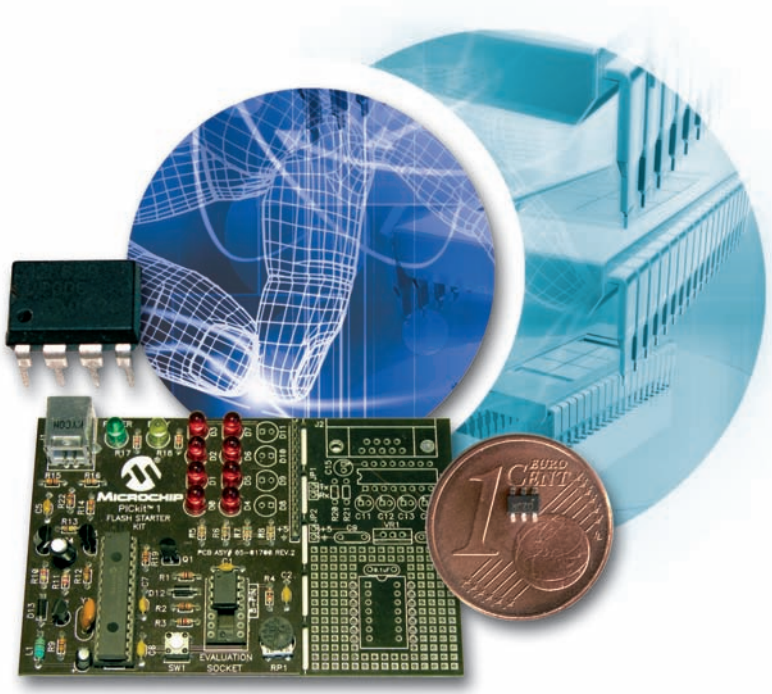
Elektronik

Thorsten Mumm



PICs für Einsteiger

Tipps und Tricks rund um das PICkit™ 1 FLASH STARTER KIT

Auf CD-ROM

- Programmbeispiele
- Datenblätter zu den PICs
- Befehlsübersicht

FRANZIS

Mumm

PICs für Einsteiger

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte Daten sind im Internet über **http://dnb.ddb.de** abrufbar.

Satz: Fotosatz Pfeifer, 82166 Gräfelfing
art & design: www.ideehoch2.de
Druck: Legoprint S.p.A., Lavis (Italia)
Printed in Italy

ISBN 3-7723-**4994-3**

Vorwort

Dieses Buch möchte sich an Elektronikbastler wenden, die auf der Suche nach einer kleinen Einführung in die Welt von Mikrocontrollern und deren Programmierung sind.

Es werden keine Softwarekenntnisse in irgendeiner Programmiersprache vorausgesetzt, lediglich Grundkenntnisse in der Elektrotechnik sollten vorhanden sein.

Es sollen in dem Buch gesammelte Erfahrungen mit den PIC-Mikrocontrollern 12F629 und 12F675 weitergegeben werden, um Neulingen auf diesem Gebiet die Angst vor den dicken Datenbüchern zu nehmen, oder ihnen einfach nur zu helfen, Fehler, die immer wieder gemacht werden, zu vermeiden. Als Programmiersprache kommt in den Aufgaben der PIC-Assembler mit seinen nur 35 Befehlen zum Einsatz, die alle einzeln und anhand von Beispielen erklärt werden.

Als erstes erfolgt eine kurze Vorstellung der im Buch benutzten Entwicklungsumgebung, dem »PICkit™ 1 FLASH STARTER KIT« von Microchip. Diese Entwicklungsumgebung ist mit etwa 30,– € recht preisgünstig. Sie unterstützt zwar nur eine kleine Anzahl an Controllern mit bis zu 12 I/Os, die sich aber sehr gut zum Einstieg in das Arbeiten mit PIC-Mikrocontrollern eignen. Der Schwerpunkt in diesem Buch wird auf dem PIC-Controller 12F629 liegen. Alle vorgestellten Programme laufen aber auch – mit einer kleinen Änderung – auf dem PIC 12F675. Mit einigen weiteren Anpassungen laufen sie auch auf jedem anderen PIC-Controller.

Das PICkit 1 verfügt über eine USB-Schnittstelle und kann so an jeden modernen PC mit einer USB-Schnittstelle und einem Betriebssystem ab Windows 98SE angeschlossen werden.

Es wird kein Treiber und auch **keine** extra Stromversorgung für den Betrieb des PICkits 1 benötigt.

Auf der PIC-Platine steht, neben der Programmierungsmöglichkeit für die Controller, auch eine kleine Testumgebung zur Verfügung. Microchip bietet darüber hinaus Erweiterungskarten zum PICkit 1 an, zum Beispiel das PICtail™

Daugther Board »Signal Analysis«, welches mit der beiliegenden Software als kleines, aber sehr langsames Digital-Scope benutzt werden kann.

Wenn man schon im Besitz eines Brenners für PIC-Mikrocontroller ist, kann man sich aber auch nur die Testumgebung nachbauen, um die Beispiele aus dem Buch durchzuarbeiten.

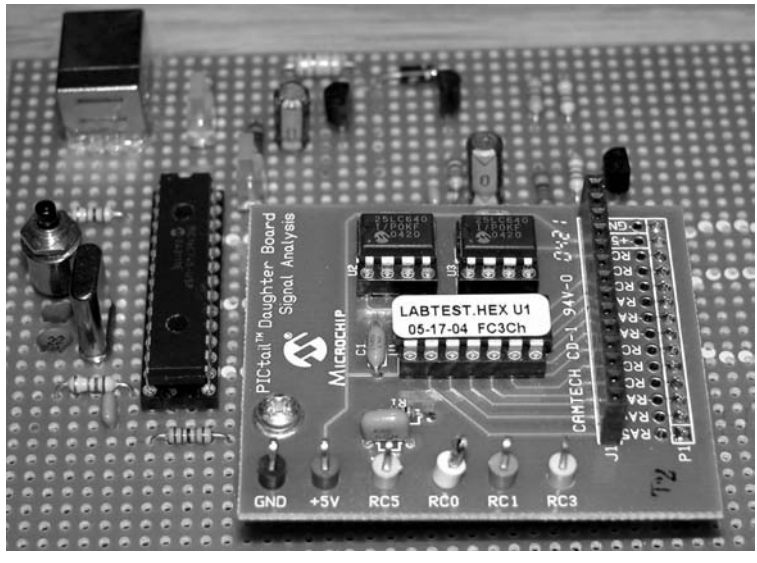


Abb. 1: daugther board »Signal analysis« mit einem selbstgebauten PICkit

Bearbeitet werden Grundlagen, die alle mit dem PICkit 1 ohne jegliche Hardware-Erweiterung möglich sind. Zum Beispiel: Das Einlesen einer Taste und ein daraus resultierendes Schalten einer Leuchtdiode. So werden diese Grundlagen schrittweise erklärt und dann immer weiter ausgebaut. Es folgt ein Blink- und Lauflicht, wo sich am Ende die Geschwindigkeit über einen verstellbaren Widerstand beeinflussen lässt. Dabei wird der im PIC 12F675 eingebaute A/D-Wandler benutzt.

Inhaltsverzeichnis

1 PIC-Controller, der unbekannte Chip

Was ist ein PIC-Mikrocontroller?

Die Abkürzung PIC steht schlicht für »**P**rogrammable **I**ntegrated **C**ircuit«.

Etwas übertrieben gesagt, ist ein Mikrocontroller ein Micro-PC ohne Tastatur, der die verschiedensten Aufgaben erfüllen kann. Mikrocontroller sind aus der Technik heute nicht mehr wegzudenken. Sie sitzen in Waschmaschinen, Gefriertruhen, Fernbedienungen, Autos und so weiter und so fort. Es gibt in der Zwischenzeit eine große Anzahl von verschiedenen Typen für die unterschiedlichsten Anwendungen. Auch die Angebotspalette der Firma Microchip ist sehr groß. Die zurzeit leistungsstärksten sind die dsPICs, die aber in diesem Buch keine Rolle spielen werden. Mit diesen doch sehr komplexen und vielseitigen Mikrocontrollern sollte man nicht den Einstieg in die Welt des Programmierens wagen. Dafür eignen sich die mit dem PICkit 1 auch programmierbaren zwei Typen 12F629/675 und 16F630/676 besser, die sich nur durch den bei dem 12F675 und 16F676 vorhandenen A/D-Wandler und der Anzahl der zur Verfügung stehenden I/O-Pins unterscheiden. Im Buch werden nur die kleineren PICs der Reihe 12F629 und 12F675 Verwendung finden. Sie sind erheblich einfacher, als die dsPICs, bieten aber trotzdem noch eine sehr gute Basis für den Einstieg mit ihren Möglichkeiten.

Abb. 2: Ein PIC10F202 in der SOT-23 Bauform auf einem 1-Cent-Stück

Für Hobby-Bastler auch sehr interessant sind die im Herbst 2004 erschienenen PICs der 10F20x Familie mit 4 oder 6 I/Os. Sie sind auch mit dem PICkit 1 programmierbar und wer etwas sehr Kleines bauen möchte, der kann bei diesen Controllern auf Gehäuse der Bauart SOT-23 mit den Außenmaßen von 2,95 x 2,8 mm zurückgreifen. Die kann man zum Beispiel zum Steuern einer Umleuchte auf einem Modell-Polizeiwagen benutzen. Laut der Werbung von Microchip sind es die zurzeit kleinsten Mikrocontroller der Welt. Aber auch die

anderen PIC-Conroller sind in einer SMD-Ausführung erhältlich. Am Ende des Buches wird hierzu noch kurz ein kleines Hilfsmittel für das Programmieren von SMD Mikrocontrollern vorgestellt.

Alle in diesem Buch vorgestellten Programme sind auch mit kleinen Anpassungen auf anderen Controller-Typen lauffähig. Dazu gibt es mehr Informationen im Kapitel »Tipps & Tricks«.

Hier nun einige technischen Daten der zwei im Buch benutzten PICs in Kurzform:

- nur 35 Assembler Befehle, die gelernt werden müssen
- alle Befehle, bis auf Sprünge, werden in einem Schritt (Takt) abgearbeitet
- 2,0 V – 5,5 V Betriebsspannung
- Low power Power-on Reset (POR)
- bei 20 MHz Oscillator 200 ns Instruction cycle
- 1024 Words FLASH Program Memory
- Internal Oscillator mit 4 MHz und ±1%
- 128 Byte EEprom
- bis zu 4 A/D-Wandlerkanäle
- 6 I/Os
- In-Circuit Serial Programming™ (ICSP™) über zwei Pins
- und vieles mehr...

Wie schon erwähnt, ist die Angebotspalette von Mikrocontrollern der Firma Microchip sehr groß. Es gibt viele verschiedene Mikrocontroller, mit besonderen Hardware-Eigenschaften für ganz spezielle Aufgaben. So zum Beispiel für die unterschiedlichsten Schnittstellen, wie USB und zu anderen BUS-Systemen, oder für die Steuerung von Motoren mit PWM (Pulsweitenmodulation).

Die unterschiedlichen Hardware-Funktionen in den Controllern machen es nötig, dass vor der Inbetriebnahme eines Controllers verschiedene Konfigurations-Bits gesetzt werden müssen.

Und hier beginnt nun schon das Lernen:

Etwas, was die ganze Sache erschwert, ist, dass die Dokumentation in den Datenbüchern der verschiedenen Controller-Typen teilweise mit verschiedenen Abkürzungen arbeiten, um dasselbe zu beschreiben. Dies trifft leider auch auf

die im Buch beschriebenen Controller zu. So kann man leider ein Programm nicht, ohne Änderungen vorzunehmen, auf einen anderen Controller übertragen. Auf was man hierbei achten muss, wird in dem Kapitel »Tipps und Tricks« angesprochen. Dieses Abkürzungsproblem taucht auch wieder auf, wenn man die Entwicklungsumgebung MPLAB verwendet. Dort gibt es dann wieder andere Begriffe und Abkürzungen, als in den Datenblättern. So zum Beispiel bei der Auswahl der Konfigurations-Bits. Also bitte nicht verwirren lassen!

Im nächsten Abschnitt sollen nun die Konfigurations-Bits der zwei im Buch benutzten Controller näher vorgestellt werden. Solche Grundeinstellungen sind zu jedem Programm nötig. Um die Thematik zu vereinfachen, wird für alle folgenden Aufgaben die gleiche Einstellung gewählt. Diese Einstellungen gelten sowohl für den 12F629 als auch für den 12F675.

1.1 Der PIC12F629 und 12F675

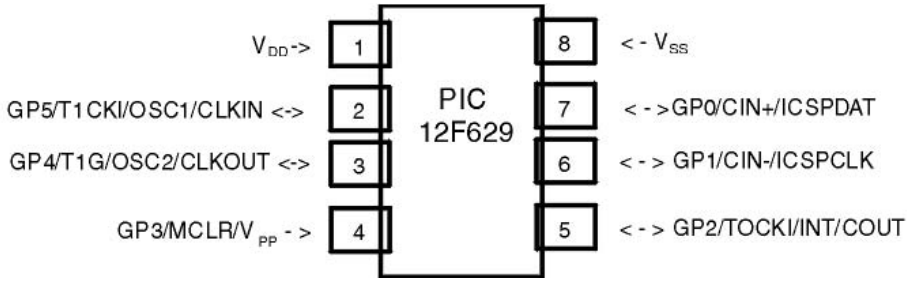


Abb. 3: Pin-Layout zum PIC 12F629

Wie man aus dem Bild bereits erahnen kann, sind selbst bei diesem kleinen Controller die Pins mit verschiedenen Funktionen belegt, so dass eine Auswahl getroffen werden muss. Für den PIC12F675 kommen dann noch die vier Analogeingänge an den Pins 3 und 5-7 hinzu. Es gibt dabei Eigenschaften, die einmalig festgelegt werden müssen, sowie Funktionen, die auch im Betrieb geändert werden können.

Funktionen, die nicht im Betrieb geändert werden können, werden mit den Konfigurations-Bits eingestellt. Für diese Controller gibt es sieben dieser Bits.

Die Bits im Einzelnen:

- WDT: steht für den Wachhund, den **W**atch**d**og **T**imer. Mit diesem Timer ist es möglich, dass ein Controller sich im Betrieb selbst überwacht. Ist dieser Timer aktiviert, muss er immer in bestimmten Zeitintervallen wieder zurück-

gesetzt werden, sonst löst der Watchdog Timer automatisch einen RESET aus.

- BODEN: ist die Abkürzung für das **B**rown-**o**ut **D**etect **En**able Bit. Dies ist eine Art Sicherheitsfunktion der Controller. Hier wird erkannt, wenn die Betriebsspannung unter den zulässigen Minimal-Wert von etwas mehr als 2 V fällt. Tritt dies ein, wird im Controller ein RESET erzeugt und aufrechterhalten. Erst, wenn die Spannung wieder den Vorgaben entspricht, beginnt der Controller das Programm mit einer Zeitverzögerung vom 72 ms erneut von vorne abzuarbeiten.
- MCLRE: steht für **M**aster **Cl**ear **E**nable. Das sieht schlimmer aus, als es ist. Es legt nur die Funktion des Pins 4 fest, ob dieser ein I/O-Pin, oder ob der Pin ein Reseteingang ist. Dann würde der Controller immer bei einer fallenden Flanke von 5 V nach 0 V einen RESET ausführen und das Programm würde von vorne starten, egal an welcher Stelle es gewesen ist.
- PWRTE: **P**o**wer**-up **T**imer **E**nable ist verwandt mit der Brown out Funktion. Hiermit wird sichergestellt, dass der Controller erst mit Erreichen der für einen sicheren Betrieb notwendigen Spannungshöhe und einer darauf folgenden Zeitverzögerung von nominal 72 ms mit der Programmabarbeitung beginnt. Dies sichert ein sauberes Anlaufen des Controllers beim Einschalten der Betriebsspannung.
- CP: Ist das **C**ode **P**rotection Bit. Wird dies Bit gesetzt, ist es nicht mehr möglich, das im Chip abgelegte Programm auszulesen.
- CPD: **D**ata **C**ode **P**rotection Bit. Mit diesem Bit wird das im Controller integrierte EEprom geschützt, so dass kein ungewolltes Schreiben auftreten kann.
- Oscillator: hier muss nun der Typ des Taktgenerators ausgewählt werden. Hierfür stehen acht Unteroptionen zur Auswahl. Es werden dadurch gleichzeitig die Eigenschaften von mehreren Pins beeinflusst!

    1. LP: **L**ow **P**ower Oscillator: Diese Einstellung sollte bei einem Quarz, Keramikresonator oder bei einer externen Taktquelle mit geringer Frequenz ausgewählt werden. Diese niedrigen Frequenzen helfen, Energie zu sparen, wodurch zum Beispiel Batterien länger halten. Wird diese Funktion ausgewählt, entfallen die Eingänge GPIO4 und GPIO5 und werden zu Takteingängen.

2. XT: Crystal/resonator: Diese Einstellung sollte bei einer externen Taktquelle im Bereich von 100 kHZ bis etwa 4 MHz gewählt werden. Ansonsten ist das Verhalten mit LP identisch.
3. HS: High speed crystal/resonator: Diese Einstellung sollte bei einer externen Taktquelle im Bereich von 4 MHZ bis 20 MHz gewählt werden. Ansonsten ist das Verhalten mit LP identisch.
4. EC: **E**xternal **c**lock: Gibt es in der Schaltung bereits einen Systemtakt, der auch für diesen Controller benutzt werden kann, so ist das die zu wählende Einstellung. Der Takt wird an GPIO5 angeschlossen und GPIO4 steht als I/O-Pin zur Verfügung.
5. INTOSC_NOCLKOUT: Diese Einstellung aktiviert eine interne 4 MHz Taktquelle, die eine Genauigkeit von 1% hat. Dadurch stehen bei dieser Funktion beide Pins GPIO4 und GPIO5 als I/O-Pin zur Verfügung.
6. INTOSC_CLKOUT: Diese Einstellung aktiviert auch den internen 4 MHz Takt. Der Unterschied ist jener, dass dieser Takt an GPIO5 zum Abgriff zur Verfügung steht und nur GPIO4 als I/O-Pin benutzt werden kann.
7. RC – Oscillator (EXTRC_OSC_NOCLKOUT): Diese Einstellung bietet die Möglichkeit, den Takt mit einer RC-Kombination zu erzeugen. Dabei dient GPIO5 als Takteingang und GPIO4 steht als I/O-Pin zur Verfügung.
8. RC – Oscillator (EXTRC_OSC_CLKOUT): Ist fast identisch mit dem Punkt sieben, nur dass hier GPIO4 keine I/O Funktion bietet. Dort steht der Takt der RC-Kombination zum Abgriff zur Verfügung.

Jetzt noch kurz den doch sehr trockenen Stoff beenden und die Einstellungen für alle in diesem Buch behandelten Aufgaben betrachten. Die gewählten Einstellungen sind in den Programmköpfen der Quellcodes mit einem Sternchen markiert.

1.2 Die Config-Bits für die Aufgaben

Das Code Protection Bit braucht nicht gesetzt zu werden. Dies wird erst interessant, wenn man die geschriebenen Programme gegen unbefugtes Auslesen schützen möchte.

Da das EEprom in keiner Aufgabe benutzt wird, ist diese Einstellung auch ohne Bedeutung. Für die Beispiele wurde die Einstellung »off« gewählt.

Der Wachhund (watchdog timer) darf auch zu Hause bleiben und wird nicht aktiviert. Dies erfordert in der Anfangszeit einen zu großen Aufwand.

Den Power up Timer und die Brown out Funktion machen schon Sinn, denn so ist sichergestellt, dass der Controller nur unter den richtigen Betriebsbedingungen arbeitet und kontrolliert anläuft. Dies erspart einem die sonst dafür zusätzlich erforderliche Beschaltung.

MCLRE = OFF: Da alle Beispiele auch auf der Testplatine laufen sollen, muss hier der Eingang GPIO3 gewählt werden, so dass die Rest-Funktion nicht zur Verfügung steht.

Für die Oszillator-Einstellung ist die Funktion »interner OSC« ohne ‚Clock out' zu wählen, da auf der PIC-Platine keine andere Taktquelle zur Verfügung steht.

Um diese Einstellungen nun in den Controller zu übertragen, gibt es zwei Wege:

Einmal über die Entwicklungsumgebung MPLAB, dazu aber erst mehr im Kapitel »MPLAB«.

Oder alternativ über den Compiler im Quellcode. Dazu müssen diese Informationen dem Compiler zur Verfügung gestellt werden. Das geschieht über die Zeile: ‚_config...' für die es zwei Schreibweisen gibt: Die Ausführliche, wie sie hier als erstes folgt:

```
__CONFIG  _CP_OFF & _WDT_OFF & _BODEN_ON & _PWRTE_ON &
_INTRC_OSC_NOCLKOUT & _MCLRE_OFF & _CPD_OFF
```

Achtung: Wenn die obige Zeile in ein Programm eingefügt werden soll, darf hier kein Zeilenumbruch erfolgen.

Oder aber auch die verkürzte Schreibweise »_config H'3FC4'«, was dem Compiler genau dasselbe sagt. Als ungeübter Programmierer liest sich dies aber sicher nicht ganz so verständlich.

Tipp: Für die Einstellung der Konfigurationsbits sollte man immer die ausführliche Schreibweise wählen, denn wenn man nach Tagen oder gar Wochen mal wieder mit seinem PIC arbeitet, weiß man den Code der Kurzschreibweise sicher nicht mehr.

Für die ersten eigenen Projekte findet man diese Einstellungen, ohne einen anhängenden Quellcode, in der Datei »Konfig_Bits.asm« auf der CD in dem Ordner »Aufgaben«.

Sicher, bei kleinen Projekten sagt man sich: »Was soll diese ganze Dokumentation? Ich weiß doch, was ich geschrieben habe!« – Ja, so habe ich auch angefangen. Die Strafe folgt sicher, spätestens dann, wenn man nach langer Zeit das Projekt wieder einmal weiter bearbeiten möchte, weil es in der Freizeit wichtigere Dinge gab, als das Programmieren oder man nach einiger Betriebszeit eine Erweiterung einarbeiten möchte. Aber das sind vermutlich Erfahrungen, die man selber machen muss. Ich konnte auch nicht hören und musste manches wieder von vorne beginnen.

Um sich diese Fleißarbeit etwas zu erleichtern, kann man später immer wieder auf die Datei »Konfig_Bits.asm« zurückgreifen. Diese Datei stellt neben den Konfigurationsbits einen Programmkopf bereit, mit dem sich der Aufwand für die Dokumentation eines Projektes etwas verringert. Hier kann man zum Beispiel eine Kurzbeschreibung des Programms einfügen und alle weiteren Informationen, die man für wichtig hält. Lieber etwas mehr schreiben, als zu wenig. Diese Datei sollte man von Zeit zu Zeit immer um die neuen Funktionen, die man dokumentieren möchte, erweitern. So wird im Laufe der Zeit ein Programmkopf entstehen, der nur zu jedem neuen Projekt angepasst werden muss. Hierdurch lässt sich der Dokumentationsaufwand erheblich vereinfachen.

Hier der Ausschnitt aus der Datei mit den Konfigurationsbits:

```
; CP: Code Protection bit
; *   OFF = Program Memory code protection is disabled
;     ON  = Program Memory code protection is enabled
; WDT: Watchdog Timer Enable bit
;     ON  = WDT enabled
; *   OFF = WDT disabled
; BODEN: Brown-out Detect Enable bit
; *   ON  = BOD enabled
;     OFF = BOD disabled
; PWRTE: Power-up Timer Enable bit
;     OFF = PWRT disabled
; *   ON  = PWRT enabled
; MCLRE: GP3/MCLR pin function select
;     ON  = GP3/MCLR pin function is MCLR
; *   OFF = GP3/MCLR pin function is digital I/O
; CPD: Data Code Protection bit
; *   OFF = Data memory code protection is disabled
;     ON  = Data memory code protection is enabled
; Oscillator Variationen:
;     LP_OSC   = Low power crystal on GPIO4 and GPIO5
;     XT_OSC   = Crystal/resonator on GPIO4 and GPIO5
;     HS_OSC   = High speed crystal/resonator on GPIO4 and GPIO5
```

```
;     EC_OSC   = I/O function on GPIO4 pin, CLKIN on GPIO5
; *   INTRC_OSC_NOCLKOUT  = I/O function on GPIO4 pin, I/O function on GPIO5
;     INTRC_OSC_CLKOUT  = CLKOUT function on GPIO4 pin, I/O function on GPIO5
;     EXTRC_OSC_NOCLKOUT  = I/O function on GPIO4 pin, RC on GPIO5
;     EXTRC_OSC_CLKOUT    = CLKOUT function on GPIO4 pin, RC on GPIO5
```

Abb. 4: Programmkopf zur Erklärung der Configurations-Bits

Soweit die Konfiguration der einmalig zu treffenden Einstellungen. Um nicht zuviel Theorie an den Anfang zu stellen, sollen die im Betrieb veränderbaren Einstellungen erst in den Beispielen erklärt werden, wo sie dann auch gleich angewendet werden können.

6 Der PIC Assembler

In diesem Kapitel sollen nun alle 35 Befehle des PIC-Assemblers erklärt werden. Dazu gehört, welche Flags durch einen Befehl im Status-Register beeinflusst werden könnten und wie sich die Befehle auf die benutzten Universalregister auswirken. Das wird sich sicher sehr trocken lesen lassen, aber es lässt sich leider nicht umgehen. Zu jedem Befehl gibt es dann ein kurzes Beispiel, wie der Befehl in einem Programm stehen könnte oder geschrieben werden müsste.

Was einerseits ein großer Vorteil der PIC-Mikrocontroller ist, ist die geringe Anzahl der zu lernenden Befehle. Auf der anderen Seite ist dies aber auch ein gewisser Nachteil, denn manche Dinge, die man selbst von dem billigsten Taschenrechner erwartet, müssen hier mit erheblichem Aufwand von einem selber programmiert werden, um sie zu realisieren. Dazu gehört vor allem das Rechnen und im Speziellen die Division von Zahlen. Bei der Multiplikation ist es noch nicht ganz so schlimm, solange das Ergebnis kleiner als 255 bleibt.

Bei Zahlen mit verschiedenen Vorzeichen, oder auch solchen mit Kommastellen, sollte man es erst gar nicht in Assembler versuchen. Hier sollte man möglichst auf sogenannte Hochsprachen, wie »C« oder Ähnliches, ausweichen. (Zu diesem Thema bietet Microchip auch einige Technical Notes im Internet an.)

Außerdem gibt es auch noch andere Hersteller, die Demos im Internet anbieten, welche ohne Probleme in MPLAB eingebunden werden können. Zum Beispiel von der Firma B Knudsen Data aus Trontheim, die im Internet eine freie Version ihres CC5x C Compilers zur Verfügung stellt. Mit dieser Version ist es möglich, Programme mit bis zu 1024 Befehlen zu schreiben, was für einen Einstieg locker reicht.

Assembler hat seine Stärken in der ‚Hardware-Nähe'. Mit ihm ist es sehr leicht, die Ein- und Ausgänge eines Controllers abzuarbeiten und die Ergebnisse wieder auszugeben.

Auch wenn die Controller immer schneller werden und die Grenzen sinken, ist Assembler bei dem Schlagwort »echtzeitfähig« in der Computertechnik auch immer noch ein Thema. Aber auch hier geht es immer um hardwarenahe Probleme, die extrem schnelle Reaktionen der Geräte auf meist externe Ereignisse erfordern.

Wofür man sich langfristig entscheidet, sollte jeder für sich entscheiden. Man kann auch gut mit einer Kombination aus beidem, Assembler und Hochsprache, in einer Aufgabe arbeiten.

Bevor es aber nun an die Befehle geht, sollen hier noch ein paar besondere Register und Bits vorgestellt werden, mit denen es möglich ist, den Controller Entscheidungen treffen zu lassen. Zum Beispiel, wenn etwas ‚null' wird, oder wenn ein »Überlauf« auftritt. Es muss ja nicht nur ein externes Ereignis sein, auf das der Controller reagieren soll.

6.1 Das W-Register

Das ‚W-Register' ist das Arbeitsregister des Controllers, weswegen es oft auch als solches bezeichnet wird. Alle Operationen, die für die Ausführung ein zweites Register benötigen, arbeiten in Verbindung mit dem W-Register. Soll zum Beispiel etwas verglichen werden, wird im ersten Schritt ein Wert in das W-Register geladen und im nächsten Schritt kann der Wert eines anderen Registers dann mit diesem Wert im W-Register verglichen werden. Am Ende des Vergleiches kann man wählen, ob das Ergebnis im W-Register oder im zweiten Register abgelegt wird.

Liest man die Zustände der Eingänge eines Controllers, steht dieses Ergebnis auch im W-Register. Benötigt man diesen Wert später wieder, muss er als erstes in ein anderes Register gerettet werden, da er im nächsten Schritt sonst schon wieder überschrieben werden könnte.

6.2 Das Carry-Bit

Das Verhalten des Carry-Bits ist recht einfach, es wird immer dann gesetzt, wenn ein Überlauf auftritt. Nur, was ist als ein »Überlauf« definiert?

Zu Deutsch: Wann wird es gesetzt und wann nicht ?

Das Carry-Bit wird immer dann gesetzt, wenn das Arbeitsregister (das W-Register) von B'11111111' nach B'00000000' wechselt, oder umgekehrt. Ob dies in Dezimal oder Hexadezimal definiert ist und hier der Übertrag an anderen Stellen erfolgt, ist egal. Der Controller arbeitet binär mit acht Bits. Alles andere ist nur eine andere Darstellung für den Programmierer. Das Carry-Bit symbolisiert dabei einen Übertrag in ein nächstes Register. Ähnlich wie beim Kopfrechnen: »...ich habe dann einen im Sinn!«

Achtung: Dies ist auch der Fall, wenn das Ergebnis ‚null' 0 ist.

Man könnte es so betrachten:

5+5 sind 0 und Carry -> die Zehnerstelle steht im Carry-Bit. Dies wäre sicher schön, ist aber leider nicht so. Hier tritt leider kein Überlauf auf. Ein 8-Bit Register läuft bei dezimal 255 über!

Also zum Beispiel:

4 – 5 = 255 und Carry-Bit gesetzt

W-Register	=	B'00000100'
– zweites Register	=	B'00000101'
= W-Register	=	B'11111111' und Carry = 1

Aber auch 250 + 10 = 5 führt zu einem Überlauf in dem Ziel-Register und das Carry-Bit wird gesetzt.

Dies entspricht leider nur nicht unserem Dezimalsystem und erfordert deshalb besondere Lösungen bei vielen Rechnungen. Eine Lösung ist zum Beispiel, jede Stelle einer möglichen Zahl alleine in einem Register zu verarbeiten. Eine vierstellige Zahl benötigt dann vier Register. Um Register zu sparen, kann man auch zwei Zahlen in einem Register verarbeiten. Eine Zahl steht dann in den oberen 4-Bits und die zweite Zahl in den unteren 4-Bits. Hier kann man Überläufe mit dem Hilfs-Carry-Bit erkennen. Es signalisiert den Überlauf zwischen den oberen und den unteren vier Bits.

Hochsprachen sind in diesem Fall dem Assembler überlegen, da die Verwaltung des Carry-Bits von der Hochsprache gelöst wird. Aber bei kleinen ganzen Zahlen hält sich der Aufwand in Grenzen und ist mit einfachen Abfragen zu lösen. Im Kapitel »Tipps und Tricks« wird ein Lösungsansatz zum Rechnen mit ganzen Zahlen beschrieben.

6.3 Verhalten des Zero-Bits

Wann wird es gesetzt und wann nicht?

Das Zero-Bit wird nur gesetzt, wenn das Ergebnis gleich ‚null' 0 ist.

Dies gilt nicht nur für Rechenoperationen, sondern auch für Vergleiche oder Verknüpfungen von Registern.

Beispiele:

4 – 4 = 0 Carry und Zero -> beide Bits werden gesetzt!

4 – 5 = -1 nur Carry wird gesetzt, auch wenn Null durchlaufen wurde!

Es gilt immer: Ist nach einer UND- oder ODER-Verknüpfung im Ergebnis-Register kein Bit gesetzt, wird auch das Zero-Bit gesetzt. Dies wird noch deutlicher im Zusammenhang mit den Befehlen, die das Zero-Bit beeinflussen, erklärt.

Achtung: Das Zero-Bit wird auch nach dem Löschen eines Registers gesetzt!

6.4 Lesen von Registern, wo ist das Bit ‚null' 0 ?

Dies ist eine Sache, an die man sich einfach gewöhnen muss: Nämlich dass der PIC (und auch viele andere Controller) hier entgegen unserer gewohnten Lese-Richtung arbeiten. Hier schleichen sich immer wieder Fehler ein und man wundert sich, weswegen das Programm nicht funktioniert: Weil man mal wieder nicht mit ‚null' 0 begonnen hat, oder auf der falschen Seite angefangen hat, zu zählen.

Dies passiert vor allem dann, wenn man sich nur gelegentlich mit den PICs beschäftigt und einem die Übung fehlt.

Hier mal an einem Beispiel dargestellt:

```
MOVLW    B'11001010'    ; Wert wird geladen
MOVWF    GPIO           ; Ausgabe an Port GPIO
```

Welche Ausgänge am Port GPIO sind nun ‚high' (an) und welche ‚low' (aus)?

Hier ist es sehr hilfreich, wenn man sich die verschiedenen Möglichkeiten der Eingabeformen von Zahlen zunutze macht. Es schadet nicht, wenn man in einem Programm nicht einheitlich arbeitet. Man sollte immer die für die Situation beste Schreibweise wählen. Hier ist die binäre Schreibweise sehr hilfreich. Man sieht anhand des Wertes die Zustände der Ausgänge, welche ‚high' und ‚low' sind, und muss diese nicht erst umrechnen.

Bit	7	6	5	4	3	2	1	0
PortB	1	1	0	0	1	0	1	0
Ausgang	AN	AN	AUS	AUS	AN	AUS	AN	AUS

Das gleiche Ergebnis erhält man auch beim Benutzen der Hex-Schreibweise: ‚XX'h

```
MOVLW    CAh     ; Maske wird geladen
MOVWF    GPIO    ; Ausgabe an Port GPIO
```

oder mit Dezimalwerten: D'xxx'

```
MOVLW    D'202'  ; Maske wird geladen
MOVWF    GPIO    ; Ausgabe an Port GPIO
```

Leider ist es hier bei diesen Schreibweisen ungleich schwerer, als ungeübter Programmierer auf den gesetzten Ausgang zu schließen.

Natürlich ist es aber einfacher, bei Rechnungen mit den gewohnten Dezimalwerten zu arbeiten. Was die gesamte Arbeit vereinfacht, ist, dass auch eine Kombination der Eingabeformen jederzeit möglich ist. So kann ein Binärwert mit einem Dezimalwert addiert oder auch verglichen werden. Bei der Programmierung muss nur auf die korrekte Schreibweise der Eingabe geachtet werden und man muss sich über das Ergebnis, welches immer ‚binär' im Register steht, im Klaren sein.

Achtung: Auch wenn man mit zwei dezimalen Werten arbeitet, treten im PIC-Controller die Überträge nach den binären Regeln auf!

6.5 Die Schreibweise von Befehlen

Die Schreibweise von Befehlen im Programm-Editor ist egal. Der Compiler ist so angelegt, dass er mit großen und kleinen Buchstaben umgehen kann. Selbst eine gemischte Schreibweise, von kleinen und großen Buchstaben führt nicht zu einer Fehlermeldung. Dass der Compiler den Befehl erkannt hat, merkt man immer daran, dass sich bei der Eingabe dann die Farbe der Schrift geändert hat.

Auch gibt es noch einige weitere spezielle Befehle. Diese basieren aber alle auf den hier vorgestellten 35 Hauptbefehlen. Es sind lediglich Kurzschreibweisen für sehr häufig vorkommende Befehlsstrukturen, um Schreibarbeit zu sparen und gegebenenfalls die Lesbarkeit von Programmen zu erhöhen.

Hier sei nur ein Beispiel erwähnt:

```
movfw   Zielregister      (Wie zum Beispiel in Aufgabe 9:  »movfw   einer«)
```

Dies ist die Kurzschreibweise für »movf f,d« und bewirkt nichts anderes, als die Zeile:

```
movf    Zielregister,0
```

...was bedeutet: Verschiebe den Inhalt des angegebenen Zielregisters in das W-Register.

Dazu gibt es weitere Befehlsvarianten, die jeweils wie ein Macro aufgebaut sind, was wiederum ähnlich wie ein Unterprogramm arbeitet.

6.6 Legende zum Assembler

k	Konstante oder Label
d	Ziel
f	Registeradresse
W	W-Register / Arbeitsregister
C	Carry-Bit
DC	4 Bit Überlauf / Hilfscarry
Z	Zerobit
TO	Time-out Bit
PD	Power-down Bit
PC	Programm Counter / Zähler
H'....'	Angabe in Hex oder auch 0x..
D'...'	Angabe in Dezimal
B'.....'	Angabe in Binärschreibweise
variable	ein beliebiges Register im Controller

Elektronik

Thorsten Mumm

PICs für Einsteiger

Tipps und Tricks rund um das PICkit™1 FLASH STARTER KIT

Die Hürde zum Selbstprogrammieren von PIC-Mikrocontrollern liegt längst nicht so hoch, wie vielfach angenommen wird. Das Problem besteht dabei weniger in der Sache selbst als in der Frage, wie man als Einsteiger zu konkreten Ergebnissen kommt, ohne erst stundenlang dicke Datenbücher studieren zu müssen.

Dieses Buch informiert über alles, was zu einem schnellen Start mit PIC-Controllern erforderlich ist.

Das Buch bietet anhand des im Handel erhältlichen PICkit 1FLASH Starter Kit einen Einstieg in die Welt der PIC-Mikrocontroller. Das Starter Kit liefert die erforderliche Entwicklungsumgebung, die Brennfunktion und eine kleine Testumgebung. Zu Beginn werden die im Buch verwendeten PICs vom Typ 12F629 und 12F675 vorgestellt.

Es folgt die Platinenbeschreibung sowie eine Einführung in die Entwicklungsumgebung MPLAB von Microchip. Der Anwender wird dabei nur über die wesentlichen Punkte informiert, die für eine erste Inbetriebnahme unbedingt erforderlich sind. Anschließend sollen die ersten Erfolge den Leser zum Weitermachen animieren. Selbstverständlich werden alle 35 Assembler-Befehle ausführlich erörtert.

Aus dem Inhalt:

- PIC-Grundlagen
- Einführung in die Hard- und Software des PICkit
- Programmbeispiele
- Erläuterung der 35 Assembler-Befehle
- Praktische Applikationen wie z. B. Blinklicht, Lauflicht, Impulszähler
- Tipps und Tricks für Eigenentwicklungen

Auf CD-ROM:

- Programmbeispiele
- Datenblätter zu den PICs
- Befehlsübersicht

ISBN 3-7723-4994-3

9 783772 349942

Euro 19,95 [D]

FRANZIS

Besuchen Sie uns im Internet: www.franzis.de