# KLEIN, aber PICfein

*Microchip's PIC-Mikrocontroller*

Robert SCHWAGER PCN-DSK-520:\pic

Der Mikrocontroller von Microchip - PIC - ist eine 'neue' Art von „Mikroprozessörchen“ mit wenigen Befehlen (, was eine kurze Einarbeitszeit ermöglichen sollte,) mit integrierten Speichereinheiten für Programm und Daten, wobei die ‘breiten’ Daten-, Adreß- und Steuerbusleitungen auf die äußere Prozessorperipherie entfallen. Die integrierte Peripherie des PICs ist ideal für Kleinsteuerungen, Überwachungen von Prozessen, Motorüberwachungen und PAL-Ersatz (Emulation) in RISC-Architektur.

Der Name PIC soll von *Periphery Intergrated Controller* abgeleitet sein.

PIC16/17-Mikrocontroller vom Hersteller Microchip ist eine Kombination von „Hochleistungsprozessor“, niedrigem Preis und kleinem Gehäuse, und bietet ein gutes Preis-Leistungsverhältnis für Industrieanwendungen. Über 200 Millionen PICs werden in kostengünstigen Konsumgütern, Computerperipherien, Büroautomatisations-, Sicherheits- und Telekommunikationsanwendungen verwendet.

Durch die von Microchip mitangebotene Entwicklungsumgebung ist die kostengünstige Softwareentwicklung und Softwareprüfung auch für Hobbyelektroniker möglich.

## Überblick

Die folgende Überblickstabelle zeigt, welche PIC-Prozessoren mit welcher integrierter Peripherie- und Speicheroption erhältlich sind.

| | | Clock | Memory | | | | | Peripherals | | | | | | | | Features | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Maximum Frequency of Operation (MHz) | Program Memory EPROM | Program Memory ROM | Program Memory EEPROM | Data Memory (Bytes) | Data EEPROM (Bytes) | Timer Module(s) † | Capture/Compare/PWM Module | Serial Port (s) (SPI/I²C, SCI) | Parallel Slave Port | Analog to Digital Converter (8-bit) | External Interrupts | Interrupt Sources | I/O | Voltage Range (Volts) | Number of Instructions | Packages |
| High-end | PIC17C42A | 25 | 2K | — | — | 128 | — | TMR0, TMR1, TMR2, TMR3 | — | SCI | — | — | Yes | 11 | 33 | 4.5 - 5.5 | 58 | 40-pin DIP, 44-pin PLCC 44 pin QFP |
| Mid-range | PIC16C64 | 20 | 2K | — | — | 128 | — | TMR0, TMR0, TMR2 | 1 | SPI/I²C | Yes | — | Yes | 8 | 33 | 2.5 - 6.0 | 35 | 40-pin DIP, 44-pin PLCC 44 pin QFP |
| | PIC16C71 | 16 | 1K | — | — | 36 | — | TMR0 | — | — | — | 4 ch | Yes | 4 | 13 | 3.0 - 6.0 | 35 | 18-pin DIP, 18-pin SOIC 44 pin QFP |
| | PIC16C74 | 20 | 4K | — | — | 192 | — | TMR0, TMR1, TMR2 | 2 | SPI/I²C, SCI | Yes | 8 ch | Yes | 12 | 33 | 2.5 - 6.0 | 35 | 40-pin DIP, 44-pin PLCC 44 pin QFP |
| | PIC16C84 | 10 | — | — | 1K | 36 | 64 | TMR0 | — | — | — | — | Yes | 4 | 13 | 2.0 - 6.0 | 35 | 18 pin DIP, 18 pin SOIC 20-pin SSOP |
| Base Line | PIC16C54 | 20 | 512 | — | — | 25 | — | RTCC | — | — | — | — | — | 0 | 12 | 2.5 - 6.25 | 33 | 18-pin DIP, 18-pin SOIC 20-pin SSOP |
| | PIC16C54A | 20 | 512 | — | — | 25 | — | RTCC | — | — | — | — | — | 0 | 12 | 2.5 - 6.25 | 33 | 18-pin DIP, 18-pin SOIC 20-pin SSOP |
| | PIC16CR54 | 20 | — | 512 | — | 25 | — | RTCC | — | — | — | — | — | 0 | 12 | 2.5 - 6.25 | 33 | 18-pin DIP, 18-pin SOIC 20-pin SSOP |
| | PIC16C55 | 20 | 1K | — | — | 25 | — | RTCC | — | — | — | — | — | 0 | 20 | 2.5 - 6.25 | 33 | 28-pin DIP, 28-pin SOIC 28-pin SSOP |
| | PIC16C56 | 20 | 1K | — | — | 25 | — | RTCC | — | — | — | — | — | 0 | 12 | 2.5 - 6.25 | 33 | 18-pin DIP, 18-pin SOIC 20-pin SSOP |
| | PIC16C57 | 20 | 2K | — | — | 72 | — | RTCC | — | — | — | — | — | 0 | 20 | 2.5 - 6.25 | 33 | 28-pin DIP, 28-pin SOIC 28-pin SSOP |
| | PIC16CR57A | 20 | — | 2K | — | 72 | — | RTCC | — | — | — | — | — | 0 | 20 | 2.5 - 6.25 | 33 | 28-pin DIP, 28-pin SOIC 28-pin SSOP |
| | PIC16C58A | 20 | 2K | — | — | 72 | — | RTCC | — | — | — | — | — | 0 | 12 | 2.5 - 6.25 | 33 | 18-pin DIP, 18-pin SOIC 20-pin SSOP |

***Tabelle 1:*** **Microchip - PIC - Family**

## Erläuterung

| | |
|---|---|
| TMRx | Timer |
| RTCC | Real Timer Clock Counter |
| SCI | Serial Communication Interface (USART) |
| SPI | Synchron Serial Port |
| I²C | Inter-IC-Bus |

Natürlich kann jeder digitale I/O-Pin des PIC als Ein- bzw. Ausgabeport geschaltet werden (siehe Befehl TRIS).

Als Grundtype der PIC-Familie kann man den PIC16C5x ansehen. Daher wird in dieser Mikrocontrollerbeschreibung zunächst die PIC16C5x-Familie beschrieben, welcher als Grundlage für weitere Prozessortypen dienen soll.

Die Tabelle 1 ist in 3 Gruppen geteilt.
Die **‘Base-Line-Familiy’** ist ein guter „Grundprozessor“, welcher die grundlegendste und kostengünstige Peripherie-Einheit beinhaltet. Die Spannungsversorgung kann bis zu 2,5 Volt absinken, was ideal für batteriebetriebene Steuerungen ist. Den PIC16C5x kann man sozusagen als Grundbaustein für die zwei weiteren Einteilungen ansehen.

Die **‘Mid-Range-Family’** bietet eine größere Vielzahl an intern vorhandener Peripherie (serielle Schnittstelle, I²C-Bus, interner Analog-Digital-Converter, größerer Speicher und Timermodule und Interrupteingänge).

Die **‘High-End-Family’** übertrifft dieses noch mit mehr I/O-Pins und etwas mehr interner Peripherie.

*Anmerkung: Im Originalaufsatz werden in 3 Abschnitten die Familien PIC16C5x, PIC16C71 und PIC16C84 besprochen. Der vorliegende Beitrag beschränkt sich auf die Familie PIC16C5x. Interessierte Leser finden sowohl die Beschreibungen (Datei PIC.DOC) als auch die Programme der beiden anderen Familien auf der Begleitdiskette. Der abschließende Teil „Entwicklungsumgebung“ ist wieder vollständig abgedruckt.*

# PIC16C5x

## Aufbau

Der grundsätzliche Aufbau des Mikrocontrollers kann man mit einem Blockschaltbild am einfachsten erklären.

Wie man in ***Abbildung 1*** erkennt, ist das EPROM (auch als One Time Programable-Ausführung [d.h. nur einmal programmierbar] ausgestattet), welches das Programm beinhaltet, welches gleich im Baustein implementiert ist.

Als Register dient der Block mit der Beschriftung GENERAL PURPOSE REGISTER FILE, welches je nach Prozessortype verscheiden groß ist, die Anzahl der General Purpose Register File beträgt minimal 24 und bei manchen Typen maximal 256. Hingegen kann man das ‘Arbeitsregister’ oder WORK-Register - W - als den Akkumulator ansehen, die dazugehörenden obligaten Flagbits befinden sich im STATUS-Register. Der interne Daten-’Arbeits’-Bus beinhaltet 8-Bits, hingegen ist die Programmdatenbreite 12 Bits, und der Adressierungsraum benötigt maximal 11 Bits. Die Breite des Programmadressierungsraumes ist vom Prozessortyp abhängig. Einen Stackbereich von 2 Rücksprungadressen (nur bei der PIC16C5x-Familie) bis zu 16 (abhängig vom Prozessortyp) dienen für strukturierte Programmabläufe. Die Ein/Ausgabeports ein 4 Bit und ein bzw. zwei (nur bei den Prozessoren PIC16C55/C57) 8 Bit (bis zu 33 I/O-Pins - abhängig vom Prozessortyp), welche als Ein- und Ausgabeport dienen können, sind für direkte Steuerung an die Umwelt verantwortlich. Die ALU (Arithmetic Logic Unit) ist eine 8-Bit-breite Operationseinheit, wobei die zweite Operationsgröße fast immer das W-(Work)-Register ist. Das Ergebnis kann in das W-Register bzw. in ein General-Purpose-Register-File (allgemein gesagt in ein Register) abgelegt werden. Ein Real-Time-Clock-Counter bzw. ein Watch-Dog-Timer befindet sich ebenfalls standardmäßig in PICs. Die Oszillatorbeschaltung kann auf 3 Arten realisiert werden (Achtung: Abhängig von der Prozessorspezifikation) (1) RC-Glied, (2) TTL-Oszillator, (3) Quarz-Ozillator.

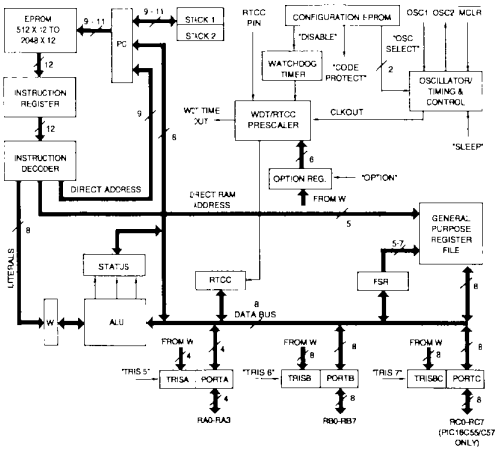


***Abb. 1:*** **Blockschaltbild der PIC16C5x-Familie**

Der Aufbau der PIC-Prozessoren basiert daher auf Havard-Architektur, welche auf Registerfiles basiert mit separatem Bus und Speicher für Daten und Instruktionen und mit einer RISC-Prozessorarchitektur.

Die PIN-Belegung der PIC16C5x-Familie ist daher minimal und sehr übersichtlich, die Gehäuseform ist hauptsächlich in DIL in zwei Varianten erhältlich. Das CERDIP bietet ein Löschfenster für den Programmspeicher - EPROM - an, hingegen besitzt PDIP - die billigere Variante - kein Löschfester für den EPROM-Programmspeicher daher auch die OTP-EPROM-Ausführung. Natürlich ist der PIC auch in SOIC- und SSOP-Gehäuseform erhältlich.

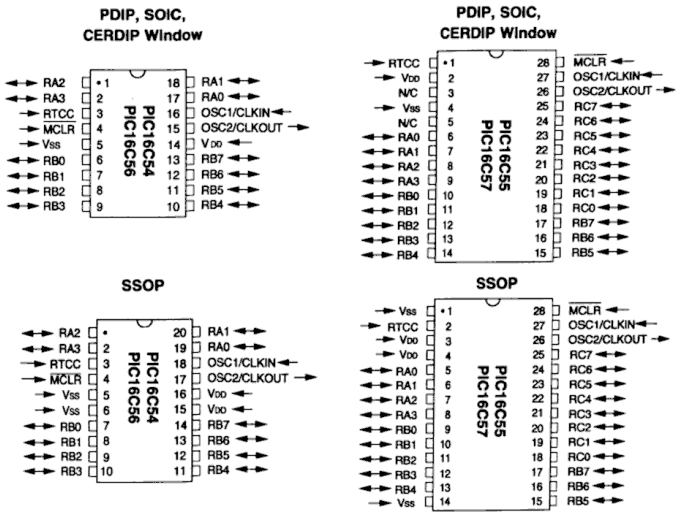


***Abb. 2:*** **PIN-Belegung**

## Buslos? - extern

Der PIC ist grob gesehen ein externer busloser Prozessor, d.h. daß der PIC keine Daten-, Adreß- und Steuerbusleitungen an die Umwelt ausgeführt hat, wie man es von der MSC51-Familie, Z80, etc. gewohnt ist (Ausnahme 87xx, o.ä.). Man kann jedoch mit einem gewissen Algorithmus der Ein-Ausgabe einen Bus 'emulieren' und somit einen seriellen oder parallelen Bus verwalten. Jedoch muß man sagen, wenn man mit einem PIC einen externen Bus verwalten möchte, ist dieses vielleicht eine Erweiterung der internen Peripherie-Möglichkeiten, aber die Arbeitsgeschwindigkeit nimmt bezüglich der 'Emulation' bei gleicher Taktrate ab.

## Die Fileregister

Die Fileregister, auch Register genannt, sind direkt mit Befehlen ansteuerbar, Ausnahmen siehe im Text.

Die Register 00 bis 07 sind bei den meisten Typen gleich.

Überblick der Register mit Adresse und Bedeutung
**00** indirect Addr. (8bits)
**01** RTCC
**02** PC (8bits schreib/lesbar, interne Verwaltung 12 Bits)
**03** Status (PA2|PA1|PA0|/TO|Z|DC|C)
**04** FSR
**05** Port A
**06** Port B
**07** Port C nur bei PIC16C54/56
**08** .. **0F** general purpose register files (8bits)
**10** .. **1F** gen. purpose Reg. bei allen Typen (Seite 0)
**30** .. **3F** gen. purpose Reg. nur bei PIC16C57 (Seite 1)
**50** .. **5F** gen. purpose Reg. nur bei PIC16C57 (Seite 2)
**70** .. **7F** gen. purpose Reg. nur bei PIC16C57 (Seite 3)

Indirekt Data Adressing (INDF)
INDF [00h]=(7) d7 d6 d5 d4 d3 d2 d1 d0 (0)
INDF ist kein physikalisch implementiertes Register. Es ist ideal als indirekte adressierter Zeiger. z.B.: ADD WF IND,W

Real Time Clock/Counter Register (RTCC)
RTCC [01h]=(7) d7 d6 d5 d4 d3 d2 d1 d0 (0)
Dieses Register kann wie ein gewöhnliches Register durch das Programm geladen und ausgelesen werden. Mit dem Setzen der OPTION-Registers kann es als Zähler mit externem Eingang - RTCC - oder mit dem Viertel des Oszillatortaktes angesteuert werden. (siehe Option-Register und Prescaler)

Program-Counter (PC)
PC [02h]=(7) d7 d6 d5 d4 d3 d2 d1 d0 (0)
Der PC generiert die Adresse für den maximal 2048x12-On-Chip-Eprom. Der PC wird nach einem RESET auf die höchstwertige Adresse gesetzt! Daher braucht man das eigentliche Programm nicht von der Adresse NULL starten, sondern von jeder beliebigen Adresse jedoch innerhalb der 9-Bit-Adressierung (nur bei PIC16C56/57, da der PIC16C54/55 nur 9-Bit PC hat.) also in der untersten Programmseite (siehe STATUS-Register). Diese Methode ist die einfachste.

Man kann natürlich den PC im Programm gezielt manipulieren, z.B. *ADDWF PC,1* , wobei PC ein EQU von 02 steht. Dieses kann verwenden um ein in Basic *ON..GOTO* bzw. *ON..GOSUB* bzw. in Borland Pascal ein eingeschränktes *CASE..OF* zu realisieren.

Das STATUS-Wort-Register
STATUS [3h]= (7) PA2 | PA1 | PA0 | /TO | /PD | Z | DC | C[RS1] (0)
Dieses Register beinhaltet arithmetische Statuse der ALU, Reset-Status und die Seitenadressierung des Programmspeichers.

Wird der Befehl *CLRF 02h* (= Löschen des Statusregisters) durchgeführt, so besitzt das Status-Register den Wert **000UU100**, wobei U als ungeändert steht.

***C****: CARRY|/BORROW Bit & DIGIT CARRY|BORROW-Bit*
Das CARRY-Bit ist ein Übertrag bei Additionsoperationen (ADDWF) und ein negiertes 'Ausleih'-Bit bei Subtraktionsoperationen (SUBWF). Es wird für die Rotierbefehle RRF und RLF als weiteres Bit verwendet.

```
; Programm: PRG1.ASM

   LIST P=16C54

   ORG 0h
Start MOVLW 0FFh ; W:=0FFh
   MOVWF 8       ; GPFR-8 := W

   MOVLW 01h     ; W:=1
                 ; W  F8  Z DC C
                 ; 01 FF  0 0  0
   ADDWF 8,1     ; F8:=F8+W
   MOVLW 01h     ; 01 00  1 1  1
   ADDWF 8,1     ; F8:=F8+W
   MOVLW 01h     ; 01 01  0 0  0
   SUBWF 8,1     ; F8:=F8-W
   MOVLW 01h     ; 01 00  1 1  1
   SUBWF 8,1     ; F8:=F8-W
   NOP           ; 01 FF  0 0  0
   ORG 1FF
   GOTO Start
   END
```

*Programmlisting 1:* **Beispiel für STATUS-Register-Optionen**

***DC****: DIGIT CARRY|/BORROW Bit*
Das Digit-Carry-Bit operiert auf die gleiche Weise wie das CARRY-Bit, jedoch nur zwischen den höheren und unteren Nibbles.

***Z:*** *ZERO Bit*
Das Zero-Bit oder Null-Bit signalisiert, ob das aktuelle Ergebnis nach einer Operation gleich Null ist.

***/PD****: POWER DOWN Bit* und ***/TO****: TIME OUT Bit*
Das /TO- und /PD-Bit kann als Reset-Überprüfung dienen.

| Ereignis | /TO | /PD | Bemerkung |
|---|---|---|---|
| Versogungsspg. einschalten | 1 | 1 | |
| WDT Timeout | 0 | X | kein Effekt an /PD |
| SLEEP Befehl | 1 | 0 | |
| CLRWDT Befehl | 1 | 1 | |

*Tabelle 2:* **/TO & /PD Ereignisse**

Nach einem Reset sind diese 2 Bits des Status-Register wie folgt gesetzt:

| /TO | /PD | RESET-Ursache |
|---|---|---|
| 0 | 0 | WDT 'Aufgeweckt' vom SLEEP-Befehl |
| 0 | 1 | WDT-Time-Out (nicht durch SLEEP verursacht) |
| 1 | 0 | /MCLR 'Aufgeweckt' von SLEEP-Befehl |
| 1 | 1 | Versorgung eingeschaltet |
| X | X | =LOW-Pulse an /MCLR-Eingang |

***Tabelle 3:*** **/PD & /TO nach Reset**

***File Reset Register*** *PA0/1*

PA0, PA1 PIC16C54/55 2 generelle Lese/Schreibbits
PIC16C56 PA0=0..Page 0 (000-1FF)
PA0=1..Page 1 (200-3FF)
PA1 .. generelles Lese/Schreibbit (siehe PA2)
PIC16C57 00 = Page 0 (000-1FF)
01 = Page 1 (200-3FF)
10 = Page 2 (400-5FF)
11 = Page 3 (600-7FF)

PA2..generelles Lese/SchreibBit, reserviert für zukünftige Anwendungen, d.h. der noch unbenutzte Bit kann als setz- und lesbarer Bit für diverse Programmaufgaben genutzt werden.

Generell kann man sagen, daß das STATUS-Byte die wichtigsten Flags beinhaltet.

File Select Register (FSR)
FSR [04h] = (7) d7|d6|d5|d4|d3|d2|d1|d0 (0)
Bei den PIC16C54/C55/C56 sind die Bits 0 bis 4 die 'Adresse' der Register, also 32 verfügbare Registeransteuerungen. Bits 5 bis 7 sind nur lesbar und beinhalten eine EINs. Wird das FSR-Register nicht für die indirekte Adressierung benötigt, so kann es als 5-Bit-langes generelles Lese/Schreibe-Register verwendet werden.

Nur beim PIC16C57 sind die Bits 5 und 6 als Auswahl der Seiten für die Datenbankseite zuständig, jedoch die unteren 16 Register sind auf allen Seiten der Bank physikalisch ident. Bit 7 des FSR ist nur lesbar und beinhaltet eine EINs.

EIN- und AUS-Gabeports PortA, PortB & PortC
PortA [05h] = (7) X|X|X|X|PA3|PA2|PA1|PA0 (0)
PortB [06h] = (7) PB7|PB6|PB5|PB4|PB3|PB2|PB1|PB0 (0)
PortC [07h] = (7) PC7|PC6|PC5|PC4|PC3|PC2|PC1|PC0 (0)
Die Ein- und Ausgaben des Prozessors an die Umgebung werden mittels Registern durchgeführt. Diese I/O- bzw. E/A-Register werden durch die MOV- bzw. Transportbefehle und registerarbeitenden Befehle (jene welche die E/A-Register manipulieren) angesprochen (z.B. MOVWF PortB, W; ANDWF 0Fh,f).

Bevor man jedoch Ein- oder Ausgaben vom Programm durchführen lassen kann, muß man dem Ein/Ausgabe-Interface mitteilen, ob es sich um einen Eingabe- oder Ausgabe-Pin (individuell) handelt. Hier wird im W-Register an derselbigen Position eine **0 für AUS**gabe und eine **1 für EIN**gabe positioniert. Dann mit dem Befehl TRIS und dessen Port-Adresse (5 bis 7) dem Interface die Richtungen mitgeteilen.

(zB. MOVLW 0h TRIS 6h)

*Port A:* Das Port A besteht auf nur 4 Bits des unteren Bytes (bei PIC16C5x).
*Port B:* Ein-8-Bit Register
*Port C:* Ein 8-Bit-Register nur bei PIC16C55/C57
Bei PIC16C54/C56 als General Purpose Register verwendbar.

OPTION Register
OPTION = (7) - | - | RTS | RTE | PSA | PS2 |PS1 |PS0 (0)
Mit dem OPTION-Register kann man den Real-Time-Clock-Counter konfigurieren.

*RTS - RTcc signal Source*
Echtzeit-Zähl-Signal-Quelle
0 .. Zähleingang = CLKOUT (¼ $f_{OSZ}$)
1 .. Zähleingang = RTCC-Eingang

*RTE - RTcc Signal Edge*
Echtzeit-Zähler-Signal-Flankenerkennung
0 .. Inkrement by LOW-HIGH-Flanke
1 .. Inkrement by HIGH-LOW-Flanke

*PSA - PreScale Assigment bit*
Vorteilerzuweisung
0 .. Anweisung von RTCC
1 .. Anweisung von WDT
*PS2,PS1,PS0 - PreScal Value*
Vorzählereinstellung
RTCC-Rate = $2^{PS[2:0]+1}$
WDT-Rate = $2^{PS[2:0]}$

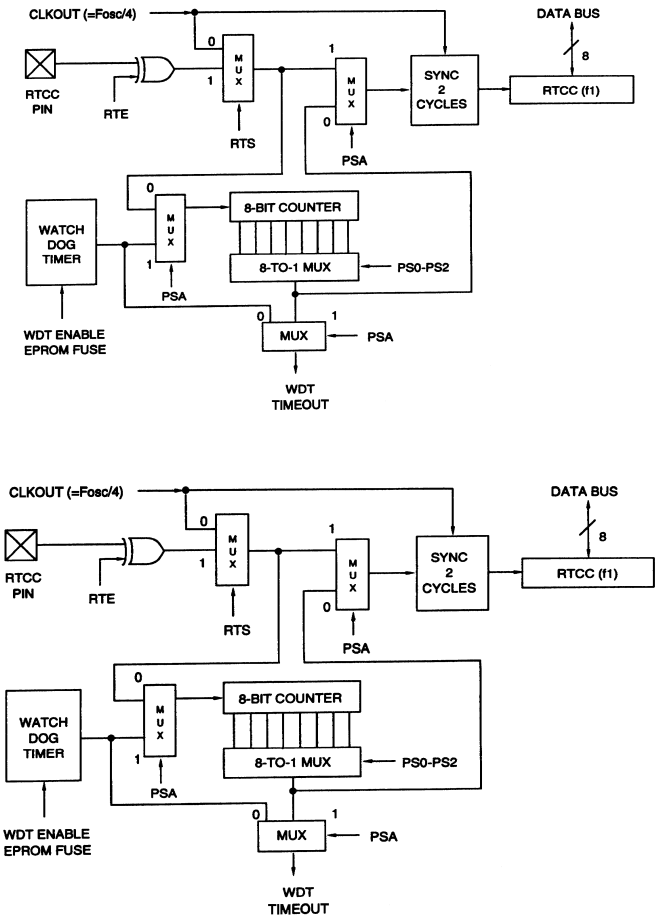


***Abb. 3:*** **Blockdiagramm v. RTCC/WDT Prescaler**

In dem folgenden Programmbeispiel wurde der RTCC-Teiler und die Länge des Programmes so gewählt, daß das Zählregister (im General Purpose Register File Nr. 8) und das RTCC-Register gleichen Inhalt vor dem Befehl 'NOP' haben.

```
; Programm: PRG2.ASM
LIST P=16C55
Start  ORG 0
   CLRF 8    ; Lösche GPRF 8h
   CLRF 1    ; Lösche RTCC-Zähler
   MOVLW 1      ; Option: --000001
         ; RTS=0
         ; RTE=0
         ; PSA=0
         ; PS=001 => RTCC-Rate 1:4
   OPTION    ; W -> Option-Register
Marke  INCF 8,1  ; F8 := F8 +1 (1 Zyklus)
   NOP    ;            (1 Zyklus)
   GOTO Marke;             (2 Zyklen)
         ;            ===
         ; Programm:    4 Zyklen
   ORG 1FFh
   GOTO Start
   END
```

***Programmlisting 2:*** Beispiel für RTCC

## Die Befehle

(Kurzfassung)
Mit nur 33 Befehlen (beim PIC16C5x, sonst 35 bzw. 55) ist der PIC ein sogenannter „SMALL-COMMAND-PROCESSOR“. Jedoch mit so wenige Befehlen muß man manchmal einige Tricks anwenden, um gewisse Programmabläufe zu realisieren.

***f*** Registerfile
***d*** Zielregister (d=0 -> W (Workfile) d->1 selbige Registerfile)
***d*** Bitstelle
***k*** 8-Bit-Konstante
***(x)*** Inhalt(e) von x
**->** Zuordnung
Die Befehlslänge beträgt 1 Zeile.

### Transportbefehle:

***MOVF f,d***
(f) -> d

Der Inhalt des Registers f wird in das in d angegebene Register geladen. Ist d=0, so ist das Zielregister W, bei d=1 ist es das ursprüngliche Register.

Statusbits: keine Zykluszeit: 1

***MOVWF f***
W -> f

Es werden die Daten, die im Register W stehen, in das Register f geladen.

Statusbits: keine Zykluszeit: 1

### arithmetische Operationen:

***ADDWF f,d***
(W + f) -> d

Addiert den Inhalt von W mit dem Inhalt von Register f. Ist d gleich 0, so wird das Resultat in W und bei 1 in das ursprüngliche Register f gespeichert.

Statusbits: C, CD, Z Zykluszeit: 1

***SUBWF f,d***
(f - W) -> d

Subtrahiert den Inhalt von W von dem Inhalt von Register f. Ist d gleich 0, so wird das Resultat in W und bei 1 in das ursprüngliche Register f gespeichert.

Statusbits: C, CD, Z Zykluszeit: 1

***DECF f,d***
(f - 1) -> d

Dekrementieren von Register f. Ist d gleich 0, so wird das Resultat in W und bei 1 in das ursprüngliche Register f gespeichert.

Statusbit: Z Zykluszeit: 1

***INCF f,d***
(f + 1) -> d

Inkrementieren von Register f. Ist d gleich 0, so wird das Resultat in W und bei 1 in das ursprüngliche Register f gespeichert.

Statusbit: Z Zykluszeit: 1

### logische Operationen:

***ANDWF f,d***
(W .AND. f) -> d

Logische UND-Verknüpfung zwischen den Inhalten von W und dem Register f. Ist d gleich 0, so wird das Resultat in W und bei 1 in das ursprüngliche Register f gespeichert.

Statusbits: Z Zykluszeit: 1

***ANDLW k***
(W .AND. k) -> W

Logische UND-Verknüpfung zwischen den Inhalten von W und der Konstanten k. Das Ergebnis wird in das W-Register gespeichert.

Statusbits: Z Zykluszeit: 1

***IORWF f,d***
(W .OR. f) -> d

Logische INKLUSIVE-ODER-Verknüpfung zwischen den Inhalten von W und dem Register f. Ist d gleich 0, so wird das Resultat in W und bei 1 in das ursprüngliche Register f gespeichert.

Statusbits: Z Zykluszeit: 1

***IORLW k***
(W .OR. k) -> W

Logische inklusive ODER-Verknüpfung zwischen den Inhalten von W und der Konstante k. Das Ergebnis wird in das W-Register gespeichert.

Statusbits: Z Zykluszeit: 1

***XORWF f,d***
(W .XOR. f) -> d

Logische EXKLUSIVE-ODER-Verknüpfung zwischen den Inhalten von W und dem Register f. Ist d gleich 0, so wird das Resultat in W, und bei 1 in das ursprüngliche Register f gespeichert.

Statusbits: Z Zykluszeit: 1

***XORLW k***
(W .XOR. k) -> W

Logische exklusive ODER-Verknüpfung zwischen den Inhalten von W und der Konstanten k. Das Ergebnis wird in das W-Register gespeichert.

Statusbits: Z Zykluszeit: 1

***COMF f,d***
/f -> d

Der Inhalt vom Register f wird negiert, d.h. es wird jedes einzelne Bit des f-Registers invertiert, und je nach Wertigkeit von d wird das Ergebnis in das W (d=0) bzw. in das Ursprungsregister f (d=1) gespeichert.

Statusbits: Z Zykluszeit: 1

### Lösch- und Setzbefehle:

***CLRF f***
00h -> f

Löscht den Inhalt von Register f.

Statusbits: Z Zykluszeit: 1

***CLRW***
00h -> W

Löscht das W-Register.

Statusbits: Z Zykluszeit: 1

***MOVLW k***
k -> W

Die 8-bit-lange Konstante k wird in das W-Register geladen.

Statusbits: keine Zykluszeit: 1

***BCF f,b***
0 -> f(b)

Das b-te Bit (b=0..7) des Registers f wird gelöscht, d.h. auf 0 gesetzt.

Statusbits: keine Zykluszeit: 1

***BSF f,b***
1 -> f(b)

Das b-te Bit (b=0..7) des Registers f wird gesetzt, d.h. auf 1 gesetzt.

Statusbits: keine Zykluszeit: 1

### Befehle mit Bedingungen:

***BTFSC f,b***
SKIP IF f(b)=0

Ist das b-te Bit (b=0..7) gleich 0, so wird der im Programm stehende nächste Befehl übersprungen.

Statusbits: keine Zykluszeit: 1 bzw. 2

***BTFSS f,b***
SKIP IF f(b)=1

Ist das b-te Bit (b=0..7) gleich 1, so wird der im Programm stehende nächste Befehl übersprungen.

Statusbits: keine Zykluszeit: 1 bzw. 2

***DECFSZ f,d***
(f - 1) -> d ; SKIP IF RESULT = 0

Der Inhalt des Registers f wird dekrementiert. Ist das Ergebnis der Dekrementierung bei der nächsten Abarbeitung gleich 0, so wird der nächste Befehl übersprungen.

Statusbits: keine Zykluszeit: 1 bzw. 2

***INCFSZ f,d***
(f + 1) -> d ; SKIP IF RESULT = 0

Der Inhalt des Registers f wird inkrementiert. Ist das Ergebnis der Dekrementierung bei der nächsten Abarbeitung gleich 0, so wird der nächste Befehl übersprungen.

Statusbits: keine Zykluszeit: 1 bzw. 2

## Rotierbefehle

***RLF f,d***
f<n> -> d<n+1> {für n=0..6}, f<7> -> C, C -> d<0>

Der Inhalt des Registers f wird um ein Bit nach LINKS durch das CARRY verschoben. Ist d=0, so ist das Zielregister W, bei d=1 ist es das ursprüngliche Register f.

Statusbits: keine Zykluszeit: 1

***RRF f,d***
f<n> -> d<n-1> {für n=1..7}, f<0> -> C, C -> d<7>

Der Inhalt des Registers f wird um ein Bit nach RECHTS durch das CARRY verschoben. Ist d=0, so ist das Zielregister W, bei d=1 ist es das ursprüngliche Register f.

Statusbits: keine Zykluszeit: 1

## Tauschbefehl:

***SWAP f,d***
f<0:3> -> d<4:7>, f<4:7> -> d<0:3>

Die untere und obere Nibbles des Registers werden getauscht. Das Endregister kann mit d=0 als W-Register oder mit d=1 als Ursprungsregister ausgewählt werden

Statusbits: keine Zykluszeit: 1

## Sprungbefehle:

***GOTO k***
k -> PC<8:0>; PA2, PA1, PA0->PC<11:9>

Die unteren neun Bits werden von den direkt eingegeben Wert bestimmt. Die oberen drei Bits werden vom den Bits PA2, PA1 und PA0 des STATUS-Register-Bit bestimmt. Die Konstante k wird in den Programmzähler geladen und verursacht einen Sprung im Programm.

Statusbits: keine Zykluszeit: 2

***CALL k***
PC+1 -> TOS; k -> PC>7:0>; '0'->PC>8>;

PA2, PA1, PA0 -> PC<11:9>

Subroutinenaufruf. Als erstes die Rücksprungadresse (PC+1) wird in den Stack-Speicher gespeichert. Das Byte wird in den PC Bit (0 bis 7) geladen. Der achte Bit des PC wird gelöscht. PA<2:0> wird in den PC<11:9> geladen.

Statusbits: keine Zykluszeit: 2

***RETLW k***
k -> W; TOS -> PC

Das W-Register wird mit der 8-bit-langen Konstanten k geladen. Der Programm Counter wird mit dem oberen Stack-Inhalt, welcher die Return-Adresse beinhaltet, geladen.

Statusbits: keine Zykluszeit: 2

## Steuerbefehle:

***TRIS f***
W -> TRIS register f

TRIS-Register f (f=5,6 oder 7) wird mit dem Inhalt des W-Registers geladen. Eine 0 bedeutet eine AUS-Gangsschaltung, eine 1 EIN-Gangsschaltung.

Statusbits: keine Zykluszeit: 1

***CLRWDT***
00h -> WDT; 0 -> WDT prescaler

CLRWDT-Befehl setzt den Watchdog-Timer zurück. Dieser Befehl setzt ebenfalls den Prescaler des WDT zurück. Die Statusbits /TO und /PD werden gesetzt.

Statusbits: 1 -> /TO; 1-> /PD Zykluszeit: 1

***SLEEP***
0 -> PD; 1 -> TO

Das POWER-DOWN-Status-Bit (/PD) wird gelöscht. TIME-OUT-Bit (/TO) wird gesetzt. Der Watchdog-Timer und dessen Prescaler werden gelöscht. Dieser Prozeß fällt in den SLEEP-Modus und schaltet den Oszillator aus.

Statusbits: /TO; /PD Zykluszeit: 1

***OPTION***
W -> OPTION

Der Inhalt des W-Registers wird in das OPTION-Register geladen.

Statusbits: keine Zykluszeit: 1

***NOP***
keine Operation

Statusbits: keine Zykluszeit: 1

# Takt

Takt-Typen
Die PIC16CXX-Serie ist in vier Taktvarianten verfügbar.

Quarz-Oszillator
Die PIC16C5X-***XT***, -***HS*** oder -***LP*** benötigen einen Kristall oder Keramik-Schwingquarze

Die Typenbezeichnungserweiterung bedeuten:

***XT*** = Standard (bis ca. 4MHz)
***HS*** = Hochgeschwindig (bis ca. 20Mhz)
***LP*** = Niedergeschwindigkeitsschwingquarz (bis ca. 32kHz)

| Osc-Type | Quarz-Type | Frequenz | $C_{OSC1-GND}$ | $C_{OSC2-GND}$ |
|---|---|---|---|---|
| LP | Kristall | 32kHz | 15pF | 15pF |
| XT | Kristall | 100kHz | 15-30pF | 300-300pF |
| XT | Kristall | 200kHz | 15-30pF | 100-200pF |
| XT | Kristall | 455kHz | 15-30pF | 100-200pF |
| XT | Keramik | 455kHz | 150-330pF | 150-300pF |
| XT | Kristall | 1MHz | 15-30pF | 15-30pF |
| XT | Kristall | 2MHz | 15pF | 15pF |
| XT | Keramik | 2MHz | 20-330pF | 20-330pF |
| XT | Kristall | 4MHz | 15pF | 15pF |
| XT | Keramik | 4MHz | 20-330pF | 20-330pF |
| HS | Kristall | 4MHz | 15pF | 15pF |
| HS | Kristall | 8MHz | 15pF | 15pF |
| HS | Keramik | 8MHz | 20-200pF | 20-200pF |
| HS | Kristall | 20MHz | 15pF | 15pF |

***Tabelle 4:*** **Quarz-Oszillatoren**

RC-Oszillator
Für zeitunempfindliche Anwendungen ist der PIC16C5X mit der Typenerweiterungsbezeichnung ***RC*** ausreichend. Der Widerstand $R_{ext}$, welcher zwischen der Versorgungsleitung und dem Eingang OSC1 geschaltet ist, soll einen Wertebereich **über** 2,2kOhm haben, da sonst die

RC-Schaltung unstabil wird. Man sollte daher einen Widerstandswert zwischen 5kOhm und 100kOhm wählen. Der Kondensator $C_{ext}$, welcher zwischen OSC1 und Masse (0Volt) geschaltet ist, sollte aus Stabilitätsgründen mindestens 20pF haben.

| $C_{ext}$ | $R_{ext}$ | $F_{osz}$ (bei 5V, 25°C) | Toleranz |
|---|---|---|---|
| 20pF | 3,3kΩ | 4,973MHz | ±27% |
| 20pF | 5kΩ | 3,82MHz | ±21% |
| 20pF | 10kΩ | 2,22MHz | ±21% |
| 20pF | 100kΩ | 262,15kHz | ±31% |
| 100pF | 3,3kΩ | 1,63MHz | ±13% |
| 100pF | 5kΩ | 1,19MHz | ±13% |
| 100pF | 10kΩ | 648,64kHz | ±18% |
| 100pF | 100kΩ | 71,56kHz | ±25% |
| 300pF | 3,3kΩ | 660,0kHz | ±10% |
| 300pF | 5kΩ | 484,1kHz | ±14% |
| 300pF | 10kΩ | 267,63kHz | ±15% |
| 300pF | 100kΩ | 29,44kHz | ±19% |

**Tabelle 5:** RC-Takt

Die durch vier geteilte Oszillatorfrequenz ist an dem Ausgang OSC2 bzw. CLKOUT verfügbar.

externe Takteingang
Die einfachste „Betaktung" des PICs ist es einen externen Taktgenerator bzw. Taktschaltung an dem Eingang OSC1 zu legen, dieses ist bei den Typen HS, XT und LP möglich.

## WATCHDOG TIMER (WDT)

Der Watchdog-Timer ist durch einen freilaufenden RC-Oszillator, welcher durch keine externe Komponente beeinflußbar ist, realisiert. Durch Ausführen eines SLEEP-Befehls wird die Taktoszillation gestoppt. Eine WDT-Auszeit generiert ein Bausteinresetzustand. Der WDT kann durch eine Null in einem speziellen Bit im EPROM bei der Programmierung ausgeschaltet werden.

WDT Periode
Der WDT hat eine Nenn-"Auszeit"-Periode von (typisch) 18ms, laut Datenbuch von 9ms bis 30ms (ohne Vorteiler). Wird eine längere „Auszeit" benötigt, so kann ein maximales Verhältnis von 1:128 im OPTION-Register eingestellt werden. Dieses kann eine maximale Zeit typisch 2,304s jedoch zwischen 1,152s bis 3,84s bewirken.

Der CLRWDT- und der SLEEP-Befehle löschen den Watch-Dog-Timer (WDT) und den Zähler-Vorteiler (Prescaler). Ist der WDT zugewiesen, so wird eine Auszeit verhindert, und ein Reset wird ausgelöst.

Das /TO-Bit im STATUS-Register wird durch einen WDT-Auszeit gelöscht. Die WDT-Periode hat eine Zeitperiode von ca. 18ms (typischer Wert).

Ein WDT-Outtime bzw. eine WDT-Auszeit kann man als Interruptquelle bzw. Programmabbruch ansehen.

Abhängigkeit der Länge der WDT-Auszeit
Die WDT-Zeit ist abhängig von der Temperatur (-55..+125°C) und der angelegten Versorgungsspannung ($V_{DD}$=0..+7,5V).

| WDT | $V_{DD}$ | | | | | |
|---|---|---|---|---|---|---|
| T | 3V | 4V | 4,5V | 5V | 5,5V | 6V |
| -40°C | 18,5ms | 10ms | 9,7ms | 9ms | 9,25ms | 8,75ms |
| 0°C | 21,9ms | 13,75ms | 13,75ms | 12,5ms | 12,2ms | 11,25ms |
| 25°C | 24,1ms | 20,6ms | 20ms | 18ms | 17,5ms | 16,25ms |
| 70°C | 36ms | 30,6ms | 28,75ms | 27ms | 25ms | 23,75ms |
| 85°C | 37ms | 32,5ms | 33,2ms | 38ms | 26,9ms | 26,7ms |

***Tabelle 6:*** **WDT-Zeitperiode abhängig von Temperatur und Versorgungsspg.**

## Bereitschaftsmodus (Power Down Mode -PDM [Sleep])

Der Bereitschaftsmodus (PDM) wird durch einen SLEEP-Befehl initialisiert.

Falls ein SLEEP-Befehl durchgeführt worden ist, wird der WDT gelöscht, jedoch bleibt erhalten. Das /PD-Bit im STATUS-Register wird gelöscht und das /TO-Register wird gesetzt. Der Oszillatortreiber wird ausgeschaltet, und die Pegel der Ein-Ausgabeports werden beibehalten, welche sie vor dem Ausführungsbefehl hatten (Low, High).

Die Ein/Ausgänge sollten in diesem Modus in den niedrigsten Stromverbrauch geschaltet werden. Ein hochohmiger Anschluß sollte mit einem Pull-Up- bzw. Pull-Down-Widerstand ausgestattet werden, um Schwingungen zu verhindern.

Der /MCLR-Eingang muß eine maximale Eingangsspannung von $V_{IHMC}$ aufweisen.

| Signal | Wertigkeit | Symbol | Min. | Max. | Einheit |
|---|---|---|---|---|---|
| $/M_{CLR}$ | Low-Input | $V_{ILMC}$ | $V_{SS}$ | 0,15 $V_{DD}$ | V |
| | High-Input | $V_{IHMC}$ | 0,85 $V_{DD}$ | $V_{DD}$ | V |

***Tabelle 7:*** **Spannungspegel an /MCLR**

„Aufwecken" (Wake Up)
Der Baustein kann durch einen Watchdog-Timer-Auszeit, wenn der WDT-Eprombit gesetzt wurde, oder durch einen externen Impuls an /MCLR-Eingang ausgelöst werden. In beiden Fällen geht der PIC16C5x in den RESET-Modus für eine Periode des Oscillator-Start-up-Timer - $t_{OST}$ - bevor das 'normale' Programm abgearbeitet wird.

Das /PD-Bit im STATUS-Register, welches wieder gesetzt (/PD=1) wird, (nachdem es beim SLEEP-Befehl gelöscht (/PD=0) wurde), kann verwendet werden, um zu erkennen, daß der Prozessor eingeschaltet worden ist oder 'aufgeweckt' von einem Bereitschaftsmodus (PDM) worden ist.

Das /TO-Bit im STATUS-Register kann Auskunft geben, ob der Prozessor durch ein Signal an /MCLR (hier ist /TO=1) oder durch eine Watchdog-Auszeit (hier ist /TO=0) 'aufgeweckt' worden ist.

## Oscillator Start-Up Timer (OST)

Taktoszillatoren, welche mit Kristall- oder Keramikquarze aufgebaut sind, benötigen eine gewisse Einschwingzeit (= Start-Up-Time) für eine stabile Taktschwingung. Ein im Chip eingebaute Einschaltverzögerung haltet den Baustein in einem RESET-Zustand für ca. 10ms nach dem erreichen eines Spannungswertes $V_{IHMC}$ am /MCLR-Eingang. Eine Resetschaltung mit einem RC-Netzwerk ist daher in den meisten Fällen an dem Eingang /MCLR nicht notwendig. (siehe Power-On-Reset)

Der OST wird mit dem Watchdog-Auszeit getriggert, was für das verwenden des „Aufwecken" vom SLEEP-Befehl wichtig ist.

Der OST ist für Niedrigfrequenzschwingquarze, die eine höhere Einschwingzeit als ca. 18ms besitzen nicht geeignet.

$T_{OST}$ = 9ms(Minimum) bis 30ms(Maximum) typisch 18ms

automatischer Einschaltreset (Power-On-Reset [POR])
Der PIC16C5x besitzt auf dem Chip eine automatische Einschaltresetschaltung, welche einen Reset bei (den meisten) Einschaltsituationen der Versorgungsspannung auslöst, wenn man den /MCLR-Eingang auf $V_{DD}$-Potential (positive Versorgungsspannung) hält.

Der am Chip befindliche Einschaltresetmechanismus arbeitet garantiert, wenn die Anstiegsflanke der Versorgungsspannung kleiner als 0,05V/ms ist und von 0V beginnt. Der POR ist nicht für Niedrigfrequenzen mit höheren Einschwingzeit als etwa 18ms geeignet.

Aufbauend auf dieser Prozessorinformation kann nun auf die zwei weiteren Prozessoren der BASE-LINE (siehe Abb. 1) vorgehen.

# Entwicklungsumgebung

## Assembler

Der Assembler von Microchip (MPASM = Microchip Processor AsSeMbler) ist ein universaler Assembler für alle Microchip-Prozessoren. Die Handhabung des Assembler sowie die Programmierung der mnemonischen Quellprogramme ist sehr einfach. Neben den mnemonischen Befehlen stehen dem Programmierer noch (ca.) 39 Assemblerbefehle zur Verfügung.

### Der Assembleraufruf

X:>*MPASM* [/<Option>[,/Option>...)) <file_name>

Einige Optionen kurz erklärt:

**/h**, **/?** zeigt die Optionserklärung an.
**/l**± erzeugt eine (+) / keine (-) Dokumentationsdatei (opt. +)
**/e**± erzeugt eine (+) / keine (-) Fehlerdatei (optional +)
**/p**{xx} Wählen der Prozessortype
PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C71, PIC16C84, PIC17C42, PIC16C58, PIC16C64
**/x**± erzeugen einer (+) / keiner (-) Cross-Referenz-Liste (optional -)

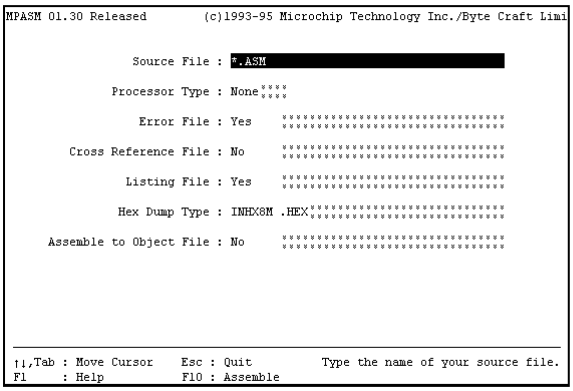


***Abb.8:*** **Maske des MPASM**

Wenn man nur MPASM ohne Aufrufparameter eingibt, so kann in einer Maske der Dateiname und einige Optionen ausgewählt werden.

### Die wichtigsten Dateitypen

Der Assembler erzeugt gewisse Dateien, die hier kurz erklärt werden sollen.

Die Quelldatei -**.ASM**
Die Quelldatei, welcher in einen Texteditor geschrieben ist, und eine Dateiendung mit ASM besitzt, beinhaltet das mnemonische Programm mit den Assemblerbefehlen.

Die Fehlerdatei -**.ERR**
Die Fehlerdatei (bzw. ERROR-Datei), welche vom Assembler erzeugt wurde (mit der Endung ERR) ist eine ASCII-Datei, welche die Fehler während Assemblierung mitprotokolliert. Es wird die Art der Fehler und Zeilen bekanntgegeben.

Beispiel: PRG4.ERR

```
Message PRG4.ASM 59 : Argument out of range.  Least significant bits used.
Message PRG4.ASM 68 : Argument out of range.  Least significant bits used.
Message PRG4.ASM 73 : Argument out of range.  Least significant bits used.
Message PRG4.ASM 78 : Argument out of range.  Least significant bits used.
```

Die Dokumentationsdatei -**.LST**
Die Dokumentationsdatei, welche vom Assembler erzeugt wurde (mit der Endung LST) beinhaltet das ganze Programm mit den Mnemonikcodes, deren Adresse und hexadezimalen Adressen.

Beispiel: PRG1.LST

```
MPASM 01.21 Released            PRG1.ASM   3-28-1996
22:34:07         PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ; Programm: PRG1.ASM
                     00002
                     00003         LIST P=16C54
                     00004
0000                 00005         ORG 0h
0000 0CFF            00006 Start   MOVLW 0FFh ; W:=0FFh
0001 0028            00007         MOVWF 8     ; GPFR-8 := W
                     00008
0002 0C01            00009         MOVLW 01h  ; W:=1
                     00010                      ; W  F8  Z DC C
                     00011                      ; 01 FF  0 0  0
0003 01E8            00012         ADDWF 8,1  ; F8:=F8+W
0004 0C01            00013         MOVLW 01h      ; 01 00  1 1  1
0005 01E8            00014         ADDWF 8,1  ; F8:=F8+W
0006 0C01            00015         MOVLW 01h      ; 01 01  0 0  0
0007 00A8            00016         SUBWF 8,1  ; F8:=F8-W
0008 0C01            00017         MOVLW 01h      ; 01 00  1 1  1
0009 00A8            00018         SUBWF 8,1  ; F8:=F8-W
000A 0000            00019         NOP            ; 01 FF  0 0  0
01FF                 00020         ORG 1FF
01FF 0A00            00021         GOTO Start
                     00022         END

MPASM 01.21 Released            PRG1.ASM   3-28-1996  22:34:07
PAGE  2

SYMBOL TABLE
  LABEL                             VALUE

Start                             00000000
__16C54                           00000001


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXX----- ---------------- ---------------- ----------------
0040 : ---------------- ---------------- ---------------- ----------------
0180 : ---------------- ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.


Errors   :    0
Warnings :    0
Messages :    0
```

Das codierte Programm -**.COD**
Das codierte Programm mit der Endung COD beinhaltet (unlesbar) die Symbole und Daten.

Das assemblierte Programm in Intel-HEX-Format -**.HEX**
Der vom Assembler erzeugte Code im Intel-HEX-Format, dient zur Programmierung des PIC-Microchip-Prozessors.

Beispiel: PRG1.HEX

```
:10000000FF0C2800010CE801010CE801010CA8001C
:06001000010CA800000035
:0203FE00000AF3
:00000001FF
```

### Assembler-Befehle

Die 39 Befehle erlauben das Mnemonikprogramm komfortabel zu gestalten.

Überblick (ohne Erläuterung) der Assemblerbefehle
`CBLOCK, CONSTANT, DATA, DB, #DEFINE, DW, ELSE, END, ENDC, ENDIF, ENDM, ENDW, EQU, ERROR, EXITM, EXPAND, FILL, IF, IFDEF, IFNDEF, INCLUDE, LIST, LOCAL, MACRO, MESSG, NOEXPAND, NOLIST, ORG, PAGE, PROCESSOR, RADIX, RES, SET, SPACE, SUBTITLE, TITLE, #UNDEFINE, VARIBALE, WHILE`

Die wichtigsten Assemblerbefehle
***PROCESSOR***
Wählen der Prozessortype im Assemblerprogramm.

Syntax: PROCESSOR <Type>
Type = PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C71, PIC16C84, PIC17C42, PIC16C58, PIC16C64
Beispiel: PROCESSOR 16C54

***TITLE***
Der Text nach diesem Befehl, in zwischen zwei " " (double quotes) eingeschlossen (maximal 60 Zeichen), wird in der Dokumentationsdatei (LST) in den ersten Zeile gedruckt.

Syntax: TITLE "<Titel_Text>"
Beispiel: TITLE "Motorsteuerung Version 5.0"

***SUBTITLE***
siehe TITLE

Syntax: SUBTITLE "<Untertitel_Text>"
Beispiel: SUBTITLE "Diagnostikprogramm"

***LIST***
Der List-Befehl kann einige Parameter direkt im ASM-Programm einstellen.

Syntax: LIST [<list_option>, ... , <list_option>]

Die wichtigsten Optionen in Kürze:
C=nnn setzen der Zeichenbreite (optional 80)
N=nnn setzen der Zeile-pro-Seite (optional 59)
P=<Typ> Wählen der Prozessortype
PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C71, PIC16C84, PIC17C42, PIC16C58, PIC16C64
R=nnn setzen des Zahlensystems
HEX (16), DEC (10), OCT (8) (optional HEX)

Beispiel: LIST p=16C54

***EQU***
Der Symbolname wird einer Konstanten, welche eine Zahl darstellt, im ganzen Programm zugewiesen.

Syntax: <Symbol-Name> EQU <Konstante>
Beispiel: Vier EQU 4

***#DEFINE***
Der Befehl DEFINE weist eine Zeichenkette einen Symbolnamen zu.

Syntax: #DEFINE <Symbol-Name> <Zeichenkette>
Beispiel: #DEFINE IntCon,3

***ORG***
Dieser Befehl setzt den Programmzähler auf die "neue" Adresse <Adresse>. Dieses kann gleich mit Setzen einer Sprungmarke <Label>, welche maximal 31 Zeichen lang sein darf, (GOTO, CALL) verbunden werden.

Syntax: <Label> ORG <Adresse>
Beispiel: Init ORG 0010h

***END***
Beendet das Assemblerprogramm (Mnemonikprogramm).

Darstellung von Zahlen im Assemblerprogramm
Wenn das Zahlensystem nicht explizit mit den Befehlen RADIX, dem Befehl LIST r=nnn oder der Option des MPASM-Aufrufes vereinbart wurde, so gilt das hexadezimale Zahlensystem im ganzen Mnemonikprogramm.

***Dezimalzahlen*** [0,1,2,3,4,5,6,7,8,9]
Darstellung: **D'**<Zahlen>**'**
Beispiel: D'123'

***Hexadezimalzahlen*** [0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F]
Darstellung: **H'**<Zahlen>**'**
Beispiel: H'0FF'

***Oktalzahlen*** [0,1,2,3,4,5,6,7]
Darstellung: **O'**<Zahlen>**'**
Beispiel: O'176'

***Binärzahlen*** [0,1]
Darstellung: **B'**<Zahlen>**'**
Beispiel: B'10101010'

***Zeichen*** [0,..,9,A,..,Z,a,..,z]
Darstellung: **'**<Zeichenkette>**'** oder **A'**<Zeichenkette>**'**
Beispiel: **A**'Hallo'

## Aufbau des Mnemonikprogramms mit Assemblerbefehlen

Der Aufbau eines Mnemonikprogramm sollte in der Art und Weise erfolgen, wie es in den Programmbeispielen im Text (1 bis 6, 6a, 6b) gezeigt worden ist.

# Simulator

Der Microchip Prozessorsimulator (MPSIM=Microchip Prozessor SIMulator) ist ein guter Ersatz für ein Monitorprogramm. Dieser Simulator kann zwar keine Echtzeitsimulation, wie es bei dem Microchip-Emulator der Fall ist, durchführen, jedoch ist er eine gute Kontrolle für das ganze Programm oder Teilen davon.

Es können alle Register, der Stack und beim 16C84 sogar das EEPROM angezeigt und manipuliert werden.

Nur eine kurze Einarbeitungszeit ist notwendig, um mit Hilfe dieses Simulators alle PIC-Programme zu testen.

## Der Programmaufruf

Der Programmaufruf ist sehr einfach.
X:>*MPSIM*

## Der Umgang mit MPSIM

Die Bildschirmmaske des Simulationsprogramm wird in 3 Bereiche geteilt:

Die *Titelleiste*, welche den Programmnamen, das Zahlensystem, den Prozessortyp, die Laufzeit und die Zyklenanzahl angibt.

Der *Überwachungsmonitor*, welcher den Inhalt der veränderbaren Register anzeigt.

Der *Kommandobereich*, welcher als Schaltzentrale für den Simulator dient. Hier werden einzelne Befehle mit bzw. ohne Optionen für den Simulationsbetrieb eingegeben.

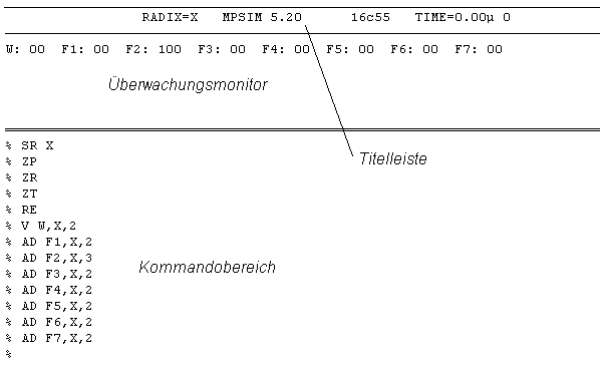


***Abb.9:*** **Bildschirmmaske des MPSIM**

## Die Simulatorbefehle

Dem Anwender stehen (ca.) 64 Befehle zur Verfügung.

Darstellung von Register und anderes
Register werden hier üblich mit F und anschließend der Registeradresse angegeben. Die Ports (PA, PB, PC) werden hier mit Register (RA, RB, RC) angeben. Das Eingabeformat (Zahlensystem) ist das Standardzahlensystem.

Die wichtigsten Befehle
*Hilfe anzeigen:* ***H***
Syntax: ***H***
Anzeige der einzelnen Befehle mit deren Optionen.

*Laden eines Programmes:* ***LO***
Syntax: ***LO*** *Dateiname*
Dieser Befehl lädt eine Datei (konkret eine Dokumentationsdatei (LST), wenn nicht anders angegeben) in den Simulator.

Bsp: *LO PRG6*

*Anzeigen des Programmes:* ***DI***
Syntax: ***DI*** *[Anfangsadresse [,Endadresse]]*
Zeigt das Programm im Kommandofenster (auch in Mnemonikcode) von der Anfangsadresse (falls angegeben), bis zur Endadresse (falls Anfangsadresse angegeben wurde). Ist diese Anfangsadresse nicht angegeben, so wird das Programm fortlaufend angezeigt.

Bsp: *DI* .. Zeigt Programmcodes 00h-0Ah, 0Bh-15h, 16h-20h,...
*DI 3* .. Zeigt Programmcodes von 03h bis 0Dh
*DI 4,6* .. Zeigt Programmcodes von 04h bis 06h

*Dem Überwachungsmonitor ein Register zufügen:* ***AD***
Syntax: ***AD*** *Register [,Zahlensystem[,Stellen]]*
Fügt dem Überwachungsmonitor ein Register in dem Zahlensystem (nur wenn angegeben *B*inär *H*exadezimal *D*ezimal *O*ktal) und deren maximale Stelle (nur wenn dieses und Zahlensystem angegeben) hinzu.

Bsp: *AD F9* .. Fügt dem Überwachungsmonitor das Register F9 in dem Standardzahelnsystem hinzu.
*AD F9,B,4* .. Fügt dem Überwachungsmonitor das Register F9 im binären Zahlensystem der ersten 4 Bit an.

*Aus dem Überwachungsmonitor ein Register löschen:* ***DV***
Syntax: ***DV*** *Register*
Löscht das angegebene Register aus der Anzeige des Überwachungsmonitors.

Bsp: *DV F2*

*Einen Breakpoint setzen:* ***B***
Syntax: ***B*** *Adresse* bzw. ***B*** *Register [Bedingung Zahlenwert]*
Bricht die Simulation NACH Exekution einer bestimmten Adresse oder einem bestimmten Register mit einer Bedingung (=,>,>,>=,<=,!=) und einem Zahlenwert ab.

Bsp: *B 0100h* .. Abbruch nach Exekution des Befehles in 0100h
*B Marke* .. Abbruch nach Exekution des Befehles in Marke
*B F2 > 80* .. Abbruch wenn Register 80 den Wert 80 überschreitet.

*Einen Breakpoint löschen:* ***BC***
Syntax: ***BC*** , ***BC*** *Adresse* bzw. ***BC*** *Register*
Löscht alle Breakpoints, wenn keine speziellen Adressen und Register angegeben wurden oder einen bestimmt Breakpoint, welcher durch den Registernamen oder die Adresse angegeben wurde.

Bsp: *BC* .. löscht alle Breakpoints
*BC F2* .. Löscht Breakpoint mit Bedingung mit F2
*BC 0010h* .. Löscht Breakpoint in Zeile 0010h

*Die Breakpoints anzeigen:* ***DB***
Syntax: ***DB***
Der Simulator gibt eine Liste über alle Breakpoints aus.

*Ein Programm starten:* ***E***
Syntax: ***E*** bzw. ***E*** *Adresse*
Dieser Befehl exekutiert das Programm ab dem momentanen Standpunkt (Programmzähler), wenn keine Adresse angegeben wurde, oder startet das Programm bei einer bestimmten Adresse.

Bsp: *E* .. Startet das Programm ab momentanen Adresse (PC)
*E 06* .. Startet das Programm bei der Adresse 06h.

*Ein Programm vom Anfang starten:* ***GO***
Syntax: ***GO***
Dieser Befehl realisiert einen POWER-ON-RESET und startet das Programm.

*Einzelbefehlschrittfolge:* ***SS***
Syntax: ***SS*** bzw. ***SS*** *Adresse*
Dieser Befehl exekutiert einen einzelnen Befehl, wenn angegebenen, an einer bestimmten Adresse.

*Ein Registerinhalt ändern:* ***F***
Syntax: ***F*** *Register*
Dieser Befehl erlaubt es, den Inhalt des angegebene Registers zu ändern. Der Inhalt wird nach Aufruf geändert.

Bsp: *F F2* ... Ändern des Inhaltes Register F2 (02h).

*Einen Ein/Ausgabeport/pin ändern:* ***SE***
Syntax: ***SE*** *[EinAusgabeport/pin]*
Ändern des physikalischen Ein/Ausgabeports (RA, RB, RC, ..) oder eines Pins.

Bsp: *SE RA* .. Ändern des ganzen PA-Ports
*SE RA0* .. Ändern des Portes (Pin) PA0

*Die Register anzeigen:* ***DR***
Syntax: ***DR***
Zeigt in einer übersichtlichen Liste alle Register, Stacks, Portzustände (auch TRIS), Konfigurationsregister, etc. und sogar den Inhalt des EEPROM-Datenspeicher (bei PIC16C84) an.

*Den EEPROM-Inhalt modifizieren:* ***EE***
Syntax: ***EE*** *EEPROM-Adresse*
Mit diesem Befehl kann man den Inhalt einer einzelne EEPROM-Zelle ändern.

Bsp: *EE 1* .. Ändern der EEPROM-Zelle mit der Adresse 1

*Die Systemzykluszeit ändern:* ***SC***
Syntax: ***SC*** *[Zyklenzeit]*
Setzen der Zyklenzeit (in Mikrosekunde)
Bsp: *SC .2* .. Setzen der Zyklenzeit auf 200ns

*Ein Taktsignal einem Eingang zuführen:* ***CK***
Syntax: ***CK*** *[Pin, {#Hi,#Low|-}]*
Mit diesem Befehl wird ein Taktsignal an ein Pinport *Pin* des Prozessors angelegt (nur Simulation), welches eine Highzeit von *#Hi*-Takten und *#Low*-Takten besitzt. Statt der Low-High-Zeiten kann auch ein Bindestrich - eingegeben werden, welcher das Signal ausschaltet.

Bsp:*CK RC0,5,4* .. Takt (5:4) an PC0
*CK RC0,-* .. kein Takt an PC0

*Eine Taste eine "hardwaremäßige" Eingabe zuweisen:* ***DK***
Syntax: ***DK*** *[Funktionstaste#, [PortPin, Ereignis]|[-]]*
Mit diesem Befehl (ohne weitere Optionen) werden alle Tastenzuweisungen angezeigt. Bei einer Zuweisung gibt die Zahl (1-12) nach dem Code *DK* die Funktionstaste, die bei Betätigen der ALT-Funktionstaste-Nr.x an dem angegebenen Pin eines angegebenen Ports folgende angegebenes Ereignis (*H* - high, *L* - low, *T* - Toggle, *P* - Puls) simuliert, an. Ein Befehlsaufruf mit der Option Bindestrich löscht eine/alle ALT-Funktionstastendefinitionen.

Bsp: *DK* .. Zeigt alle Definitionen
*DK 1,RB0,L* .. Bei ALT-F1 ein LOW-Signal an PB0
*DK 2,MCLR,P* .. Bei ALT-F2 ein Impuls an MCLR
*DK 2,-* .. Löschen des Belegung von F2
*DK -* .. löschen aller Belegungen (F1..F12)

*Programmende:* ***Q***
Syntax: ***Q***
Quittiert das Simulationsprogramm.

Zum Exekutieren der eingegebenen Befehle muß die Taste RETURN bzw. ENTER betätigt werden.

### Das Verhalten des Simulators bei speziellen Operationen

RESET-Verhalten
Es werden alle Reset-Typen (Power-On-Reset, /MCLR=0, WDT-Auszeit) simuliert.

SLEEP
Der MPSIM simuliert auch den SLEEP-Befehl; eine SLEEP-Funktion kann durch ein Aufwecken durch z.B. einen Watchdog-Auszeit durchgeführt werden.

Watch-Dog-Auszeit
Der WDT wird voll im Simulator unterstützt.

Die INItialisierungsdatei
Mit Hilfe einer in einem Editor geschriebenen INI-Datei, welche eine Liste von Simulatorbefehlen beinhaltet, kann mit Hilfe des Befehles GE eine Datei geladen und ausgeführt werden.

## Programmierung

Neben Assembler und Simulator hat Microchip auch noch ein kleines Programmiergerät entwickelt. Es können nicht alle PIC-Prozessoren mit ein und demselben Gerät programmiert werden. Das Entwicklungstool PICSTART16B1 kann die PIC16C54, PIC16C54A, PIC16CR54, PIC16C55, PIC16C56, PIC16C57, PIC16C58A, PIC16C61, PIC16C620, PIC16C621, PIC16C622 PIC16C71, PIC16C71A und PIC16C84 programmieren. Die Steuerungssoftware ist einfach zu bedienen, da es Pull-Down-Menüs

besitzt, wie man in der folgenden Abbildung erkennen kann. Und das Programmiergerät (Netzgerät beiliegend) wird an die serielle Schnittstelle angesteckt.

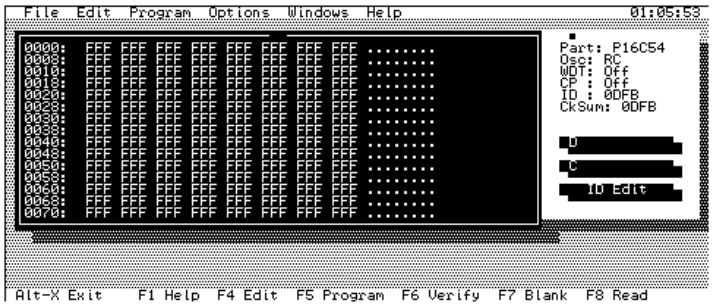


***Abb.10:*** **Bildschirmmaske des MPS16B**

## Entwicklungssysteme

Microchip bietet für wenig Geld ein gesamtes Entwicklungstool inkl. Assembler, Simulator, Programmiergerät, Datenbücher, Netzadapter und Musterprozessoren an.

Daher kann man sagen, das der PIC vom Microchip klein aber „oho" in der Prozessortechnik und Entwicklungsumgebung und „PIC-fein" in Sachen Service ist.

## *Quellenverzeichnis*

Literaturquelle:

*MICROCHIP Databook 1994*
*Microchip Technology Inc.*
*2355 West Chandler Blvd. Chandler, AZ 85224-6199*

*MPASM Assembler User's Guide*
*Microchip Technology Incorporation 1994*

*MPSIM Simulator User's Guide*
*Microchip Technology Incorporation 1994*

Tabellen- und Bilderquellen

*Abb. 1:* **Blockschaltbild der PIC16C5x-Familie**
*Microchip Databook 1994, „Figure 2.1.1 - PIC16C5X Series Block Diagram", Page 2-6, DS30015K-page 4*

*Abb. 2:* **PIN-Belegung**
*Microchip Databook 1994, „Figure A - PIN Configuration", Page 2-3, DS30015K-page 1*

*Abb. 3:* **Blockdiagramm v. RTCC/WDT Prescaler**
*Microchip Databook 1994, „Figure 9.0.1 - Block Diagram RTCC/WDT Prescaler", Page 2-19, DS30015K-page 17*

*Abb. 4:* **Blockschaltbild des PIC16C71**
*Microchip Databook 1994, „Figure B - PIC16C71 Block Diagram", Page 2-328, DS30150E-page 2*

*Abb. 5:* **Pinbelegung des PIC16C71**
*Microchip Databook 1994, „Figure A - Pin Configuration", Page 2-327, DS30150E-page 1*

*Abb. 6:* **Blockschaltbild des PIC16C84**
*Microchip Databook 1994, „Figure B - PIC16C84 Block Diagram", Page 2-536, DS30081C-page 2*

*Abb. 7:* **Pinbelegung des PIC16C84**
*Microchip Databook 1994, „Figure A - Pin Configuration", Page 2-535, DS30081C-page 1*

*Abb.8:* **Maske des MPASM**
*Microchip Assembler MPASM Version 1.21 (DOS)*

*Abb.9:* **Bildschirmmaske des MPSIM**
*Microchip Simulator MPSIM Version 5.11 (DOS)*

*Abb.10:* **Bildschirmmaske des MPS16B**
*Microchip Programmiergerät PIC-Start 16 B 1 (DOS)*

*Tabelle 1:* **Microchip - PIC - Family**
*Microchip Databook 1994, „PIC16/17 Family of 8-Bit Mikrocontrollers Cross-Reference Guide", Page 2-1, DS30352A-page1*

*Tabelle 2:* **/TO & /PD-Ereignisse**
*Microchip Databook 1994, „Table 4.5.2.1 - Event Affecting /PD|/TO Status Bits", Page 2-14, DS30015K-page 12*

*Tabelle 3:* **/PD & /TO nach Reset**
*Microchip Databook 1994, „Table 4.5.2.2 - /PO|/TO Status after Reset", Page 2-14, DS30015K-page 12*

*Tabelle 4:* **Quarz-Oszillator**
*Microchip Databook 1994,*
*„Table 12.2.1 - Capacitor Selection For Ceramic Resonant" &*
*„Table 12.2.2 - Capacitor Selection For Crystal Oscillator",*
*Page 2-26, DS30015K-page 24*

*Tabelle 5:* **RC-Takt**
*Microchip Databook 1994, „Table 18.0.1 - RC Oscillator Frequencies", Page 2-40, DS30015K-page 38*

*Tabelle 6:* **WDT-Zeitperiode abhängig von Temperatur und Versorgungsspg.**
*Werte aus folgendem Diagramm abgelesen:*
*Microchip Databook 1994, „Figure18.0.15 - WDT Timeout Period vs. $V_{DD}$", Page 2-44, DS30015K-page 42*

*Tabelle 7:* **Spannungspegel an /MCLR**
*Microchip Databook 1994, „16.6. DC CHARACTERISTICS: PIC16C5X-RC, XT, HS, LP (Commercial), PIC16C5XI-RC, XT, HS, LP (Industrial); 16.7. DC CHARACTERISTICS: PIC16C5X-RC, XT, HS, LP (Automotive)", Page 2-35 & 36, DS30015K-page 33 & 34*

*Programmlisting 1:* **Beispiel für STATUS-Register-Optionen**

*Programmlisting 2:* **Beispiel für RTCC**

*Programmlisting 3:* **Beispiel für INTERRUPT an PB0**

*Programmlisting 4:* **Beispiel für 16-Bit-Zähler mit INTERRUPT**

*Programmlisting 5:* **Beispiel für AD-Betrieb**

*Programmlisting 6a:* **Beispiel für EEPROM-LESEN**

*Programmlisting 6b:* **Beispiel für EEPROM-SCHREIBEN**

*Programmlisting 6:* **Beispiel für EEPROM**

weitere Literatur

„PIC Controller", Michael Thieser, Franzis-Verlag

„Mikrocontroller mit RISC-Struktur; Die PIC-16C5X-Familie", C.F. Urban, elektor

„PIC Anwendungssammlung; komplexe Schaltung einfach lösen mit PICs", elektor

„Klein, aber PICfein", Prof. Dr. Anne Frohn-König + Dipl. Ing. (FH) Manfred König, ELRAD Heft 5/93 und 6/93

„PIC16Cxx/PIC17Cxx Mikrocontroller; Architektur, Werkzeuge, Applikationen", Johann Wiesböck + Martin Burghart, Design&Elektronik + Microchip

„Das Parallax Assembler Arbeitsbuch zu dem Mikrocontrollern PIC16Cxx", Scott Edwards + Claus Kühnel, Electronic Media GmbH

□