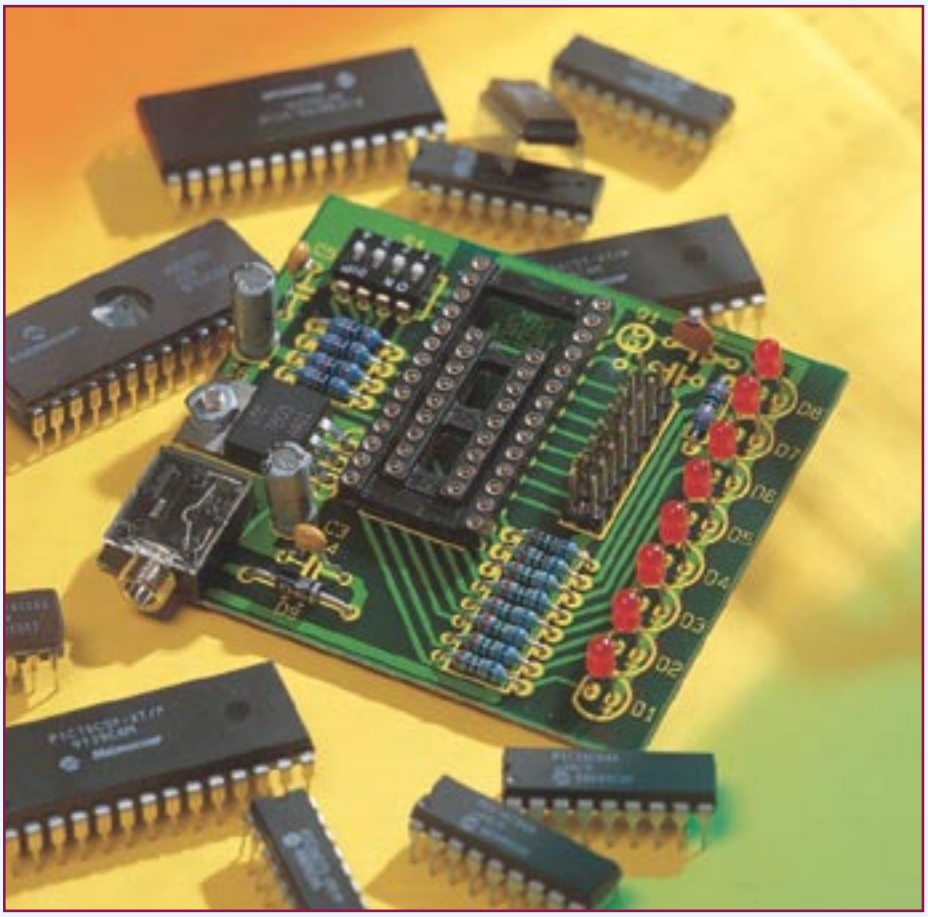

PIC-Grundlagen Teil 6

In diesem Teil der Artikelserie stellen wir den PIC 16F84 vor. Dieses Derivat der PIC-Familie verfügt über einen internen Programmspeicher, der mehrfach elektrisch gelöscht und neu programmiert werden kann.

Allgemeines

Wer sich bereits mit den PICs der 16C5X-Familie beschäftigt hat, stößt bei steigenden Anforderungen leicht an die Grenzen der zur Verfügung stehenden Möglichkeiten. Die 16C5X-Familie bietet zum Beispiel keine Interruptfunkionen, so daß zum Überwachen eines Portpins dieser ständig abzufragen und auszuwerten ist. Ebenso muß zur Realisierung von zeitlichen Abläufen über den internen Timer ein ständiges Auslesen des Timers erfolgen, um einen Überlauf zu erkennen.

Dazu sind die „kleinen" PICs intern nur mit einem 2-Ebenen-Hardwarestack ausgerüstet, so daß die Anzahl der ineinander geschachtelten Funktionen auf zwei begrenzt ist.

Für Anwendungen, die ein wenig mehr Flexibilität vom Mikrocontroller erfordern, ist daher der PIC16F84 besser geeignet.

Der PIC 16F84

Der PIC 16F84 befindet sich in einem 18poligen DIP-Gehäuse. Er ist pinkompatibel zu den PICs 16C54 und 16C56, so daß man ihn einfach gegen diese austauschen kann. Die Anschlußbelegung des PIC 16F84 ist in Abbildung 24 dargestellt.

Es soll zu Beginn noch kurz darauf hingewiesen werden, daß der PIC 16F84 der Nachfolger des heute nicht mehr lieferbaren PIC 16C84 ist. Für Anwendungen, bei denen Daten auch nach dem Ausschalten der Betriebsspannung erhalten bleiben müssen, verfügt der PIC 16F84 über ein

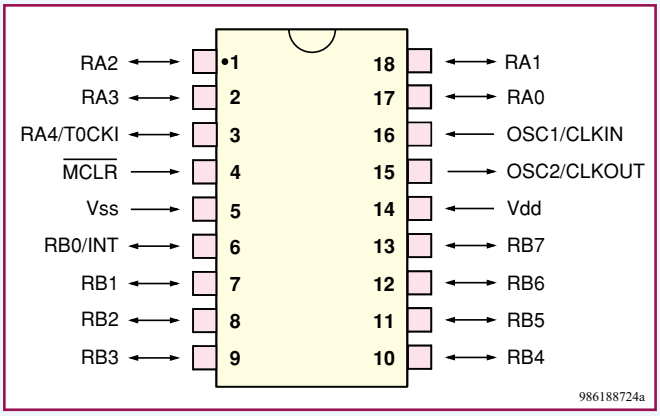

Bild 24: Anschlußbelegung des PIC 16F84

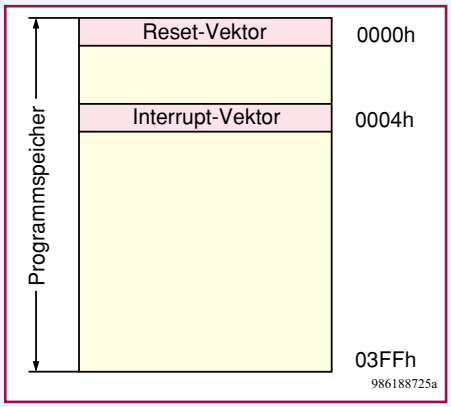

Bild 25: Aufteilung des Programmspeichers

internes EEPROM mit 64 Speicherstellen, in denen jeweils ein Byte abgelegt werden kann. Hier sind zum Beispiel Abgleichwerte für ein Meßgerät speicherbar. Der Hersteller garantiert für das EEPROM 1.000.000 Lösch- und Schreib-Zyklen, wobei die Daten mindestens 40 Jahre erhalten bleiben.

Als Besonderheit des 16F84 ist dessen Programmspeicher als Flash-ROM ausgeführt, der elektrisch lösch- und beschreibbar ist. So sind besonders für die Entwicklungsphase keine speziellen UV-löschbaren PICs erforderlich. Diese sind sehr teuer und der Löschvorgang nimmt stets mehrere Minuten in Anspruch.

Für das Löschen und Beschreiben des Programmspeichers verfügt der PIC 16F84 über ein serielles Interface. Da zur seriellen Programmierung nur wenige Pins erforderlich sind, ist es sogar möglich, den PIC innerhalb einer Anwendungsschaltung zu programmieren. Es ist lediglich dafür zu sorgen, daß die Pins des Interfaces während der Programmierung nicht durch die Anwendung beschaltet sind.

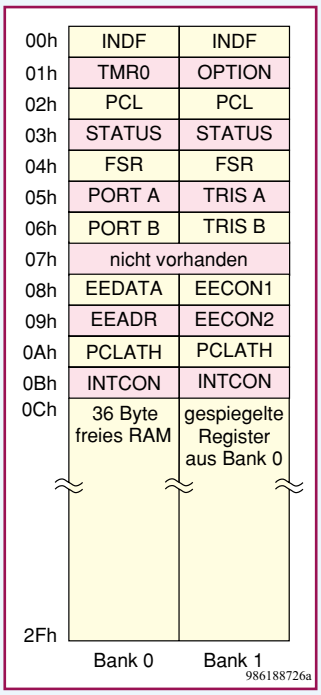

Bild 26: Datenspeicher des PIC 16F84

So ist es zum Beispiel auch möglich, die Software eines Gerätes jederzeit anzupassen, ohne den Mikrocontroller austauschen bzw. aus der Anwendung entnehmen zu müssen. Aus diesem Grund ist der PIC 16F84 auch bereits in einigen Chipkartentypen integriert. Damit besteht die Möglichkeit, die Karten individuell zu programmieren und einzusetzen.

Die wesentlichen Merkmale des PIC 16F84 sind:

- 35 Ein-Wort-Befehle
- Alle Befehle (bis auf Sprungbefehle) benötigen nur einen Zyklus (400 ns bei 10 MHz)
- Wählbarer Oszillator mit einer Taktfrequenz von 0 bis 10 MHz
- 14 Bit breite Befehle
- 1024 x 14 Bit interner Programmspeicher (Flash-ROM)
- 36 x 8 Bit Datenspeicher (RAM)
- 64 x 8 Bit Datenspeicher (EEPROM)
- 8-Ebenen-Hardwarestack
- 4 Interruptquellen:
  1. externer Interrupt über RB0/INT
  2. Timerüberlauf
  3. PORT B
  4. EEPROM-Schreiben beendet
- Integrierter Watch-Dog-Timer
- Integrierter Zähler/Teiler
- 13 I/O-Pins
- Betriebsspannung von 2,0 V bis 6,0 V
- Niedriger Stromverbrauch
- SLEEP-Modus
- Power-On-Reset
- Serielle Programmierung

Programmspeicher

Da der PIC16F84 mit einer Befehlsbreite von 14 Bit arbeitet, besteht der Programmspeicher aus 1024 Speicherstellen zu je 14 Bit. Die Aufteilung des Programmspeichers ist in Abbildung 25 dargestellt. In Gegensatz zu den PIC16C5x-Mikrocontrollern liegt der Reset-Vektor nicht am Ende des Programmspeichers, sondern am Anfang auf Adresse 0000h.

Der PIC 16F84 ist gegenüber den Vorgängern interruptfähig, so daß er bei Eintreffen eines Ereignisses (z. B. Überlauf des Timers) den normalen Programmablauf unterbrechen und eine bestimmte Funktion ausführen kann. Ist diese Funktion abgearbeitet, so wird der Programmablauf fortgesetzt. Dabei führt der Mikrocontroller bei einem Interrupt den Programmcode, der an der Adresse 0004h steht, aus. Hier befindet sich im Normalfall ein Sprungbefehl zur Interruptfunktion. Nach dem Beenden der Interruptfunktion erfolgt ein Fortsetzung des Programmablaufs an der Stelle des Hauptprogramms, an der der Interrupt zuvor das Programm unterbrochen hat. Die genaue Funktion der Interrupts erläutern wir später noch genauer.

Aufgrund der Befehlsbreite von 14 Bit kann man beim PIC 16F84 Unterprogramme im gesamten Bereich des Programmspeichers anordnen. Dagegen waren bei der PIC 16C5x-Familie Unterprogramme immer zwingend in der unteren Hälfte eines Speicherblocks zu lokalisieren.

Datenspeicher

Die Aufteilung des Datenspeichers ist in Abbildung 26 dargestellt. Wie bei der PIC16C5X-Familie sind auch hier im unteren Speicherbereich (00h bis 0Bh) die Register zur Steuerung untergebracht. Die 26 darauffolgenden Register bis zur Adresse 2Fh stehen zur freien Verfügung. Jede Speicherstelle kann 1 Byte (8 Bit Breite) aufnehmen.

Der Speicher ist auf zwei Seiten angeordnet, die durch das STATUS-Register auswählbar sind. Seite 1 enthält die gleiche Anzahl von Speicherstellen wie Seite 0, wobei die meisten Speicherstellen aus Seite 0 gespiegelt sind. D. h., auch wenn Seite 1 aktiv ist, kann man auf die Register auf Seite 0 zugreifen.

Einige Register sind jedoch nur über eine Speicherbank zu erreichen. Dazu zählen das OPTION- und das TRIS-Register, die auf Seite 1 als normale Register zu erreichen sind. Im Gegensatz zur PIC 16C5X-Familie, bei denen diese Register nur mit Hilfe eines Befehls beschreiibbar waren, sind diese beim PIC18F84 direkt zu beschreiben.

Das INDF-Register an der Adresse 00h,

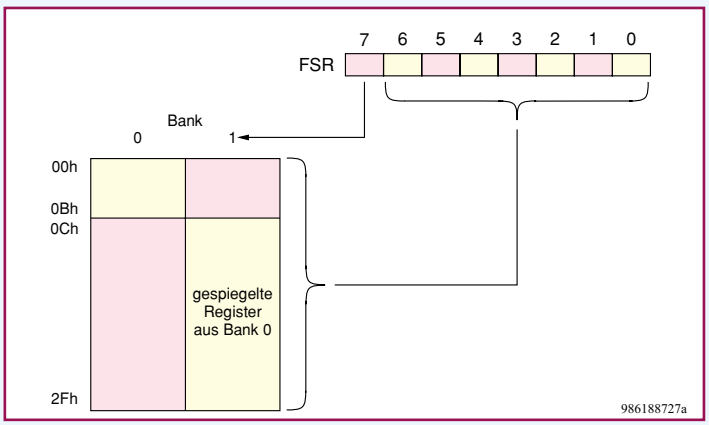

Bild 27: Funktion der einzelnen Bits des FSR-Registers

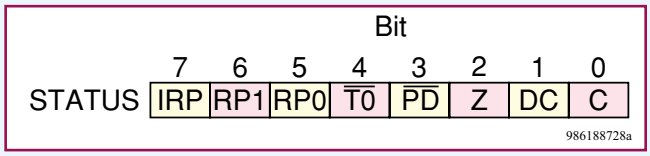

Bild 28: Das STATUS-Register

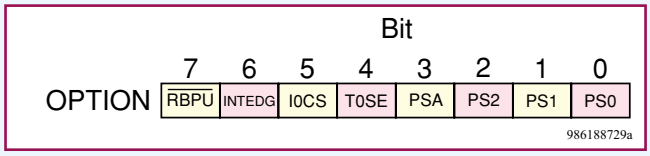

Bild 29: Das OPTION-Register

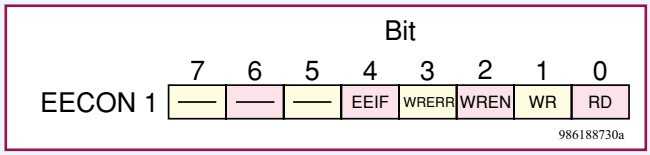

Bild 30: Bedeutung der einzelnen Bits des EECON1-Registers

dient, wie bereits von der PIC16C5x-Familie bekannt, zur indirekten Adressierung einer Speicherstelle, deren Adresse im FSR-Register steht. In Abbildung 27 ist die Funktion der einzelnen Bits des FSR-Registers dargestellt. Die unteren 7 Bit bestimmen dabei die Adresse, mit Bit 7 erfolgt die Auswahl der anzusprechenden Speicherbank.

Das STATUS-Register befindet sich auf der Adresse 03h. Abbildung 28 zeigt seinen Aufbau. Die Funktionen der unteren 5 Bits sind identisch mit denen der PIC 16C5x-Familie und sollen an dieser Stelle nicht erneut beschrieben werden. Das Bit RP0 dient zum Umschalten zwischen den beiden Speicherbänken. Ist das Bit gelöscht, so ist die Speicherbank 0 aktiv.

Die Bits RP1 und IRP sind für eventuelle Erweiterungen vorgesehen und sollen den Zugriff auf vergrößerte Datenspeicher ermöglichen. Diese werden beim PIC 18F84 jedoch nicht genutzt, man sollte sie nicht beschreiben (im gelöschten Zustand belassen).

Der Aufbau des OPTION-Registers ist in Abbildung 29 dargestellt. Neu hinzugekommen ist hier das $\overline{\text{RBPU}}$-Bit (Bit 7), das gelöscht werden muß, um interne Pull-Up-Widerstände für die Ports RB0 bis RB7 zu aktivieren.

Der Port RB0 ist zur Auslösung eines Interrupts nutzbar. Mit dem INTEDG-Bit (Bit 6) kann man dabei die Flanke wählen. Ist das Bit gesetzt, so löst eine steigende Flanke den Interrupt aus. Andernfalls führt eine fallende Flanke zum Interrupt.

Die Funktionen der übrigen Register entsprechen denen der PIC16C5X-Familie.

EEPROM

Der PIC 16F84 ist mit einem EEPROM ausgestattet, das 64 Byte umfaßt und sowohl beschreibbar als auch lesbar ist. Das EEPROM bietet sich an, wenn Daten zu speichern sind, die auch nach dem Abschalten der Betriebsspannung erhalten bleiben müssen. So können hier zum Beispiel das Speichern von Abgleichwerten für ein Meßgerät oder der Anzahl der Betriebsstunden erfolgen. Es ist auch möglich, das EEPROM direkt bei der Programmierung des PICs mit zu beschreiben. So sind bereits hier Parameter setzbar, die sich später jederzeit verändern lassen.

Auf das EEPROM kann dabei nicht direkt wie auf ein Register zugegriffen werden, sondern der Zugriff erfolgt über vier spezielle Register.

Dabei dienen die Register EECON1 und EECON2 der Steuerung des EEPROM-Zugriffs, das EEADR-Register der Adressierung der Speicherstelle und das EEDAT-Register der Übergabe des zu schreibenden oder gelesenen Bytes. Die Steuerregister liegen an der Adresse 08h und 09h in den Speicherbänken 0 und 1. Die Bedeutung der einzelnen Bits des EECON1-Registers ist in Abbildung 30 dargestellt.

Tabelle 5: Auslesen einer EEPROM-Speicherstelle

```
BCF     STATUS, RP0     ; Wechsel auf Bank 0
MOVLW   ADRESSE         ; EEPROM-Adresse in das W-Register kopieren
MOVWF   EEADR           ; Adresse in das EEADR-Register kopieren
BSF     STATUS, RP0     ; Wechsel auf Bank 1
BSF     EECON1, RD      ; Lesevorgang starten
BCF     STATUS, RP0     ; Wechsel auf Bank 0
MOVF    EEDATA, 0       ; Inhalt des EEDATA- in das W-Register kopieren
```

Bit 0 dieses Registers dient dem Auslesen einer EERPOM-Speicherstelle. Dazu werden zuerst die Adresse der auszulesenden Speicherstelle in das EEADR-Register geschrieben und anschließend das Bit 0 des EECON1-Registers gesetzt.

Hierdurch erfolgt der Start des Lesevorgangs, der Inhalt der Speicherstelle kann danach aus dem EEDATA-Register gelesen werden. Der Lesevorgang benötigt die Dauer eines Maschinenzyklus`, so daß der Controller nach dem Start des Lesevorgangs warten muß, bis der Inhalt der Speicherstelle ausgelesen ist.

Dabei erfolgt automatisch die Löschung von Bit 0 des EECON1-Registers durch den Controller.

Das Auslesen einer Speicherstelle erfolgt wie in Tabelle 5 dargestellt.

Der zum Auswerten der EEPROM-Speicherstelle benötigte Maschinenzyklus wird im Beispiel durch den Wechsel auf die Speicherbank 0 überbrückt.

Der Schreibvorgang ist etwas komplexer, da zum einen das Schreiben einer Speicherstelle bis zu 10 ms dauert und zum anderen einige Schutzmechanismen das versehentliche Beschreiben des EEPROMs erschweren.

Zuerst ist über das Setzen des WREN-Bits (Bit 2) des EECON1-Registers der Schreibzugriff auf das EEPROM zu ermöglichen. Nach dem Schreibvorgang sollte das Bit jedoch sofort wieder gelöscht werden, um ein späteres versehentliches Beschreiben zu verhindern.

Danach ist die Adresse der zu beschreibenden Speicherstelle in das EEADR-Regsiter und das zu schreibende Datenbyte in das EEDATA-Register zu kopieren. Vor dem Starten des Schreibvorganges müssen dann noch die Werte 55h und AAh in das EECON2-Register geschrieben werden, bevor der Schreibvorgang durch Setzen des WR-Bits (Bit 1) des EECON1-Registers startet.

Ist der Schreibvorgang beendet, so löscht der Mikrocontroller das WR-Bit. Es besteht aber auch die Möglichkeit, am Ende eines Schreibvorganges einen Interrupt auszulösen. Dabei muß der Interrupt freigeschaltet und eine entsprechende Interruptroutine vorhanden sein. Nach Beenden eines Schreibvorgangs erfolgt dabei das Setzen des EEIF-Bits (Bit 4) des EECON1-Registers. Das Bit muß innerhalb der Interruptroutine gelöscht werden, damit der Interrupt erneut auslösbar ist.

Zu beachten ist noch, daß während der Initialisierung eines Schreibvorganges die Interrupts zu sperren sind.

Das Beschreiben einer Speicherstelle erfolgt dabei wie in Tabelle 6 dargestellt.

Vor dem Schreiben jedes Bytes ist die Initialisierung des EECON2-Registers mit 55h und AAh erforderlich. Ebenso darf der

Tabelle 6: Beschreiben einer EEPROM-Speicherstelle

```
MOVLW   ADRESSE         ; EEPROM-Adresse in das W-Register kopieren
MOVWF   EEADR           ; Adresse in das EEADR-Register kopieren
MOVLW   DATEN           ; Datenbyte das W-Register kopieren
MOVWF   EEDATA          ; Datenbyte in das EEDATA-Register kopieren
BSF     STATUS, RP0     ; Wechsel auf Bank 1
BCF     INTCONJ, GIE    ; Interrupts sperren
BSF     EECON1, WREN    ; Schreibzugriff auf das EEPOM ermöglichen
MOVLW   055h
MOVWF   EECON2          ; 55h in das EECON2-Register schreiben
MOVLW   0AAh
MOVWF   EECON2          ; AAh in das EECON2-Register schreiben
BSF     EECON1, WR      ; Schreibvorgang einleiten
BSF     INTCON, GIE     ; Interrupts wieder freigeben
```

Schreibzugriff auf das EEPROM erst über das WREN-Bit gesperrt werden, wenn der letzte Schreibvorgang beendet ist.

Wird während eines Schreibvorganges ein Reset ausgeführt oder spricht der Watchdog an, so erfolgt keine korrekte Ausführung des Schreibvorganges. Das Setzen des WRERR-Bits (Bit 3) im EECON1-Register signalisiert diesen Zustand.

Interrupts

Der PIC 16F84 kann auf interne und externe Ereignisse reagieren und einen Interrupt auslösen, bei dem der normale Programmablauf unterbrochen und eine Interruptfunktion ausgeführt wird. Danach erfolgt die Fortsetzung des Hauptprogramms an der zuvor unterbrochenen Stelle.

Bei Auslösung eines Interrupts führt der Controller den Programmcode aus, der an der Speicherstelle 04h des Programmspeichers steht. Hier befindet sich im Regelfall ein Sprungbefehl zur Interruptfunktion.

Der PIC 16F84 kann durch 4 Ursachen einen Interrupt auslösen:

- Externer Interrupt am RB0/INT-Pin
- Überlauf des Timers
- Pegeländerung eines der Ports RB4 bis RB7
- EEPROM-Schreibvorgang beendet

Die Steuerung der Interruptfunktion erfolgt durch das INTCON-Register, das sich auf Adresse 0Bh befindet (Abbildung 31).

Die 4 Interruptquellen sind einzeln aktivierbar. Beim Auslösen eines Interrupts muß der Microcontroller anhand der gesetzten Interruptflags überprüfen, von welcher Quelle der Interrupt ausgelöst wurde, um die entsprechende Funktion auszuführen.

Die Interruptbearbeitung ist durch das GIE-Bit (Bit 7) ein- und ausschalbar. Ist das Bit gesetzt, so erfolgt die Ausführung der Interruptfunktion.

Die 4 Interruptquellen werden jeweils über ein Bit im INTCON-Register aktiviert. Ebenso erfolgt bei Auslösen des Interrupts das Setzen eines entsprechenden Interruptflags. Diese Flags sind durch die Software zurückzusetzen, um sie später erneut setzen zu können.

Befindet sich der Controller im SLEEP-Mode, so wird er bei aktiviertem und auftretendem Interrupt gestartet. Betrachten wir die vier Interrupt-Auslös-Ursachen näher:

Der **RB0/INT-Pin** kann einen Interrupt bei fallender oder steigender Flanke auslösen. Die Flankenart wird durch das Setzen des INTEDG-Bits (Bit 6) des OPTION-Registers bestimmt. Ist das Bit gelöscht, so führt eine fallende Flanke zu einem Interrupt. Das INTE-Bit aktiviert den Interrupt, bei Auftreten der Flanke erfolgt das Setzen des INTF-Bits.

Der **Timer** kann einen Interrupt auslösen, sobald der Zähler/Timer überläuft (Zählerstand wechselt von FFh auf 00h). Dabei wird das T0IF-Bit gesetzt, die Aktivierung erfolgt über das T0IE-Bit.

Wechselt der Pegel einer der Ports RB4 bis RB7, so setzt der Controller das RBIF-Bit. Das RBIE-Bit aktiviert die Interruptquelle.

Wie schon zuvor beschrieben, kann der Controller einen Interrupt auslösen, wenn der **EEPROM-Schreibvorgang beendet** ist. Die Aktivierung des Interrups erfolgt durch das EEIE-Bit. Nach dem Beenden eines Schreibvorganges wird das EEIF-Bit (befindet sich im EECON-Register) gesetzt.

Befehle

Die Assemblerbefehle des PIC 16F84 haben im Gegensatz zur PIC16C5X-Familie eine Befehlsbreite von 14 Bit. Die Syntax der Befehle bleibt jedoch gleich. Lediglich die Umsetzung in den Maschinencode, die der Assembler vornimmt, erfolgt anders.

Im Gegensatz zur PIC 16C5x-Familie sind die Befehle OPTION und TRIS nicht mehr vorhanden, da auch diese Register beim PIC 16F84 wie die anderen Register beschreibbar ausgeführt sind.

Hinzugekommen sind jedoch die Befehle in Tabelle 7 auf der nächsten Seite.

Damit ist die Beschreibung des PIC 16F84 abgeschlossen und im nächsten Teil der Artikelserie stellen wir ein kleines Programmiergerät für den PIC 16F84 vor. Das Gerät bietet zusätzlich einen Adapter, der anstelle des PIC 16F84 in die Anwenderschaltung eingesetzt werden kann und so die sofortige Überprüfung des Programms ermöglicht, ohne den PIC aus dem Programmiergerät in die Schaltung umstekken zu müssen.

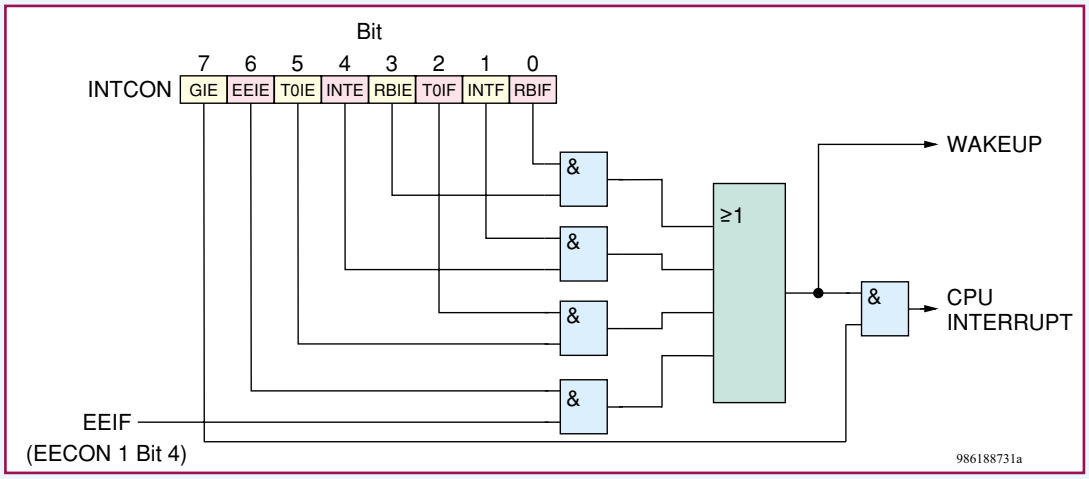

Bild 31: Das INTCON-Register

Tabelle 7: Zusätzliche Befehle des PIC 16F84

RETFIE

Syntax:	RETFIE
Funktion:	Rücksprung aus Interruptroutine
Status-Flags:	keine
Beschreibung:	Der Mikrocontroller kann beim Auftreten einiger Ereignisse einen Interrupt auslösen und eine spezielle Routine abarbeiten. Dabei wird das Hauptprogramm unterbrochen und die aktuelle Adresse auf dem Stack gesichert. Ist die Interruptfunktion beendet, so bewirkt dieser Befehl, daß die Adresse vom Stack geholt und das Hauptprogramm fortgesetzt wird.

Beispiel:

```
INT     ; Interruptfunktion
        ; hier steht der Programmcode, der bei einem
        ; Interrupt ausgeführt wird
        RETFIE;     Rücksprung zum Hauptprogramm
```

ADDLW

Syntax:	ADDLW k
Funktion:	Addiere Konstante zum W-Register
Staus-Flags:	Z, C, DC
Beschreibung:	Der Befehl addiert die Konstante k zum Inhalt des W-Registers und legt das Ergebnis im W-Register ab. Das C-Flag wird gesetzt, wenn ein Überlauf des 8Bit-Registers erfolgt und das DC-Flag zeigt einen Überlauf der unteren 4 Bits an. Ist das Ergebnis der Operation Null, so ist das Z-Flag gesetzt.

Beispiel:

```
; das W-Register beinhaltet 010h
ADDLW   015h   ; Konstante zum W-Register addieren
; das W-Register enthält (010h + 015h =) 025h
```

SUBLW

Syntax:	SUBLW k
Funktion:	W-Register von Konstante subtrahieren
Staus-Flags:	Z, C, DC
Beschreibung:	Der Befehl subtrahiert den Inhalt des W-Registers von der Konstanten k und legt das Ergebnis im W-Register ab. Ist das Ergebnis negativ, so ist wird das C-Flag gelöscht. Andernfalls wird das C-Flag gesetzt. Ist das Ergebnis der Operation Null, so ist das Z-Flag gesetzt.

Beispiel:

```
; das W-Register beinhaltet 010h
ADDLW   015h   ; W-Register von der Konstante subtrahieren
; das W-Register enthält (015h - 010h =) 05h
; da das Ergebnis positiv ist, wird das C-Flag gesetzt
```

RETURN

Syntax:	RETURN
Funktion:	Rücksprung aus Unterprogramm
Staus-Flags:	keine
Beschreibung:	Der Befehl beendet eine Unterfunktion, die mit einer CALL-Anweisung aufgerufen wurde. Im Gegensatz zum RETLW-Befehl wird hierbei keine Konstante im W-Register übergeben.

Beispiel:

```
; Programmablauf
CALL U_FKT      ; Unterprogarmmaufruf
; Rücksprung zum Hauptprogramm
;       ...
```

```
U_FKT   ; Unterprogrammablauf
;       ...
        RETURN    ; Unterprogramm beenden
```