Es handelt sich hierbei um eine negative (, signed) Zahl im Zweierkomplement.
   Overflow: Eine Bereichsüberschreitung vom positiven zum negativen (oder umgekehrt) Zahlenbereich fand statt.
   Bsp. Für eine 8 Bit lange Zahl liegt der Wertebereich zwischen -128 bis 127. Wird nun 126 + 4 gerechnet müsste das 130 (10000010_B) ergeben. Da es sich aber hierbei um eine signed Zahl handelt und das erste Bit 1 ist, wird die Zahl als -2 interpretiert.

4. Wieso wird in MC mit binären Zahlen gerechnet?

   Da der MC nur die Zustände, *Strom (1)* oder *kein Strom (0)* kennt.

5. Welche Zahlensysteme werden in der MC Technik neben dem Dezimalsystem sonst noch verwendet?

   Dezimal, Oktal (eher veraltet), Hexadezimal, Binär.

6. Welche Begriffe für Gruppen von Bits sind Ihnen bekannt?

   NIBBLES: Eine 4er Bit-Gruppe, die gebildet wird um Binärzahle in Hex-Werte umzuformen.
   BYTE: 1 Byte entspricht 8 Bits.
   WORT: Länge eines Befehls, der von einer CPU verarbeitet werden kann. (beim HCS08 sind es 8 Bit, also 1 Byte)

7. Eignen Sie sich Routine in der Formatwandlung von Zahlen an. z.B.: welcher Dezimalzahl entsprechen 1000H und FFFFH?

   $$\underline{\underline{1000H}} = 16^3 = \underline{\underline{4096}}$$
   $$\underline{\underline{FFFFH}} = (15 \cdot 16^3) + (15 \cdot 16^2) + (15 \cdot 16^1) + (15 \cdot 16^0) = 61440 + 3840 + 240 + 15 = \underline{\underline{65535}}$$

8. Wie alt ist der erste Mikroprozessor; welcher Typ?

1972, also 34 Jahre alt. Typ 4004.

9. Was ist in der Mikrocomputertechnik eine Operation, was ein Operand?

   Operation: Verarbeitungsart
   Operand: Adresse der zu verarbeitenden Daten

10. Welche 3 Hauptteile machen die Von Neumann Architektur aus?

    CPU, Speicher (RAM), I/O-Teil

11. Zählen Sie Komponenten eines Computersystems auf, welche Sie bis jetzt kennen gelernt haben.

    A/D-Wandler, CPU, RAM, I/O-Teil, Datenbaus, Adressbus, Steuerbus

12. Welche Bussysteme können bei Mikrocontrollern unterschieden werden?

    Datenbus, Adressbus, Steuerbus

13. Welche IC-Ausgangsschaltungen werden punkto Busfähigkeit unterschieden?

    Uni- und bidirektional

2 Kapitel 5 – Grundlegende Programmiertechniken

2.1 Datentransfer

Datentransfer werden in 3 verschiedene Arten unterteilt.

Lade ins Register	Register speichern	Transfer
Load: LDA LDX, LDHX	Store: STA, STX	CPU-Register: TAP TAX TSX
Clear: CLRA (0 laden)	Push: PSHX (im Stack ablegen)	Move: MOV
Pull: PULA (vom Stack holen)		

2.2 Arithmetische Programmschnipsel

Nachfolgend einige Hinweise zur elementaren Arithmetik

2.2.1 8-Bit Addition

```
LDA    ADR1  ;Lade OP1 in den Akkumulator
ADD    ADR2  ;Addiere OP2 zu OP1, Resultat in den Akku A
STA    DR3   ;Speichere das Ergebnis im Akku A nach ADR3 ab
```

beachte Carry und Overflow-Flags

2.2.2 16-Bit Addition

```
LDA  ADR1     ;Lade das Low Byte von OP1 nach A
ADD  ADR2     ;Addiere das Low Byte von OP2 hinzu
STA  ADR3     ;Speichere Ergebnis (Low Byte) auf ADR3 ab
LDA  ADR1-1   ;Lade das High Byte von OP1 nach A
ADC  ADR2-1   ;High Byte addieren: (OP1 + OP2) + ev.Übertragsbit
STA  ADR3-1   ;Ergebnis unter ADR3-1 abspeichern
```

Im 5. Befehl wird ADC an der Stelle von ADD verwendet, da das Carry-Bit aus der ersten Addition beachtet werden muss.

2.2.3 Subtraktion

Die Subtraktion gestaltet sich analog der Addition (ADD->SUB) jedoch müssen die Flags anders interpretiert werden.
Bei der Subtraktion wird das Carry-Bit auch gesetzt, nur hat es hier die Bedeutung des "Borrow" (Borger). Es weist auf das Überschreiten der Grenze bei 0 hin.
Wird bei der Subtraktion im Zweierkomplement-Format gerechnet, dann ist das Auftreten des Borrows (Carry) bedeutungslos. Hingegen deutet das Borrow (Carry) im vorzeichenlosen Format auf einen ungewollten Vorzeichenwechsel hin!
Das Overflow-Bit wird genau dann gesetzt, wenn eine Bereichsüberschreitung an der Vorzeichengrenze bei den grössten Beträgen (bei Bytes: $7F □ $80) stattfindet. Im 2er-Komplement Format bedeutet das einen Fehler und sollte durch Programmierung speziell behandelt werden.

2.2.4 Multiplikation und Division

Es ist zu beachten, das bei der Multiplikation das Resultat 2Byte gross ist und das die zu multiplizierendenen Zahlen als Vorzeichenlos betrachtet werden.

Der HCS08 verfügt über einen Multiplikationsbefehl ("MUL"). Dieser Befehl multipliziert den Inhalt des Akkumulators A mit dem Inhalt des Indexregisters X und speichert das 16 Bit Resultat im Indexregister X:A (MSB in X, LSB in A) ab. Die Register X und A sind dabei nicht anderweitig verwendbar!

Die Division ist analog, jedoch ist hier der Dividend 2 Byte lang. (auch vorzeichenlos)

2.2.5 BCD-Arithmetik

BCD-Notation: ein 4-Bit-Nibble zur Darstellung einer Dezimalziffer (0...9) verwendet. Die Pseudotetrade (A....F Hex) wird nicht benötigt. Somit lassen sich in einem Byte 2 Dezimalziffern, also 100 verschiedene Zahlen darstellen.
Der Überlauf in die nächste Dekade erfordert eine spezielle Korrektur, da ja normalerweise nach der Zahl 9 die Hexzahlen A, B, etc. folgen. Der HCS08 stellt mit dem Befehl DAA diese BCD-Korrekturoperation zur Verfügung.

Beispiel Es sollen die Dezimalzahlen 08 und 03 addiert werden. Das duale Ergebnis 00001011 ist im BCD-Format nicht korrekt! 1011 entspricht der Zahl 11d. 1011 ist in der BCD-Notation ein nicht erlaubtes Codewort. Das korrekte Ergebnis müsste 0001'0001 lauten. Für diese BCD-Korrektur stellt der HCS08 den Ausgleichsbefehl DAA (Decimal Adjust Akku A) zur Verfügung, welcher auch die H und C-Flags beachtet.

! Merke:
Damit bekannt ist, ob ein Überlauf in der "Mitte" zwischen dem 3. und 4. Bit, also zwischen der tieferen und höheren Dezimalziffer, stattgefunden hat, wird beim HCS08 das H (:=Halfcarry)-Bit des Statusregisters gesetzt.

2.2.6 Logische Operatoren und Bit-Manipulationen

Es gibt Befehle um Bits des Statusregisters zu manipulieren. CLC, CLI, SEC, SEI (Lösche C,I/ Setze C,I).
Zum Ändern einzelner Bytes lassen sich jedoch auch Bit-Masken mit logischen Operatoren verwenden.
Das Setzen von Bits lässt sich z.B. mit dem Befehl ORA bewerkstelligen:

```
ORA  #$40     ;Setze das 7. Bit im Akku A
```

Das Löschen von Bits lässt sich z.B. mit dem Befehl AND bewerkstelligen:

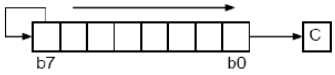
```
AND  #$FD     ;Lösche das 2. Bit im Akku A
```

Die Hex-Werte $40 bzw. $FD werden in dieser Verwendung "Bit-Masken" genannt!

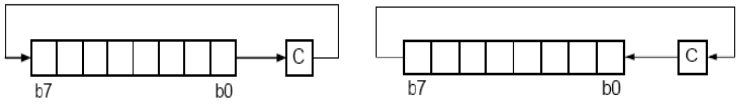
2.2.7 Schiebe und Rotationsbefehle

Das herausfallende Bit wird im Carry-Flag gespeichert

	Schieberichtung

Anwendung	nach links (in Richtung MSB)	nach rechts (in Richtung LSB)
Bitweise Abarbeitung	Logical Shift Left (**LFL)** LSB = 0	Logical Shift Right (**LSR)** MSB=0
Arithmetische Operation	Arithmetic Shift Left (**ASL**): LSB = 0 (Multiplikation mal 2)	Arithmetic Shift Right (**ASR**): MSB behält seinen ursprünglichen Wert (Division durch 2), damit das Vorzeichen erhalten bleibt 

Im Gegensatz zum Schieben wird bei der Rotation (ROR, ROL) den Carry-Wert „hineingeschoben“



2.2.8 Programmverzweigungen

Es gibt mehrere Arten von Programmverzweigungen:

1. Aufgrund eines Statusbits
   z.B. BCC Branch if Carry Bit Clear
2. Es werden bestimmte Bits im in Datenbytes abgefragt
   BRCLR Branch if Bit n in Memory Clear (Nur DIR)
   BRSET Branch if Bit n in Memory SET (nur DIR)
3. Zahlenwerte werden verglichen
   CMP Compare Accumulator with Memory
   CPX Compare X (Index Register Low) with Memory

Sprungbefehle nach CMP, CPX, CPHX, SUB etc.

Befehl	Abfrage	Format
BGT BHI	> >	signed unsigned
BGE BHS, BCC	>= >=	signed unsigned
BLE BLS	<= <=	signed unsigned
BLT BLO, BCS	< <	signed unsigned
BEQ	=	signed und unsigned

Einzelne CCR-Bit-Abfragen

Befehl	Bit	Bedeutung	Format
BEQ	Z=1	Abfrage Z-Bit: Equal	simple
BNE	Z=0	Abfrage Z-Bit: Not Equal	simple
BCS	C=1	Abfrage C-Bit: Equal	simple
BCC	C=0	Abfrage C-Bit: Not Equal	simple
BMI	N=1	Abfrage N-Bit: Minus	simple
BPL	N=0	Abfrage N-Bit: Plus	simple

BRA	Always	unconditional
BRN	Never	unconditional
BSR	To Subroutine	unconditional

Unbedingte Sprungbefehle

Befehl	Adressierung	Bedeutung
JMP	DIR, EXT, IX2, IX1, IX	GOTO
JSR	DIR, EXT, IX2, IX1, IX	Subroutinenaufruf
RTS	inherent	Rücksprungbefehl aus Subroutine
RTI	inherent	Rücksprung von Interrupt-Service-Routine

Absolute Sprungbefehle

2.3 Arbeiten mit Tabellen

Das abarbeiten von Tabellen lässt sich mit der indexierten Addressierung stark vereinfachen.

2.4 Geräteetreiber-Programmierung

2.4.1 Callback

Die Software registriert sich beim Treiber mit einer Funktion. Der Treiber ruft dann, sobald das Ereignis auftritt, die übergebene Funktion auf. (Vgl. Java Observer)

2.4.2 Interrupts

Bei einem Ereignis wird eine ISR (Interrupt SubRoutine) aufgerufen oder eine Callback-Methode

2.4.3 Polling

Es gibt die Blockierende Abfrage, welche in einer Schlaufe wartet, bis sich etwas ändert, oder die nicht blockierende Anfrage, welcche bei jedem Durchgang überprüft, ob sich etwas geändert hat.

3 Kapitel 6 – Weitere Funktionen eines Mikrocomputers

3.1 Behandeltete Themen

- Subroutinen
- Stack
- Timer
- Interrupts
- Test & Measurement
- A/D Wandler
- IIC-Bus

3.2 Subroutinen

Unterprogramme sind in sich abgeschlossene Befehlsfolgen, die wiederholt aufgerufen und ausgeführt werden können. Nach Abarbeitung der Subroutine wird das übergeordnete Programm hinter der Aufrufstelle fortgesetzt. Das Unterprogramm kann als eigenständiges Sourcefile gehandelt werden.

3.2.1 Vorteile

- Sich oft wiederholende Befehlsfolgen brauchen nur einmal programmiert und gespeichert zu werden. Der Programmspeicherbedarf wird dadurch kleiner.
- Programme lassen sich modular aufbauen; sie werden dadurch übersichtlicher, sind leichter zu testen und besser zu dokumentieren → Programmieraufwand und Fehlerhäufigkeit wird verringert.
- Grössere Programme können in Unterprogramme zerlegt und von **mehreren Personen** gleichzeitig geschrieben werden → Teamarbeit

3.2.2 Nachteile

- Für den Unterprogrammaufruf und den Rücksprung wird Rechenzeit benötigt.

3.2.3 Unterprogrammaufruf

Ein Unterprogrammaufruf erfolgt mit JSR (absolute 16-bit Adressen) oder BSR (relative 8-bit Offset-Adressen).
Bei JSR kann das Unterprogramm im ganzen Speicherbereich 64k liegen. Bei BSR hingegen liegt es in einer relativen Distanz von -128 bis +127 Byte (Zweierkomplement) ausgehend von der aktuellen Position im Programm Counter (PC).

```
JSR    $2234      ;absolute Adresse
BSR    $56        ;relative Adresse
```

Die Sprungadresse zeigt auf den ersten auszuführenden Befehl des Unterprogramms.

JSR oder BSR schreibt die aktuelle Position des PC als Rücksprungadresse in den Stack und überschreibt den PC mit der auf JSR oder BSR stehenden Unterpogrammadresse.
Der Rücksprung ins Oberprogramm erfolgt mit dem Befehl RTS, der das Unterprogramm abschliesst. RTS liest den **letzten** Stackeintrag, also die Rücksprungadresse und lädt ihn in den PC.

3.3 Stack (Stapelspeicherorganisation)

Der Stack dient als Zwischenspeicher und ist Teil des Datenspeichers. Er wird vor allem beim Aufruf von Subroutinen und Interrupt-Serviceroutinen (ISR) benötigt um,

- wichtige Informationen des Oberprogramms (wie Register, …) zwischenzuspeichern, damit bei der Weiterführung desselben, darauf zurückgegriffen werden kann,
- die Parameterübergabe zwischen Ober- und Unterprogramm zu bewerkstelligen und
- allgemein Daten vorübergehend abzuspeichern.

Der Stack hat eine LIFO-Struktur (Last In, First-Out). Der Stack hat somit eine chronologische Struktur von oben nach unten bezogen auf die Adressgrösse. Das erste auf dem Stack abgelegte Datenwort bildet immer das obere Ende des Stacks (grösste Adresse).
Der PUSH-Befehl legt ein Element zuoberst auf den Stack. PULL hingegen entfernt das zuletzt eingefügte Element aus dem Stack.
Die Verwaltung des Stacks erfolgt mit dem Stackpointer (SP). Der Stackpointer enthält die Adresse des letzten Elements im Stack.
PUSH und PULL greifen über den SP auf das Element zu und setzen den SP auf die neue aktuelle Adresse. Es ist darauf zu achten, dass RTS denjenigen SP vorfindet, der unmittelbar nach dem Unterprogrammaufruf vorlag; andernfalls erfolgt ein unkontrollierter Rücksprung.
Bei komplexeren Aufgaben sollte man den Stackbedarf der einzelnen Subroutinen und allfälligen ISRs wissen um zu ermitteln welche Grösse für den Stack reserviert werden soll. Es ist zu achten, dass ein geschachtelter Aufruf von mehreren Subroutinen den Stack zum Überlaufen bringen könnte.

3.3.1 Der Stack beim HCS08

Beim HCS08 kann ein Stack mithilfe des SP-Registers realisiert werden. Am Anfang des Programms muss der SP mit der Anfangsadresse des Stacks geladen werden. Diese Adresse ist die höchste im Stack, da der HCS08 bei einer Schreiboperation (PUSH) den SP dekrementiert. Der Beginn des Stacks wird mit durch Laden der Startadresse ins SP-Register festgelegt.

```
LDHX   $4001      ;Adresse ins H:X Register laden
TXS               ;Stackpointer mit Adresse laden
```

Der Bereich des Stacks sollte vorhin jedoch in der Linkerdatei (*.prm) definiert werden. Siehe dazu im Skript auf Seite 6-6.

3.3.2 Retten des Status beim Unterprogrammaufruf (Context-Save)

Wenn Unter- und Oberprogramm die Register unabhängig benutzen, so muss beim Unterprogrammaufruf der Prozessorstatus gerettet und bei Rückkehr zum Oberprogramm wieder geladen werden. Zum Prozessorstatus gehören nebst dem PC die Registerinhalte und evtl. auch das Statusregister.

Das Retten des PC erfolgt bei den Befehlen JSR, BSR und RTS automatisch, für die übrigen Register muss dies explizit programmiert werden. Folgende Befehle werden zum Retten und Laden der Register verwendet:

PSHA, PSHH, PSHX und PULX, PULH, PULA

Auf die Reihenfolge beim Speichern und Laden der Register muss geachtet werden (LIFO).
Das Retten des Statusregisters CCR ist nur mit dem Befehl TPA (Transfer Statusregister zum Akkumulator) und erneutem Retten des Akkumulators möglich.

Als Empfehlung sollte am Unterprogrammanfang immer die von ihm benutzten Register gerettet werden. Es brauchen nur die Register gerettet werden, die auch vom Unterprogramm verwendet werden.

3.3.3 Berechnung der Stackgrösse

Der Stackbedarf einer Anwendung ist die Summe des Stackbedarfs von allen verschachtelten Unterprogrammen, welche im worst-case in einer Kette zur Laufzeit auftreten könnten. Deshalb ist es von Vorteil, wenn der Stackbedarf jedes Unterprogramms dokumentiert wird. Auch bei Rekursion muss zudem die maximale Anzahl von Selbstaufrufen bekannt bzw. begrenzt sein. Auch welche Prozesse sich wo unterbrechen können (bsp. Interrupts) soll analysiert werden.
Mit einer Grafik kann der Stackbedarf zur Laufzeit visualisiert werden:

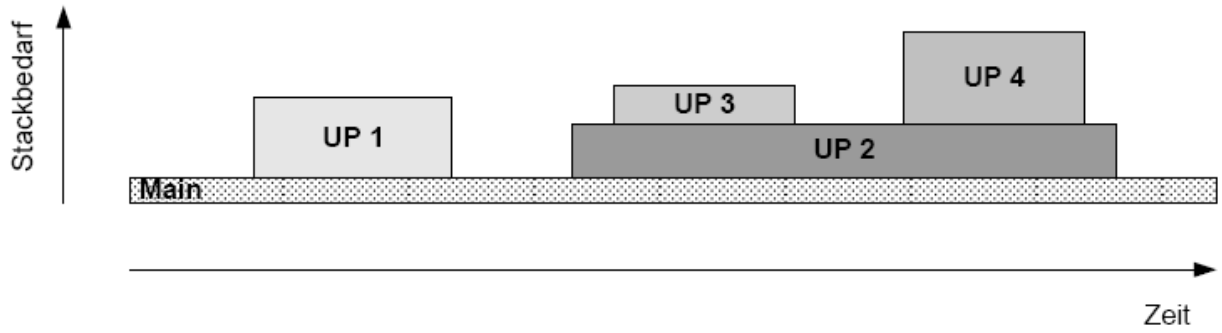


Abbildung 1: Ermittlung des Stackbedarfs

3.4 Parameterübergabe

Parameter sind Daten oder Adressen, die dem Unterprogramm Änderungen gestatten. Die Übergabe der Daten vom aufrufenden Programm zum Unterprogramm nennt sich **Parameterübergabe**.

3.4.1 Art der Parameterübergabe

- **call by value**: Nur der **Wert** wird als Parameter in den Datenbereich des Unterprogramms kopiert. Der ursprüngliche Wert kann im Unterprogramm jedoch **nicht** verändert werden.
- **call by reference**: Die **Adresse** zu einem Wert wird als Parameter dem Unterprogramm übergeben. Der Wert selbst verbleibt im Oberprogramm und wird nicht kopiert. Der Zugriff auf den Operanden durch die Subroutine erfolgt indirekt über die Adresse. Somit kann der ursprüngliche Wert **verändert** werden.

Üblicherweise wird die Wertübergabe nur für Einzelwerte benutzt; für die Übergabe von Feldern wird die Adressübergabe wegen der Speicherplatzersparnis bevorzugt.

Parameter werden auch anhand ihrer Funktion unterteilt. Es gibt:

- Eingangsparameter: Werden dem Unterprogramm übergeben. Die Werte werden dort benutzt können aber **nicht** verändert werden.
- Ausgangsparameter: Ihnen werden erst im Unterprogramm Werte zugewiesen und können erst danach (im Oberprogramm) benutzt werden.
- Übergangsparameter: Werden vom Unterprogramm zuerst benutzt und können danach verändert werden.

3.4.2 Ort der Parameterübergabe

3.4.2.1 Übergabe in Registern (call by value)

Für die Übergabe von wenigern Parametern bieten sich die CPU-Register. Diese Art hat den geringsten Organisationsaufwand für die Vorbereitung und Durchführung der Parameterübergabe.

```
LDA    #$43  ;43h in Akkumulator laden
STA    PTAD  ;Wert vom Akku auf den Port A ausgeben
```

3.4.2.2 Übergabe in festen Speicherzellen (call by value)

Die Parameter liegen hier auf festen Speicherstellen, deren Adressen dem Unterprogramm bekannt sind. Die Adressen werden bei der Programmerstellung festgelegt.

```
DATA_CHAR   DS    1     ;Speicherplatz reservieren
LDA         #$43        ;43h in Akku laden
STA         DATA_CHAR   ;Inhalt von Akku in Speicherplatz laden
```

3.4.2.3 Adressübergabe von Datenbereichen (call by reference)

Hier fasst man die Parameter im Datenbereich des Oberprogramms zu einem Parameterfeld zusammen und übergibt dessen Anfangsadresse dem Unterprogramm. Der Parameterzugriff erfolgt über diese Anfangsadresse. Meistens wird die indexierte Adressierung verwendet, d.h. dass die Adresse aus der Summe des Indexregisters und des angegebenen Offsets gebildet wird.

```
LDHX   #$80        ;80h in Register laden
LDA    #$43        ;43h in Akku laden
STA    3,X         ;Wert von Akku in Adresse 83h laden
```

3.4.2.4 Übergabe mit dem Stack (call by value / reference)

Das Oberprogramm kann die Parameter im Stack platzieren, von wo sie das Unterprogramm zurückgewinnen kann.

```
LDA    $80        ;Wert von Adresse 80h in Akku laden
PSHA              ;Akku retten
JSR    method     ;Sprung auf Subroutine
PULA              ;Akku wiederherstellen
```

3.4.2.5 Übergabe im Hauptprogrammierbereich (call by value / reference)

Die Parameter im Programmierbereich des Oberprogramms werden zu einem Parameterfeld zusammengefasst, das unmittelbar hinter dem Subroutinenaufruf steht. Der Parameterzugriff erfolgt über die im Stack abgespeicherte Rücksprungadresse.

3.5 Verschachtelung von Subroutinen

Ein Unterprogramm kann seinerseits wieder Unterprogramme aufrufen und bekommt die Rolle eines Oberprogramms. Der Aufrufmechanismus kann sich über mehrere Unterprogrammstufen erstrecken. Dann handelt es sich um eine Schachtelung.
Folgende Arten von geschachtelten Unterprogrammen existieren:

3.5.1 Einfache Unterprogramme

Verschiedene Unterprogramme rufen sich, ausgehend vom Hauptprogramm **nacheinander** auf. Statusretten und Parameterübergabe erfolgen in der gewohnten Weise.

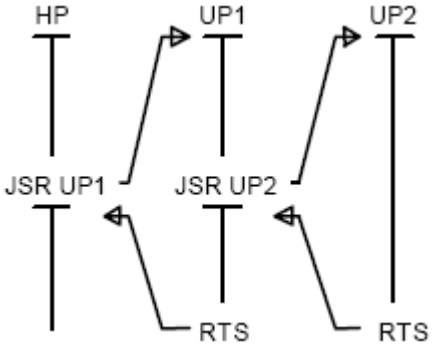


Abbildung 2: Sequentieller Aufruf

3.5.2 Reentrante Unterprogramme

Reentrante (wiedereintrittsfreie) Unterprogramme können von unterschiedlichen Unterbrechungsprogrammen (Interrupts) unterbrochen und gleichzeitig in ihnen erneut aufgerufen werden. Die Zeitpunkte der Unterbrechung sind nicht genau bestimmbar, da sie von Unterbrechungsbedingungen abhängen.
Dadurch sorgt das Unterprogramm selbst für das Retten und Laden der Unterprogrammdaten des vorangegangenen Unterprogrammablaufs.

3.5.3 Rekursive Unterprogramme

Ein rekursiver Aufruf entsteht dann, wenn ein Unterprogramm sich selbst aufruft. Der Aufruf ist *direkt*, wenn das Unterprogramm sich selbst aufruft oder *indirekt*, wenn der Wiederaufruf über ein oder mehrere Unterprogramm erfolgt.
Bei rekursiven Unterprogrammen muss gesorgt werden, dass bei jedem Aufruf ein neuer Datenbereich für das Unterprogramm bereitgestellt wird, damit der zuletzt aktuelle Datenbereich nicht überschrieben wird.

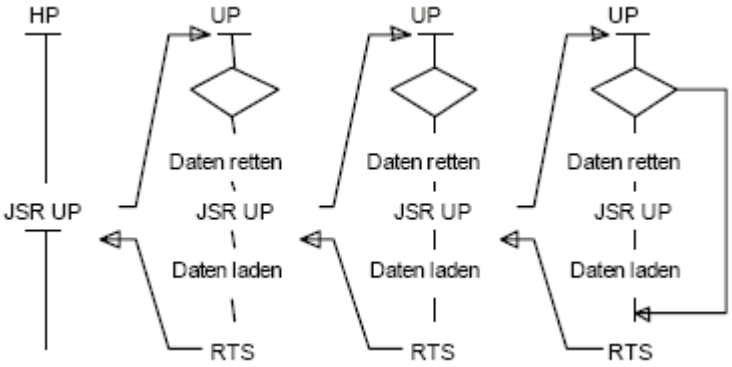


Abbildung 3: Schachtelung durch Rekursion

3.6 Interrupts

Interrupts sind Unterbrechungen sog. Ausnahmebehandlungen im Programmablauf, welche durch bestimmte Ereignisse ausgelöst werden können. Interrupts unterbrechen den normalen Programmablauf und verzweigen unmittelbar auf das Ereignis in ein dafür vorgesehenes Unterprogramm, die sog. Interrupt-Serviceroutine (ISR).

3.6.1 Anwendungsbeispiel

Steigt die Temperatur eines Kernkraftwerks über einen Grenzwert, muss sofort darauf reagiert werden. Es wäre nicht sinnvoll wenn zuerst eine längere Programmsequenz abgearbeitet werden müsste, bis das System die hohe Temperatur wahrnimmt. Das Detektieren der zu hohen Temperatur muss sofort etwas bewirken, also einen Interrupt auslösen, damit in der sofort abgearbeiteten ISR eine Gegenmassnahme wie bspw. eine Notabschaltung eingeleitet werden kann.

3.6.2 Eigenschaften von Interrupts

- Auf solche Ereignisse muss immer zwingend eine „sofort“ eine bestimmte Handlung (Reaktion) vorgenommen werden.
- Die Ereignisse kommen meist durch das Zusammenspiel von der Computer-Peripherie, sowie von der Prozessorperipherie (A/D Wandler, Sensoren etc.)
- Die ISR wird völlig **unvorhersehbar, spontan** aufgerufen. Es muss also immer damit gerechnet werden, dass ein Interrupt auftritt.

3.6.3 Polling versus Interrupt

- Die Art und Weise auf ein Ereignis zu reagieren kann grundsätzlich durch Interrupts (Ausnahmebehandlung) oder durch Polling (zyklisches Abfragen) erfolgen.
- Polling kommt dann zum Zuge, wenn nicht höchste Echtzeitanforderungen vorliegen, d.h. das System nicht schnellstmöglich reagieren muss.
- Polling hat gegenüber Interrupts den Vorteil, dass der Zeitpunkt der Abfrage nachvollziehbar ist. Wo hohe Prioritäten verlangt sind, müssen Interrupts verwendet werden.
- Tritt das Ereignis nur sehr selten auf, verschwendet Polling zuviel Rechenkapazität mit unnötigen Abfragen.
- Interrupts benötigen gegenüber Polling für die Programmunterbrechung mit Rücksprung mehr Rechenzeit, da der Status gerettet und geladen werden muss.

3.6.4 Prinzipielle Funktionsweise

3.6.4.1 Interrupt-Festlegung & Aufruf der ISR

Am Ende von jedem Befehlszyklus überprüft der Mikrocontroller die Interruptleitung, ob eine Anforderung für einen Interrupt vorliegt. Ist dies der Fall, so wird geprüft ob dieser Interrupt angenommen werden kann. Ist das Interrupt-Signal inaktiv, so wird der folgende Befehl abgearbeitet. Ansonsten wird die aktuelle Adresse des Program Counters in den Stack gespeichert und die Adresse auf die der Interruptvektor zeigt (Pointer auf ISR) in den PC geladen und der Programmablauf wird an dieser Adresse fortgesetzt. Ist die Abarbeitung der ISR abgeschlossen erfolgt der Rücksprung im Gegensatz zu den Subroutinen mit `RTI`.

3.6.4.2 Status retten (Stack)

Alle CPU-Register werden beim Aufruf eines Interrupts und vor dem Start der ISR beim HCS08 **automatisch** in den Stack gerettet und nach Abarbeitung wieder geladen. Dies ist aber nicht für alle Prozessoren selbstverständlich.
Es ist zu beachten, dass das H-Register (MSB des Indexregisters) jedoch nicht automatisch gerettet wird. Es ist daher dringend empfohlen, H am Anfang der ISR auf den Stack abzulegen. Für weitere Informationen siehe Abb. 6-20 im Skript.

3.6.4.3 Quittierung eines Interrupts, Melde-Flag löschen

Als Erkennungszeichen, das ein Interrupt-Ereignis aufgetreten ist, wird in einem internen Register ein Flag gesetzt. Das Ereignis, welches einen Interrupt auslöst, führt immer zum Setzen eins Flags. Zur Auslösung des Interrupts kommt es erst, wenn dieser **freigegeben** ist.
Das Lesen dieses Flags und das Reagieren darauf (jedoch keine Freigabe) ermöglicht die Realisierung des Pollings.
Solange die Meldeflags gesetzt sind, wird ein freigegebener Interrupt sofort ausgelöst. Ist beispielsweise ein Flag immer noch gesetzt wird beim Verlassen der zugehörigen ISR sofort wieder ein Interrupt ausgelöst. Dieses Verhalten ist nur in Ausnahmefällen sinnvoll.

Vor dem Verlassen der ISR muss also das Flag für diesen Interrupt gelöscht werden. Mittels Bit-Maskierung kann das entsprechende Bit (Flag) gelöscht werden.
Zu beachten: Das Meldeflag tritt unabhängig vom Freigabe-Bit des Interrupts auf und kennzeichnet das Auftreten des Ereignisses an.

Folglich unterscheidet sich die ISR von einer Subroutine nur durch das unvorhersehbare, spontane Aufrufen (statt mit `JSR` bzw. `BSR`) und beim HCS08 automatische Retten der Register (ausgenommen H-Register) in den Stack. Zudem erfolgt der Rücksprung mit `RTI`.

3.6.5 Freigabelogik von Interrupts

Viele Interrupts können freigegeben werden. Es gibt auch nicht maskierbare Interrupts (sog. Traps), die nicht gesperrt werden können, also immer aktiv sind. Es gibt einen übergeordneten und untergeordneten Freigabemechanismus von Interrupts. Der übergeordnete Mechanismus hat die Funktion eines Hauptschalters. Falls dieser **nicht** freigegeben ist, so werden untergeordnet freigegebene Interrupts ebenfalls nicht ausgelöst. Beim HCS08 erfolgt die übergeordnete Freigabe mit dem Assemblerbefehl `CLI` (dadurch wird das I-Bit im CCR Register **gelöscht**) und mit `SEI` werden die Interrupts gesperrt.
Bei der untergeordneten Freigabe ist es hingegen umgekehrt. `TOIE = 1` gibt bspw. den Timer Overflow Interrupt frei. Das Löschen des Bits hingegen sperrt den Interrupt.
Wird eine Interrupt-Anforderung angenommen, setzt der HCS08 das I-Bit im CCR und sperrt somit alle weiteren maskierbaren Interurpts.
Innerhalb einer ISR weitere Interrupts zuzulassen ist zwar möglich sollte aber wegen strukturellen Problemen gemieden werden. Deshalb sollte innerhalb einer ISR nicht mit den Befehlen `CLI` und `SEI` operiert werden.

3.6.6 Interruptvektoren

Ein Interruptvektor ist ein Speicherplatz, der die Startadresse der zugehörigen ISR enthält. Je nachdem ob die Vektoren fest oder variabel zugeordnet werden, spricht man von Autovektoren oder Non-Autovektoren.

3.6.6.1 Autovektoren

Der Prozessor hat **mehrere** Interruptanforderungseingänge. Jedem Eingang oder Bitkombination ist ein Interruptvektor **fest** zugeordnet. In diesen sind die die Startadressen der ISRs gespeichert.

3.6.6.2 Non-Autovektoren

Der Prozessor hat **einen** Interruptanforderungseingang (IRQ) und einen Interruptquittierausgang (INT_ACK). Nach der Anforderung (IRQ) sendet der Prozessor das Bestätigungssignal (INT_ACK) und erwartet die Eingabe des Interruptvektors über den Datenbus. Erst danach wird der entsprechende Programmabschnitt ausgeführt.

3.6.7 Mechanismen der Interrupterkennung für Peripheriegeräte

3.6.7.1 Hardware-Variante (Interrupt-Vectoring)

Für jedes externe Gerät gibt es einen Interrupt-Hardware-Eingang mit seinem zugeordneten Interruptvektor. Bei sehr hohem HW-Aufwand erzielt man damit sehr schnelle Reaktionszeiten.

3.6.7.2 Interrupt-Polling (SW)

Bei dierser Variante wirken alle externen Interrupts auf **denselben** Prozessor-Eingang (z.B. IRQ-Eingang). Der Prozessor fragt der Reihe nach ab wer diesen Interrupt verursacht hat (Polling).

3.6.7.3 Daisy-Chain

Bei der Daisy-Chain kaskadiert man die externen Geräte mittels einer Interrupt-Enable Leitung (IE_I, IE_O) so, dass man hardwaremässig auch eine **Prioritätenregelung** erhält. Alle Interruptanforderungen gehen auf denselben IRQ und lösen dort das INT_ACK aus, das beim auslösenden Gerät bewirkt (bspw. Gerät 2), dass einerseits die Enable-Leitung unterbrochen wird (IE_O von Gerät 2 bis 0) und andererseits der Interruptvektor für den Prozessor auf den Datenbus gelegt wird. Tritt in Geräten mit niedrigerer Priorität ein Interrupt auf, wird dieser zurückgestellt bis der anliegende Interrupt abgearbeitet worden ist.
Die Daisy-Chain hat eine einfache Struktur und benötigt keinen zusätzlichen HW-Aufwand; dafür ist sie zeitaufwendig und unflexibel.

3.6.7.4 Interrupt-Controller

Eine weitere Lösung für einen Prozessor mit nur einem IRQ ist der Einsatz eines externen Interruptverwalters oder sog. Programmable Interrupt Controller (PIC). Dieser übernimmt die Koordination der anfallenden Interrupts.

3.6.7.5 DMA: Blocktransferprozessor

Diese Variante kommt meistens für den schnellen Datenverkehr zwischen Datenspeicher und beliebigen Interface-Bausteinen zum Einsatz. Ein DMA-Controller benützt mit höchster Priorität den Datenbus um den Datenverkehr zu bewerkstelligen. Der Prozessor wird über eine Interruptleitung vorübergehend vom Datenbus abgehängt.

3.6.8 Interrupt-Prioritäten (HCS08)

Sobald ein Interrupt anliegt, wird dieser fertig verarbeitet, auch wenn er eine kleinere Priorität hat als ein neu auftretender Interrupt mit höherer Priorität. Die Prioritäten gelten nur für den Fall des gleichzeitigen Anliegens mehrerer Interrupts. In diesem Fall mit Autovektoren hat jeder Autovektor eine fest zugeordnete Priorität. Beim HCS08 haben Interrupt-Vektoren mit einer tieferen Adresse ebenfalls eine niedrigere Priorität. Siehe Abb. 6-26.

Bei Non-Autovektoren müssen die Bausteine durch eine HW-Schaltung so verriegelt sein, dass nur der Baustein mit der höchsten Priorität seinen Vektor (nach dem INT_ACK) auf den Bus legen kann (bspw. bei Daisy-Chain).

3.7 Timer-System

3.7.1 Aufbau Timersystem

Die Aufgabe von Timersystemen sind Impulszählung und –generierung, Zeitmessung und zeitsynchrone Ereignisse auslösen.
Der HCS08 hat ein Timersystem und zwei Timermodule:

- `TPM1` hat 3 Kanäle (`TPM1CH0` bis `TPM1CH2`, entsprechend Port D Bit 0 bis 2)
- `TPM2` hat 5 Kanäle (`TPM1CH0` bis `TPM1CH2`, entsprechend Port D Bit 0 bis 2)

Die Kanäle jedes Timer-Moduls haben folgende Eigenschaften:

- der Clock kann von extern oder ab Busclock gewählt und mit einem Prescaler verlangsamt werden.
- Timersystem ist einschaltbar
- Verschiedene Modi: Input-Capture, Output-Compare, PWM
- Triggerung bei Input-Capture: Rising-Edge, Falling-Edge oder bei jeder Flanke
- Triggerung bei Output-Compare: Setzen, Löschen oder Toggeln des Ausgangssignals
- Bei PWM kann die Polarität gewählt werden
- freilaufender Up- oder Downcounter
- 16-bit Modulus Register zur Festlegung des Zählbereichs vom Timersystem
- Je **ein** Interrupt pro Kanal verfügbar

Die Steuerregister zur Programmierung der Timerfunktionen sind für jedes Timermodul identisch aufgebaut. Deshalb wird der Kanal mit „n“ angegeben und die Modulnummer mit „x“ (bspw. TPxCHn).

3.7.1.1 16-bit Counter

Jedes Timermodul hat einen eigenen 16-bit Counter, welcher von 0 bis 65535 zählt. Mit dem Register `TPMxSC` können folgende Einstellungen vorgenommen werden:

- `TOF`: Timer Overflow Flag. Sobald der Counter überläuft, also seinen Grenzwert überschreitet, fängt er wieder von vorne zu zählen an. Bei diesem Überlauf wird das `TOF` Flag gesetzt.
- `TOIE`: Timer Overflow Interrupt Enable. Ist dieses Flag gesetzt, so wird ein Timer Overflow Interrupt ausgelöst, sobald das `TOF` gesetzt gesetzt ist.
- `CLKSB,CLKSA`: Mit diesen zwei Bits können vier verschiedenene Clock-Quellen eingestellt werden.
    - 0:0 (`CLKSB:CLKSA`) Kein Clock selektiert (TPM deaktiviert)
    - 0:1 Verwendung des Bus-Clocks (`BUSCLK`)
    - 1:0 Fixierter System-Clock (`XCLK`)

  - 1:1 Externe Clockquelle
- PS2,PS1,PS0: Damit wird der Prescaler gesetzt. Der Clock kann um die Faktoren 1, 2, 4, 8, 16, 32, 64, 128 verlangsamt werden. D.h. dass der Counter um diesen Faktor langsamer zählt. Dadurch wird ebenfalls der Counter verlangsamt, der bei jedem Clock-Zyklus hinauf bzw. hinunterzählt.

Im Modulo-Register TPMxMODH, TPMxMODL kann für den Timer der Grenzwert gesetzt werden. Überschreitet der Counter diesen Wert, so fängt er wieder von vorne zu zählen an. Das TOF Flag wird dann gesetzt.
Mit den Registern TPMxCNTH (high-Byte), TPMxCNTL (low-byte) kann der Counterwert gelesen werden. Da der HCS08 ein 8-bit Prozessor ist, kann der 16-bit Wert nicht in einem Durchgang gelesen werden. Es spielt jedoch keine Rolle ob man zuerst das High- oder Low-Byte ausliest. Das andere Byte wird zum Zeitpunkt des Lesens automatisch in einen Buffer geladen. Wird also nun das andere Byte ausgelesen, wird der Wert aus dem Buffer entnommen und entspricht dem Wert des ersten Lesezeitpunkts. Neue Werte können vom Counter nur ausgelesen werden, wenn beide Bytes des alten Werts ausgelesen wurden.
Jede Schreiboperation auf TPMxCNT setzt den Counter auf den spezifizierten Wert. Der Counter wird jedoch nur auf den Wert gesetzt, wenn das High- und Low-Byte beschrieben wurde. Eine häufige Fehlerquelle ist, dass in gewissen Fällen von Counterwerten (bspw. 4400h) nur das High-Byte im Counter gesetzt wird. Der Counter setzt diesen Wert aber erst wenn auch das Low-Byte mit 00h beschrieben wurde.

3.7.2 Aufbau eines Timermodul-Kanals

Für jeden Kanal eines Timermoduls kann der Input-Capture bzw. Output-Compare Mode eingestellt werden mittels CPWMS, MSnB, MSnA, ELSnB, ELSnA. Siehe dazu die Vorgaben in Abb. 6-36.

3.7.2.1 Output-Compare

Im Output-Compare Modus dienen die Register TPMxCnVH (High-Byte), TPMxCnVL (Low-Byte) als 16-bit Comparator. D.h. dass der gesetzte Wert in diesen Registern als Vergleichswert dient. Jedesmal wenn der Counter inkrementiert wird, überprüft der Prozessor, ob der Counterwert mit dem Comparatorwert identisch ist. Falls ja, so wird das CHnF gesetzt. Ist noch das Bit CHnIE (Interrupt Enable) gesetzt, so wird ein Interrupt ausgelöst. Je nachdem wie die Register gesetzt sind (Abb. 6-36) kann beim Auftreten des Interrupts das Ausgangssignal, am zugehörigen Pin von Port D des Kanals, gesetzt, gelöscht oder getoggelt werden.

3.7.2.2 Input-Capture

Hierbei dienen die Register TPMxCnVH , TPMxCnVL als 16-bit Latch, also als Datenbuffer. Das Timermodul kann so eingestellt werden, dass es bei fallenden, steigenden oder allen Flanken, die von aussen kommen (entsprechender Pin an Port D) einen Interrupt auslöst. Der Zeitpunkt des Ereignisses (Counter-Wert) wird dann in den Latch abgespeichert. Wie bei Output-Compare, wird beim Ereignis das Flag CHnF gesetzt und der Interrupt mit CHnIE.

3.8 Test & Measurement

Nebst dem Debugger sind das Kathodenoszilloskop (KO) sowie der Logic Analyzer wichtige Werkzeuge zum Messen und Überprüfen.
Ein Logic Analyzer erlaubt die Analyse von digitalen Signalen. Der zu analysierende Kanal wird digitalisiert, abgetastet und dauernd in einen Ringbuffer gespeichert. Der Datenfluss im Speicher wird durch eine Trigerrung gestoppt.
Ein Logic-Analyzer ersetzt nicht einen KO, da er nur digitale Signale verarbeitet.

3.8.1 Wichtige Begriffe

- Abtastrate (Sampling Rate): Die Geschwindigkeit eines Logic-Analyzers. Eine Rate von 100MHz erlaubt alle 10ns eine Triggerentscheidung und das Speichern des Signals.
- Threshold (Level): Bezeichnet die Schwellspannungen der Logikzustände 0 und 1
- Triggerbedingung: Die Triggerwörter können durch Bedingungen (logische und auch programmgesteuerte) verknüpft werden.
- Triggering: Das gezielte Anhalten des Datenflusses im Speicher bei erfüllter Triggerbedingung
- Trigger Delay: Anzahl Abtastungen, die nach erfolgter Triggerung noch gespeichert werden.

3.9 A/D Wandler

Analog-Digital-Wandler (A/D) sind nötig, damit analog vorliegende Signale (z.B. Spannungsmesswerte) in Form von digitalen Signalen in die Berechnungs-Software überführt werden können.
Beim HCS08 ist ein 8-Kanal A/D Wandler System auf dem Chip integriert und spielt mit dem Port B zusammen.
Der A/D Wandler hat zwei Referenzspannungen High und Low. Das eintreffende Signal wird in Abhängigkeit zu den beiden Referenzspannungen analysiert und als digitales 8-bit bzw. 10-bit Signal in die Register `ATDRL` (Low-Byte) und `ATDRH` (High-Byte) gespeichert. Entspricht also das Eingangssignal der Low Referenzspannung, so ist dann der gewandelte Wert `00000000(00)`.

Es kann immer nur ein Kanal gewandelt werden. Die Auswahl des Kanals erfolgt im Register `ATDSC` im Bereich `ATDCH` Bit 4:0 (siehe Abb. 6-43). Will man mehrere Kanäle erfassen, so müssen diese durch Ändern von `ATDCH` einzeln verarbeitet werden.

Um Signale wandeln zu können sind folgende Schritte notwendig:

- A/D Wandler aktivieren: `ATDC_ATDPU = 1`
- Resolution einstellen: `ATDC_RES8 = 1` (1 = 8bit, 0 = 10bit)
- Entsprechenden Kanal-Pin auf Port B aktivieren: `ATDPE_ATDPEx = 1` (x entspricht dem Wandlerkanal 1 – 8)
- Kanal selektieren: `ATDSC_ATDCH = x` (Selektiert Kanal 7)
- Gewandelten Wert auslesen: `ATDRH` bzw. `ATDRL`

Ist der 8-bit Mode aktiv, so wird der Wert im Register ATDRH (High-Byte) gespeichert. Im 10-bit Mode wird der Wert links- oder rechtsausgerichtet auf das High- und Low-Byte verteilt (siehe Abb. 6-46 bis 6-48).

3.10 I²C Bus

Der I²C Bus ermöglicht die Kommunikation zwischen mehreren IC-Bausteinen und hat folgende Eigenschaften:

- seriell
- bidirektional
- synchron
- Aktuelle Datenübertragungsrate: 3.4Mbit/s
- Max. zulässige Bus-Kapazität: 400pF
- besteht aus nur zwei Leitungen
    - SDA: Datenleitung (Serial Data Line)
    - SCL: Taktleitung (Serial Clock Line)

Die beiden Leitungen sind jeweils an der positiven Versorgungsspannung über einen Pull-Up Widerstand angeschlossen. Ist der Bus frei sind beide Leitungen auf HIGH.

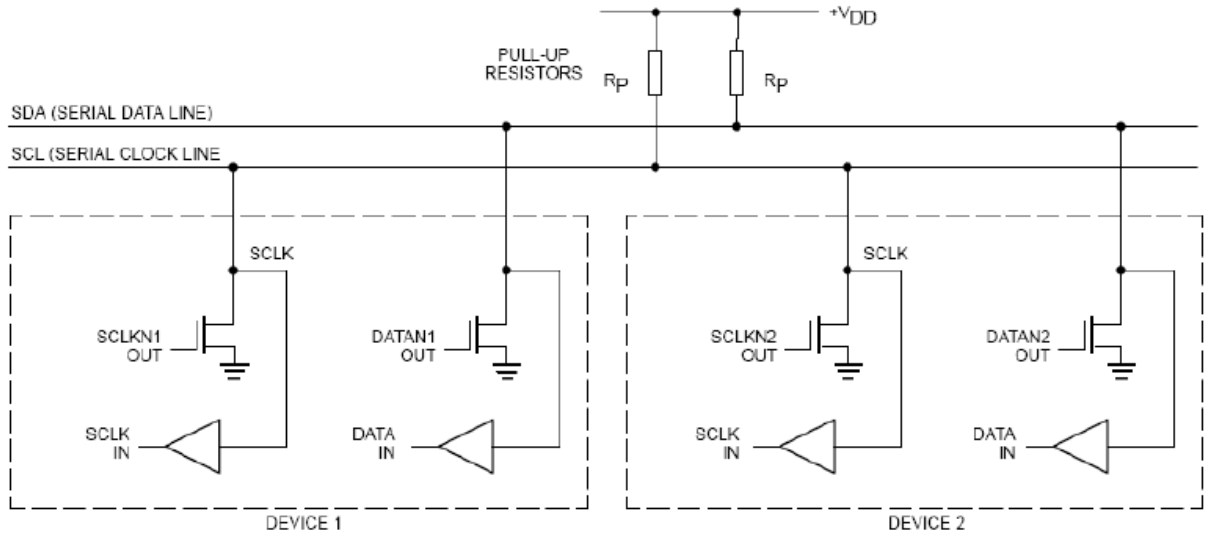


Abbildung 4: Prinzipdarstellung des IIC Bus

3.10.1 I²C-Modul des HCS08

Das Modul arbeitet auf den Pins PTC3 (SCL) und PTC2 (SDA). Master oder Slave Betrieb ist möglich. Die max. Übertragungsrate beim HCs08 beträgt 400kBit/s.

3.10.2 I²C-Protokoll

Eine Standardkommunikation ist aus folgenden vier Teilen aufgebaut:

- START Signal generieren
- Slave-Adresse übermitteln

- Daten übermitteln
- STOP Signal generieren

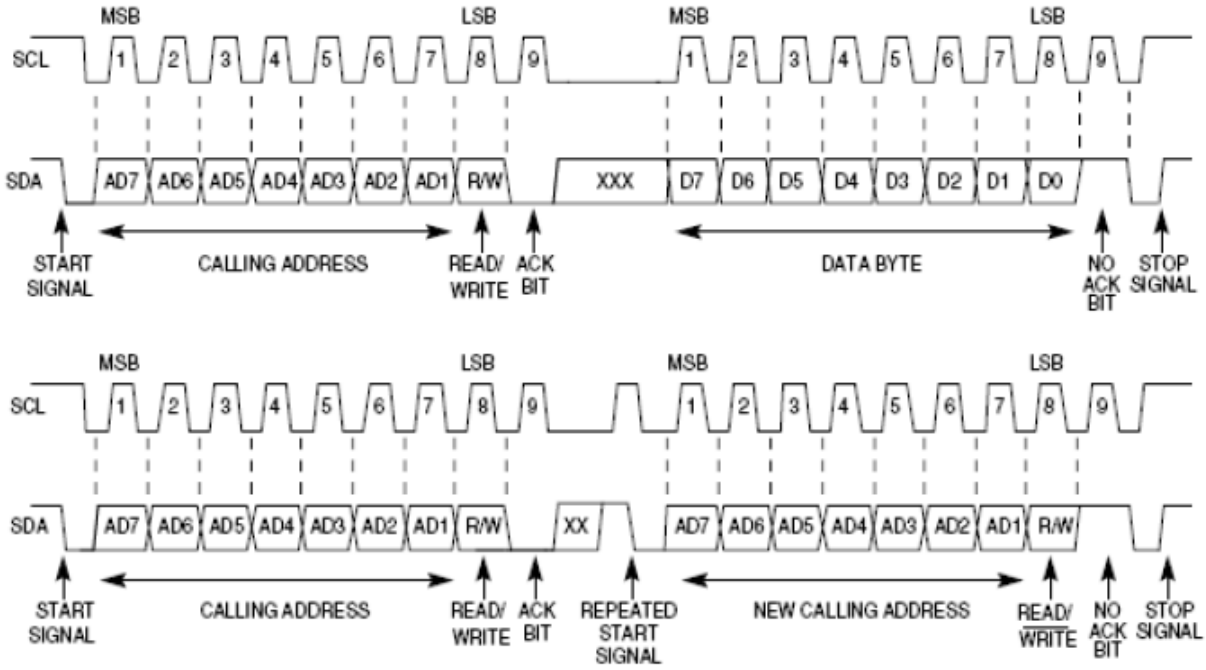


Abbildung 5: IIC Protokoll

3.10.2.1 START Signal

Ist der Bus frei (beide Leitungen auf High), also kein Master Device besetzt den Bus, so kann ein Master eine Kommunikation initiieren indem er ein START Signal sendet. Ein START Signal ist definiert durch einen High-to-Low Übergang während SCL High ist (siehe Abb. 6-53). Das START Signal informiert über den Beginn eines Datentransfers und soll alle Slave Devices aufwecken.

3.10.2.2 Slave-Adresse übermitteln

Das erste Byte, das nach dem START Signal übertragen wird ist die Slave-Adresse. Die Adresse ist 7-bit lang gefolgt von einem R/W Bit, das die Richtung des Datentransfers vorgibt.
Das R/W Bit definiert zwei Zustände:

- 0: Write transfer: Master überträgt Daten zum Slave
- 1: Read transfer: Slave überträgt Daten zum Master

Ein Slave Device antwortet mit einem Bestätigungs-Bit (ACK), wenn die übermittelte Slave-Adresse mit dessen des Slaves übereinstimmt. Ein ACK erfolgt durch Runterziehen der SDA Leitung auf Low im neunten Clock des SCL (siehe obige Abbildung).
Zwei Slave Devices dürfen nicht die gleiche HW-Adresse nutzen. Ein IC kann nicht gleichzeitig Master und Slave Device sein.

3.10.2.3 Daten übermitteln

Nach der Übermittlung der Slave-Adresse folgen die eigentlichen Nutzdaten. Diese können sogar Sub-Adressen beinhalten. Jedes Datenbyte ist 8-bit lang. Das MSB wird zuerst übermittelt. Jedes erfolgreich empfangene Datenbyte wird vom Empfänger mit einem Bestätigungs-Bit (ACK) an den Sender bestätigt. Zum erfolgreichen Versenden eines Datenbytes sind demnach 9 SCL Clocks notwendig.

Reaktion auf fehlende Bestätigung:

- Empfängt das Master Device kein ACK-Bit vom Slave, muss der Slave SDA auf High halten.
- Empfängt das Slave Device kein ACK-Bit vom Master, gibt der Slave die SDA Leitung frei.

In beiden Fällen wird der Transfer unterbrochen und der Master unternimmt eine von zwei Aktionen:

- STOP Signal generieren und somit Bus freigeben
- Repeated START Signal generieren und einen neuen Datentransfer initiieren

3.10.2.4 STOP Signal

Der Master kann die Kommunikation beenden, indem er ein STOP Signal sendet um somit den Bus freizugeben. Das STOP Signal ist definiert durch einen low-to-high Übergang auf SDA während SCL auf High ist.
Der Master kann aber auch ein START Signal senden ohne vorher ein STOP Signal zu generieren. Diese Methode wird Repeated START Signal genannt und dient der erneuten Kommunikation mit einem (desselben) Slave ohne den Bus freizugeben.

3.10.2.5 Beispiel – Master sendet Adressbyte und liest Daten vom Slave

Master sendet das START Signal und initiiert die Kommunikation. Anschliessend gefolgt von der 7-bit Slave-Adresse und das R/W Bit = 1, welches einen Read transfer (Slave übermittelt Daten an Master) angibt. Der Slave antwortet mit ACK (zieht während Bit9 SDA auf 0). Anschliessend sendet der Slave ein oder mehrere Datenbytes. Nach jedem empfangenen Byte bestätigt der Master mit einem ACK. Die Übertragung endet mit einem STOP Signal.

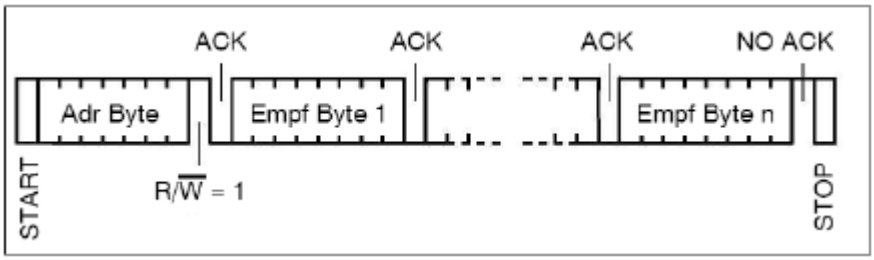


Abbildung 6: Master sendet Adressbyte und empfängt Daten vom Slave

3.10.3 HCS08 als I²C-Bus Master initialisieren

3.10.3.1 Frequenzregister

Zuerst muss definiert werden welche Baudrate der Master erzeugen soll.
Die Baudrate berechnet sich nach folgender Formel:

*IIC baud rate = bus speed[Hz] / (mul * SCL divider)*

Hierfür ist das Register IIC1F (IIC Frequency Divider Register) zuständig.

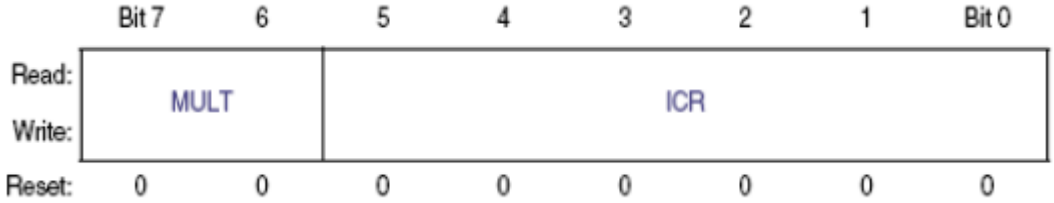


Abbildung 7: Frequenzregister IIC1F

Dabei wird der Multiplikator mit den Bits MULT eingestellt. Die ICR Bits gelten für den SCL Divider.

Zu beachten ist dabei auch der SDA Hold Value (siehe Abb. 6-55). Dieser Wert ist abhängig vom angeseuerten Baustein. Es ist die Zeit, wie lange das Datensignal noch nach der fallenden Clock Flanke anliegen muss.

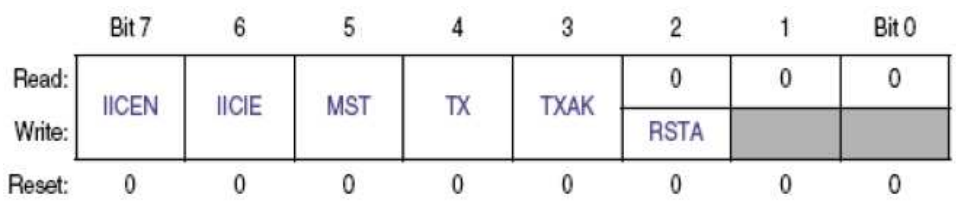
3.10.3.2 Kontrollregister

Im IIC1C (IIC Control Register) werden die entsprechenden Einstellungen vorgenommen:

Abbildung 8: IIC1C Kontrollregister

Folgende Kontrolleigenschaften bieten sich an:

- *IICEN (IIC Enable):* Aktiviert den I²C Bus (IICEN = 1, schaltet das Modul ein)
- *IICIE (IIC Interrupt Enable):* Definiert ob ein I²C Interrupt ausgelöst wird (IICIE = 1, aktiv)



- *MST (Ma*

ster Mode Select): Definiert ob der HCS08 als Master (1) oder Slave (0) Device agiert. Ändert sich MST von 0 auf 1 so wird ein START Signal generiert und der Master Mode ist aktiv. Ändert es hingegen von 1 auf 0 wird ein STOP Signal generiert und der HCS08 agiert als Slave.
- *TX (Transmit Mode Select):* Definiert die Richtung des Datentransfers (1 = Write transfer, 0 = Read transfer).
- *TXAK (Transmit Acknowledge Enable):* Definiert ob nach Empfang eines Datenbytes ein ACK-Bit zurückgesendet wird (0 = Nein, 1 = Ja).
- *RSTA (Repeated START):* Wird dieses Bit auf 1 gesetzt wird ein Repeated START Signal generiert.

3.10.3.3 Datenregister

Das IIC1D (IIC Data I/O) Register ist zuständig um Daten an den Bus zu senden oder zu empfangen. Im Master Write Transfer Mode wird ein Datentransfer **automatisch** initiiert wenn ein Byte in das Register geschrieben wird.
Im Master Read Transfer Mode wird beim Auslesen des Registers **automatisch** der Empfang des nächsten Datenbytes initiiert (ACK wird gsesendet).

3.10.3.4 Statusregister

Das IIC1S (IIC Status Register) enthält mehrere Flags, die über den Status des Busses informieren:

- *TCF (Transfer Complete Flag):* Wird dieses Bit gelesen wird es automatisch auf 0 gesetzt. (1 = Datentransfer komplett, 0 = Transfer noch in Ausführung)
- *IAAS (Addressed as a slave):* Dieses Bit wird gesetzt, wenn die aufzurufende Adresse mit derer des HCS08 übereinstimmt.
- *BUSY (Bus Busy):* Auskunft über den Status des Busses egal ob Master oder Slave Modus gewählt ist. Flag wird gesetzt wenn ein START Signal detektiert wurde und gelöscht wenn ein STOP Signal eintrifft.
- *ARBL (Arbitration Lost):* Ist dieses Bit auf 1 erfolgte ein Fehler bei der Datenübertragung.
- *SRW (Slave Read/Write):* Ist der HCS08 als ein Slave konfiguriert, dann entspricht eine 1 dem Slave transmit Mode und ein 0 dem Slave receive Mode.
- *IICIF (IIC Interrupt Flag):* Gibt an, ob ein Interrupt-Bedingung auftrat (1) oder nicht (0). Falls IICIE gesetzt ist, wird ein Interrupt ausgelöst. Das IICIF Bit wird bei folgenden Events ausgelöst:
  - Ein Datenbyte wurde transferiert
  - Slave Adresse entspricht der aufzurufenden Adresse
  - Fehler bei der Datenübertragung
- *RXAK (Receive Acknowledge):* 1 = Kein ACK empfangen, 0 = ACK empfangen.

3.10.4 HCS08 als I^2C-Bus Slave initialisieren

Wird der Prozessor als Slave betrieben, braucht er eine Adresse auf welche er anspricht. Diese wird im IIC1A (IIC Address Register) programmiert.

3.11 Betriebssysteme

3.11.1 Allgemein

Der Einsatz eines Multitask-Betriebssystemes ist sinnvoll, wenn in einem Rechner einzelne Teile eines Programms „parallel“ ausgeführt werden. Ein Task (oder auch Prozess, Thread) ist ein in sich abgeschlossener Programmteil, der „parallel“ zu anderen Tasks eine bestimmte Aufgabe zu bewältigen hat. (z.B. RS-232-Treiber, Tastaturscanner, Displaytreiber oder Berechnungsprozedur)

3.11.2 Begriffe

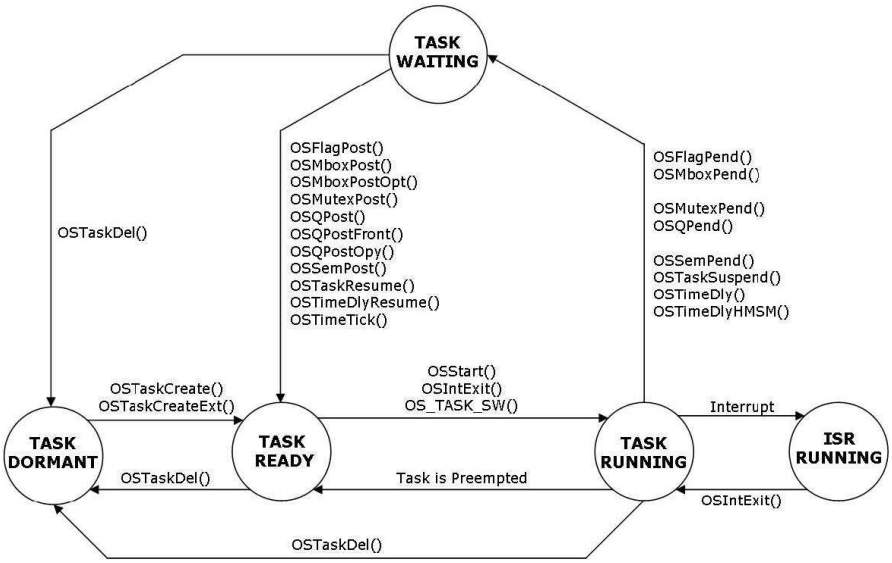
- *Kernel:* Die Grundfunktionen des Multitask-Betriebssystems (Scheduler, Task-Kommunikation, Task-Synchronisation, Timerverwaltung) sind im Betriebssystemkernel zusammengefasst.
- *Scheduler:* Der Scheduler teilt einem Task die CPU zu oder unterbricht ihn und setzt ihn in den Warte/Bereitschaftszustand. Bedient der Scheduler alle bereiten Tasks sequentiell, so spricht man von round-robin. Oft bieten Scheduler auch die Möglichkeit an, verschiedene Prioritätsstufen zu unterscheiden. (uCOS-II)
- *Preemptiv:* Bei Verwendung von preemptivem Multitasking wird einem Task die CPU durch den Scheduler entzogen (wenn z.B. ein Task höherer Priorität ablaufbereit wird). Der Task bemerkt dies nicht d.h. er muss sich nicht kooperativ verhalten, um unterbrochen werden zu können
- *Kooperativ:* Bei kooperativem Multitasking wird ein Task erst unterbrochen, wenn er selbständig (durch Warten mittels Betriebsystem-Funktionsaufruf) die CPU abgibt. Eine Endlos-Schlaufe, ohne Freigabe des Tasks durch Warten oder Suspendieren, würde den Prozessor blockieren.
- Das uCOS-II-System arbeitet preemptiv
- *Echtzeit OS:* Gemäss Definition müssen Echtzeit-Anwendungen auf äussere Ereignisse innerhalb eines vorhersagbaren Zeitraums reagieren. Dies gilt besonders für "harte" Echtzeit-Systeme, wo nicht eingehaltene Dead-Lines schwerwiegende oder sogar katastrophale Konsequenzen nach sich ziehen würden.
- *Portabel:* uCOS-II kann auf einer Vielzahl von Mikrocomputern angewendet werden. Wichtige Voraussetzung damit dies auch Funktioniert ist das Vorhandensein eines Stackpointers. Ebenfalls muss der C-Compiler Inline-Assemblierung zulassen, da der Grossteil der Anpassungen in ASM geschrieben ist. uCOS-II kann auf 8-, 16-, 32-, oder 64 bit MC’s zum Laufen gebracht werden.
- *Skalierbar:* Mit Hilfe des Konfigurationsfile *Os_cfg.h* ist es sehr einfach möglich Dienste wie Semaphor, Mutex und andere unerwünschte Funktionen auszuschalten, damit der kompilierte Code möglichst klein wird.

3.11.3 Tasks

3.11.3.1 Beschreibung eines Tasks

Ein Task (oder auch Prozess, Thread) ist ein in sich abgeschlossener Programmteil, der „parallel“ zu anderen Tasks eine bestimmte Aufgabe zu bewältigen hat. Ein Task erhält zu einem bestimmten Zeitpunkt die Ressourcen des Systems (CPU-Zeit, Programmcode, Speicher, IO-Geräte,...) zugeteilt.	**Task besteht aus** Daten Stack Programmcode { : : } akkuzustand

3.11.3.2 Task-Zustandsdiagramm



Ready: Der Task ist bereit zum Laufen und wartet, bis er vom Scheduler aktiviert wird.
Running: Der Task läuft. (CPU wird von diesem Task benützt)
Waiting: Der Task wartet auf ein Ereignis (Timer, Semaphor, Input,…). Sobald dieses Ereignis eintritt, wird er in den Ready-Zustand versetzt.
Dormant: Im Zustand Dormant (ruhend) ist der Task vom Scheduler nicht mehr aufrufbar. (Der Speicher wird nicht gelöscht)
ISR-Running: Sobald ein Interrupt auftritt wird in den Zustand ISR-Running gewechselt. (Bis zu einer Tiefe von 255)

3.11.3.3 Operationen auf einen Task

Starten eines Tasks (OSTaskCreate(…))

Anhalten eines Tasks (OSTaskSuspend(…))
Fortsetzen eines Tasks (OSTaskResume(…))
Taskpriorität ändern (OSTaskChangePrio(…))
Vernichten eines Tasks (OSTaskDel(…))

3.11.3.4 Initialisierung der Tasks

Ein Task kann Statisch (vor der Laufzeit) oder Dynamisch (während der Laufzeit) sein.

3.11.3.5 Task starten

OSTaskCreate(void (*task)(void *pd), void *pdata, OS_STK *ptos, INT8U pri0)

pd ein Pointer auf den Task Code (Funktion)
pdata ein Pointer auf ein Argument, dass dem Task bei der Ausführung übergeben wird
ptos ein Pointer auf den zugehörigen Stack (Kann bei jedem Task verschieden gross sein)
prio die gewünschte Taskpriorität

Code Beispiel:

```
#define TASK_STK_SIZE 40 /* Stackgrösse pro Task */

INT8U InitTaskStk[TASK_STK_SIZE];      // pro Task ein Stack
void InitTask(*pdata);
void main(void)
{
  OSInit();    /* OS initialisieren */
  OSTaskCreate(InitTask, (void*)0, (void*)&InitTaskStk[TASK_STK_SIZE], 2);
  OSStart();    /* Betriebssystem starten */
}
```

3.11.3.6 Formen eines Tasks

- Form1: Endloos loop
- Form2: Selbstzerstörend

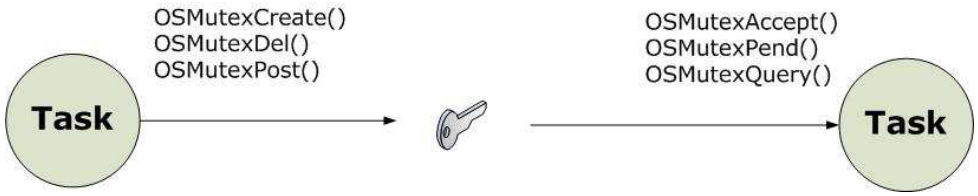
Ein Task kann nie etwas zurückgeben und ist immer void!

Code Beispiel Endlostask:

```
void yourTask1 (void *pdata)
{
  for(;;)
{
```

Code Beispiel selbstzerstörender Task:

```
void yourTask2 (void *pdata)
{
  OSTaskDel(OS_PRIO_SELF);
}
```



3.11.3.7 Kooperation zwischen Tasks

Tasks, die nicht unabhängig voneinander funktionieren, sondern an der Lösung einer gemeinsamen Aufgabe beteiligt sind, müssen miteinander kooperieren. Man unterscheidet zwei Aspekte dieser

Kooperation:

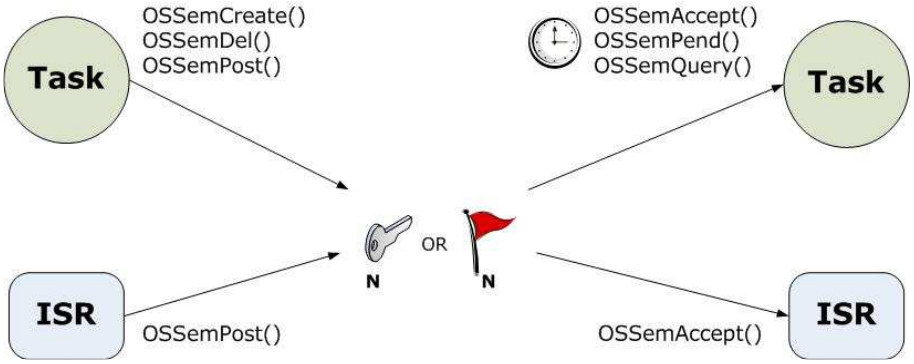
Kommunikation	bezeichnet den Austausch von Daten zwischen den Prozessen.
Synchronisation	dient dazu, die Ausführung verschiedener Tasks zu koordinieren, d.h. die Aktivität eines Tasks vom Zustand eines anderen Tasks abhängig zu machen.

3.11.4 Synchronisation

Die Bedingungen der Synchronisation werden von Semaphoren und Eventflags erfüllt.

3.11.4.1 Semaphor

Ein Semaphor (Signalisation) ist im Prinzip ein Zähler, der durch die Funktion ***OSSemPost(…)*** erhöht wird, wenn z.B. Daten gültig sind oder ein kritischer Abschnitt frei wird.
Mit der Funktion ***OSSemPend(…)*** wird der Zugriff auf den kritischen Abschnitt angefordert. Falls er nicht frei ist (Semaphor = 0), geht der Task warten, bis das Semaphor ≥ 1 wird. Danach wird er wieder Ready und erniedrigt das Semaphor, falls seine Priorität genügend hoch ist.



Code Beispiel:

```
OS_EVENT *sem_X;                        /* Semaphor definieren */
sem_X = OSSemCreate(100);               /*Semaphor kreieren und auf 100 setzen */
OSSemPend(sem_X, 0, OS_NO_ERR);  /*Warte auf ein freies Semaphor */
```

3.11.4.2 Mutual Exclusion Semaphore

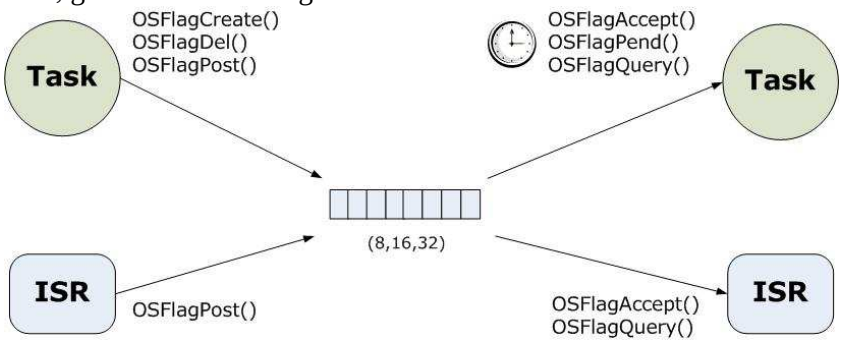
Es ist ein so genanntes Binäres Semaphor, welches nur die Zustände 0 und 1 kennt. Dieses Mutex wird bei der Erzeugung automatisch auf 1 initialisiert.
Code Beispiel:

```
OS_EVENT *mut_X;                        /* Mutex definieren */
mut_X = OSMutexCreate(9, OS_NO_ERR)     /* Mutex kreieren */
OSMutexPend(mut_X, 0, OS_NO_ERR);       /* Warte auf ein freies Mutex */
OSMutexPost(mut_X);                     /* Gibt Mutex frei */
```

Beim kreieren des Mutex muss der Funktion ***OSMutexCreate(…)*** an erster Stelle eine Priorität mitgegeben werden. Diese Priorität sollte höher sein als die höchste Priorität, die auf diese Ressource zugreift. Dient zur Vermeidung von Verklemmungen.

3.11.4.3 Eventflag

Mit Hilfe der Eventflags lässt sich eine schnelle 1-Bit Synchronisation/Kommunikation erreichen. Die Eventflags werden in Gruppen zu 8, 16 oder 32-Bit zusammengefasst. Die einzelnen Eventflag-Funktionen beziehen sich immer auf eine solche Gruppe. Die Funktionen können benutzt werden, um Bits (vor allem auch Hardware-Register) zu setzen oder auf ein Bit oder eine bestimmte Bitkombination zu warten. Der Anwendungsprogrammierer muss dafür besorgt sein, dass die Eventflags im richtigen Zeitpunkt initialisiert, gesetzt oder zurückgesetzt werden.

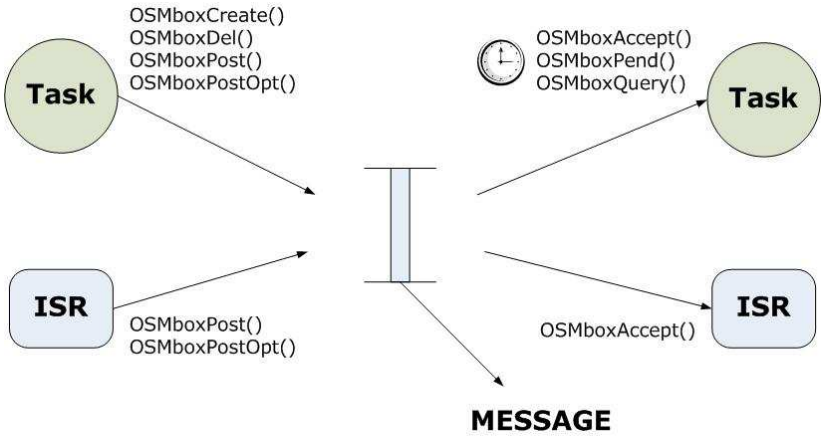


3.11.5 Kommunikation

Der Datenaustausch zwischen Tasks wird entweder durch eine gemeinsame Variable, die mit einem Semaphor vor wechselseitigem Zugriff geschützt ist, oder direkt durch Kommunikationshilfsmittel (Mailboxen, Queues) realisiert.

3.11.5.1 Mailbox

Eine Mailbox ist sinngemäss ein Briefkasten, über welcher ein Task einem anderen Task Daten, Botschaften oder Meldungen mitteilen kann. Beim Betriebssystem uCOS-II ist die Mailbox ein Pointer auf eine Anwenderspezifische Struktur, welche die Nachricht enthält. Eine Mailbox kann nur einen Pointer ungleich NULL enthalten (Mailbox ist voll) oder ein Pointer nach NULL (Mailbox ist leer).



3.11.5.2 Queue

Eine Queue ist im Prinzip ein FIFO-Buffer, der vom Kernel verwaltet wird. Dieser FIFO-Buffer besteht aus einer Anzahl Pointer, welche auf eine benutzerspezifische Datenstruktur zeigen. Die Queue-Funktionen garantieren einen exklusiven Zugriff auf die Queue. Kann eine Queue-Funktion nicht ausgeführt werden (Queue-Zugriff gesperrt, Queue leer beim Lesen, Queue voll beim Schreiben), so wird der aufrufende Task in den Wartezustand versetzt, bis die Queue bereit ist.

3.11.6 Timer-Verwaltung

Ein Multitask-Betriebssystem besitzt normalerweise ein Timer-System, mit dessen Hilfe jeder einzelne Task sich eine gewisse Zeit schlafen legen kann, ohne das ganze System zu blockieren
Beim uCOS-II hat jeder Task die Möglichkeit, sich um ein Vielfaches von 20 ms zur Ruhe zu legen (waiting). Sobald die Wartezeit abgelaufen ist, wird der Task vom Scheduler wieder in den Ready-Zustand versetzt. Der Timer muss im Initialisierungsteil mit dem Befehl OSTimerInit() gestartet werden.

Code Beispiel:

```
OSTimeDly(100); /* Timer auf 2000 ms (2s) setzten*/
OSTimeDlyHMSM(1,0,0,0); /* Timer auf 1h setzen */

OSTimeDlyResume(5);   /* Task mit Priorität 5 wieder fortsetzen */
OSTimeGet();          /* Anzahl Ticks seit OSStart() oder OSTimeSet() */
OSTimeSet();
```

Nachdem das Multitaskingsystem mit dem Befehl ***OSStart()*** gestartet wird, läuft im Hintergrund ein 32-Bit Zähler mit, der mit jedem Tick erhöht wird. Dieser Zähler läuft also nach 2^32 Ticks über. Mit einem Abstand von 20ms zwischen zwei Ticks läuft dieser Zähler nach 994 Tagen über. (2.72 Jahre)
Wenn eine andere Tickzeit gewünscht wird, kann diese mit Hilfe des HCS08 Register ***SRTISC*** geändert werden. Die Einstellungen für den Timer befinden sich im File
uCOS\Ports\OS_CPU_C.c