8. RESET und INTERRUPT

8.1. Der Reset

Bei jeder Inbetriebnahme eines Mikrocontroller-Systems oder beim Drücken der Reset-Taste (oder beim Ansprechen des "Wachhundes" = Watchdogs) erfolgt ein **"Reset".**

Hierbei geschieht folgendes:

a) die verschiedenen SFR-Register werden mit genau festgelegten Bitkombinationen beschrieben, alle Flags werden gelöscht.

b**) sämtliche Bits aller Ports werden auf "1" gesetzt** (also führen alle Portbits High-Pegel. Durch diese Maßnahme sind **alle Ports** automatisch auf "**Eingabe**" geschaltet....).

c) der Programmzähler wird auf "0000" zurückgesetzt. **Dadurch beginnt der Controller zwangsweise bei der Programmadresse 0000 mit seiner Arbeit.**

Der Punkt c) erfordert nun bei der Programmierung einen wichtigen Schritt:

Nur die Adressen 0000 / 0001 / 0002 sind frei belegbar, da ab Adresse 0003 bereits die sogenannten **"Interrupt-Vektoren**" folgen. Deshalb **muss** jede Einschaltroutine an der Stelle 0000 mit ein **Sprungbefehl** beginnen (um über die Interrupt - Vektoren und das meist dahinter angeordnete Betriebssystem (=**"Monitorprogramm**") hinwegzukommen!).

Beispiel: a) in Assembler-Schreibweise:

```
ORG 0000H     ;Startanweisung
JMP 4000H     ;Hauptprogramm beginnt bei 4000H
```

b) zugehörige EPROM-Speicherinhalte:

```
0000:  02
0001:  00
0002:  40
```

(Achtung: schon der nächste Speicherplatz 0003H ist für die Bearbeitung des “Externen Interrupts IE0” reserviert!)

8.2. Interrupts

8.2.1 Grundlagen

Ein **Interrupt** bewirkt die **Unterbrechung** eines **gerade laufenden Programms**, um auf besondere und wichtige Ereignisse sofort zu reagieren.

Hierbei läuft folgender Vorgang ab:

a) Sobald der **Interrupt erkannt** ist, wird der gerade in Arbeit befindliche Befehl erst abgeschlossen und manchmal -- aber nicht immer!!!! -- das auslösende Interrupt-Flag gelöscht.

b) Die gerade geltende **Hauptprogramm-Adresse** wird gelesen, **um 1 erhöht** und in einem dafür **vorgesehen Speicher ("Stack") abgelegt**. Dadurch kann nach der Bearbeitung des Interrupts wieder mit dem nächsten Schritt des Hauptprogramms weitergemacht werden.

c) Entsprechend der Bezeichnung des Interrupts springt der Controller nun zum dafür gültigen **"Interrupt-Vektor**". Das sind die vorhin erwähnten **Programmspeicher-Adressen ab Adresse 0003H**.

d) Der "**Interrupt-Vektor**" selbst ist nun nichts anderes als ein **8 Byte langes Stück** des **Programmspeichers.** Darin stehen entweder direkt die **einzelnen Programmschritte** einer kurzen Interruptroutine oder der **Aufruf** (z. B. ACALL...) für ein **längeres Programm** (mit einer Adresse irgendwo weiter hinten im Programmspeicher).

Jede Interruptroutine beginnt (falls nötig) mit **"Rettungsbefehlen" für den Akku oder andere wichtige Register** ("PUSH A, PUSH PSW usw.), sagt dann dem Controller, was getan werden soll und endet mit **dem Zurückladen der geretteten Register-Inhalte** (POP A, POP PSW usw.).

Nie vergessen: **Die Interrupt - Routine muss in Assembler *immer* mit RETI (= return from interrupt**) abgeschlossen werden!

Übersichtstabelle der Interrupt-Vektoren beim 80C535

Quelle bzw. abgefragtes Flag	Vektor	Vektoradresse im EPROM
IE0	Externer Interrupt 0	0003H
TF0	Timer 0 - Interrupt (Überlauf)	000BH
IE1	Externer Interrupt 1	0013H
TF1	Timer 1 - Interrupt (Überlauf)	001BH
RI + TI	Interrupt der Seriellen Schnittstelle	0023H
TF2 + EXF2	Timer 2 - Interrupt	002BH
IADC	A / D - Wandler - Interrupt	0043H
IEX2	Externer Interrupt 2	004BH
IEX3	Externer Interrupt 3	0053H
IEX4	Externer Interrupt 4	005BH
IEX5	Externer Interrupt 5	0063H
IEX6	Externer Interrupt 6	006BH

8.2.2 Interrupt - Programmierung und zugehörige Register

Wichtig:

a) Werden **verschiedene** Interruptquellen benützt, so müssen sie (durch Setzen bestimmter Bits in bestimmten Registern) extra **freigegeben** werden. Außerdem gibt es einen **"Interrupt-Hauptschalter EAL".** Nur wenn er eingeschaltet ist, kann man mit Interrupts arbeiten! Diese Bits finden sich im wesentlichen in den beiden Registern

IEN0 und IEN1

b) Beim 80C535 gibt es **vier Prioritätsebenen** für die Interrupts (= Hierarchie bzw. Kommando-Ebene wie beim Militär). Dadurch wird der **"Vorrang" eines bestimmten Interrupts gegenüber den anderen** festgelegt und der Interrupt mit dem höheren Rang wird dann beim gleichzeitigen Auftreten mehrerer Interrupts zuerst bearbeitet! (Allerdings müssen sich **zwei Interrupts** (aus Platzgründen) immer **eine Prioritätsebene teilen)**.

Die Festlegung der Prioritäten geschieht durch das Setzen entsprechender Bits in den beiden Interrupt-Prioritätsregistern

IP0 und IP1

Achtung: diese beiden Register sind leider NICHT bit-adressierbar!!!

c) Sobald ein **Interrupt erkannt** wurde, wird ein zugehöriges **Flag gesetzt**. Die meisten dieser Flags finden sich im Interrupt-Request-Control-Register

IRCON

Die genauen Details und Zusammenhänge können den folgenden Blättern entnommen werden!

Hinweise zur eigentlichen Programmierarbeit

Die Assemblerprogrammierung unterscheidet zwischen den beiden Fällen:

a) Wird das Programm in das **EPROM eingebrannt**, dann beginnt unser CODE - Bereich schon bei 0000h. Wir lassen also unser Hauptprogramm (durch die Direktive „CSEG at 0000h" und dem Befehl „jmp START") in gewohnter Weise beginnen und organisieren unsere Interrupt - Bearbeitungen an den Interrupt - Vektoren entsprechend der obigen Tabelle jeweils mit „CSEG at....." (Beispiel für den A-D-Wandler: CSEG at 0043h).

b) Arbeiten wir dagegen mit KEIL-µvision2, dann schicken wir ja unser geschriebenes Programm über die serielle Schnittstelle zum Controllerboard, wo es üblicherweise ab Adresse 4000h angeordnet wird. Also verschieben wir einfach unsere Interrupt- Vektoren ebenfalls (durch eine CSEG at... - Direktive) und schreiben z. B. für unseren AD-Wandler-Interrupt:

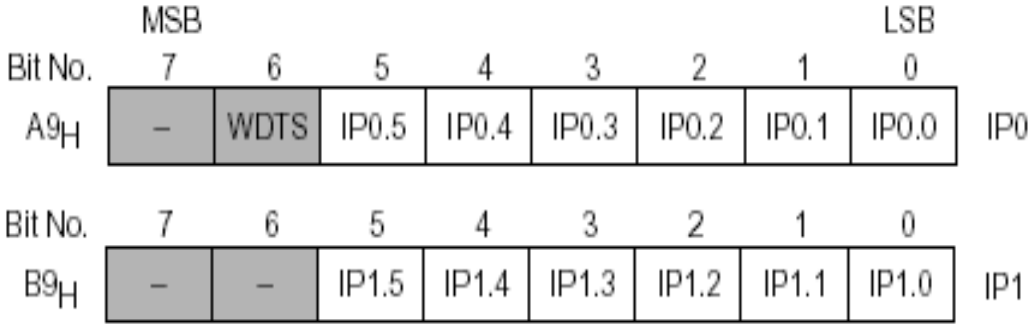
CSEG AT 4043h bzw. **CSEG AT 43h + OFFSET**, wenn mit „OFFSET EQU 4000h" gearbeitet wird.

Es folgen nun die Originalseiten aus dem Controllerhandbuch mit den Informationen zur Interruptprogrammierung.

7.1.3 Interrupt Priority Registers

The lower six bits of these two registers are used to define the interrupt priority level of the interrupt groups as they are defined in **table 7-1** in the next section.

Special Function Register IP0 (Address $A9_H$) **Reset Value : $X0000000_B$**
Special Function Register IP1 (Address $B9_H$) **Reset Value : $XX000000_B$**



The shaded bits are not used for interrupt control.

Bit	Function
IP1.x IP0.x	Interrupt group priority level bits (x=1-6, see **table 7-1**)

IP1.x	IP0.x	Function
0	0	Interrupt group x is set to priority level 0 (lowest)
0	1	Interrupt group x is set to priority level 1
1	0	Interrupt group x is set to priority level 2
1	1	Interrupt group x is set to priority level 3 (highest)

7.2 Interrupt Priority Level Structure

The following table shows the interrupt grouping of the C515 interrupt sources.

Table 7-1
Interrupt Source Structure

Interrupt Group	Associated Interrupts High Priority → Low Priority		Priority
1	External interrupt 0	A/D converter interrupt	High
2	Timer 0 overflow	External interrupt 2	↓
3	External interrupt 1	External interrupt 3	
4	Timer 1 overflow	External interrupt 4	
5	Serial channel interrupt	External interrupt 5	
6	Timer 2 interrupt	External interrupt 6	Low

Each pair of interrupt sources can be programmed individually to one of four priority levels by setting or clearing one bit in the special function register IP0 and one in IP1. A low-priority interrupt can be interrupted by a high-priority interrupt, but not by another interrupt of the same or a lower priority. An interrupt of the highest priority level cannot be interrupted by another interrupt source.

If two or more requests of different priority leveis are received simultaneously, the request of the highest priority is serviced first. If requests of the same priority level are received simultaneously, an internal polling sequence determines which request is to be serviced first. Thus, within each priority level there is a second priority structure determined by the polling sequence, as follows.

- Within one interrupt group the „left“ interrupt is serviced first
- The interrupt groups are serviced from top to bottom of the table.

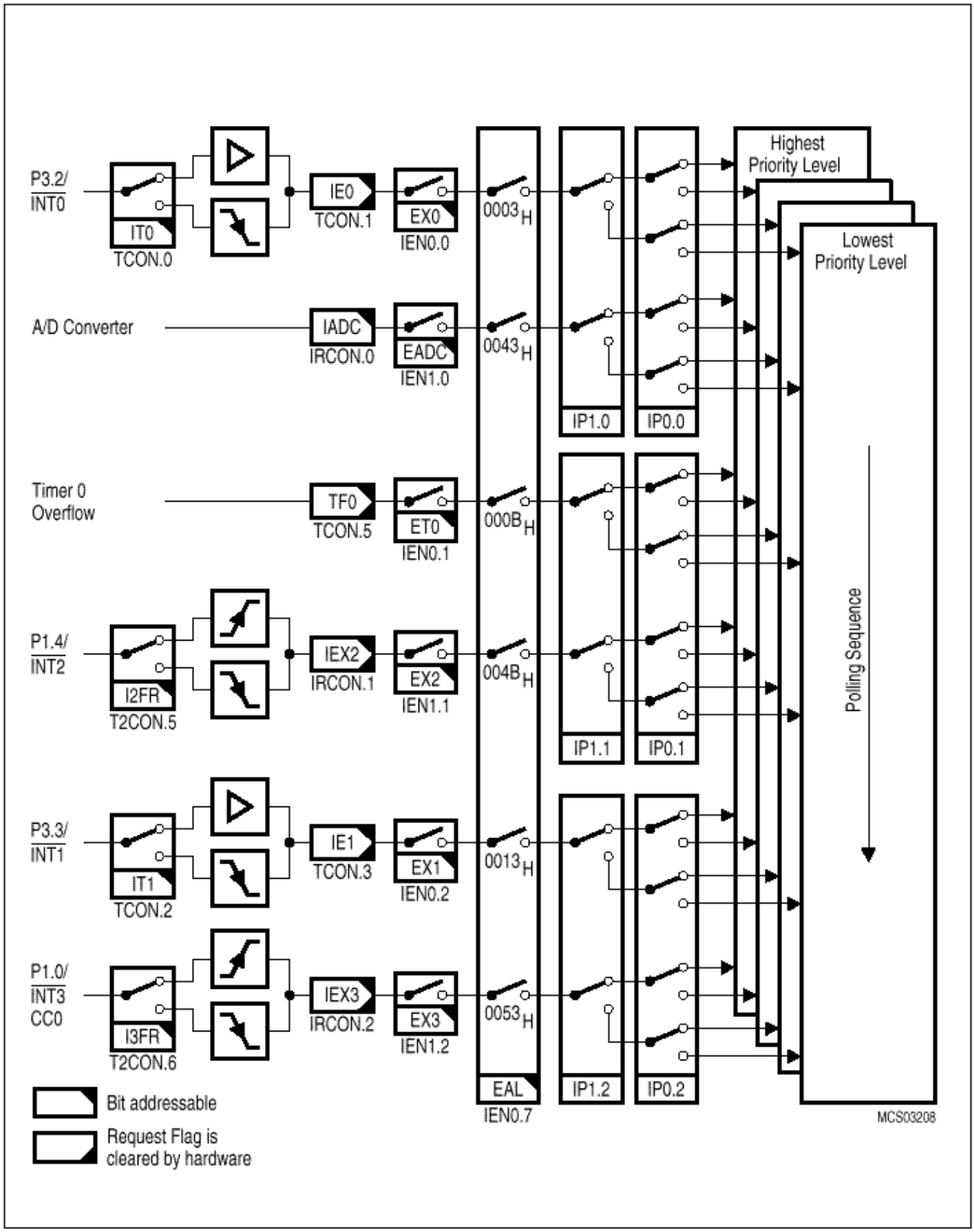


Figure 7-1
Interrupt Structure, Overview Part 1

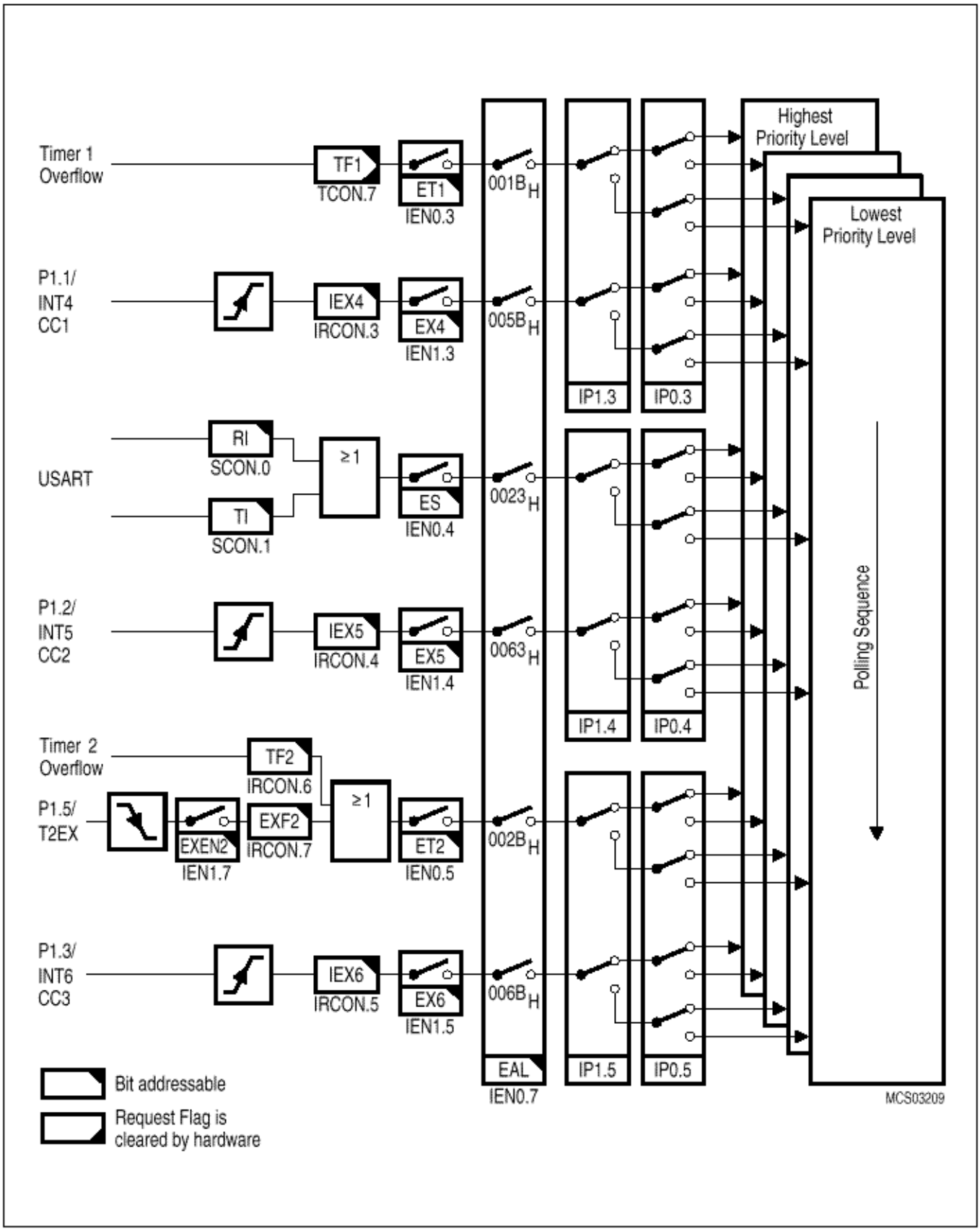
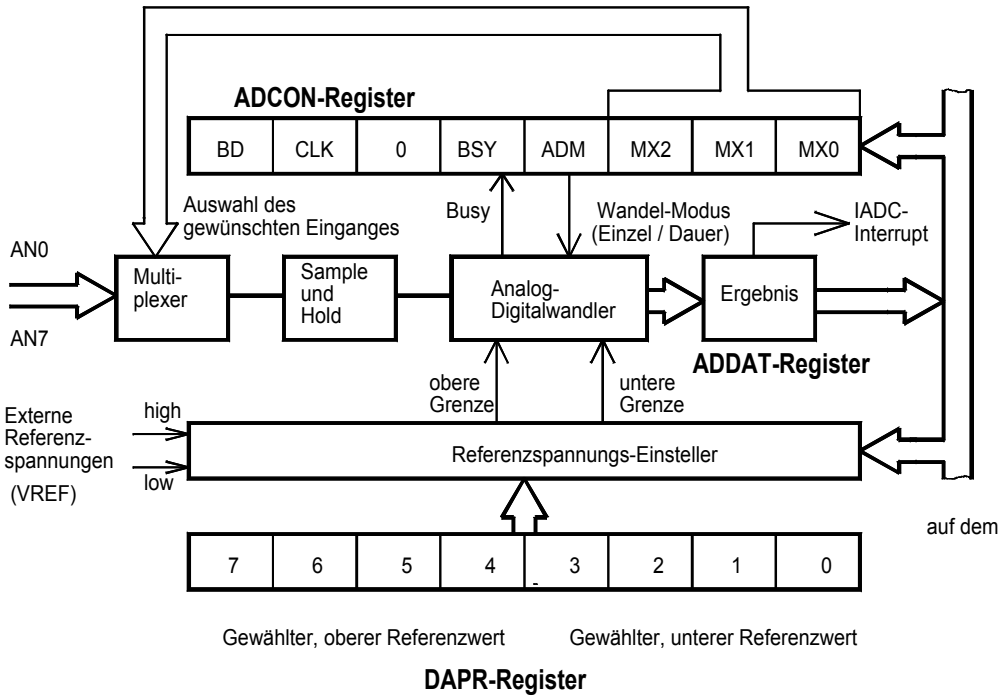


Figure 7-2
Interrupt Structure, Overview Part 2

9. Verarbeitung von Analogsignalen im 80C535

9.1. Arbeitsweise des A-D-Teiles auf dem Chip

Der A-D-Teil des 80C535 ist nach folgendem Übersichts-Schaltplan aufgebaut **und verträgt nur Analogsignale, die zwischen Null Volt und +5 Volt** liegen (Die Controller**eingänge** sollten deshalb mit geeigneten **Schutzschaltungen versehen sein!).**



Beim Einsatz dieses Wandlers müssen wir uns um 3 Register kümmern:

a) **ADCON-Register**

Mit den **unteren 3 Bits (MX0....MX2)** wird der **gewünschte Analogeingang** ausgewählt.

ADM bestimmt den **Wandler-Modus** (HIGH ergibt Dauermessung, LOW dagegen Einzelmessung).

BUSY ist HIGH, solange gewandelt wird und geht auf LOW, sobald das Ergebnis an das ADDAT - Register übergeben wird. Damit lässt sich leicht eine **Abfrageschleife für die Wartezeit** während der AD - Wandlung programmieren.

(Die übrigen Bits erfüllen andere Aufgaben: BD schaltet den internen Baudratengenerator für die Serielle Schnittstelle ein, CLK ermöglicht die Ausgabe der durch 12 geteilten Oszillatorfrequenz an Portpin P1.6, “0” heißt Null und muss immer auf Null bleiben).

b) **ADDAT - Register**

Hier kann mit einem MOV-Befehl das Wandelergebnis abgeholt werden. Der Controller meldet das **Wandlungsende** aber nicht nur durch das **Zurücksetzen des BUSY-Flags**, sondern auch durch das **Setzen des IADC-Interrupt-Flags** (es befindet sich im IRCON - Register).

Soll dadurch in eine Interrupt - Routine gesprungen werden, dann muss man vorher nicht nur den EADC - Interrupt, sondern auch die “globale Interrupt - Freigabe” per Software einschalten (sie finden sich in den Registern IEN0 und IEN1).

Daraus erkennt man, dass man als Programmierer das Ende des Wandelvorgang entweder durch dauerndes Abfragen des Busy - Flags oder durch Auslösung des IADC - Interrupts erkennen bzw. auswerten kann....

c) **DAPR - Register**

Dazu vorher einige Worte zum Eingangsspannungsbereich des AD - Wandlers:

Er benötigt immer **zwei Referenzpotentiale** zum korrekten Arbeiten:

a) der **tiefere** Wert heißt "V_{AGND}" und entspricht meist dem **Massepotential** (= Null Volt).

b) der **höhere** Wert heißt **"V_{AREF}"** und wird normalerweise durch die **Betriebsspannung (+5V)** gebildet.

Die (interne oder externe) Referenzquelle speist auf dem Chip einen **Spannungsteiler aus mehreren Widerständen**, zwischen dessen Abgriffen elektronisch umgeschaltet werden kann. Auf diese Weise lässt sich

1) der **"Endausschlag des Messbereiches**" bei kleinen Signalen verändern oder

2) die Auswertung von **kleinen Änderungen** bei einer **relativ großen Gleichspannung** erleichtern.

Der Spannungsbereich zwischen diesen beiden Referenzpunkten wird stets mit einer Genauigkeit von 8 Bit aufgelöst, d. h., er ergibt max. 255 verschiedene Digital-Werte in Form eines 8-Bit-Wortes. (Dieses 8-Bit-Wort steht am Ende des Wandelvorganges im Register **"ADDAT**" und muss dort abgeholt werden!).

Vorsicht: der Hersteller verlangt **mindestens 1 Volt Unterschied** zwischen den beiden Referenzpunkten. Damit beträgt die **feinste und kleinste Auflösung bei einer Messung mindestens ca. 4 mV** bei 255 Stufen.....

Bei hohen Präzisionsanforderungen oder starken Störungen auf der Versorgungsleitung kann auch eine **getrennte, externe Referenzspannungsquelle** benützt werden. Dazu zieht man 2 Jumper auf dem Board und führt die externe Spannung über den entsprechenden Steckverbinder zu.

Die **Programmierung der ausgewählten Referenzpunkte** sowie der **Beginn der A-D-Wandlung** erfolgt immer durch **Beschreiben des DAPR - Registers** mit einem entsprechenden Codewort!

Für die Auswahl der beiden Referenzpunkte gilt dafür folgende Codewort-Tabelle:

Oberes Nibble des DAPR	**Oberer Ref.-Punkt**	**Unteres Nibble des DAPR**	**Unterer Ref.-Punkt**
0000	5,0V	0000	0V
0001	-----	0001	0,3125V
0010	-----	0010	0,625V
Oberes Nibble des DAPR	**Oberer Ref.-Punkt**	**Unteres Nibble des DAPR**	**Unterer Ref.-Punkt**
0011	-----	0011	0,9375V
0100	1,25V	0100	1,25V
0101	1,5625V	0101	1,5625V
0110	1,875V	0110	1,875V
0111	2,1875V	0111	2,1875V
1000	2,5V	1000	2,5V
1001	2,8125V	1001	2,8125V

1010	3,125V	1010	3,125V
1011	3,4375V	1011	3,4375V
1100	3,75V	1100	3,75V
1101	4,0625V	1101	------
1110	4,375V	1110	------
1111	4,6875V	1111	------

9.2. Programmier-Beispiel “AD_WAND1” (Verwendung des BUSY - Flags)

Programmaufgabe:	Die an **Eingang AD0 anliegende Gleichspannung** soll mit dem Integrierten A-D-Wandler **pausenlos gemessen** und das Ergebnis an **Port 5** ausgegeben werden. Dort wird dann die Zusatz-Platine mit den 8 LED angeschlossen, um die einzelnen Bits des Messergebnisses sichtbar zu machen.Das Ende jedes Messvorgangs soll durch Abfragen des BUSY - Flags erkannt werden. Schreiben Sie erst eine Assembler-, anschließend eine C-Version dieses Programms.

Die veränderbare Spannung für den Wandlereingang bezieht man von der Zusatzplatine mit dem D-A-Wandler. Auf ihr befinden sich 8 Einstellpotentiometer, die an +5V angeschlossen sind und die abgegriffenen Spannungen an einen 10-poligen Pfostenfeld-Verbindungsstecker weitergeben.

Assembler - Lösung für AD_WAND1 (File: ad_wand1.a51):

```
;Dieses Programm misst die Gleichspannung an
;Portpin AN0 und übergibt das Ergebnis an
;Port P1. Dort ist die LED-Kette angeschlossen

;Programmiert am 10.01.2005 durch G. Kraus
;-------------------------------------------------------------------------------------------------
$nomod51                                ;8051-Modus ausschalten
$include(reg515.inc)                    ;Dafür 80535-Modus wählen

Offset equ 4000h                        ;Programme beginnen bei 4000h

?STACK  SEGMENT IDATA                   ;Stack-Deklaration
        RSEG ?STACK
        DS 5

CSEG at 0 + Offset                      ;Reset-Location
        jmp START

;-------------------------------------------------------------------------------------------------
PROG1   SEGMENT CODE                    ;Dieses Code-Segment heißt "PROG1"
        RSEG PROG1

START:  mov SP,#?STACK-1                ;Stack initialisieren
        mov ADCON,#00000000B            ;Eingang AN0 und Einzelmessung wählen
LOOP:   mov DAPR,#00H                   ;Start der A-D-Wandlung
        jb BSY,$                        ;Wandlungsende abwarten
        mov P1,ADDAT                    ;Ergebnis an P1 übergeben
        jmp LOOP                        ;Schleife drehen
;-------------------------------------------------------------------------------------------------
end
```

C - Lösung für AD_WAND1 (File: ad_wand1.c):

```
/*ad_wand1.c
Dieses Programm misst die Spannung an Portpin AN0 und übergibt das Ergebnis an Port P1 */

// Erstellt am 10-01-2005 durch G. Kraus

#include <reg515.h>
#include <stdio.h>

sfr ausgang=0x90;                       // Spezialfunktion-Register P1 heißt nun "ausgang"

void main (void)

{       ADCON=0x00;                     // Eingang AN0 und Einzelmessung wählen
        while(1)                        // Endlosschleife
        {       DAPR=0x00;              // Wandlerstart mit Messbereich 0...+5V
                while(BSY==1);          // Wandlungsende abwarten
                ausgang=ADDAT;          // Ergebnis an Port 1 ausgeben
        }
}
```

9.3. Programmierbeispiel „ad_wand2“ (mit Interrupt - Steuerung)

Programmaufgabe:	Die an **Eingang AD0 angelegte Gleichspannung** soll mit dem Integrierten A-D-Wandler **pausenlos gemessen** und das Ergebnis an **Port 5** ausgegeben werden. Dort wird dann die Zusatzplatine mit den 8 LEDs angeschlossen, um die einzelnen Bits des Messergebnisses sichtbar zu machen. Der Messablauf soll durch einen **Interrupt** gesteuert werden. Das Programm soll wieder in Assembler und C geschrieben werden

Nun sind folgende weiteren Maßnahmen erforderlich:

a) Der **Interrupt** muss **freigegeben und programmiert** werden.

b) Es ist eine “**Interrupt-Service-Routine**” nötig.

c) Der **Interrupt - Vector** muss in den RAM-Bereich (ab 4000H) **verschoben** werden.

Assembler - Lösung für ad_wand2 (File: ad_wand2.a51)

```
;Dieses Programm misst die Gleichspannung an
;Portpin AN0 unter Verwendung des AD-Interruptes
;und übergibt das Ergebnis an Port P1.
;Dort ist die LED-Kette angeschlossen

;Programmiert am 10.01.2005 durch G. Kraus
;-------------------------------------------------------------------------------------------------------------------
$nomod51                                    ;8051-Modus ausschalten
$include(reg515.inc)                        ;Dafür 80535-Modus wählen

Offset equ 4000h                            ;Programme beginnen bei 4000h

?STACK   SEGMENT IDATA                      ;Stack-Deklaration
    RSEG ?STACK
    DS 5

CSEG at 0 + Offset                          ;Reset-Location
    jmp START

CSEG at 43h + Offset                        ;Interrupt-Vektor verschieben
    jmp ADW_INT
;=====================================================================
PROG1    SEGMENT CODE                       ;Dieses Code-Segment heißt "PROG1"
         RSEG PROG1

START:   mov SP,#?STACK-1                   ;Stack initialisieren
         mov ADCON,#00000000B               ;Eingang AN0 und Einzelmessung wählen
         clr IADC                           ;Interrupt-Bit des AD-Wandlers löschen
         setb IEN1.0                        ;Freigabe d. AD-Wandler-Interrupts (Bit IEN1.0)
         mov IP0,#00000001B                 ;IADC auf Prioritätsebene 3 legen
         mov IP1,#00000001B    ;
         setb EAL                           ;Freigabe der Interruptsteuerung durch
                                            ;Setzen des "EAL"-Flags IEN0.7
         mov DAPR,#00H                      ;Start der A-D-Wandlung
         jmp $                              ;Programm dreht Warteschleife und wartet auf
                                            ;den Interrupt des AD-Wandlers
;=====================================================================
ADW_ISR  SEGMENT CODE                       ;ISR für AD-Interrupt
             RSEG ADW_ISR

ADW_INT:     clr IADC                       ;auslösendes AD-Interruptflag löschen
             mov P1,ADDAT                   ;Messergebnis an Port 1 weitergeben.
             mov DAPR,#00h
             reti                           ;Zurück aus der Interrupt-Routine

;-------------------------------------------------------------------------------------------------------------------
end
```

C - Lösung für ad_wand2 (File: ad_wand2.c)

```
/*adwand2.c
Dieses Programm misst die Spannung an Portpin AN0 und übergibt das Ergebnis an Port P1 */

// Erstellt am 10-01-2005 durch G. Kraus

#include <reg515.h>
#include <stdio.h>

sfr ausgang=0x90;                    //Spezialfunktion-Register P1 heißt nun "ausgang"

void ISR_ADW(void);                  //Prototypen anmelden

void main (void)

{       ADCON=0x00;                  //Eingang AN0 und Einzelmessung wählen
        IADC=0;                      //Interrupt-Bit löschen
        EAL=1;                       //Interrupt-Hauptschalter auf "EIN"
        EADC=1;                      //ADW-Interrupt freigeben
        DAPR=0x00;                   //Wandlerstart mit Messbereich 0.....+5V
        while(1);                    //Endlossschleife
}

void ISR_ADW(void)          interrupt 8          //AD-Wandler hat Interrupt-Index 8
{       IADC=0;                                  //Flag löschen
        ausgang=ADDAT;                           //Wandelergebnis an Port 1 ausgeben
        DAPR=0x00;                               //Neuer Start des Wandlers
}
```

Reihenfolge zur Bestimmung des „Interrupt – Index“ (hier : „interrupt 8“) bei C-Programmen:

a) Zuerst entnimmt man der Interrupt-Vektor-Tabelle die Einsprungadresse für den vorgesehenen Interrupt:

Quelle bzw. abgefragtes Flag	Vektor	Vektoradresse im EPROM
IE0	Externer Interrupt 0	0003H
TF0	Timer 0 - Interrupt (Überlauf)	000BH
IE1	Externer Interrupt 1	0013H
TF1	Timer 1 - Interrupt (Überlauf)	001BH
RI + TI	Interrupt der Seriellen Schnittstelle	0023H
TF2 + EXF2	Timer 2 - Interrupt	002BH
IADC	A / D - Wandler - Interrupt	0043H
IEX2	Externer Interrupt 2	004BH
IEX3	Externer Interrupt 3	0053H
IEX4	Externer Interrupt 4	005BH
IEX5	Externer Interrupt 5	0063H
IEX6	Externer Interrupt 6	006BH

Das ist für den Analog-Digital-Wandler die Vektoradresse 43h.

a) Dann wandelt man diese als Hex-Zahl angegebene Adresse in eine Dezimalzahl um:

   43h entspricht **4** x 16 + **3**x1, also der Dezimalzahl „67“.

b) Diese Dezimalzahl teilt man nun durch 8 und verwendet die Stelle vor dem Komma als Interrupt-Index:

   67 : 8 = 8,375 ..... das ergibt **„8“** als Index.

9.4. Programmierung eines Digitalvoltmeters mit LCD-Display-Anzeige

Erweitern Sie das Programm um eine Displayanzeige. In der ersten Zeile steht das Wort "**Spannung:**".
In der zweiten Zeile soll das **Ergebnis in Volt mit drei Nachkommastellen** angezeigt werden.

Dazu müssen Sie in einem ersten Teil den Messvorgang sowie die Umrechnung des Messwertes von Hexadezimal (= Ausgabeform des AD-Wandlers) in BCD und anschließender Umcodierung der einzelnen Ziffern in ASCII (= gewünschte Form für den LCD-Display-Controller) programmieren.

Der aus dem letzten Kapitel bekannte Modul „LCD_CTR.C" wird dann zusätzlich in die Projektliste von KEIL-µvision1 aufgenommen. Er soll die komplette Display-Ausgabe ausführen.

Beide Files werden dann vom Linker miteinander verknüpft und so ein lauffähiges Gesamtprogramm erzeugt.

Teil 1: Mess- und Umcodierungsmodul

```
/*adwand2.c
Erstellt am 10.01.2005 durch G. Kraus

Die am Analogeingang liegende Gleichspannung wird vom internen A-D-Wandler gemessen und das Ergebnis von
ADDAT an Port P1 übergeben. Dort ist eine LED-Kette angeschlossen, die das Datenbyte optisch anzeigt.
Außerdem ist an Port P5 das LCD-Display angeschlossen, bei dem in der ersten Zeile der Text "Messwert: ", in der
zweiten Zeile dagegen die gemessene Spannung (in Volt, mit drei Nachkommastellen) angezeigt  wird.*/

/* Achtung: In der Projektliste muss außer diesem Modul noch der Anzeige-Modul "LCD_01.C" oder "LCD_CTRL:c"
eingetragen werden, damit der Linker alles zu einem lauffähigen Gesamtprogramm verbinden kann. */

#include <reg515.h>
#include <stdio.h>

sfr ausg_led=0x90;                  // Spezialfunktion-Register P1 heißt nun "ausgang". Hier die LED-Kette anschließen!

void ISR_ADW(void);                 // Prototyp: Interrupt-Service-Routine des AD-Wandlers
void zeit_s(void);                  // Prototypen der vom Anzeige-Modul verwendeten Funktionen
void zeit_l(void);
void lcd_com(void);
void lcd_ein(void);
void lcd_ini1(void);
void switch_z1(void);
void switch_z2(void);
void show_text(char *ptr);

code unsigned char zeile_1[21]=     {"Spannung:          "};            // Text für Zeile 1 ins EPROM schreiben

unsigned int z;

void main(void)
{
        lcd_ein();                          // Einschaltroutine des Displays
        lcd_ini1();                         // Function Set für Display
        switch_z1();                        // Umschalten auf Zeile 1
        show_text(zeile_1);                 // Text  „Messwert: " in Zeile 1 anzeigen

        ADCON=0x00;                         // Eingang AN0 und Einzelmessung wählen
        IADC=0;                             // Interrupt - Bit löschen
        EAL=1;                              // Alle Interrupts freigeben
        EADC=1;                             // ADW - Interrupt freigeben
        DAPR=0x00;                          // Wandlerstart
        while(1);                           // Endlos - Schleife
}

//-------------------------------------------------------

void ISR_ADW(void)          interrupt 8             // AD-Wandler hat Interrupt-Index 8

{         char zeile_2[21]={"U = 0,000V        "};

        IADC=0;                                     // Flag löschen
        ausg_led=ADDAT;                             // Wandelergebnis an Port 1 an LEDs übergeben
```

```
        z=ADDAT;                                        //Umkopieren in INT-Variable z
        zeile_2[8]=((z*5000)/255) % 10 + 0x30;          //Dritte Nachkommastelle berechnen
        zeile_2[7]=((z*(5000/10))/255) % 10 +0x30;      //Zweite Nachkommastelle berechnen
        zeile_2[6]=((z*(5000/100))/255) % 10 +0x30;     //Erste Nachkommastelle berechnen
        zeile_2[4]=((z*(5000/1000))/255) % 10 + 0x30;   //Stelle vor dem Komma berechnen

        switch_z2();                    //auf Zeile 2 schalten
        show_text(zeile_2);             //Messergebnis anzeigen

        DAPR=0x00;                      //Neuer Start des Wandlers
}
```

Und hier zur Erinnerung nochmals der zugehörige Anzeigemodul „LCD_CTRL_new.c“:

```
/*-------------------------------------------------------------------
LCD_CTRL_new.c
C-Programm-Modul  zur Ausgabe von Texten auf dem LCD-Display
02.03.2005 / G. Kraus
-------------------------------------------------------------------*/
#include <reg515.h>
#include <stdio.h>

void zeit_s(void);
void zeit_l(void);
void lcd_com(void);
void lcd_ein(void);
void lcd_ini1(void);
void switch_z1(void);
void switch_z2(void);
void switch_z3(void);
void switch_z4(void);
void show_text(char *ptr);
void show_char(char *zeichen);

sfr ausg_lcd=0xf8;                      // Port 5 als Display-Ausgang

sbit enable=ausg_lcd^4;                 // Portpin P5.4 ist "enable"
sbit RW_LCD=ausg_lcd^5;                 // Portpin P5.5 ist "READ - WRITE"
sbit RS_LCD=ausg_lcd^6;                 // Portpin P5.6 ist "Register - Select"

/*-------------------------------------------------------------------
Zusatzfunktionen
-------------------------------------------------------------------*/

void zeit_s(void)                       // Funktion „kurze Wartezeit“ gibt etwa 100 Mikrosekunden
{
        unsigned char x;
        for(x=0;x<=30;x++);
}


void zeit_l(void)                       // Funktion „lange Wartezeit“ ergibt 4ms
{       unsigned int x;
        for(x=0;x<=1000;x++);
}


void lcd_ein(void)                      // Einschaltroutine für Display
{       unsigned char x,y;
        for(x=0;x<=2;x++)               // mindestens 3 x 4ms = 12 ms warten
        {       zeit_l();
        }

        ausg_lcd=0x03;                  // Dreimal nacheinander dasselbe Codewort
        for(y=0;y<=2;y++)               // “0011B”  =  0x03 mit Wartepausen dazwischen zum Display schicken
        {       enable=1;               // Damit wird der “Function set” verwirklicht
                for(x=0;x<2;x++);
                enable=0;
                zeit_l();
        }
```

```
}

void lcd_ini1(void)                     // Initialisierung des Displays
{       ausg_lcd=0x02;                  // „0010B“  = 0x02 ergibt 4-Bit-Darstellung beim Display

        lcd_com();                      // Code senden

        ausg_lcd=0x02;                  // Oberes Nibble für „4 Bit / 2 Zeilen / 5x7 dots“

        lcd_com();                      // losschicken

        ausg_lcd=0x08;                  // Unteres Nibble für „4 Bit / 2 Zeilen / 5x7 dots“

        lcd_com();                      // losschicken
        ausg_lcd=0x00;                  // Oberes Nibble von „Display OFF“

        lcd_com();                      // losschicken

        ausg_lcd=0x08;                  // Unteres Nibble von „Display OFF“

        lcd_com();                      // losschicken

        ausg_lcd=0x00;                  // Oberes Nibble von „Clear all“

        lcd_com();                      // losschicken

        ausg_lcd=0x01;                  // Unteres Nibble von „Clear all“

        lcd_com();                      // losschicken

        zeit_l();                       // Beim Löschen ist hier eine lange Wartezeit nötig

        ausg_lcd=0x00;                  /* Oberes Nibble von „Entry mode“ (Cursor steht, Display wandert
                                           nach rechts */

        lcd_com();                      // losschicken

        ausg_lcd=0x06;                  /* Unteres Nibble von „Entry mode“ (Cursor steht, Display wandert
                                            nach rechts */

        lcd_com();                      // losschicken

        ausg_lcd=0x00;                  // Oberes Nibble von „Display ON, Cursor OFF, Kein Blinken“

        lcd_com();                      // losschicken

        ausg_lcd=0x0c;                  // Unteres Nibble von „Display ON, Cursor OFF, Kein Blinken“

        lcd_com();                      // losschicken
}

void lcd_com(void)                      // Übertragungs-Routine für Daten zum Display
{       unsigned char x;
        enable=1;                       // ENABLE für einige Mikrosekunden auf HIGH
        for(x=0;x<2;x++);
        enable=0;                       // ENABLE wieder zurück auf LOW
        zeit_s();                       // Kurze Wartezeit aufrufen
}

void switch_z1(void)                    // Schalte auf Zeile 1 um und stelle den Cursor an den Anfang der Zeile
{       ausg_lcd=0x08;                  // Erforderliches Codewort hat als oberes Nibble „1000B“ = 0x08
        lcd_com();                      // losschicken
        ausg_lcd=0x00;                  // Erforderliches Codewort hat als unteres Nibble „0000B = 0x00
        lcd_com();                      // losschicken
}

void switch_z2(void)                    // Schalte auf Zeile 2 um und stelle den Cursor an den Anfang der Zeile
```

```
{       ausg_lcd=0x0c;                  // Erforderliches Codewort hat als oberes Nibble „1100B“ = 0x0C
        lcd_com();                      // losschicken
        ausg_lcd=0x00;                  // Erforderliches Codewort hat als oberes Nibble „0000“ = 0x00
        lcd_com();                      // losschicken
}

void switch_z3(void)                    // Schalte auf Zeile 3 um und stelle den Cursor an den Anfang der Zeile
{       ausg_lcd=0x09;                  // Erforderliches Codewort hat als oberes Nibble „1001B“ = 0x09
        lcd_com();                      // losschicken
        ausg_lcd=0x04;                  // Erforderliches Codewort hat als unteres Nibble „0100B = 0x04
        lcd_com();                      // losschicken
}


void switch_z4(void)                    // Schalte auf Zeile 4 um und stelle den Cursor an den Anfang der Zeile
{       ausg_lcd=0x0D;                  // Erforderliches Codewort hat als oberes Nibble „1101B“ = 0x0D
        lcd_com();                      // losschicken
        ausg_lcd=0x04;                  // Erforderliches Codewort hat als oberes Nibble „0100“ = 0x04
        lcd_com();                      // losschicken
}


void show_text(char *ptr)               // Anzeige eines Textes, als Array vorgegeben
{
        while(*ptr)                     /* Startadresse des Arrays wird übergeben. Schleife wird solange
                                           wiederholt, bis Code „0x00“ (entspricht dem Zeichen „\0“) gefunden
                                           wird.
        {
                ausg_lcd=(*ptr/0x10)|0x40;      /*  Oberes Nibble des Zeichen-Bytes wird um 4 Stellen nach rechts
                                                    geschoben und anschließend um die erforderlichen Steuersignale
                                                    ergänzt. */

                lcd_com();              //  Oberes Nibble losschicken

                ausg_lcd=(*ptr&0x0f)|0x40;      /*  Unteres Nibble des Zeichen-Bytes wird durch Löschen des
                                                    oberen Nibbles gewonnen und anschließend um die erforderlichen
                                                    Steuersignale ergänzt. */

                lcd_com();              //  Unteres Nibble losschicken

                ptr++;                  //  Nächstes Zeichen des Arrays adressieren
        }
}


void show_char(char *ptr)               //  Anzeige eines einzigen ASCII-Zeichens. Adresse wird übergeben.
{
        ausg_lcd=(*ptr/0x10)|0x40;      /*  Oberes Nibble des Zeichen-Bytes wird um 4 Stellen nach rechts
                                            geschoben und anschließend um die erforderlichen Steuersignale
                                            (für Datenübertragung) ergänzt. */

        lcd_com();                      //  Oberes Nibble losschicken

        ausg_lcd=(*ptr&0x0f)|0x40;      /*  Unteres Nibble des Zeichen-Bytes wird durch Löschen des
                                            oberen Nibbles gewonnen und anschließend um die erforderlichen
                                            Steuersignale (für Datenübertragung) ergänzt. */

        lcd_com();                      //  Unteres Nibble losschicken
}
```