Mikrocontroller-Programmierung mit dem

neuen Board „EST-ATM1“

der Elektronikschule Tettnang unter

und

Band 2: Erfolgreiche „C“ – Programmierung anhand getesteter Beispiele und Projekte

Version 2.3

Von
Gunthard Kraus, Oberstudienrat
Elektronikschule Tettnang

15. Juni 2008

Änderungshinweise:

==========

Software:

Die Firma ATMEL hat die verwendete Programmiersoftware „FLIP“ *am 1. Juni 2008 wesentlich geändert*. Deshalb gilt diese neue Einsteiger-Anleitung ab

„Flip 3_3_1 mit integrierter JAVA-Umgebung (*)“

Bitte gleich diesen Software-Stand aus dem Internet (oder von der ATMEL-ATM1-CD) herunterladen und auf dem eigenen Rechner installieren!

(*) = Dadurch wird die Software auf unterschiedlichen Plattformen (z. B. auch unter LINUX) lauffähig, aber nach dem Programmstart dauert es etliche Zeit, bis die JAVA-Umgebung vollständig installiert ist und sich etwas tut…Außerdem stieg die Größe des Programmpakets von ca. 2 MB auf über 20MB an.

==========

Hardware:

1) Die beiden Widerstände **R5 und R4** (bisher: 6,8k und 10k) wurden am 14. Juni 2008 auf **680Ω und 1kΩ vermindert**, da der integrierte AD-Wandler des Controllers beim Wandelvorgang die mit diesen Widerständen erzeugte Referenzspannung zu stark belastete. Durch diese Änderung bleibt die Abweichung unter 2%.

---

2) Die an einen Analogport angelegte Spannung wird nur während eines Messvorganges durch die Wandlerelektronik des Controllers ebenfalls stark belastet. Deshalb sollte der Innenwiderstand der zugehörigen Spannungsquelle (z. B. Sensor) deutlich **kleiner als 5 kΩ** sein.

---

3) Der Vorwiderstand für die grüne POWER-LED in der Stromversorgung wurde auf **1kΩ** erhöht.

Aber jetzt geht es wirklich los….

Inhaltsverzeichnis Seite

1. Vorbemerkungen

Dieses Manuskript soll zeigen, wie unter „C“ die verschiedensten praktischen Anwendungen mit unserem neuen ATMEL-Board „ATM1“ verwirklicht werden können. Aus dem bisherigen Unterricht und Umgang mit dem 80C535-Board heraus lagen bereits viele Beispiele vor, die auf das neue Board portiert und dann dort ausgetestet wurden. Sie sind alle garantiert lauffähig und sollen sowohl bei der eigenen Arbeit (als Nachschlagewerk) helfen wie auch zu neuen Projekten und Entwürfen anregen.

Folgende Dinge sind allerdings für eine befriedigende Arbeit außer diesem Tutorial (= **Band 2**) erforderlich:

a) Ein PC oder Laptop mit der neuesten Version von **KEIL µvision3** und **ATMEL FLIP**

b) Für viele Laptops: ein **USB-RS232-Konverter** (Preis: ca. 10 Euro bei Pollin oder Reichelt Elektronik)

c) Das **EST-ATM1-Board** mit einer passenden **Stromversorgung** (U_{DC} min +8V, am besten ca. **+9V**, aber nicht mehr als +12V vorsehen. Bitte bei Verwendung des zusätzlichen AC-Eingangs am Board nachmessen!

d) Ein **RS232-Null-Modem-Kab**el („Crossed Link“) zur Verbindung mit dem PC.

e) Ein **Quarz mit 11,059 MHz** für Schnittstellenprogramme zusätzlich **zur Normalbestückung mit12 MHz**.

f) Die **Zusatzplatinen samt LCD-Display-Platine**.

g) Ein griffbereiter Ausdruck des **ATM1-Einsteiger-Tutorials (= Band 1a für USB-Betrieb oder Band 1b für RS232-Betrieb)**, denn am Anfang muss man doch noch manchmal nachschlagen, wie das so alles geht.

h) Dann sollte man (ggf. in Form der ATMEL-Board-CD) notfalls bei Fragen oder Problemen mal im **Datenblatt des Controllers AT89C51AC3** nachsehen können. Dieses Datenblatt findet sich darauf als pdf-File.

==========

Beim Board stehen dem Anwender folgende Ports zur Verfügung:

a) **Port 1** für beliebige Ein- und Ausgabezwecke (einschließlich LCD-Display-Steuerung). Breite 8 Bit = 1 Byte

b) **Port 2** dient normalerweise als Adressbus und kann hier -- da das interne Flash-Eprom verwendet wird -- ebenfalls als 8 Bit – Eingabe- und Ausgabeport (einschließlich LCD-Display-Steuerung) verwendet werden.

c) **Vorsicht**: **Port 0** ist der gemultiplexte Daten- / Adressbus und kann deshalb als **zusätzlicher Ausgabeport** manchmal Schwierigkeiten machen **= also stets genau austesten!**
Ist jedoch das zusätzlich nötige externe R-Array (= 8 x 47 Kilo-Ohm) in die Fassung gesteckt oder in die Platine eingelötet , dann kann er **ohne Probleme als Eingabeport** verwendet werden (Ohne die Pullup-Widerstände würden die Pins beim Umschalten auf „Lesen“ sonst frei floaten). **Allerdings müssen nun vor dem Einlesen die verwendeten Portpins (wie auch bei den anderen Ports nötig…) extra auf „HIGH“ gesetzt werden.**

d) Bitte nie vergessen, vor dem Einlesen eines Ports **alle seine Pins durch Beschreiben mit 0xFF auf „HIGH“ zu setzen** und so auf „Lesen“ umzuschalten!

e) Bei **Port 3** sind die **Pins P3.0 und P3.1** durch die **Serielle Schnittstelle** belegt. Ebenso liegen auf den **Pins P3.6 und P3.7** das „**Write“ und „Read**“-Signal des Steuerbusses. Zur Verwendung der restlichen Pins: Siehe Datenbuch.

f) Bei **Port P4** dürfen **nur die beiden Leitungen P4.0 und P4.1** für Ein- und Ausgabezwecke verwendet werden.

Grundsätzlich gilt:
Ein Portpin kann nur dann zum Einlesen verwendet werden, wenn er zuvor auf „HIGH“ gesetzt wurde. Das gilt natürlich auch für einen kompletten Port, der folglich vorher mit „0xFF“ beschrieben werden muss!

Bitte alle übrigen Informationen zu den Ports und ihre Pin-Belegungen aus dem Datenblatt des Controllers entnehmen.

==========

Achtung:
Der von der Firma KEIL mitgelieferte Header „t89c51ac2.h“ war für das Vorgängermodell des verwendeten Controllers vorgesehen und ist deshalb etwas unvollständig für den hier eingesetzten AT89C51AC3.
Im Anhang dieses Manuskripts und auf der neuen ATMEL-Board – CD findet sich deshalb ein selbst geschriebener Header „AT89C51AC3.h“, der nun ALLE vom Chip her möglichen Deklarationen enthält und bei Bedarf verwendet werden kann.
Soweit nichts anderes angegeben, sind aber die Beispiele mit dem „alten“ (und von KEIL mitgelieferten) Header realisiert.

==========

Das Controllerboard wird normalerweise mit einem **Quarz** bestückt, der eine

Taktfrequenz von 12 MHz

ergibt**. Damit arbeiten alle Programme mit einem „minimalen Befehlstakt“ von 1 MHz und die Timer zählen im Rhythmus von exakt einer Mikrosekunde.**
Dafür sind auch alle Timerprogramme geschrieben.

Warnung:
Beim Flashen der Programme über die Serielle Schnittstelle mit dem Programm „FLIP“ darf beim 12MHz-Quarztakt höchstens mit einer Übertragungsrate von 9600 Baud gearbeitet werden -- sonst gibt es „Time Out Error“

Sollen jedoch korrekt funktionierende Programme erstellt werden, die die Serielle Schnittstelle verwenden, dann muss der Quarz gegen ein Exemplar mit

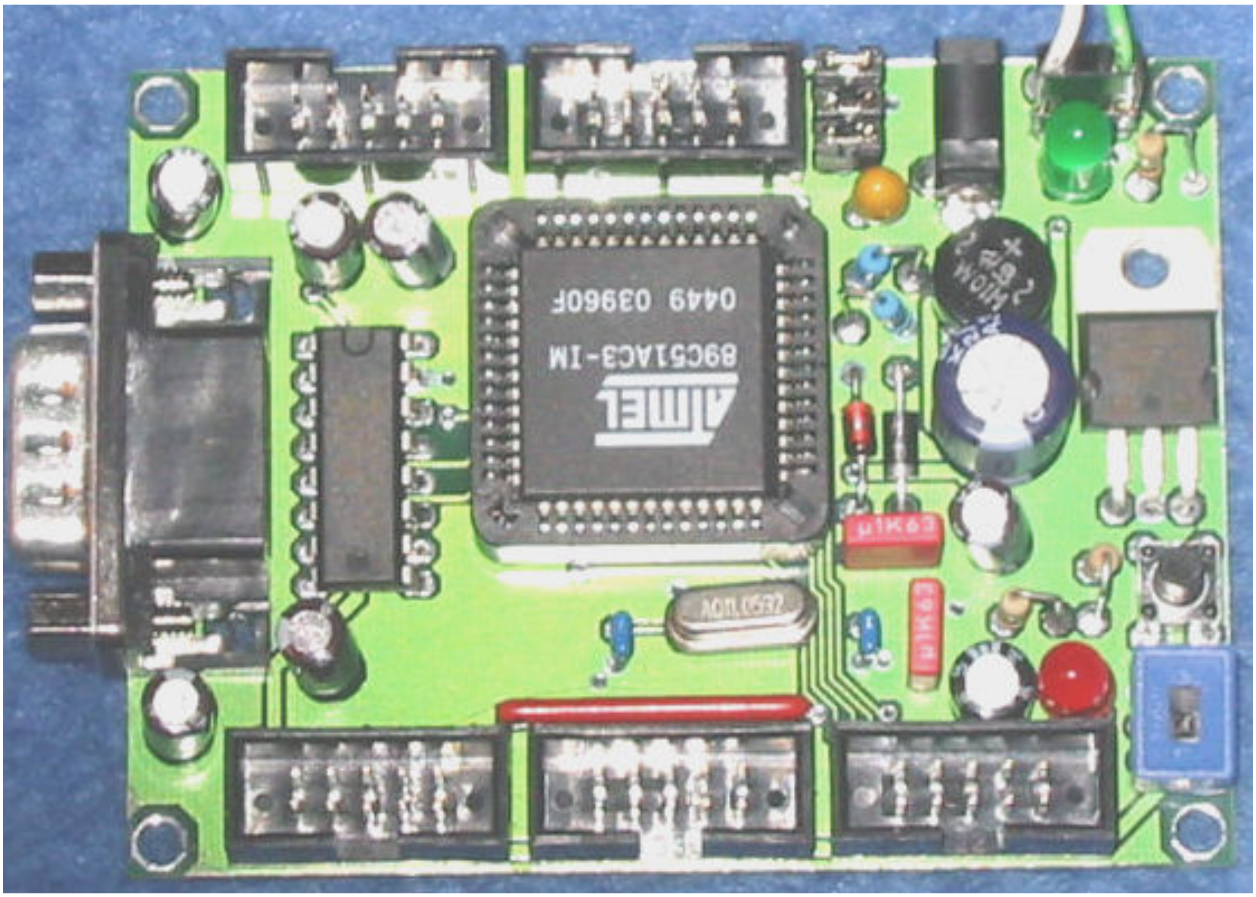
11,0592 MHz

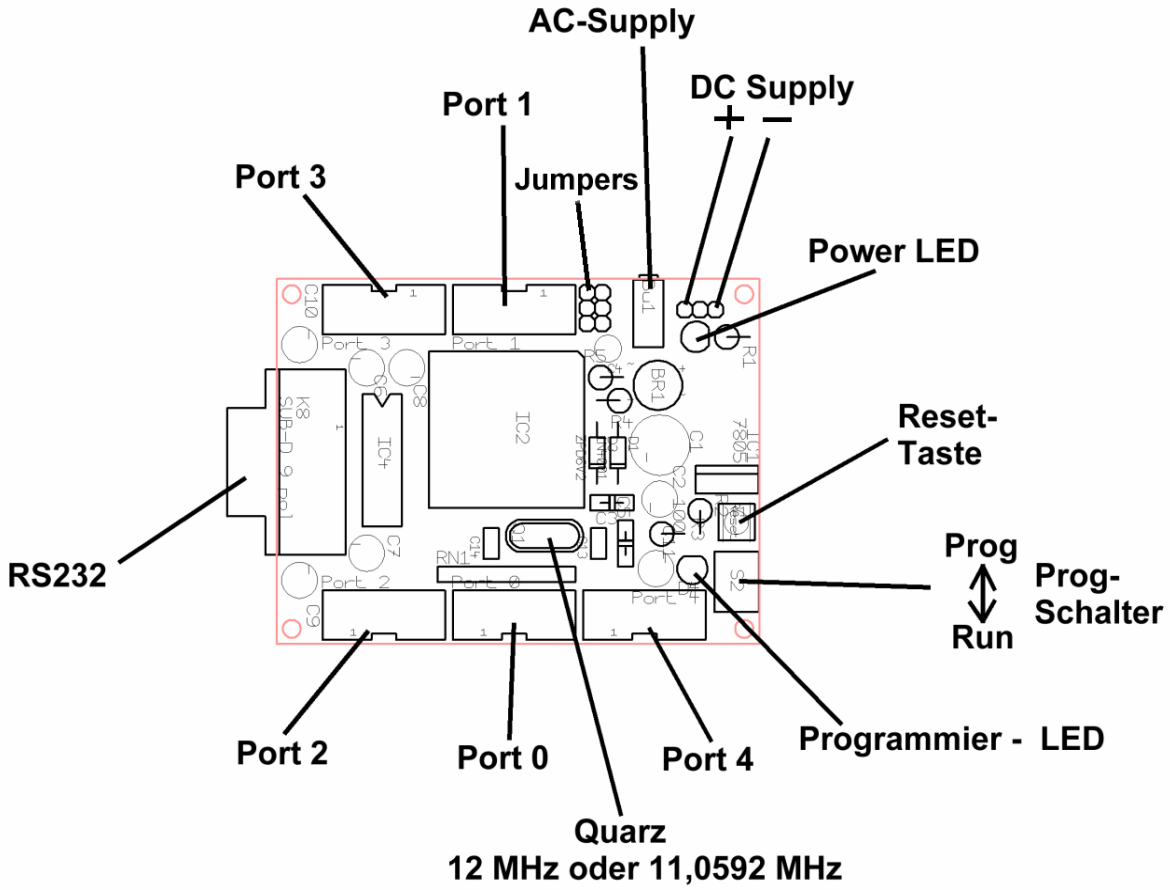
getauscht werden (…deshalb wurde eine Steckfassung zum leichten Wechsel vorgesehen). Nur damit werden die Toleranzgrenzen für die üblichen Baudraten (2400 / 4800 / 9600 Baud usw.) eingehalten und das Board arbeitet korrekt mit anderen RS232-Baugruppen zusammen! Allerdings werden dadurch die von den Timern produzierten Zeiten nun um ca. 9% länger….

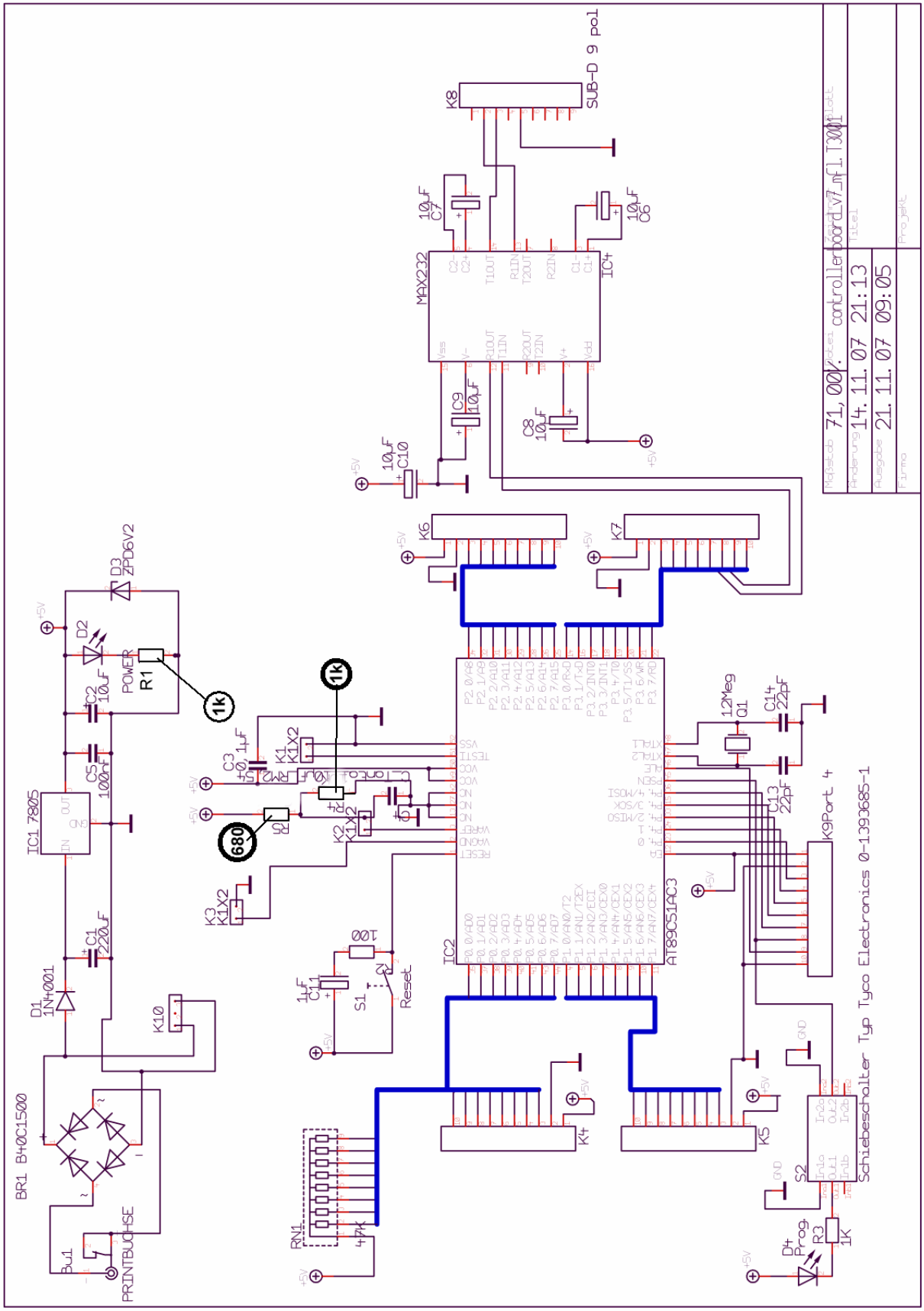
Außerdem ergibt sich ein weiterer Vorteil durch diese Umstellung:

Wegen des korrekten Timings können nun die erstellten Hex-Files ohne Probleme und Übertragungsfehler sogar mit maximal 115 200 Baud mittels FLIP in das Board geflasht werden! (bei 12 MHz Quarzfrequenz sind es maximal nur 9600 Baud)

==========
Auf dem nächsten Blatt folgen ein Foto des Boards sowie ein Lageplan mit den wichtigsten Bauteilen und Baugruppen. Anschließend kann der Stromlaufplan studiert werden.


0449 03960F
89C51AC3-IM
ATMEL


AC-Supply
DC Supply
+
–
Port 1
Port 3
Jumpers
Power LED
Reset-
Taste
RS232
Prog
Run
Prog-
Schalter
Port 2
Port 0
Port 4
Programmier - LED
Quarz
12 MHz oder 11,0592 MHz


SUB-D 9 pol
MAX232
AT89C51AC3
IC1 7805
BR1 B40C1500
PRINTBUCHSE
Schiebeschalter Typ Tyco Electronics 0-1393685-1
controllerboardv7.mfl
14. 11. 07 21:13
21. 11. 07 09:05
71, 001

2. Einstieg mit Port-Spielereien

2.1. Tastentest (mit Endlosschleife)

Wir wollen mit einer ganz einfachen Sache beginnen:

„Schließen Sie an Port P0 die Drucktasten über ein Flachbandkabel an. Verbinden Sie dann den Ausgang P1 mit der LED-Anzeigekette. Schreiben Sie

a) ein Programm, das den Zustand von Port P0 in Port P1 kopiert und dort in Form von aufleuchtenden LEDs sichtbar macht (= gedrückte Taste schaltet die betreffende LED aus) und

b) ein weiteres Programm, das die Zustände an Port P0 invertiert anzeigt (…so dass eine gedrückte Taste durch das Aufleuchten einer LED zu erkennen ist).

Lösung für a)

In einer „**while(1)"-Endlos-Schleife** wird dauernd der Zustand von Port P0 in den Port P1 kopiert.

```
#include <t89c51ac2.h>       // Header für AT89C51AC3
#include <stdio.h>           // Standard – Eingabe – Ausgabe - Header

void main(void)
{     P0=0xFF;               // Port P0 auf „Einlesen" schalten
      while(1)               // Endlosschleife
      {  P1=P0;              // Port 0 wird in Port 1 kopiert
      }
}
```

Lösung für b)

Der Zustand von Port P0 wird vor der Zuweisung mit Hilfe der „Tilde" (~) **komplementiert.**

```
#include <t89c51ac2.h>       // Header für AT89C51AC3
#include <stdio.h>           // Standard – Eingabe – Ausgabe - Header

void main(void)
{     P0=0xFF;               // Port P0 auf „Einlesen" schalten
      while(1)               // Endlosschleife
      {  P1=~P0;             // Port 0 (invertiert) wird Port 1 zugewiesen
      }
}
```

2.2. Lauflicht - Links

Aufgabe:

„Schließen Sie Port P0 an die LED-Kette an. Schreiben Sie dann ein Programm, bei dem eine „1“ schrittweise von Rechts nach Links geschoben wird und sorgen Sie bei jedem neuen Schiebetakt für eine Zeitverzögerung von ca. 0,2 Sekunden.“

Lösung:
Wir setzen den Port auf „0x01“ und warten 0,2 Sekunden. Dann wird der augenblickliche Wert von P0 mit 2 multipliziert und dadurch der Schiebetakt verwirklicht. Anschließend wird wieder 0,2 Sekunden gewartet. Diese Prozedur (Multiplizieren + Warten) wiederholen wir in einer bedingten Schleife, bis die „1“ an der höchsten Stelle angekommen ist. Dann beginnt der Vorgang neu (= Endlosschleife).

```
/*-------------------------------------------------------------
Name:           Lauflicht_links.c
Funktion:       Am Port P1 wird eine "1" schrittweise nach links geschoben
Datum:          24. 10. 2007
Autor:          G. Kraus

-------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------*/
#include <t89c51ac2.h>  // Header für Controller "AT89C51AC3"
#include <stdio.h>      // Standard-Eingabe-Ausgabe-Header

void zeit(void);        // Prototypen-Anmeldung

/*-------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------*/

void main(void)
{     unsigned char x;          // Zählvariable
      while(1)                  // Endlos-Schleife
      {     P1=0x01;            // Port P1 zeigt “Eins”
            zeit();
            for(x=0;x<=6;x++)   // Leuchtende “1” nach links schieben
            {     P1=P1*2;
                  zeit();
            }
      }
}
/*-------------------------------------------------------------
Zusatzfunktionen
-------------------------------------------------------------*/
void zeit(void)
{
      unsigned int x;
      for(x=0;x<=30000;x++);    // Int-Wert 30000 ergibt ca. 0,2 Sekunden Delay
}
```

2.3. Lauflicht - Rechts

Aufgabe:

„Schließen Sie Port P0 an die LED-Kette an. Schreiben Sie dann ein Programm, bei dem eine „1“ schrittweise von Links nach Rechts geschoben wird und sorgen Sie bei jedem neuen Schiebetakt für eine Zeitverzögerung von ca. 0,2 Sekunden.“

Lösung:
Wir setzen den Port auf „0x80“ und warten 0,2 Sekunden. Dann wird der augenblickliche Wert von P0 durch 2 dividiert und dadurch der Schiebetakt verwirklicht. Anschließend wird wieder 0,2 Sekunden gewartet. Diese Prozedur (Dividieren+ Warten) wiederholen wir in einer bedingten Schleife, bis die „1“ an der tiefsten Stelle angekommen ist. Dann beginnt der Vorgang neu (= Endlosschleife).

```
/*------------------------------------------------------------------
Name:               Lauflicht_rechts.c
Funktion:           Am Port P1 wird eine "1" schrittweise nach
                    rechts geschoben

Datum:              24. 10. 2007
Autor:              G. Kraus

------------------------------------------------------------------
Deklarationen und Konstanten
------------------------------------------------------------------*/

#include <t89c51ac2.h>     // Header für Controller "AT89C51AC3"
#include <stdio.h>         // Standard-Eingabe-Ausgabe-Header

/*------------------------------------------------------------------
Hauptprogramm
------------------------------------------------------------------*/
void zeit(void);           // Prototypen-Anmeldung

void main(void)
{     unsigned char x;
      while(1)
      {     P1=0x80;
            zeit();
            for(x=0;x<=6;x++)
            {     P1=P1/2;        // Nach Rechts schieben
                  zeit();
            }
      }
}
/*------------------------------------------------------------------
Zusatzfunktionen
------------------------------------------------------------------*/
void zeit(void)
{
      unsigned int x;
      for(x=0;x<=30000;x++);     // Int-Wert 30000 ergibt 0,2 Sekunden Delay
}
```

2.4. Lauflicht – Links / Rechts

Aufgabe:

„Schließen Sie Port P0 an die LED-Kette an. Schreiben Sie dann ein Programm, bei dem eine „1“ erst schrittweise von Links nach Rechts bis zum „linken Anschlag“ und dann wieder schrittweise bis zum „rechten Anschlag“ zurückgeschoben wird. Sorgen Sie bei jedem neuen Schiebetakt für eine Zeitverzögerung von ca. 0,2 Sekunden.“

Lösung:
Wir setzen den Port auf „0x01“ und warten 0,2 Sekunden. Dann wird der augenblickliche Wert von P0 mit 2 multipliziert und so der Schiebetakt verwirklicht. Anschließend wird wieder 0,2 Sekunden gewartet. Diese Prozedur (Multiplizieren + Warten) wiederholen wir in einer bedingten Schleife, bis die „1“ an der höchsten Stelle angekommen ist. Dann wird die bisherige Multiplikation mit 2 durch eine „Division durch 2“ ersetzt usw.

```
/*------------------------------------------------------------------
Name:               Lauflicht_links_rechts.c
Funktion:           Am Port P1 wird eine "1" schrittweise nach links und dann wieder
                    nach rechts geschoben

Datum:              24. 10. 2007
Autor:              G. Kraus

------------------------------------------------------------------
Deklarationen und Konstanten
------------------------------------------------------------------*/
#include <t89c51ac2.h>  // Header für Controller "AT89C51AC3"
#include <stdio.h>      // Standard-Eingabe-Ausgabe-Header

void zeit(void);        // Prototypen-Anmeldung

/*------------------------------------------------------------------
Hauptprogramm
------------------------------------------------------------------*/

void main(void)
{     unsigned char x;
      P1=0x01;                  // Unterste leuchtet
      zeit();
      while(1)                  // Endlos-Schleife
      {     for(x=0;x<=6;x++)   // Schiebe nach links
            {     P1=P1*2;
                  zeit();
            }

            for(x=0;x<=6;x++)   // Schiebe nach rechts
            {     P1=P1/2;
                  zeit();
            }
      }
}
/*------------------------------------------------------------------
Zusatzfunktionen
------------------------------------------------------------------*/
void zeit(void)
{
      unsigned int x;
      for(x=0;x<=30000;x++);    // Int-Wert 30000 ergibt 0,2 Sekunden Delay
}
```

3. Umgang mit ERAM und XRAM

3. 1. Erläuterung

Unser ATMEL-Controller AT89C51AC3 besitzt ein „**Expanded RAM = ERAM“ mit 2048 Byte auf dem Chip**. Es dient als On-Chip-Ersatz für einen zusätzlich möglichen externen RAM-Speicherbaustein (XRAM). Der könnte theoretisch eine Kapazität von 64 Kilobyte aufweisen, benötigt aber zusätzlichen Platz auf der Leiterplatte und die Zugriffe sind langsamer.

Deshalb haben wir beim ATM1-Board auf diesen externen XRAM-Baustein verzichtet und begnügen uns mit den 2048 Byte auf dem Chip. Sie werden -- wie ein echtes XRAM -- mit „MOVX“-Befehlen angesprochen.

Dazu ist allerdings am **Anfang jedes Hauptprogrammes ein Umschaltvorgang** nötig. Er lautet:

Lösche das EXTRAM-Bit (= AUXR.1) im AUXR-Register.

(Beim Einschalten des Controllers ist es normalerweise nämlich gesetzt und damit „echter externer XRAM-Betrieb“ gewählt…). Da das AUXR-Register nicht bit-adressierbar ist, muss man das über eine

logische UND-Verknüpfung des AUXR-Registers mit der Binärzahl 11111101 = 0xFD

erledigen. Sie lautet:

```
AUXR&0xFD;        // EXTRAM-Bit löschen = ERAM verwenden
```

3.2. ERAM-Demonstration: Balkenanzeige

```
/*-------------------------------------------------------------
Programmbeschreibung
-------------------------------------------------------------
Name:       xram_01.c
Funktion:   Im XRAM werden einige Werte in einem Array gespeichert und
            anschließend nach dem Auslesen an einer LED-Kette sichtbar
            gemacht.
            So wird der Umgang mit dem XRAM demonstriert.

Datum:      07. 11. 2007
Autor:      G. Kraus

-------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------*/

#include <t89c51ac2.h>
#include <stdio.h>

xdata unsigned char a,b,c,d,e,f,g,h,i;

/*-------------------------------------------------------------
Prototypen
-------------------------------------------------------------*/
void wait(void);
```

```
/*---------------------------------------------------------------------------
Hauptprogramm
---------------------------------------------------------------------------*/

void main(void)
{    P1=0;
     AUXR=AUXR&0xFD;       // auf internes ERAM umschalten = EXTRAM löschen
     a=0;
     b=1;
     c=3;
     d=7;
     e=15;
     f=31;
     g=63;
     h=127;
     i=255;

     while(1)
     {
          P1=a;
          wait();
          P1=b;
          wait();
          P1=c;
          wait();
          P1=d;
          wait();
          P1=e;
          wait();
          P1=f;
          wait();
          P1=g;
          wait();
          P1=h;
          wait();
          P1=i;
          wait();
     }
}


void wait(void)
{
     unsigned int x;
     for(x=0;x<=20000;x++);    Delay
}
```

4. Verschiedene Piepsereien

4.1. Einfacher 1kHz-Test-Ton (Endlos-Schleife)

Dieses Beispiel wird auch im Einführungs- und Programmier-Tutorial benützt. Es lautet:

„Erzeugen Sie am Portpin P1.0 ein Rechtecksignal mit der Frequenz f = 1kHz. Prüfen Sie die erzeugte Frequenz mit einem Frequenzzähler nach und gleichen Sie auf den exakten Wert ab."

Dahinter steckt ein ganz einfaches Prinzip:

Ein Portpin wird „getoggelt", also sein Zustand in regelmäßigen Abständen umgekehrt. Die zwischen zwei Umkehrungen eingefügte Wartezeit legt die Tonhöhe fest. Dieser Vorgang wird in eine Endlosschleife eingebunden und so entsteht ein Rechtecksignal am Portpin P1.0.

```
/*-------------------------------------------------------------
Name:           1khz_c.c
Funktion:       Am Portpin P1.0 wird ein Rechtecksignal mit der
                Frequenz 1 kHz ausgegeben
Datum:          24. 10. 2007
Autor:          G. Kraus
--------------------------------------------------------------
Deklarationen und Konstanten
--------------------------------------------------------------*/
#include <t89c51ac2.h>     // Header für Controller "AT89C51AC3"
#include <stdio.h>         // Standard-Input-Output-Header
sbit ausgang=P1^0;         // Ausgabe-Pin P1.0 heißt nun „ausgang"

void zeit(void);           // Prototypen-Anmeldung
/*--------------------------------------------------------------
Hauptprogramm
--------------------------------------------------------------*/
void main(void)
{     AUXR=AUXR&0xFD;       // auf internes ERAM umschalten = EXTRAM löschen
      while(1)                 // Endlos-Schleife
      {
            zeit();            // Delay von 500 Mikrosekunden
            ausgang=~ausgang;  // Bit komplementieren
      }
}
/*--------------------------------------------------------------
Zusatzfunktionen
--------------------------------------------------------------*/
void zeit(void)
{
      unsigned int x;          // Zählvariable deklarieren
      for(x=0;x<=62;x++);      // Int-Wert 62 ergibt ca. 500 Mikrosekunden Delay
}
```

4.2. Klingelknopf für 1kHz-Ton (Bedingte while-Schleife)

Aufgabe:
„Schließen Sie den Port P0 an die Drucktastenplatine an und sorgen Sie dafür, dass beim Drücken der Taste an P0.0 (= LOW-aktiv!) ein 1KHz-Rechtecksignal an Portpin P1.0 erzeugt wird. Prüfen Sie die korrekte Tonhöhe mit einem Frequenzzähler nach.“

Dazu wird eine „while“-Schleife verwendet. Sie wird ausgeführt, solange der Klingelknopf gedrückt wird.

```
/*-------------------------------------------------------------
Name:              1khz_klingel.c
Funktion:          Am Portpin P1.0 wird ein Rechtecksignal mit der
                   Frequenz 1 kHz ausgegeben, wenn die Taste an P0.0
                   gedrückt wird

Datum:             24. 10. 2007
Autor:             G. Kraus

-------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------*/
#include <t89c51ac2.h>     // Header für Controller "AT89C51AC3"
#include <stdio.h>         // Standard-Input-Output-Header

sbit ausgang=P1^0;         // Ausgabe-Pin P1.0 heißt nun „ausgang“
sbit klingel_1khz=P0^0;    // Portpin P0.0 als Klingelknopf

void zeit(void);           // Prototypen-Anmeldung

/*-------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------*/
void main(void)
{  AUXR=AUXR&0xFD;         // auf internes ERAM umschalten = EXTRAM löschen
   P0==0xFF;                        // P0 auf „Einlesen“ schalten
   while(1)
   {     while(klingel_1khz==0)     // Klingelknopf gedrückt?
         {     zeit();              // Delay von 500 Mikrosekunden
               ausgang=~ausgang;    // Bit komplementieren
         }
   }
}
/*-------------------------------------------------------------
Zusatzfunktionen
-------------------------------------------------------------*/
void zeit(void)
{
     unsigned int x;
     for(x=0;x<=62;x++);           // Int-Wert 62 ergibt 500 Mikrosekunden Delay
}
```

4.3. Mehrere Klingelknopf -Töne (Funktion mit Parameter-Übergabe)

Aufgabe:

„Schließen Sie den Port P0 an die Drucktastenplatine an. Sorgen Sie dafür, dass beim Drücken der Tasten P0.0 / P0.1 / P0.2 (= LOW-aktiv!) ein Rechtecksignal mit den Frequenzen 250 / 500 / 1000Hz an Portpin P1.0 erzeugt wird“.

Wir schreiben dazu nur eine einzige Funktion „Piepsen“, die beim Drücken einer Taste aufgerufen wird. Die unterschiedlichen Tonhöhen werden in Form einer Int-Variable mit dem passenden Wert für diese Frequenz an die aufgerufene Funktion übergeben.

```
/*-------------------------------------------------------------
Name:           klingeltoene.c
Funktion:       Am Portpin P1.0 wird ein Rechtecksignal mit
                verschiedenen Frequenzen ausgegeben.
                Dabei gilt folgende Zuordnung:
                P0.0 gedrückt ergibt 100Hz
                P0.1 gedrückt ergibt 250Hz
                P0.2 gedrückt ergibt 500Hz
                P0.2 gedrückt ergibt 1000Hz
Datum:          24. 10. 2007
Autor:          G. Kraus
---------------------------------------------------------------
Deklarationen und Konstanten
--------------------------------------------------------------*/
#include <t89c51ac2.h>       // Header für Controller "AT89C51AC3"
#include <stdio.h>           // Standard-Input-Output-Header

void piepsen (unsigned int wait);       // Prototypen-Anmeldung
sbit ausgang=P1^0;                      // Ton-Ausgang
sbit klingel_100Hz=P0^0;                // Taste für 100Hz
sbit klingel_250Hz=P0^1;                // Taste für 250Hz
sbit klingel_500Hz=P0^2;                // Taste für 500Hz
sbit klingel_1khz=P0^3;                 // Taste für 1000Hz
/*-------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------*/
void main(void)
{  AUXR=AUXR&0xFD;       // auf internes ERAM umschalten = EXTRAM löschen
   P0==0xFF;                    // P0 auf „Einlesen“ schalten
   while(1)
   {  while(klingel_100Hz==0)   // Taste „100Hz“ gedrückt?
      {     piepsen(480);
      }
      while(klingel_250Hz==0)   // Taste „350Hz“ gedrückt?
      {     piepsen(240);
      }
      while(klingel_500Hz==0)   // Taste „500Hz“ gedrückt?
      {     piepsen(120);
      }
      while(klingel_1khz==0)    // Taste „1000Hz“ gedrückt?
      {     piepsen(62);
      }
   }
}
/*-------------------------------------------------------------
Zusatzfunktionen
-------------------------------------------------------------*/
void piepsen (unsigned int wait)
{     unsigned int x;
      for(x=0;x<=wait;x++);     // Wartezeit erzeugen
      ausgang=~ausgang;
}
```

4.4. Tongenerator für verschiedene Frequenzen mit AUS-Taste (Merker-Prinzip)

Aufgabe:

„Schließen Sie den Port P0 an die Drucktastenplatine an. Sorgen Sie dafür, dass schon bei einem kurzen Druck auf eine der Tasten P0.1 / P0.2 / P0.3 / P0.4 (= LOW-aktiv!) Rechtecksignale mit den Frequenzen 100 / 250 / 500 / 1000Hz an Portpin P1.0 erzeugt werden“.

Wir schreiben dazu nur eine einzige Funktion „Piepsen“, die beim Drücken einer Taste aufgerufen wird. Die unterschiedlichen Tonhöhen werden in Form einer Int-Variable mit dem passenden Wert für diese Frequenz an die aufgerufene Funktion übergeben. Eine eigene Funktion „Tastentest“ prüft dauernd, welche Taste gedrückt wird und speichert diesen Wunsch in einem zugehörigen Bit (= „Merker“). Solange dieses Bit auf „1“ steht, wird der gewünschte Ton erzeugt -- oder der Generator ausgeschaltet.

```
// generator_01.c
// Erstellt am 22.10.2007 durch G. Kraus

/* Es stehen an Port P0 fünf Drucktasten zur Verfügung.
Taste P0.0 bildet die AUS-Taste, während mit den nächsten
Tasten Rechtecksignale mit den Frequenzen 100 / 250 / 500 / 1000 Hz
gewählt werden können. Ein einmaliger kurzer Tastendruck reicht,
um eine dieser vier Frequenzen zu erzeugen.
Mit AUS kann man jederzeit die Tonerzeugung ausschalten*/

#include<t89c51ac2.h>              // Header für AT89C51AC3
#include<stdio.h>                  // Standard-Eingabe-Ausgabe-Header

sbit ausgang=P1^0;                 // Portpin P1.0 ist der Ausgang

sbit Taste_AUS=P0^0;               // Tastendeklaration
sbit Taste_Rechteck_100Hz=P0^1;
sbit Taste_Rechteck_250Hz=P0^2;
sbit Taste_Rechteck_500Hz=P0^3;
sbit Taste_Rechteck_1000Hz=P0^4;

bit AUS;                           // Merker-Deklaration
bit Rechteck_100Hz;
bit Rechteck_250Hz;
bit Rechteck_500Hz;
bit Rechteck_1000Hz;

void Rechteck(unsigned int wait);    // Prototypen
void Tastenabfrage(void);

void main(void)
{    AUXR=AUXR&0xFD;  // auf internes ERAM umschalten = EXTRAM löschen
     AUS=1;           // Generator ausschalten
     P0=0xFF;         // P0 auf „Einlesen“ schalten
     while(1)
     {
          while(AUS==1)                // Generator ist ausgeschaltet
          {    Tastenabfrage();
          }

          while(Rechteck_100Hz==1)     // Rechteck 100 Hz wird erzeugt
          {    Rechteck(480);
               Tastenabfrage();
          }

          while(Rechteck_250Hz==1)     // Rechteck 250 Hz wird erzeugt
          {    Rechteck(240);
               Tastenabfrage();
          }

          while(Rechteck_500Hz==1)     // Rechteck 500 Hz wird erzeugt
          {    Rechteck(120);
               Tastenabfrage();
          }
```

```
          while(Rechteck_1000Hz==1)    // Rechteck 500 Hz wird erzeugt
          {     Rechteck(62);
                Tastenabfrage();
          }
     }

}


void Rechteck(unsigned int wait)        // Wartezeit-Funktion
{    int x;
     for(x=0;x<=wait;x++);
     ausgang=~ausgang;
}


void Tastenabfrage(void)                // Tastenabfrage
{    if(Taste_AUS==0)                     // Aus-Taste gedrückt?
     {    AUS=1;
          Rechteck_100Hz=0;
          Rechteck_250Hz=0;
          Rechteck_500Hz=0;
          Rechteck_1000Hz=0;
     }
     if(Taste_Rechteck_100Hz==0)        // 100Hz-Taste gedrückt?
     {    AUS=0;
          Rechteck_100Hz=1;
          Rechteck_250Hz=0;
          Rechteck_500Hz=0;
          Rechteck_1000Hz=0;
     }
     if(Taste_Rechteck_250Hz==0)        // 250-Taste gedrückt?
     {    AUS=0;
          Rechteck_100Hz=0;
          Rechteck_250Hz=1;
          Rechteck_500Hz=0;
          Rechteck_1000Hz=0;
     }
     if(Taste_Rechteck_500Hz==0)        // 500Hz-Taste gedrückt?
     {    AUS=0;
          Rechteck_100Hz=0;
          Rechteck_250Hz=0;
          Rechteck_500Hz=1;
          Rechteck_1000Hz=0;
     }
     if(Taste_Rechteck_1000Hz==0)       // 1000Hz-Taste gedrückt?
     {    AUS=0;
          Rechteck_100Hz=0;
          Rechteck_250Hz=0;
          Rechteck_500Hz=0;
          Rechteck_1000Hz=1;
     }
}
```

4.5. Martinshorn der Polizei (mit Parameter-Übergabe und Merker-Steuerung)

Aufgabe:

An Port P0 sind zwei Drucktasten angeschlossen. Die Taste P0.1 löst ein "Martinshorn" an Portpin P1.0 aus, die Taste P0.0 schaltet es wieder aus. Die beiden Töne des Martinshornes sind g' = 392Hz und c" = 523 Hz. Der Martinshorn-Ton soll mit einer einzigen Funktion (und Parameterübergabe) erzeugt werden.

```
// martinshorn.c, erstellt am 22.10.2007 durch G. Kraus

#include<t89c51ac2.h>      // Header für AT89C51AC3
#include<stdio.h>          // Standard-Eingabe-Ausgabe-Header

sbit ausgang=P1^0;         // Portpin P1.0 ist der Ausgang

sbit Taste_AUS=P0^0;       // Tastendeklaration
sbit Taste_Horn=P0^1;

bit AUS;                   // Merker-Deklaration
bit Martinshorn;

void M_Horn(unsigned int wait1,wait2,dauer1,dauer2);

void Tastenabfrage(void);

void main(void)
{     AUXR=AUXR&0xFD;  // auf internes ERAM umschalten = EXTRAM löschen
      AUS=1;                              // Horn ausschalten
      P0=0xFF;                            // P0 auf „Einlesen" schalten
      while(1)
      {
            while(AUS==1)                 // Horn ist ausgeschaltet
            {     Tastenabfrage();
            }

            while(Martinshorn==1)         // Martinshorn tutet
            {     M_Horn(120,80,900,1400  );
                  Tastenabfrage();
            }
      }
}

void M_Horn(unsigned int wait1,wait2,dauer1,dauer2)

{     unsigned int x,y;
      for(x=0;x<=dauer1;x++)
      {     for(y=0;y<=wait1;y++);
            ausgang=~ausgang;
      }

      for(x=0;x<=dauer2;x++)
      {     for(y=0;y<=wait2;y++);
            ausgang=~ausgang;
      }
}

void Tastenabfrage(void)      // Tastenabfrage
{     if(Taste_AUS==0)        // Aus-Taste gedrückt
      {     AUS=1;
            Martinshorn=0;
      }
      if(Taste_Horn==0)     // Martinshorn einschalten
      {     AUS=0;
            Martinshorn=1;
      }
}
```

4.6. Alarmanlage (mit „do-while“-Schleife, Parameter-Übergabe und „break“-Anweisung“)

Aufgabe:

Dieses Programm soll eine Alarmanlage simulieren.
Wird die Taste P0.0 gedrückt, so ertönt ein Alarmsignal, das an Portpin P1.0 ausgegeben wird.
Die Anlage kann nur durch eine Taste an Portpin P0.1 wieder zurückgesetzt werden.
Das Alarmsignal besteht aus ca. 0,2 Sekunden Piepsen mit 1 kHz, gefolgt von einer Pause mit 0,2 Sekunden. Das wird dauernd wiederholt.

Lösung:
Wir fragen die „Alarm“-Taste dauernd mit einer „do – while“ – Schleife ab. Sobald sie ausgelöst wird, geht das Programm in eine Endlos-Schleife und erzeugt den Alarm. Diese Endlos-Schleife kann nur durch Betätigen der Reset-Taste wieder verlassen werden, da hierdurch eine „break“-Anweisung erzeugt wird.

```
// 24.10.2007 / G. Kraus

#include<t89c51ac2.h>        // Header für AT89C51AC3
#include<stdio.h>            // Standard-Input-Output-Header

sbit ausgang=P1^0;           // Ausgang an P1.0
sbit alarm=P0^0;             // Alarm durch P0.0
sbit reset=P0^1;             // Reset durch P0.1

pieps_alarm(unsigned int wait1, dauer1);

pause(unsigned int dauer2);

//------------------------------------------------------
void main(void)
{    AUXR=AUXR&0xFD;  // auf internes ERAM umschalten = EXTRAM löschen
     P0=0xFF;                        // P0 auf „Einlesen“ schalten
     while(1)
     {    do {} while(alarm==1);     // Alarm gedrückt?

          while(1)
          {    pieps_alarm(62,1000);

               if(reset==1)          // reset gedrückt?
               {    pause(50000);    // Wenn Nein: Pause erzeugen
               }
               else {break;}         // Wenn Ja: raus aus Schleife
          }

     }
 }

pieps_alarm(unsigned int wait1,dauer1)
{    unsigned int x,y;

     for(x=0;x<=dauer1;x++)
     {    for(y=0;y<=wait1;y++);
          ausgang=~ausgang;
     }

}

pause(unsigned int dauer2)
{    unsigned int x;
     for(x=0;x<=dauer2;x++);
}
```

4.7. Projekt „Mäuse-Orgel“

4.7.1. Aufgabenstellung

Sie sollen mit Hilfe der beiden Zusatzplatinen „**Drucktasten-Eingabe“ an Port P0** und „**NF-Verstärker mit Lautsprecher“ an Portpin P1^0** eine Mini-Orgel programmieren, bei der nach dem Drücken einer Taste (....8 Stück stehen zur Verfügung.....) der zugehörige Ton aus der C-Dur-Tonleiter erklingt. Das Spielen von Akkorden oder von mehrstimmigen Stücken ist nicht möglich. Schreiben Sie dieses Programm in „C“.

4.7.2. Etwas Musik-Grundlagen

Bei einer Tonleiter geht man immer von einem ***Grundton*** mit einer bestimmten Frequenz aus. In 8 Schritten (= „Intervallen“) erhöht man nun die Tonfrequenzen, bis man bei der ***doppelten Frequenz*** des Grundtones angelangt ist. Auf diese Weise hat man eine „***Oktave***“ durchlaufen. Dann wiederholt man erneut die einzelnen Schritte, bis sich wieder die Frequenz verdoppelt hat usw. Die verschiedenen Oktaven werden durch „Strichbezeichnungen“ oder Nummern (z. B. „C1“)unterschieden.

Innerhalb der C-Dur-Tonleiter haben wir es im deutschsprachigen Raum mit folgenden Einzeltönen zu tun:

C D E F G A H C

Nun ist es wichtig zu wissen, welchen Frequenzabstand die einzelnen Töne voneinander haben. Und hier geht es schon los: eigentlich ist eine Oktave sogar in 12 Einzeltöne (= „**Halbtöne**“) aufgeteilt. Leider klingt das nicht sehr angenehm, wenn man damit Musik macht -- aber es wird gemacht...(Name: „Zwölfton-Musik“)
Am angenehmsten waren und sind für uns Menschen immer noch folgende Abstände in einer Tonleiter:

C	**D**	**E**	**F**	**G**	**A**	**H**	**C**
2 Halbtöne	2 Halbtöne	1 Halbton	2 Halbtöne	2 Halbtöne	2 Halbtöne	1 Halbton	

Also, ganz perfekt stimmt das auch nicht, aber es geht da nur noch um einzelne Abweichungen in der 1... 2 Hz – Größenordnung (die man natürlich beim Stimmen eines Orchesters merkt). Johann Sebastian Bach war der Erste, der eine Angleichung vorschlug und in seiner „**wohltemperierten Stimmung**“ festlegte, dass alle Halbtöne mit demselben Faktor in ihre Brüder zur rechten oder linken Seite umgerechnet werden können.

Dieser Faktor ist $\mathbf{k} = \sqrt[12]{2}$ (Das ergibt die Zahl 1,059463094)

Und die letzte nötige Information ist die exakte Frequenz eines „Referenztons“, von dem aus alle übrigen Ganz- und Halbtonfrequenzen mit diesem Faktor bestimmt werden können. Das ist der

Kammerton „A“ mit f = 440 Hz

Beispiel: der Ton „G“ liegt um zwei Halbtöne tiefer als „A“. Also beträgt seine Frequenz

$$G = \frac{440Hz}{1{,}059463094 \bullet 1{,}059463094} = 392Hz$$

Der Ton „H“ liegt dagegen um zwei Halbtöne höher als „A“. Deshalb gehört dazu die Frequenz

$$H = 440Hz \bullet 1{,}059463094 \bullet 1{,}059463094 = 493{,}88Hz$$

Aufgabe: Bestimmen Sie alle Einzelfrequenzen für die C-Dur-Tonleiter und tragen Sie die Ergebnisse in eine Tabelle ein:

4.7.3. Programmierung des Kammertones „A“

Wir wollen ein C-Programm schreiben, bei dem alle 8 Tasten nacheinander abgefragt werden und bei einer gedrückten Taste der gewünschte Ton erzeugt wird.

Bitte beachten:

a) Die Tastenabfrage erfolgt mit bedingten „while“-Schleifen, die in eine Endlosschleife eingebettet sind.

b) Um auch zu sehr tiefen Tönen zu kommen, muss für die Zeitverzögerung eine „unsigned integer“ – Zahl als Vorgabe benützt werden, denn damit sind Werte zwischen Null und 64535 zulässig.

Außerdem sollte man jetzt schon die einzelnen Tasten (die über Port P0 abgefragt werden!) mit den passenden Tönen bezeichnen.
Das an Portpin P1^0 ausgegebene Tonsignal erhält die Bezeichnung „ausgang“.
Damit erhalten wir folgendes Listing für unser erstes „Test“-Programm:

Nehmen Sie dieses Programm in Betrieb und gleichen Sie es mit einem Frequenzzähler auf die richtige Frequenz des Kammertons “a” ab.

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

sbit ausgang =   P1^0;          // Bit-Deklaration für den Tonausgang

sbit C1    =   P0^0;            // Klaviertasten für Tonleiter von Ton „C1“ bis „C2“
sbit D     =   P0^1;
sbit E     =   P0^2;
sbit F     =   P0^3;
sbit G     =   P0^4;
sbit A     =   P0^5;
sbit H     =   P0^6;
sbit C2    =   P0^7;

unsigned int x;                 // Global deklarierte Zählvariable vom Integer-Typ

void main(void)
{       AUXR=AUXR&0xFD;  // auf internes ERAM umschalten = EXTRAM löschen
        P0=0xFF;                // P0 auf „Einlesen“ schalten
        while(1)                // Endlosschleife
        {
                while(A==0)     // Taste für Ton A” gedrückt?
                {
                        for(x=0;x<=140;x++);    // Zeitverzögerung für eine halbe Periode von 440 Hz
                        ausgang=~ausgang;       // Ausgangsbit umkehren
                }
        }
}
```

4.7.4. Vollständige Orgel

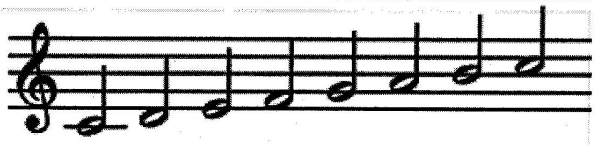
Ergänzen Sie nun die fehlenden 7 Töne (….dazu gibt es auf dem letzten Blatt einen interessante Information aus dem Internet…) , bringen Sie die Frequenzen auf die richtigen Werte aus Ihrer Tabelle und spielen Sie anschließend ein kleines Lied.

Hier ist das fertige Programm-Listing. Damit stimmen die erzeugten Frequenzen mit einer maximalen Abweichung von 1,5 Hz.....

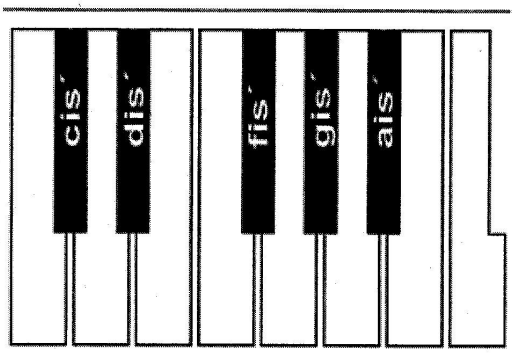
```
#include <t89c51ac2.h>     // Header für AT89C51AC3
#include <stdio.h>         // Standard - Input - Output - Header

sbit ausgang=P1^0;         // Port P1 als Ton-Ausgang
sbit C1=P0^0;              // Tastendeklarationen für P0
sbit D=P0^1;
sbit E=P0^2;
sbit F=P0^3;
sbit G=P0^4;
sbit A=P0^5;
sbit H=P0^6;
sbit C2=P0^7;

unsigned int x;               // Variable für Zeitschleifen

void main(void)
{     AUXR=AUXR&0xFD;     // auf internes ERAM umschalten = EXTRAM löschen
      P0=0xFF;            // Port P0 auf "Einlesen" schalten
      while(1)
      {     while(C1==0)
            {     for(x=0;x<=236;x++);
                  ausgang=~ausgang;
            }

            while(D==0)
            {     for(x=0;x<=210;x++);
                  ausgang=~ausgang;
            }

            while(E==0)
            {     for(x=0;x<=187;x++);
                  ausgang=~ausgang;
            }

            while(F==0)
            {     for(x=0;x<=177;x++);
                  ausgang=~ausgang;
            }

            while(G==0)
            {     for(x=0;x<=157;x++);
                  ausgang=~ausgang;
            }

            while(A==0)
            {     for(x=0;x<=140;x++);
                  ausgang=~ausgang;
            }

            while(H==0)
            {     for(x=0;x<=125;x++);
                  ausgang=~ausgang;
            }

            while(C2==0)
            {     for(x=0;x<=118;x++);
                  ausgang=~ausgang;
            }
      }
}
```

Tonhöhe und Frequenz:

Die C-Dur-Tonleiter



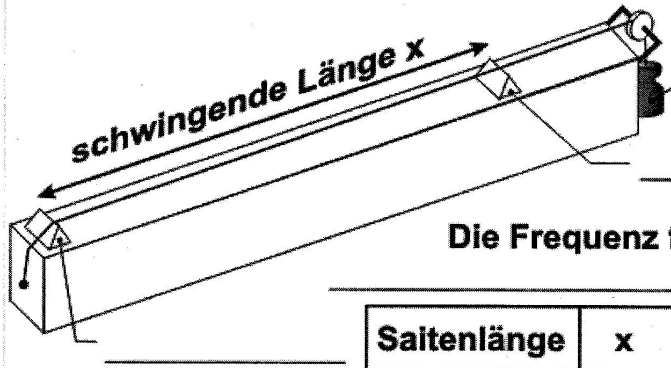
umfasst die__ Töne (_____)
von einem C-Ton zum nächsten C-Ton (hier von__ zu__).
Auf der Klaviatur benötigt man dafür _____
Töne, die eine Oktave höher liegen haben immer die_____ Frequenz.
Beispiel: c´ _____
c´´ _____



261,6Hz 329,6Hz 392,0Hz 493,9Hz
293,7Hz 349,2Hz 440,0Hz 523,2Hz

Der Kammerton a´ mit 440Hz ist der Ton, _____

Monocord: (Instrument mit nur einer Saite)

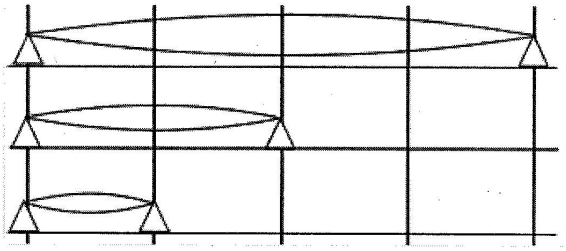


_____
_____
_____

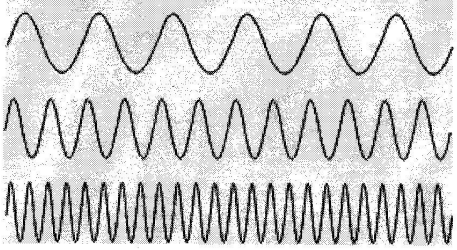
Die Frequenz f hängt von _____ ab.

Saitenlänge	x	$\frac{x}{2}$	$\frac{x}{4}$
Frequenz	f		

Saitenschwingung:



Schwingunsbild im Oszilloskop:

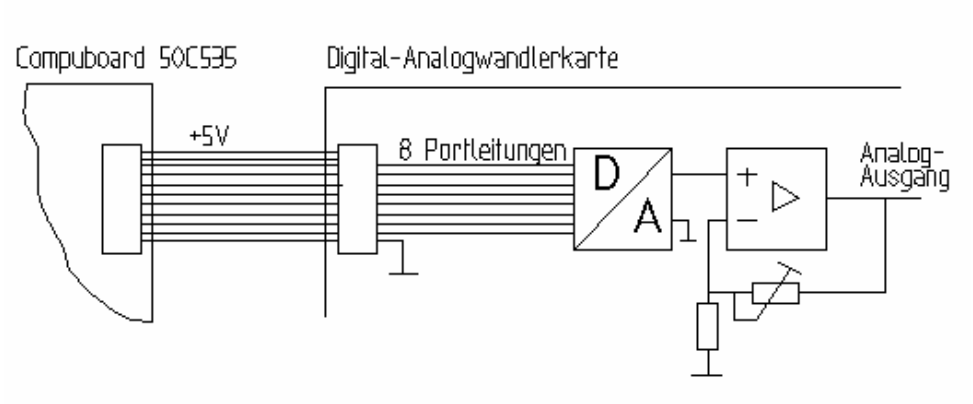


http://www.zum.de/dwu/depot/pas201k.gif 18.10.04

5. Erzeugung von Analogsignalen mit dem Controllerboard

5.1. Einführung: die D-A-Wandler-Platine

Als Eigenentwicklung steht in der EST eine “Analogplatine” zur Verfügung. Sie wird über ein 10-poliges Flachbandkabel an einen Controllerport angeschlossen und enthält einen 8 Bit-Digital-Analog-Wandler-IC mit nachfolgendem Pufferverstärker:



Der Digital-Analogwandler selbst besitzt eine Auflösung von

10 Millivolt pro Bit.

Damit erhält man bei einer Datenbusbreite von 8 Bit an seinem Ausgang eine Gleichspannung, die sich in Schritten von 10 mV zwischen

Null x 10mV = Null Volt **und** **255 x 10mV = 2,55 V**

ändern lässt. Der nachfolgende Pufferverstärker liefert schließlich beim Verdrehen des Potis eine Verstärkung zwischen V = 1 und V = 3.

Gibt man nun am Controllerport irgendeine Hexadezimalzahl aus, so erhält man nach etlichen Mikrosekunden am Analogausgang der Platine den zugehörigen Gleichspannungswert.

Vor dem Einstieg noch eine Erläuterung:

Analogsignale kann man entweder „Stück für Stück“ direkt programmieren ODER ihren Verlauf durch das Auslesen einer Wertetabelle für verschiedene Zeitpunkte vorgeben.

Wir schauen uns beide Methoden mal an.

5.2. Direkte Programmierung einer Dreieck-Spannung

Aufgabe:

Erzeugen Sie mit der D-A-Wandlerplatine eine symmetrische Dreieckspannung mit der Frequenz f = 50 Hz. Der Minimalwert soll 0,5 V, der Maximalwert dagegen 1,5 V betragen. Kontrollieren Sie den Frequenzwert mit einem Oszilloskop oder Frequenzzähler und bringen Sie ihn (durch Korrektur der Wartezeit) möglichst exakt auf 50 Hz.

Lösung:
Stellen wir das Potentiometer beim Ausgangsverstärker auf „Minimale Verstärkung = 1“, dann müssen wir für 0,5 Volt den Wert „50“ ausgeben. Zu 1,5 V gehört dann der Ausgabewert 150.

Wir starten also beim Wert 50 und inkrementieren bis 150, wobei wir bei jeder neuen Stufe eine Wartezeit einlegen. Anschließend dekrementieren wir von 150 bis herunter zu 50.

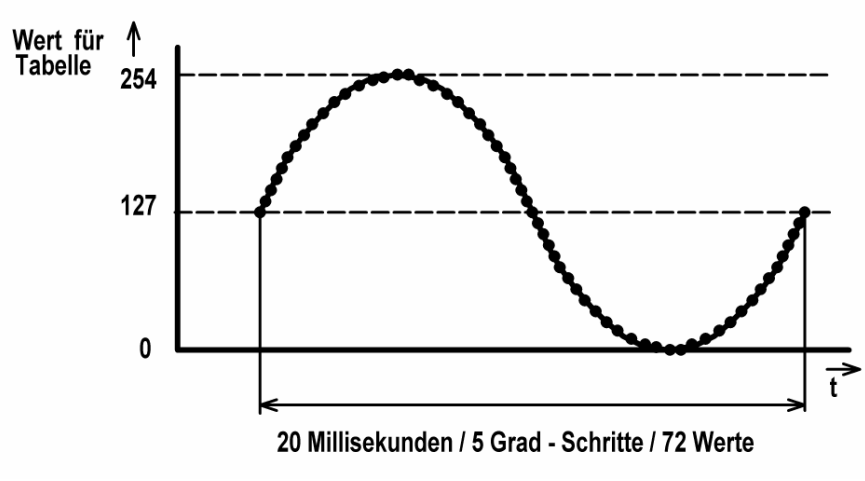
```
// Dreieck_01.c
// Erstellt am 20.10.2007 durch G. Kraus

#include<t89c51ac2.h>      // Header für AT89C51AC3
#include<stdio.h>          // Standard-Input-Output-Header

sfr ausgang=0x90;          // Port P1 ist der Digital-Ausgang

void warten(void);         // Prototyp

void main(void)
{    unsigned char x;
     AUXR=AUXR&0xFD;       // auf internes ERAM umschalten = EXTRAM löschen
     while(1)
     {    for(x=50;x<150;x++)
          {    ausgang=x;
               warten();
          }


          for(x=150;x>50;x--)
          {    ausgang=x;
               warten();
          }
     }
}

void warten(void)             // Wartezeit
{    char y;
     for(y=0;y<=27;y++);
}
```

5.3. Erzeugung eines Sinus-Signals (über eine Wertetabelle)

Aufgabe:

Programmieren Sie am Analogausgang der D-A-Platine eine sinusförmige Spannung mit der Frequenz f = 50Hz. Benutzen Sie dazu eine Wertetabelle mit 72 Werten pro Periode (= Schrittweite von 5 Grad).

Anleitung:



Man geht nach der nebenstehenden Skizze vor, berechnet zuerst die 72 Werte der Kurve nach der untenstehenden Formel und legt sie in einem „Array“ ab. Anschließend wird nacheinander jeder Wert des Arrays (von Punkt Null bis 71) ausgelesen und an den D-A-Wandler weitergegeben. Sind alle 72 Werte ausgelesen, dann beginnt man wieder von vorne.

Zwischen das Auslesen aus der Tabelle und dem Ausgeben von zwei Werten wird eine Wartezeit eingefügt. Sie bestimmt die Periodendauer und damit die Frequenz der erzeugten Sinusschwingung. Wird sie geändert, dann ändert sich die Ausgangsfrequenz der Sinusschwingung.

Die Formel zur Berechnung des Wertes für einen bestimmten Punkt lautet:

$$\mathbf{Wert} = \mathbf{127} + \mathbf{127} \bullet \mathbf{sin}(\boldsymbol{\alpha})$$

Diese Berechnung wird nun für Winkel von Null bis 355 Grad durchgeführt und das Ergebnis im Array „Sinus“ abgelegt. Es wird in „C“ folgende Form aufweisen, wenn man gleich auf- und abrundet:

```
code unsigned char Sinus[72]=

{  127,138,149,160,170,181,191,200,209,217,224,231,237,
       242,246,250,252,253,254,253,252,250,246,242,237,231,
       224,217,209,200,191,181,170,160,149,138,127,116,105,
       95,84,73,64,54,45,37,30,23,17,12,7,4,2,1,0,1,2,4,7,
       12,17,23,30,37,45,54,64,73,84,95,105,116
};
```

Nach dem Funktionstest des Programms wird die Frequenz mit einem Zähler oder Oszilloskop kontrolliert und die Wartezeit zwischen zwei Punkten solange verändert, bis die geforderte Frequenz von f = 50 Hz möglichst exakt erreicht ist.

```
// Sinus_01.c
// Erstellt am 24.10.2007 durch G. Kraus

/* Dieses Programm erzeugt eine Sinusspannung mit 50 Hz, wenn am Port P1
die D-A-Platine angeschlossen wird*/

//-------------------------------------------------------------
#include<t89c51ac2.h>     // Header für AT89C51AC3
#include<stdio.h>         // Standard - Input - Output - Header

sfr ausgang=0x90;         // Port P1 wird der Digital-Ausgang

void sinus_out(unsigned char wait);        // Prototyp

code unsigned char Sinus[72]=
   {
   127,138,149,160,170,181,191,200,209,217,224,231,237,242,246,250,252,
   253,254,253,252,250,246,242,237,231,224,217,209,200,191,181,170,160,
   149,138,127,116,105,95,84,73,64,54,45,37,30,23,17,12,7,
   4,2,1,0,1,2,4,7,12,17,23,30,37,45,54,64,73,84,95,105,116
   };
//-------------------------------------------------------------

void main(void)
{    AUXR=AUXR&0xFD;  // auf internes ERAM umschalten = EXTRAM löschen
     while(1)
     {    sinus_out(33);
     }
}


void sinus_out(unsigned char wait)    //Wartezeit
{    unsigned char x,y;

     for(x=0;x<72;x++)                // inkrementieren
     {    ausgang=Sinus[x];
          for(y=0;y<=wait;y++);       // Wartezeit nach jedem Hochzählen
     }
}
```

5.4. Sinus-Rechteck-Generator mit AUS-Taste für drei Festfrequenzen

Aufgabe:

Es soll ein kombinierter Sinus-Rechteck-Generator programmiert werden.
Ein Druck auf die Tasten P0.1 / P0.2 / P0.3 erzeugt einen Sinus mit 50 / 100 / 200 Hz als Dauersignal.
Ein Druck auf die Taste P0.4 / P0.5 / P0.6 erzeugt dagegen ein Rechteck mit 50 / 100 / 200 Hz als Dauersignal.
Jedes Signal kann nur durch einen Druck auf die Taste P0.0 wieder ausgeschaltet werden. Zwischen den einzelnen Signalen kann aber beliebig umgeschaltet werden.

Lösung:
Wir verwenden sowohl die Sinustabelle aus dem letzten Beispiel wie auch die „Merkersteuerung“ (= Speicherung eines Tastendruckes in einem speziell dafür angelegten „Merker-Bit“ -- Siehe auch das „Martinshorn“ – Beispiel). Das Rechteck-Signal wird wieder durch eine Zeitverzögerung mit nachfolgendem „Toggeln“ des Ausgangs-Portpins realisiert.

```
//-------------------------------------------------------------------------------------------------------------------------------
// Sinus_05.c
// Erstellt am 25.10.2007 durch G. Kraus

/* Ein Druck auf die Tasten P0.1 /  P0.2 / P0.3 erzeugt einen Sinus
mit 50 / 100 / 200 Hz  als Dauersignal.
Ein Druck auf die Taste P0.4 /  P0.5 / P0.6 erzeugt dagegen
ein Rechteck mit 50 / 100 / 200 Hz  als Dauersignal.
Jedes Signal kann nur durch einen Druck auf die Taste P0.0
wieder ausgeschaltet werden. */

#include<t89c51ac2.h>     // Header für AT89C51AC3
#include<stdio.h>         // Standard-Input-Output-Header

sfr ausgang=0x90;         // Port P1 wird der Digital-Ausgang

sbit Taste_AUS=P0^0;      // Tastendeklaration
sbit Taste_Sin50Hz=P0^1;
sbit Taste_Sin100Hz=P0^2;
sbit Taste_Sin200Hz=P0^3;
sbit Taste_Rechteck_50Hz=P0^4;
sbit Taste_Rechteck_100Hz=P0^5;
sbit Taste_Rechteck_200Hz=P0^6;


bit AUS;                        // Merker-Deklaration
bit Sin50Hz;
bit Sin100Hz;
bit Sin200Hz;
bit Rechteck50Hz;
bit Rechteck100Hz;
bit Rechteck200Hz;

void Rechteckper(int Wert);     // Prototypen
void Sinusper(char Wert);
void Tastenabfrage(void);



code unsigned char Sinus[72]=
     {127,138,149,160,170,181,191,200,209,217,224,231,237,242,246,250,252,
      253,254,253,252,250,246,242,237,231,224,217,209,200,191,181,170,160,
      149,138,127,116,105,95,84,73,64,54,45,37,30,23,17,12,7,
      4,2,1,0,1,2,4,7,12,17,23,30,37,45,54,64,73,84,95,105,116
     };
```

```
void main(void)
{     AUXR=AUXR&0xFD;     // auf internes ERAM umschalten = EXTRAM löschen
      P0=0xFF;                        // Port Po auf "Einlesen" schalten
      AUS=1;                          // Generator ausschalten
      while(1)
      {
            while(AUS==1)             // Generator ausgeschaltet
            {     Tastenabfrage();
            }

            while(Sin50Hz==1)         // Sinus 50 Hz wird erzeugt
            {     Sinusper(20);
                  Tastenabfrage();
            }

            while(Sin100Hz==1)        // Sinus 100 Hz wird erzeugt
            {     Sinusper(8);
                  Tastenabfrage();
            }

            while(Sin200Hz==1)        // Sinus 200 Hz wird erzeugt
            {     Sinusper(3);
                  Tastenabfrage();
            }

            while(Rechteck50Hz==1)    // Rechteck 50 Hz wird erzeugt
            {     Rechteckper(750);
                  Tastenabfrage();
            }

            while(Rechteck100Hz==1)   // Rechteck 50 Hz wird erzeugt
            {     Rechteckper(375);
                  Tastenabfrage();
            }

            while(Rechteck200Hz==1)   // Rechteck 50 Hz wird erzeugt
            {     Rechteckper(185);
                  Tastenabfrage();
            }

      }
}

void Sinusper(char Wert)
{     char x,y;
      for(x=0;x<72;x++)
      {     ausgang=Sinus[x];
            for(y=0;y<=Wert;y++);
      }
}

void Rechteckper(int Wert)
{     int y;
      ausgang=0x00;
      for(y=0;y<=Wert;y++);
      ausgang=0xFF;
      for(y=0;y<=Wert;y++);
}

void Tastenabfrage(void)          // Tastenabfrage
{     if(Taste_AUS==0)            // Aus-Taste gedrückt?
      {     AUS=1;
            Sin50Hz=0;
            Sin100Hz=0;
            Sin200Hz=0;
            Rechteck50Hz=0;
            Rechteck100Hz=0;
            Rechteck200Hz=0;
      }
```

```
      if(Taste_Sin50Hz==0)         // Sinus 50Hz-Taste gedrückt?
      {     AUS=0;
            Sin50Hz=1;
            Sin100Hz=0;
            Sin200Hz=0;
            Rechteck50Hz=0;
            Rechteck100Hz=0;
            Rechteck200Hz=0;
      }

      if(Taste_Sin100Hz==0)        // Sinus 100Hz--Taste gedrückt?
      {     AUS=0;
            Sin50Hz=0;
            Sin100Hz=1;
            Sin200Hz=0;
            Rechteck50Hz=0;
            Rechteck100Hz=0;
            Rechteck200Hz=0;
      }

      if(Taste_Sin200Hz==0)           // Sinus 200Hz-Taste gedrückt?
      {     AUS=0;
            Sin50Hz=0;
            Sin100Hz=0;
            Sin200Hz=1;
            Rechteck50Hz=0;
            Rechteck100Hz=0;
            Rechteck200Hz=0;
      }

      if(Taste_Rechteck_50Hz==0)        // Rechteck 50Hz-Taste gedrückt?
      {     AUS=0;
            Sin50Hz=0;
            Sin100Hz=0;
            Sin200Hz=0;
            Rechteck50Hz=1;
            Rechteck100Hz=0;
            Rechteck200Hz=0;
      }

      if(Taste_Rechteck_100Hz==0)     // Rechteck 100Hz-Taste gedrückt?
      {     AUS=0;
            Sin50Hz=0;
            Sin100Hz=0;
            Sin200Hz=0;
            Rechteck50Hz=0;
            Rechteck100Hz=1;
            Rechteck200Hz=0;
      }

      if(Taste_Rechteck_200Hz==0)     // Rechteck 200Hz-Taste gedrückt?
      {     AUS=0;
            Sin50Hz=0;
            Sin100Hz=0;
            Sin200Hz=0;
            Rechteck50Hz=0;
            Rechteck100Hz=0;
            Rechteck200Hz=1;
      }
}
```

5.5. Praktisches Beispiel aus der Medizintechnik: EKG-Signal

Aufgabe:

Erzeugen Sie mit dem Mikrocontroller ein "EKG - Signal" für 60 Pulsschläge in der Minute und mit einem Spitzenwert vom 2,55 Volt.

Anleitung:

a) Digitalisieren Sie dazu "von Hand" den Verlauf der beigefügten Musterkurve mit einer Auflösung von 8 Bit (= 256 Stufen von 0 bis 255) bei der Amplitude und 100 Messwerten pro Periode (das bedeutet: alle 10 Millisekunden). Tragen Sie die erhaltenen **Bitwerte** in eine Tabelle ein.

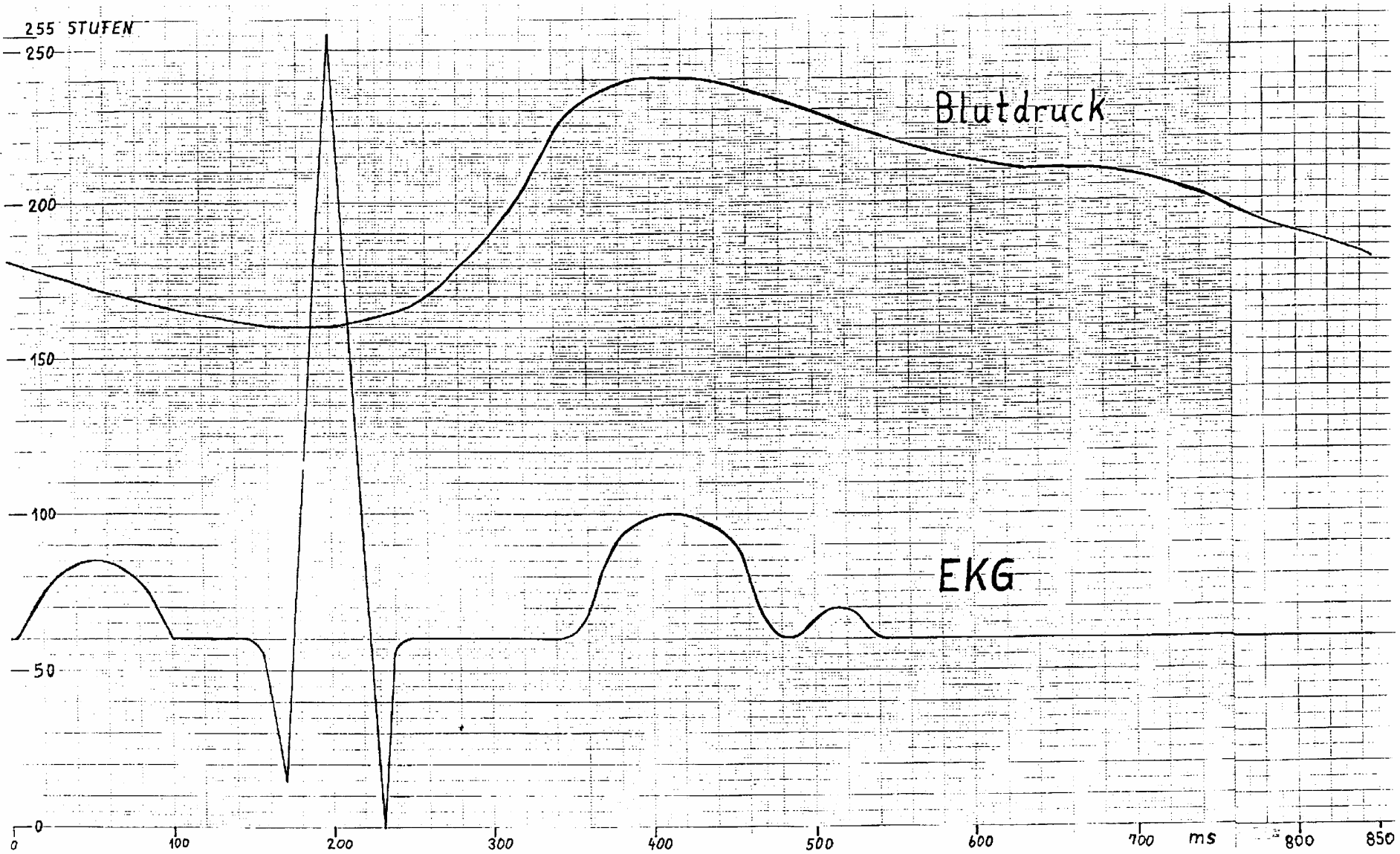
b) Erstellen Sie ein Programm, durch das immer ein Messwert aus der Tabelle geholt und anschließend die passende Wartezeit erzeugt wird.

c) Verbinden Sie den Ausgangsport P1 mit der D-A-Platine, messen Sie den Spitze-Spitze-Wert des entstehenden Analogsignals und stellen Sie mit dem Verstärkungs-Poti eine Amplitude von U_p =2,55 V ein (= minimale Verstärkung beim OPV hinter dem DA-Wandler).

Zu a)
Wenn man bei der auf dem nächsten Blatt folgenden EKG-Kurve alle 10 Millisekunden den Kurvenwert bestimmt, dann kann man folgendes Array zusammenstellen:

```
code unsigned char EKG [100]=     // Array wird im EPROM abgelegt
      {
         65,75,80,82,85,84,82,78,70,60,60,60,60,60,58,
         45,15,105,170,255,165,80,0,50,60,60,60,60,60,60,
         60,60,60,60,62,70,82,92,96,99,100,99,97,95,91,
         80,66,60,60,65,70,69,65,60,60,60,60,60,60,60,
         60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,
         60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,
         60,60,60,60,60,60,60,60,60,60
      };
```

Bitte genau nachzählen: es müssen exakt 100 Werte sein und nach dem letzten Wert darf **KEIN KOMMA** stehen!!


255 STUFEN
250
200
150
100
50
0
Blutdruck
EKG
0
100
200
300
400
500
600
700
ms
800
850

EKG-Lösung in C:

```
#include <t89c51ac2.h>          // Header für AT89C51AC3
#include <stdio.h>              // Standard-Input-Output-Header

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

void zeit (void);               // Prototyp anmelden

code unsigned char EKG [100]=   // Array wird im EPROM angelegt
   {
      65,75,80,82,85,84,82,78,70,60,60,60,60,60,58,
      45,15,105,170,255,165,80,0,50,60,60,60,60,60,60,
      60,60,60,60,62,70,82,92,96,99,100,99,97,95,91,
      80,66,60,60,65,70,69,65,60,60,60,60,60,60,60,
      60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,
      60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,
      60,60,60,60,60,60,60,60,60,60
   };
//-------------------------------------------------------------------------------

void main (void)
{     AUXR=AUXR&0xFD;       // auf internes ERAM umschalten = EXTRAM löschen
      while (1)
      {     unsigned char x;

            for(x=0; x<=99; x++)       // Schleife 100mal wiederholen
            {     ausgang=EKG[x];      // Tabellenwert ausgeben
                  zeit ();             // Aufruf der Wartezeit
            }

      }
}


void zeit (void)                       // Wartezeit

{     unsigned int y;                  // y als Zählvariable
      for(y=0; y<=1000; y++);          // Zeit verbrauchen
}
```

6. Einsatz des LCD-Displays mit HD44780-Controller

6.1. Gebrauchsanleitung

6.1.1. Grundlagen

In der Industrieanwendung der Mikrocontroller nimmt die Anzeige von Messergebnissen, die "Bedienerführung" und die Ausgabe von Kommentaren usw. mit den "Intelligenten LCD-Displays" einen immer größeren Umfang an. Als Standard haben sich für die meisten Anwendungen die ein- bis vierzeiligen Displays mit dem integrierten **Hitachi-Controller HD44780** (bzw. entsprechenden Lizenzfertigungen) durchgesetzt.
Sie besitzen insgesamt **80 Speicherplätze** in ihrem RAM, wobei pro Platz ein ASCII-Zeichen (oder ein selbst entworfenes Symbol) abgelegt werden kann. Die verschiedenen Displays unterscheiden sich lediglich in der Zahl der gleichzeitig anzeigbaren Zeichen, sie sind in folgenden Ausführungen erhältlich:

1 x 8 Zeichen / 2 x 16 Zeichen / 2 x 20 Zeichen / 2 x 24 Zeichen / 2 x 40 Zeichen / 4 x 16 Zeichen / 4 x 20 Zeichen, usw.

wobei auch Versionen mit Hintergrundbeleuchtung lieferbar sind.

Bedarf an "Leitungen":
Grundsätzlich wird jedes Zeichen als **8-Bit-Wert (= 1 Byte) vom Mikrocontroller an das Display übergeben**. Da zusätzlich noch **3 Steuerleitungen nötig** sind, haben wir insgesamt zwei verschiedene Möglichkeiten zur Kommunikation zwischen Display und Controller:

A) Man verwendet **einen Mikrocontrollerport** für die Ausgabe des Zeichenbytes, muss aber noch **zusätzlich** von einem weiteren Port **drei Leitungen** für die Display-Steuerung belegen.

B) Man arbeitet nur mit "**4-Bit-Daten-Übertragung**" und kommt dadurch **mit 4 + 3 = 7 Leitungen**, also einem **einzigen Port**, aus. Aber nun müssen das obere und untere Nibble **nacheinander übertragen** werden, was etwas mehr Zeit kostet..

Wir werden uns an die Methode B) der 4-Bit-Übertragung halten und **nur den** **Port P2** dafür spendieren!

6.1.2. Überblick über die Anschlussleitungen der LCD-Displays

Jedes Display weist insgesamt 14 Leitungen auf, die in einer Reihe oder in zwei Reihen angeordnet sind.
Es gilt immer:

Pin - Nr:	1	2	3	4	5	6	7	8	9	10
Funktion:	Masse	+5V	Kontrast	RS	R/W#	EN	D0	D1	D2	D3

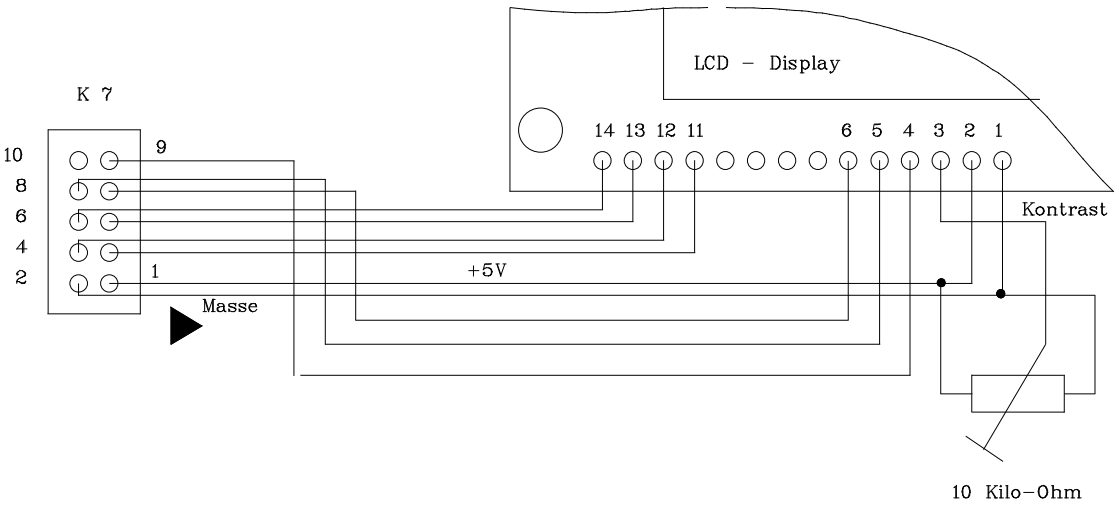
11	12	13	14
D4	D5	D6	D7

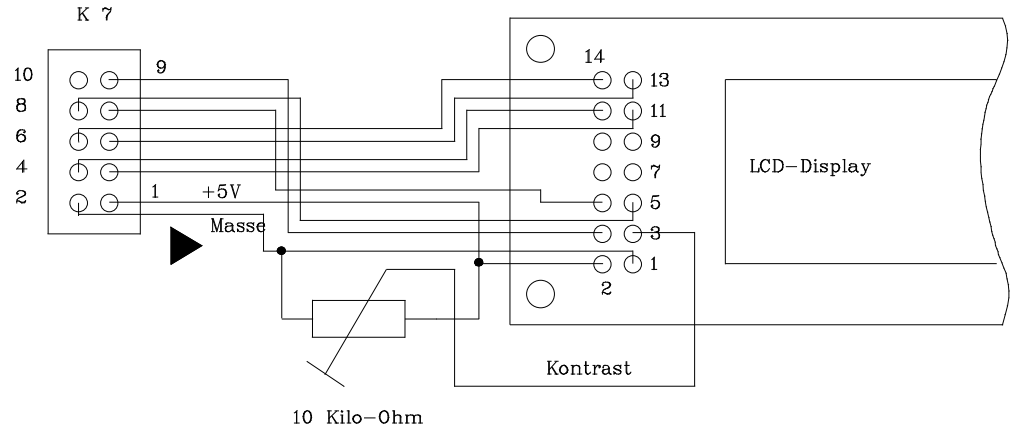
Bedeutung:
RS.....Register Select (log 1: Datenübertragung. log 0: Codebyte-Übertragung)
R/W#.....READ bzw. WRITE NOT (log 1: Lesen. log 0: Schreiben)
EN.....ENABLE (Aktivierung des LCD-Controllers durch Highpegel-Impuls mit der Pulslänge 1 Mikrosekunde. Anschließend sind aber bei verschiedenen Fällen Wartezeiten bis zu 5 Millisekunden nötig).

D0....D7 sind die Datenleitungen, wobei die 4**-Bit-Übertragung nur die Anschlüsse 11....14 verwendet!** (Anschluss 7....11 werden unbeschaltet gelassen).

Die **Kontrasteinstellung** wird durch ein **Potentiometer** vorgenommen, das mit **+5V versorgt** wird (Siehe nächste Seite).

Verbindung zwischen Controller und den verschiedenen Displaytypen über Flachbandkabel und Pfostenfeldstecker:





Hinter dieser “Verdrahtungsvorschrift” steckt folgende Zuordnung:

Portpin P2.6 = Register Select

Portpin P2.5 = Read / Write#

Portpin P2.4 = Enable

Portpin P2.3 = Databit 7

Portpin P2.2 = Databit 6

Portpin P2.1 = Databit 5

Portpin P2.0 = Databit 7

(Der letzte Portpin P2.7 bleibt unbenutzt, mit ihm kann man z. B. einen Lautsprecher über einen Trennverstärker ansteuern).

6.1.3. Korrekter Einsatz des Displays

Jedes dieser LCD-Displays muss beim **Einschalten** eine **streng festgelegte Prozedur**

„LCD_ein( )“

durchlaufen, da es sonst entweder **nicht richtig oder überhaupt nicht funktioniert**.

Anschließend werden die „Betriebswerte“ (= Anzahl der Zeilen, Cursor EIN / AUS, Cursor blinkt usw. in einer weiteren Routine

„LCD_ini( )“

festgelegt. Man lässt deshalb das „**Main“-Programm stets mit diesen beiden Prozeduren beginnen**, damit das Display später zu jeder beliebigen Zeit benützt werden kann.

Alle erforderlichen Routinen zum erfolgreichen Betrieb des Displays sind in einer extra geschriebenen Datei

„LCD_Control_ATMEL.c“

enthalten. Sie muss stets in den aktuellen Projektordner kopiert und dort in die Dateiverwaltung (zusammen mit der „Startup.A51“ und dem aktuellen C-Programm) eingebunden werden.

Sie steht jedem Anwender frei zur Verfügung**.**

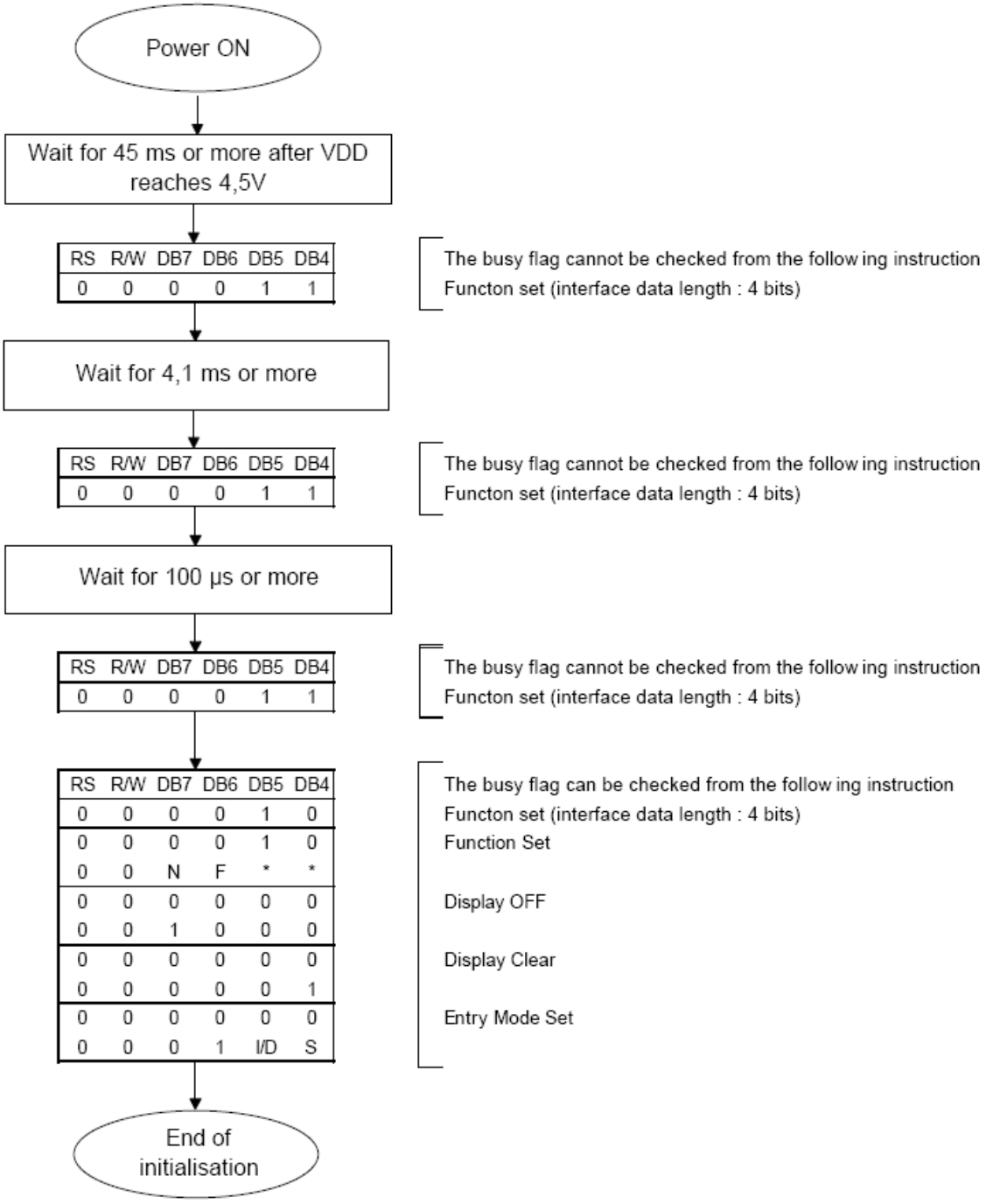
Zum richtigen Gebrauch des Displays muss es der Anwender an Port 2 des Controllers anschließen, die Einschalt-Routine und den Function-Set in seinem „main-Programm vorsehen und schließlich in seinem C-Programm folgende Funktionen verwenden:

switch_z1( )	schaltet den Cursor auf den Anfang von Zeile 1
switch_z2( )	schaltet den Cursor auf den Anfang von Zeile 2
switch_z3( )	schaltet den Cursor auf den Anfang von Zeile 3
switch_z4( )	schaltet den Cursor auf den Anfang von Zeile 4
show_text(char *ptr)	zeigt einen ASCII-Text in Form eines Arrays an, wenn der Arrayname (…und damit die Startadresse des ersten Zeichens im Display) übergeben wird
show_char(char *zeichen)	zeigt ein einzelnes ASCII-Zeichen an, wenn seine Adresse übergeben wird

6.1.4. Aus dem Display-Datenblatt: Einschaltroutine, Function – Set, Zeichensatz

Einschaltroutine (verwirklicht als Funktion **„LCD_ein( )“**

Table 6. Interface Data Length : Four bits



Note : in M4024, execute initialisation on E1 and E2 respectively

Function-Set , verwirklicht als Funktion „**LCD-ini( )“.**

Damit werden alle Einstellungen für den Einsatz des Displays im eigenen Projekt vorgenommen (einzeilige oder mehrzeilige Darstellung, Cursor EIN / AUS, Cursor blinkt, Cursor steht fest und Schrift wandert, Cursor wandert mit der Schrift usw. ….)

Die genauen Details können der folgenden Tabelle entnommen werden (…und für eigene Projekte angewandt..) werden.

Operating Instruction

AN No.SIG-CHMO9805A

Table 4 List of Instruction

Instruction	Code										Function	Execution time **
	RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0		
(1) Display Clear	0	0	0	0	0	0	0	0	0	1	Clears all display and returns cursor to home position (address 0)	1,64 ms
(2) Cursor Home	0	0	0	0	0	0	0	0	1	*	Returns cursor to home position, shifted display returns to home position and DD RAM contents do not change	1,64 ms
(3) Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets direction of cursor movement and whether display will be shifted when data is written or read	40 µm
(4) Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Turns ON/OFF total display (D) and cursor (C), and makes cursor position column start blinking (B)	40 µm
(5) Cursor/Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves cursor and shifts display without changing DD RAM contents.	40 µm
(6) Function Set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), the duty (N), and character fonts (F)	40 µm
(7) CG RAM Address Set	0	0	0	1	A_{CG}						Sets CG RAM address to start transmitting or receiving CG RAM data	40 µm
(8) DD RAM Address Set	0	0	1	A_{DD}							Sets DD RAM address to start transmitting or receiving DD RAM data	40 µm
(9) BF/Address Read	0	1	BF	AC							Reads BF indicating module in internal operation and AC contents (use for both CG RAM and DD RAM)	0 µm
(10) Data Write to CG RAM or DD RAM	0	1	Write Data								Writes data into DD RAM or CG RAM	40 µm t_{ADD}=8µm
(11) Data Read from CG RAM or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM	40 µm t_{ADD}=8µm

* : Don't care bit
A_{CG} : CG RAM address
A_{DD} : DD RAM address
AC : Address counter

I/D = 1 : Increment
I/D = 0 : Decrement

S = 1 : Display shift
S = 0 : No display shift

B = 1 : Blink ON
B = 0 : Blink OFF

S/C = 1 : Display shift
S/C = 0 : Cursor movement

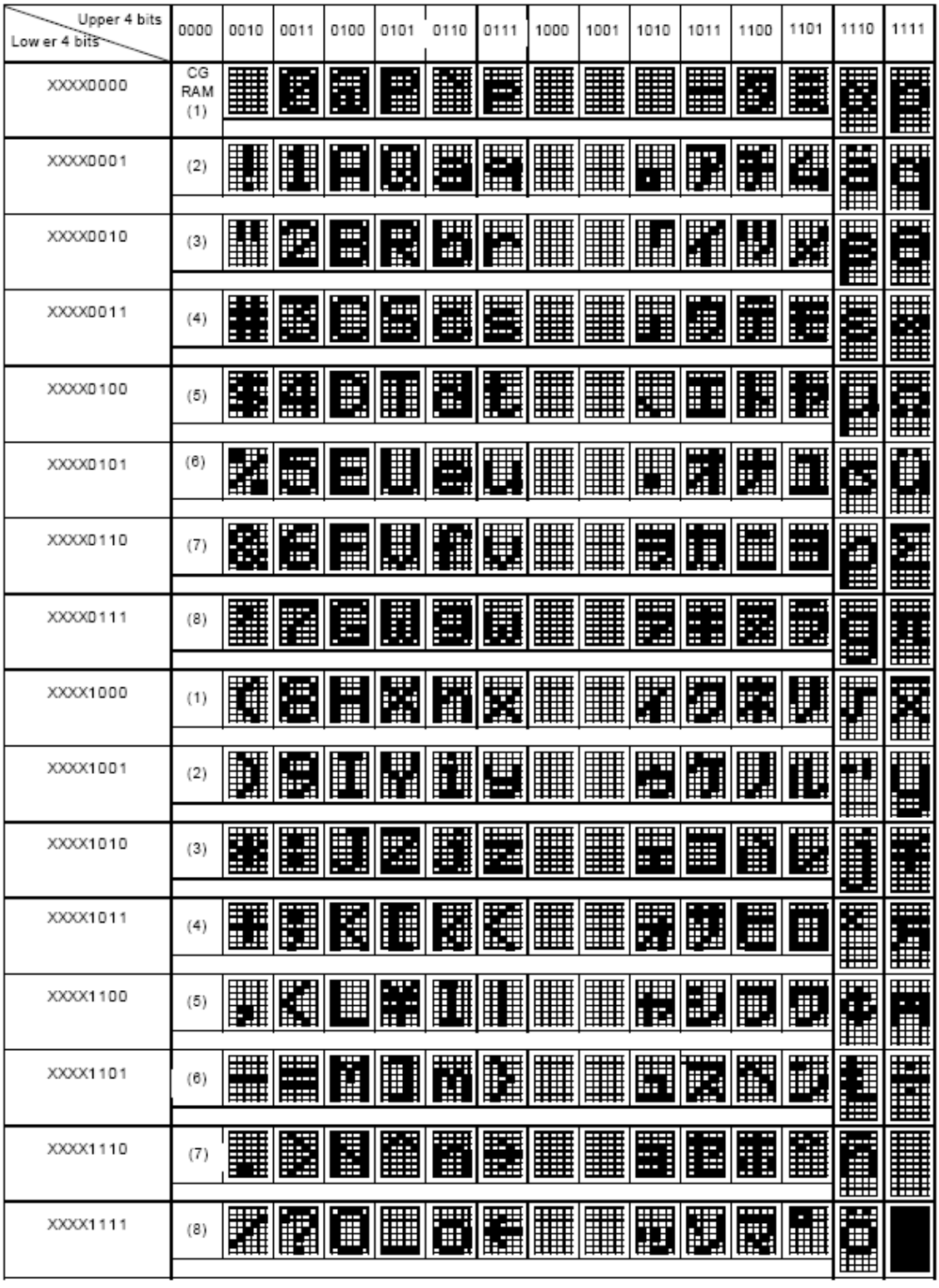
N = 1 : 1/16 duty
N = 0 : 1/8 duty or 1/11 duty

F = 1 : 5 x 10 dot matrix
F = 0 : 5 x 7 dot matrix

Zeichensatz des Displays mit zugehörigen Code-Bytes.

Achtung: Dieser Zeichensatz ist bei verschiedenen Herstellern unterschiedlich!
Nur die wichtigsten ASCII-Zeichen sind überall identisch (Zahlen, Buchstaben etc.)

Table 2 Correspondence between character codes and character pattern (5x7 dot -matrix)



6.1.5. Die komplette Datei „“LCD_Ctrl_ATMEL.c“

Hier können bei Bedarf einzelne Einstellungen geändert werden. Dazu muss darin die zugehörige Anweisung gesucht und dann das neue Codewort laut der vorhin vorgestellten Liste „Function Set“ eingetragen werden.

```
/*-------------------------------------------------------------------
LCD_Ctrl_ATMEL.c
C-Programm-Modul  zur Ausgabe von Texten auf dem LCD-Display
02.11.2007 / G. Kraus
-------------------------------------------------------------------*/
#include <t89c51ac2.h>            // Header für AT89C51AC3
#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=0xA0;              // Immer Port 2 (Adresse 0xA0) als Display-Ausgang!

sbit enable=ausg_lcd^4;         // Portpin P2.4 ist "enable"
sbit RW_LCD=ausg_lcd^5;         // Portpin P2.5 ist "READ - WRITE"
sbit RS_LCD=ausg_lcd^6;         // Portpin P2.6 ist "Register - Select"

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

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


void zeit_l(void)       // "lange Wartezeit" ergibt 4 Millisekunden
{  unsigned int x;
   for(x=0;x<=1000;x++);
}


void lcd_ein(void)      // Einschaltroutine für Display
{  unsigned char x,y;
   for(x=0;x<=11;x++)   // mindestens 12 x 4 ms = 48 ms warten
   {  zeit_l();
   }

   ausg_lcd=0x03;       // Dreimal nacheinander dasselbe Codewort
   for(y=0;y<=2;y++)    // "0011B"  =  0x03 mit Wartepausen zum Display schicken
   {  enable=1;         // Damit wird der "Function set" gestartet
      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;       // Codewort hat als oberes Nibble "1000B" = 0x08
      lcd_com();           // losschicken
      ausg_lcd=0x00;       // 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;       // Codewort hat als oberes Nibble "1100B" = 0x0C
      lcd_com();           // losschicken
      ausg_lcd=0x00;       // 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;       // Codewort hat als oberes Nibble "1001B" = 0x09
      lcd_com();           // losschicken
      ausg_lcd=0x04;       // 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;       // Codewort hat als oberes Nibble "1101B" = 0x0D
      lcd_com();           // losschicken
      ausg_lcd=0x04;       // 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 dann 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
}
```

6.2. Ein kleines Demoprogramm zur Anzeige von Text und Zeichen

Bitte bei allen Display-Einsätzen daran denken:

Die „Source Group1“ in der Projektverwaltung muss nun IMMER drei Files enthalten:

die „STARTUP.A51“

die „LCD_Ctrl_ATMEL.c“

und natürlich das selbstgeschriebene C-Anwendungprogramm.

Bitte vor dem Compilieren stets kontrollieren!

```
/*-------------------------------------------------------------------
LCD_DEMO.c
22.10.2007 / G.Kraus

Angezeigt wird in der ersten Zeile ein Text, der im EPROM gespeichert ist. In der
zweiten Zeile wird ein im Controller-RAM angelegter Character-String ausgegeben,
gefolgt von einem einzigen Zeichen (beispielsweise einem Messwert als Variable y).
-------------------------------------------------------------------*/
#include <t89c51ac2.h>          // Header für AT90C51AC3
#include <stdio.h>              // Standard-Input-Output-Header

void zeit_s(void);              // Protoypen der Display-Steuerung immer angeben!!
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);


code unsigned char zeile_1[21]={"LCD-Display in C    "};     // Text in Zeile 1

void main(void)
{     AUXR=AUXR&0xFD;     // auf internes ERAM umschalten = EXTRAM löschen
      char zeile_2[21]={"Messwert: "};        // Deklaration des Anzeige-Arrays
      unsigned char y='5';                    // Variable y = ASCII-Wert für 5

      lcd_ein();                              // Einschaltroutine
      lcd_ini1();                             // Function Set
      switch_z1();                            // Schalte auf Zeile 1
      show_text(zeile_1);                     // Text in Zeile 1 zeigen
      switch_z2();                            // Schalte auf Zeile 2
      show_text(zeile_2);                     // Text in Zeile 2 zeigen
      show_char(&y);                          // y zusätzlich anzeigen

      while(1);                               // Endlosschleife
}
```

6.3. Etwas anspruchsvoller: Testprogramm zur Anzeige des LCD-Zeichensatzes

```
/*-------------------------------------------------------------
Demo-Programm zur Ausgabe der verfügbaren Zeichen bei einem LCD-Display
25.10.2007 / G. Kraus

Angezeigt wird in der ersten Zeile das Wort "Hex: ", gefolgt von einer Hex-Zahl.
Diese wird von 0x00 bis 0xFF hochgezählt, zwischen jeden Zählschritt eine
Pause gelegt und zum LCd-Display gesendet.
In der zweiten Zeile steht "gibt:  ", gefolgt vom ASCII-Zeichen, das bei diesem
Display zu der jeweiligen Hex-Zahl gehört.

ACHTUNG: Das LCD-Display wird stets an Port P2(Adresse 0xA0) angeschlossen!!
-------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------------------*/
#include <t89c51ac2.h>     // Header für AT89C51AC3
#include <stdio.h>         // Standard-Input-Output-Header
/*-------------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------------------*/
void zeit_s(void);      // Prototypen für "LCD_CTRL.C" - 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);
void show_char(char *zeichen);

code unsigned char hex_ascii[17]={"0123456789ABCDEF"}; // Hex-Zahlen als ASCII

void wait(void);     // Prototyp für Warteschleife

void main(void)
{  unsigned char zeile_1[5]={"Hex "};   // erste Zeile
   unsigned char zeile_2[7]={"gibt: "}; // Zweite Zeile
   unsigned char x=0;                   // Inkrementierte Hex-Zahl
   AUXR=AUXR&0xFD;       // auf internes ERAM umschalten = EXTRAM löschen
   lcd_ein();                           // Einschaltroutine
   lcd_ini1();                          // Function Set

   while(1)
   {     unsigned char upper_nibble;     // Oberes Nibble der Hex-Zahl
         unsigned char lower_nibble;     // Unteres Nibble der Hex-Zahl

         upper_nibble=hex_ascii[x/16];      // Berechnung des Oberen Nibbles
         lower_nibble=hex_ascii[x&0x0F];    // Berechnung des Unteren Nibbles
         switch_z1();                       // Schalte auf Zeile 1
         show_text(zeile_1);                // Text in Zeile 1 zeigen
         show_char(&upper_nibble);       // Zeige ASCII-Version des Oberen Nibbles
         show_char(&lower_nibble);       // Zeige ASCII-Version des Unteren Nibbles

         switch_z2();                    // Schalte auf Zeile 2
         show_text(zeile_2);             // Text in Zeile 2 zeigen
         show_char(&x);             // Zeige das zugehörige Zeichen auf dem Display

         x++;                       // Hex-Zahl inkrementieren
         wait();                    // warten
   }
}

void wait(void)                     // Warte-Funktion
{     unsigned int y;
      for(y=0;y<=40000;y++);
}
```

6.4. Und die Krönung: Sinus-Rechteck-Generator mit Frequenzanzeige

Aufgabe:

Es soll ein kombinierter Sinus-Rechteck-Generator programmiert werden.
Ein Druck auf die Tasten P0.1 / P0.2 / P0.3 erzeugt einen Sinus mit 50 / 100 / 200 Hz als Dauersignal.
Ein Druck auf die Taste P0.4 / P0.5 / P0.6 erzeugt dagegen ein Rechteck mit 50 / 100 / 200 Hz als Dauersignal.
Jedes Signal kann nur durch einen Druck auf die Taste P0.0 wieder ausgeschaltet werden. Zwischen den einzelnen Signalen kann aber beliebig umgeschaltet werden.
Über die ausgegeben Frequenz und Kurvenform soll über das Display informiert werden, ebenso soll der AUS-Zustand angezeigt werden.

Lösung:
Wir verwenden sowohl eine Sinustabelle wie auch die „Merkersteuerung“ (= Speicherung eines Tastendruckes in einem speziell dafür angelegten „Merker-Bit“ -- Siehe auch das „Martinshorn“ – Beispiel).
Das Rechteck-Signal wird wieder durch eine Zeitverzögerung mit nachfolgendem „Toggeln“ des Ausgangs-Portpins realisiert.
Alle anzuzeigenden Texte werden als „Code-Arrays“ deklariert. Dadurch werden sie im EPROM abgelegt und fressen keinen kostbaren RAM-Speicher.

```
//-------------------------------------------------------------------------------------------------------------------------------
// Generator_mit_Display.c
// Erstellt am 27.10.2007 durch G. Kraus
// Tasten an P0, Display an P2, D-A-Wandler an P1
// Frequenzen: 50 / 100 / 200 Hz

#include<t89c51ac2.h>             // Header für AT89C51AC3
#include<stdio.h>                 // Standard-Input-Output-Header

sfr ausgang=0x90;                 // Port P1 wird der Digital-Ausgang

sbit Taste_AUS=P0^0;              // Tastendeklaration für Port P0
sbit Taste_Sin50Hz=P0^1;
sbit Taste_Sin100Hz=P0^2;
sbit Taste_Sin200Hz=P0^3;
sbit Taste_Rechteck_50Hz=P0^4;
sbit Taste_Rechteck_100Hz=P0^5;
sbit Taste_Rechteck_200Hz=P0^6;


bit AUS;                          // Merker-Deklaration
bit Sin50Hz;
bit Sin100Hz;
bit Sin200Hz;
bit Rechteck50Hz;
bit Rechteck100Hz;
bit Rechteck200Hz;

void Rechteckper(int Wert);       // Protoypen für Generator
void Sinusper(char Wert);
void Tastenabfrage(void);

void zeit_s(void);                // Prototypen für Display
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);
```

```
code unsigned char Sinus[72]=
     {  127,138,149,160,170,181,191,200,209,217,224,231,237,242,246,250,252,
        253,254,253,252,250,246,242,237,231,224,217,209,200,191,181,170,160,
        149,138,127,116,105,95,84,73,64,54,45,37,30,23,17,12,7,
        4,2,1,0,1,2,4,7,12,17,23,30,37,45,54,64,73,84,95,105,116
     };

code unsigned char            Text_Aus[21]={"Generator AUS       "};
code unsigned char     Text_Sinus_50Hz[21]={"Sinus f = 50 Hz     "};
code unsigned char    Text_Sinus_100Hz[21]={"Sinus f = 100 Hz    "};
code unsigned char    Text_Sinus_200Hz[21]={"Sinus f = 200 Hz    "};
code unsigned char  Text_Rechteck_50Hz[21]={"Rechteck f = 50 Hz  "};
code unsigned char Text_Rechteck_100Hz[21]={"Rechteck f = 100 Hz "};
code unsigned char Text_Rechteck_200Hz[21]={"Rechteck f = 200 Hz "};


void main(void)
{  AUXR=AUXR&0xFD;  // auf internes ERAM umschalten = EXTRAM löschen
   lcd_ein();
   lcd_ini1();
   switch_z1();
   show_text(Text_Aus);
   AUS=1;                                 // Generator ausschalten

   while(1)
   {
      while(AUS==1)                       // Generator ausgeschaltet
      {  Tastenabfrage();
      }

      while(Sin50Hz==1)                   // Sinus 50 Hz wird erzeugt
      {  Sinusper(20);
         Tastenabfrage();
      }

      while(Sin100Hz==1)                  // Sinus 100 Hz wird erzeugt
      {  Sinusper(8);
         Tastenabfrage();
      }

      while(Sin200Hz==1)                  // Sinus 200 Hz wird erzeugt
      {  Sinusper(3);
         Tastenabfrage();
      }

      while(Rechteck50Hz==1)              // Rechteck 50 Hz wird erzeugt
      {  Rechteckper(650);
         Tastenabfrage();
      }

      while(Rechteck100Hz==1)             // Rechteck 50 Hz wird erzeugt
      {  Rechteckper(325);
         Tastenabfrage();
      }

      while(Rechteck200Hz==1)             // Rechteck 50 Hz wird erzeugt
      {  Rechteckper(162);
         Tastenabfrage();
      }

   }
}

void Sinusper(char Wert)              // Wartezeit
{  char x,y;
   for(x=0;x<72;x++)
   {  ausgang=Sinus[x];
      for(y=0;y<=Wert;y++);
   }
```

```
}

void Rechteckper(int Wert)           // Wartezeit
{  int y;
   ausgang=0x00;
   for(y=0;y<=Wert;y++);
   ausgang=0xFF;
   for(y=0;y<=Wert;y++);
}


void Tastenabfrage(void)             // Tastenabfrage
{  if(Taste_AUS==0)                  // Aus-Taste gedrückt?
   {  AUS=1;
      Sin50Hz=0;
      Sin100Hz=0;
      Sin200Hz=0;
      Rechteck50Hz=0;
      Rechteck100Hz=0;
      Rechteck200Hz=0;
      switch_z1();                      // Auf den Anfang von Zeile 1 schalten
      show_text(Text_Aus);              // Text anzeigen
   }

   if(Taste_Sin50Hz==0)                 // Sinus 50Hz-Taste gedrückt?
   {  AUS=0;
      Sin50Hz=1;
      Sin100Hz=0;
      Sin200Hz=0;
      Rechteck50Hz=0;
      Rechteck100Hz=0;
      Rechteck200Hz=0;
      switch_z1();                      // Auf den Anfang von Zeile 1 schalten
      show_text(Text_Sinus_50Hz);       // Text anzeigen
   }

   if(Taste_Sin100Hz==0)                // Sinus 100Hz--Taste gedrückt?
   {  AUS=0;
      Sin50Hz=0;
      Sin100Hz=1;
      Sin200Hz=0;
      Rechteck50Hz=0;
      Rechteck100Hz=0;
      Rechteck200Hz=0;
      switch_z1();                      // Auf den Anfang von Zeile 1 schalten
      show_text(Text_Sinus_100Hz);      // Text anzeigen
   }

   if(Taste_Sin200Hz==0)                // Sinus 200Hz-Taste gedrückt?
   {  AUS=0;
      Sin50Hz=0;
      Sin100Hz=0;
      Sin200Hz=1;
      Rechteck50Hz=0;
      Rechteck100Hz=0;
      Rechteck200Hz=0;
      switch_z1();                      // Auf den Anfang von Zeile 1 schalten
      show_text(Text_Sinus_200Hz);      // Text anzeigen
   }

   if(Taste_Rechteck_50Hz==0)           // Rechteck 50Hz-Taste gedrückt?
   {  AUS=0;
      Sin50Hz=0;
      Sin100Hz=0;
      Sin200Hz=0;
      Rechteck50Hz=1;
      Rechteck100Hz=0;
      Rechteck200Hz=0;
      switch_z1();                      // Auf den Anfang von Zeile 1 schalten
      show_text(Text_Rechteck_50Hz);    // Text anzeigen
```

```
  }

    if(Taste_Rechteck_100Hz==0)       // Rechteck 100Hz-Taste gedrückt?
  {  AUS=0;
     Sin50Hz=0;
     Sin100Hz=0;
     Sin200Hz=0;
     Rechteck50Hz=0;
     Rechteck100Hz=1;
     Rechteck200Hz=0;
     switch_z1();                     // Auf den Anfang von Zeile 1 schalten
     show_text(Text_Rechteck_100Hz);  // Text anzeigen
  }

    if(Taste_Rechteck_200Hz==0)       // Rechteck 200Hz-Taste gedrückt?
  {  AUS=0;
     Sin50Hz=0;
     Sin100Hz=0;
     Sin200Hz=0;
     Rechteck50Hz=0;
     Rechteck100Hz=0;
     Rechteck200Hz=1;
     switch_z1();      // Auf den Anfang von Zeile 1 schalten
     show_text(Text_Rechteck_200Hz);  // Text anzeigen
  }
}
```

7. Der Integrierte A-D-Wandler

7.1. Einführung und Grundlagen

Auf dem Chip des Controllers findet sich ein integrierter AD-Wandler mit folgenden Eigenschaften:

==========

a) **10 Bit Auflösung**

b) **Wahl zwischen „Standard“- Modus (8 Bit) und „High Precision Modus“ (10 Bit)**

c) **8 umschaltbare Analog-Eingänge (= Zweitfunktion von Port P1)**

d) **Trennung zwischen Portpin-Umschaltung auf Zweitfunktion und Auswahl des Analog-Einganges**

e) **Interrupt-Steuerung möglich**

f) **Keine integrierte Referenzspannung. Verwendete Referenzspannung muss zwischen +2,4 und maximal +3 V liegen. Umschaltung auf Externe Referenzspannung durch Jumper möglich.**

==========

Erläuterungen:

Das **Messergebnis** findet sich als **8 Bit-Wert im ADDH-Register**, während die **beiden weiteren Bits** als Bit 0 und Bit 1 im **ADDL-Register** untergebracht sind.

Egal, ob Standard- oder High-Precision-Mode: Es **wird IMMER mit 10 Bit Auflösung gemessen** (R / 2R-Prinzip, Wandelzeit ca. 17µs). Der entscheidende Unterschied ist, dass im „High Precision Mode“ während der Wandlung die CPU komplett abgeschaltet wird, um „Digital Noise“ auf dem Chip zu vermeiden und so die Genauigkeit zu erhöhen bzw. sicherzustellen. Allerdings können in diesem Fall der Wandlerstart und die Auswertung nur durch einen **Interrupt** erfolgen, der nach der Messung die CPU wieder aufweckt.
Alle zugehörigen Steuerfunktionen finden sich **im ADCON-Register** und müssen vom Anwender programmiert werden.

Da keine zusätzlichen getrennten Analogeingänge vorhanden sind, muss man zuerst den gewünschten Portpin von P1 auf „Zweitfunktion AD-Eingang“ umschalten (..die übrigen Pins können ganz normal weiterverwendet werden). Anschließend wird -- wieder über das ADCON-Register! -- die Auswahl des Messeingangs über einen Analog-Multiplexer vorgenommen.

Der Referenzeingang verlangt mindestens eine Referenzspannung von +2,4 V, darf aber nicht mehr als +3 V erhalten. Deshalb wird er über einen Spannungsteiler von der Versorgungsspannung (+5 V) gespeist. Zwei Jumper ermöglichen aber die Umschaltung auf eine externe Referenzquelle.

Vorsicht:Keine höheren Spannungen als +3V und keine negativeren Spannungen als -0,2V an die Analogeingänge anlegen. Lieber vorsichtshalber Spannungsteiler vorsehen!

Warnung: Die an einen Analogport angelegte Spannung wird während eines Messvorganges durch die Wandlerelektronik des Controllers stark belastet. Deshalb sollte der Innenwiderstand der zugehörigen Spannungsquelle (z. B. Sensor) deutlich **kleiner als 5 kΩ** sein.

7.2. Register des AD-Wandlers

Zuerst muss der benutzte P1-Portpin auf seine Zweitfunktion umgeschaltet werden. Das geschieht **im ADCF-Register**:

Table 58. ADCF Register

ADCF (S:F6h)
ADC Configuration

7	6	5	4	3	2	1	0
CH 7	CH 6	CH 5	CH 4	CH 3	CH 2	CH 1	CH 0

Bit Number	Bit Mnemonic	Description
7-0	CH 0:7	**Channel Configuration** Set to use P1.x as ADC input. Clear to use P1.x as standart I/O port.

Reset Value =0000 0000b

Bitte beachten:

Beim auszuwählenden Portpin muss eine „1“ gesetzt werden. Bei der Programmerstellung passiert aber leicht ein Fehler, denn es handelt sich um eine „lineare Adressierung im Dezimalsystem“. Wir müssen aber im C-Programm die zugehörige Hex-Zahl angeben!!

Beispiel: wenn wir Portpin P1^3 = Channel 3 wählen, sitzt die „1“ auf der vierten Stelle von Rechts. Die Eingabe im Programm muss jetzt lauten:

ADCF=0x08;

Dann geht es an das ADCON-Register:

Table 59. ADCON Register

ADCON (S:F3h)
ADC Control Register

7	6	5	4	3	2	1	0
-	PSIDLE	ADEN	ADEOC	ADSST	SCH2	SCH1	SCH0

Bit Number	Bit Mnemonic	Description
7	-	
6	PSIDLE	**Pseudo Idle Mode (Best Precision)** Set to put in idle mode during conversion Clear to convert without idle mode.
5	ADEN	**Enable/Standby Mode** Set to enable ADC Clear for Standby mode (power dissipation 1 uW).
4	ADEOC	**End Of Conversion** Set by hardware when ADC result is ready to be read. This flag can generate an interrupt. Must be cleared by software.
3	ADSST	**Start and Status** Set to start an A/D conversion. Cleared by hardware after completion of the conversion
2-0	SCH2:0	**Selection of Channel to Convert** see Table 57

Reset Value =X000 0000b

Sehen wir uns das mal von Links nach Rechts an:

Bit 7 hat keine Wirkung

Wird **Bit 6** gesetzt, dann arbeiten wir im „Best Precision Mode“) und die CPU wird während der Messung abgeschaltet. **Achtung: Die Auswertung kann dann nur über einen Interrupt erfolgen.**

Mit **ADEN** = AD-Enable = **Bit 5** = 1 wird der AD-Wandler überhaupt erst freigegeben.

Sobald das Wandelergebnis vorliegt, setzt der Controller das **Bit 4 = ADEOC** = AD – End of Conversion. Es muss vom Anwender wieder per Software gelöscht werden.

Durch das Setzen von Bit 3 = ADSST = AD-Start and -Status wird eine Wandlung gestartet . Wird von der Hardware automatisch nach der Messung wieder gelöscht.

Die **Bits 0…2** dienen zur Auswahl des gewünschten Channels. Achtung: hier ist die **Eingabe der Kanalnummer im Dualcode** erforderlich.

7.3. Einfaches Einführungsbeispiel

Aufgabe:

Port P1 wird an die Potentiometer-Kette auf der D-A-Platine über ein Flachbandkabel angeschlossen. Die Spannung an Portpin P1.3 soll im Standard-Modus gemessen und das Ergebnis direkt in Port P2 kopiert werden. Dort befindet sich die LED-Kette und zeigt das Hex-Ergebnis durch Aufleuchten der LEDs an.

Bitte diesen erprobten Lösungsweg genau analysieren und stets verwenden… alles Andere gab Ärger…..

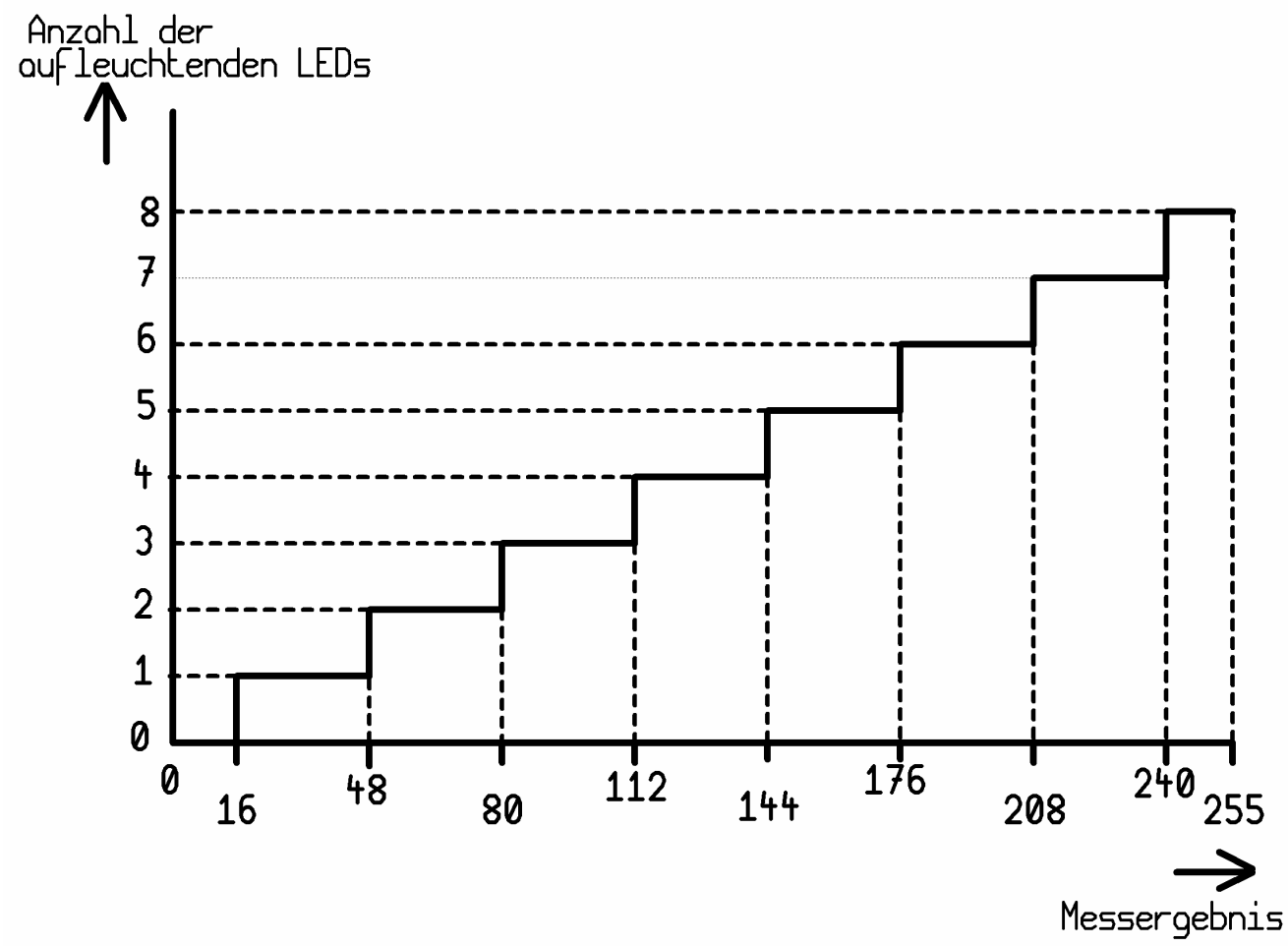
```
:

/*-------------------------------------------------------------
Programmbeschreibung: Einführung AD-Wandler
-------------------------------------------------------------
Name:           ADC_01.c
Funktion:       Am Portpin P1.3 = AN3 wird die anliegende Gleichspannung dauernd im
                Standard-Modus gemessen und das Ergebnis an Port P2 mit der LED-
                Kette angezeigt.
WARNUNG :
Bitte keine höheren Eingangsspannungen als max. +3V an die Analogeingänge anlegen!

Datum:          24. 10. 2007
Autor:          G. Kraus
-------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------*/
#include <t89c51ac2.h>       // Header für Controller "AT89C51AC3"
#include <stdio.h>           // Standard-Input-Output-Header

unsigned char channel;       // Deklaration der Kanal-Nummer
/*-------------------------------------------------------------
Prototypen
-------------------------------------------------------------*/
void wait_ADC_ini(void);     // Wartezeit nach der Wandlerfreigabe

/*-------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------*/

void main(void)
 {      AUXR=AUXR&0xFD;  // auf internes ERAM umschalten = EXTRAM löschen
        channel=0x03;             // Kanal AN3 gewählt
        ADCF=0x08;                // Pin P1.3 auf „A-D-Channel" umschalten
        ADCON=0x20;               // AD-Wandlung freigeben
        wait_ADC_ini();           // warten, bis ADC initialisiert

        while(1)
        {     ADCON=ADCON&0xF8;     // Alte Kanaleinstellung (SCH0...SCH2) löschen
              ADCON=ADCON|channel;  // Auf Channel AN3 schalten
              ADCON=ADCON|0x08;     // Wandlung durch Setzen von ADSST starten
              while((ADCON&0x10)!=0x10);   // Warten, bis am Ende ADEOC gesetzt wird
              ADCON=ADCON&0xEF;            // ADEOC wieder löschen
              P2=ADDH;                     // Ergebnis-Übergabe an die LEDs
        }
}

void wait_ADC_ini(void)              // Einschalt-Wartezeit beim Wandler nötig !
{       unsigned char x;
        for(x=0;x<5;x++);
}
```

7.4. LED-Balkenanzeige für Messergebnis

Sehr oft reicht es, nur die ungefähre Amplitude des Messergebnisses optisch gut sichtbar zu machen. Eine beliebte Lösung dafür ist die „**Balkenanzeige**“. Deshalb soll das Beispiel aus dem vorigen Kapitel etwas umgebaut werden.

Aufgabe:
Programmieren Sie mit den 8 LEDs auf der Zusatzplatine (an Port 2) eine Balkenanzeige für die gemessene Analogspannung am Eingang AN3. Eine maximale Eingangsspannung von ca. +3V soll alle 8 LEDs aufleuchten lassen. Die „Umschaltstufen“ sind gleichmäßig über den Messbereich zu verteilen.

Lösung:
Zuerst müssen anhand eines Diagrammes die Umschaltpunkte (an denen die nächste LED dazukommt) festgelegt werden. Bei 8 Bit Auflösung unseres Wandlers teilt man deshalb den Gesamtbereich in 16 gleich große Segmente auf, um „in der Mitte eines Fensters“ umschalten zu können. Das sieht dann so aus:



Im Programm muss nach der Bereitstellung des Messergebnisses eine Entscheidung getroffen werden, in welches der Fenster es hineinpasst -- und anschließend die korrekte Anzahl und Reihenfolge der LEDs für diesen Bereich eingeschaltet werden. Das sieht z. B. so aus:

```
if(ADDH<16)                    {  LED_Ausgang=0x00;  }

if((ADDH>=16)&&(ADDH<48))      {  LED_Ausgang=0x01;  }

if((ADDH>=48)&&(ADDH<80))      {  LED_Ausgang=0x03;  }

if((ADDH>=80)&&(ADDH<112))     {  LED_Ausgang=0x07;  }

if((ADDH>=112)&&(ADDH<144)     {  LED_Ausgang=0x0F;  }
```

usw.

```
/*-------------------------------------------------------------
Programmbeschreibung: Balkenanzeige
-------------------------------------------------------------
Name:           ADC_Balken.c
Funktion:       Am Portpin P1.3 = AN3 wird die anliegende Gleichspannung dauernd im
                Standard-Modus gemessen und das Ergebnis an Port P2 mit der LED-
                Kette in Form eines Leuchtbalkens angezeigt.
WARNUNG :
Bitte keine höheren Eingangsspannungen als max. +3V an die Analogeingänge anlegen!

Datum:          24.02.2008
Autor:          G. Kraus
-------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------*/
#include <t89c51ac2.h>       // Header für Controller "AT89C51AC3"
#include <stdio.h>           // Standard-Input-Output-Header

unsigned char channel;       // Deklaration der Kanal-Nummer
sfr LED_Ausgang=0xA0;        // Port P2 als LED-Ausgang

/*-------------------------------------------------------------
Prototypen
-------------------------------------------------------------*/
void wait_ADC_ini(void);     // Wartezeit nach der Wandlerfreigabe

/*-------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------*/

void main(void)
 {      AUXR=AUXR&0xFD; // auf internes ERAM umschalten = EXTRAM löschen
       channel=0x03;              // Kanal AN3 gewählt
        ADCF=0x08;                // Pin P1.3 auf „A-D-Channel“ umschalten
        ADCON=0x20;               // AD-Wandlung freigeben
        wait_ADC_ini();           // warten, bis ADC initialisiert
        ADCON=ADCON&0xF8;         // Alte Kanaleinstellung (SCH0...SCH2) löschen
        ADCON=ADCON|channel;      // Auf Channel AN3 schalten

        while(1)
        {    ADCON=ADCON|0x08;    // Wandlung durch Setzen von ADSST starten
             while((ADCON&0x10)!=0x10);   // Warten, bis am Ende ADEOC gesetzt wird
             ADCON=ADCON&0xEF;          // ADEOC wieder löschen

             if(ADDH<16)
             {    LED_Ausgang=0x00;   }

             if((ADDH>=16)&&(ADDH<48))
             {    LED_Ausgang=0x01;  }

             if((ADDH>=48)&&(ADDH<80))
             {    LED_Ausgang=0x03;  }

             if((ADDH>=80)&&(ADDH<112))
             {    LED_Ausgang=0x07;   }

             if((ADDH>=112)&&(ADDH<144))
             {    LED_Ausgang=0x0F;   }

             if((ADDH>=144)&&(ADDH<176))
             {    LED_Ausgang=0x1F;  }

             if((ADDH>=176)&&(ADDH<208))
             {    LED_Ausgang=0x3F;   }

             if((ADDH>=208)&&(ADDH<240))
             {    LED_Ausgang=0x7F;   }
```

```
            if(ADDH>=240)
            {     LED_Ausgang=0xFF;    }


        }
}

void wait_ADC_ini(void)              // Einschalt-Wartezeit beim Wandler nötig !
{           unsigned char x;
            for(x=0;x<5;x++);
}
```

7.5. Projekt: Digitalvoltmeter mit Displayanzeige

Aufgabe:

Port P1 wird an die Potentiometer-Kette auf der D-A-Platine über ein Flachbandkabel angeschlossen. Die Spannung an Portpin P1.3 soll im Standard-Modus gemessen und das Ergebnis auf dem Display angezeigt werden.

In der ersten Zeile soll **Spannung:**

in der zweiten Zeile dagegen z. B. **U = 1,25V**

erscheinen.

Lösung:

```
/*------------------------------------------------------------------
Programmbeschreibung:
------------------------------------------------------------------
Name:       ADC_02.c
Funktion:  Am Portpin P1.3 = AN3 wird die anliegende Gleichspannung dauernd im
           Standard-Modus gemessen.
           Zusätzlich ist an Port P2 das LCD-Display angeschlossen und zeigt das
           Ergebnis als Spannung im Bereich 0....+3V an.

WARNUNG 1:
Bitte keine höheren Eingangsspannungen als max. +3V an die Analogeingänge anlegen!

WARNUNG 2:
Bitte das LCD-Display IMMER an Port P2 betreiben und die Datei „LCD_Ctrl_ATMEL.c"
ins Projekt einbinden!

Datum:          24. 10. 2007
Autor:          G. Kraus

------------------------------------------------------------------
Deklarationen und Konstanten
------------------------------------------------------------------*/

#include <t89c51ac2.h>  // Header für Controller "AT89C51AC3"
#include <stdio.h>      // Standard-Eingabe-Ausgabe-Header

code unsigned char zeile_1[21]=  {"Spannung:           "};

unsigned char channel;
unsigned int x;
/*------------------------------------------------------------------
Prototypen
------------------------------------------------------------------*/
void wait_ADC_ini(void);

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);
```

```
/*---------------------------------------------------------------------
Hauptprogramm
---------------------------------------------------------------------*/
void main(void)

{  char zeile_2[21]={"U = 0,00V           "};   // Anzeige-Array
   AUXR=AUXR&0xFD;       // auf internes ERAM umschalten = EXTRAM löschen
   lcd_ein();
   lcd_ini1();
   switch_z1();
   show_text(zeile_1);

   channel=0x03;              // Kanal AN3 gewählt
   ADCF=0x08;                 // Pin P1.3 als A-D-Channel betreiben
   ADCON=0x20;                // AD Enable freigeben
   wait_ADC_ini();            // warten, bis ADC initialisiert

   while(1)
   {
        ADCON=ADCON&0xF8;          // SCH0...SCH2 löschen
        ADCON=ADCON|channel;       // Auf Channel AN3 schalten
        ADCON=ADCON|0x08;          // ADSST setzen, "Single Conversion" starten
        while((ADCON&0x10)!=0x10);      // Warten, bis ADEOC setzt
        ADCON=ADCON&0xEF;          // ADEOC wieder löschen


        x=ADDH;                    // Umkopieren in Variable

        zeile_2[7]=x*150*2/255 % 10 +0x30;   //Zweite Nachkommastelle berechnen
        zeile_2[6]=x*150/255/5 % 10 +0x30;   //Erste Nachkommastelle berechnen
        zeile_2[4]=x*150/255/50 % 10 + 0x30; //Stelle vor dem Komma berechnen

        switch_z2();                 // Auf Zeile 2 umschalten
        show_text(zeile_2);          // Array in Zeile 2 anzeigen

   }
}

void wait_ADC_ini(void)
{  unsigned char x;
   for(x=0;x<10;x++);
}
```

8. Interrupts

8.1. Grundlagen

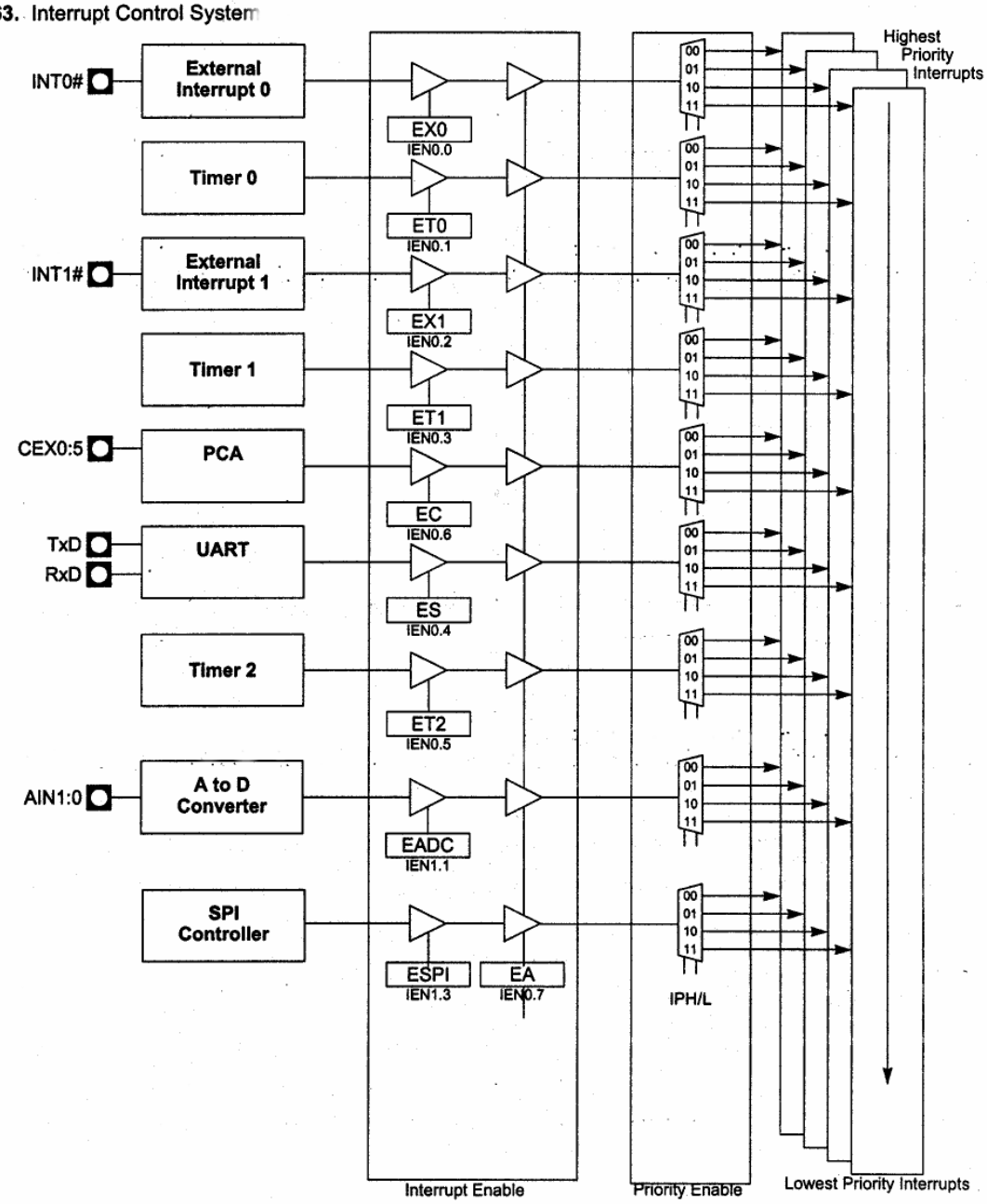
So äußert sich das Datenblatt:

Interrupt System

Introduction

The Micro-controller has a total of 9 interrupt vectors: two external interrupts ($\overline{INT0}$ and $\overline{INT1}$), three timer interrupts (timers 0, 1 and 2), a serial port interrupt, a PCA, a timer overrun interrupt and an ADC. These interrupts are shown below.

Figure 63. Interrupt Control System



Man erkennt daraus:

Es gibt 9 Interrupt-Möglichkeiten zur Unterbrechung des gerade laufenden Hauptprogramms:

Zwei Externe Interrupts (External Interrupt 0 und 1)

Drei Timer Interrupts (Timer 0 / Timer 1 / Timer 2)

Einen AD-Wandler-Interrupt

Einen UART – Interrupt (= Serielle Schnittstelle)

Einen PCA-Interrupt (= Programmable Counter Array)

Einen SPI- Interrupt (= Serial Programmable Interface)

a) Soll grundsätzlich mit Interrupts gearbeitet werden, dann muss der **„Hauptschalter EA = Enable all Interrupts“** eingeschaltet sein **(EA = 1).**

b) Dann wird der gewünschte Interrupt durch Setzen des entsprechenden **„Enable-Bits (EX0 bis ESPI)“** freigegeben. Diese Bits können direkt mit ihrer Bezeichnung im Programm verwendet und angesprochen werden.

c) Abschließend muss man noch den ausgewählten Interrupt auf eine der vier zur Verfügung stehenden **Prioritätsebenen** legen und damit den „Vorrang“ festlegen, wenn mehr als ein Interrupt eingesetzt wird.

Zu c):
Für die **Priorität** sind zwei Bits zuständig, die (leider!) aus zwei verschiedenen Registerpaaren

IPL0 / IPH0 /

oder

IPL1 / IPH1

zusammensuchen werden müssen (…9 Interrupts passen eben nicht in 1 Byte…).
Die Sache wird dann noch wilder, weil nur die beiden Register IPL0 (= 0xB8) und IPL1 (0xF8) Bit-adressierbar sind. Es gilt nämlich folgende Spielregel beim ATMEL AT89C51AC3 :

Nur Spezial-Funkionsregister (=SFR), deren Adressen auf „0“ oder „8“ enden, sind Bit-adressierbar.

Da außerdem im Keil-Header “t89c51ac2.h“ die möglichen Bit-Deklarationen für diese Register weggelassen wurden, muss man stets die gewünschten Bits über UND- bzw. ODER-Verknüpfungen und Neubeschreiben des kompletten Registers setzen oder löschen!

Table 63. Priority Level Bit Values

IPH.x	IPL.x	Interrupt Level Priority
0	0	0 (Lowest)
0	1	1
1	0	2
1	1	3 (Highest)

Und so sieht die Prioritätenliste schließlich aus.

Dann stellt sich noch die Frage: ***In welcher Reihenfolge werden Interrupts bearbeitet, die auf die gleiche Prioritätsebene gelegt wurden und zufälligerweise alle gleichzeitig ausgelöst werden?***

Da gelten diese Spielregeln.

Table 64. Interrupt priority Within level

Interrupt Name	Interrupt Address Vector	Priority Number
external interrupt (INT0)	0003h	1
Timer0 (TF0)	000Bh	2
external interrupt (INT1)	0013h	3
Timer1 (TF1)	001Bh	4
PCA (CF or CCFn)	0033h	5
UART (RI or TI)	0023h	6
Timer2 (TF2)	002Bh	7
ADC (ADCI)	0043h	8
SPI interrupt	0053h	9

Bitte genau hinschauen:

Diese Tabelle enthält auch die Interrupt Adress Vektoren, die wir für die C-Programm-Erstellung benötigen werden!

Beim Schreiben der Interrupt-Service-Routine (meist mit „ISR_.....“ bezeichnet!) müssen wir nämlich in verschlüsselter Form diese Adress-Vektoren übergeben!

==========

Reihenfolge zur Bestimmung des „Interrupt – Index“ bei C-Programmen:

Zuerst entnimmt man der Tabelle die Einsprungadresse für den vorgesehenen Interrupt:

Das ist beim Analog-Digital-Wandler die Vektoradresse 43h = 0x43

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

0x43 entspricht 4 x 16 + 3x1, also der Dezimalzahl „67“.

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.

Beim Aufruf der Interrupt-Service-Routine muss dann dieser Index in derselben Zeile angegeben werden.

Beispiel:

```
void ADC_Interrupt(void)        interrupt 8     // ADC-Interrupt Vector = 0x43 = 67
```

Alles klar?

8.3. Interrupt-Anwendungsbeispiel: Digitalvoltmeter mit Displayanzeige

Aufgabe:

Port P1 wird an die Potentiometer-Kette auf der D-A-Platine über ein Flachbandkabel angeschlossen. Die Spannung an Portpin P1.3 soll im Standard-Modus gemessen und der AD-Interrupt zum Einsatz kommen. Legen Sie diesen Interrupt auf die höchste Priorität.
Das Ergebnis soll in folgender Form auf dem Display angezeigt werden:

In der ersten Zeile steht: **Spannung:**

in der zweiten Zeile dagegen z. B. **U = 1,25V**

Lösung:
Diese Aufgabe entspricht (mit Ausnahme der Interrupt-Steuerung) exakt dem Beispiel 6.4. Deshalb können die meisten Programmteile von dort übernommen werden.

8.2.1. Lösung mit dem „normalen“ Header „t89c51ac2.h“

Hier kann die Priorität nur über Logische Verknüpfungen bei den beiden zuständigen Registern IPL1 und IPH1 festgelegt werden (..im Text fett markiert…):

```
/*-------------------------------------------------------------
Programmbeschreibung:
--------------------------------------------------------------
Name:         ADC_Interrupt.c
Funktion:     Am Portpin P1.0 = AN0) wird die anliegende Gleichspannung dauernd im
              Standard-Modus gemessen. Die Steuerung soll über den AD-Interrupt
              erfolgen.
              Zusätzlich ist an Port P2 noch das LCD-Display angeschlossen und zeigt
              das Ergebnis als
              Spannung im Bereich 0....+3V an.
WARNUNG 1:
Bitte keine höheren Eingangsspannungen als max. +3V
an die Analogeingänge anlegen!

WARNUNG 2:
Bitte das LCD-Display IMMER NUR an Port P2 betreiben und die Datei
„LCD_Ctrl_ATMEL.c“ ins Projekt einbinden!

Datum:         31. 10. 2007
Autor:         G. Kraus

--------------------------------------------------------------------
Deklarationen und Konstanten
--------------------------------------------------------------------*/

#include <t89c51AC2.h>  // Normaler Header für Controller "AT89C51AC3"
#include <stdio.h>      // Standard-Input_Output_Header

code unsigned char zeile_1[21]=  {"Spannung:           "};
/*-------------------------------------------------------------------
Prototypen
--------------------------------------------------------------------*/
void wait_ADC_ini(void);
void ADC_Interrupt(void);

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);


/*---------------------------------------------------------------------
Hauptprogramm
---------------------------------------------------------------------*/

void main(void)
{     unsigned char channel;
      AUXR=AUXR&0xFD;  // auf internes ERAM umschalten = EXTRAM löschen
      lcd_ein();
      lcd_ini1();
      switch_z1();
      show_text(zeile_1);

      EA=1;                    // Interrupt-Hauptschalter EIN
      EADC=1;                  // ADC-Interrupt freigeben

      IPL1=IPL1|0x02;          // ADC-Interrupt auf höchste
      IPH1=IPH1|0x02;          // Priorität legen

      channel=0x00;            // Kanal AN0 gewählt

      ADCF=0x01;               // Pin P1.0 als A-D-Channel betreiben
      ADCON=0x20;              // AD Enable freigeben
      wait_ADC_ini();          // warten, bis ADC initialisiert
      ADCON=ADCON&0xF8;        // SCH0...SCH2 löschen
      ADCON=ADCON|channel;     // Auf Channel AN0 schalten
      ADCON=ADCON|0x08;        // ADSST setzen, "Single Conversion" starten
      while(1);                // Endlos-Schleife
}

void wait_ADC_ini(void)        // Initialisierungs-Wartezeit erzeugen
{     unsigned char x;
      for(x=0;x<10;x++);
}

void ADC_Interrupt(void)    interrupt 8  // ADC-Interrupt Vector = 0x43 = 67
{     char zeile_2[21]={"U = 0,00V           "};
      unsigned int x;
      ADCON=ADCON&0xEF;        // ADEOC wieder löschen
      x=ADDH;                  // Umkopieren in Variable

      zeile_2[7]=x*150*2/255 % 10 +0x30;   //Zweite Nachkommastelle berechnen
      zeile_2[6]=x*150/255/5 % 10 +0x30;   //Erste Nachkommastelle berechnen
      zeile_2[4]=x*150/255/50 % 10 + 0x30; //Stelle vor dem Komma berechnen

      switch_z2();                 // Auf Zeile 2 umschalten
      show_text(zeile_2);          // Array in Zeile 2 anzeigen
      ADCON=ADCON|0x08;            // ADSST setzen, "Single Conversion" starten
}
```

8.2.2. Lösung mit dem neuen Header „AT89C51AC3.h“

Bei diesem neuen Header ist unter Anderem auch das zuständige Bit „PADCL“ in Register IPL1 extra als „bit-adressierbar“ aufgeführt und kann nun direkt angesprochen werden (…im Text fett markiert..).

```
/*-------------------------------------------------------------
Programmbeschreibung:
--------------------------------------------------------------
Name:           ADC_Interrupt.c
Funktion:       Am Portpin P1.0 = AN0) wird die anliegende Gleichspannung dauernd im
                Standard-Modus gemessen. Die Steuerung soll über den AD-Interrupt
                erfolgen.
                Zusätzlich ist an Port P2 noch das LCD-Display angeschlossen und
                zeigt das Ergebnis als
                Spannung im Bereich 0....+3V an.
WARNUNG 1:
Bitte keine höheren Eingangsspannungen als max. +3V
an die Analogeingänge anlegen!

WARNUNG 2:
Bitte das LCD-Display IMMER NUR an Port P2 betreiben und die Datei
„LCD_Ctrl_ATMEL.c“ ins Projekt einbinden!

Datum:          31. 10. 2007
Autor:          G. Kraus

--------------------------------------------------------------------
Deklarationen und Konstanten
--------------------------------------------------------------------*/

#include <AT89C51AC3.h>      // Neuer Header für Controller "AT89C51AC3"
#include <stdio.h>           // Standard-Input_Output_Header

code unsigned char zeile_1[21]=  {"Spannung:           "};
/*-------------------------------------------------------------------
Prototypen
--------------------------------------------------------------------*/
void wait_ADC_ini(void);
void ADC_Interrupt(void);

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);
/*-------------------------------------------------------------------
Hauptprogramm
--------------------------------------------------------------------*/

void main(void)
{    unsigned char channel;
     AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen
     lcd_ein();
     lcd_ini1();
     switch_z1();
     show_text(zeile_1);

     EA=1;              // Interrupt-Hauptschalter EIN
     EADC=1;            // ADC-Interrupt freigeben
```

```
      PADCL=1;                // ADC-Interrupt auf höchste
      IPH1=IPH1|0x02;         // Priorität legen

      channel=0x00;           // Kanal AN0 gewählt

      ADCF=0x01;              // Pin P1.0 als A-D-Channel betreiben
      ADCON=0x20;             // AD Enable freigeben
      wait_ADC_ini();         // warten, bis ADC initialisiert
      ADCON=ADCON&0xF8;       // SCH0...SCH2 löschen
      ADCON=ADCON|channel;    // Auf Channel AN0 schalten
      ADCON=ADCON|0x08;       // ADSST setzen, "Single Conversion" starten
      while(1);               // Endlos-Schleife
}

void wait_ADC_ini(void)           // Initialisierungs-Wartezeit erzeugen
{     unsigned char x;
      for(x=0;x<10;x++);
}

void ADC_Interrupt(void)    interrupt 8   // ADC-Interrupt Vector = 0x43 = 67
{     char zeile_2[21]={"U = 0,00V           "};
      unsigned int x;
      ADCON=ADCON&0xEF;           // ADEOC wieder löschen
      x=ADDH;                     // Umkopieren in Variable

      zeile_2[7]=x*150*2/255 % 10 +0x30;   //Zweite Nachkommastelle berechnen
      zeile_2[6]=x*150/255/5 % 10 +0x30;   //Erste Nachkommastelle berechnen
      zeile_2[4]=x*150/255/50 % 10 + 0x30; //Stelle vor dem Komma berechnen

      switch_z2();                // Auf Zeile 2 umschalten
      show_text(zeile_2);         // Array in Zeile 2 anzeigen
      ADCON=ADCON|0x08;           // ADSST setzen, "Single Conversion" starten
}
```

8.3. Digitalvoltmeter mit Display und „Drucktaste“ (Externer Interrupt)

Aufgabe:
Am Portpin P1.3 (= AN3) wird die anliegende Gleichspannung einmal gemessen, wenn eine Drucktaste an Portpin P3.2 geschlossen wird. Dadurch wird der externe Interrupt EX0 ausgelöst.
Er startet eine einzige Messung mit anschließender Ergebnisausgabe am LCD-Display (Messbereich 0...+3V).

Lösung:
Der durch den Tastendruck erzeugte „Externe Interrupt EX0“ startet eine einzige Messung. Anschließend wird gewartet, bis das Messergebnis vorliegt. Es wird (als Hex-Zahl) erst in eine Variable umkopiert und dann mit einem Dreisatz auf den Messbereich von 0…+3V umgerechnet. Nach einer weiteren Wandlung in die BCD-Form kann jede einzelne BCD-Ergebnisstelle schließlich als ASCII-Zeichen in einem Array abgelegt und zum Display geschickt werden.

```
/*-------------------------------------------------------------
Programmbeschreibung:
-------------------------------------------------------------
Name:      ADC_Drucktaste.c

WARNUNG 1:
Bitte keine höheren Eingangsspannungen als max. +3V an die Analogeingänge anlegen!

WARNUNG 2:
Bitte das LCD-Display IMMER NUR an Port P2 betreiben!

Datum:         26. 02. 2008
Autor:         G. Kraus

-------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------*/

#include <AT89C51AC3.h> // Neuer Header für Controller "AT89C51AC3"
#include <stdio.h>   // Standard-Input_Output_Header

code unsigned char zeile_1[21]=  {"Spannung:           "};
/*-------------------------------------------------------------
Prototypen
-------------------------------------------------------------*/
void wait_ADC_ini(void);
void ISR_INT0(void);

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);
```

```
/*---------------------------------------------------------------------
Hauptprogramm
---------------------------------------------------------------------*/

void main(void)
{     unsigned char channel;     // Gewählter AN-Eingang
      lcd_ein();                 // LCD-Initialisierung
      lcd_ini1();                // Einstellung der LCD-Betriebswerte
      switch_z1();               // Schalte auf Zeile 1
      show_text(zeile_1);        // Zeige „Spannung“ in Zeile 1 an

      EA=1;                      // Interrupt-Hauptschalter EIN
      EX0=1;                     // INT0 freigeben

      channel=0x03;              // Kanal AN3 gewählt
      ADCF=0x08;                 // Pin P1.3 als A-D-Channel betreiben
      ADCON=0x20;                // AD Enable freigeben
      wait_ADC_ini();            // warten, bis ADC initialisiert
      ADCON=ADCON&0xF8;          // SCH0...SCH2 löschen
      ADCON=ADCON|channel;       // Auf Channel AN3 schalten

      while(1);                  // Endlos-Schleife
}

void wait_ADC_ini(void)          // Initialisierungs-Wartezeit erzeugen
{     unsigned char x;
      for(x=0;x<10;x++);
}

void ISR_INT0(void)    interrupt 0   // INT0-Vector = 0x03
{     char zeile_2[21]={"U = 0,00V          "};
      unsigned int x;                // Temp-Variable für Ergebnis-Umrechnung

      ADCON=ADCON|0x08;              // ADSST setzen, "Single Conversion" starten
      while((ADCON&0x10)!=0x10);     // Warten, bis ADEOC setzt
      ADCON=ADCON&0xEF;              // ADEOC wieder löschen

      x=ADDH;                        // Umkopieren in Variable

      zeile_2[7]=x*150*2/255 % 10 +0x30;      // Zweite Nachkommastelle berechnen
      zeile_2[6]=x*150/255/5 % 10 +0x30;      // Erste Nachkommastelle berechnen
      zeile_2[4]=x*150/255/50 % 10 + 0x30;    // Stelle vor dem Komma berechnen

      switch_z2();                            // Auf Zeile 2 umschalten
      show_text(zeile_2);                     // Array in Zeile 2 anzeigen

}
```

9. Die verschiedenen Timer des Controllers

9.1. Überblick

Der Mikrocontroller enthält **3 verschiedene Timer**, nämlich: **Timer 0 Timer 1 Timer 2**

Von den "Vorfahren der 8051-Familie" (Typen 8031, 8032, 8051, usw.) wurden die beiden **Timer 0 und 1** direkt übernommen.
Jeder enthält ein 16 Bit - Register, das aus der Reihenschaltung von 2 Registern zu je 8 Bit zusammengesetzt ist.
Je nach Wunsch können diese Register als Frequenzzähler, als Ereigniszähler oder als Zeitgeber betrieben werden.

Vorsicht:
Der Timer 2 dagegen ist kompatibel mit dem Timer 2 des 80C52. Er besitzt ebenfalls zwei 8 Bit-Register wie Timer 0 und 1, aber man kann man hier z. B. einen 16 Bit – Autoreload-Timer realisieren. Deshalb erhält er ein eigenes Kapitel.

9.2. Arbeitsweise von Timer 0 und 1

Zur Einstellung der verschiedenen Betriebsarten dient das Register

"Timer modus" TMOD (Adresse 089H im internen RAM):

Bits:	Gate	C/#T	M1	M0	Gate	C/#T	M1	M0
	(Kontrolle für Timer 1)				(Kontrolle für Timer 0)			

Bedeutung: a) **Gate**
Wird hier eine **Null** hineingeschrieben, so lässt sich der entsprechende Zähler / Timer **nur** durch das passende **Timer-Run-Bit (TR0 oder TR1) im TCON - Register starten oder stoppen**.
Steht das Gate - Bit auf **Eins**, so läuft der Zähler nur los, wenn **TR-Bit und der zugehörige "externe Interrupt - Pin"** -- z. B #INT0 -- gleichzeitig auf HIGH liegen. (Damit lasse sich sehr einfach Impulsbreiten bestimmen).

b) **C / #T** (= Umschaltung zwischen Timer- und Zählerbetrieb)
Setzt man dieses Bit, so wird die Schaltung zum **Zähler.**
Löscht man dieses Bit, so arbeitet die Baugruppe als **Timer**. (Dem Eingang wird hierbei intern eine Taktfrequenz von 1 MHz zugeführt. Beim Überlauf wird das zugehörige Timer-Flag gesetzt).

c) **M0 und M1**
Mit diesen beiden Bits wird der "Modus" des Zählers gemäß folgender Tabelle eingestellt:

Mode	M1	M0	Wirkung
Null	0	0	**13 Bit-Timer**: das Low - Byte wird als 5-Bit-Vorteiler für das High – Byte benützt (Überrest aus uralten MCS-48-Zeiten.....)
Eins	0	1	**Normalbetrieb als 16 - Bit - Zähler/ Timer!!**
Zwei	1	0	**Reload - Betrieb** (= **8-Bit- Timer oder Zähler mit Selbstnachladung)**. Beim Überlauf wird der Inhalt des High - Bytes wieder als neuer "Anfangswert" ins Low - Byte geschrieben und von diesem Wert aus weitergezählt.
Drei	1	1	Recht komplizierte Wirkung: a) Schaltet man Timer 1 in diesen Modus, so bleibt er einfach stehen und speichert seinen alten Inhalt. b) Schaltet man Timer 0 in diesen Modus, so werden aus seinen beiden Bytes zwei von einander unabhängige 8-Bit-Timer.

Im **Timer – Control - Register TCON** finden sich in der **oberen Hälfte** diejenigen Bits, die

a) den **entsprechenden Timer starten bzw. stoppen,**

b) beim **Überlauf gesetzt** werden und damit zum Auslösen eines entsprechenden **Interrupts** für die Weiterverarbeitung des Zählerstandes verwendet werden können,

c) die **unteren 4 Bits** zur **Interrupt - Programmierung von #INT 0 und #INT1.**

TCON - Register, Adresse 088H im internen RAM, bitadressierbar

Bits:	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	

Erklärung:

a) **TF0 und TF1** sind die erwähnten **Überlauf-Flags.**

b) **TR0 und TR1** sind die "**Timer - Run - Bits**" zum **Starten / Stoppen** der Zähler/Timer-Baugruppen**.**

c) IE0 und IE1 heißen **"Interrupt - Edge-Flags**". Sie werden gesetzt, sobald die CPU eine **auslösende Impulsflanke** bei #INT0 bzw. #INT1 **erkannt** hat. Damit erzwingt man bei der nächsten Abfrage der Flags durch die CPU den Sprung in die Interrupt - Routine. Bei diesem Sprung werden sie dann wieder gelöscht.

d) **IT0 und IT1** sind die "**Interrupt - Type - Flags**".
Soll der Controller auf eine **negative** Impuls-***Flanke*** am Interrupt-Eingang reagieren, dann muss dieses Flag gesetzt werden. Steht das Flag auf LOW, dann löst das **Erreichendes LOW-Pegels am #INT0- bzw. #INT1-Pin** einen Interrupt aus.

9.3. Erzeugung eines 50 Hz-Tones mit Timer 1 (Interrupt-Steuerung)

Aufgabe:

Der Timer1 wird im 16 Bit-Normalbetrieb (Mode 1) zur Erzeugung eines 50 Hz-Tones am Portpin P4.0 verwendet. Dabei wird auch der Timer1 - Interrupt eingesetzt.

Lösung:
Man sorgt durch einen passenden Startwert von 55535 dafür, dass der Timer beim weiteren Hochzählen bis 65535 genau 10000 Zählschritte braucht. Das entspricht einer Zeit von 10000 Mikrosekunden = 10 Millisekunden und damit der halben Periodendauer einer 50 Hz-Schwingung. Der genaue Startwert während der Interrupt-Service-Routine in die beiden Register des Timers hineinkopiert und muss durch eine exakte Frequenzmessung bestimmt werden.

Achtung:
Der Ausgangswert für den genauen Frequenzabgleich ist die **Dezimalzahl 55535**. Das entspricht der Hex-Zahl **0xD8EF**, die auf die beiden Timerregister aufgeteilt werden muss.

```
/*-------------------------------------------------------------
Name:  tim1int_50_16bit.c
       3. 11. 2007 / G. Kraus
---------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------*/
#include <t89c51ac2.h>      // Verwendeter Header für AT89C51AC3
#include <stdio.h>          //  Standard-Eingabe-Ausgabe-Header

sbit ausgang=P4^0;          // Portpin P4.0 als Ausgang
unsigned char x;            // Zählvariable für ISR
void Timer_1_ISR(void);     // Prototyp für Interrupt-Service-Routine

/*-------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------*/
void main(void)
{  AUXR=AUXR&0xFD;       // auf internes ERAM umschalten = EXTRAM löschen
   x=0;
   TMOD=0x10;            // Timer1 im 16 Bit-Betrieb (Mode 1)
   TL1=0xEF;             // Reload-Wert ist 0xD8EF = 55535
   TH1=0xD8;             //
   EA=1;                 // Alle Interrupts freigeben
   ET1=1;                // Timer1 - Interrupt freigeben
   TF1=0;                // Timer1-Überlaufflag löschen
   TR1=1;                // Timer1 einschalten

   while(1);             // Endlosschleife
}
/*-------------------------------------------------------------
Interrupt-Service-Routine (ISR)
-------------------------------------------------------------*/
void Timer_1_ISR(void)  interrupt 3     // Interruptvektor = 0x1B = 27

{    TF1=0;                //Überlaufflag löschen
     ausgang=~ausgang;     // Ausgang komplementieren
     TL1=0xEF;             // Reload-Wert ist 0xD8EF = 65535
     TH1=0xD8;             //
}
```

9.4. Erzeugung eines 50 Hz-Tones mit Timer 1 im Reloadbetrieb und mit Interrupt-Steuerung

Aufgabe:

Der Timer1 wird im 8 Bit-Reloadbetrieb (Mode 2) zur Erzeugung eines 50 Hz-Tones am Portpin P4.0 verwendet. Dabei wird auch der Timer1 - Interrupt eingesetzt.

Lösung:
Man sorgt durch einen Reloadwert von 5 dafür, dass der Timer beim Hochzählen bis 255 genau 250 mal zählt und deshalb 250 Mikrosekunden braucht, um den Interrupt auszulösen. Dann wird der Reloadwert von 5 wieder neu nachgeladen.
In der ISR inkrementiert man dabei eine Variable bei jedem Überlauf und sorgt dafür, dass erst nach 40 Überläufen der Ausgang invertiert wird. Das entspricht einer Zeit von **40 x 0,25 Millisekunden = 10 Millisekunden** und damit der halben Periodendauer einer 50 Hz-Schwingung.

```
/*-------------------------------------------------------------
Name:         tim1int_100.c
3. 11. 2007 / G. Kraus
---------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------*/
#include <t89c51ac2.h>      // Verwendeter Header für AT89C51AC3
#include <stdio.h>          //  Standard-Eingabe-Ausgabe-Header

sbit ausgang=P4^0;          // Portpin P4.0 als Ausgang
unsigned char x;
void Timer_1_ISR(void);     // Prototyp für Interrupt-Service-Routine

/*-------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------*/
void main(void)
{  AUXR=AUXR&0xFD;       // auf internes ERAM umschalten = EXTRAM löschen
   x=0;
   TMOD=0x20;            // Timer1 im Reload-Betrieb (Mode 2)
   TL1=0x05;             // Start-Wert ist 5
   TH1=0x05;             // Reload-Wert ist 5
   EA=1;                 // Alle Interrupts freigeben
   ET1=1;                // Timer1 - Interrupt freigeben
   TF1=0;                // Timer1-Überlaufflag löschen
   TR1=1;                // Timer1 einschalten

   while(1);             // Endlosschleife
}
/*-------------------------------------------------------------
Interrupt-Service-Routine (ISR)
-------------------------------------------------------------*/
void Timer_1_ISR(void)  interrupt 3      // Interruptvektor = 0x1B

{    TF1=0;                              // Überlaufflag löschen
     x++;                                // Zählvariable inkrementieren
     if(x==40)                           // Schon 40 Durchgänge?
     {    x=0;                           // Zählvariable zurücksetzen
          ausgang = ~ausgang;            // Ausgang umdrehen
     }
}
```

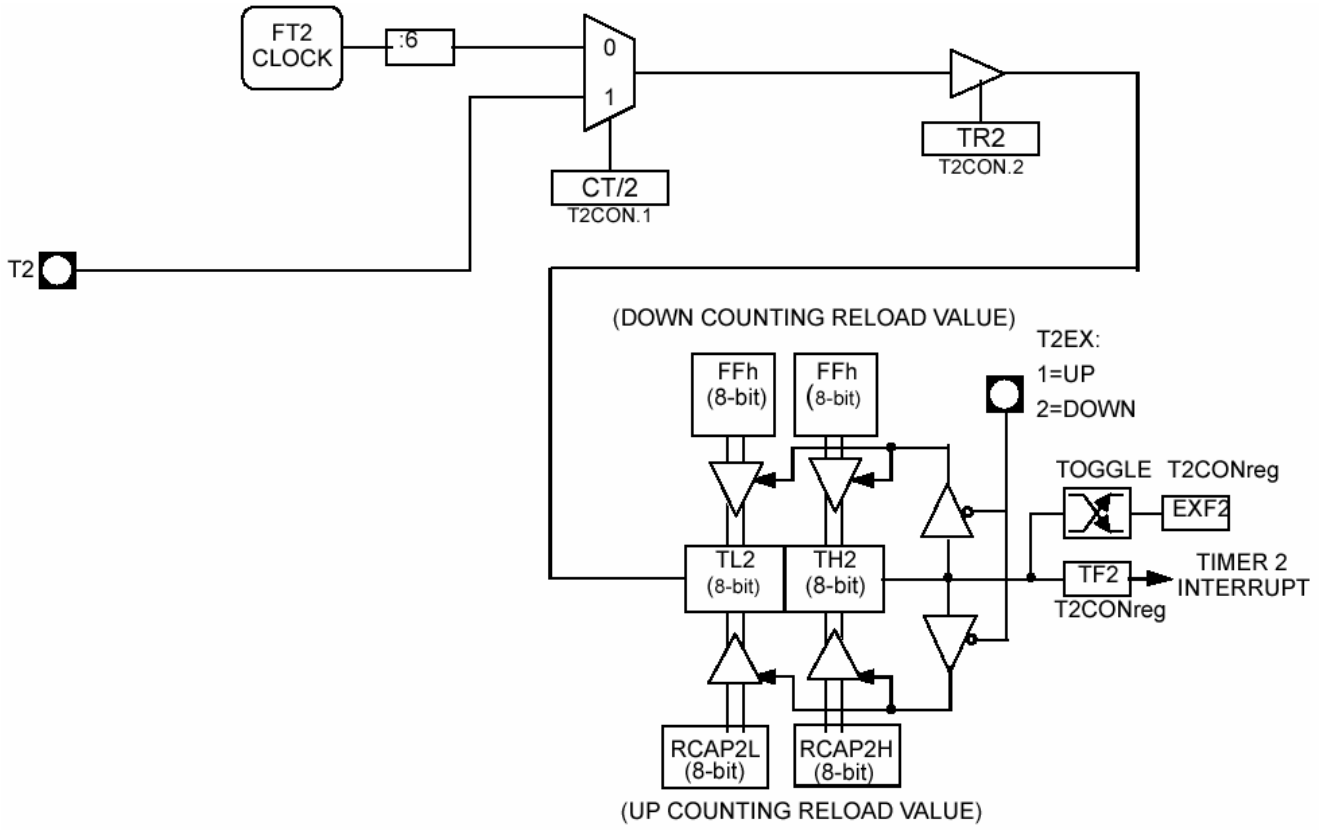
9.5. Der Timer 2

9.5.1. Ein kurzer Überblick

Hier handelt es sich um eine etwas aufwendigere Maschine, die bereits beim Familien-Vorgänger „80C52“ eingebaut war.
Timer 2 bildet einen **16-Bit-Zähler/Timer**, der auch im **16-Bit-Autoreload-Modus betrieben** werden kann (..und damit gegenüber dem 8-Bit-Autoreload-Modus von Timer 0 und Timer 1 viel mehr Möglichkeiten bietet!).
Eine weitere Anwendung ist der „**Programmable Clock Output**“. Er dient z. B. als Baudratengenerator für exotische Baudraten. Genauere Details zu dieser Anwendung (samt der Formel für die erzeugte Clockfrequenz):

Siehe ab Seite 74 des Datenblattes!

Für den Reload-Betrieb gilt folgender Übersichtsschaltplan:



Die Funktion ist leicht zu verstehen:

Es kann entweder der interne Quearztakt (**FT2-CLOCK**) oder ein externes zu zählendes Signal (**T2**) als Eingangssignal dienen.. Zwischen beiden Signalen kann mit dem Bit **C/T2#** umgeschaltet und damit **zwischen Counter- und Timerbetrieb gewählt** werden.
Das darauf folgende Bit **TR2 (= Timer Run Bit)** gibt dann im gesetzten Zustand den Weg zum 16-Bit – Zähler (TL2 und TH2) frei. Man erkennt sehr schön die beiden **Reload-Register RCAP2L und RCAP2H**, die den Reloadwert speichern. Allerdings gibt es noch eine Besonderheit:

Je nach dem Pegel am Controllerpin T2EX kann man aufwärts- oder abwärts zählen! Beim Aufwärtszählen erfolgt der Interrupt beim Überlauf, also Zählerstand 0xFFFF. Beim Abwärtszählen wird dagegen der Interrupt beim Unterschreiten des Reloadwertes ausgelöst, dadurch der Zähler wieder auf 0xFFFF zurückgesetzt und erneut mit dem Abwärtszählen begonnen.

TF2 bildet das ausgelöste Interrupt-Flag, während das Bit **EXF2 im T2CON-Register** zusätzlich „**getoggelt“** (= umgedreht) wird. Damit kann man theoretisch eine 17 Bit-Auflösung beim Zählen erreichen.

Wichtig:
Alle für die Programmerstellung wichtigen Bits finden sich in den beiden Registern T2CON und T2MOD (Siehe Datenblatt des Controllers). Dazu kommen noch die für den Interruptbetrieb nötigen Flags „EA“ und „ET2“.

9.5.2. Einfacher Warn-Blitz mit Timer 2

Aufgabe:

Mit dem Timer 2 soll jede Sekunde kurzer „Lichtblitz“ mit der Länge von 0,1 Sekunde an Portpin P1.0 erzeugt werden. Dabei wird auch der Timer 2 - Interrupt eingesetzt.

Lösung:
Man sorgt durch einen Reloadwert von 15535 dafür, dass der Timer bis 65535 genau 50000 mal hochzählt und deshalb 50 Millisekunden braucht, um den Interrupt auszulösen. Dann wird der Reloadwert von 15535 wieder neu nachgeladen.
In der ISR inkrementiert man dabei eine Variable bei jedem interrupt und sorgt dafür, dass erst nach 18 Überläufen wieder ein Blitz ausgelöst wird. Der Blitz selbst dauert 2 x 50 = 100 Millisekunden. Damit wiederholt sich der Zyklus erst nach 20 Durchläufen und das ergibt den zeitlichen Abstand von einer Sekunde zwischen den Blitzen.

```
/*-------------------------------------------------------------------
Programmbeschreibung
-------------------------------------------------------------------
Name:          blitz_01.c
Funktion:      Der im 16 Bit-Reload-Modus betriebene Timer 2
               erzeugt Sekundenimpulse an Portpin P4.0, die
               eine LED für 0,2 Sekunden zum Leuchten bringen.


Datum:         07. 11. 2007
Autor:         G. Kraus
-------------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------------*/
#include <t89c51ac2.h>
#include <stdio.h>

sbit ausgang=P1^0;         // LED-Ausgang

unsigned char x;           // Zählvariable
/*-------------------------------------------------------------------
Prototypen
-------------------------------------------------------------------*/
void ISR_Timer2(void);     //Interrupt-Service-Routine für Timer 2

/*-------------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------------*/

void main(void)
{  AUXR=AUXR&0xFD;     // auf internes ERAM umschalten = EXTRAM löschen
   P1=0x00;            // Alle LEDs an Port P1 löschen
   C_T2=0;             // Timer-Betrieb wählen
   RCLK=0;             // Timer 2 nicht für Serial Port verwenden
   TCLK=0;             //     "    "    "    " "
   CP_RL2=0;           // Autoreload bei Überlauf
   EA=1;               // Interrupt-Hauptschalter EIN
   ET2=1;              // Timer 2 - Interrupt EIN
   RCAP2L=0xAF;        // Reload-Wert ist 15535 (gibt 50 Millisekunden Zeit)
   RCAP2H=0x3C;        // (= 0x3CAF)
   TF2=0;              // Überlaufflag löschen

   x=0;                // Zählvariable auf Null setzen
   TR2=1;              // Timer 2 starten

   while(1);           // Endlosschleife
}
```

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

void ISR_Timer2(void)   interrupt 5      // Timer 2 hat Vector 0x2B
{    TF2=0;                              // Überlaufflag löschen
     x++;                                // Zählvariable inkrementieren

     if(x==18)                           // Nach 0,9 Sekunden:
     {    ausgang=1;                     // LED einschalten
     }

     if(x==20)                           // Nach 1 Sekunde:
     {    ausgang=0;                     // LED wieder ausschalten
          x=0;                           // Zählvariable zurücksetzen
     }

}
```

9.5.3. Einfacher Warn-Blitz mit Timer 2 und Start-Stopp-Steuerung

Aufgabe:

Mit dem Timer 2 soll jede Sekunde kurzer „Lichtblitz“ mit der Länge von 0,1 Sekunde an Portpin P1.0 erzeugt werden. Dabei wird auch der Timer 2 - Interrupt eingesetzt.
Es soll eine Start-Stopp-Steuerung mit Tasten an den Portpins P0.0 und P0.1 vorgesehen werden.

Lösung:
Man sorgt durch einen Reloadwert von 15535 dafür, dass der Timer beim Hochzählen bis 65535 genau 50000 mal zählt und deshalb 50 Millisekunden braucht, um den Interrupt auszulösen. Dann wird der Reloadwert von 15535 wieder neu nachgeladen.
In der ISR inkrementiert man dabei eine Variable bei jedem Überlauf und sorgt dafür, dass erst nach 18 Überläufen wieder ein Blitz ausgelöst wird. Der Blitz selbst dauert 2 x 50 = 100 Millisekunden. Damit wiederholt sich der Zyklus erst nach 20 Durchläufen und das ergibt den zeitlichen Abstand von einer Sekunde zwischen den Blitzen.
Drückt man nun die Starttaste an Portpin P0.0, dann wird ein Speicherbit („Merker“) gesetzt und das sorgt für die Blitzerei (= Timer wird gestartet). Drückt man dagegen die Stopp-Taste, dann wird über ein zweites Merker-Bit der Timer gestoppt und dieser Zustand gespeichert, bis wieder eingeschaltet wird.

```
/*-------------------------------------------------------------------
Programmbeschreibung
-------------------------------------------------------------------
Name:          blitz_02.c
Funktion:      Der im 16 Bit-Reload-Modus betriebene Timer 2
               erzeugt Sekundenimpulse an Portpin P1.0, die
               eine LED für 0,2 Sekunden zum Leuchten bringen.
               Dafür ist eine Starttaste an P0.0 und eine
               Stopptaste an P0.1 vorhanden.

Datum:         08. 11. 2007
Autor:         G. Kraus
-------------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------------*/
#include <t89c51ac2.h>
#include <stdio.h>

sbit starte=P0^0;          // Starten an P1.0
sbit stoppe=P0^1;          // Stoppen an P1.1

sbit ausgang=P1^0;         // LED-Ausgang

bit Blinken;               // Merker-Bits
bit Licht_aus;

unsigned char x;           // Globale Zählvariable
/*-------------------------------------------------------------------
Prototypen
-------------------------------------------------------------------*/
void ISR_Timer2(void);     //Interrupt-Service-Routine für Timer 2
void Tastenabfrage(void);
/*-------------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------------*/
void main(void)
{    AUXR=AUXR&0xFD;     // auf internes ERAM umschalten = EXTRAM löschen
     P1=0x00;       // Port P1 löschen
     C_T2=0;        // Timer-Betrieb wählen
     RCLK=0;        // Timer 2 nicht für Serial Port verwenden
     TCLK=0;        //     "    "    "    " "
     CP_RL2=0;      // Autoreload bei Überlauf
     EA=1;          // Interrupt-Hauptschalter EIN
```

```
     ET2=1;           // Timer 2 - Interrupt EIN
     RCAP2L=0xAF;     // Reload-Wert ist 15535 (gibt 50 Millisekunden Zeit)
     RCAP2H=0x3C;     // (= 0x3CAF)
     TF2=0;           // Überlaufflag löschen

     x=0;             // Zählvariable auf Null setzen

     Licht_aus=1;     // passendes Bit für "AUS" setzen
     TR2=0;           // Timer 2 stoppen


     while(1)                       // Endlosschleife
     {    while(Licht_aus==1)       // AUS-Bit ist gesetzt
          {    TR2=0;               // Zähler steht still
               Tastenabfrage();     // Wird eine Taste gedrückt?
          }

          while(Blinken==1)         // BLINK-Bit ist gesetzt
          {    TR2=1;               // Timer 2 starten
               Tastenabfrage();     // Wird eine Taste gedrückt?
          }
     }
}

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

void ISR_Timer2(void)   interrupt 5      // Timer 2 hat Interrupt-Index 5
{    TF2=0;                              // Überlaufflag löschen
     x++;                                // Überläufe zählen

     if(x==18)                           // 18 Überläufe = 0,9 Sekunden
     {    ausgang=1;                     // LED einschalten
     }

     if(x==20)                           // 20 Überläufe = 1 Sekunde
     {    ausgang=0;                     // LED wieder ausschalten
          x=0;                           // Zählvariable zurücksetzen
     }

}


void Tastenabfrage(void)                 // Tastenabfrage
{    if(starte==0)                       // „Blinken" gedrückt?
     {    Blinken=1;                     // Wenn ja: Blinkerbit setzen
          Licht_aus=0;                   // AUS-Bit löschen
     }

     if(stoppe==0)                       // „Stopp!" gedrückt?
     {    Blinken=0;                     // Wenn ja: Blinkerbit löschen
          Licht_aus=1;                   // AUS-Bit setzen

     }
}
```

9.5.4. Baustellen-Blitz mit Timer 2 und Start-Stopp-Steuerung

Aufgabe:

An Port P1 wird die LED-Kette angeschlossen. Mit dem Timer 2 soll ein kurzer „Lichtblitz“ mit der Länge von 0,1 Sekunde erzeugt werden, der nun in 800 Millisekunden „von Unten nach Oben“ in dieser LED-Kette hoch läuft.
Dann ist eine Sekunde Pause, bevor der nächste „Kettenblitz“ beginnt. Dazu wird der Timer 2 - Interrupt eingesetzt.
Es soll gleichzeitig eine Start-Stopp-Steuerung mit Tasten an den Portpins P0.0 und P0.1 vorgesehen werden.

Lösung:
Man sorgt durch einen Reloadwert von 15535 dafür, dass der Timer beim Hochzählen bis 65535 genau 50000 mal zählt und deshalb stets 50 Millisekunden braucht, um den nächsten Interrupt auszulösen. Dann wird der Reloadwert von 15535 wieder neu nachgeladen.
In der ISR inkrementiert man dabei eine Variable und steuert mit ihr die Blitzfolge. Jeder Blitz dauert 2 x 50 = 100 Millisekunden.
Drückt man nun die Starttaste an Portpin P0.0, dann wird ein Speicherbit („Merker“) gesetzt und das sorgt für die Blitzerei (= Timer wird gestartet). Drückt man dagegen die Stopp-Taste, dann wird über ein zweites Merker-Bit der Timer gestoppt und dieser Zustand gespeichert, bis wieder eingeschaltet wird.

```
/*-------------------------------------------------------------
Programmbeschreibung
-------------------------------------------------------------
Name:          blitz_02.c
Funktion:      An Port P1 wird die LED-Kette angeschlossen. Mit dem
               Timer 2 soll ein kurzer "Lichtblitz" mit der Länge von 0,1
               Sekunde erzeugt werden, der nun in 800 Millisekunden "von
               Unten nach Oben" in dieser LED-Kette hoch läuft.
               Dann ist eine Sekunde Pause, bevor der nächste
               "Kettenblitz" beginnt. Dazu wird der Timer 2 - Interrupt
               eingesetzt.
               Es soll gleichzeitig eine Start-Stopp-Steuerung
               mit Tasten an den Portpins P0.0 und P0.1 vorgesehen werden.


Datum:         08. 11. 2007
Autor:         G. Kraus
-------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------*/
#include <t89c51ac2.h>
#include <stdio.h>

sbit starte=P0^0;          // Start-Taste an P1.0
sbit stoppe=P0^1;          // Stopp-Taste an P1.1

sfr ausgang=0x90;          // Port P1 als LED-Ausgang

bit Blinken;               // Merker-Deklarationen
bit Licht_aus;

unsigned char x;           // Zählvariable

/*-------------------------------------------------------------
Prototypen
-------------------------------------------------------------*/

void ISR_Timer2(void);       // Interrupt-Service-Routine für Timer 2
void Tastenabfrage(void);    // Zählvariable
```

```
/*---------------------------------------------------------------------
Hauptprogramm
---------------------------------------------------------------------*/

void main(void)
{       AUXR=AUXR&0xFD;  // auf internes ERAM umschalten = EXTRAM löschen
        P1=0x00;         // Port P1 löschen
        C_T2=0;          // Timer-Betrieb
        RCLK=0;          // Timer 2 nicht für Serial Port verwenden
        TCLK=0;          //      "    "    "    "  "
        CP_RL2=0;        // Autoreload bei Überlauf
        EA=1;            // Interrupt-Hauptschalter EIN
        ET2=1;           // Timer 2 - Interrupt EIN
        RCAP2L=0xAF;     // Reload-Wert 15535 gibt 50 Millisekunden
        RCAP2H=0x3C;     // (= 0x3CAF)
        TF2=0;           // Überlaufflag löschen

        x=0;             // Zählvariable auf Null setzen

        Licht_aus=1;     // passendes Bit für "AUS" setzen
        TR2=0;           // Timer 2 stoppen


        while(1)                        // Endlosschleife
        {       while(Licht_aus==1)     //  AUS-Zustand
                {       TR2=0;          // Zähler steht still
                        Tastenabfrage();//  Tasten abfragen
                }

                while(Blinken==1)       // Blitzen!
                {       TR2=1;          // Timer 2 starten
                        Tastenabfrage();// Tasten abfragen
                }
        }
}

/*---------------------------------------------------------------------
Zusatzfunktionen
---------------------------------------------------------------------*/
void Tastenabfrage(void)
{       if(starte==0)               // Start-Taste gedrückt?
        {       Blinken=1;          // Wenn Ja: Blinker-Bit setzen
                Licht_aus=0;        // AUS-Bit löschen
        }

        if(stoppe==0)               // Stopp-Taste gedrückt?
        {       Blinken=0;          // Wenn Ja: Blinker-Bit löschen
                Licht_aus=1;        // AUS-Bit setzen
        }
}


void ISR_Timer2(void)   interrupt 5   // Timer 2 hat Vector 0x2B
{  TF2=0;                             // Überlaufflag löschen
   x++;                               // Zähler inkrementieren

   if(x==1)                           // Portpin P1.0 wird gesetzt
   {  ausgang=0x01;                   // Alle anderen Pins auf LOW
   }
   if(x==3)                           // Portpin P1.1 wird gesetzt
   {  ausgang=0x02;                   // Alle anderen Pins auf LOW
   }
```

```
   if(x==5)                          // Portpin P1.2 wird gesetzt
   {    ausgang=0x04;                // Alle anderen Pins auf LOW
   }
   if(x==7)                          // Portpin P1.3 wird gesetzt
   {    ausgang=0x08;                // Alle anderen Pins auf LOW
   }
   if(x==9)                          // Portpin P1.4 wird gesetzt
   {    ausgang=0x10;                // Alle anderen Pins auf LOW
   }
   if(x==11)                         // Portpin P1.5 wird gesetzt
   {    ausgang=0x20;                // Alle anderen Pins auf LOW
   }
   if(x==13)                         // Portpin P1.6 wird gesetzt
   {    ausgang=0x40;                // Alle anderen Pins auf LOW
   }
   if(x==15)                         // Portpin P1.7 wird gesetzt
   {    ausgang=0x80;                // Alle anderen Pins auf LOW
   }
   if(x==17)                         // Alle Pins auf LOW
   {    ausgang=0x00;
   }
   if(x==36)                         // Alle Pins für 1 Sekunde auf LOW
   {    x=0;
   }
}
```

10. Unterrichtsprojekt: Vom Blinker zur Stoppuhr mit LCD-Display = Timer 2 im Reloadbetrieb

10.1. Vorbemerkung

Der Timer 2 bietet bekanntlich die Möglichkeit, im 16 Bit – Reload-Modus Zeitverzögerungen bis 65 535 Mikrosekunden (bei Verwendung eines 12MHz-Quarzes) zu produzieren. Zusammen mit einer Interruptsteuerung wollen wir uns hier von einer einfachen Blinkschaltung bis zu einer Stoppuhr mit LCD-Anzeige und einer Auflösung von 0,1 Sekunden hocharbeiten.
Voraussetzung hierzu ist die Kenntnis der entsprechenden Grundlagen aus Kapitel 8.5.: „Der Timer 2“.

==========

Achtung:
Die in den Beispielprogrammen benützten Reloadwerte beim Timer 2 gelten für einen Quarztakt von 12 MHz, denn nur damit wird im Rhythmus von einer Mikrosekunde hochgezählt.

Wird dagegen der „Schnittstellenquarz mit 11,0592 MHz“ eingesetzt, dann ist ist die Periodendauer des Taktes um den Faktor

12MHz / 11,0592MHz = 1,08506944

länger. Dann muss folgende Rechnung aufgemacht werden:

a) Das Hochzählen von 0 bis 65535 dauert nun 65535 x 1,08507µs = 71110µs

b) Will man z. B. auf eine Zählzeit von 50ms = 50000µs kommen, dann müssen (71110 – 50000)µs = 21110µs als Reloadwert vorgesehen werden. 21110µs entsprechen aber einem Reloadwert von

21110 / 1,08507 = 19455

Das ergibt eine Hex-Zahl von

0x4BFF

die in die beiden Timer-Register beim Überlauf hineinkopiert werden muß.

==========

10.2. Der Einstieg: Blinken im Sekundentakt

Grundgedanke:
Der Reloadwert wird so festgelegt, dass der Timer immer nach exakt **50 Millisekunden überläuft**. In der Interrupt-Service-Routine wird dann eine Variable hochgezählt und nach 10 Durchgängen = 500 Millisekunden ein Portpin invertiert. Damit erreicht man eine Blinkfrequenz von 1 Hz. An diesen Portpin ist eine LED angeschlossen.

Lösung:

Der erforderliche **Reloadwert** für einen Quarztakt von 12 MHz ist **65535 – 50000 = 15535**.
Das entspricht einer **Hexzahl von 0x3CAF**

Verwendet man dagegen einen **Quarz mit 11,0592 MHz**, dann ergibt sich ein Reloadwert von (Siehe oben!) **0x4BFF**

Die LED **an Portpin P1.0** soll **„Blinker“** heißen.

Bei **einem Interrupt-Vektor von 0x2B (= 43)** für den Timer 2 muß in der Interrupt-Service-Routine **der „Interrupt 5“** verwendet werden.

```
/*-------------------------------------------------------------
Programmbeschreibung
--------------------------------------------------------------
Name:       stoppuhr_01.c
Funktion:   Mit dem im 16 Bit-Reload-Modus betriebenen Timer 2 wird eine
            LED an P1.0 im Sekundenrhythmus zum Blinken gebracht (ergibt
            ein symmetrisches Rechtecksignal mit der Frequenz von 1Hz).
            Timer 2 läuft dabei im 50 Millisekunden-Rhythmus über.

Datum:      27. 11. 2007
Autor:      G. Kraus
--------------------------------------------------------------------
Deklarationen und Konstanten
--------------------------------------------------------------------*/

#include <t89c51ac2.h>          // Header für AT89C51AC3
#include <stdio.h>              // Standard-Input-Output-Header verwenden

sbit blinker=P1^0;              // Blink-LED an P1.0
unsigned char y,x;              // Zählvariablen für lange Zeiten
/*-------------------------------------------------------------------
Prototypen
--------------------------------------------------------------------*/

void ISR_Timer2(void);
/*-------------------------------------------------------------------
Hauptprogramm
--------------------------------------------------------------------*/

void main(void)
{    AUXR=AUXR&0xFD;        // auf internes ERAM umschalten = EXTRAM löschen
     P1=0x00;               // Port P1 löschen
     C_T2=0;                // Timer-Betrieb wählen
     RCLK=0;                // Timer 2 nicht für Serial Port verwenden
     TCLK=0;                //     "    "    "    "  "
     CP_RL2=0;              // Autoreload bei Überlauf wählen
     EA=1;                  // Interrupt-Hauptschalter EIN
     ET2=1;                 // Timer 2 - Interrupt EIN
     RCAP2L=0xAF;     // Reload-Wert ist 15535 (gibt 50 Millisekunden Zeit)
     RCAP2H=0x3C;     // (= 0x3CAF)

   /* Bei 11,0592 MHz wäre der Reloadwert 19455 = 0x4BFF) */
```

```
     TF2=0;                    // Überlaufflag löschen

     x=0;                      // Zählvariablen auf Null setzen
     y=0;

     TR2=1;                    // Timer 2 starten

     while(1);                 // Endlosschleife
}

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

void ISR_Timer2(void)   interrupt 5   // Timer 2 hat Interrupt-Index 5
{    TF2=0;                           // Überlaufflag löschen
     x++;                             // Überläufe zählen

     if(x>=2)                         // Zwei Überläufe erreicht?
     {    x=0;                        // dann x löschen
          y++;                        // und Zehntelsekunde zählen
     }

     if(y>=5)                       // 500 Millisekunden erreicht?
     {    blinker=~blinker;         // Dann den Blinkausgang invertieren
          y=0;               // und den Zehntelsekundenzähler zurücksetzen
     }
}
```

10.3. Blinken beim Drücken eines Klingelknopfes

Schließen Sie nun an den Portpin P0.0 eine LOW-aktive Drucktaste an. Nur wenn **diese Taste gedrückt** ist, darf die LED blinken. Erweitern Sie Ihr Programm entsprechend.

Lösung:

```
/*------------------------------------------------------------------
Programmbeschreibung
------------------------------------------------------------------
Name:            stoppuhr_02.c
Funktion:        Mit dem im 16 Bit-Reload-Modus betriebenen Timer 2
                 wird eine LED an P1.0 im Sekundenrhythmus zum
                 Blinken gebracht (ergibt ein symmetrisches
                 Rechtecksignal mit der Frequenz von 1Hz).
                 Timer 2 läuft dabei im 50 Millisekunden-Rhythmus über.
                 Das geschieht, solange die Taste P0.0 gedrückt wird.

Datum:           29. 11. 2007
Autor:           G. Kraus

------------------------------------------------------------------
Deklarationen und Konstanten
------------------------------------------------------------------*/

#include <t89c51ac2.h>      // Header für AT89C51AC3
#include <stdio.h>          // Standard-Input-Output-Header verwenden

sbit blinker=P1^0;          // Blinker-LED an P1.0
sbit starte=P0^0;           // Starttaste an P0.0

unsigned char y,x;          // Zählvariablen für lange Zeiten

/*------------------------------------------------------------------
Prototypen
------------------------------------------------------------------*/

void ISR_Timer2(void);

/*------------------------------------------------------------------
Hauptprogramm
------------------------------------------------------------------*/

void main(void)
{    AUXR=AUXR&0xFD;       // auf internes ERAM umschalten = EXTRAM löschen
     P1=0x00;              // Port P1 komplett löschen
     P0=0xFF;              // Port P0 erst auf „Einlesen" schalten!
     C_T2=0;               // Timer-Betrieb wählen
     RCLK=0;               // Timer 2 nicht für Serial Port verwenden
     TCLK=0;               //     "    "    "    "  "
     CP_RL2=0;             // Autoreload bei Überlauf wählen
     EA=1;                 // Interrupt-Hauptschalter EIN
     ET2=0;                // Timer 2 - Interrupt AUS
     RCAP2L=0xFF;          // Bei 11,0592 MHz Quarztakt beträgt der
     RCAP2H=0x4B;          // Reload-Wert 19455 = 0x4BFF
                           // (das gibt 50 Millisekunden Zeit)

  /* Bei 12 MHz wäre der Reloadwert 15535 = 0x3CAF) */

     TF2=0;                // Überlaufflag löschen

     x=0;                  // Zählvariablen auf Null setzen
     y=0;
```

```
    TR2=1;                // Timer 2 starten

    while(1)
    {       while(starte==0)    // Wenn Starttaste gedrückt:
            {    ET2=1;         // lasse blinken
            }

            ET2=0;              // Blinker aus und
            P1=0;               // alle LEDs gelöscht

    }
}

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

void ISR_Timer2(void) interrupt 5    // Timer 2 hat Interrupt-Index 5
{    TF2=0;                          // Überlaufflag löschen
     x++;                            // Überläufe zählen

     if(x>=2)                        // Zwei Überläufe erreicht?
     {    x=0;                       // dann x löschen
          y++;                       // und Zehntelsekunde zählen
     }

     if(y>=5)                        // 500 Millisekunden erreicht?
     {    blinker=~blinker;          // Dann den Blinkausgang invertieren
          y=0;              // und den Zehntelsekundenzähler zurücksetzen
     }
}
```

10.4. Blinkersteuerung über Start- und Stopptaste

```
/*------------------------------------------------------------------
Programmbeschreibung
------------------------------------------------------------------
Name:           stoppuhr_03.c

Funktion:       Mit dem im 16 Bit-Reload-Modus betriebenen Timer 2
                wird eine LED an P1.0 im Sekundenrhythmus zum
                Blinken gebracht (ergibt ein symmetrisches
                Rechtecksignal mit der Frequenz von 1Hz).
                Timer 2 läuft dabei im 50 Millisekunden-Rhythmus über.
                Dafür ist eine Starttaste an P0.0 und eine Stopptaste
                an P0.1 vorhanden.

Datum:          29. 11. 2007
Autor:          G. Kraus

------------------------------------------------------------------
Deklarationen und Konstanten
------------------------------------------------------------------*/

#include <t89c51ac2.h>    // Header für AT89C51AC3
#include <stdio.h>        // Standard-Input-Output-Header verwenden

sbit blinker=P1^0;        // Blinker-LED an P1.0
sbit starte=P0^0;         // Starttaste an P0.0
sbit stoppe=P0^1;         // Stopptaste an P0.1

unsigned char y,x;        // Zählvariablen für lange Zeiten
/*------------------------------------------------------------------
Prototypen
------------------------------------------------------------------*/

void ISR_Timer2(void);
/*------------------------------------------------------------------
Hauptprogramm
------------------------------------------------------------------*/

void main(void)
{    AUXR=AUXR&0xFD;      // auf internes ERAM umschalten = EXTRAM löschen
     P1=0x00;             // Port P1 komplett löschen
     P0=0xFF;             // Port P0 erst auf „Einlesen“ schalten!
     C_T2=0;              // Timer-Betrieb wählen
     RCLK=0;              // Timer 2 nicht für Serial Port verwenden
     TCLK=0;              //     "    "    "    " "
     CP_RL2=0;            // Autoreload bei Überlauf wählen
     EA=1;                // Interrupt-Hauptschalter EIN
     ET2=0;               // Timer 2 - Interrupt AUS
     RCAP2L=0xFF;         // Bei 11,0592 MHz Quarztakt beträgt der
     RCAP2H=0x4B;         // Reload-Wert 19455 = 0x4BFF
                          // (das gibt 50 Millisekunden Zeit)

  /* Bei 12 MHz wäre der Reloadwert 15535 = 0x3CAF) */

     TF2=0;               // Überlaufflag löschen

     x=0;                 // Zählvariablen auf Null setzen
     y=0;

     TR2=1;               // Timer 2 starten
```

```
     while(1)                  // Endlosschleife
     {    if(starte==0)        // Starttaste gedrückt?
          {
               ET2=1;         // dann lasse blinken
          }

          if(stoppe==0)        // Stopptaste gedrückt?
          {
               ET2=0;         // Timer 2 stoppen und
               P1=0;          // alle LEDs an Port P1 löschen
          }
     }
}

/*---------------------------------------------------------------------
zusatzfunktionen
---------------------------------------------------------------------*/

void ISR_Timer2(void) interrupt 5    // Timer 2 hat Interrupt-Index 5
{    TF2=0;                          // Überlaufflag löschen
     x++;                            // Überläufe zählen

     if(x>=2)                        // Zwei Überläufe erreicht?
     {    x=0;                       // dann x löschen
          y++;                       // und Zehntelsekunde zählen
     }

     if(y>=5)                        // 500 Millisekunden erreicht?
     {    blinker=~blinker;          // Dann den Blinkausgang invertieren
          y=0;                  // und den Zehntelsekundenzähler zurücksetzen
     }
}
```

10.5. Jetzt mit Display: Zählen im Sekundentakt und Anzeige der Sekunden

Erzeugen Sie mit dem Reload-Timer echte Sekundenimpulse und zählen Sie diese Impulse. Zeigen Sie diese Sekunden auf dem LCD-Display an.

In der ersten Zeile soll dabei ein hinweisender Text (hier: „**Timer 2 – Uebung**“, in der zweiten Zeile dagegen

„xxx Sekunden“

erscheinen.

Verwenden Sie wieder eine **Start- und Stopptaste an Portpin P0.0 und P0.1**

Schließen Sie das **Display an Port P2** an.

Verbinden Sie **die LED-Anzeige mit Port P**1

```
/*------------------------------------------------------------
Programmbeschreibung
------------------------------------------------------------
Name:           stoppuhr_04.c

Funktion:       Der im 16 Bit-Reload-Modus mit Interrupt betriebene
                Timer 2 erzeugt Sekundenimpulse. Diese werden gezählt
                und auf dem LCD-Display an Port P2 zur Anzeige gebracht.
                (Zählzeit = 0.....59 Sekunden)
                Dafür ist eine Starttaste an P0.0 und eine
                Stopptaste an P.1 vorhanden.
                Eine LED an Portpin P1.0 wird jede Sekunde invertiert.

                Achtung:
                Nun muß das File "LCD_Ctrl_ATMEL.c" mit in das Projekt
                eingebunden werden!

Datum:          29. 11. 2007
Autor:          G. Kraus

------------------------------------------------------------------
Deklarationen und Konstanten
------------------------------------------------------------------*/

#include <t89c51ac2.h>          // Header für AT89C51AC3
#include <stdio.h>              // Standard-Input-Output-Header verwenden

sbit starte=P0^0;               // Starten an P0.0
sbit stoppe=P0^1;               // Stoppen an P0.1
sbit LED=P1^0;                  // LED an Pin P1.0

unsigned char overflow;         //Overflow-Zähler
unsigned char Zehntel;          //Zehntelsekunden-Zähler
unsigned char Sekunden;         //Sekunden-Zähler

code unsigned char zeile_1[21]={"Timer 2 - Uebung    "};
code unsigned char clear[21]={"000 Sekunden!       "};
/*------------------------------------------------------------------
Prototypen
------------------------------------------------------------------*/

void ISR_Timer2(void);          // Prototyp der Interrupt Service Routine

void zeit_s(void);              // LCD-Display-Prototypen
```

```
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);

/*-------------------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------------------*/

void main(void)
{    AUXR=AUXR&0xFD;       // auf internes ERAM umschalten = EXTRAM löschen
     P1=0x00;              // Port P1 komplett löschen
     P0=0xFF;              // Port P0 erst auf „Einlesen“ schalten!
     C_T2=0;               // Timer-Betrieb wählen
     RCLK=0;               // Timer 2 nicht für Serial Port verwenden
     TCLK=0;               //     "    "    "    " "
     CP_RL2=0;             // Autoreload bei Überlauf wählen
     EA=1;                 // Interrupt-Hauptschalter EIN
     ET2=0;                // Timer 2 - Interrupt AUS
     RCAP2L=0xFF;          // Bei 11,0592 MHz Quarztakt beträgt der
     RCAP2H=0x4B;          // Reload-Wert 19455 = 0x4BFF
                           // (das gibt 50 Millisekunden Zeit)

     /* Bei 12 MHz wäre der Reloadwert 15535 = 0x3CAF) */

     lcd_ein();            // Display-Einschaltroutine
     lcd_ini1();           // Display-Initialisierung
     switch_z1();          // Schalte auf Zeile 1
     show_text(zeile_1);   // Zeige Starttext an

     TF2=0;                // Überlaufflag löschen

     overflow=0;;          // Overflow-Zähler
     Zehntel=0;;           // Zehntelsekunden-Zähler
     Sekunden=0;;          // Sekunden-Zähler
     LED=0;                // LED ausschalten

     TR2=1;                // Timer 2 starten

     while(1)              // Endlosschleife
     {    if(starte==0)           // Starttaste gedrückt?
          {    overflow=0;        // Overflow-Flag zurücksetzen
               Zehntel=0;         // Zehntelsekunden zurücksetzen
               Sekunden=0;        // Sekunden zurücksetzen
               switch_z1();       // Schalte Display auf Zeile 1
               show_text(clear);  // Zeige "000 Sekunden!"
               ET2=1;             // dann lasse blinken
          }

          if(stoppe==0)           // Stopptaste gedrückt?
          {
               ET2=0;             // dann Timer 2 - Interrupt stoppen
          }
   }
}
```

```
/*------------------------------------------------------------------------
zusatzfunktionen
------------------------------------------------------------------------*/

void ISR_Timer2(void)   interrupt 5   // Timer 2 hat Interrupt-Index 5
{    unsigned char zeit[21]={"000 Sekunden!       "};

     TF2=0;                          // Überlaufflag löschen
     overflow++;                     // Überlaufzähler inkrementieren

     if(overflow>=2)                 // Zweimal gezählt?
     {    overflow=0;
          Zehntel++;                 // Dann die Zehntelsekunden zählen
     }

     if(Zehntel>=10)                 // 10 x 100 = 1000 Millisekunden vorbei?
     {    Zehntel=0;                 // Zehntelzähler wieder auf Null setzen
          Sekunden++;                // Sekunden zählen
          if(Sekunden==60)
          {    overflow=0;
               Zehntel=0;
               Sekunden=0;
          }

          zeit[2]=Sekunden %10 +0x30;    // Sekunden für Display berechnen
          zeit[1]=(Sekunden/10) %10 +0x30;
          zeit[0]=(Sekunden/100) %10 +0x30;
          switch_z1();                        // Auf Zeile 1 umschalten
          show_text(zeit);                    // Zeit anzeigen
          LED=~LED;                           // LED invertieren

     }
}
```

10.6. Nochmals derselbe Sekundenzähler samt Display, aber nun zusätzlich mit einer Löschtaste

```
/*-------------------------------------------------------------------
Programmbeschreibung
-------------------------------------------------------------------
Name:         stoppuhr_05.c

Funktion:     Der im 16 Bit-Reload-Modus mit Interrupt betriebene
              Timer 2 erzeugt Sekundenimpulse. Diese werden gezählt
              und auf dem LCD-Display an Port P2 zur Anzeige gebracht.
              (Zählzeit = 0.....59 Sekunden)
              Dafür ist eine Starttaste an P0.0 und eine
              Stopptaste an P.1 vorhanden.
              Eine LED an Portpin P1.0 wird jede Sekunde invertiert.
              Zusätzlich wird das invertierte Blinksignal an P1.1
              ausgegeben.
              Nach einem Stopp kann mit "Start" wieder weitergezählt
              werden.
              Ausserdem gibt es eine Löschtaste an P1.2, die den Zähler
              und die Anzeige auf Null zurücksetzt.

              Achtung:
              Nun muß das File "LCD_Ctrl_ATMEL.c" mit in das Projekt
              eingebunden werden!


Datum:        29. 11. 2007
Autor:        G. Kraus

-------------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------------*/

#include <t89c51ac2.h>      // Header für AT89C51AC3
#include <stdio.h>          // Standard-Input-Output-Header verwenden

sbit starte=P0^0;           // Starten an P0.0
sbit stoppe=P0^1;           // Stoppen an P0.1
sbit clear=P0^2;            // Setze alles durch "Clear" an P0.2 zurück
sbit LED=P1^0;              // Blinker an P^1.0
sbit blink_inv=P1^1;        // Invertierter Blinker an P1.1

unsigned char Zehntel;      // Zehntelsekunden-Zähler
unsigned char Sekunden;     // Sekunden-Zähler
unsigned char overflow;

code unsigned char zeile_1[21]={"Timer 2 - Uebung    "};
code unsigned char clear_LCD[21]={"000 Sekunden!       "};
/*-------------------------------------------------------------------
Prototypen
-------------------------------------------------------------------*/

void ISR_Timer2(void);    // Prototyp der Interrupt Service Routine

void zeit_s(void);        // LCD-Display-Prototypen
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);
/*---------------------------------------------------------------------
Hauptprogramm
---------------------------------------------------------------------*/
void main(void)
{    AUXR=AUXR&0xFD;       // auf internes ERAM umschalten = EXTRAM löschen
     P1=0x00;              // Port P1 komplett löschen
     P0=0xFF;             // Port P0 erst auf „Einlesen" schalten!
     C_T2=0;               // Timer-Betrieb wählen
     RCLK=0;               // Timer 2 nicht für Serial Port verwenden
     TCLK=0;               //     "    "    "    "  "
     CP_RL2=0;             // Autoreload bei Überlauf wählen
     EA=1;                 // Interrupt-Hauptschalter EIN
     ET2=0;                // Timer 2 - Interrupt AUS
     RCAP2L=0xFF;          // Bei 11,0592 MHz Quarztakt beträgt der
     RCAP2H=0x4B;          // Reload-Wert 19455 = 0x4BFF
                           // (das gibt 50 Millisekunden Zeit)

  /* Bei 12 MHz Quarztakt wäre der Reloadwert 15535 = 0x3CAF) */

     lcd_ein();       // Display-Einschaltroutine
     lcd_ini1();      // Display-Initialisierung
     switch_z1();     // Schalte auf Zeile 1
     show_text(zeile_1);  // Zeige Starttext an

     TF2=0;           // Überlaufflag löschen

     overflow=0;;     // Overflow-Zähler
     Zehntel=0;;      // Zehntelsekunden-Zähler
     Sekunden=0;;     // Sekunden-Zähler
     LED=0;           // LED ausschalten

     TR2=1;           // Timer 2 starten


     while(1)                  //Endlosschleife
     {    if(clear==0)         // Löschtaste Starttaste gedrückt?
          {    ET2=0;                  //Timer 2 AUS
               overflow=0;             //Alle Zähler zurücksetzen
               Zehntel=0;
               Sekunden=0;
               switch_z1();            //Schalte Display auf Zeile 1 um
               show_text(clear_LCD);   //Zeige Starttext in Zeile 1
               LED=0;                  //Blinker-LED AUS
               blink_inv=!LED;         //Invertierter Blinker EIN
               }

          if(stoppe==0)                //Stopptaste gedrückt?
          {    ET2=0;                  //Timer 2 stoppen
          }

          if(starte==0)                //Starttaste gedrückt?
          {    ET2=1;                  //Timer 2 starten
          }
     }


}
```

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

void ISR_Timer2(void)   interrupt 5   // Timer 2 hat Interrupt-Index 5
{
     unsigned char zeit[21]={"000 Sekunden!       "};

     TF2=0;                           // Überlaufflag löschen
     overflow++;                      // Überlaufzähler inkrementieren

     if(overflow>=2)                  // Zweimal gezählt?
     {    overflow=0;
          Zehntel++;                  // Dann die Zehntelsekunden zählen
     }

     if(Zehntel>=10)            // 10 x 100 = 1000 Millisekunden vorbei?
     {    Zehntel=0;            // Zehntelzähler wieder auf Null setzen
          Sekunden++;           // Sekunden zählen
          if(Sekunden==60)
          {    overflow=0;
               Zehntel=0;
               Sekunden=0;
          }

          zeit[2]=Sekunden %10 +0x30;    // Sekunden für Display berechnen
          zeit[1]=(Sekunden/10) %10 +0x30;
          zeit[0]=(Sekunden/100) %10 +0x30;
          switch_z1();                   // Auf Zeile 1 umschalten
          show_text(zeit);               // Zeit anzeigen
          LED=~LED;                      // LED invertieren
          blink_inv=!LED;                // Invertierte LED ansteuern

     }
}
```

10.7. Fast am Ziel: die Sekunden-Stoppuhr

```
/*------------------------------------------------------------------
Programmbeschreibung
------------------------------------------------------------------
Name:          stoppuhr_06.c

Funktion:      Der im 16 Bit-Reload-Modus mit Interrupt betriebene
               Timer 2 erzeugt Sekundenimpulse. Diese werden gezählt
               und auf dem LCD-Display an Port P2 zur Anzeige gebracht.
               (Anzeige in Sekunden, Minuten und Stunden)

               Dafür ist eine Starttaste an P0.0 und eine
               Stopptaste an P0.1 vorhanden.
               Eine LED an Portpin P1.0 wird jede Sekunde komplementiert.
               Zusätzlich wird das invertierte Blinksignal an P1.1
               an einer weiteren LED ausgegeben.

               Nach einem Stopp kann mit "Start" wieder weitergezählt
               werden. Ausserdem gibt es eine Löschtaste an P0.2, die den
               Zähler und die Anzeige auf Null zurücksetzt.

               Achtung:
               Nun muß das File "LCD_Ctrl_ATMEL.c" mit in das Projekt
               eingebunden werden!


Datum:        29. 11. 2007
Autor:        G. Kraus

------------------------------------------------------------------
Deklarationen und Konstanten
------------------------------------------------------------------*/

#include <t89c51ac2.h>      // Header für AT89C51AC3
#include <stdio.h>          // Standard-Input-Output-Header verwenden

sbit starte=P0^0;           // Starten an P0.0
sbit stoppe=P0^1;           // Stoppen an P0.1
sbit clear=P0^2;            // Setze alles durch "Clear" an P0.2 zurück
sbit LED=P1^0;              // Blinker an P^1.0
sbit blink_inv=P1^1;        // Invertierter Blinker an P1.1


unsigned char overflow;
unsigned char Zehntel;
unsigned char  Sekunden;
unsigned char  Minuten;
unsigned char  Stunden;

code unsigned char clear_lcd[21]={"Zeit: 00:00:00      "};

/*------------------------------------------------------------------
Prototypen
------------------------------------------------------------------*/

void ISR_Timer2(void);      // Prototyp der Interrupt Service Routine

void zeit_s(void);          // LCD-Display-Prototypen
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);
/*---------------------------------------------------------------------
Hauptprogramm
---------------------------------------------------------------------*/

void main(void)
{    AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen
     P1=0x00;           // Port P1 komplett löschen
     P0=0xFF;           // Port P0 erst auf „Einlesen" schalten!
     C_T2=0;            // Timer-Betrieb wählen
     RCLK=0;            // Timer 2 nicht für Serial Port verwenden
     TCLK=0;            //     "    "    "    "  "
     CP_RL2=0;          // Autoreload bei Überlauf wählen
     EA=1;              // Interrupt-Hauptschalter EIN
     ET2=0;             // Timer 2 - Interrupt AUS
     RCAP2L=0xFF;       // Bei 11,0592 MHz Quarztakt beträgt der
     RCAP2H=0x4B;       // Reload-Wert 19455 = 0x4BFF
                        // (das gibt 50 Millisekunden Zeit)

  /* Bei 12 MHz wäre der Reloadwert 15535 = 0x3CAF) */

     TF2=0;             // Überlaufflag löschen

     overflow=0;;       // Overflow-Zähler auf Null
     Zehntel=0;;        // Zehntelsekunden-Zähler auf Null
     Sekunden=0;;       // Sekunden-Zähler auf Null
     Minuten=0;         // Minuten-Zähler auf Null
     Stunden=0;         // Stunden-Zähler auf Null

     LED=0;             // LED ausschalten
     blink_inv=!LED;    // Invertiertes LED-Signal an P1.1 ausgeben

     lcd_ein();                 //Display-Einschaltroutine
     lcd_ini1();                //Display-Initialisierung
     switch_z1();               //Schalte auf Zeile 1
     show_text(clear_lcd);      //Zeige "000 Sekunden" an
     TR2=1;                     // Timer 2 starten


     while(1)                   // Endlosschleife
     {    if(clear==0)          // Starttaste gedrückt?
          {    overflow=0;      // Overflow zurücksetzen
               Zehntel=0;       // Zehntelzähler zurücksetzen
               Sekunden=0;      // Sekundenzähler zurücksetzen
               Minuten=0;
               Stunden=0;
               switch_z1();     // Schalte auf Zeile 1
               show_text(clear_lcd);     // Zeige "000 Sekunden" an
               LED=0;
               blink_inv=!LED;
               ET2=0;           // Timer 2 stoppen
          }

          if(stoppe==0)         // Stopptaste gedrückt?
          {    ET2=0;           // Timer 2 stoppen
          }
```

```
          if(starte==0)         // Starttaste gedrückt?
          {    ET2=1;           // Timer 2 starten
          }
     }
}

/*---------------------------------------------------------------------
zusatzfunktionen
---------------------------------------------------------------------*/

void ISR_Timer2(void)   interrupt 5   //Timer 2 hat Interrupt-Index 5
{
     unsigned char zeit[21]={"Zeit: 00:00:00      "};

     TF2=0;                     // Überlaufflag löschen
     overflow++;                // Überlaufzähler inkrementieren

     if(overflow>=2)            // Zweimal gezählt?
     {    overflow=0;
          Zehntel++;            // Dann die Zehntelsekunden zählen
     }

     if(Zehntel>=10)            // 1000 Millisekunden vorbei?
     {    Zehntel=0;            // Zehntelzähler wieder auf Null setzen
          Sekunden++;           // Sekunden zählen

     if(Sekunden==60)           // 60 Sekunden vorbei?
     {    Sekunden=0;           // Dann Sekunden auf Null und
          Minuten++;            // Minute zählen
     }

     if(Minuten==60)            // 60 Minuten vorbei?
     {    Minuten=0;            // Dann Minuten auf Null und
          Stunden++;            // und Stunde zählen
     }

     if(Stunden==24)            // 24 Stunden vorbei?
     {    Stunden=0;            // dann Stunden wieder auf Null
     }

     LED=~LED;                  // Blinken
     blink_inv=!LED;            // Invertiert blinken

     zeit[13]=Sekunden %10 +0x30;        // Sekunden für Display berechnen
     zeit[12]=(Sekunden/10) %10 +0x30;
     zeit[10]=Minuten %10 +0x30;         // Minuten für Display berechnen
     zeit[9]=(Minuten/10) %10 +0x30;
     zeit[7]=Stunden %10 +0x30;          // Stunden für Display berechnen
     zeit[6]=(Stunden/10) %10 +0x30;
     switch_z1();                        // Schalte auf Zeile 1
     show_text(zeit);                    // zeige die Zeit an
   }
}
```

10.8. Geschafft: die Zehntelsekunden-Stoppuhr

```
/*------------------------------------------------------------------
Programmbeschreibung
------------------------------------------------------------------
Name:          stoppuhr_07.c

Funktion:      Der im 16 Bit-Reload-Modus betriebene Timer 2 erzeugt
               50-Millisekundenimpulse. Diese werden als
               Zehntelsekunden gezählt und auf dem LCD-Display an Port 2
               zur Anzeige gebracht (Maximale Zeit = 24 Stunden).
               Die Anzeige erfolgt in Sekunden mit Zehntel, Minuten
               und Stunden.
               Dafür ist eine Starttaste an P0.0 und eine
               Stopptaste an P0.1 vorhanden. Nach einem Stopp kann
               mit "Start" wieder vom letzten Zählerstand aus weitergezählt
               werden. Ausserdem gibt es eine Löschtaste an P0.2, die den
               Zähler und die Anzeige auf Null zurücksetzt.
               Zur optischen Demonstration blinken zwei LEDs an den
               Portpins P1.0 und P1.1 gegenphasig im Zehntelsekundentakt.

               Achtung:
               Nun muß das File "LCD_Ctrl_ATMEL.c" mit in das Projekt
               eingebunden werden!


Datum:         29. 11. 2007
Autor:         G. Kraus

------------------------------------------------------------------
Deklarationen und Konstanten
------------------------------------------------------------------*/

#include <t89c51ac2.h>      // Header für AT89C51AC3
#include <stdio.h>          // Standard-Input-Output-Header verwenden

sbit starte=P0^0;           // Starten an P0.0
sbit stoppe=P0^1;           // Stoppen an P0.1
sbit clear=P0^2;            // Setze alles durch "Clear" an P0.2 zurück
sbit LED=P1^0;              // Blinker an P^1.0
sbit blink_inv=P1^1;        // Invertierter Blinker an P1.1

unsigned char overflow;     // Overflow-Zähler
unsigned char Zehntel;      // Zehntelsekunden-Zähler
unsigned char  Sekunden;    // Sekunden-Zähler
unsigned char  Minuten;     // Minuten-Zähler
unsigned char  Stunden;     // Stunden-Zähler
code unsigned char clear_lcd[21]={"Zeit: 00:00:00,0    "};
/*------------------------------------------------------------------
Prototypen
------------------------------------------------------------------*/

void ISR_Timer2(void);      // Prototyp der Interrupt Service Routine

void zeit_s(void);          // LCD-Display-Prototypen
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);
/*---------------------------------------------------------------------
Hauptprogramm
---------------------------------------------------------------------*/
void main(void)
{    AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen
     P1=0x00;           // Port P1 komplett löschen
     P0=0xFF;           // Port P0 erst auf "Einlesen" schalten!
     C_T2=0;            // Timer-Betrieb wählen
     RCLK=0;            // Timer 2 nicht für Serial Port verwenden
     TCLK=0;            //     "    "    "    "  "
     CP_RL2=0;          // Autoreload bei Überlauf wählen
     EA=1;              // Interrupt-Hauptschalter EIN
     ET2=0;             // Timer 2 - Interrupt AUS
     RCAP2L=0xFF;       // Bei 11,0592 MHz Quarztakt beträgt der
     RCAP2H=0x4B;       // Reload-Wert 19455 = 0x4BFF
                        // (das gibt 50 Millisekunden Zeit)

  /* Bei 12 MHz wäre der Reloadwert 15535 = 0x3CAF) */

     TF2=0;             // Überlaufflag löschen

     overflow=0;;       // Overflow-Zähler auf Null
     Zehntel=0;;        // Zehntelsekunden-Zähler auf Null
     Sekunden=0;;       // Sekunden-Zähler auf Null
     Minuten=0;         // Minuten-Zähler auf Null
     Stunden=0;         // Stunden-Zähler auf Null
     LED=0;             // LED ausschalten
     blink_inv=!LED;    // Invertiertes LED-Signal an P1.1 ausgeben

     lcd_ein();                 // Display-Einschaltroutine
     lcd_ini1();                // Display-Initialisierung
     switch_z1();               // Schalte auf Zeile 1
     show_text(clear_lcd);      // Zeige "00:00:00,0" an
     TR2=1;                     // Timer 2 starten

     while(1)                   // Endlosschleife
     {    if(clear==0)          // Starttaste gedrückt?
          {    overflow=0;      // Overflow zurücksetzen
               Zehntel=0;       // Zehntelzähler zurücksetzen
               Sekunden=0;      // Sekundenzähler zurücksetzen
               Minuten=0;       // Minutenzähler zurücksetzen
               Stunden=0;       // Stundenzähler zurücksetzen

               switch_z1();              // Schalte auf Zeile 1
               show_text(clear_lcd);     // Zeige "000 Sekunden" an
               LED=0;
               blink_inv=!LED;
               ET2=0;           // Timer 2 stoppen
          }

          if(stoppe==0)         // Stopptaste gedrückt?
          {    ET2=0;           // Timer 2 stoppen
          }

          if(starte==0)         // Starttaste gedrückt?
          {    ET2=1;           // Timer 2 starten
          }
     }
}
```

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

void ISR_Timer2(void)   interrupt 5   //Timer 2 hat Interrupt-Index 5
{
     unsigned char zeit[21]={"Zeit: 00:00:00,0    "};

     TF2=0;                  // Überlaufflag löschen
     overflow++;             // Überlaufzähler inkrementieren

     if(overflow>=2)         // Zweimal gezählt?
     {    overflow=0;        // Dann Overflow zurücksetzen und
          Zehntel++;         // die Zehntelsekunden inkrementieren
     }

     if(Zehntel>=10)         // 1000 Millisekunden vorbei?
     {    Zehntel=0;         // Zehntelzähler wieder auf Null setzen
          Sekunden++;        // und Sekunden inmkrementieren
     }

     if(Sekunden==60)        // 60 Sekunden vorbei?
     {    Sekunden=0;        // dann Sekunden wieder auf Null
          Minuten++;         // und Minuten inkrementieren
     }

     if(Minuten==60)         // 60 Minuten vorbei?
     {    Minuten=0;         // dann Minuten wieder auf Null
          Stunden++;         // und Stunden inkrementieren
     }

     if(Stunden==24)         // 24 Stunden erreicht?
     {    Stunden=0;         // dann Stundenzähler wieder auf Null
     }

     LED=~LED;                        // blinken
     blink_inv=!LED;                  // invertiert blinken

     zeit[15]=Zehntel %10 +0x30;    // Zehntelsekunde für Display errechnen
     zeit[13]=Sekunden %10 +0x30;       // Sekunden für Display errechnen
     zeit[12]=(Sekunden/10) %10 +0x30;
     zeit[10]=Minuten %10 +0x30;        // Minuten für Display errechnen
     zeit[9]=(Minuten/10) %10 +0x30;
     zeit[7]=Stunden %10 +0x30;         // Stunden für Display errechnen
     zeit[6]=(Stunden/10) %10 +0x30;
     switch_z1();                       // auf Zeile 1 umschalten
     show_text(zeit);                   // Zeit anzeigen
}
```

11. Die Serielle Schnittstelle (Serial I / O Port)

Warnung:

Wird ein 12 MHz-Quarz auf dem Board verwendet, dann lassen sich ausschließlich die Baudraten „2400 / 4800 / 9600 Baud“ und dazu nur mit einer Abweichung von ca. 10% realisieren (… das mögen manche anderen Geräte nicht!)

Abhilfe und vollen Erfolg (bis 115 200 Baud) bringt jedoch der Einsatz des im Bausatz enthaltenen Quarzes mit 11,059 MHz.

11.1. Grundlagen

Diese Baugruppe wurde (mit kleinen Änderungen) vom 80C32-Vorgänger übernommen. Leider fehlt (wie dort!) der von unserem Compuboard her bekannte und geschätzte Baudratengenerator….

Ansonsten gelten folgende Eigenschaften:

Es können vier verschiedene Betriebsarten eingestellt werden.

„**Mode 0**“ ist der Schieberegister-Betrieb. Dabei wird außer dem seriellen Byte zusätzlich am Sendepin der Schiebetakt ausgegeben.

„**Mode 1“ ist unsere Standard-Betriebsart**, nämlich **UART** (= Universal Asynchronous Receiver and Transmitter) mit **10 Bit Wortlänge, nämlich Startbit / 8 Datenbits / Stoppbit.**

Die Baudrate kann vom Anwender durch den Einsatz eines Timers selbst festgelegt werden!

„**Mode 2“ und „Mode 3“** arbeiten mit **11 Bit Wortlänge**, nämlich Startbit **/ 8 Datenbits / Parity-Bit / Stoppbit**. Der Unterschied beider Betriebsarten liegt nur in **der möglichen Baudrate**:

Bei Mode 2 kann man nur zwischen f_{Quarz} / 32 oder f_{Quarz} / 64 wählen.

Bei Mode 3 kann die Baudrate durch Einsatz eines Timers selbst festgelegt werden.

Als Einsteiger benötigt man folgende Informationen zur Programmierung:

Im Register „SCON“ finden sich fast alle zur Steuerung und zum korrekten Betrieb erforderlichen Bits.

Der Sendevorgang wird gestartet, wenn das zu sendende Byte in das Register SBUF kopiert wird.

Für den Fortgeschrittenen gibt es einige zusätzliche Spielereien, denn ATMEL bietet sowohl eine „**Framing Error Detection**“ wie auch eine „**Automatic Adress Recognition**“ an. In diesen Fällen muss man sich zusätzlich um die Register PCON, SADEN und SADDR kümmern… Siehe ab Seite 60 des Datenblattes.

11.2. Einstieg: Wiederholtes Senden verschiedener Zeichen

Aufgabe:

Schließen Sie die Tastenplatine über ein Flachbandkabel an Port P0 an und schreiben Sie ein Programm, das beim Drücken einer Taste pausenlos ein einziges Zeichen, gefolgt von einer Pause von etwa 1 Millisekunde, ausgibt. (….Wem dabei das dauernde Drücken der Taste zuviel wird: es gibt da auf der Platine einen DIP-Schalter, der den Drucktasten parallelgeschaltet ist….)

Belegen Sie jede einzelne der acht Tasten mit einem unterschiedlichen Zeichen (= Buchstabe oder Zahl).

Verbinden Sie dann Port P3 über ein Flachbandkabel mit der Buchsenplatine und sehen Sie sich mit dem Oszilloskop das Signal direkt am Sendepin TxD (= Pin P3^1) des Controllers an (…Bitte 10:1 – Tastkopf verwenden!).

Vergleichen Sie dieses Signal mit dem Spannungsverlauf am Pin TxD beim Sub-D 9-Stecker, an dem das Nullmodem-Kabel zur Verbindung mit dem PC angeschlossen ist.

Öffnen Sie anschließend bei Ihrem PC über den Pfad „Programme / Zubehör / Kommunikation“ das Programm „Hyperterminal“. Stellen Sie eine Verbindung mit 9600 Baud her (Genaue Anleitung: Siehe Anhang dieses Manuskriptes) und kontrollieren Sie, ob Ihre Zeichen sowohl korrekt zum PC gesendet wie auch dort richtig erkannt und dargestellt werden.

Lösung:

```
/*------------------------------------------------------------------
Programmbeschreibung
------------------------------------------------------------------
Name:         rs_232_2.c
Funktion:     Über den Pin TxD (= Portpin P3.1) der Seriellen Schnittstelle
              des Controllers wird beim Drücken einer Taste dauernd dasselbe
              Zeichen mit 9600 Baud ausgegeben, wobei zwischen jeden
              Sendevorgang eine Pause von 1 Millisekunden gelegt wird.
               Es wird der interne Baudratengenerator benützt. Das
              Ausgangssignal kann nun mit dem Oszilloskop gemessen und
              analysiert werden. Es kann zwischen 8 Tasten an Port P0.0
              gewählt werden, um unterschiedliche Zeichen zu senden.

Datum:        10. 11. 2007
Autor:        G. Kraus
------------------------------------------------------------------
Deklarationen und Konstanten
------------------------------------------------------------------*/
#include <t89c51ac2.h>  //Registersatz des AT89C51AC3 verwenden
#include <stdio.h>      // Standard-Eingabe-Ausgabe-Header

sbit Taste_0=P0^0;      // Taste an Portpin P0^0
sbit Taste_1=P0^1;      // Taste an Portpin P0^1
sbit Taste_2=P0^2;      // Taste an Portpin P0^2
sbit Taste_3=P0^3;      // Taste an Portpin P0^3
sbit Taste_4=P0^4;      // Taste an Portpin P0^4
sbit Taste_5=P0^5;      // Taste an Portpin P0^5
sbit Taste_6=P0^6;      // Taste an Portpin P0^6
sbit Taste_7=P0^7;      // Taste an Portpin P0^7
void zeit(void);        // Prototypen-Anmeldung der Verzögerung
/*------------------------------------------------------------------
Hauptprogramm
------------------------------------------------------------------*/
void main(void)
{    AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen
     SCON=0x40;         // Mode 1 (Startbit, 8 Datenbits, no parity)
     P0=0xFF;           // Port P0 auf Lesen schalten
     TI=0;              // Sendeflag löschen
```

```
TMOD=0x20;          // Timer1 im 8 Bit-Reload-Betrieb (Mode 2)
TL1=0xFD;           // Start-Wert ist FD
TH1=0xFD;           // Reload-Wert ist FD für 9600 Baud

TF1=0;              // Timer1-Überlaufflag vorsichtshalber löschen
TR1=1;              // Timer1 einschalten
while(1)
{    while(Taste_0==0)
     {  SBUF='1';             // Sende ASCII-Wert für 1
        while(TI==0);         // Senden beendet?
        TI=0;                 // Wenn ja: Sendeflag löschen
        zeit();               // Delay von 1000 Mikrosekunden
     }

     while(Taste_1==0)
     {  SBUF='2';             // Sende ASCII-Wert für 2
        while(TI==0);         // Senden beendet?
        TI=0;                 // Wenn ja: Sendeflag löschen
        zeit();               // Delay von 1000 Mikrosekunden
     }

     while(Taste_2==0)
     {  SBUF='3';             // Sende ASCII-Wert für 3
        while(TI==0);         // Senden beendet?
        TI=0;                 // Wenn ja: Sendeflag löschen
        zeit();               // Delay von 1000 Mikrosekunden
     }

     while(Taste_3==0)
     {  SBUF='5';             // Sende ASCII-Wert für 5
        while(TI==0);         // Senden beendet?
        TI=0;                 // Wenn ja: Sendeflag löschen
        zeit();               // Delay von 1000 Mikrosekunden
     }

     while(Taste_4==0)
     {  SBUF='A';             // Sende ASCII-Wert für A
        while(TI==0);         // Senden beendet?
        TI=0;                 // Wenn ja: Sendeflag löschen
        zeit();               // Delay von 1000 Mikrosekunden
     }

     while(Taste_5==0)
     {  SBUF='d';             // Sende ASCII-Wert für d
        while(TI==0);         // Senden beendet?
        TI=0;                 // Wenn ja: Sendeflag löschen
        zeit();               // Delay von 1000 Mikrosekunden
     }

     while(Taste_6==0)
     {  SBUF='?';             // Sende ASCII-Wert für ?
        while(TI==0);         // Senden beendet?
        TI=0;                 // Wenn ja: Sendeflag löschen
        zeit();               // Delay von 1000 Mikrosekunden
     }

     while(Taste_7==0)
     {  SBUF='$';             // Sende ASCII-Wert für $
        while(TI==0);         // Senden beendet?
        TI=0;                 // Wenn ja: Sendeflag löschen
        zeit();               // Delay von 1000 Mikrosekunden
     }
```

```
    }
}


/*-------------------------------------------------------------------------
Zusatzfunktionen
-------------------------------------------------------------------------*/
void zeit(void)
{
  unsigned int x;
  for(x=0;x<=200;x++);  // Int-Wert 200 ergibt etwa 1 Millisekunde
}
//-------------------------------------------------------------------------
```

11.3. Empfangsprogramm mit Ausgabe des Zeichens auf dem Display

Aufgabe:

Schreiben Sie ein Programm, das einlaufende Zeichen (mit einer Baudrate von 9600) erkennt und in der ersten Zeile des Displays ausgibt. Löschen Sie alle übrigen Zeichen des Displays und sorgen Sie dafür, dass nach 16 empfangenen Zeichen die Zeile wieder gelöscht und neu links begonnen wird.
Verwenden Sie die Interrupt-Steuerung (Interrupt-Vector für UART ist 0x23) zur Darstellung der Zeichen.
Senden Sie mit dem „Hyperterminal“

Achtung:
Nach dem Hinüberflashen des Programms muss der Programmierschalter sofort auf „Run“ gestellt werden, bevor das Hyperterminal aufgerufen und die Verbindung zum Board hergestellt wird. Dann drückt man einmal auf den Reset-Knopf am Board und alles sollte funktionieren.

Lösung:

```
/*-------------------------------------------------------------
Programmbeschreibung
-------------------------------------------------------------
Name:          rs_232_receive.c
Funktion:      Empfangs- und Anzeigeprogramm für die Serielle Schnittstelle.
               Das Programm wartet auf das Einlaufen eines Zeichens in der
               Schnittstelle und stellt es auf dem Display dar.
               Maximal 16 Zeichen können nacheinander dargestellt werden,
               dann wird alles gelöscht und wieder links begonnen.

ACHTUNG:       Dieses Programm muss vom Linker anschließend mit dem
               LCD_Anzeigeprogramm "LCD_CTRL_ATMEL.C" zusammengebunden
               werden. Deshalb ist ein entsprechender Eintrag in
               der Projektliste und eine Prototypen-Deklaration
               aller im Anzeige-Modul verwendeten Funktionen (im
               Hauptprogramm) erforderlich

Datum:        11. 11. 2007
Autor:        G. Kraus

-------------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------------*/

#include <t89c51ac2.h>     // Registersatz für AT89C51AC3
#include <stdio.h>         // Standard-Eingabe-Ausgabe-Header

void zeit_s(void);         //Prototypen des LCD-Display-Moduls
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);

code unsigned char leer[20]={"                    "};
unsigned char x,y;
void ISR_RS232(void);
```

```
/*-------------------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------------------*/

void main(void)
{       AUXR=AUXR&0xFD;         // auf internes ERAM umschalten
        lcd_ein();              // Einschaltroutine für Display
        lcd_ini1();             // Function Set für Display
        switch_z1();            // Stelle Cursor auf Anfang von Zeile 1
        show_text(leer);        // Zeige den Text "leer" an
        switch_z2();            // Schalte auf den Anfang von Zeile 2
        show_text(leer);        // Zeige den Text "leer" an
        switch_z3();            // Schalte auf den Anfang von Zeile 3
        show_text(leer);        // Zeige den Text "leer" an
        switch_z4();            // Schalte auf den Anfang von Zeile 4
        show_text(leer);        // Zeige den Text "leer" an
        switch_z1();            // Schalte auf den Anfang von Zeile 1
        x=0;
        y=0;
        SCON=0x40;              // Mode 1 (Startbit, 8 Datenbits, no parity)
        REN=1;                  // Empfänger einschalten

        TI=0;                   // Sendeflag löschen
        RI=0;                   // Empfangsflag löschen

        ES=1;

        TMOD=0x20;              // Timer1 im 8 Bit-Reload-Betrieb (Mode 2)
        TL1=0xFD;               // Start-Wert ist FD
        TH1=0xFD;               // Reload-Wert ist FD für 9600 Baud

        TF1=0;                  // Timer1-Überlaufflag vorsichtshalber löschen
        TR1=1;                  // Timer1 einschalten
        EA=1;

        while(1);

}


void ISR_RS232(void)    interrupt 4      // RS232 hat Interrupt-Vector 0x23
{       RI=0;                   // RI-Flag löschen
        y=SBUF;                 // Eingelaufenes Byte kopieren

        if(x==16)
        {    switch_z1();
             show_text(leer);
             switch_z1();
             x=0;
        }
        show_char(&y);
        x++;
}
```

11.4.Unterrichtsprojekt: Spannungs-Fernmessung mit RS232-Übertragung

11.4.1. Übersicht

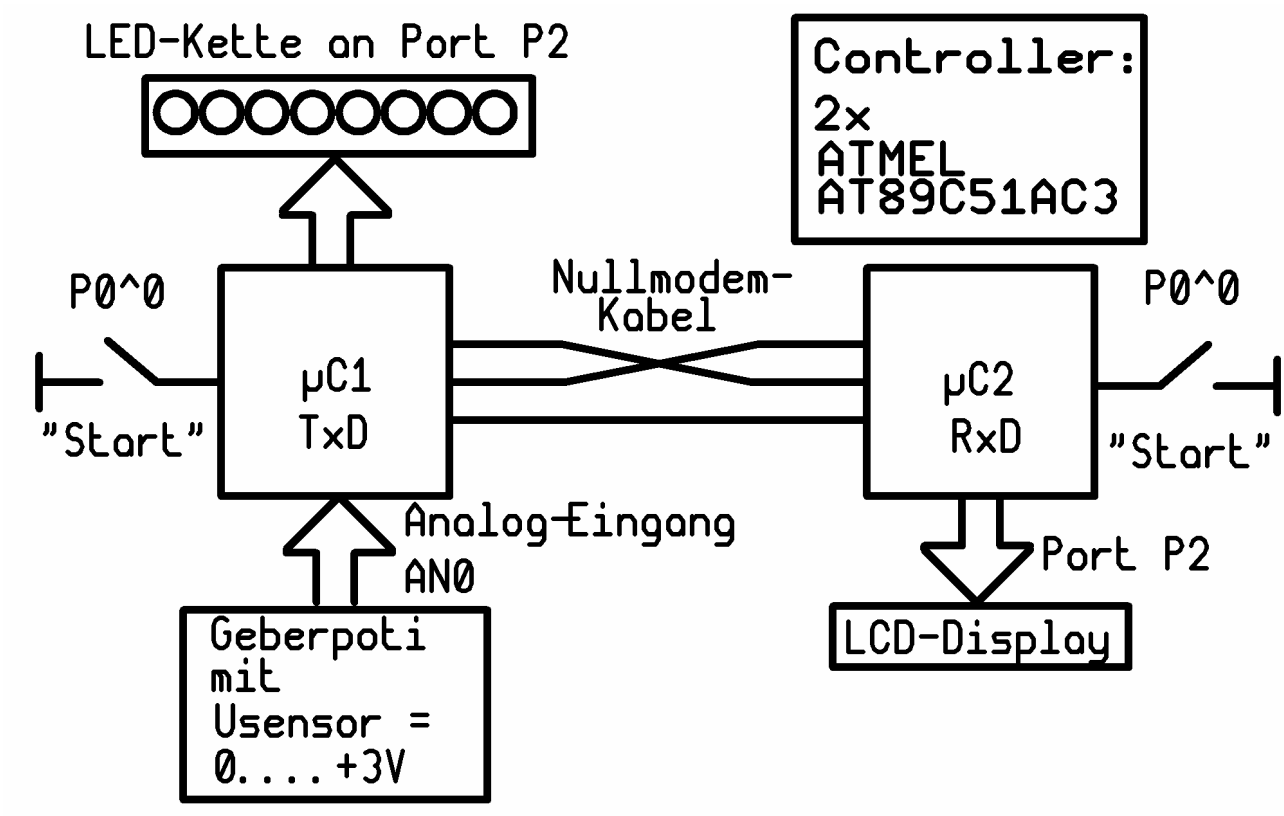
Mit diesem Projekt soll demonstriert werden, wie ein Sensorsignal mit einem Microcontroller erfasst und der Messwert über eine Serielle Schnittstellenverbindung zu einem zweiten Controller übertragen wird.
An den zweiten Controller ist über Port P2 ein LCD-Display angeschlossen, auf dem das Messergebnis als Spannung zwischen Null und +3V angezeigt wird.
Die Aufgabe für den „Sendecontroller“ sieht dann so aus:

```
/* sender.c
Dieses Programm misst alle 0,2 Sekunden die Spannung an Portpin AN0 im Bereich 0....+3V, übergibt das
Ergebnis als Hex-Zahl an die LEDs an Port P2 und verschickt es über die Serielle Schnittstelle mit 9600 Baud.
Achtung:
Das Programm startet erst nach einem Druck auf die LOW-aktive Starttaste an Portpin P0^0. */
```

Das Empfangscontroller-Programm wird dagegen folgendermaßen kommentiert:

```
/* empf.c
Dieses Programm empfängt den über die Schnittstelle gesendeten Hex-Messwert des DVM's und zeigt das
Ergebnis auf dem LCD-Display als Spannungswert (Bereich 0....+3V) an. Das Display wird an Port P2
angeschlossen.
Das Programm startet erst durch Druck auf die LOW-aktive Starttaste an Portpin P0^0. */
```

Dazu gehört folgende Verdrahtung:



11.4.2. Das Sende-Controller-Programm

Nach den Headern und der Deklaration der Starttaste werden **sechs Zusatzfunktionen** als Prototypen angemeldet. Das Hauptprogramm beginnt dann mit den **Initialisierungen von AD-Wandler, Timer 1, Timer 2 und Serieller Schnittstelle**. Die beiden benützen Variablen (Zählvariable und Ergebnisvariable) werden auf Null gesetzt, dann wartet das Programm auf den Start durch Taste P1^0. Ist der Start erfolgt, dann werden **alle Interrupts freigegeben** und der Controller landet in einer Endlosschleife.

Folgende Funktionen wurden geschrieben:

a) **void init_Timer1(void);**
Timer 1 soll im „8 Bit-Reloadbetrieb durch Überlauf“ mit Interrupt arbeiten und für eine Baudrate von 9600 einen Reloadwert von 0xFD aufweisen.

b) **void init_Timer2(void);**
Timer 2 soll im „16 Bit - Reloadbetrieb durch Überlauf“ mit Interrupt arbeiten und einen Zeittakt von 50 Millisekunden erzeugen. Dazu ist ein Reloadwert von 15535 zu programmieren und in die entsprechenden Register zu laden.
Nach vier Durchläufen ist die geforderte Zeit von 200 Millisekunden vergangen und eine neue Messung wird gestartet.

c) **void ISR_Timer2(void);**
Das ist die Interrupt-Service-Routine für den Timer 2 im 16 Bit - Reloadbetrieb zur Erzeugung des 0,2 Sekunden-Taktes. Nach Ablauf dieser Zeit wird wieder neu gemessen und das Ergebnis an die LEDs bzw. über die Serielle Schnittstelle verschickt. Timer 2 besitzt den Interrupt-Vektor 0x2B = 43d und benötigt deshalb den Interrupt-Index „5“.

d) **void init_ADC(void);**
Damit wird der AD-Wandler initialisiert (Einzelmessung an Eingang AN0).

e) **void init_RS232(void);**
Die Schnittstelle wird auf “1 Startbit / 8 Datenbits / 1 Stoppbit” eingestellt. Timer 1 dient n als Taktgenerator für die Schnittstelle. Mit einem 11,059MHz-Quarz und dem Reloadwert „0xFD“ sorgt er für eine Baudrate von exakt 9600 Baud.

f) **void wait_ADC_ini(void);**
Beim ersten Einschalten des Ad-Wandlers ist eine bestimmte Wartezeit vorzusehen, bevor die erste Messung gestartet wird.

Programm:

```
/* sensor_transmit.c
Dieses Programm misst alle 0,2 Sekunden die Spannung an Portpin AN0 im
Bereich 0....+3V und sendet es über die Serielle Schnittstelle mit 9600
Baud. Das Programm beginnt erst nach einem Druck auf die Starttaste an
Portpin P1^0 zu laufen */

// Erstellt am 12-11-2007 durch G. Kraus

#include <t89c51ac2.h>      // Verwendeter Header für AT89C51AC3
#include <stdio.h>          // Standard-Eingabe-Ausgabe-Header

sbit start=P0^0;            // Starttaste

void init_Timer1(void);     // Prototypen anmelden
void init_Timer2(void);
void init_ADC(void);
void ISR_Timer2(void);
void init_RS232(void);
void wait_ADC_ini(void);

unsigned char count;        // Zählvariable
unsigned char z;            // Ergebnis-Variable
unsigned char channel;      // AD-Kanal-Nummer
```

```
/*--------------------------------
Hauptprogramm
---------------------------------*/

void main(void)
{    AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen
     init_Timer1();          // Timer 1  initialisieren
     init_ADC();             // ADC initialisieren
     init_RS232();           // Schnittstelle initialisieren
     init_Timer2();          // Timer 2  initialisieren

     count=0,                // Zähler auf Null
     z=0;                    // Ergebnis auf Null
     while(start==1);        // Auf Starttaste warten
     EA=1;                   // Alle Interrupts freigeben

     while(1);               // Endlossschleife
}

/*---------------------------------
Zusatz-Funktionen
----------------------------------*/

void init_Timer1(void)
{    TMOD=0x20;              // Timer1 im Reload-Betrieb (Mode 2)
     TL1=0xFD;               // Start-Wert ist 0xFD für 9600 Baud
     TH1=0xFD;               // Reload-Wert ist 0xFD für 9600 Baud

     TF1=0;                  // Timer1-Überlaufflag vorsichtshalber löschen
     TR1=1;                  // Timer1 einschalten
}


void init_Timer2(void)
{    C_T2=0;        // Timer-Betrieb
     RCLK=0;     // Timer 2 nicht für Serial Port - Takt (Receive) verwenden
     TCLK=0;     //     "    "    "    "              "   (Transmit) "
     CP_RL2=0;      // Autoreload bei Überlauf
     ET2=1;         // Timer 2 - Interrupt EIN
     RCAP2L=0xAF;   // Reload-Wert ist 15535 (gibt 50 Millisekunden Zeit)
     RCAP2H=0x3B;   // (= 0x3BAF)
     TF2=0;         // Überlaufflag löschen
     TR2=1;         // Timer 2 starten
}

void ISR_Timer2(void) interrupt 5  // Timer 2 hat Int-Vektor "0x2B" = 43d)
{
     TF2=0;                    // Überlaufflag löschen
     count++;                  // Überläufe zählen

     if(count==4)              // 4 gibt 200ms Verzögerung
     {    count=0;             // Zähler zurücksetzen
          ADCON=ADCON|0x08;    // ADSST setzen, "Single Conversion" starten

          while((ADCON&0x10)!=0x10);     // Warten, bis ADEOC setzt
          ADCON=ADCON&0xEF;              // ADEOC wieder löschen
          z=ADDH;                        // Ergebnis nach z
          P2=z;                          // Ergebnis an LED-Kette
          SBUF=z;                        // z senden
          while(TI==0);                  // Senden beendet?
          TI=0;                          // Wenn ja: Sendeflag löschen
     }
}
```

```
void init_ADC(void)
{    channel=0x00;           // Kanal AN0 gewählt
     ADCF=0x01;              // Pin P1.0 als AD-Channel betreiben
     ADCON=0x20;             // AD Enable freigeben
     wait_ADC_ini();         // warten, bis ADC initialisiert
     ADCON=ADCON&0xF8;       // SCH0...SCH2 löschen
     ADCON=ADCON|channel;    // Auf Channel AN0 schalten
}

void init_RS232(void)
{    SCON=0x40;              // Mode 1 (Startbit, 8 Datenbits, no parity)
}

void wait_ADC_ini(void)      // Wartezeit beim Start des ADC
{    unsigned char x;
     for(x=0;x<10;x++);
}
```

11.4.3. Das Empfangs-Controller-Programm

Nach den Headern und der Deklaration der Starttaste P1^0 werden drei Zusatzfunktionen als Prototypen angemeldet. Das Hauptprogramm startet mit der Initialisierung von Display, Timer 1 und Serieller Schnittstelle. Dann wird gewartet, bis Jemand die Starttaste drückt. Geschieht das, dann zeigt das Display „Fernmessung“, die Ergebnis-Variable wird auf Null gesetzt und die Interrupts werden freigegeben. Anschließend geht das Programm in eine Endlosschleife und wartet auf das gesendete Byte.

a) **void init_Display(void);**
Es werden die beiden Startroutinen „lcd_ein()“ und „lcd_ini1()“ abgearbeitet sowie der Cursor auf den Anfang von Zeile 1 geschaltet.

b) **void init_RS232(void);**
Die Schnittstelle wird auf “1 Startbit / 8 Datenbits / 1 Stoppbit” eingestellt. Dazu muß der Empfänger durch Setzen des Bits „REN“ (= Receive Enable) eingeschaltet werden.

c) **void init_Timer1(void);**
Timer 1 soll im „8 Bit-Reloadbetrieb durch Überlauf“ mit Interrupt arbeiten und muß für eine Baudrate von 9600 einen Reloadwert von 0xFD aufweisen.

c) **void ISR_RS232(void);**
Das ist die Interrupt-Service-Routine für die Serielle Schnittstelle. Läuft ein Byte beim Pin „RxD“ ein, so springt der Controller in diese Routine (mit Interrupt-Index „4“ bzw. interrupt-Vektor 0x23 = 35d) und muss zuerst das auslösende RI-Flag löschen. Das empfangene Byte wird dann in eine Variable umkopiert und an der LED-Kette ausgegeben. Dann erfolgt nicht nur die Ergebnis-Umrechnung vom Hexadezimalsystem in BCD, sondern auch die Ermittlung der zugehörigen Spannung im Messbereich 0….+3V sowie die Umwandlung jeder Stelle in den zugehörigen ASCII-Code. Jede einzelne Stelle des Ergebnisses wird in ein Array geschrieben. Das Gesamtergebnis wird schließlich an das Display geschickt und dort in der zweiten Zeile angezeigt.

Programm:

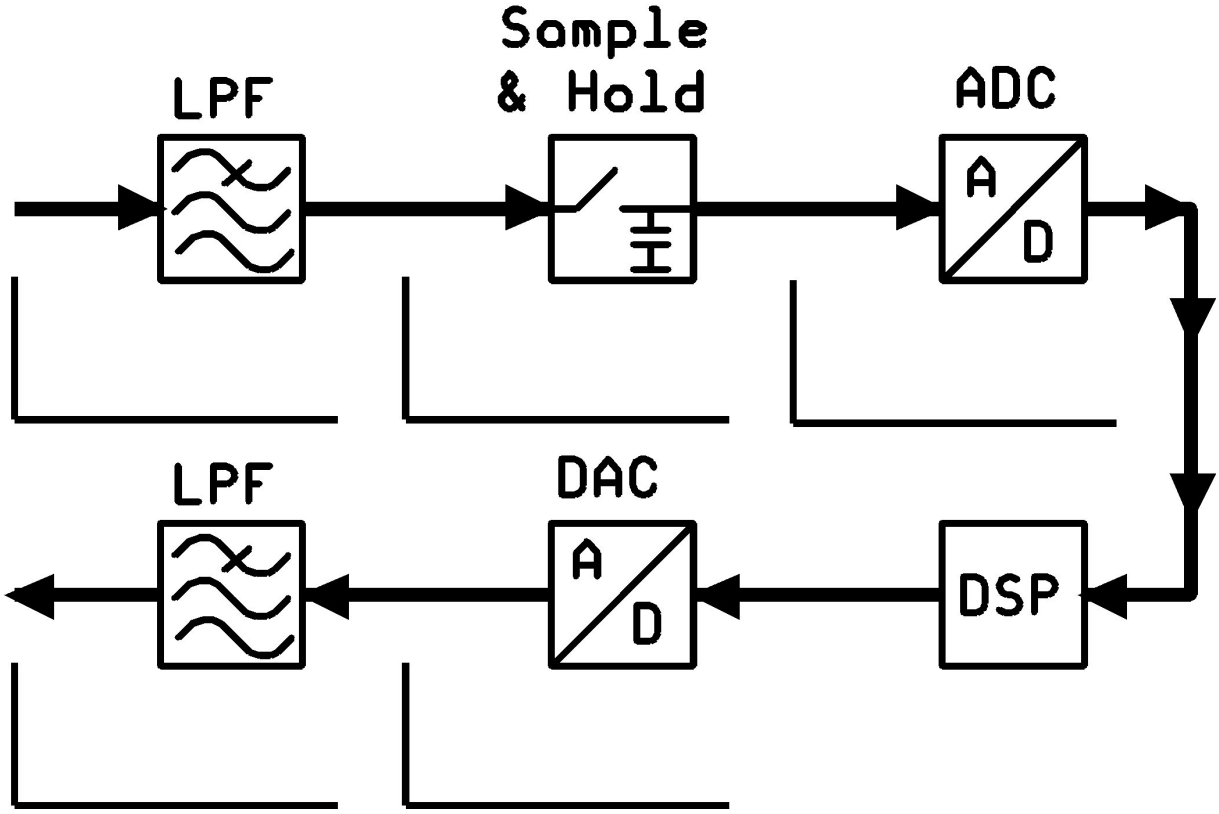
```
/* sensor_receive.c
Dieses Programm empfängt den über die Schnittstelle gesendeten
Hex-Messwert des DVMs und zeigt das Ergebnis auf dem LCD-Display
als Spannungswert (Bereich 0....+3V) an.
Das Programm startet erst durch Druck auf die Starttaste an Portpin P1^0.
*/

// Erstellt am 12-11-2007 durch G. Kraus

#include <t89c51ac2.h>
#include <stdio.h>

sbit starten=P0^0;          // P1^0 als Starttaste

void zeit_s(void);          // Prototypen anmelden
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);

void ISR_RS232_receive(void);
void init_RS232_receive(void);
void init_Display(void);
void init_Timer1(void);
```

```
code char start[21]={"Fernmessung         "};     // Starttext
unsigned char z;          // Empfangsspeicher
/*--------------------------------
Hauptprogramm
---------------------------------*/

void main (void)
{    AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen
     init_Display();           // Display initialisieren
     init_Timer1();            // Timer 1 anwerfen
     init_RS232_receive();     // Schnittstelle initialisieren

     while(starten==1);        // Auf Starttaste warten
     show_text(start);         // Starttext anzeigen
     z=0;                      // Empfangsspeicher  auf Null setzen
     EA=1;                     // Alle Interrupts freigeben
     while(1);                 // Endlosschleife
}

/*---------------------------------
Zusatz-Funktionen
----------------------------------*/
void init_Timer1(void)
{    TMOD=0x20;              // Timer1 im Reload-Betrieb (Mode 2)
     TL1=0xFD;               // Start-Wert ist 0xFD für 9600 Baud
     TH1=0xFD;               // Reload-Wert ist 0xFD für 9600 Baud

     TF1=0;                  // Timer1-Überlaufflag vorsichtshalber löschen
     TR1=1;                  // Timer1 einschalten
}

void ISR_RS232_receive(void) interrupt 4        // RS232 hat Int-Vektor 0x23
{
  char zeile_1[21]={"U = 0,00V           "};   // Anzeige-Array deklarieren
  RI=0;                                        // RI-Flag löschen
  z=SBUF;                                 // Empfangenes Byte an z übergeben

  zeile_1[7]=z*150*2/255 % 10 +0x30;    //Zweite Nachkommastelle berechnen
  zeile_1[6]=z*150/255/5 % 10 +0x30;    //Erste Nachkommastelle berechnen
  zeile_1[4]=z*150/255/50 % 10 + 0x30;  //Stelle vor dem Komma berechnen

  switch_z2();                          // Cursor auf Anfang von Zeile 2
  show_text(zeile_1);                   // Ergebnis-Array anzeigen
}

void init_RS232_receive(void)
{    SCON=0x40;              // Mode 1 (Startbit, 8 Datenbits, no parity)
     REN=1;                  // Empfänger einschalten
     TI=0;                   // Sendeflag löschen
     RI=0;                   // Empfangsflag löschen
     ES=1;                   // Interrupt der Schnittstelle freigeben
}


void init_Display(void)
{    lcd_ein();              // Einschaltroutine für Display
     lcd_ini1();             // Betriebsdaten: 2 Zeilen, kein Cursor
     switch_z1();            // Auf Zeile 1 umschalten
}
```

12. Unterrichtsprojekt: Einführung in die Digitale Verarbeitung von Analogen Signalen (= DSP-Grundlagen) mit dem Microcontroller

12.1. Prinzip der Digitalen Signalverarbeitung

Das Analogsignal wird zuerst durch einen Tiefpass geschickt und auf diese Weise „bandbegrenzt". Anschließend speist es eine „Sample and Hold-Schaltung", in der sein Augenblickswert durch extrem kurze „Sample-Impulse" laufend abgefragt wird. Diese „Abfrage-Ergebnis" wird in einem Kondensator zwischengespeichert und einem Analog-Digital-Wandler zugeführt. So entsteht eine laufende Folge von „probes" mit einer konstanten Frequenz (= sample rate), die in Form eines seriellen 8 Bit- oder 16 Bit-Datenstroms im darauf folgenden Rechner oder Digitalem Signalprozessor -- oder wie bei uns: in einem Microcontroller -- verarbeitet wird.
Die Bearbeitung im Rechner liefert anschließend ein Ergebnis, als ob man z. B. das Analogsignal durch eine Filterschaltung geschickt hätte (….wobei man durch das Programm festlegt, ob das ein Tiefpass, ein Hochpass, ein Bandpass oder eine Bandsperre ist).
Übergibt man die so bearbeitete Wertefolge einem Digital-Analog-Wandler, so braucht man an dessen Ausgang nur noch einen passenden Tiefpassfilter zur Unterdrückung unerwünschter Signalanteile (speziell der Abtastfrequenz und ihrer Oberwellen). So ergibt sich wieder die echte, ursprüngliche Analogform.



Vorteile der Digitalen Signalverarbeitung:

Man kann nun allein durch ausreichende Rechenleistung Schaltungen und Methoden realisieren, bei denen eine „Analoge Ausführung" viel zu teuer oder nur mit gigantischem technischem Aufwand oder überhaupt nicht zu verwirklichen wäre. Außerdem lassen sich die Filterwerte (Grenzfrequenz, Filtertyp, Sperrdämpfung usw.) sehr leicht durch geänderte Faktoren im Rechenprogramm umschalten. Auch „selbstanpassende" (= adaptive) Filterschaltungen lassen sich programmieren, Verschlüsselungen oder Entschlüsselungen können vorgenommen werden usw.

12.2. Signalfrequenzen und Sample Rate

Folgende eiserne Grundregel muss eingehalten werden, wenn das alles richtig funktionieren soll. Sie ist unter dem Namen „**Shannon-Bedingung**“ oder „**Nyquist-Kriterium**“ allgemein bekannt:

Handelt es sich beim Eingangssignal um eine

rein sinusförmige Spannung mit der Frequenz f_{ein}, dann muss die Abtastfrequenz (= Sample Rate) mindestens doppelt so hoch gewählt werden wie diese analoge Eingangsfrequenz.

(….sicherer ist natürlich eine mehr als doppelt so hohe Sample Rate….).

Nur dann lässt sich ganz am Ende nach der Digital-Analog-Wandlung und der darauf folgenden Filterung das Analogsignal wieder eindeutig und ohne zusätzliche Fehler rekonstruieren.

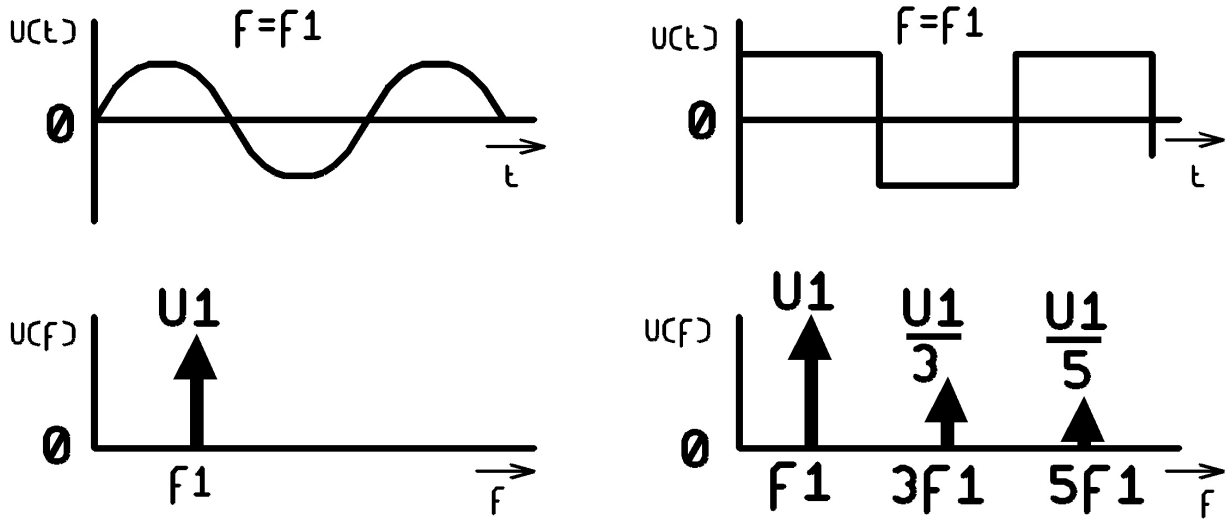
Aber Vorsicht:
Sobald die Kurvenform vom Sinus abweicht, enthält ein solches Signal außer der Grundfrequenz ein ganzes Sortiment an zusätzlichen, sinusförmigen Teilspannungen. Das sind die „Oberwellen“ oder „Harmonische“ und sie stellen z. B. bei Audiosignalen den „Klirrfaktor“ dar. Sie sind stets ganzzahlige Vielfache der Grundfrequenz.

Beispiel:
Oben:
Rein sinusförmige Eingangsspannung auf dem Bildschirm eines Oszilloskops

Unten:
Zugehöriges Schirmbild beim Spectrum-Analyzer

Beispiel:
Oben:
Rechteckspannung auf dem Bildschirm eines Oszilloskops

Unten:
Zugehöriges Schirmbild beim Spectrum-Analyzer



Jetzt müssen wir die Shannon-Bedingung präziser formulieren:

Die Sample Rate muss nun mindestens doppelt so hoch sein wie die höchste vorkommende Einzelfrequenz (oder Oberwelle) im Eingangssignal.

Wer sich etwas mit Oberwellen auskennt, wird da gleich fragen:

„***Die Oberwellen eines nicht sinusförmigen Signals reichen theoretisch bis Unendlich, aber dafür werden ihre Amplituden immer kleiner. Welche Sample Rate soll ich denn nun wählen?“***

Da gibt es nur eine einzige rigorose Möglichkeit. Nämlich einen Kompromiss, bei dem man irgendwo brutal abschneidet und auf die letzten Feinheiten des Analogsignals verzichtet:

Am Eingang der Schaltung (also noch vor dem A-D-Wandler) wird deshalb ein Tiefpass („Anti-Aliasing-Lowpass-Filter“) mit einer bestimmten Grenzfrequenz und einem sehr, sehr steilen Anstieg der Dämpfung im Sperrbereich eingebaut. Dann wählt man eine Sample Rate, die mehr als doppelt so hoch ist wie die Grenzfrequenz des Tiefpasses.

(Beispiel: Bei einem CD-Player beträgt die **Audio-Grenzfrequenz exakt 20 kHz.** Anschießend wird aber mit **44,1 kHz** abgetastet).

Hier steckt man natürlich schon wieder in einer Entscheidungsklemme:
Wählt man die **Tiefpass-Grenzfrequenz sehr niedrig**, dann kommt man mit einer recht niedrigen Sample Rate aus. Das bedeutet weniger Rechenleistung, aber es besteht die Gefahr, dass nun Oberwellen des Eingangssignals fehlen, die noch wichtig für die korrekte Kurvenform des Analogsignals sind.

Wählt man dagegen die **Grenzfrequenz und die Sample Rate höher als erforderlich**, dann stimmt hinterher noch die Kurvenform des Audiosignals. Aber es ergeben sich folgende Kehrseiten:

a) Die höhere Bandbreite beim Eingangssignal erhöht den „**Eigenrauschpegel**“ der Anordnung. Das kann bei Nutzsignalen in der Mikrovolt-Größenordnung recht bitter sein.

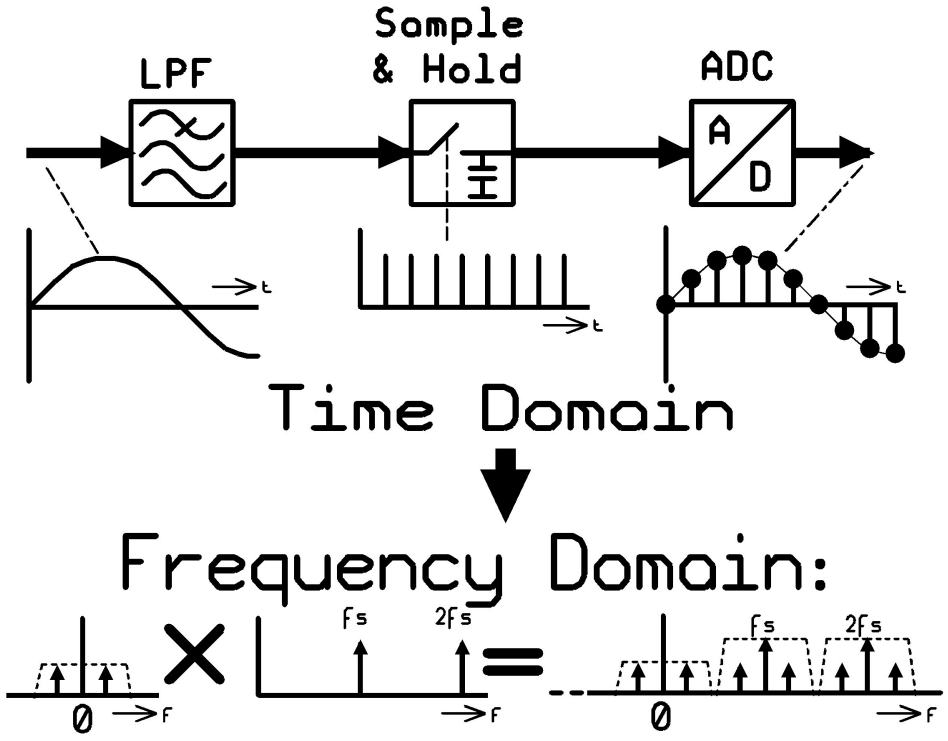
b) Die höhere Sample Rate erfordert (bei Echtzeit-Betrieb) unter Umständen so **hohe Rechenleistungen**, das selbst modernste PCs und Prozessoren an ihre Grenzen stoßen. Da helfen nur echte „DSPs“, die eine andere Hardware-Struktur aufweisen und z. T. mehr als eine Milliarde 32-Bit-Multiplikationen pro Sekunde ausführen können.

(Zur Information: da die Leistungsfähigkeit der Rechner und DSP’s sozusagen stündlich steigt, ist die ausreichende Rechenleistung nicht mehr das Problem. Deshalb wählt man heute oft Lösung b) und tastet viel öfter ab als grundsätzlich notwendig wäre. Der Datenstrom wird dann nach der A-D-Wandlung über **ein „Digitales Decimation-Filter“** geschickt, das die Grenzfrequenz wieder künstlich reduziert und damit auch eine „neue“, niedrigere Sample-Rate zulässt, die nach Shannon nur noch mindestens doppelt so hoch sein muss wie die Informationsbandbreite.)

Nun wollen wir zum Abschluss noch folgende Frage klären:

Was geschieht, wenn die Vorschrift: „Abtastfrequenz immer höher als die doppelte maximale Eingangsfrequenz’“ nicht eingehalten wird?

Dazu sehen wir uns alle beteiligten Signale sowohl in der Time Domain wie auch in der Frequency Domain genauer an:

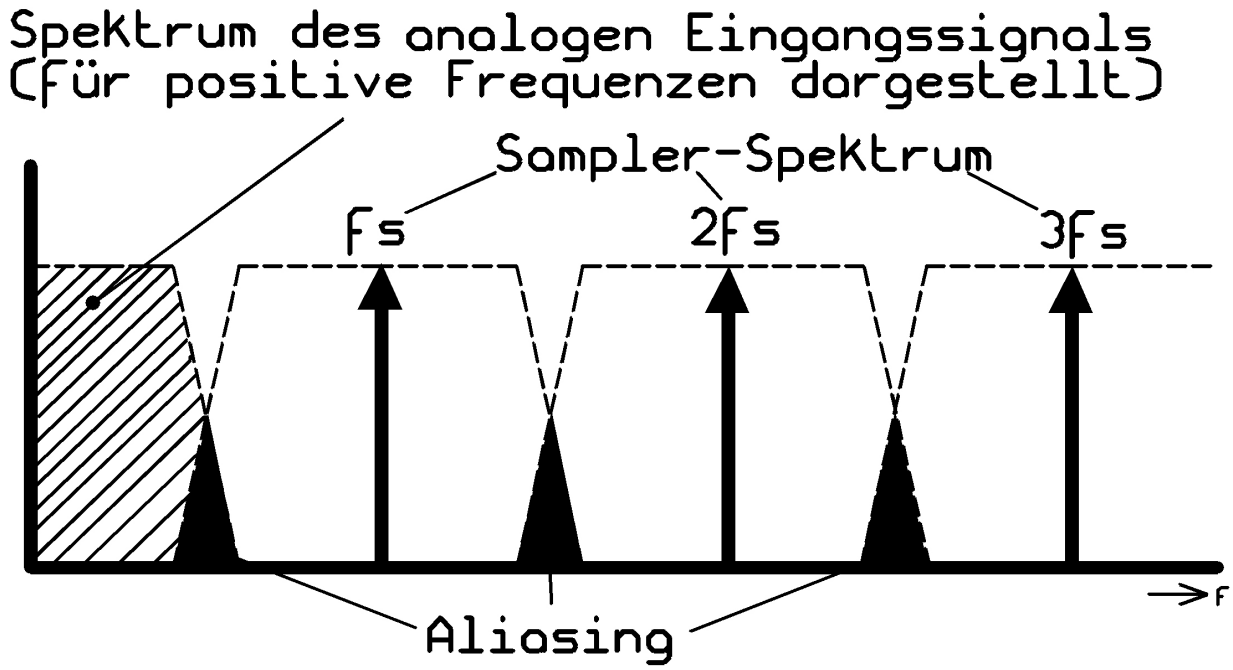


Den Abtastvorgang kann man (im Zeitbereich, also in der **„Time Domain“**) als ***Multiplikation*** des Analog-Eingangssignals mit der Folge der Sample-Pulse betrachten. Folglich erhält man am Ausgang des Samplers wieder eine Impulsfolge, deren Amplituden dem Analogsignal am Eingang folgen.

Im Frequenzbereich (= **Frequency Domain**) ergibt diese **Multiplikation** jedoch die **Bildung von Summen- und Differenzfrequenz der beiden beteiligten Signale.**

Bitte beachten:

Der ***Samplepuls*** besitzt ein Spektrum, bei dem sämtliche gerad- und ungeradzahligen Oberwellen mit derselben Amplitude drinstecken. Jede enthaltene Oberwelle wird also mit dem Analog-Eingangssignal **amplitudenmoduliert**. Folglich findet man als Endergebnis bei jeder Oberwelle nun zusätzlich die **Summen- und Differenzfrequenz aus dieser Oberwelle und der analogen Eingangsfrequenz**. Es entstehen (da das analoge Eingangssignal meist mehr als nur eine einzige Frequenz enthält) folglich so genannte „**untere und obere Seitenbänder" bei der Sample-Grundfrequenz und allen ihren Oberwellen!!**



Jetzt ist gut zu erkennen, was bei zu niedriger Abtastfrequenz geschieht:

Die AM-Spektren beginnen sich zu überschneiden und das führt zu äußerst unangenehmen Verzerrungen. Dieser Vorgang heißt

ALIASING

und die Verzerrungen heißen logischerweise „Aliasing-Verzerrungen". Leider klingen sie nicht nur sehr unangenehm, sondern lassen sich -- wenn dieses Unglück passiert ist -- ***auf keine Weise mehr beseitigen***!

Diesen Effekt soll der Tiefpass vor dem A-D-Wandler verhindern. Deshalb trägt er den treffenden Namen „**ANTI-ALIASING-LOWPASS-FILTER"**.

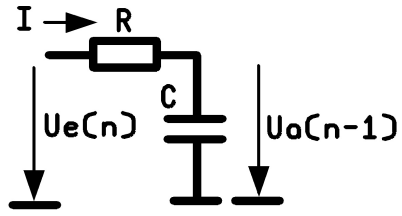
Also nochmals gut einprägen:

Beim Eingangssignal muss ALLES oberhalb der halben Abtastfrequenz unterdrückt werden -- koste es, was es wolle! Sonst gibt es furchtbaren Ärger……

12.3. Praxisprojekt: Ein einfacher Digitaler Filter

12.3.1. Einführung

In einem Digitalen Filter wird der vom Sampler ausgegebene Serielle Datenstrom einer mathematischen Behandlung unterzogen: es werden die verschiedenen Abtastwerte mit Faktoren multipliziert und nach einem bestimmten System zusammenaddiert. Das wirkt sich so aus, als wenn das Analogsignal über eine echte diskrete Filterschaltung gelaufen wäre. Um zu verstehen, wie dabei vorgegangen werden muss, sehen wir uns eine Schaltung an, die Jeder kennt: den simplen **analogen RC-Tiefpass**.



Zu einem bestimmten Zeitpunkt beobachtet man am Kondensator die **Ausgangsspannung $U_{a(n-1)}$**.

Jetzt ändert sich plötzlich die Eingangsspannung, weil die Sampleschaltung den nächsten Abtastwert anliefert. Damit liegt nun in diesem Moment am **Eingang** die **neue Spannung $U_{e(n)}$**, während die **Kondensatorspannung noch den alten Wert $U_{a(n-1)}$** hat.

Wir fragen jetzt nach der ***neuen*** **Ausgangsspannung $U_{a(n)}$**, die sich in der Zwischenzeit bis zum Eintreffen des nächsten Abtastwertes -- also nach dem Zeitunterschied

$\Delta t = t_{sample}$

am Kondensator bilden wird.

$$U_{a(n)} = U_{a(n-1)} + \Delta U_C = U_{a(n-1)} + \frac{\Delta Q}{C} = U_{a(n-1)} + \frac{I \bullet \Delta t}{C} = U_{a(n-1)} + I \bullet \left(\frac{t_{Sample}}{C} \right)$$

$$U_{a(n)} = U_{a(n-1)} + \left(\frac{U_{e(n)} - U_{a(n-1)}}{R} \right) \bullet \left(\frac{t_{Sample}}{C} \right) = U_{a(n-1)} + \left[U_{e(n)} - U_{a(n-1)} \right] \bullet \left(\frac{t_{Sample}}{RC} \right)$$

Jetzt sollte man zuerst „t_{sample}“ durch die Samplefrequenz (**$f_{sample} = 1 / t_{sample}$**) und dann die Zeitkonstante RC durch die Grenzfrequenz des Tiefpasses ausdrücken. Dafür gilt in der Nachrichtentechnik:

$$R = \frac{1}{\omega_{grenz} \bullet C} \Rightarrow \omega_{grenz} = \frac{1}{RC} \Rightarrow 2\pi f_{grenz} = \frac{1}{RC} \Rightarrow \mathbf{f_{grenz} = \frac{1}{2\pi \bullet RC}}$$

Setzt man beide Erkenntnisse bei der vorigen Formel für $U_{a(n)}$ in die ganz rechte Klammer ein, dann erhält man:

$$U_{a(n)} = U_{a(n-1)} + \left[U_{e(n)} - U_{a(n-1)} \right] \bullet \left(\frac{2\pi \bullet f_{grenz}}{f_{sample}} \right)$$

Um etwas Überblick zu schaffen, sollte man noch den ganz rechten Klammerausdruck abkürzen und ihn z. B. mit dem Buchstaben „k“ bezeichnen. Das ergibt nach dem Ausmultiplizieren:

$$U_{a(n)} = U_{a(n-1)} + k \bullet \left(U_{e(n)} - U_{a(n-1)}\right) = U_{a(n-1)} + k \bullet U_{e(n)} - k \bullet U_{a(n-1)}$$

$$\mathbf{U_{a(n)} = k \bullet U_{e(n)} + (1-k) \bullet U_{a(n-1)}}$$

Wir sehen, dass wir nun den ***neuen Eingangswert*** **mit „k", den** ***alten Ausgangswert*** **dagegen mit „(1-k)" multiplizieren** und beide Ergebnisse **addieren** müssen, um die Ausgangsreaktion auf den neuen Eingangswert zu erhalten.

Das ist eine gängige und dauernd nötige Prozedur bei der Digitalen Signalverarbeitung, sie trägt die Kurzbezeichnung „**MAC" (= multiply and accumulate).**

12.3.2. Übungsaufgabe: Untersuchung der RC-Schaltung „von Hand"

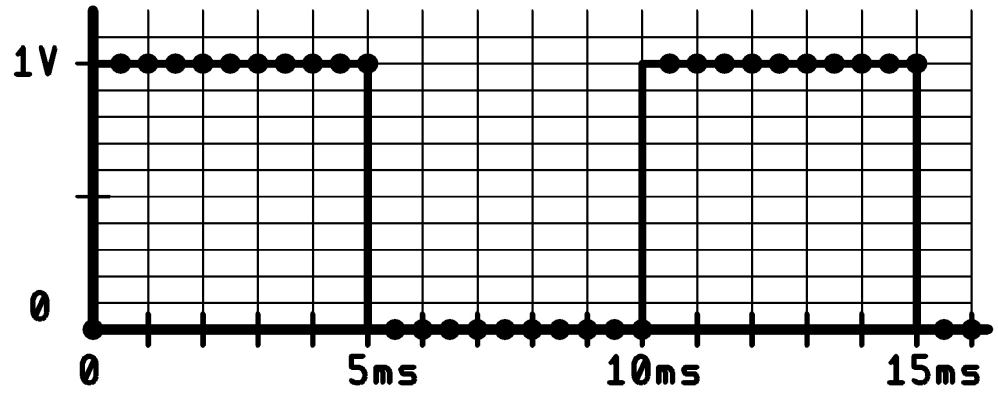
Wir wollen uns eine Tiefpass-Grenzfrequenz von 100 Hz sowie eine Samplefrequenz von 2 kHz vorgeben. Damit ist die Shannon-Bedingung (= Analoge Eingangsfrequenz kleiner als die halbe Samplefrequenz) bequem erfüllt.

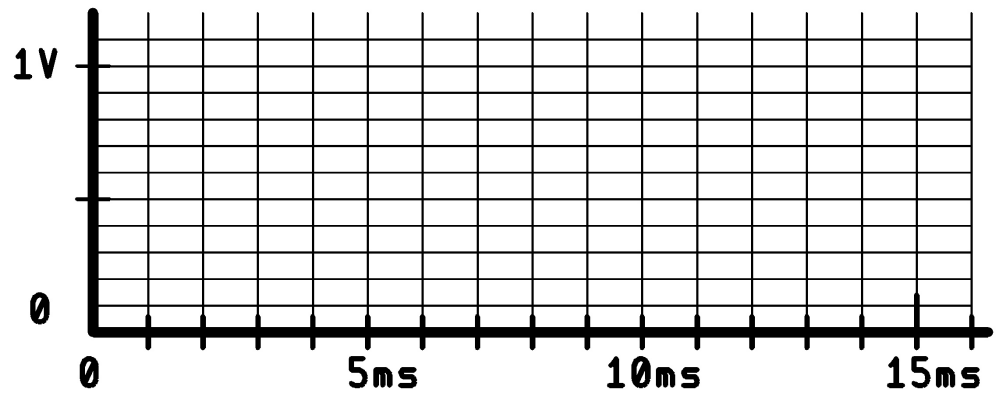
1. Schritt:
Wir berechnen die beiden Faktoren „k" und „(1-k)":

$$k = \frac{2\pi \bullet 100Hz}{2000Hz} = 0{,}314$$

$$(1-k) = 1 - 0{,}314 = 0{,}686$$

2. Schritt:
Der Startwert der Ausgangsspannung bei t = Null sei Null Volt. Beim Zeitpunkt t = Null wird als Eingangssignal eine Rechteckspannung mit der Frequenz f = 100 Hz und den Amplitudenwerten EINS und NULL zugeführt. Wir skizzieren den zeitlichen Verlauf für t = Null bis t = 15 Millisekunden und markieren darin die Abtastungen:





3. Schritt:
Jetzt legen wir eine **Tabelle** für alle beteiligten Größen an und berechnen (….wer möchte: mit ***Excel***, alle anderen mit dem Taschenrechner…..) bei jedem neuen Eingangssample das erzeugte Ausgangssignal am Kondensator.

Zur besseren Kontrolle wurden die ersten 5 Ausgangswerte vorgerechnet. **Bitte vervollständigen Sie die Tabelle und tragen Sie anschließend die berechneten Werte für Ua(n) in das obige Diagramm ein!**

Sample-Nr. **n**	Neue Eingangs-Spannung **Ue(n)**	k * Ue(n) = **0,314 * Ue(n)**	Alte Ausgangs-Spannung **Ua(n-1)**	(1 – k)*Ua(n-1) = **0,686*Ua(n-1)**	Neue Ausgangs-Spannung: Ua(n) = **0.314*Ue(n) +0,686*Ua(n-1)**
0	0	0	0	0	0
1	1	0,314	0	0	0,314
2	1	0,314	0,314	0,215	0,529
3	1	0,314	0,529	0.363	0.677
4	1	0,314	0,677	0,464	0,778
5	1	0,314			
6	1	0,314			
7	1	0,314			
8	1	0,314			
9	1	0,314			
10	1	0,314			
11	0	0			
12	0	0			
13	0	0			
14	0	0			
15	0	0			
16	0	0			
17	0	0			
18	0	0			
19	0	0			
20	0	0			
21	1	0,314			
22	1	0,314			

12.3.3. Umsetzung dieses Filters in ein C-Microcontrollerprogramm und dessen praktischer Test

Das angelegte Analogsignal wird 2000 Mal pro Sekunde abgetastet. Dieser Zeittakt von 500 Mikrosekunden wird durch einen Autoreload-Betrieb von Timer 2 im Interrupt-Betrieb realisiert. In dieser Interrupt-Service-Routine wird auch erneut mit dem ADC gemessen und damit der nächste Samplewert bestimmt.
Das Analog-Eingangssignal wird vom AD-Wandler an Pin ***AN0*** übernommen und als ***Einzelmessung mit dem Messbereich 0….+3V*** abgefragt.

VORSICHT: Bitte den Höchstwert von +3V beim Eingangssignal NICHT überschreiten!!

Digitales Filterprogramm:

```
/*------------------------------------------------------------------
Programmbeschreibung
------------------------------------------------------------------
Name:     dsp_01.c
Aufgabe:
Es handelt sich um einen Digitalen Rekursiver Filter mit der Filterfunktion
"Uan = 0,314*Uen + 0,686*Uan_1"
Um diese mathematischen Operationen mit dem Microcontroller möglichst
schnell und einfach durchzuführen, ist folgender Trick nötig:
Man multipliziert die Faktoren 0,314 bzw. 0,686 mit der Zahl 256,
um auf Werte größer als Null zu kommen. Das ergibt die neuen
Faktoren 80 bzw. 176. Das Filterergebnis wird in einer Integer-Zahl
gespeichert und muss vor der Ausgabe wieder durch 256 geteilt werden.
Allerdings muss dabei sehr darauf geachtet werden, dass bereits alle
Zwischenergebnisse NIE größer als der vorgesehene INT-Speicher (= 16 Bit)
werden, da sonst jeder Überlauf verloren geht und das Endergebnis dadurch
vollkommen falsch ist. Deshalb erfolgt die nötige Division durch 256
bereits bei diesen Zwischenergebnissen.
Das Analogsignal (Amplitude im Bereich von 0....+3V) wird an den Pin AN0
angelegt und 2000mal pro Sekunde abgetastet. Die Ausgabe der gefilterten
Werte erfolgt als Bytes an Port P2.
Dort wird dann der D/A-Wandler angeschlossen.

Datum:    13. 11. 2007
Autor:    G. Kraus

------------------------------------------------------------------
Header, Deklarationen, Konstanten, Prototypen....
------------------------------------------------------------------*/
#include <t89c51ac2.h>    // Registersatz für AT89C51AC3
#include <stdio.h>        // Standard-Eingabe-Ausgabe-Header

sfr ausgang=0xA0;         // Port P2 als Filterausgang

unsigned char Uan_1;      // Vorheriger Ausgangswert
unsigned char Uen;        // Aktueller Eingangswert
unsigned int c;           // 16 Bit-Speicher für Multiplikationen
unsigned char channel;    // Ausgewählter Analog-Channel

void ISR_Timer2(void);    // Prototypen
void init_ADC(void);
void init_Timer2(void);
void wait_ADC_ini(void);
```

```
/*---------------------------------------------------------------------
Hauptprogramm
---------------------------------------------------------------------*/

void main(void)

{    AUXR=AUXR&0xFD;       // auf internes ERAM umschalten = EXTRAM löschen
     init_ADC();           // ADC-Betriebswerte einstellen
     init_Timer2();        // Timer 2 im 16 Bit-Autoreload-Modus

     EA=1;                 // Interrupt-Hauptschalter EIN
     while(1);             // Endlosschleife
}
/*---------------------------------------------------------------------
Zusatzfunktionen
---------------------------------------------------------------------*/
void ISR_Timer2(void)     interrupt 5
{       TF2=0;                          // Überlaufflag löschen
        Uan_1=ausgang;                  // Letzten Ausgangswert umspeichern

        ADCON=ADCON|0x08;     // ADSST setzen, "Single Conversion" starten
        while((ADCON&0x10)!=0x10);      // Warten, bis ADEOC setzt
        ADCON=ADCON&0xEF;               // ADEOC wieder löschen
        Uen=ADDH;                       // Neuen Eingangs-Samplewert holen

        c=((80*Uen)/256)+((176*Uan_1)/256);   // Digitale Filterung

        ausgang=c;                      // Neuen Ausgangswert ausgeben
}

void init_Timer2(void)
{       C_T2=0;      // Timer-Betrieb
        RCLK=0;      // T2 nicht für Serial Port-Takt (Receive) verwenden
        TCLK=0;      //      "    "    "    "       "   (Transmit)
        CP_RL2=0;    // Autoreload bei Überlauf
        ET2=1;       // Timer 2 - Interrupt EIN

        RCAP2L=0x0B;   // Reload-Wert ist 65535-500=65035 (= 0xFE0B)
        RCAP2H=0xFE;   // bei 12 MHz Quarztakt für 500 Mikrosekunden

/* Bei 11,0592MHz Quarztakt wäre der Reloadwert 59936 = 0xEA20 */

        TF2=0;       // Überlauffflag löschen
        TR2=1;       // Timer 2 starten
}


void init_ADC(void)
{       channel=0x00;           // Kanal AN0 gewählt
        ADCF=0x01;              // Pin P1.0 als AN-Channel betreiben
        ADCON=0x20;             // AD-Enable freigeben
        wait_ADC_ini();         // warten, bis ADC initialisiert
        ADCON=ADCON&0xF8;       // Channel-Wahlbits SCH0...SCH2 löschen
        ADCON=ADCON|channel;    // Auf Channel AN0 schalten
}

void wait_ADC_ini(void)      // Wartezeit beim Start des ADC
{       unsigned char x;
        for(x=0;x<10;x++);
}
```

Aufgabe:
Schreiben Sie jetzt ein Programm für einen Sinus-Rechteck-Generator, mit dem Sie diesen Digitalen Filter ansteuern können.
Auf Tastendruck (an Port P0) sollen dort wahlweise Sinusspannungen mit 50 Hz / 100 Hz / 200Hz oder Rechteckspannungen mit 50 Hz / 100 Hz / 200 Hz über Port P2 an den DA-Wandler ausgegeben werden. Außerdem ist eine „Aus-Taste“ vorzusehen. Die Sinuserzeugung erfolgt mit einer Wertetabelle.

Achten Sie besonders darauf, dass der Spitzenwert des Analog-Ausgangssignals einen Wert von +3V nicht überschreitet!

Lösung:
Wir verwenden den Sinus-Rechteck-Generator aus Kapitel 4.4 und **rechnen dabei die Wertetabellen so um , dass der Spitzenwert des Ausgangssignals höchstens +3V beträgt.**

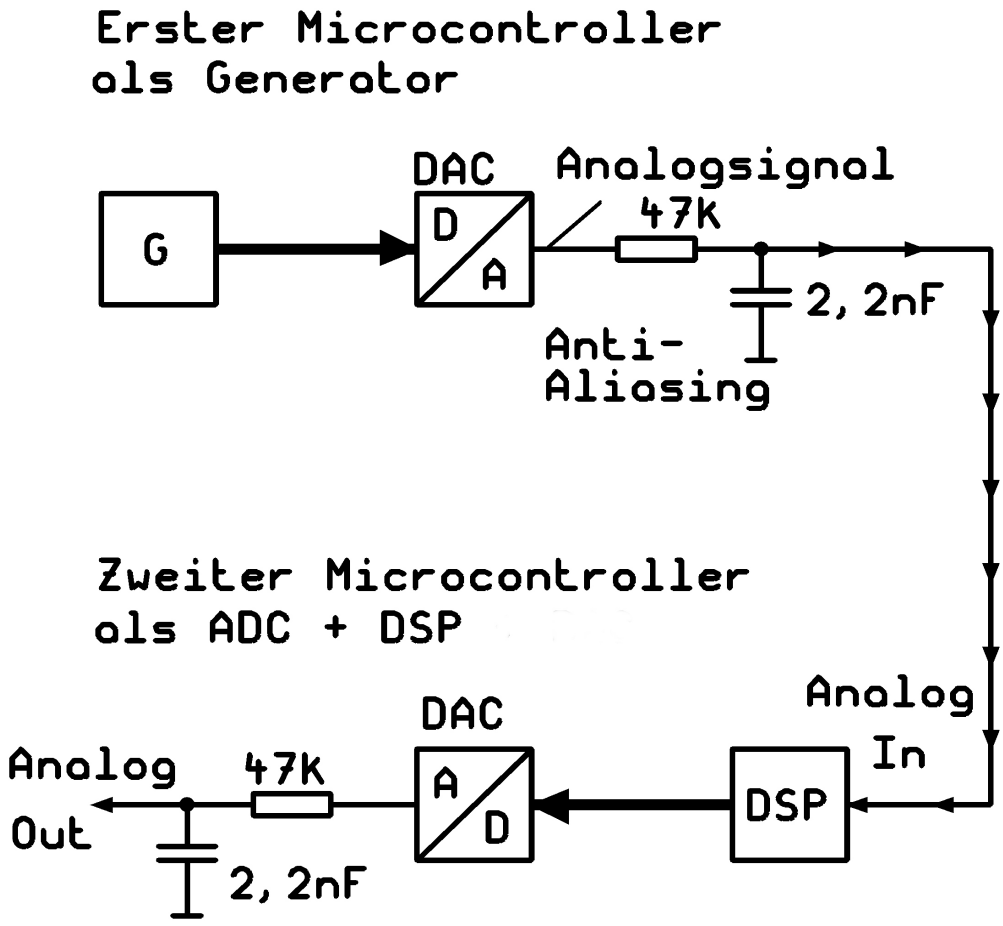
```
// SRG_01.c
// Erstellt am 16.11.2007 durch G. Kraus

/* Ein Druck auf die Tasten P0.1 /  P0.2 / P0.3 erzeugt einen Sinus
mit 50 / 100 / 200 Hz  als Dauersignal.
Ein Druck auf die Taste P0.4 /  P0.5 / P0.6 erzeugt dagegen
ein Rechteck mit 50 / 100 / 200 Hz  als Dauersignal.
Jedes Signal kann nur durch einen Druck auf die Taste P0.0
wieder ausgeschaltet werden. */

#include<t89c51ac2.h>       // Header für AT89C51AC3
#include<stdio.h>

sfr ausgang=0x90;           // Port P1 wird der Digital-Ausgang

sbit Taste_AUS=P0^0;        // Tastendeklaration
sbit Taste_Sin50Hz=P0^1;
sbit Taste_Sin100Hz=P0^2;
sbit Taste_Sin200Hz=P0^3;
sbit Taste_Rechteck_50Hz=P0^4;
sbit Taste_Rechteck_100Hz=P0^5;
sbit Taste_Rechteck_200Hz=P0^6;

bit AUS;                      // Merker-Deklaration
bit Sin50Hz;
bit Sin100Hz;
bit Sin200Hz;
bit Rechteck50Hz;
bit Rechteck100Hz;
bit Rechteck200Hz;

void Rechteckper(int Wert);    // Prototypen
void Sinusper(char Wert);
void Tastenabfrage(void);

code unsigned char Sinus[72]=
     {  127,138,149,160,170,181,191,200,209,217,224,231,237,242,246,250,252,
        253,254,253,252,250,246,242,237,231,224,217,209,200,191,181,170,160,
        149,138,127,116,105,95,84,73,64,54,45,37,30,23,17,12,7,
        4,2,1,0,1,2,4,7,12,17,23,30,37,45,54,64,73,84,95,105,116
     };

unsigned char Sinus_3V[72];    // Sinus-Wertetabelle für max. +3V
```

```
void main(void)
{    AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen
     unsigned char x;
     unsigned int w;

     for (x=0;x<72;x++)          // 3V-Wertetabelle anlegen
     {    w=Sinus[x]*6/10;
          Sinus_3V[x]=w;
     }


     P0=0xFF;                    // Port P0 auf "Einlesen" schalten
     AUS=1;                      // Generator ausschalten

     while(1)                    // Endlosschleife
     {       while(AUS==1)       // Generator ausgeschaltet
             {    Tastenabfrage();
             }

             while(Sin50Hz==1)           // Sinus 50 Hz wird erzeugt
             {    Sinusper(20);
                  Tastenabfrage();
             }

             while(Sin100Hz==1)          // Sinus 100 Hz wird erzeugt
             {    Sinusper(8);
                  Tastenabfrage();
             }

             while(Sin200Hz==1)          // Sinus 200 Hz wird erzeugt
             {    Sinusper(3);
                  Tastenabfrage();
             }

             while(Rechteck50Hz==1)      // Rechteck 50 Hz wird erzeugt
             {    Rechteckper(750);
                  Tastenabfrage();
             }

             while(Rechteck100Hz==1)     // Rechteck 50 Hz wird erzeugt
             {    Rechteckper(375);
                  Tastenabfrage();
             }

             while(Rechteck200Hz==1)     // Rechteck 50 Hz wird erzeugt
             {    Rechteckper(185);
                  Tastenabfrage();
             }

       }
}

void Sinusper(char Wert)                 // Wartezeit nach Wert-Ausgabe
{       char x,y;
        for(x=0;x<72;x++)
        {    ausgang=Sinus_3V[x];
             for(y=0;y<=Wert;y++);
        }
}

void Rechteckper(int Wert)          // Wartezeit bei Rechteck-Erzeugung
{       int y;
        ausgang=0x00;
```

```
        for(y=0;y<=Wert;y++);
        ausgang=152;                // Höchstwert soll +3V sein
        for(y=0;y<=Wert;y++);
}


void Tastenabfrage(void)            // Tastenabfrage
{       if(Taste_AUS==0)            // Aus-Taste gedrückt?
        {    AUS=1;
             Sin50Hz=0;
             Sin100Hz=0;
             Sin200Hz=0;
             Rechteck50Hz=0;
             Rechteck100Hz=0;
             Rechteck200Hz=0;
        }

        if(Taste_Sin50Hz==0)        // Sinus 50Hz-Taste gedrückt?
        {    AUS=0;
             Sin50Hz=1;
             Sin100Hz=0;
             Sin200Hz=0;
             Rechteck50Hz=0;
             Rechteck100Hz=0;
             Rechteck200Hz=0;
        }

        if(Taste_Sin100Hz==0)       // Sinus 100Hz--Taste gedrückt?
        {    AUS=0;
             Sin50Hz=0;
             Sin100Hz=1;
             Sin200Hz=0;
             Rechteck50Hz=0;
             Rechteck100Hz=0;
             Rechteck200Hz=0;
        }

        if(Taste_Sin200Hz==0)         // Sinus 200Hz-Taste gedrückt?
        {    AUS=0;
             Sin50Hz=0;
             Sin100Hz=0;
             Sin200Hz=1;
             Rechteck50Hz=0;
             Rechteck100Hz=0;
             Rechteck200Hz=0;
        }

        if(Taste_Rechteck_50Hz==0)          // Rechteck 50Hz-Taste gedrückt?
        {    AUS=0;
             Sin50Hz=0;
             Sin100Hz=0;
             Sin200Hz=0;
             Rechteck50Hz=1;
             Rechteck100Hz=0;
             Rechteck200Hz=0;
        }

        if(Taste_Rechteck_100Hz==0)    // Rechteck 100Hz-Taste gedrückt?
        {    AUS=0;
             Sin50Hz=0;
             Sin100Hz=0;
```

```
            Sin200Hz=0;
            Rechteck50Hz=0;
            Rechteck100Hz=1;
            Rechteck200Hz=0;
        }

        if(Taste_Rechteck_200Hz==0)      // Rechteck 200Hz-Taste gedrückt?
        {   AUS=0;
            Sin50Hz=0;
            Sin100Hz=0;
            Sin200Hz=0;
            Rechteck50Hz=0;
            Rechteck100Hz=0;
            Rechteck200Hz=1;
    }
}
```

Verbinden Sie nun den Analogausgang Ihres „Generators“ mit dem Eingangspin AN0 des zweiten Controllerboards entsprechend dem Übersichtsschaltplan auf der folgenden Seite und prüfen Sie die korrekte Funktion der Anordnung mit dem Oszilloskop. Messen Sie dazu die Signale an den Ein- und Ausgängen der beiden Tiefpässe und tragen Sie die Verläufe im folgenden Schaltplan ein..



Aufgabe:
Stellen Sie jetzt den Filter auf eine **Grenzfrequenz von 200 Hz** sowie die **Abtastfrequenz auf 8 KHz** um. Ändern Sie dazu Ihr Filterprogramm und prüfen Sie die geänderte Filterwirkung mit dem Oszilloskop an der praktischen Schaltung.

Lösung: Digitales Filterprogramm, umgestellt auf 8 KHz Abtastfrequenz und 200 Hz Grenzfrequenz

1. Schritt:
Wir berechnen die beiden Faktoren „k“ und „(1-k)“:

$$\mathbf{k = \frac{2\pi \bullet 200Hz}{8000Hz} = 0{,}157}$$

$$\mathbf{1 - k = 1 - 0{,}157 = 0{,}843}$$

Multipliziert man (wie beim ersten Filter besprochen) nun diese beiden Faktoren mit 255, dann ergeben sich die **Werte 40 bzw. 215 (**sie werden später durch 255 dividiert, dann stimmt alles wieder).
Diese beiden neuen Werte verwendet man in der Interrupt-Service-Routine zur Berechnung des neuen Ausgangswertes.

2. Schritt:
Wir sorgen dafür, dass der Timer 2 bereits nach 125 Mikrosekunden (= Periodendauer von 8 kHz) überläuft.

Der dafür erforderliche Reloadwert ist **65535 – 125 = 65410.**

Das entspricht einer Hex-Zahl von **0xFF82** , die in das RCAP2L- und RCAP2H-Register geladen werden muss.

Programm:

```
/*------------------------------------------------------------------
Programmbeschreibung
------------------------------------------------------------------
Name:     dsp_02.c
Aufgabe:
Es handelt sich um einen Digitalen Rekursiver Filter mit der Filterfunktion
"Uan = 0,157*Uen + 0,843*Uan_1"
Um diese mathematischen Operationen mit dem Microcontroller möglichst
schnell und einfach durchzuführen, ist folgender Trick nötig:
Man multipliziert die Faktoren 0,157 bzw. 0,843 mit der Zahl 255,
um auf Werte größer als Null zu kommen. Das ergibt die neuen
Faktoren 40 bzw. 215. Das Filterergebnis wird in einer Integer-Zahl
gespeichert und muss vor der Ausgabe wieder durch 256 geteilt werden.
Allerdings muss dabei sehr darauf geachtet werden, dass bereits alle
Zwischenergebnisse NIE größer als der vorgesehene INT-Speicher (= 16 Bit)
werden, da sonst jeder Überlauf verloren geht und das Endergebnis dadurch
vollkommen falsch ist. Deshalb erfolgt die nötige Division durch 256
bereits bei diesen Zwischenergebnissen.
Das Analogsignal (Amplitude im Bereich von 0....+3V) wird an den Pin AN0
angelegt und 8000mal pro Sekunde abgetastet. Diesen Zeittakt erzeugt Timer
2 im Reloadbetrieb mit Interrupt. Der dafür erforderliche Reloadwert ist
(bei 12 MHz Quarztakt)  65410 = 65535 - 125 (als Hexzahl: 0xFF82).
Die Ausgabe der gefilterten Werte erfolgt als Bytes an Port P2. Dort wird
dann der D/A-Wandler angeschlossen.
```

```
Datum:    13. 11. 2007
Autor:    G. Kraus
------------------------------------------------------------------------
Header, Deklarationen, Konstanten, Prototypen....
------------------------------------------------------------------------*/
#include <t89c51ac2.h>    // Registersatz für AT89C51AC3
#include <stdio.h>        // Standard-Eingabe-Ausgabe-Header

sfr ausgang=0xA0;         // Port P2 als Filterausgang

unsigned char Uan_1;      // Vorheriger Ausgangswert
unsigned char Uen;        // Aktueller Eingangswert
unsigned int c;           // 16 Bit für Multiplikationen bereitstellen
unsigned char channel;    // Ausgewählter Analog-Channel

void ISR_Timer2(void);    // Prototyp
void init_ADC(void);
void init_Timer2(void);
void wait_ADC_ini(void);
/*------------------------------------------------------------------------
Hauptprogramm
------------------------------------------------------------------------*/
void main(void)

{  AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen
   init_ADC();
   init_Timer2();

   EA=1;              // Interrupt-Hauptschalter EIN
   while(1);          // Endlosschleife
}
/*------------------------------------------------------------------------
Zusatzfunktionen
------------------------------------------------------------------------*/
void ISR_Timer2(void)     interrupt 5
{    TF2=0;                    // Überlaufflag löschen
     Uan_1=ausgang;            // Letzten Ausgangswert umspeichern

     ADCON=ADCON|0x08;       // ADSST setzen, "Single Conversion" starten
     while((ADCON&0x10)!=0x10);   // Warten, bis ADEOC setzt
     ADCON=ADCON&0xEF;            // ADEOC wieder löschen
     Uen=ADDH;                    // Neuen Eingangs-Samplewert holen

     c=((40*Uen)/256)+((215*Uan_1)/256);   // Digitale Filterung

     ausgang=c;                   // Neuen Ausgangswert ausgeben

}

void init_Timer2(void)
{    C_T2=0;                   // Timer-Betrieb
     RCLK=0;        // Timer 2 nicht für Serial Port - Takt (Receive)
     TCLK=0;        //      "    "    "    " "   (Transmit) "
     CP_RL2=0;                 // Autoreload bei Überlauf
     ET2=1;                    // Timer 2 - Interrupt EIN
     RCAP2L=0x82;     // Reload-Wert ist 65410 (gibt 125 Mikrosekunden )
     RCAP2H=0xFF;     // (= 0xFF82) bei 12 MHz Quarztakt

     /* Bei 11,0592 MHz Quarztakt wäre der Reloadwert 60282 = 0xEB7A */

     TF2=0;                // Überlauffflag löschen
     TR2=1;                // Timer 2 starten
}
```

```
void init_ADC(void)
{    channel=0x00;         // Kanal AN0 gewählt
     ADCF=0x01;            // Pin P1.0 als AD-Channel betreiben
     ADCON=0x20;             // AD Enable freigeben
     wait_ADC_ini();         // warten, bis ADC initialisiert
     ADCON=ADCON&0xF8;       // SCH0...SCH2 löschen
     ADCON=ADCON|channel;    // Auf Channel AN0 schalten
}

void wait_ADC_ini(void)      // Wartezeit beim Start des ADC
{    unsigned char x;
     for(x=0;x<10;x++);
}
```

12.4. FIR – Filter = Nichtrekursive Filter

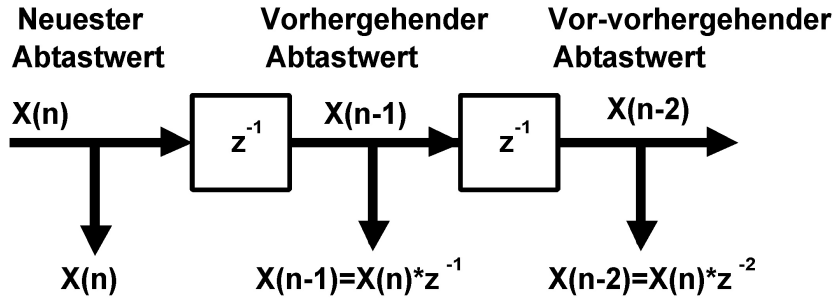
In der Praxis benötigt man wesentlich aufwendigere Filter als den eben behandelten einfachen Tiefpass. Ihr Entwurf erfordert Kenntnisse der höheren Mathematik (= Komplexe Rechnung und Integralrechnung), da hier die „Z-Transformation“ verwendet wird.
Deren Sinn auf Aufgabe wird aber klarer, wenn man (in etwas vereinfachter Form) mit folgender Tatsache arbeitet:

Jedes Mal, wenn im Blockschaltbild ein Klötzchen mit der Bezeichnung

$$z^{-1}$$

auftaucht, muss an seinem Ausgang der (bereits vergangene, aber hoffentlich irgendwo gespeicherte) vorige Samplewert benützt werden!

So könnte dann beim Filterentwurf ein Teil des Schaltbildes aussehen:

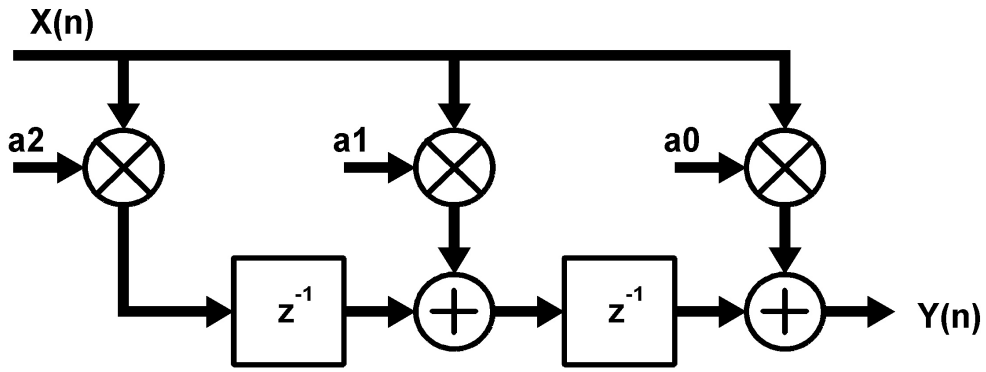


Mit diesem Wissen können wir schon den prinzipiellen Aufbau der „FIR“-Filter verstehen.

Erläuterung:
„**FIR**“ bedeutet „**finite impulse response**“ (also: „**endliche Impulsantwort**“).Solche Filter antworten auf ein Eingangssignal in Form eines Nadelimpulses immer mit einem Ausgangssignal, **das wieder bis auf Null abklingt. Diese Filter können also nie schwingen und sind grundsätzlich stabil.**

Man erkennt sie stets an der „Einbahnstraßenstruktur vom Eingang in Richtung Ausgang“ -- es gibt NIEMALS eine Rückführung des Ausgangssignals in Richtung Eingang. Eine weitere Bezeichnung dieser Filterart lautet: „**Nicht-Rekursive Filter**“.

Beispiel:



Für das Ausgangssignal gilt:

$$Y_{(n)} = a_0 \bullet X_{(n)} + a_1 \bullet X_{(n)} \bullet z^{-1} + a_2 \bullet X_{(n)} \bullet z^{-2}$$

$$Y_{(n)} = a_0 \bullet X_{(n)} + a_1 \bullet X_{(n-1)} + a_2 \bullet X_{(n-2)}$$

Klammert man aus der rechten Seite der Gleichung den Faktor „X(n)“ aus, so lässt sich anschließend sehr leicht die **Übertragungsfunktion (= Transfer Function)** darstellen:

$$\mathbf{Y_{(n)} = X_{(n)} \bullet \left[a_0 + a_1 \bullet z^{-1} + a_2 \bullet z^{-2} \right]}$$

$$\Rightarrow \frac{\mathbf{Y_{(n)}}}{\mathbf{X_{(n)}}} = \mathbf{H(z) = a_0 + a_1 \bullet z^{-1} + a_2 \bullet z^{-2}}$$

(Ein kleiner Hinweis: ein Filter ist dann absolut stabil, wenn in dieser Übertragungsfunktion der Ausdruck „$\mathbf{z^{-1}}$“ nur im **Zähler** vorkommt).

Vorteile der FIR-Filter: a) Absolute Stabilität, also keinerlei Schwingneigung

b) Wird der Filter symmetrisch aufgebaut (also wenn a0 = a2), dann weist er einen linearen Phasengang und damit eine „konstante Gruppenlaufzeit“ auf.

c) Nicht so empfindlich gegen „Streuungen“ oder Lässigkeiten beim Entwurf, also z. B. Auf- oder Abrundungen oder Vereinfachungen im Programm

Nachteile der FIR-Filter: Nicht so steile Filterflanken bei gleichem Filtergrad wie IIR-Filter.

Einige Hinweise zum Entwurf solcher FIR-Filter.

Neben umfangreichen Anleitungen (..die aber gute Mathematikkenntnisse voraussetzen) gibt es immer mehr Programme im Internet, die einem diese Arbeit abnehmen. Trotzdem sollte man einige grundsätzliche Dinge wissen:

1) Die Summe der Koeffizienten (a0 / a1 / a2…) ergibt die **Gleichspannungsverstärkung** des Filters.

2) Beim Entwurf darf **nicht die gewünschte Grenzfrequenz** verwendet werden, sondern nur die „**normierte Grenzfrequenz Fg**“. Darunter versteht man

das Verhältnis der gewünschten echten Grenzfrequenz zur Samplefrequenz

und das muss immer deutlich unter 0,5 liegen (…da war doch was mit der „Shannon-Bedingung“…).

3) Der **Filtergrad sollte immer ungerade sein**, denn dann hat man automatisch bei der **halben Samplefrequenz eine Nullstelle im Frequenzgang** (und das unterstützt das Einhalten der Shannonbedingung).

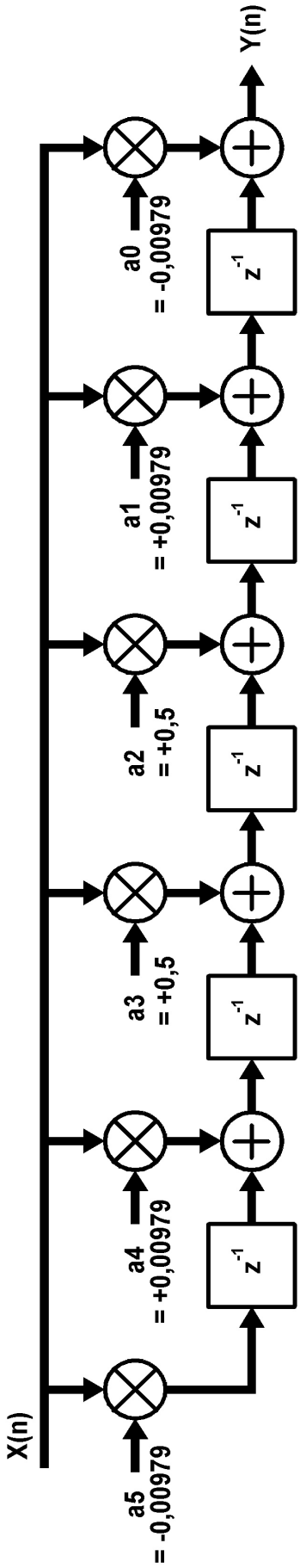
Beispiel:

Gegeben sei ein Filter für Fg = 0,25 und Filtergrad N = 5 mit den Filterkoeffizienten

a0 = a5 = -0,00979 a1 = a4 = +0,00979 a2 = a3 = +0,5

Berechnen Sie die „echte Grenzfrequenz, wenn mit 8 kHz abgetastet wird und skizzieren Sie die Schaltung.

Lösung:


X(n)
Y(n)
a5
= -0,00979
a4
= +0,00979
a3
= +0,5
a2
= +0,5
a1
= +0,00979
a0
= -0,00979
z-1
z-1
z-1
z-1
z-1

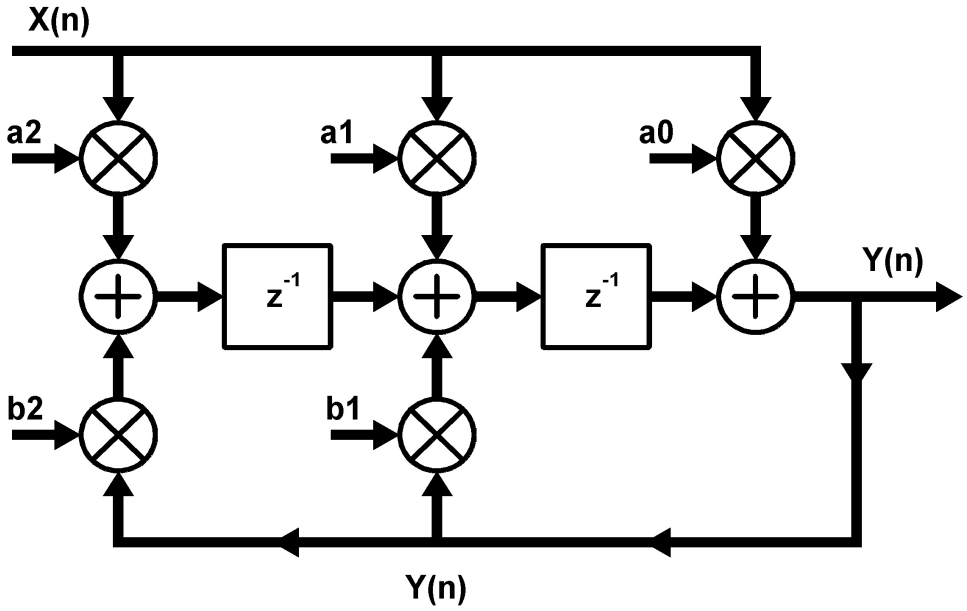
12.5. IIR-Filter = Rekursive Filter

Sie besitzen immer eine „Rückführung vom Ausgang zum Eingang“ und brauchen deshalb wesentlich mehr Sorgfalt beim Entwurf -- sie neigen bei der kleinsten Schlamperei zum Schwingen (..das kennt man in der Analogtechnik von rückgekoppelten Verstärkern…). Dazu reicht schon ein zu frühes Aufrunden oder Abrunden oder Abbrechen bei langen Zahlen!.

Dafür erhält man bei gleichem Filtergrad wesentlich steilere Flanken und damit bessere Filterwirkungen gegenüber den vorhin besprochenen FIR-Filtern.

Übrigens: „IIR“ heißt „Infinite Impulse Response“ = „unendlich lange Impulsantwort“. Die Antwort auf einen Nadelimpuls am Eingang klingt also niemals ganz bis auf Null ab, sondern wird (hoffentlich“) nur immer kleiner. Außerdem liefern diese komischen Vögel bereits eine Antwort am Ausgang, bevor noch der Eingangsimpuls angelegt wird…..also eine ganz verrückte Rasse, die man vorsichtig behandeln muss.

Beispiel für eine solche Filteranordnung:



$$Y_{(n)} = a_0 \bullet X_{(n)} + a_1 \bullet X_{(n)} \bullet z^{-1} + a_2 \bullet X_{(n)} \bullet z^{-2} + b_1 \bullet Y_{(n)} \bullet z^{-1} + b_2 \bullet Y_{(n)} \bullet z^{-2}$$

$$\Rightarrow Y_{(n)} \bullet \left(1 - b_1 \bullet z^{-1} - b_2 \bullet z^{-2}\right) = X_{(n)} \bullet \left(a_0 + a_1 \bullet z^{-1} + a_2 \bullet z^{-2}\right)$$

Daraus folgt die Übertragungsfunktion:

$$\frac{Y_{(n)}}{X_{(n)}} = \frac{\left(a_0 + a_1 \bullet z^{-1} + a_2 \bullet z^{-2}\right)}{\left(1 - b_1 \bullet z^{-1} - b_2 \bullet z^{-2}\right)}$$

Wie man sieht, kann der Nenner der Transfer Function möglicherweise Null werden -- speziell dann, wenn die Koeffizienten b1 und b2 positive Werte aufweisen. **Dann schwingt die Anordnung!**
Außerdem müssen wir zur Abwechslung nun auch die vorausgegangenen ***Ausgangswerte*** für die Berechnung speichern und heranziehen.

Man ahnt schon: …bei sehr aufwendigen Strukturen muss man ein ganzes Sortiment an Eingangs- UND Ausgangswerten dauernd bereithalten…

Einen solchen Filter haben wir in einfacher Form schon kennengelernt: es war die Realisierung des RC-Tiefpasses in Kapitel 3. Erinnern Sie sich noch? Dort galt:

$$U_{a(n)} = k \bullet U_{e(n)} + (1-k) \bullet U_{a(n-1)}$$

und die echte Grenzfrequenz war 100 Hz bei einer Samplefrequenz von 2 kHz.

Dafür erhielten wir die Koeffizienten

k = 0,314 und **(1-k) = 0,686**

Aufgabe:

a) Leiten Sie die Übertragungsfunktion dieses Filters mit diesen beiden Koeffizienten ab.

b) Zeichnen Sie die Schaltung dieses Filters und tragen Sie die korrekten Werte ein.

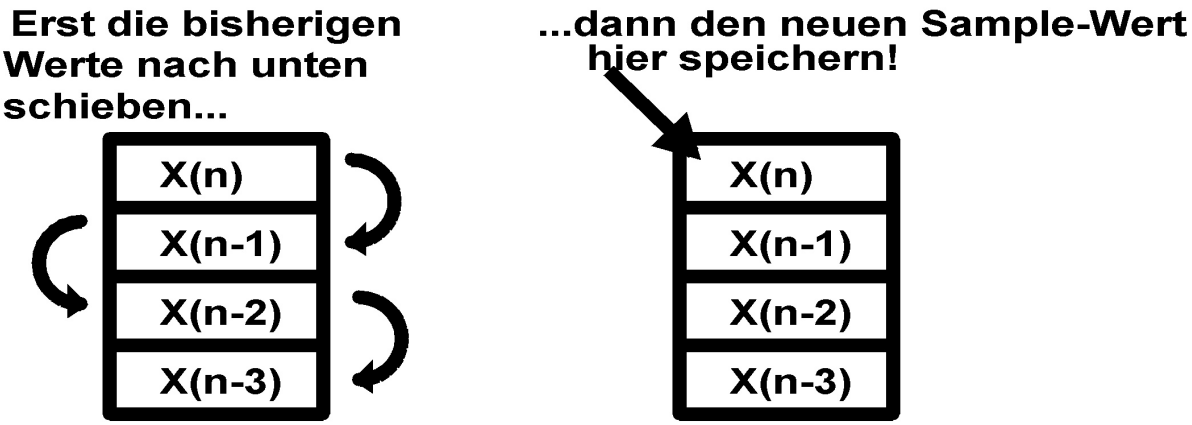
Lösung:

12.6. Organisation der Abtastwerte im Speicher

Wie besprochen, muss man nicht nur den aktuellen, neu einlaufenden Eingangswert X_n abspeichern, sondern auch vorausgegangene Eingangs- oder Ausgangswerte. Beim Einlaufen des nächsten Samples ist nämlich eine ***Aktualisierung*** nötig (= der älteste Wert wird verworfen, aber der neueste Wert an die Spitze der Liste gesetzt. Zusätzlich müssen dann alle übrigen Werte in der Liste „umnummeriert" werden)

a)
Bei einer kleinen Anzahl an gespeicherten Informationen geht es am schnellsten, wenn man nach der Berechnung des Ausgangssignals gleich alle Werte umspeichert und an der Spitze Platz für den nächsten Wert schafft.

Beispiel für die Verarbeitung von 4 Samples:



b)
Bei einer größeren Anzahl von Samples dauert das viel zu lange, deshalb wird folgender Weg gewählt:

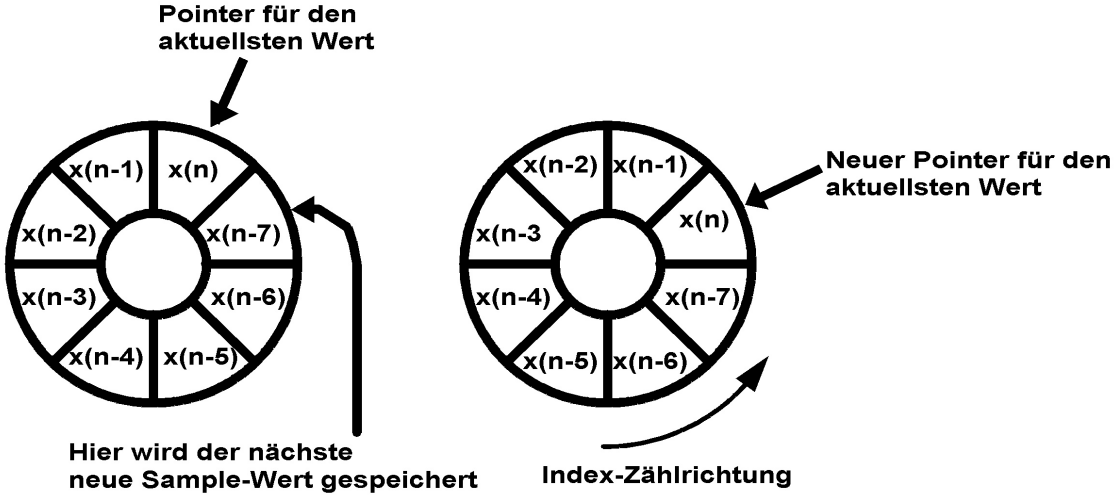
Man arbeitet immer mit einer Sample-Anzahl, die einer ***Zweierpotenz*** entspricht (= 2, 4, 8, 16, 32, 64, 128….1024, 2048, 4096…).
Dann organisiert man den Samplespeicher als ***Array in Ringform*** und speichert die Adresse des „Anfangs der Kette" als „***POINTER***" in einer Variablen. Dieser Pointer zeigt damit immer auf den neuesten Samplewert.

Läuft nun der nächste Samplewert ein, dann wird er auf den

letzten Platz des Ringes gespeichert!

Damit wird der älteste Wert überschrieben. Folglich muss man nun den Pointer auf diesen neuesten Samplewert (= neue Startadresse des Ring-Arrays) umstellen und kann dann mit der nächsten Berechnung anfangen.

Beispiel für einen Filter, der immer 8 Werte verarbeitet:



13. Kurzer Überblick: Das Serial Port Interface (SPI)

Hier handelt es sich um eine spezielle Baugruppe, mit der eine vollständige Kommunikation zwischen mehreren Microcontrollern oder Peripheriegeräten eingerichtet werden kann.

Der Kern der Sache ist ein Dreileiter-Bussystem, das alle Geräte miteinander verbindet.

Übersichts-Schaltplan:

Serial Port Interface (SPI)

The Serial Peripheral Interface Module (SPI) allows full-duplex, synchronous, serial communication between the MCU and peripheral devices, including other MCUs.

Features

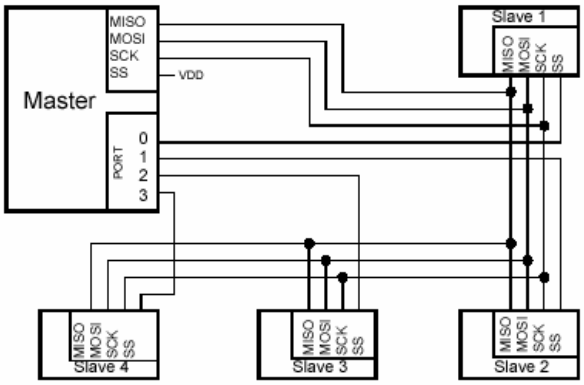
Features of the SPI Module include the following:

- Full-duplex, three-wire synchronous transfers
- Master or Slave operation
- Six programmable Master clock rates in master mode
- Serial clock with programmable polarity and phase
- Master Mode fault error flag with MCU interrupt capability

Signal Description

Figure 43 shows a typical SPI bus configuration using one Master controller and many Slave peripherals. The bus is made of three wires connecting all the devices.

Figure 43. SPI Master/Slaves Interconnection



The Master device selects the individual Slave devices by using four pins of a parallel port to control the four $\overline{SS}$ pins of the Slave devices.

Wer sich damit genauer befassen möchte: Siehe ab Seite 83 des Controller-Datenblattes!

14. PCA = „Programmable Counter Array“

14.1. Überblick

Hier haben wir eine große Maschine vor uns, deren Möglichkeiten weit über die von Timer 0 / 1 / 2 hinausgehen.

Im Prinzip handelt es sich um eine Anordnung, bei der
ein spezieller Timer / Counter (als „PCA-Timer“ bezeichnet)

die Zeitbasis für 5 weitere Baugruppen (= Capture / Compare Modules) bildet

---
Diese **„Zeitbasis**“ kann mit 4 unterschiedlichen Taktsignalen (= „Clocks“) gespeist werden:

a) **PCA-Clock / 6**

b) **PCA-Clock / 2**

c) **Timer 0 Overflow**

d) **External Input Signal vom Pin „ECI“ (= Portpin P1^2)**

Jeder einzelne **Capture / Compare – Modul** kann auf folgende Betriebsarten („Modes“) umgeschaltet werden:

a) rising and / or falling edge capture

b) Software Timer

c) High-Speed Output

d) Pulse Width Modulator

Bei den Betriebsarten a) bis c) ist **Interrupt-Betrieb** möglich.

Zusätzlich kann Modul 4 auch noch als „WatchDog Timer“ eingesetzt werden.

Und so liest sich die genauere Info im Datenblatt:

Each one of the five compare/capture modules has six possible functions. It can perform:

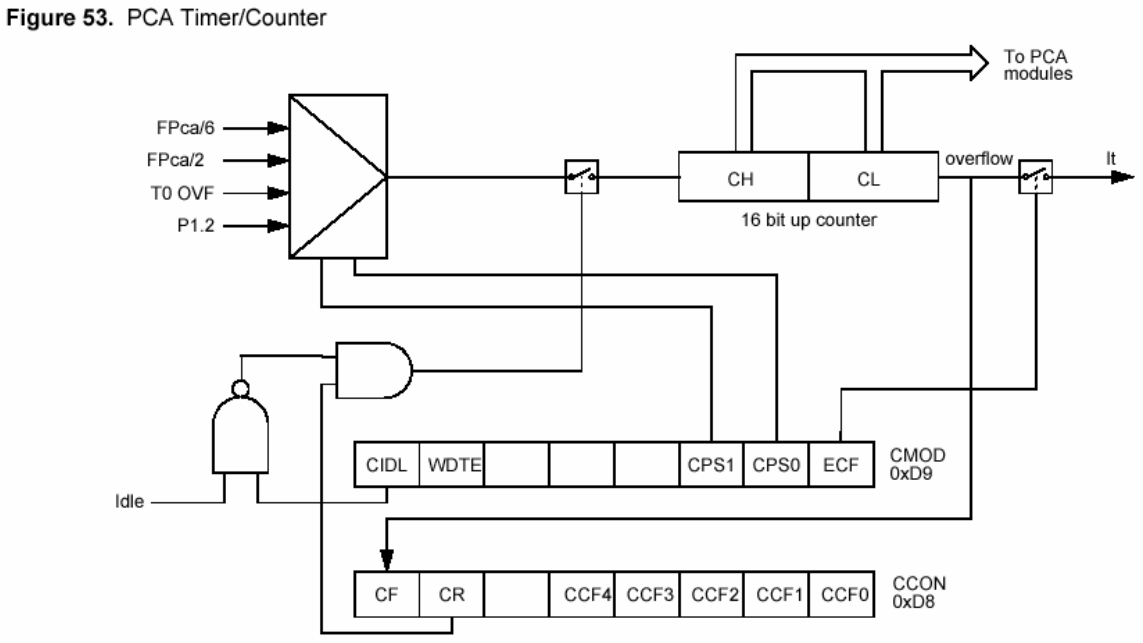
- 16-bit Capture, positive-edge triggered
- 16-bit Capture, negative-edge triggered
- 16-bit Capture, both positive and negative-edge triggered
- 16-bit Software Timer
- 16-bit High Speed Output
- 8-bit Pulse Width Modulator.

In addition module 4 can be used as a WatchDog Timer.

14.2. Der Umgang mit der Zeitbasis (= PCA-Timer)

Dazu gehen wir erst mal den Übersichtsschaltplan durch:

Figure 53. PCA Timer/Counter



Bitte genau hinschauen:

Links oben im Eck sehen wir die vier möglichen **Eingangs-Taktsignale**

a) PCA-Clock / 6

b) PCA-Clock / 2

c) Timer 0 Overflow

d) External Input Signal vom Pin „ECI“ (= Portpin P1^2)

Eines davon muss durch die **beiden Bits “CPS0” und “CPD1” im CMOD-Register** ausgewählt werden.

Hinter diesem Eingangsumschalter folgt ein „**Hauptschalter**“, der per Software geschlossen werden muss --
sonst läuft gar nix!

Dazu wird das **„Counter Run Bit CR“ im Register CCON gesetzt** und das „**Counter Idle Bit CIDL“ im CMOD-Register gelöscht**.

Nun kann das Taktsignal den 16 Bit-Aufwärtszähler erreichen. Der augenblickliche Zählerstand (= Inhalt von CL- und CH-Register) wird dabei an die fünf PCA-Module weitergereicht.

Läuft nun der Zähler über, dann wird

a) das **Counterflag CF gesetzt** . Es muss per Software wieder gelöscht werden.

b) Falls das **ECF-Flag in Register CMOD gesetzt ist**, wird zusätzlich ein **Interrupt ausgelöst**.

Hinweis
Im **CMOD-Register** findet sich noch das „**Watchdog Timer Enable Bit WDTE**“, im **CCON-Register** dagegen die **Flags der angesteuerten fünf PCA-Module**. Sie werden bei erfolgreichem Capture- oder Comparevorgang gesetzt und müssen per Software wieder gelöscht werden.

14.3. Einstellung der Betriebsart bei einem PCA-Modul

Table 54. CCAPMn Registers

CCAPM0 (S:DAh)
CCAPM1 (S:DBh)
CCAPM2 (S:DCh)
CCAPM3 (S:DDh)
CCAPM4 (S:DEh)

Jeder der fünf Module muss vom Anwender jeweils über ein eigenes „**Capture-Compare-Modus-Register CCAPMn**“ programmiert werden.

Das ist natürlich etwas anstrengend, denn die folgende Tabelle muss man dazu ganz genau verstanden haben:

PCA Compare/Capture Module n Mode registers (n=0..4)

7	6	5	4	3	2	1	0
-	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn

Bit Number	Bit Mnemonic	Description
7	-	**Reserved** The Value read from this bit is indeterminate. Do not set this bit.
6	ECOMn	**Enable Compare Mode Module x bit** Clear to disable the Compare function. Set to enable the Compare function. The Compare function is used to implement the software Timer, the high-speed output, the Pulse Width Modulator (PWM) and the WatchDog Timer (WDT).
5	CAPPn	**Capture Mode (Positive) Module x bit** Clear to disable the Capture function triggered by a positive edge on CEXx pin. Set to enable the Capture function triggered by a positive edge on CEXx pin
4	CAPNn	**Capture Mode (Negative) Module x bit** Clear to disable the Capture function triggered by a negative edge on CEXx pin. Set to enable the Capture function triggered by a negative edge on CEXx pin.
3	MATn	**Match Module x bit** Set when a match of the PCA Counter with the Compare/Capture register sets CCFx bit in CCON register, flagging an interrupt.
2	TOGn	**Toggle Module x bit** The toggle mode is configured by setting ECOMx, MATx and TOGx bits. Set when a match of the PCA Counter with the Compare/Capture register toggles the CEXx pin.
1	PWMn	**Pulse Width Modulation Module x Mode bit** Set to configure the module x as an 8-bit Pulse Width Modulator with output waveform on CEXx pin.
0	ECCFn	**Enable CCFx Interrupt bit** Clear to disable CCFx bit in CCON register to generate an interrupt request. Set to enable CCFx bit in CCON register to generate an interrupt request.

Reset Value = X000 0000b

Eine kurze Erläuterung der einzelnen Bits eines solchen CCAPn-Registers folgt auf der nächsten Seite!

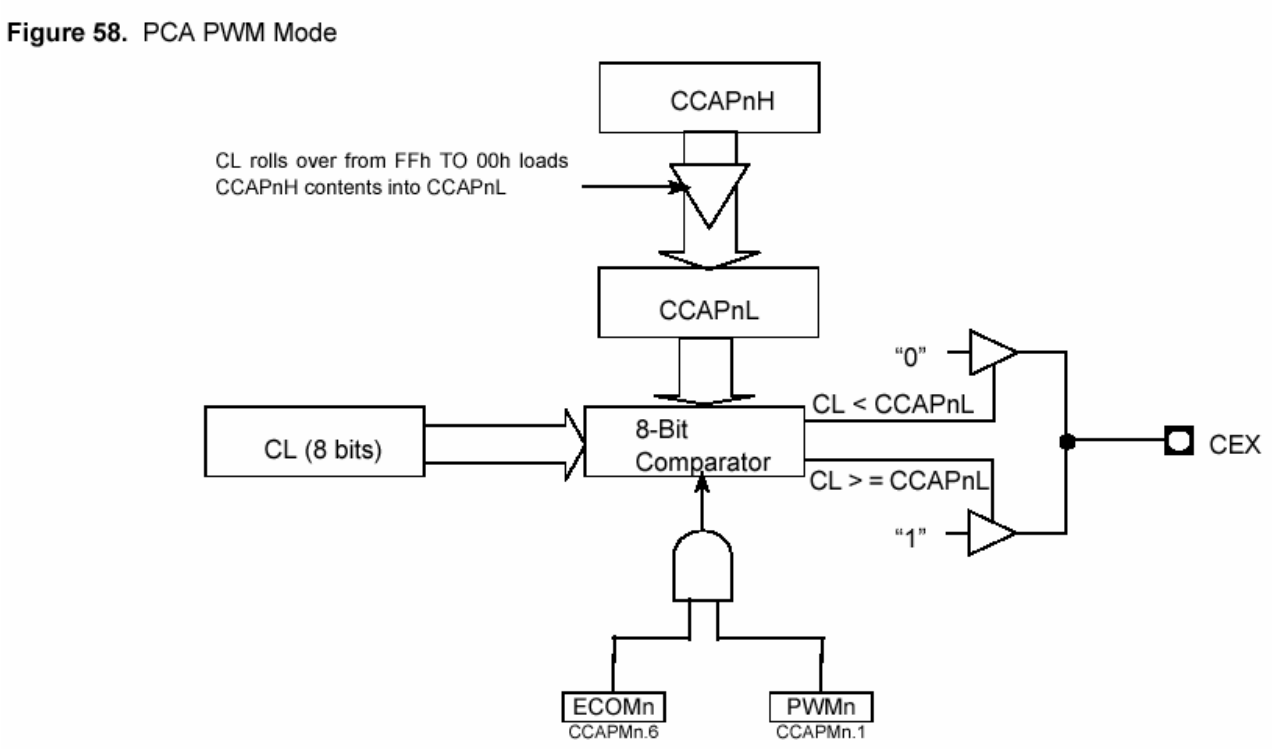
Bit 7	Reserviert für andere Zwecke, also bitte nicht benutzen.
Bit 6 = ECOMn	Muss gesetzt sein, um mit der **Compare-Funktion** zu arbeiten
Bit 5 = CAPPn	Der Compare-Vorgang wird durch eine **positive Flan**ke am entsprechenden „Externen Capture Pin CEXn“ des Controllers ausgelöst
Bit 4 = CAPNn	Der Compare-Vorgang wird durch eine **negative Flanke** am entsprechenden „Externen Capture Pin CEXn“ des Controllers ausgelöst
Bit 3 = MATn	Dieses „Match Module Bit n“ zeigt die Übereinstimmung („**Match**“) an, wenn bei einem Gleichstand von PCA-Counter und dem Capature / Compare-Register das zugehörige Flag CCFn im Register CCON gesetzt wird. Dadurch wird der Interrupt signalisiert bzw. ausgelöst.
Bit 2 = TOGn	Es handelt sich hier um das Toggle Module Bit. Der **Toggle Mode** wird programmiert, indem das ECOMn-, das MATn- und das TOGn-Bit gesetzt werden. Eine Übereinstimmung der Inhalte von PCA-Counter und Compare / Capture-Register toggelt den zugehörigen CEXn-Pin am Controllergehäuse.
Bit 1 = PWMn	Dieses „Pulse Width Modulation Module n Mode Bit“ lässt, wenn gesetzt, den Modul n als **Puls-Weiten-Modulator** arbeiten. Das PWM-Signal wird am zugehörigen CEXn-Pin des Controllergehäuses ausgegeben
Bit 0 = ECCFn	Dieses „Enable CCFn – Interrupt Bit“ muss **gesetzt** werden, wenn das entsprechende Bit CCFn im CCON-Register einen **Interrupt** anfordern soll

14.4. Praxis-Anwendung: Erzeugung von PWM-Signalen

14.4.1. Arbeitsweise und Programmierung der PWM-Funktion

Dazu sehen wir uns folgenden Übersichtsschaltplan etwas genauer an:

Figure 58. PCA PWM Mode



In das CCAPnL-Register wird immer automatisch der Wert des CCAOnH-Registers geladen.
Von links wird nun der augenblickliche Zählerstand des LOW-Registers im PCA-Counter angeliefert und dauernd mit dem Inhalt des CCAPnL-Registers verglichen (…bei Modul 0 würde dieses Register „CCAP0L“ heißen!). Solange der CL-Register-Inhalt kleiner ist als der CCPnL-Wert, bleibt das CEXn-Ausgang (…hier wäre es „CEX0“) auf LOW. Sobald jedoch der CL-Wert größer ist, wird CEXn auf HIGH gesetzt. Das CL-Register zählt aber weiter bis zum Überlauf und dieser Überlauf kopiert erneut den Reloadwert von CCAPnH nach CCAPnL. Jetzt ist CL wieder kleiner als CCAPnL und damit erhält man erneut LOW-Pegel an CEXn usw. usw.

Der Anwender muß folgende Initialsierungen vornehmen:

a) Clockfrequenz über die Bits CPS0 und CPS1 im CMOD-Register auswählen

b) ECOM0-Bit und PWM0-Bit im zugehörigen CCAPMn-Register werden gesetzt (= Compare-Mode und PWM-Mode werden freigegeben)

c) Reloadwert in das zum Modul gehörende CCAPnH-Register schreiben

d) Nun kann mit dem CR (= Counter Run)-Bit im Register „CCON“ der PCA-Counter gestartet werden und die PWM-Erzeugung läuft.

Eine erste Anwendung gibt es auf dem nächsten Blatt.

14.4.2. PWM-Steuerung über Drucktasten ohne Interrupt

Als Einstieg wollen wir uns folgende Aufgabe stellen:

Der PCA-Counter wird zur Erzeugung eines PWM-Signals mit Modul 0 verwendet.

Das PWM-Signal wird an Pin CEX0 (= Portpin P1.3) ausgegeben.

Als Taktfrequenz wird der „PCA Clock / 6“ benützt.

Durch Drucktasten an Port P 0 kann man zwischen 5 verschiedenen Pulsweiten wählen.

Wird keine Taste gedrückt, dann entsteht auch kein Ausgangssignal.

Lösung:
Wir packen die Initialsierung der PCA in eine eigene Funktion, die beim Start des Hauptprogramms aufgerufen wird. In einer Endlosschleife werden pausenlos die 5 Drucktasten abgefragt. Ist keine gedrückt, dann steht alles still. Ist eine Taste gedrückt, dann wird der PCA-Timer gestartet und der passende Reloadwert in das „Capture-HIGH-Register“ (= CCAP0H) geschrieben. Dieser Wert wird dann bei jedem Match-Ereignis von dort aus automatisch neu in das „Capture-LOW-Register“ (=CCAO0L) kopiert.

```
/*-------------------------------------------------------------
Programmbeschreibung
-------------------------------------------------------------
Name:            PWM_01.c
Funktion:        Das PCA wird zur Erzeugung eines PWM-Signals mit Modul 0
                 verwendet.
                 Das PWM-Signal wird an Pin CEX0 (= Portpin P1.3)
                 ausgegeben.
                 Als Taktfrequenz wird PCA Clock Frequency / 6 verwendet.
                 Durch Drucktasten an Port P 0 kann man zwischen
                 verschiedenen Pulsweiten wählen.
                 Wird keine Taste gedrückt, dann entsteht auch kein
                 Ausgangssignal.

Datum:           06. 01. 2008
Autor:           G. Kraus

-------------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------------*/

#include <t89c51ac2.h>
#include <stdio.h>

sbit ausgang=P1^3;              // CEX0-Pin als Ausgang

sbit pulsweite_1=P0^0;          // Drucktasten für Pulsweiten
sbit pulsweite_2=P0^1;
sbit pulsweite_3=P0^2;
sbit pulsweite_4=P0^3;
sbit pulsweite_5=P0^4;

/*-------------------------------------------------------------------
Prototypen
-------------------------------------------------------------------*/
void PWM_INI(void);             // Initialisierung der PWM
```

```
/*---------------------------------------------------------------------
Hauptprogramm
---------------------------------------------------------------------*/
void main(void)
{       AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen
        PWM_INI();                      // PWM-Initialisierung

        while(1)                        // Endlosschleife
        {
                while(pulsweite_1==0)           // Erste Taste gedrückt
                {    CR=1;
                     CCAP0H=50;
                     while(pulsweite_1==0);
                }

                while(pulsweite_2==0)           // Zweite Taste gedrückt
                {    CR=1;
                     CCAP0H=100;
                     while(pulsweite_2==0);
                }

                while(pulsweite_3==0)           // Dritte Taste gedrückt
                {    CR=1;
                     CCAP0H=150;
                     while(pulsweite_3==0);

                }

                while(pulsweite_4==0)           // Vierte Taste gedrückt
                {    CR=1;
                     CCAP0H=200;
                     while(pulsweite_4==0);
                }

                while(pulsweite_5==0)           // Fünfte Taste gedrückt
                {    CR=1;
                     CCAP0H=240;
                     while(pulsweite_5==0);
                }
                CR=0;

}
}

/*---------------------------------------------------------------------
Zusatzfunktionen
---------------------------------------------------------------------*/
void PWM_INI(void)
{       CMOD=0x00;     // Interner Clock fpca/6  --  kein Interruptbetrieb
        CCON=0x00;     // PCA steht still
        CCAPM0=0x42;   // ECOM0-Bit und PWM0-Bit werden gesetzt
}
```

14.4.3. Analogsteuerung der PWM für einen LED-Dimmer oder DC-Motor

```
/*-------------------------------------------------------------
Programmbeschreibung
-------------------------------------------------------------
Name:         PWM_analog_01.c

Funktion:
Das PCA wird zur Erzeugung eines PWM-Signals durch den PCA-Modul 0
verwendet. Das PWM-Signal wird an Pin CEX0 (= Portpin P1.3) ausgegeben.
Als Taktfrequenz wird „PCA Clock Frequency / 6" verwendet.
Die Gleichspannung an Poti 0 auf der DA-Platine wird dem Eingang AN0 =
Portpin P1^0 zugeführt und dann mit dem integrierten AD-Wandler gemessen.
Das Ergebnis dient als Reloadwert für die PWM, wobei zuvornoch die
"Drehrichtung des Poti's" umgekehrt wird.
Das Ausgangssignal an Pin CEX0 kann durch eine LED sichtbar gemacht werden.
So entsteht ein LED-Dimmer.Oder es kann (nach Zwischenschaltung eines
Treibers)zur Speisung eines Gleichstrom-Motors dienen.

Datum:        06. 01. 2008
Autor:        G. Kraus

-------------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------------*/

#include <t89c51ac2.h>
#include <stdio.h>

sbit ausgang1=P1^3;       // CEX0-Pin als Ausgang

/*-------------------------------------------------------------------
Prototypen
-------------------------------------------------------------------*/
void PWM_INI(void);            // Initialisierung der PWM
void wait_ADC_ini(void);  /* Nötige Wartezeit  nach AD-Wandler-
                             Initialisierung */

/*-------------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------------*/
void main(void)
{  unsigned char z;            // Erforderlicher Zwischenspeicher
   unsigned char channel;      // Gewählter AD-Messkanal

   AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen
   PWM_INI();
   channel=0x00;         // Kanal AN0 gewählt
   ADCF=0x01;            // Pin P1.0 als A-D-Channel betreiben

   ADCON=0x20;           // AD Enable freigeben
   wait_ADC_ini();       // warten, bis ADC initialisiert
   P1=0xFF;
   CR=1;                 // PWM starten


   while(1)
   {    ADCON=ADCON&0xF8;     // Alte Kanaleinstellung (SCH0...SCH2) löschen
        ADCON=ADCON|channel;  // Auf Channel AN3 schalten
```

```
        ADCON=ADCON|0x08;     // ADSST setzen, "Single Conversion" starten
        while((ADCON&0x10)!=0x10);   // Warten, bis ADEOC setzt
        ADCON=ADCON&0xEF;            // ADEOC wieder löschen
        z=ADDH;                      // Messergebnis abholen
        CCAP0H=(255-z);              /* Messergebnis in CCAP0H kopieren
                         und dabei die Drehrichtung des Potis umkehren */
   }

}

/*---------------------------------------------------------------------------
Zusatzfunktionen
---------------------------------------------------------------------------*/
void PWM_INI(void)
{    CMOD=0x00;              // Interner Clock FPca/6, kein Interruptbetrieb
     CCON=0x00;              // PCA steht still
     CCAPM0=0x42;            // ECOM0-Bit und PWM0-Bit werden gesetzt
}


void wait_ADC_ini(void)      /* Erzeugt die nötige Wartezeit nach der
                             Initialisierung des AD-Converters */
{    unsigned char x;
     for(x=0;x<=5;x++);
}
```

Hinweis:
Wählt man den „Internen Clock FPca/6" mit der Anwweisung

CMOD = 0x00;

dann entspricht das einer Zähltaktfrequenz von 1 MHz.

Schaltet man dagegen mit

CMOD = 0x02;

auf „Internen Clock FPca/2" um, dann steigt die Taktfrequenz auf 3 MHz.

14.5. PCA-Interrupt-Programmierung: Verwendung des High – Speed – Output - Modes

In dieser Betriebsart wird der CEXn-Pin bei jedem Gleichstand von PCA-Timer und dem ausgewählten PCA-Modul CCAPn „getoggelt“ (= invertiert). So lassen sich symmetrische Rechtecksignale bis ca. 50 kHz erzeugen.

Aufgabe:
Erzeugen Sie am Pin CEX0 (= Portpin P1^3) ein symmetrisches Rechtecksignal mit der Frequenz f = 10 kHz.

Lösung:
Es wird der High-Speed-Output-Mode für Modul CCAP0 programmiert. Sobald der PCA-Timer hochzählt und den im Modul gewählten Einstellwert erreicht, wird ein Interrupt ausgelöst.
In der Interrupt-Service-Routine löscht man das auslösende Interrupt-Flag und setzt zusätzlich den PCA-Timer auf den Startwert „Null“ zurück.

```
/*-------------------------------------------------------------------
Programmbeschreibung
-------------------------------------------------------------------
Name:          High_Speed_Output_01.c

Funktion:      Es soll ein 10 kHz-Signal an Pin CEX0 (= P1^3) erzeugt
               werden. Dazu wird mit dem "High Speed Output Mode" und
               Interrupt gearbeitet.

Datum:         03. 12. 2007
Autor:         G. Kraus

-------------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------------*/

#include <t89c51ac2.h>
#include <stdio.h>

/*-------------------------------------------------------------------
Prototypen
-------------------------------------------------------------------*/

void PCA_Highspeed_INI(void);

void PCA_Highspeed_ISR(void);
/*-------------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------------*/

void main(void)
{    AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen
     PCA_Highspeed_INI();      // High Speed Output Mode initialisieren
     EC=1;                     // PCA-Interrupt-Hauptschalter ON
     EA=1;                     // Interrupt-Hauptschalter ON
     CR=1;                     // PCA-Timer starten

     while(1);                 // Endlosschleife
}
/*-------------------------------------------------------------------
Zusatz-Funktioen
-------------------------------------------------------------------*/
void PCA_Highspeed_INI(void)
{
```

```
    CCAP0H=0x00;          // High-Byte von CCAP0 mit "0" laden
    CCAP0L=3(;            // LOW-Byte von CCAP0 mit "255" laden
    CL=0;                 // PCA-Counter LOW-Byte mit "0" starten lassen
    CH=0;                 // PCA-Counter HIGH-Byte mit "0" starten lassen

    CMOD=0x01;            // Interner Clock fpsa/6, Interruptbetrieb
    CCON=0x00;            // PCA steht still, alle Flags gelöscht
    CCAPM0=0x4D;          /* ECOM0-, TOG0-, MAT0- und ECCF0-Bit werden
                             gesetzt */

      /* VORSICHT: Wird das CCAPnL-Register NACH dem CCAPnH-
      Register oder dem CCAPMn-Register geladen, dann wird das „Compare-
      Freigabe-Bit ECOMn“ wieder gelöscht (Siehe Datenblatt, Seite 98).

      DADURCH LÄUFT DER TIMER UND DAMIT AUCH DAS
      PROGRAMM NICHT!!

      Folglich darf man das CCAPMn-Register
      immer erst am Schluß initialisieren!
      */
}

void PCA_Highspeed_ISR(void) interrupt 6     // Interrupt Vector 0x33
{    CCF0=0;              // CCAP0-Interrupt-Flag löschen
     CL=0;                // LOW-Byte des PCA-Counters zurücksetzen
}
```

14.6. PCA-Unterrichtsprojekt: Reaktionszeit-Tester mit Display-Anzeige

Aufgabe:

Es soll ein **Reaktionszeit-Tester** gebaut und programmiert werden. Dazu ist Port **P 0 an die Drucktastenplatine** und **Port 4 an die LED-Kette** angeschlossen. **Port P 2 versorgt das LCD-Display**.

Im **Ruhezustand leuchtet eine LED** bei der LED-Kette.

Drückt man nun **eine Startta**ste (= an Portpin P 0^0), dann wird eine **Zufallszahl „A“** erzeugt, die zwischen 25 und 255 liegt. Jetzt erlischt die LED und es läuft eine Verzögerungszeit von

(A) x (10Millisekunden) = 0,25…2,55 Sekunden

ab, in der die **LED dunkel** bleibt.

Sobald nach dieser Zeit die LED wieder einschaltet, muss der **Anwender die Taste an Portpin P0^2 drücken**.

Diejenige Zeit, die zwischen dem erneuten Aufleuchten der LED und dem Tastendruck des Anwenders vergeht, wird gemessen und mit passendem Begleittext als Reaktionszeit auf dem Display angezeigt.

Sobald die Stopptaste losgelassen wird, ist die Anlage zur nächsten Messung bereit, aber die Display-Anzeige ändert sich dadurch (noch) nicht.

Wird jedoch die Stopptaste gedrückt, bevor die LED aufleuchtet, so soll eine Mogelwarnung ausgegeben werden.

Lösung:

a) Timer 0 wird als 8 Bit-Autoreload-Timer betrieben und läuft dauernd, wobei der Reloadwert „25“ beträgt.

b) Sobald die Starttaste gedrückt wird, übernimmt eine Variable den Timer 0 - Zählerstand als Zufalls-Zeitvorgabe „A“ beim Timer 2. In diesem Augenblick wird die LED ausgeschaltet.

c) Der Timer 2 erzeugt nun eine Verzögerungszeit von (A) x (10 Millisekunden)

d) Ist diese Verzögerungszeit abgelaufen, dann wird die LED wieder eingeschaltet. Ab jetzt misst der PCA-Timer die Zeit bis zum Drücken der Stopptaste.

e) Der Druck auf die Stopptaste stoppt den PCA-Timer. Die Reaktionszeit wird nun berechnet und auf dem Display angezeigt. Eine Überschreitung des Grenzwertes von 0,7 Sekunden wird entsprechend kommentiert

f) Beim Loslassen der Stopptaste wird wieder die Starttaste abgefragt usw.

```
/*------------------------------------------------------------------
Programmbeschreibung
------------------------------------------------------------------
Name:          reaktionstester_03.c
Funktion:
Timer 0 wird als 8 Bit-Autoreload-Timer betrieben und läuft dauernd, wobei
der Reloadwert "25" beträgt.

Sobald die Starttaste (an P0.0) gedrückt wird, übernimmt eine Variable
"Zufall" den Timer 0 - Zählerstand als Zeitvorgabe "A" beim PCA-Timer.
In diesem Augenblick wird die LED (an P1.0) ausgeschaltet.

Der Timer 2 erzeugt nun eine Verzögerungszeit von (A) x (10 Millisekunden).
Das ergibt Zufallszeiten zwischen 0,25 und 2,55 Sekunden.

Ist diese Verzögerungszeit abgelaufen, dann wird die LED wieder
eingeschaltet. Ab jetzt misst der PCA-Timer die Zeit bis zum Drücken der
Stopptaste (an P 0.2).
```

```
Der Druck auf die Stopptaste stoppt den PCA-Timer. Die Reaktionszeit wird
nun berechnet und auf dem Display angezeigt. Eine Überschreitung des
Grenzwertes von 0,7 Sekunden wird entsprechend kommentiert.

Wird auf die Stopptaste gedrückt, bevor die LED aufleuchtet, wird eine
Mogelwarnung angezeigt.

Datum:      14. 12. 2007
Autor:      G. Kraus
-----------------------------------------------------------------
Deklarationen und Konstanten
-----------------------------------------------------------------*/
#include <t89c51ac2.h>
#include <stdio.h>

sbit LED=P1^0;              // P1^0 als LED-Ausgang
sbit start=P0^0;            // Starttaste an P0^0
sbit stopp=P0^2;            // Stopptaste an P0^2

unsigned char Zufall;       // Zufallszahl
unsigned int Reaktion;      // Reaktionszeit in 1ms-Einheiten
unsigned int z;
double r;
bit lumpbit;                // Betrüger-Erkennung

/*-----------------------------------------------------------------
Prototypen
-----------------------------------------------------------------*/

void Timer0_INI(void);      // Initialisierung von Timer 0
void Timer2_INI(void);      // Timer2 initialisieren
void PCA_Software_Timer_INI(void);  // PCA-Initialisierung als 16 Bit-Timer

void ISR_Timer2(void);      //Interrupt-Service-Routine für Timer 2
void ISR_PCA(void);         // Interrupt Service Routine für PCA

void show_reaction(void);   // Reaktionszeit anzeigen

void zeit_s(void);          // Protoypen der Display-Steuerung angeben!!
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);

code unsigned char     Starttext[21]={"Reaktionstester     "};
code unsigned char          leer[21]={"                    "};
code unsigned char      Overflow[21]={"Lahme Ente!         "};
code unsigned char          lump[21]={"Mogeln gilt nicht!! "};


/*-----------------------------------------------------------------
Hauptprogramm
-----------------------------------------------------------------*/
void main(void)
{    AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen
     lcd_ein();
     lcd_ini1();
```

```
     switch_z1();
     show_text(Starttext);
     lumpbit=0;
     Timer0_INI();         // Timer 0 programmieren (Autoreload mit 25)
     Timer2_INI();         // Timer 2 initialisieren
     PCA_Software_Timer_INI(); /* PCA als 16 Bit-Software-timer
                               initialisieren*/

     P1=0x01;              // Start-LED an P1^0 einschalten
     EC=1;                 // PCA-Interrupt-Freigabe
     EA=1;                 // Alle Interrupts freigeben

     while(1)              //Endlosschleife
     {    Zufall=0;        // Zufallszahl auf Null setzen
          Reaktion=0;
          TL0=25;
          TR0=1;           // Timer 0 starten

          while(start==1); // Starttaste gedrückt?
          Zufall=TL0;      // Zufallszahl erzeugen
          TR0=0;           // Timer 0 stoppen

          switch_z1();     // Leertext zeigen
          show_text(leer);

          LED=0;           // LED ausschalten
          TR2=1;           // Timer 2 für Zufallszeit starten
          while (Zufall!=0)
          {    if((Zufall!=0)&&(stopp==0))
               {    lumpbit=1;
               }
          }                // Warten, bis Zufallszeit vorbei

          if(lumpbit==1)
          {    switch_z1();
               show_text(lump);
               lumpbit=0;
               LED=1;
          }
          else
          {    TR2=0;         // Timer 2 wieder stoppen
               LED=1;         // LED wieder einschalten

               Reaktion=0;    // Reaktionszeit zurücksetzen
               CL=0;       // PCA-Counter LOW-Byte mit "0" starten lassen
               CH=0;       // PCA-Counter HIGH-Byte mit "0" starten lassen
               CR=1;       // PCA-Sofware-Timer starten
               while(Reaktion<700&&stopp==1); // auf Stopptaste warten
               CR=0;       // PCA-Software-Timer wieder stoppen

               show_reaction(); // Reaktionszeit anzeigen
          }
     }
}
/*---------------------------------------------------------------------
Zusatzfunktionen
---------------------------------------------------------------------*/
void Timer0_INI(void)       // Initialisierung von Timer 0
{    TMOD=0x02;             // Timer 0 im Reload-Betrieb (Mode 2)
     TL0=25;                // Start-Wert ist 25
     TH0=25;                // Reload-Wert ist 25
     ET0=1;                 // Timer 0 - Interrupt freigeben
     TF0=0;                 // Timer 0-Überlaufflag löschen
```

```
     TR0=0;                // Timer 0 ausschalten
}


void Timer2_INI(void)
{    C_T2=0;               // Timer-Betrieb
     RCLK=0;               // Timer 2 nicht für Serial Port verwenden
     TCLK=0;               //     "    "    "    "  "
     CP_RL2=0;             // Autoreload bei Überlauf
     EA=1;                 // Interrupt-Hauptschalter EIN
     ET2=1;                // Timer 2 - Interrupt EIN
     RCAP2L=0xEF;          // Reload-Wert ist 55535 (gibt 10 Millisekunden )
     RCAP2H=0xD8;          // (= 0xD8EF)
     TF2=0;                // Überlaufflag löschen
     TR2=0;                // Timer 2 stoppen
}

void PCA_Software_Timer_INI(void)
{
/* VORSICHT: Wird das CCAPnL-Register   NACH   dem CCAPnH-Register und
dem CCAPMn-Register geladen, dann wird das "Compare-Freigabe-Bit ECOMn"
wieder gelöscht (Siehe Datenblatt, Seite 98).
DADURCH LÄUFT DER TIMER UND DAMIT AUCH DAS PROGRAMM NICHT!!
Folglich sollte man das CCAPMn-Register immer erst am Schluß
initialisieren! */

     CCAP0L=0xE8;        // Zählerstand "1000" = 0x03E8
     CCAP0H=0x03;        //
     CL=0;               // PCA-Counter LOW-Byte mit "0" starten lassen
     CH=0;               // PCA-Counter HIGH-Byte mit "0" starten lassen

     CMOD=0x01;          // Interner Clock fpsa/6, Interruptbetrieb
     CCON=0x00;          // PCA steht still, alle Flags gelöscht
     CCAPM0=0x49;        // ECOM0-, MAT0- und ECCF0-Bit werden gesetzt
}


void ISR_Timer2(void)   interrupt 5    // Timer 2 hat Interrupt-Index 5
{    TF2=0;                            // Überlaufflag löschen
     Zufall--;                  // Zufallszeit in 10ms-Einheiten erzeugen
}


void ISR_PCA(void)    interrupt 6    // (PCA hat Interrupt Vector 0x33)
{    // CF=0;              // PCA-Timer-Interrupt Flag löschen

     CL=0;                 // LOW-Byte zurücksetzen
     CH=0;                 // High-Byte zurücksetzen
     Reaktion++;           // 1ms-Einheiten der Reaktionszeit zählen
     CCF0=0;               // CCAP0-Interrupt-Flag löschen
}

void show_reaction(void)     // Reaktionszeit anzeigen
{    unsigned char Anzeige[21]={"R-Zeit = 000 ms     "};
     if(Reaktion>=700)       // Reaktion länger als 0,7 Sekunden?
     {    switch_z1();       // Overflow-Text anzeigen
          show_text(Overflow);
          Reaktion=0;        // Variable zurücksetzen
     }
     else                    // Reaktion schneller als 0,7 Sekunden
     {    r=0;
          r=Reaktion;    // Reaktionszeit umkopieren in Variable
          z=r;           // Variable "z" dient zur Display-Anzeige
```

```
        /*Bei einem Quarztakt von 11,0592MHz müßte das Ergebnis um den
        Faktor 12 MHz / 11,0592MHz = 1,085 erhöht werden, um die korrekt
        verbrauchte Zeit in Millisekunden zu erhalten. Dann ist folgende
        Zeile im C-Programm einzufügen:  */

        z=(r*1085)/1000;

        Anzeige[11]=z %10 +48;       // Letzte Stelle berechen
        Anzeige[10]=z/10 %10 +48;    // Mittlere Stelle berechnen
        Anzeige[9]=z/100 %10 +48;    // Erste Stelle berechnen
        switch_z1();                 // Auf Zeile 1 umschalten
        show_text(Anzeige);          // Ergebnis anzeigen
        Reaktion=0;                  // Variable zurücksetzen
        z=0;                         //    "     "
      }
}
```

15. Unterrichtsprojekt: PWM-Fahrtregler für Märklin Miniclub-Modellbahn

15.1. Grundsätzliches

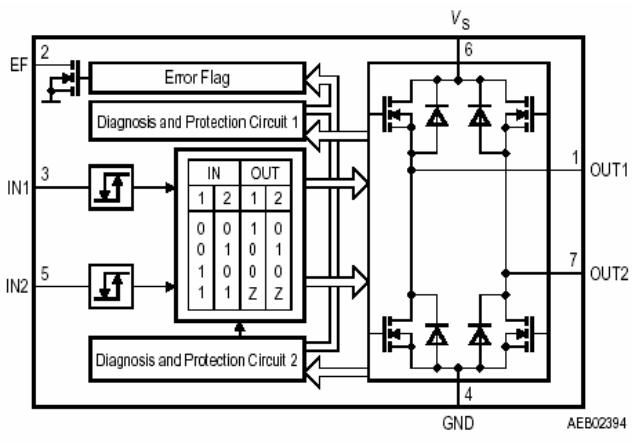
Zum Betrieb von Modelleisenbahnen gibt es zwei Speisesysteme:

Beim **Wechselstromsystem** (= nur Märklin H0) wird **ein Pol** der Speisespannung an **beide** Schienen gelegt -- folglich braucht man die beiden Räder einer Achse nicht von ihr zu isolieren. Jedoch ist in der Mitte zwischen beiden Schienen noch ein **weiterer Leiter („Mittelleiter**“) erforderlich. Er besteht heute nur noch aus einem isoliert eingesetzten Punktkontakt in jeder Schwelle. Der Vorteil ist die außergewöhnliche Betriebssicherheit auch bei korrodierten oder stark verschmutzten Schienen. Nachteile sind der erforderliche Schleifer zur Abnahme der Spannung von den Mittelleiter-Punktkontakten und das Problem der Fahrtrichtungsumkehr. Dazu muss z. B. in der Lok mittels eines Relais die Stromrichtung in der Erregerwicklung des Hauptschluss-Antriebsmotors gegenüber dem Ankerstrom umgepolt werden. Märklin arbeitete hier jahrzehntelang mit zwei getrennten (und gegensinnig gewickelten) Erregerwicklungen bei seinen Motoren, zwischen denen umgeschaltet wurde.

Beim **Gleichstromsystem** bilden die beiden (von den Schwellen isolierten) Schienen den **Plus- und Minuspol** der Speisespannung. Die Räder müssen nun von der Achse isoliert werden, die Spannung selbst wird über „Radschleifer“ abgenommen und ins Innere der Lok geführt. Der Punktkontakt-Mittelleiter fällt ebenso weg wie der zugehörige Schleifer und damit wirkt die ganze Sache viel vorbildgerechter. Die Fahrtrichtungsumkehr erfolgt durch Umpolen der Betriebsspannung an den Schienen.
Dieses System wird nun vom kleinsten System (Märklin Miniclub, Spur Z mit 6,5 mm Spurweite) über Spur N und H0 bis zur Spur 1 (Spurweite 45 mm) von allen anderen Herstellern verwendet. Es zeigt sich aber, dass dieses Prinzip wesentlich empfindlicher gegen Schienenverschmutzung und –Oxidation ist. Speziell bei kleinen angelegten Spannungen tut sich oft gar nichts und beim Aufdrehen des Fahrtreglers geht es plötzlich mit einem Ruck los.

Deshalb bietet sich hier die Versorgung mit **Pulsen** an, deren Breite zur Veränderung der Fahrgeschwindigkeit variiert wird (= PWM = Puls Weiten Modulation). Die Wiederholfrequenz ist konstant und reicht in der Praxis von einigen Kilohertz bis hinauf in den unhörbaren Bereich oberhalb von 15 kHz.
Auf diese Weise liegt am Antriebsmotor immer entweder Null oder gleich die Maximalspannung und das ganze System benimmt sich deshalb bei niedrigen Geschwindigkeiten wesentlich folgsamer. Auch die Empfindlichkeit gegen Verschmutzung und Oxidation geht zurück. Zusätzlich lässt sich dieses Prinzip leicht zu den jetzt üblichen Digitalsteuerungen der Eisenbahnen ausbauen.

15.2. DC-Motorsteuerungen über eine H-Brücke



Speziell für den PWM-Betrieb werden integrierte Schaltungen (Hier: TLE5205-2 der Firma Infineon) bereitgestellt. Bei ihnen dient ein Logikteil **(= Eingänge IN1 und IN2) zur Auswahl der Fahrtrichtung sowie der Umschaltung auf „Bremsen“ oder „Freilauf“ mit einer** Leistungsendstufe **aus vier integrierten Power-MOSFETs samt Freilaufdioden. Zusätzlich sind Sicherheitsschaltungen eingebaut, mit denen das Umpolen der Fahrtrichtung ohne Probleme vor sich geht (...beim Umpolen muss jeweils bei zwei in Reihe liegenden FETs der eine aus- und der andere eingeschaltet werden. Wird dieser Vorgang nicht sauber getrennt, dann ist für einen kurzen Moment die Versorgungsspannung kurzgeschlossen, wenn beide zugleich leiten....).**

Sehen wir uns zuerst mal die Ansteuerung des TLE5205 an:

Functional Truth Table

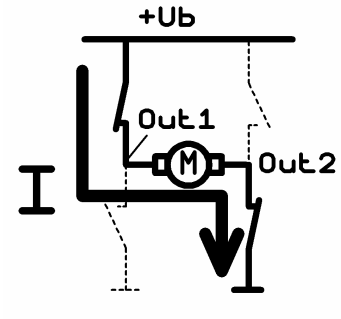
IN1	IN2	OUT1	OUT2	Comments
L	L	H	L	Motor turns clockwise
L	H	L	H	Motor turns counterclockwise
H	L	L	L	Brake; both low side transistors turned-ON
H	H	Z	Z	Open circuit detection

Notes for Output Stage

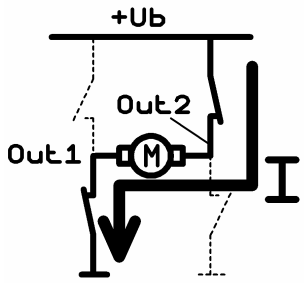
Symbol	Value
L	Low side transistor is turned-ON High side transistor is turned-OFF
H	High side transistor is turned-ON Low side transistor is turned-OFF
Z	High side transistor is turned-OFF Low side transistor is turned-OFF

Ersetzen wir jetzt die POWER-MOSFETs im IC durch Schalter, dann erhalten wir für die obige Wahrheitstabelle folgende Stromkreise:

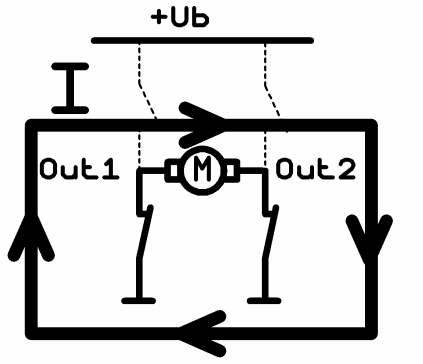
a) **Rechtslauf** (IN1 = LOW, IN2 = LOW)



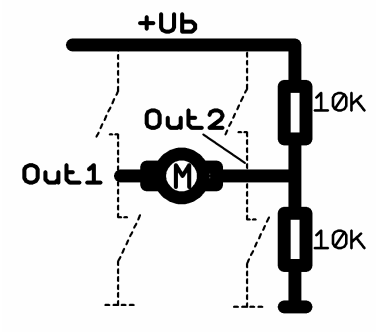
b) **Linkslauf** (IN1 = LOW, IN2 = HIGH)



c) **Bremsen** (IN1 = HIGH, IN2 = LOW)
Der Motor ist nun kurzgeschlossen, da beide unteren MOSFETs leiten



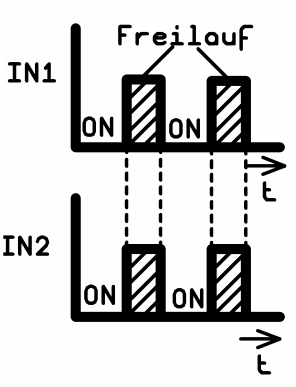
d) **Freilauf** (IN1 = HIGH, IN2 = HIGH)
An OUT2 misst man die **halbe** **Betriebsspannung gegen Masse**, da die beiden integrierten 10kΩ-Widerstände einen Spannungsteiler bilden. Falls sich der Motor noch dreht, erhält man an OUT1 (je nach Drehrichtung) eine andere Gesamtspannung, da sich dann die induzierte Motorklemmenspannung noch zu dieser Spannung an OUT2 addiert.



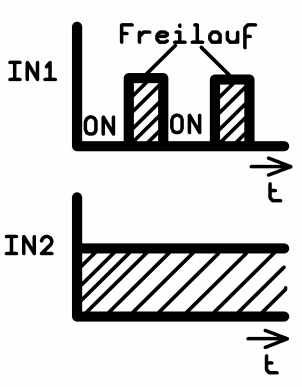
Wie man sieht, dient der Eingang IN2 zur Auswahl der Drehrichtung, während an Eingang IN1 umgeschaltet wird zwischen „Motor dreht sich“ (= „LOW“ an Pin IN1) und „Motor steht still“ (= HIGH an Pin IN1).

Bei unserer Platine wollen wir beim Abschalten jedes Pulses die durch den Stromanstieg im Magnetfeld des Ankers gespeicherte Energie zurückgewinnen. Deshalb wählen wir während der Abschaltzeit den Betriebszustand „Freilauf“ und die Ankerinduktivität schiebt nun diese Energie über die integrierten Freilaufdioden zurück in den dicken Speicherkondensator der Stromversorgung (gewählt: 2200 Mikrofarad).
Der Mikrocontroller gibt über den Portpin **P1.0 das Signal „DIR“** (= Direction = Drehrichtung) und über **Portpin P1.3 das PWM-Signal (= CEX0)** aus. Durch passende Logikbausteine werden daraus die Spannungen für die beiden IC-Eingänge IN1 und IN2 erzeugt. Dazu gehören folgende Liniendiagramme:

IN1 und IN2 bei Rechtslauf



IN1 und IN2 bei Linkslauf



==========

Genaue physikalische Vorgänge im PWM-Betrieb
Beginnen wir beim stillstehenden Motor, denn ihn können wir in diesem Augenblick als eine simple Reihenschaltung einer Induktivität mit einem Ohmschen Widerstand betrachten (….dieser Widerstand stellt zunächst die Summe aus Drahtwiderstand und dem Übergangswiderstand der Schleifkohlen zum Anker-Kollektor dar. Erst später, also nach dem Loslaufen, ändert sich das, denn dann kommt ein weiterer Widerstand dazu, der die zum Antrieb aufgewendete bzw. an der Ankerwelle abgegebene mechanische Leistung repräsentiert). Das erinnert doch sehr an die berühmte „Ladekurve“ bei Tiefpässen und dazu (Siehe nächstes Blatt!) gehören folgende Erkenntnisse:

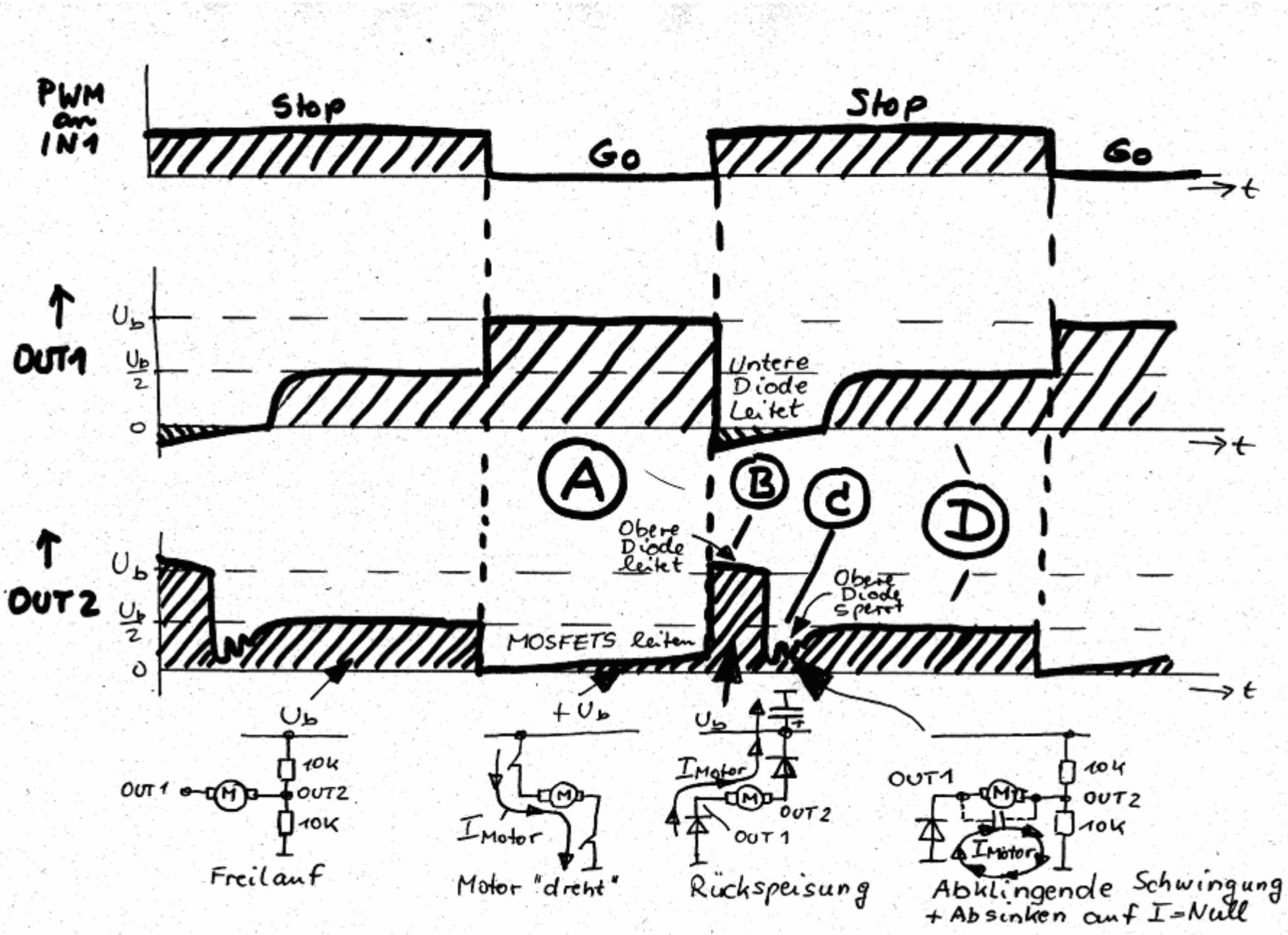
a) **Phase A**: vom Treiber kommt der Puls und wird an den Motor angelegt. Deshalb steigt der Motorstrom nach der bekannten e-Funktion an und der Motor entwickelt Drehmoment. Sehr gut ist dieser Stromanstieg bei OUT2 an der langsam ansteigenden „Restspannung“ des MOSFETS zu erkennen.

b) **Phase B**: der Puls wird aus- und dadurch der Motor auf „Freilauf“ umgeschaltet. Die gespeicherte Magnetfeldenergie treibt aber den fließenden Strom weiter über die beiden integrierten Freilaufdioden -- so wird diese Energie in den Speicherkondensator zurückgeladen. Deshalb polt sich die Motorspannung um und wird an OUT2 so hoch (= Betriebsspannung + 0,4V), dass über die obere Freilaufdiode der Strom in den Speicherkondensator fließen kann. Da gleichzeitig auch die untere Freilaufdiode leiten, misst man nun an OUT1 plötzlich eine Spannung von ca. -0,4V (…Schottky-Dioden….).

c) **Phase C**: Sobald die Rückspeisung beendet und der Strom auf Null abgeklungen ist, sperren die Dioden wieder. Da auch alle MOSFETS ausgeschaltet sind, tobt sich die verbleibende Restenergie (…da die beiden Dioden ja keine Schwellspannung von Null Volt besitzen, wird zu früh abgeschaltet…) in Form einer gedämpften Schwingung aus (…der Motor besitzt eben außer der Induktivität auch eine Wicklungskapazität, und das gibt einen Schwingkreis….).

d) **Phase D**: Ist diese Energie vollends verheizt, dann ist der Motor stromlos. Am rechten Ende des Motors (OUT2) messen wir nun die halbe Versorgungsspannung gegen Masse, denn dafür sorgen die beiden als Spannungsteiler wirkenden 10kΩ-Widerstände. Am linken Ende (OUT1) addiert sich zu dieser halben Betriebsspannung -- falls der Motor bereits rotiert! -- die in der Motorinduktivität induzierte Leerlaufspannung.

Der Motor kann sich jedoch nur dauernd drehen, wenn der **nächstfolgende Puls deutlich früher kommt, bevor die Rückspeisung und der Ausklingvorgang abgeschlossen sind** -- **kurz gesagt, bevor der Motorstrom wieder auf Null abgesunken ist.** Nur dann wird genügend Drehmoment entwickelt und mechanische Leistung abgegeben. Längere „Einschaltzeit“ und kürzere „Freilaufzeit“ erhöhen damit die Motordrehzahl und so die Geschwindigkeit der Lokomotive. Auf dem nächsten Blatt sind diese Vorgänge für „Rechtslauf“ dargestellt.


PWM an IN1
Stop
Go
Stop
Go
t
OUT1
U_b
$\frac{U_b}{2}$
0
Untere Diode Leitet
A
B
C
D
OUT2
Obere Diode leitet
Obere Diode sperrt
MOSFETS leiten
$+U_b$
10k
OUT1
OUT2
M
I_{Motor}
Freilauf
Motor "dreht"
Rückspeisung
Abklingende Schwingung
+ Absinken auf I=Null

15.3. Details der H-Brücken-Zusatzplatine

Der Einsatz dieses Brücken-ICs sollte nicht auf elektrische Eisenbahnen beschränkt bleiben, denn die Grenzwerte der Endstufe (5A / 40V) reizen dazu, auch andere Motoren zu steuern. Dabei ist es sehr gefährlich, den Mikrocontroller an dieselbe Speisespannung wie den Motor anzuschließen. Schließlich reagieren Induktivitäten sehr bösartig mit hohen Spannungsspitzen auf schnelle Stromänderungen -- jede Zündspule beweist das. Also wurden die Stromversorgungen von Mikrocontroller und DC-Motorkreis sorgfältig getrennt, ebenso wurden zwischen die steuernden Controllerausgänge und die Eingänge IN1 und IN2 der Motorsteuerung Oprokoppler zur galvanischen Trennung eingefügt.

Der TLE5205-2 besitzt eine integrierte Überwachungsschaltung für den Motorstromkreis und setzt bei bestimmten Vorkommnissen (Leerlauf oder Kurzschluss usw…..Siehe Datenblatt) ein **Errorflag**. Auch diese Information wird über einen weiteren Optokoppler in Richtung Mikrocontroller zurückgemeldet.

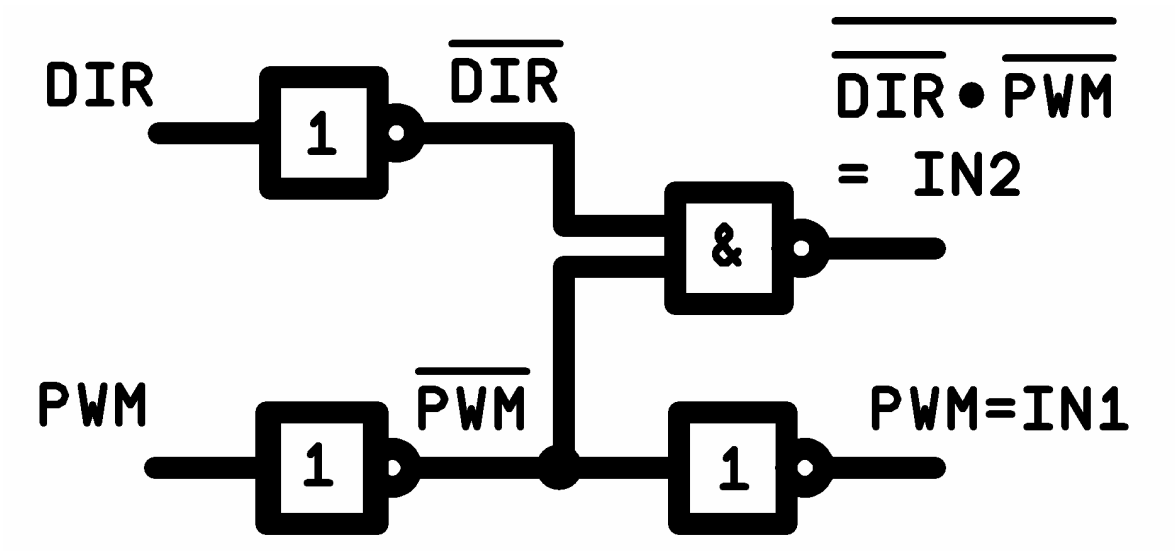
Die Ausgangssignale der Optokoppler für die Motorsteuerung sind leider durch die Schaltzeiten dieser Bauteile deutlich verformt und von idealen Rechtecksignalen weit entfernt. Deshalb wurden zur Ansteuerung NAND-Bausteine mit Schmitt-Trigger-Verhalten (74HC132) zwischengeschaltet, um die Kurvenformen zu verbessern.

15.4. Logikverknüpfung für die Ansteuerung

Sieht man sich die Verläufe der Signale IN1 und IN2 auf den vorigen Seiten an, dann kann man unter Einsatz von „De Morgan“ schreiben:

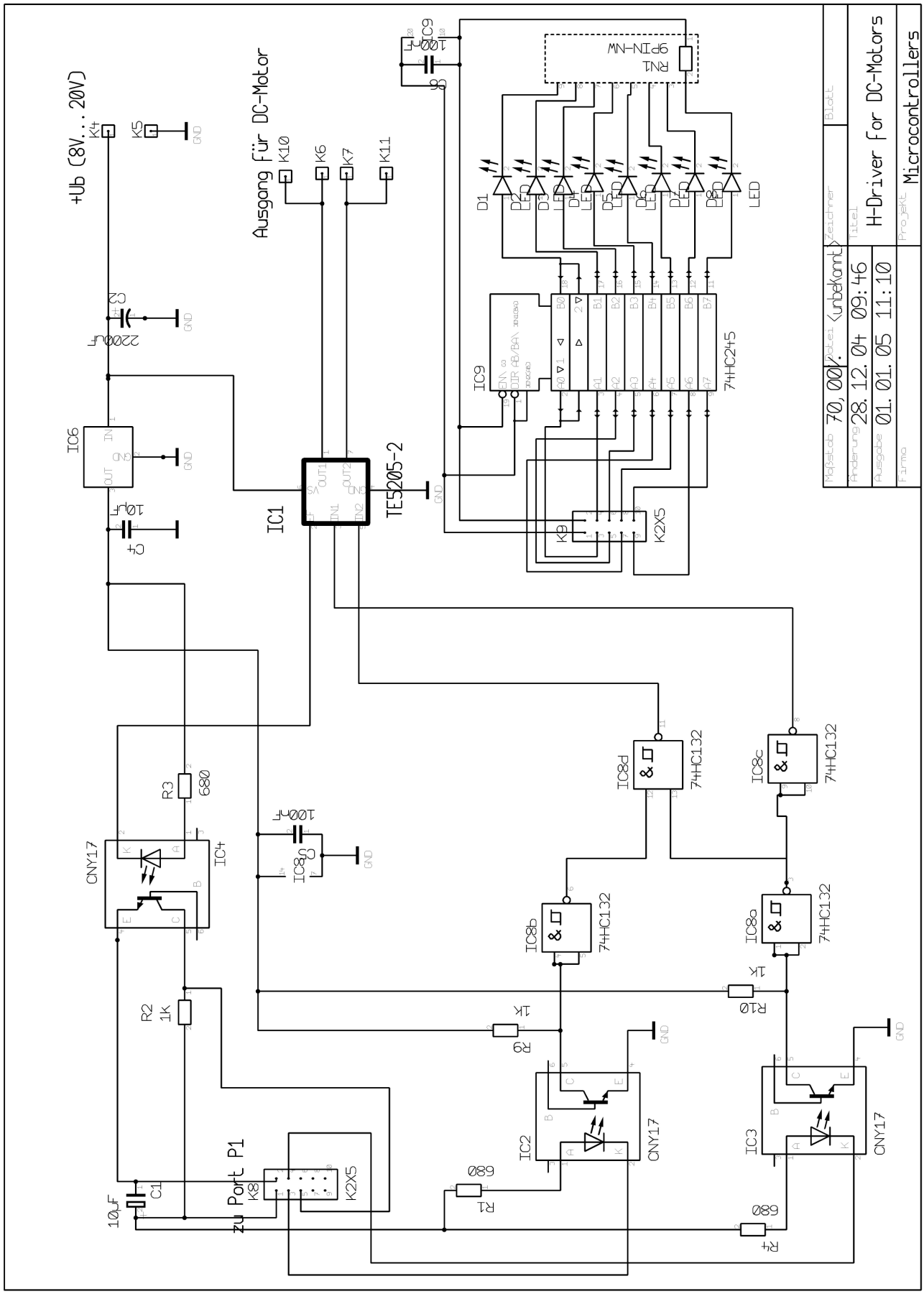
$$\mathbf{IN2} = (\mathbf{DIR}) + (\mathbf{PWM}) = \overline{\overline{\mathbf{DIR}} \bullet \overline{(\mathbf{PWM})}}$$

Um identische Signallaufzeiten zu erreichen, wurde schließlich folgende Schaltung eingesetzt:

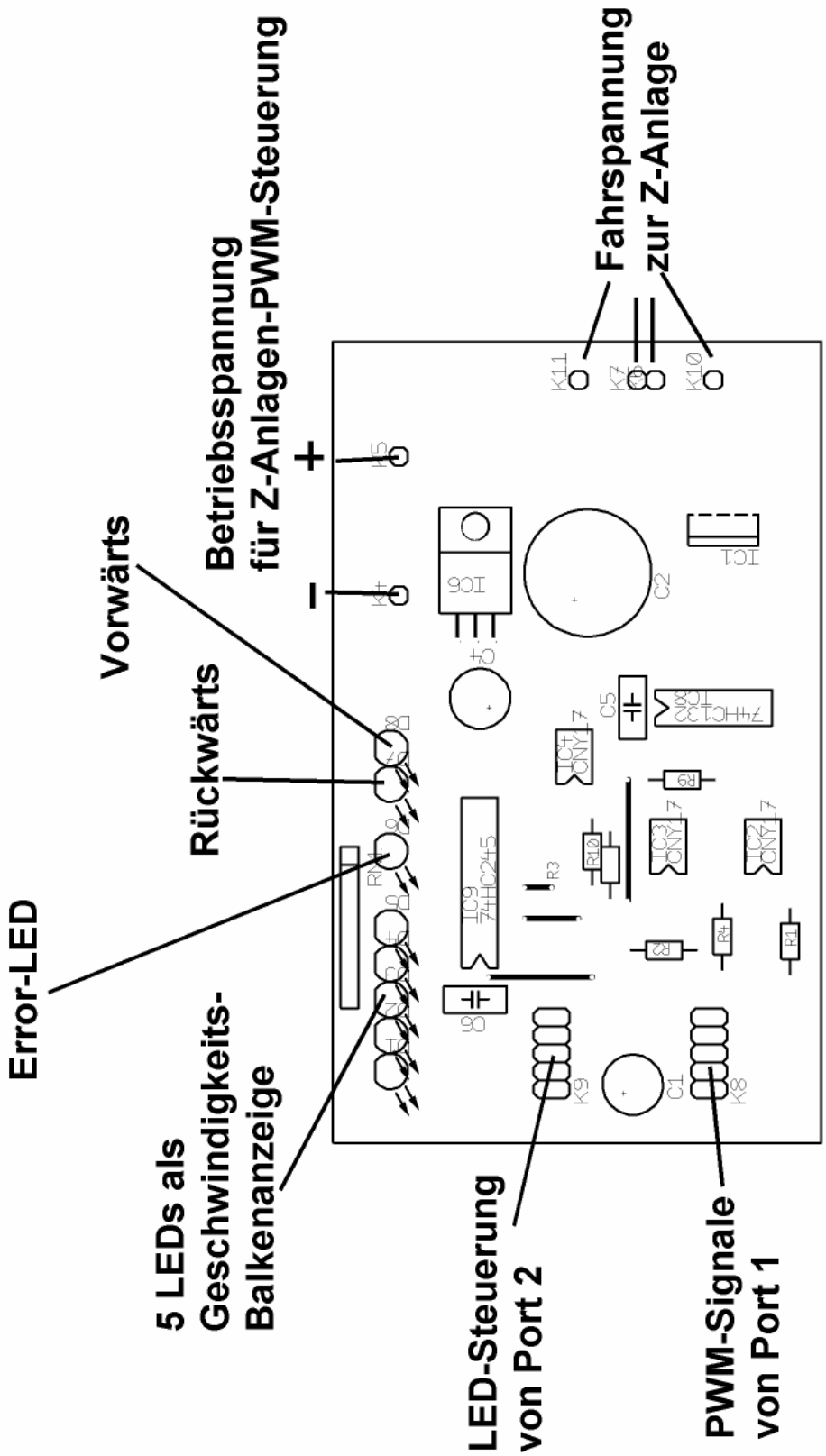


Noch ein kleines Bonbon:
Die Auswertung des vom TLE5205-2 ausgegebenen Errorflags im Mikrocontroller bringt einen hübschen Effekt, denn man erkennt nun am Aufleuchten der Error-LED, dass der Motorschaltkreis unterbrochen ist. Also erkennt man sofort, wenn die Lokomotive keinen richtigen Kontakt mit den Schienen hat.

Auch alle Anzeige-LEDs samt dem passenden Treiber 74HCT245 wurden auf die Zusatzplatine übernommen. Das nächste Blatt zeigt ihren kompletten Stromlaufplan.


+Ub (8V...20V)
Ausgang für DC-Motor
IC1
TES205-2
IC9
74HC245
IC8
74HC132
IC2
IC3
IC4
CNY17
R1
R2
R3
R4
R9
R10
R11
680
1K
10µF
2200µF
100nF
LED
9PIN-NW
K2X5
zu Port P1
H-Driver for DC-Motors
Microcontrollers
28.12.04 09:46
01.01.05 11:10
70,00%

Lageplan der PWM-Steuer-Platine:



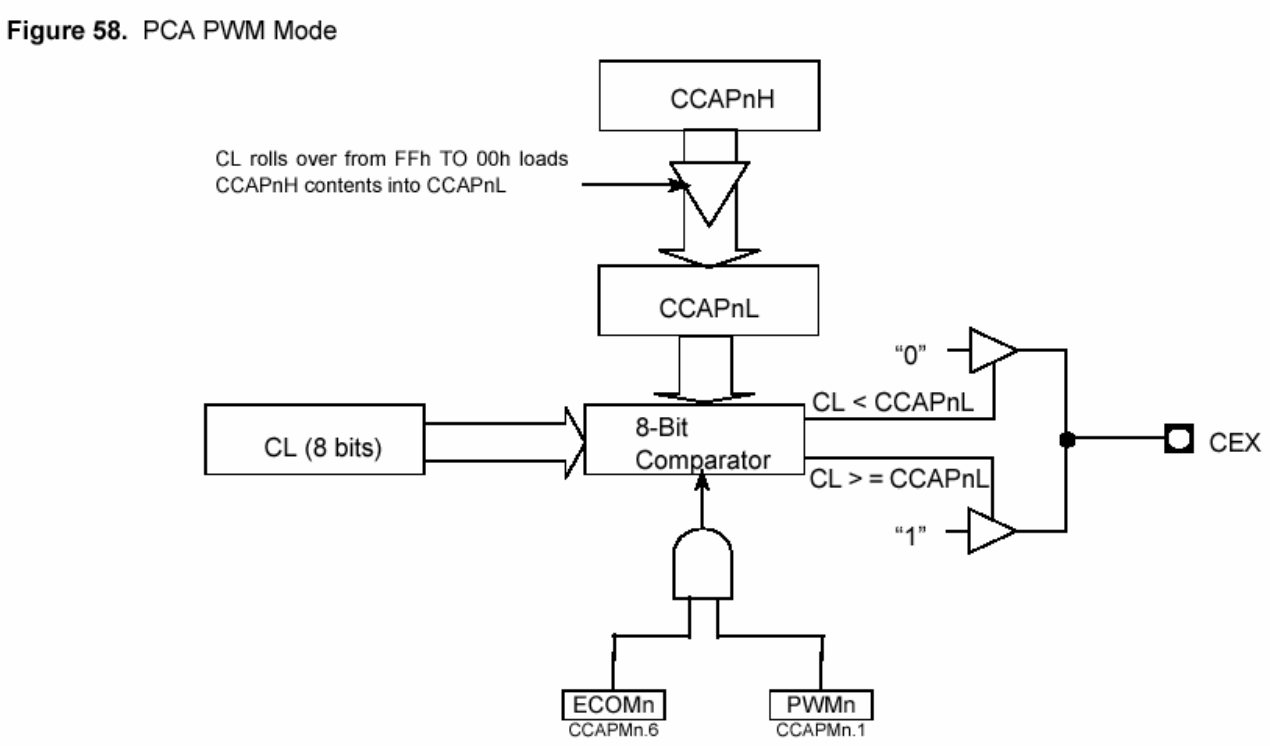
15.5. Der Software-Teil: Erzeugung von PWM-Signalen im Mikrocontroller

Hierzu wird auf

Kapitel 14.4: Praxis-Anwendung: Erzeugung von PWM-Signalen

in diesem Tutorial (Band 2) verwiesen. Deshalb folgt zuerst eine entsprechende Wiederholung des Arbeitsprinzips:

Figure 58. PCA PWM Mode



In das CCAPnL-Register wird immer automatisch der Wert des CCAOnH-Registers geladen.
Von links wird nun der augenblickliche Zählerstand des LOW-Registers im PCA-Counter angeliefert und dauernd mit dem Inhalt des CCAPnL-Registers verglichen (…bei Modul 0 würde dieses Register „CCAP0L“ heißen!). Solange der CL-Register-Inhalt kleiner ist als der CCPnL-Wert, bleibt das CEXn-Ausgang (…hier wäre es „CEX0“ = Portpin P1.3) auf LOW. Sobald jedoch der CL-Wert größer ist, wird CEXn auf HIGH gesetzt. Das CL-Register zählt aber weiter bis zum Überlauf und dieser Überlauf kopiert erneut den Reloadwert von CCAPnH nach CCAPnL. Jetzt ist CL wieder kleiner als CCAPnL und damit erhält man erneut LOW-Pegel an CEXn usw. usw.

Der Anwender muss aber folgende Initialisierungen vornehmen:

e) Clockfrequenz des PCA-Counters über die Bits CPS0 und CPS1 im CMOD-Register auswählen

f) ECOM0-Bit und PWM0-Bit im zugehörigen CCAPMn-Register werden gesetzt (= Compare-Mode und PWM-Mode werden freigegeben)

g) Reloadwert in das zum Modul gehörende CCAPnH-Register schreiben

h) Nun kann mit dem CR (= Counter Run)-Bit im Register „CCON“ der PCA-Counter gestartet werden und die PWM-Erzeugung läuft.

15.6. Programmentwicklung

15.6.1. Pflichtenheft

Es ist ein Fahrtregler für eine Märklin-Miniclub-Eisenbahn mit Gleichstrombetrieb zu programmieren. Das System verwendet hierbei eine maximale Fahrspannung von +8V. (Rückwärtsfahrt geschieht durch Umpolen der Fahrspannung am Gleis). Das erforderliche PWM-Signal wird am Portpin P1.3 (= „CEX0“-Pin) des Mikrocontrollers ausgegeben.
Zur Steuerung dienen **5 Drucktasten** unserer Zusatzplatine. Sie werden über ein Flachbandkabel an Port P0 angeschlossen und haben folgende Funktionen:

Portpin P0^0: **Rechtslauf (= Vorwärts)**
Portpin P0^1: **Linkslauf (= Rückwärts)**
Portpin P0^2: **Stopp (= Gleichmäßig bis Stillstand abbremsen)**
Portpin P0^3: **Langsamer fahren**
Portpin P0^4: **Schneller fahren**

Alle Tasten sind **LOW-aktiv.**
Die gerade gültige **Fahrtrichtung** soll durch zwei LEDs an den **Portpins P2^7 und P2^6** signalisiert werden.
Das **Errorflag** steuert eine LED an **Pin P2^5** an.
Die Portpins **P2^0 bis P2^4** versorgen eine LED-Kette mit 5 Leuchtdioden. Damit wird eine einfache **Balkenanzeige für die erzeugte Fahrspannung** verwirklicht. Bitte beachten: erst etwa ab dem Aufleuchten der vierten LED setzt sich die Lok langsam in Bewegung. Darunter wird nur die Lok-Beleuchtung aktiviert!
Die Beschleunigung beim Betätigen der Taste "schneller" ist um den Faktor 2 kleiner zu wählen als die Verzögerung beim Bremsen (= Drücken der Taste "Langsamer fahren"). „Bremsen“ und „Stopp“ ergeben dieselbe Verzögerung der Lokomotive.
Einmaliges Betätigen der **Stopptaste** senkt die **Fahrgeschwindigkeit linear bis auf Null** ab. Die bisherige Fahrtrichtung bleibt erhalten.

Wird vom **Rechtslauf auf Linkslauf** (und umgekehrt) umgeschaltet, dann wird zuerst die **Fahrgeschwindigkeit** mit der "Stopp"-Funktion **bis auf Null reduziert**, dann das **Richtungsflag invertiert** und schließlich die **neue Richtungs-LED eingeschaltet**. Mit der "Schneller"-Taste kann dann wieder beschleunigt werden.

An Port P1 ist die H-Brücken-Zusatzplatine angeschlossen. Sie arbeitet mit dem H-Bridge-Driver-IC TLE5205-2 der Firma Infineon, der über Optokoppler (zur galvanischen Trennung) mit dem Mikrocontroller kommuniziert.

Es gilt:

Portpin P1^0 = DIR = Fahrtrichtung Links oder Rechts
Portpin P1^2 = EF = Error Flag

Vorsicht und ganz wichtig:
Beim ATMEL AT89C51AC3-Controller wird das PWM-Signal am „CEX0“-Pin ausgegeben.

Der entspricht jedoch Portpin P1^3!

Folglich muss die Verdrahtung auf der bisherigen Zusatzplatinen-Version (von Hand) abgeändert werden, da bisher beim Compuboard dafür der Pin P1.1. vorgesehen war!

Die Pulsfrequenz soll im Programm leicht von etwa **4 kHz** auf ca. **12 kHz** umgestellt werden können.

15.6.2. C-Programm

Zuerst kommen (wie immer!) alle Deklarationen. Nach dem folgenden Programmstart werden zunächst alle beteiligten Register initialisiert, als Fahrtrichtung gilt zuerst „Rechts“ = „Vorwärts“.
Dann springt das Programm in eine while(1)-Endlosschleife, in der sich zwei „bedingte while-Schleifen“ für die beiden Fahrtrichtungen befinden. In jeder dieser Fahrtrichtungsschleifen werden die Bedienungstasten (Schneller / Langsamer / Stopp / Links oder Rechts) abgefragt. Geeignete zusätzliche Funktionen übernehmen dann die Ausführung des gewünschten Kommandos.

Die gewünschte Pulsfrequenz (4 kHz oder 12 kHz) wird folgendermaßen eingestellt:

Der PCA-Counter wird entweder mit einer Clockfrequenz von FPca/6 oder FPca/2 gespeist.

```
Programm-Listing
------------------------------------------------------------
// Name: Fahrtregler_01.c

// Datum: 07.01.2008 / G. Kraus

/*Aufgabe:
Es ist mit dem ATMEL AT89C51AC3 ein PWM-Fahrtregler für eine elektrische
Eisenbahn mit Gleichstrombetrieb zu programmieren. Dafür wurde eine eigene
Zusatzplatine entworfen. Die Taktfrequenz beträgt ca. 12 kHz.
Das System "Märklin - Miniclub, Spur Z" verwendet hierbei
eine maximale Fahrspannung von +8V. (Rückwärtsfahrt geschieht durch Umpolen
der Fahrspannung am Gleis). Das hierfür erforderliche PWM-Signal wird
am Portpin P1.3 (= Pin CEX0) ausgegeben.

Höhere Betriebsspannungen bis +20V und Spitzenströme bis 5 A sind
beim Treiber-IC zulässig --  so können Gleichstrombahnen der Spuren
Z / N / HO / 1 / LGB ohne jede Änderung der Schaltung versorgt werden.
Die Microcontrollersignale sind vom Leistungsteil durch Optokoppler
getrennt.

Zur Steuerung dienen 5 Drucktasten unserer entsprechenden Zusatzplatine.
Sie werden über ein Flachbandkabel an Port P5 angeschlossen und haben
folgende Funktionen:

Portpin P0^0:    Rechtslauf
Portpin P0^1:    Linkslauf
Portpin P0^2:    Stopp (= Gleichmäßig bis Stillstand abbremsen)
Portpin P0^3:    Langsamer fahren
Portpin P0^4:    Schneller fahren

Die Fahrtrichtung soll durch zwei LEDs an den Portpins P2^7 und
P2^6 signalisiert werden.

An Portpin P2^5 wird das Errorflag über eine LED sichtbar gemacht.
Unterbrochener Kontakt zwischen Lok und Schiene läßt diese Error-LED
aufleuchten.

Die Portpins P2^0 bis P2^4 dienen als Balkenanzeige für die
erzeugte Betriebsspannung. Der Lokstart beginnt etwa beim
Aufleuchten von LED 4, während beim Aufleuchten der unteren LEDs
nur die LOK-Beleuchtung aktiviert wird.

Die Beschleunigung beim Betätigen der Taste "schneller" ist
um 50% kleiner zu wählen als die Verzögerung beim Bremsen
(= Drücken der Taste "Langsamer fahren").

Einmaliges Betätigen der Stopptaste senkt die Fahrgeschwindigkeit
linear bis auf Null ab.
```

```
Wird vom Rechtslauf auf Linkslauf umgeschaltet, dann wird zuerst die
Fahrgeschwindigkeit mit der "Stopp"-Funktion bis auf Null reduziert
und erst dann die Richtungs-LED umgeschaltet. Mit der
"Schneller"-Taste kann dann wieder beschleunigt werden.
Die Pulsfrequenz soll etwa 4 kHz betragen.

An Port P1 ist die Fahrtregler-Zusatzplatine angeschlossen. Sie enthält
den H-Bridge-Treiber-IC TLE5205-2 der Firma Infinion, der über Optokoppler
(zur galvanischen Trennung) mit dem Mikrocontroller kommuniziert.
Außerdem ist eine Logikverknüpfung zum korrekten Umschalten auf "Freilauf"
nach jedem PWM-Puls erforderlich (erforderlich ist dann High-Pegel an
den beiden Eingängen IN1 und IN2).

Es gilt bei Port 1:

Portpin P1^0 = DIR = Fahrtrichtung (LOW ergibt Rechtslauf).
                     Steuert Pin IN2 des TLE5205 an.

Portpin P1^2 = EF =  Error Flag. Wird vom TLE5205 über einen
                     Optokoppler an den Controller gesendet.

Portpin P1^3 = PWM = Drehzahl = CEX0-Pin (Längerer LOW-Pegel ergibt
                     höhere Drehzahl). Steuert Pin IN1 des
                     TLE 5205 an.
----------------------
Es gilt bei Port 2:

Portpin P2^7 =   LED für "Vorwärts"

Portpin P2^6 =   LED für "Rückwärts"

Portpin P2.5  =  Error LED

Portpin P2^0 bis P2^4 = Balkenanzeige für Fahrspannung

Achtung: Beim TL5205 gilt folgende Funktionstabelle für
         die beiden Eingänge IN1 und IN2

         LL = Rechtslauf
         LH = Linkslauf
         HL = Bremsen
         HH = Freilauf

-------------------------------------
Programmbeschreibung
-------------------------------------
Name:         PWM_5205_4.c
Funktion:     Fahrtregler für Märklin-Miniclub-Lokomotiven

Datum:        07.01.2008
Autor:        G. Kraus

-----------------------------------------------------------------------
Deklarationen und Konstanten
-----------------------------------------------------------------------*/

#include <t89c51ac2.h>    // Header für AT89C51AC3
#include <stdio.h>        // Standard-Eingabe-Ausgabe Header

sbit rechts=P0^0;         // P0.0 = Rechtslauf
sbit links=P0^1;          // P0.1 = Linkslauf
sbit stopp=P0^2;          // P0.2 = Stopptaste
```

```
sbit lahmer=P0^3;          // P0.3 = langsamer fahren
sbit schneller=P0^4;    // P0.4 = schneller fahren

sbit LEDre=P2^7;      // Richtungs-LED an P2.7: rechts / HIGH-aktiv
sbit LEDli=P2^6;      // Richtungs-LED an P2.6: links / HIGH-aktiv
sbit ErrorLED=P2^5;          // Error-LED an P2.5
sbit L4=P2^4;                // Betriebsspannungsanzeige mit
sbit L3=P2^3;                // fünf LEDs an den Pins P2.0 bis P2.4
sbit L2=P2^2;                // ACHTUNG:bei den unteren drei LEDs
sbit L1=P2^1;                // beginnen nur die Lichter an der Lok zu
sbit L0=P2^0;                // leuchten. Lokstart erst etwa ab LED Nr. 4

sbit DIR=P1^0;          // Input 2 (= Drehrichtung. LOW gibt Rechtslauf)
sbit PWM=P1^3;          // PWM-Ausgang des Controllers
sbit EF=P1^2;           // Error-Flag-Eingang für den Mikrocontroller

/*--------------------------------------
Prototypen
----------------------------------------*/
void PWM_INI(void);
void zeit_ms(unsigned char z);
void changeleft(void);
void changeright(void);
void STOP(void);
void vLEDs(void);
/*--------------------------------------
Hauptprogramm
----------------------------------------*/
void main(void)
{    AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen

     P0=0xFF;           // Port P0 auf "Lesen" schalten
     P2=0;              // Alle LEDs löschen
     PWM_INI();         // PWM initialisieren
     EF=1;              // EF-Eingang auf „Einlesen“ schalten

     DIR=0;             // Richtungsübergabe an TL5205: Rechts
     LEDre=1;           // Rechts-LED ein

     while(1)
     {
          while(DIR==0)      // Rechtslauf
          {      while((schneller==0)&&(CCAP0H<0xFF))  // schneller fahren
                 {    CR=1;              // PWM läuft
                      CCAP0H++;          // Pulslänge erhöhen
                      zeit_ms(12);       // Warten
                      vLEDs();           // LED-Fahrspannungs-Anzeige
                      ErrorLED=~EF;      // Error-LED aktualisieren
                 }


                 while((lahmer==0)&&(CCAP0H>0x00))   // langsamer fahren
                 {    CCAP0H--;          // Pulslänge vermindern
                      zeit_ms(10);       // Warten
                      vLEDs();           // LED-Fahrspannungs-Anzeige
                      ErrorLED=~EF;      // Error-LED aktualisieren
                 }

                 if(stopp==0)            // sanft anhalten
                 {    STOP();
                 }
```

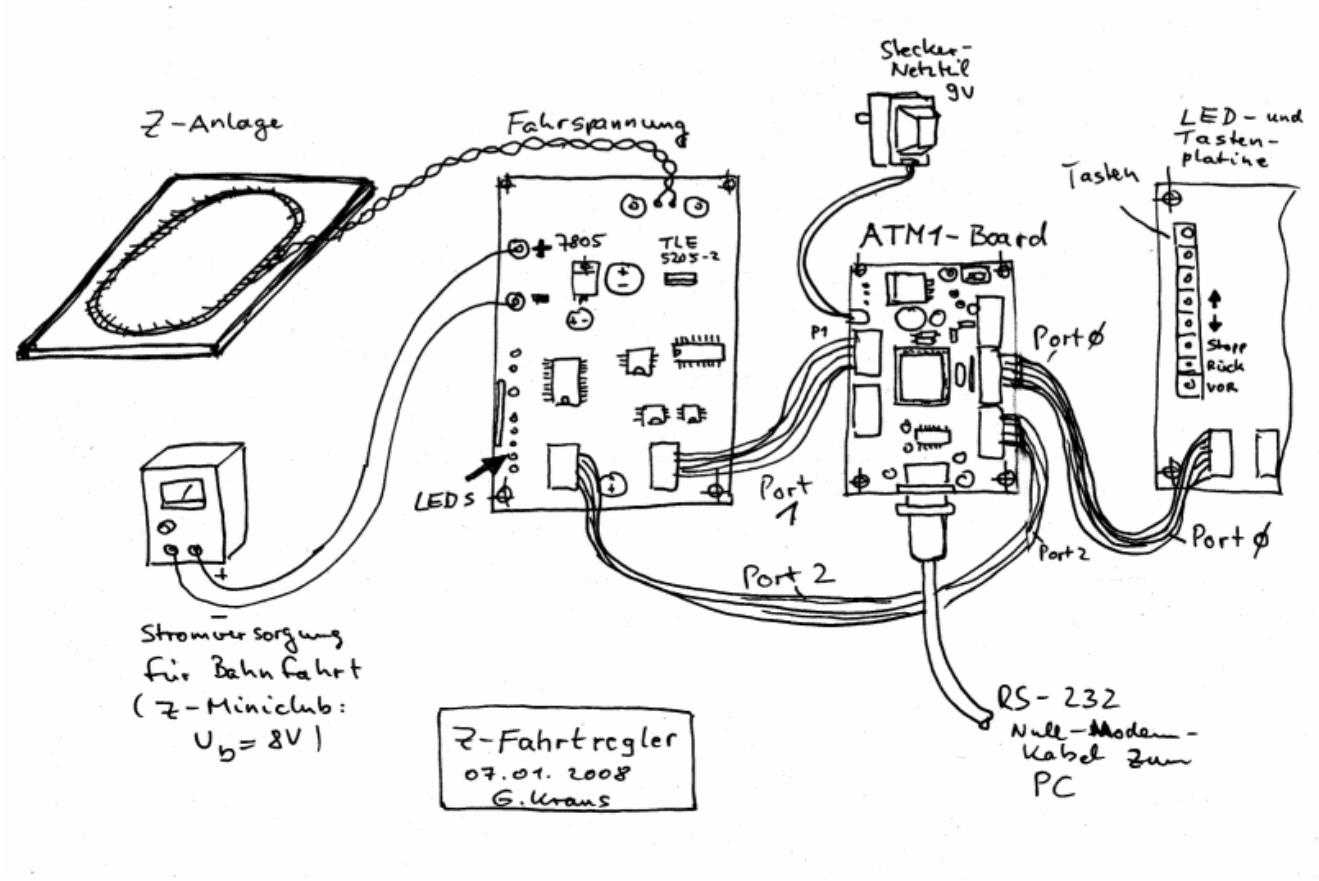
```
                if(links==0)       // Fahrtrichtung umkehren (auf links)
                {    changeleft();
                }

                ErrorLED=~EF;      // Error-Flag an Error-LED übergeben
          }


          while(DIR==1)            // Linkslauf
          {
                while((schneller==0)&&(CCAP0H<0xFF))  // schneller fahren
                {      CR=1;               // PWM läuft
                       CCAP0H++;
                       zeit_ms(10);
                       vLEDs();            // LED-Fahrspannungs-Anzeige
                       ErrorLED=~EF;       // Error-LED aktualisieren
                }


                while((lahmer==0)&&(CCAP0H>0x00))   // langsamer fahren
                {      CCAP0H--;
                       zeit_ms(7);
                       vLEDs();            // LED-Fahrspannungs-Anzeige
                       ErrorLED=~EF;       // Error-LED aktualisieren
                }

                if(stopp==0)               // sanft anhalten
                {      STOP();
                }

                if(rechts==0)    // Fahrtrichtung umkehren (auf rechts)
                {      changeright();
                }

                ErrorLED=~EF;    // Error-Flag an Error-LED übergeben
          }
     }
}
/*---------------------------------------------------------------------
Zusatzfunktionen
---------------------------------------------------------------------*/
void zeit_ms(unsigned char z)  // Verzögerungszeit = z * 1 Millisekunde
{      int x;
       char y;

       for(y=0;y<z;y++)
       {      for(x=0;x<=250;x++);
       }
}


void STOP(void)                    // Stopp-Funktion
{      while(CCAP0H>0x00)          // Solange Pulsbreite noch nicht Null:
       {      CCAP0H--;           // Dekrementiere CCH1-Register
              zeit_ms(12);        // Warte 10 Millisekunden
              vLEDs();            // LED-anzeige aktualisieren
              ErrorLED=~EF;       // Error-LED aktualisieren
       }

       CR=0;                       // PWM-Signal ausschalten
}
```

```
void changeleft(void)
{       STOP();

        CCAP0H=0x00;        // Null Volt Fahrspannung vorsehen
        LEDre=0;            // Rechts-LED aus
        LEDli=1;            // Links-LED an
        DIR=1;              // Richtungsbit auf "Links" setzen
}



void changeright(void)
{       STOP();

        CCAP0H=0x00;        // Null Volt Fahrspannung vorsehen
        LEDre=1;            // Rechts-LED an
        LEDli=0;            // Links-LED aus
        DIR=0;              // Richtungsbit auf "Rechts" setzen

}


void vLEDs(void)
{       if(CCAP0H>=250)
        {    L4=1;          // alle fünf LEDs für die
             L3=1;          // Geschwindigkeit leuchten
             L2=1;
             L1=1;
             L0=1;
        }

        if((CCAP0H>=200)&&(CCAP0H<230))
        {    L4=0;          //die unteren vier LEDs
             L3=1;          //für die Geschwindigkeit
             L2=1;          //leuchten
             L1=1;
             L0=1;
        }


        if((CCAP0H>150)&&(CCAP0H<200))
        {    L4=0;          //die unteren drei LEDs
             L3=0;          //für die Geschwindigkeit
             L2=1;          //leuchten
             L1=1;
             L0=1;
        }

        if((CCAP0H>100)&&(CCAP0H<150))
        {    L4=0;          //Beide unteren LEDs leuchten
             L3=0;
             L2=0;
             L1=1;
             L0=1;
        }

        if((CCAP0H>50)&&(CCAP0H<100))
        {    L4=0;          //Unterste Geschwindigkeits-LED
             L3=0;          //leuchtet
             L2=0;
             L1=0;
```

```
            L0=1;
        }

        if(CCAP0H<50)
        {    L4=0;          //Alle Geschwindigkeits-LEDs sind dunkel
             L3=0;
             L2=0;
             L1=0;
             L0=0;
        }
}

void PWM_INI(void)
{       // CMOD=0x02;       // Interner Clock FPsa/2  --  gibt 12 kHz
        CMOD=0x00;     // Interner Clock FPsa/6  --  gibt 4 kHz

        CCON=0x00;       // PCA steht still
        CCAPM0=0x42;     // ECOM0-Bit und PWM0-Bit werden gesetzt
        CCAP0H=0;        // Stillstand beim Start
}
```

15.7. Verdrahtungsskizze der kompletten Anlage

Wer sich so etwas bauen will, kann damit die erforderliche Verdrahtung vornehmen:



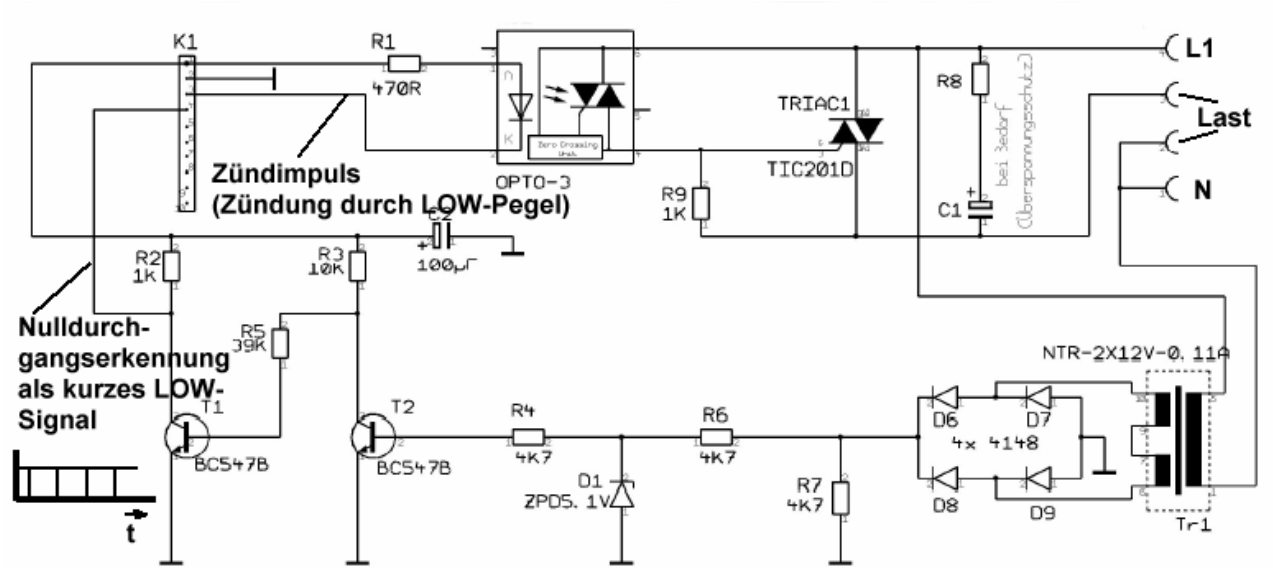
16. Unterrichtsprojekt „Phasen-Anschnittssteuerung“

16.1. Einführung

In diesem Projekt wird demonstriert, wie eine Last am Wechselstromnetz (230 V / 50Hz) durch „Phasen-Anschnittssteuerung“ mit unterschiedlichen Leistungen versorgt werden kann. Diese Leistung ist vom Controller aus steuer- oder regelbar. Außer dem ATM1-Board ist dazu noch die Zusatzplatine „Triac-Steuerung“ erforderlich.

16.2. Die Zusatzplatine „Triac-Steuerung“

Werfen wir einen Blick auf den Stromlaufplan:



Im rechten oberen Eck sehen wir den Netz-Eingang (L1 / N) und die Anschlüsse für die Last. Diese Last liegt in Reihe zum Triac 1, der über einen „Opto-Triac“ durch den Mikrocontroller gezündet wird. Durch den Opto-Triac ist der komplette Mikrocontroller galvanisch vom Lastkreis getrennt. Die Bauteile C1 und R8 dienen als Überspannungsschutz.
Parallel zum Netzeingang ist ein Print-Trafo zur Netztrennung angeordnet, dessen Sekundärwicklung eine Gesamt-Ausgangsspannung von 24 V bereitstellt. Diese Spannung wird vom Brückengleichrichter mit den Dioden D6…D9 gleichgerichtet, wobei nur ein Lastwiderstand, aber KEIN Ladekondensator vorhanden ist. An R7 erhält man deshalb die bekannte Folge von Sinushalbwellen mit der Frequenz f = 100 Hz. Die Zenerdiode D1 begrenzt diese Sinushalbwellen auf einen Spitzenwert von ca. +5V und deshalb ist an ihr die Spannung schon nahezu rechteckförmig. Zwei nachgeschaltete Verstärkerstufen (mit T1 und T2) erzeugen daraus eine Impulsspannung mit U_{max} = +5V, die **nur beim Nulldurchgang der Eingangs-Netzspannung für wenige Mikrosekunden bis auf Null Volt absinkt.** Dieses Signal bildet die **Referenz für den im Controller erzeugten phasen-verschobenen Triac-Zündimpuls**.

Der **10polige Pfostenfeldstecker K1** wird über ein Flachbandkabel mit **Port 1** des Mikrocontrollers AT89C51AC3 verbunden. Der liefert am **Portpin P1^0 den Zündimpuls für den Triac** und erhält dazu an **Portpin P1^0 das Nulldurchgangs-Referenzsignal.**

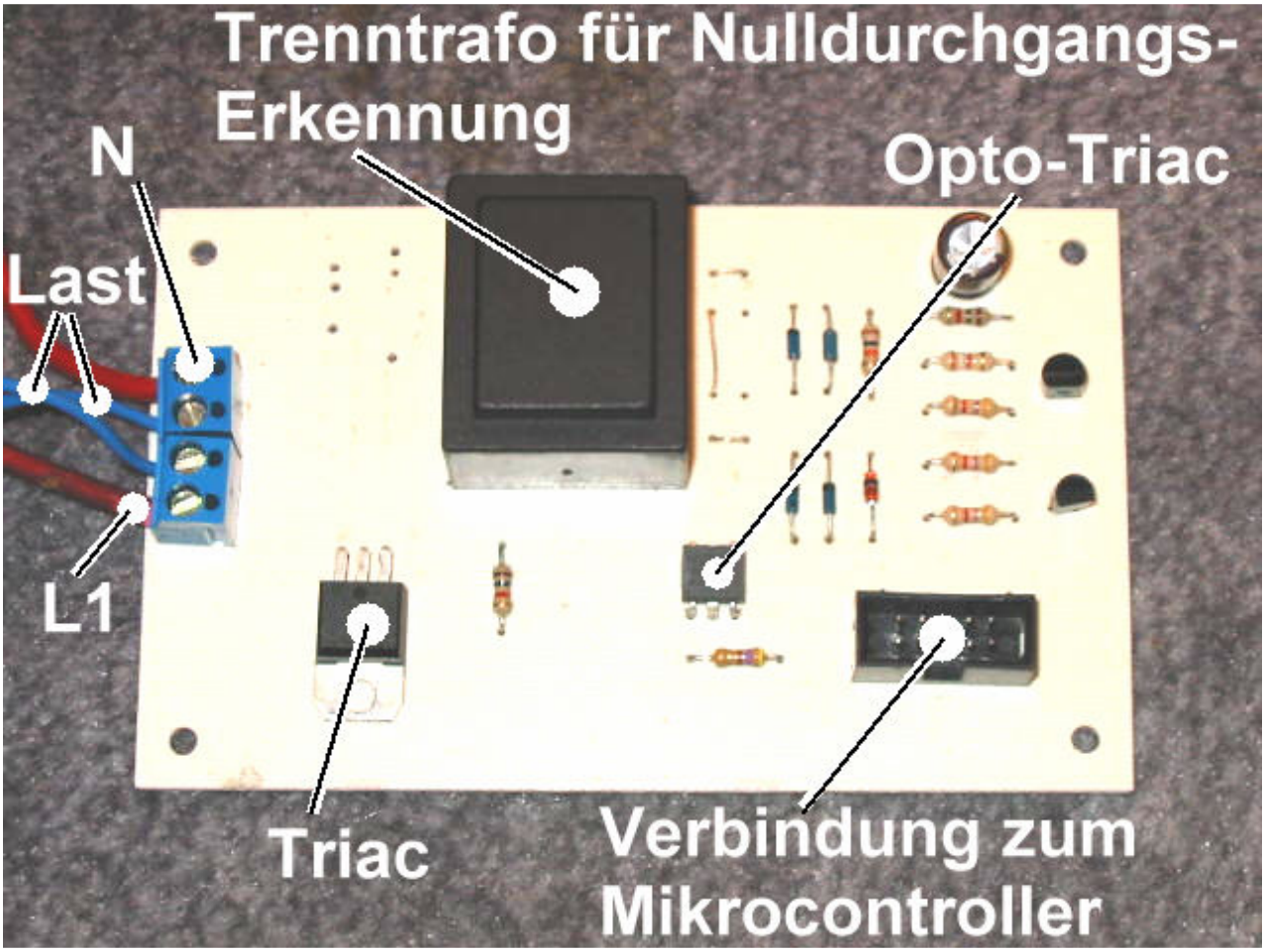
Lastkreis und Steuerkreis sind also komplett galvanisch getrennt. Trotzdem ist eine ernsthafte Warnung nötig:

Bitte beim Einsatz (im Laborversuch) alle erforderlichen Sicherheitsmaßnahmen treffen,

um eine Personengefährdung auszuschließen!

Dazu gehören z. B. die Versorgung der Lastseite über einen Trenntrafo oder Regeltrenntrafo, die genaue Kontrolle der aufgebauten Schaltung durch den Lehrer vor der Inbetriebnahme, eine ausführliche Sicherheitsbelehrung zum Thema Berührungsschutz / Schutzmaßnahmen…usw,

So sieht die Triac-Platine aus und so findet man sich darauf zurecht:



16.3. Programm-Entwicklung einer Drucktasten-Steuerung

16.3.1. Pflichtenheft

Mit Hilfe der Drucktastenplatine sollen 8 verschiedene Zündwinkel (= „Nullphasenwinkel“) zwischen 20 Grad und 160 Grad ausgewählt werden können. Die Schrittweite beträgt also 20 Grad von Taste zu Taste.

An Portpin P1^0 wird der erzeugte Zündimpuls ausgegeben (= Absenkunkung von HIGH auf LOW für eine Dauer von ca. 50 Mikrosekunden. Periodendauer = 10 Millisekunden), an Portpin P1^1 wird dagegen die Referenzspannung für den Nulldurchgang eingelesen (= ebenfalls in Form einer Absenkung von HIGH auf LOW für ca. 250 Mikrosekunden, Periodendauer = 10 Millisekunden).

Als Last soll eine Glühlampe mit den Daten „230 V / 40 W“ verwendet werden. Die Speisung der Lampenschaltung erfolgt über einen Trenntrafo.

16.3.2. Vorgehen bei der Programmentwicklung und den Tests

Es empfiehlt sich, zuerst mit einem zweiten Controllerboard das Nulldurchgangs-Signal künstlich zu erzeugen. Dieses Signal wird an **Portpin P2^0 ausgegeben** und dort getestet.

Dann wird es über ein 10poliges Flachbandkabel an den eigentlichen Steuercontroller übergeben, der es **ebenfalls an seinem Portpin P2^0** einliest und damit die geforderte Phasenanschnitts-Steuerung per Drucktasten erzeugen kann. Es empfiehlt sich hierbei, an einem **zusätzlichen Testausgang (z. B. P1^7)** die korrekte Arbeitsweise des Phasenverschiebe-Programms zusammen mit dem Zündimpuls an Portpin P1^0 mit einem Zweikanal-Oszilloskop genau zu prüfen.

Wenn alles funktioniert, geht es an den eigentlichen Hardwaretest mit der Glühlampe als Last. Allerdings muss dazu erst im Programm der **Nulldurchgangs-Eingang auf Portpin P1^1 umgestellt werden**, da die Triac-Zusatzplatine das entsprechende Signal an diesen Portpin liefert.

16.3.3. Testgenerator-Programm zur Simulation der Nulldurchgangs-Erkennung

```
/*------------------------------------------------------------
Programmbeschreibung
------------------------------------------------------------
Name:     ref_01.c
Funktion:
Der im 16 Bit-Reload-Modus betriebene Timer 2 erzeugt alle 10 Millisekunden
einen kurzen negativen Impuls an Portpin P2.0. (Dieses Signal wird an einen
zweiten Controller weitergegeben und dort ebenfalls an Portpin P2.0 als
Nulldurchgangssignal eingelesen).

Datum:    08. 01. 2008
Autor:    G. Kraus

------------------------------------------------------------------------
Deklarationen und Konstanten
------------------------------------------------------------------------*/

#include <t89c51ac2.h>
#include <stdio.h>

sbit Zero_detect=P2^0;    // Ausgangspin für Nulldurchgangs-Signal

/*------------------------------------------------------------------------
Hauptprogramm
------------------------------------------------------------------------*/

void main(void)
{       P2=0x00;          // Port P2 löschen
        C_T2=0;           // Timer-Betrieb
        RCLK=0;           // Timer 2 nicht für Serial Port verwenden
```

```
        TCLK=0;          //     "    "    "    " "
        CP_RL2=0;        // Autoreload bei Überlauf
        RCAP2L=0xEF;     // Reload-Wert ist 55535 für 10 Millisekunden
        RCAP2H=0xD8;     // (= 0xD8EF)
        TF2=0;           // Überlaufflag löschen
        TR2=1;           // Timer läuft

        Fire=1;          // Zünd-Ausgang auf HIGH

        while(1)
        {       while(TF2==0);      // Auf Timer-Überlauf warten
                TF2=0;              // Überlauf-Flag zurücksetzen

                for(y=0;y<10;y++)
                {       Zero_detect=0;      // Ausgang auf LOW
                }

                Zero_detect =1;             // Ausgang wieder auf HIGH
        }
}
```

==========

16.3.4. Testprogramm für die Phasenanschnitts-Steuerung

```
/*------------------------------------------------------------------
Programmbeschreibung
------------------------------------------------------------------
Name:         triac_01.c

Funktion:
An Portpin P2^0 wird ein "100Hz - Nulldurchgangssignal" eingelesen, das von
einem weiteren Controller erzeugt wird.
Mit dem Controller sollen nun an Portpin P1^0 LOW-aktive Zündimpulse
für den Triac im Lastkreis erzeugt werden. Mit 8 Drucktasten an Port P0
lassen sich die Zündwinkel 20 / 40 / 60 / 80 / 100 / 120 / 140 / 160 Grad
als Verzögerungen gegenüber dem Nulldurchgang für eine Halbwelle mit der
Dauer von 10 ms  einstellen.
Ein Testausgang (Pin P1^7) wird beim Eintreffen eines „Zero-Crossing-
Signals“ an Pin P2^0 auf LOW gesetzt. Dann läuft die Verzögerungszeit ab
(...sie wird durch die einzelnen Drucktasten ausgewählt!) und sobald sie
beendet ist, zündet der Triac (...kann nun an Pin P1^0 kontrolliert
werden). Gleichzeitig wird der Testausgang wieder auf HIGH zurückgeschaltet
und so ist die erzeugte Verzögerung hier gut zu sehen und zu messen.


Datum:        08. 01. 2008
Autor:        G. Kraus

------------------------------------------------------------------
Deklarationen und Konstanten
------------------------------------------------------------------*/

#include<t89c51ac2.h>
#include<stdio.h>

sbit Fire=P1^0;             // Pin P1^0 als Zündausgang
sbit Zero=P2^0;             // Pin P2^0 zur Nulldurchgangs-Erkennung
sbit Testausgang=P1^7;      // Testausgang an Pin P1^7

sbit t20deg=P0^0;           // Tastendeklarationen für Zündwinkel
```

```
sbit t40deg=P0^1;
sbit t60deg=P0^2;
sbit t80deg=P0^3;
sbit t100deg=P0^4;
sbit t120deg=P0^5;
sbit t140deg=P0^6;
sbit t160deg=P0^7;
/*---------------------------------------------------------------------
Prototypen
---------------------------------------------------------------------*/

void zuenden(unsigned char grad);     // Prototyp
void INI_Timer1(void);

/*---------------------------------------------------------------------
Hauptprogramm
---------------------------------------------------------------------*/
void main(void)
{       Fire=1;                           // Zündausgang auf HIGH
        Zero=1;                           // Zero-Crossing-Input auf HIGH
        INI_Timer1();                     // Timer 1 initialisieren

        while(1)
        {
                while(t20deg==0)          // Solange Taste gedrückt:
                {       while(Zero==1);   // Auf Nulldurchgang warten
                        Testausgang=0;    // Testausgang auf LOW
                        zuenden(20);      // Verzögerte Zündung auslösen
                        Testausgang=1;    // Testausgang wieder auf HIGH
                }

                while(t40deg==0)
                {       while(Zero==1);
                        Testausgang=0;
                        zuenden(40);
                        Testausgang=1;
                }

                while(t60deg==0)
                {       while(Zero==1);
                        Testausgang=0;
                        zuenden(60);
                        Testausgang=1;
                }

                while(t80deg==0)
                {       while(Zero==1);
                        Testausgang=0;
                        zuenden(80);
                        Testausgang=1;
                }

                while(t100deg==0)
                {       while(Zero==1);
                        Testausgang=0;
                        zuenden(100);
                        Testausgang=1;
                }

                while(t120deg==0)
                {       while(Zero==1);
                        Testausgang=0;
```

```
                        zuenden(120);
                        Testausgang=1;
                }

                while(t140deg==0)
                {       while(Zero==1);
                        Testausgang=0;
                        zuenden(140);
                        Testausgang=1;
                }

                while(t160deg==0)
                {       while(Zero==1);
                        Testausgang=0;
                        zuenden(160);
                        Testausgang=1;
                }
        }
}

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

void INI_Timer1(void)
{       TMOD=0x20;              // Timer1 im 8 Bit-Reload-Betrieb (Mode 2)
        TL1=0;                  // Start-Wert ist 0
        TH1=209;                // Reload-Wert ist 209
        EA=1;                   // Alle Interrupts freigeben
        ET1=1;                  // Timer1 - Interrupt freigeben
        TF1=0;                  // Timer1-Überlaufflag vorsichtshalber löschen
        TR1=0;                  // Timer1 steht still
}

void zuenden(unsigned char grad)
{       unsigned char x,y;
        for(x=0;x<grad;x++)
        {    TR1=1;             // Timer starten
             while(TF1==0);     // Zeit für "1 Grad" ablaufen lassen
             TR1=0;             // Timer wieder stoppen
             TF1=0;             // Überlaufflag wieder löschen

        }

        for(y=0;y<10;y++)
        {    Fire=0;        // Zündimpuls, Dauer ca. 50 Mikrosekunden
        }

        Fire=1;             // Zündung beendet
}
```

16.3.5. Programm für das vollständige Projekt „Drucktasten-Phasenanschnitts-Steuerung“

```
/*-------------------------------------------------------------
Programmbeschreibung
-------------------------------------------------------------
Name:         triac_02.c

Nun wird an Portpin P1^1 das von der Triac-Zusatzplatine kommende "100Hz -
Nulldurchgangssignal" eingelesen.
Der Controller erzeugt damit an Portpin P1^0 LOW-aktive Zündimpulse mit ca.
50 Mikrosekunden Dauer für den Triac im Lastkreis.
Mit 8 Drucktasten an Port P0 lassen sich die Zündwinkel 20 / 40 / 60 / 80 /
100 / 120 / 140 / 160 Grad als Verzögerungen (gegenüber dem Nulldurchgang
bei einer Halbwelle mit der Dauer von 10 ms)  einstellen.

Der Testausgang ist nun überflüssig.

Datum:        08. 01. 2008
Autor:        G. Kraus

-------------------------------------------------------------------
Deklarationen und Konstanten
-------------------------------------------------------------------*/

#include<t89c51ac2.h>
#include<stdio.h>

sbit Fire=P1^0;             // Pin P1^0 als Zündausgang
sbit Zero=P1^1;             // Pin P1^1 zur Nulldurchgangs-Erkennung

sbit t20deg=P0^0;           // Tastendeklarationen für Zündwinkel
sbit t40deg=P0^1;
sbit t60deg=P0^2;
sbit t80deg=P0^3;
sbit t100deg=P0^4;
sbit t120deg=P0^5;
sbit t140deg=P0^6;
sbit t160deg=P0^7;
/*-------------------------------------------------------------------
Prototypen
-------------------------------------------------------------------*/

void zuenden(unsigned char grad);   // Zündfunktion
void INI_Timer1(void);              // Timer-Initialisierung

/*-------------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------------*/
void main(void)
{      Fire=1;                          // Zündausgang auf HIGH
       Zero=1;                          // Zero-Crossing-Input auf HIGH
       INI_Timer1();                    // Timer 1 initialisieren

       while(1)
       {
              while(t20deg==0)          // Solange Taste gedrückt:
              {      while(Zero==1);    // Auf Nulldurchgang warten
                     zuenden(20);       // Verzögerte Zündung auslösen
              }
```

```
                while(t40deg==0)
                {       while(Zero==1);
                        zuenden(40);
                }

                while(t60deg==0)
                {       while(Zero==1);
                        zuenden(60);
                }

                while(t80deg==0)
                {       while(Zero==1);
                        zuenden(80);
                }

                while(t100deg==0)
                {       while(Zero==1);
                        zuenden(100);
                }

                while(t120deg==0)
                {       while(Zero==1);
                        zuenden(120);
                }

                while(t140deg==0)
                {       while(Zero==1);
                        zuenden(140);
                }

                while(t160deg==0)
                {       while(Zero==1);
                        zuenden(160);
                }
        }
}

/*---------------------------------------------------------------------------
Zusatzfunktionen
---------------------------------------------------------------------------*/
void INI_Timer1(void)
{       TMOD=0x20;              // Timer1 im 8 Bit-Reload-Betrieb (Mode 2)
        TL1=0;                  // Start-Wert ist 0
        TH1=209;                // Reload-Wert ist 209
        TF1=0;                  // Timer1-Überlaufflag vorsichtshalber löschen
        TR1=0;                  // Timer1 steht still
}

void zuenden(unsigned char grad)
{       unsigned char x,y;
        for(x=0;x<grad;x++)
        {    TR1=1;             // Timer starten
             while(TF1==0);     // Zeit für "1 Grad" ablaufen lassen
             TR1=0;             // Timer wieder stoppen
             TF1=0;             // Überlaufflag wieder löschen
        }

        for(y=0;y<10;y++)
        {    Fire=0;            // Zündimpuls, Dauer ca. 50 Mikrosekunden
        }

        Fire=1;                 // Zündung beendet
}
```

16.4. Programm-Entwicklung einer Poti-Steuerung („Dimmer“)

Aufgabe:

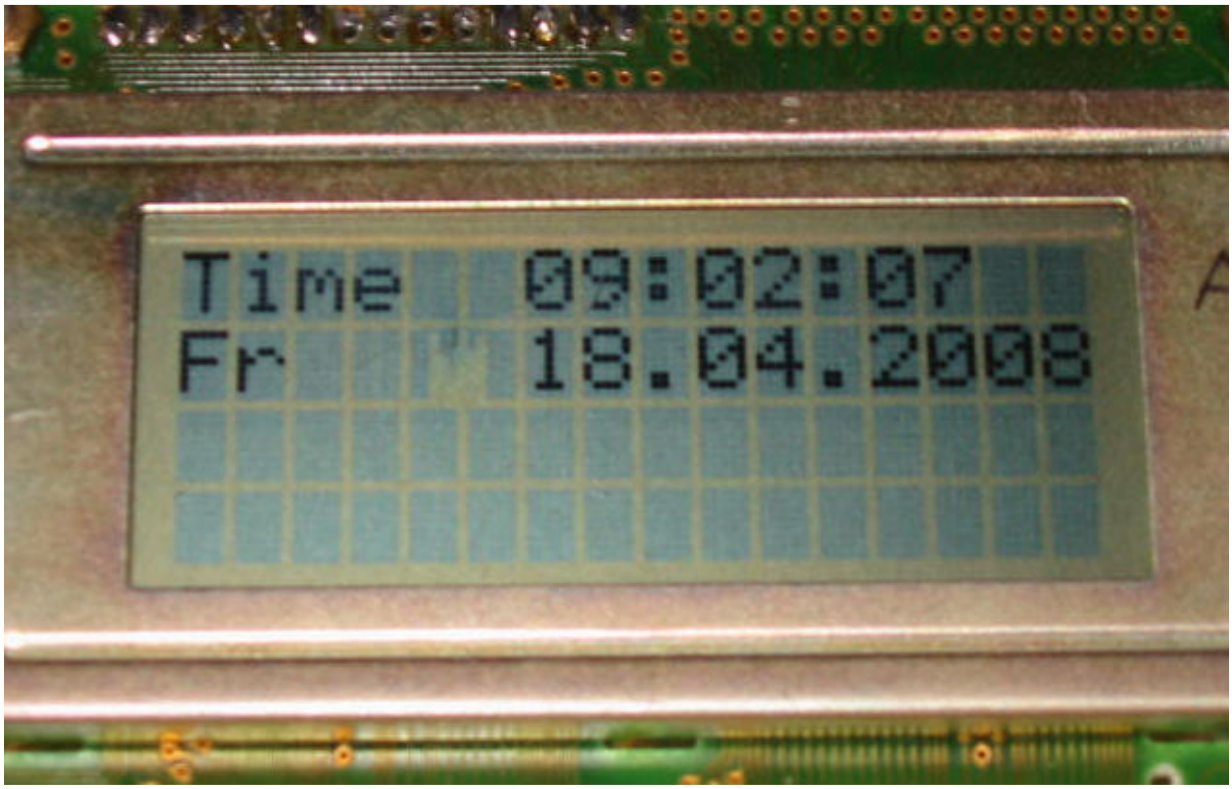
An Portpin P2^1 wird das von der Triac-Zusatzplatine kommende "100Hz - Nulldurchgangssignal" eingelesen.
Der Controller erzeugt damit an Portpin P2^0 LOW-aktive Zündimpulse für den Triac im Lastkreis mit einer Dauer von ca 50 Mikrosekunden.
Mit dem Einstellpoti auf der DA-Wandlerplatine wird nun eine Gleichspannung von 0...+3V zur Verfügung gestellt und am Analogeingang AN0 (Pin P1.0) gemessen. Damit läßt sich der Zündwinkel stufenlos verändern.
Es soll gelten: Null Volt am Poti ergeben "Ganz Dunkel".
Drei Volt am Poti ergeben "Ganz hell".

Lösung:
Nach den Initialisierungen von Timer 1 und dem AD-Wandler geht das Programm in eine Endlosschleife. Darin wird zuerst die vom Poti kommende Analogspannung gemessen und ihr Messwert invertiert, um die „Drehrichtung des Potis anzupassen“. Nun wird auf den Nulldurchgang der Netzspannung gewartet. Ist er erkannt, läuft zuerst eine Verzögerung ab (um den Nulldurchgang komplett zu überspringen) und darnach wird der AD-Meßwert in eine passende Zeitverzögerung umgesetzt. Nach dieser Zeitverzögerung erfolgt die Zündung des Triacs. Dann wird wieder die Poti-Einstellung ermittelt, der Nulldurchgang detektiert…..usw.

```
/*
------------------------------------------------------------------------
Deklarationen und Konstanten
------------------------------------------------------------------------*/
#include<t89c51ac2.h>
#include<stdio.h>

sbit Fire=P2^0;         // Pin P2^0 als Zündausgang
sbit Zero=P2^1;         // Pin P2^1 zur Nulldurchgangs-Erkennung

unsigned char z;        // Erforderliche Zwischenspeicher
unsigned int w;
/*------------------------------------------------------------------------
Prototypen
------------------------------------------------------------------------*/
void zuenden(void);
void INI_Timer1(void);
void wait_ADC_ini(void);
void Timer_1_ISR(void);
/*------------------------------------------------------------------------
Hauptprogramm
------------------------------------------------------------------------*/
void main(void)
{    unsigned char channel;  // Gewählter AD-Messkanal
     AUXR=AUXR&0xFD;    // auf internes ERAM umschalten = EXTRAM löschen
     channel=0x00;           // Kanal AN0 gewählt
     ADCF=0x01;              // Pin P1.0 als A-D-Channel betreiben
     ADCON=0x20;             // AD Enable freigeben
     wait_ADC_ini();         // warten, bis ADC initialisiert
     P1=0xFF;                // P1 auf "Einlesen" schalten
     ADCON=ADCON&0xF8;       // Alte Kanaleinstellung (SCH0...SCH2) löschen
     ADCON=ADCON|channel;    // Auf Channel AN3 schalten

     Fire=1;                 // Zündausgang auf HIGH
      Zero=1;                // Zero-Crossing-Input auf HIGH
     INI_Timer1();           // Timer 1 initialisieren
```

```
     while(1)
     {    ADCON=ADCON|0x08;   // ADSST setzen, "Single Conversion" starten
          while((ADCON&0x10)!=0x10);       // Warten, bis ADEOC setzt
          ADCON=ADCON&0xEF;                // ADEOC wieder löschen
          z=ADDH;                          // Messergebnis abholen
          P0=z;              // Bei Bedarf: Kontrolle von z über LEDs an P0
          w=284-z;           // Startverzögerung und Richtungsumkehr
          while(Zero==1);    // Auf Nulldurchgang warten
          zuenden();         // Dann verzögerte Zündung auslösen
     }
}

/*---------------------------------------------------------------------------
Zusatzfunktionen
---------------------------------------------------------------------------*/
void INI_Timer1(void)
{    TMOD=0x20;            // Timer1 im 8 Bit-Reload-Betrieb (Mode 2)
     TL1=0;                // Start-Wert ist 0
     TH1=222;              // Reload-Wert ist 222 für 37 Mikrosekunden-Takt
     EA=1;                 // Alle Interrupts freigeben
     ET1=1;                // Timer1 - Interrupt freigeben
     TF1=0;                // Timer1-Überlaufflag vorsichtshalber löschen
     TR1=0;                // Timer1 steht still
}

void zuenden(void)
{
       unsigned char y;
       TR1=1;              // Timer 1 starten
       while(w!=0);        // Warten, bis Verzögerung vorbei
       TR1=0;              // Timer stoppen
       TL1=TH1;            // Timer-Neustart vorbereiten
       for(y=0;y<10;y++)
       {    Fire=0;        // Zündimpuls, Dauer ca. 50 Mikrosekunden
       }
       Fire=1;             // Zündung beendet
}

void wait_ADC_ini(void)      /* Wartezeit nach der ADC-Initialisierung */
{    unsigned char x;
     for(x=0;x<=5;x++);
}

void Timer_1_ISR(void)  interrupt 3      // Timer1 hat Interruptvektor 1B
{      TF1=0;                            // Überlaufflag löschen
       w--;                              // Wiederholungen zählen
}
```

17. Unterrichtsprojekt: DCF77-Funkuhr



Unter Verwendung eines für ca. 12 Euro käuflichen DCF77-Empfänger-Moduls der Firma Conrad erfolgt die Auswertung der übertragenen Zeit- und Datumsinformationen mit unserem ATMEL-ATM1- Board.

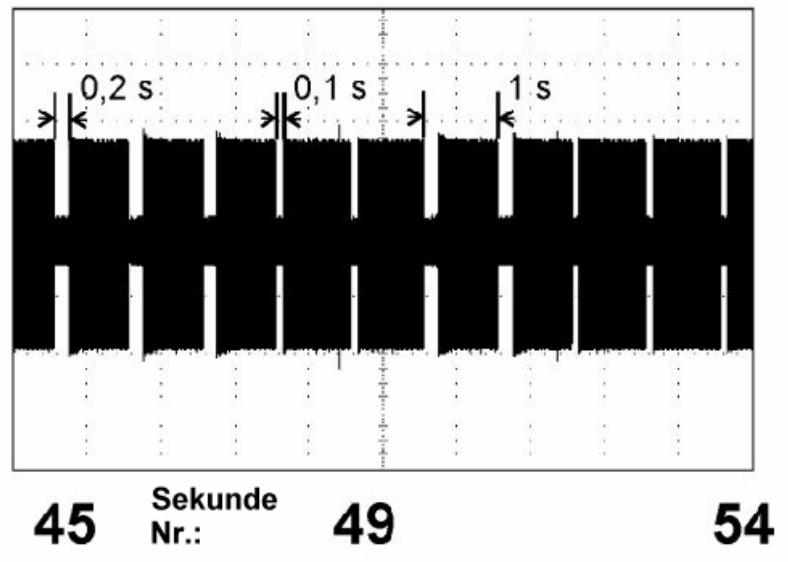
Der Empfänger wird dabei über ein Flachbankkabel und einen Pfostenfeldstecker mit Port P1 des Controllers verbunden und erhält so die an den Portsteckern zur Verfügung gestellte Systemspannung von +5V.

Der Ausgang (hier: „Ausgang – normal“) wird über einen Widerstand von 5,6 Kilo-Ohm mit der Betriebsspannung verbunden. Direkt an der „Ausgang-normal“-Klemme wird Portpin P1^0 angeschlossen und liest damit die Sekundenimpulse der Funkuhr ein.

17.1. „DCF77“ -- was steckt dahinter?

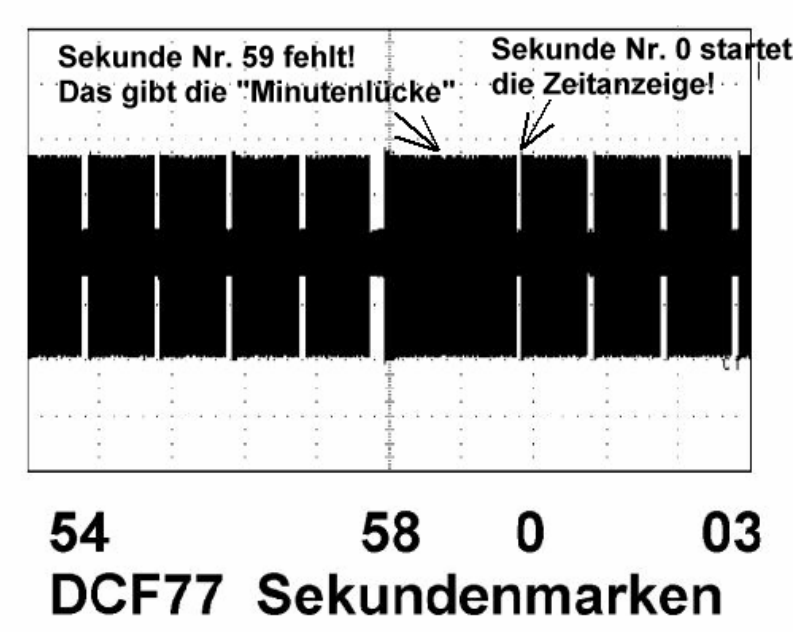
Die PTB (= Physikalisch – Technische Bundesanstalt) erzeugt über ein Cäsium-Normal eine sehr exakte Frequenz. Aus dieser Cäsiumfrequenz wird nicht nur eine „Normalfrequenz“ gewonnen, sondern auch eine äußerst präzise Zeitinformation generiert.

Basierend auf diesen beiden Informationen steuert man einen Langwellensender in Mainflingen (im Taunus, nahe Frankfurt) und versieht das über eine Antennenanlage abgestrahlte Signal zusätzlich mit einer Amplitudenmodulation, in der die Zeit und das Datum übermittelt werden (= ASK = Amplitude Shift Keying mit 1 Bit / Sekunde = 1 Baud).



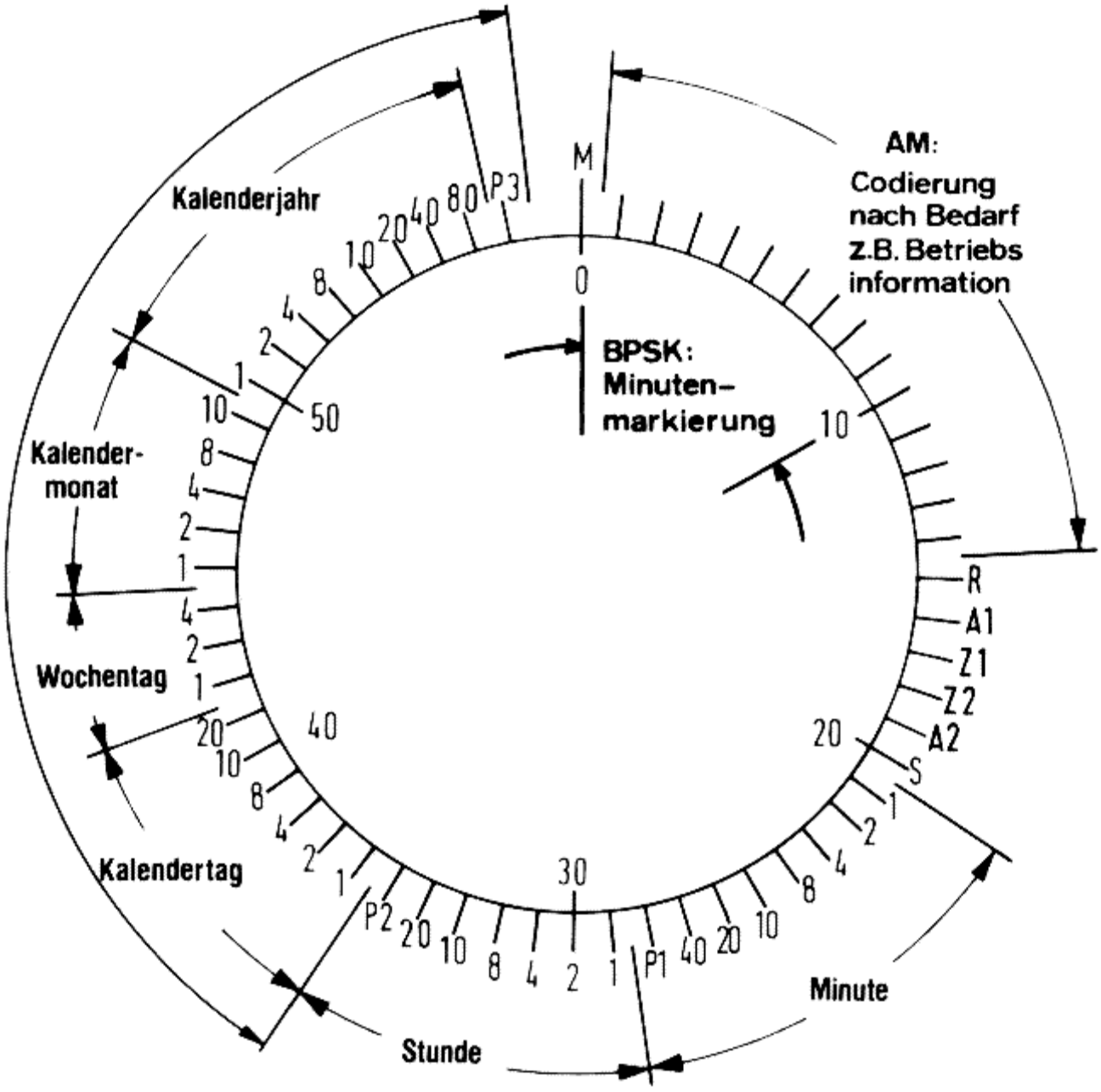
Die Sendefrequenz beträgt 77,5 kHz, die maximale Sendeleistung (PEP = Peak Envelope Power) 50 kW. Bei jeder vollen Sekunde wird die Trägerspannung auf 25% reduziert. Diese Reduktion dauert entweder 100ms (= log. Null) oder 200 ms (= log. Eins).

Im BCD-Code werden nun nacheinander (mit 1 Baud) die aktuelle Minute und Stunde, der aktuelle Wochentag und das komplette Datum übertragen. Alles mit einer Minute Vorlauf, denn erst zur nächsten vollen Minute erfolgt die gültige Anzeige. Dabei muss natürlich zuerst die nächste volle Minute über folgenden Trick erkannt werden:



Sekunde Nr. 59 wird NICHT markiert und diese „Minutenlücke“ ermöglicht den Empfängern die Synchronisation. (Diese 59. Sekunde muss also von der Decoderschaltung ergänzt werden!).
Bei der dann folgenden „Sekunde Nr. Null“ erfolgt die neue Zeitanzeige sowie die Ausgabe des Wochentages und des Datums.

Sendeschema für eine volle Minute:



Bei unserem Projekt wird nur die Zeit- und Datumsinformation ausgewertet. Das beginnt bei Sekunde 21 mit der „Einer-Stelle“ der im BCD-Code übermittelten Minutenangabe.

Auf die Übertragung der Minute bzw. Stunde folgt jeweils ein Prüfbit.

Bei der Wochentags-Information steht die „1“ für Montag.

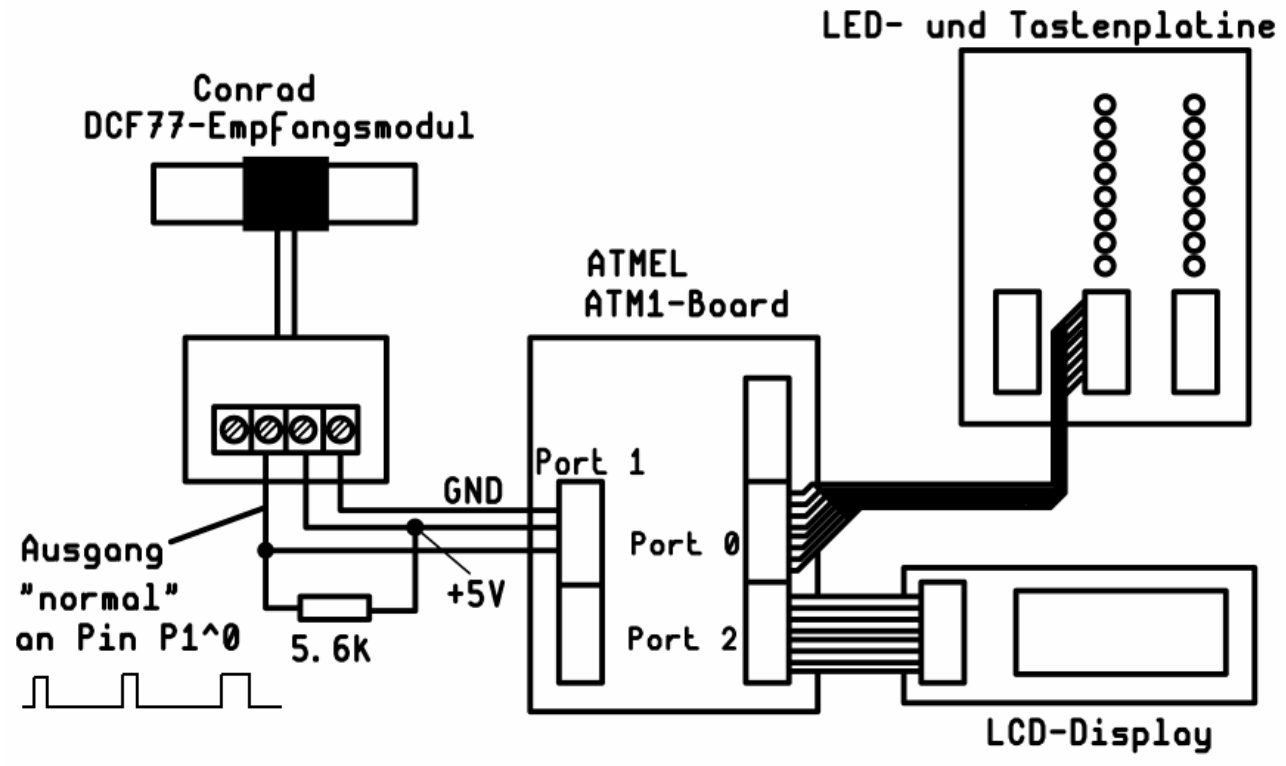
Beim Kalenderjahr werden nur die beiden letzten Stellen gesendet (z. B. „08“ für 2008).

Abgeschlossen wird diese „vorlaufende Übertragung“ durch das Prüfbit P3 und die Minutenlücke.

Hinweis: Sehr ausführliche Informationen finden sich im Internet, speziell in der Homepage der Physikalisch Technischen Bundesanstalt (PTB). Einfach in die Suchmaschine „DCF77“ eingeben!

17.2. Das Funkuhr-Projekt

17.2.1. Übersichtsschaltplan



17.2.2. Programmkonzept

Die Sekundenimpulse der Funkuhr werden über Portpin P1^0 in das Controllerboard eingelesen und direkt in Portpin P0^5 kopiert, an den eine Kontroll-LED angeschlossen ist. Zur Kontrolle werden die Sekundenimpulse, mit denen dann das Auswerteprogramm arbeitet, an einer weiteren LED (P0^7) ausgegeben -- so ist z. B. leicht zu überprüfen, ob die Ergänzung der (nicht übertragenen 59. Sekunde) funktioniert hat.

Das Auswerteprogramm gliedert sich in zwei Phasen, die ebenfalls durch LEDs (Phase 1 = LED an Pin P0^0 leuchtet, Phase 2 = LED an Pin P0^1 leuchtet) angezeigt werden:

In **Phase 1** sucht das Programm nach der Minutenlücke (= fehlender 59. Sekundenimpuls). Ist diese Lücke erkannt, beginnt der Sekundenzähler im Display zu arbeiten, die „Phase 1 – LED“ erlischt und die „Phase 2 – LED“ leuchtet auf.

In **Phase 2** werden nicht nur die ankommenden einzelnen Sekunden gezählt und sofort auf dem Display angezeigt. Zusätzlich werden ab Sekunde 21 **die „0/1“-Informationen** der im BCD-Code ausgestrahlten Minuten, Stunden, Wochentage, Kalendermonate usw. in passenden **bitadressierbaren Variablen** gespeichert. Wird die Minutenlücke erreicht, dann wird die fehlende Sekunde ergänzt und bei der darauf folgenden „Sekunde Null“ die komplette Zeit- und Datumsanzeige ausgelöst.

Zusätzlich sind im Programm noch einige Kontrollmechanismen eingebaut, um durch Störungen ausgelöste Fehlanzeigen (Beispiel: mehr als 60 Sekunden gezählt ….) zu erkennen. In solchen Fällen kehrt das Programm in Phase 1 zurück.

17.2.3. C-Programm „dcf77_atmel_03.c“

```
/*Decoderprogramm für eine DCF-77-Funkuhr unter Verwendung des ATMEL ATM1 -
Boards.

Das Funkuhr-Signal wird an Portpin P1^0 angelegt und das eingelesene Signal
an Portpin P0^5 zur Kontrolle mit einer LED angezeigt. An Pin P0^7 zeigt
eine weitere LED das für die Decodierung verwendete Signal mit beseitigter
Minutenlücke (zur Kontrolle der korrekten Ergänzung bei Sekunde 59).

Port P2 dient zur Ansteuerung des Displays.

An Port P0 hängt die LED-Kette. Die LEDs an den Pins P0^1 und P0^2
signalisieren die Startphase bzw. den Übergang in den Dauerbetrieb.

Das Funkuhrsignal ist auf HIGH-Pegel für 0,1 bzw. 0,2 Sekunden, für denRest
der Sekunde dagegen auf LOW.  Eine Dauer von 0,1 Sekunden stellt eine log.
Null, eine Dauer von 0,2 Sekunden dagegen eine log. Eins dar.

Es wird ein 11,059MHz-Quarz beim Board eingesetzt.

Programmiert durch G. Kraus am 22.04.2008

Programmname: dcf77_atmel_03.c*/

#include<stdio.h>
#include<t89c51ac2.h>       // Header für AT89C51AC3

void zeit_s(void);      // Display-Prototypen
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);

void detect_bit(void);      // Prototypen für verwendete Funktionen
void fill_Sec_59(void);
void Zeitanzeige(void);
void Datumsanzeige(void);
void Conv_Second(unsigned char a);
void del_10ms(unsigned char delay);
void ISR_Timer1(void);
void Conv_Date(void);
void Conv_Time(void);

sbit PULS_EING=P1^0;      // Funkuhrsignal (LOW-aktiv) an Portpin P1^0

sbit SEK_LED=P0^7;        // Sekunden-LED an P0.5
sbit PULS_LED=P0^5;       // Eingangssignal-Kontrolle
sbit LED_PH1=P0^0;        // LED an P0.0: Startphase
sbit LED_PH2=P0^1;        // LED an P0.1: Phase 2 (1. Minutenlücke erkannt)

bit  phase_1;             // Flag für Phase 1
bit  phase_2;             // Flag für Phase 2

data unsigned char TIME_TICK=0;  // Timer-Ticker (10ms - Intervalle)
data unsigned char SEK_TICK=0;   // Sekundenzähler
data unsigned char SEK_TICK_OLD; // Sekundenspeicher für Fehlersicherheit
```

```
bdata unsigned char MIN_EINS=0;       // Einer-Stelle der Minutenanzeige
bdata unsigned char MIN_ZEHN=0;       // Zehner-Stelle der Minutenanzeige

bdata unsigned char STD_EINS=0;       // Einer-Stelle der Stundenanzeige
bdata unsigned char STD_ZEHN=0;       // Zehner-Stelle der Stundenanzeige


bdata unsigned char TAG_EINS=0;       // Einer-Stelle des aktuellen Tages
bdata unsigned char TAG_ZEHN=0;       // Zehnerstelle des aktuellen Tages

bdata unsigned char WOCH_TAG=1;       // Wochentag (1 = Montag)

bdata unsigned char MON_EIN=0;        // Einer-Stelle des Monats
bdata unsigned char MON_ZEH=0;        // Zehner-Stelle des Monats

bdata unsigned char JAHR_EIN=0;       // Einer-Stelle des Jahres
bdata unsigned char JAHR_ZEH=0;       // Zehner-Stelle des Jahres

code unsigned char wochent[21]=      {"MoDiMiDoFrSaSo      "};
code unsigned char start_time[21]=   {"Time  xx:xx:xx      "};
code unsigned char start_date[21]=   {"Mo    xx.xx.20xx    "};
unsigned char time_array[21]=        {"Time  xx:xx:xx      "};
unsigned char date_array[21]=        {"Mo    xx.xx.20xx    "};
/*------------------------------------------------------------*/

void main(void)
{  P0=0x00;              // Alle LEDs aus
   P1=0xFF;              // P1 auf "Lesen" schalten
   TMOD=0x10;            // Timer1 im 16 Bit-Betrieb (Mode 1)
   TL1=0xFF;             // Reload-Wert für 11,059MHz-Quarz
   TH1=0xDB;             // ist 0xDBFF für 10ms
   EA=1;                 // Alle Interrupts freigeben
   ET1=1;                // Timer1 - Interrupt freigeben
   TF1=0;                // Timer1-Überlaufflag löschen
   TR1=0;                // Timer1 ausschalten

   lcd_ein();              // LCD-Einschaltroutine
   lcd_ini1();             // Einstellung der LCD-Betriebswerte
   switch_z1();            // Schalte auf Zeile 1
   show_text(start_time);  // Zeige Starttext der Zeitanzeige
   switch_z2();            // Schalte auf Zeile 2
   show_text(start_date);  // Zeige Starttext der Datumsanzeige
   LED_PH1=1;              // Phase 1 beginnt
   phase_1=1;              // " "  "
   LED_PH2=0;              // Phase 2 OFF
   phase_2=0;
   AUXR=AUXR&0xFD;         // Auf internes ERAM umschalten

   while(1)
   {  while(phase_1)            // Minutenlücke erkennen
      {  switch_z1();           // Starttext nach Rückkehr
         show_text(start_time); // aus der Fehlererkennung zeigen
         switch_z2();
         show_text(start_date);

         while(PULS_EING==1)    // Auf LOW-Pegel warten
         {  SEK_LED=1;          // Sekunden-LED ON
            PULS_LED=PULS_EING; // LED zeigt Eingangssignal
         }

         TIME_TICK=0;           // TIME_TICK soll 10ms-Einheiten zählen
```

```
  TR1=1;                  // Timer starten, um LOW-Dauer zu messen

  while(PULS_EING==0)     // LOW-Ende abwarten
  {   SEK_LED=0;          // Sekunden-LED ausschalten
      PULS_LED=PULS_EING; // LED = Eingangsssignal
  }

  SEK_LED=1;              // LOW ist vorbei, HIGH erreicht
  TR1=0;                  // Timer stoppen

  if(TIME_TICK>105)       // wenn 1,05 Sekunden vergangen:
  { phase_1=0;            // Phase_1 - Bit ausschalten
    phase_2=1;            // Phase_2 - Bit einschalten
    LED_PH2=1;            // zugehörige LEDs schalten
    LED_PH1=0;
    SEK_TICK=0;               // Sekunde "NULL" zählen
    Conv_Second(SEK_TICK);    // in ASCII konvertieren
    Zeitanzeige();            // und anzeigen

    while(PULS_EING==1)       // HIGH-Ende abwarten
    { SEK_LED=1;              // Sekunden-LED ausschalten
      PULS_LED=PULS_EING;     //LED = Eingangssignal
    }
  }
}

while(phase_2)                // Dauerbetrieb
{    while (SEK_TICK<=57)     // Sekunde 58 noch nicht erreicht
     { SEK_TICK_OLD=SEK_TICK; // Sekundenstand speichern
       while(PULS_EING==0)    // Eingang auf LOW, deshalb
       { SEK_LED=0;           // LOW-Ende abwarten
         PULS_LED=PULS_EING;  // LED = Eingangssignal
       }

       SEK_TICK++;            // HIGH kommt, deshalb Sekunde zählen

       PULS_LED=PULS_EING;    // LED = Eingangssignal
       TR1=1;                 // Timer starten
       SEK_LED=1;             // Sekunden-LED einschalten

       Conv_Second(SEK_TICK);    // in ASCII umcodieren
       Zeitanzeige();            // auf dem Display anzeigen
       while(PULS_EING==1)       // Eingang noch auf HIGH, deshalb
       {   SEK_LED=1;            // auf nächstes LOW warten
           PULS_LED=PULS_EING;   // LED = Eingangssignal
       }
       TR1=0;                    // LOw da = Timer stoppen
       detect_bit();             // Bit auswerten
       TIME_TICK=0;              // 10ms-Zählerstand wieder auf Null

       SEK_LED=0;                // Sekunden-LED ausschalten
       PULS_LED=PULS_EING;       // LED = Eingangssignal
                if(SEK_TICK - SEK_TICK_OLD!=1)    // Kontrolle: nur 1
                                                  // Sekunde
                {   phase_2=0;                   // gezählt? Wenn
                                                 // nicht, dann raus
         phase_1=1;                              // aus Phase 2!
         break;
       }
     }
```

```
          fill_Sec_59();              // Ergänze die 59. Sekunde und
        }                             // zeige Zeit und Datum an
   }
}
//---------------------------------------------------------------

void ISR_Timer1(void) interrupt 3       // Timer1 hat Interruptvektor 1B
{
    TF1=0;                               // Überlaufflag löschen
    TIME_TICK++;               // Inkrementiere Timer-Ticker im 10ms-Takt

  TH1=0xDC;                    // Reloadwert ist 0xDBFF für 11,059MHz-Quarz
    TL1=TL1-1;                 // und 10ms bis zum Überlauf
}
//---------------------------------------------------------------------

void del_10ms(unsigned char delay)  // Verzögerung in 10ms-Einheiten
{
  TIME_TICK=0;                      // Einheitenzähler auf Null
  TR1=1;                            // Zähler starten
  while(TIME_TICK<delay);           // Verzögerung ablaufen lassen
  TR1=0;                            // Zähler stoppen
  TIME_TICK=0;                      // Einheitenzähler wieder auf Null
  TL1=0xFF;                         // Reload-Wert für 11,059MHz-Quarz
  TH1=0xDB;                         // ist 0xDBFF für 10ms bis zum überlauf
}
//---------------------------------------------------------------------

void fill_Sec_59(void)          // Ergänzung der 59. Sekunde und Anzeige
{    del_10ms(95);              // 950ms Verzögerung
        if((SEK_TICK==58)&&(PULS_EING==0))     // Ist das wirklich die
                                               // Minutenlücke?
        {   SEK_LED=1;          // Wenn ja: Sekunden-LED an und 59. Sekunde
                                // zählen
      SEK_TICK=59;
      Conv_Second(SEK_TICK);    // Sekunden in ASCII konvertieren und
      Zeitanzeige();            // Zeit bei 59. Sekunde anzeigen
      del_10ms(20);             // 200ms Pulslänge bei der ergänzten
      SEK_LED=0;                // 59. Sekunde, dann wieder LOW
    }
    else               // Wenn ein Fehler vorliegt: zurück zur Startphase
    { phase_2=0;
      phase_1=1;
    }


    while(PULS_EING==0)         // Warte auf das Ende der Minutenlücke
    { SEK_LED=0;                // Sekunden-LED ist OFF
      PULS_LED=PULS_EING;       // Kontroll-LED = Eingangssignal
    }

    SEK_LED=1;                  // Sekunde Null erreicht
    PULS_LED=PULS_EING;         // Kontroll_LED = Eingangssignal
    SEK_TICK=0;                 // Sekundenzähler auf Null
    Conv_Second(SEK_TICK);      // Sekunde in ASCII konvertieren
    Conv_Time();                // Komplette Zeit ermitteln
    Conv_Date();                // Komplettes Datum ermitteln
    Zeitanzeige();              // Zeit auf Display anzeigen
    Datumsanzeige();            // Datum auf Display anzeigne
    while(PULS_EING==1)         // Ende von HIGH bei Sekunde Null
                                // abwarten
```

```
     {  SEK_LED=1;              // Sekunden-LED ist ON
        PULS_LED=PULS_EING;     // Kontroll-LED = Eingangssignal
     }
     SEK_LED=0;                 // Sekunden-LED ist OFF
     PULS_LED=PULS_EING;        // Kontroll-LED = Eingangssignal
}
//-----------------------------------------------------

void detect_bit(void)        // Bitauswertung bei jeder Sekunde
{    unsigned char ZER_ONE;  // Variable speichert erkannte Null oder
                   // Eins

     if(TIME_TICK<=10)     // Pulslänge unter 100ms ergibt Null....
     {  ZER_ONE=0;
     }
     else             // ...und mehr als 100ms ergeben Eins
     {  ZER_ONE=1;
     }

     if(SEK_TICK==21)                  // Bit von Sekunde 21 gibt
                        // Minuten-Einer
     {  if(ZER_ONE==1)                 // Falls "1" gesendet:
        {  MIN_EINS=MIN_EINS|0x01;   // Einser-Stelle setzen
        }
        else                           // sonst: löschen
        {  MIN_EINS=MIN_EINS&0x0e;
        }
         }

         if(SEK_TICK==22)                      // Bit von Sekunde 22 gibt
                           // Minuten-Zweier
     {  if(ZER_ONE==1)                 // Falls "1" gesendet:
        {  MIN_EINS=MIN_EINS|0x02;   // Zweier-Stelle setzen
        }
        else                           // sonst: löschen
        {  MIN_EINS=MIN_EINS&0x0d;
        }
         }

         if(SEK_TICK==23)                    // Bit von Sekunde 23 gibt
                                                // Minuten-Vierer
     {  if(ZER_ONE==1)                 // Falls "1" gesendet:
        {  MIN_EINS=MIN_EINS|0x04;   // Vierer-Stelle setzen
        }
        else                           // sonst löschen:
        {  MIN_EINS=MIN_EINS&0x0b;
        }
         }

         if(SEK_TICK==24)                    // Bit von Sekunde 24 gibt
                           // Minuten-Achter
     {  if(ZER_ONE==1)                 // Falls "1" gesendet:
        {  MIN_EINS=MIN_EINS|0x08;   // Achter-Stelle setzen
        }
        else                           //sonst: löschen
        {  MIN_EINS=MIN_EINS&0x07;
        }
              MIN_EINS=(MIN_EINS&0x0f) + 48;  // Ergebnis in ASCII
                                 // konvertieren
         }
```

```
if(SEK_TICK==25)         // Bit von Sekunde 25 gibt Minuten-Zehner
{  if(ZER_ONE==1)                  // Falls "1" gesendet:
   {  MIN_ZEHN=MIN_ZEHN|0x01;    // Zehner-Stelle setzen
   }
   else                            // sonst: löschen
   {  MIN_ZEHN=MIN_ZEHN&0x0e;
   }
    }

    if(SEK_TICK==26)       // Bit von Sekunde 26 gibt Minuten-Zwanziger
{  if(ZER_ONE==1)                  // Falls "1" gesendet:
   {  MIN_ZEHN=MIN_ZEHN|0x02;    // Zwanziger-Stelle setzen
   }
   else                            // sonst: löschen
   {  MIN_ZEHN=MIN_ZEHN&0x0d;
   }
    }

    if(SEK_TICK==27)       // Bit von Sekunde 27 gibt Minuten-Vierziger
{  if(ZER_ONE==1)                  // Falls "1" gesendet:
   {  MIN_ZEHN=MIN_ZEHN|0x04;    // Vierziger-Stelle setzen
   }
   else                            // sonst: löschen
   {  MIN_ZEHN=MIN_ZEHN&0x0b;
   }
        MIN_ZEHN=(MIN_ZEHN&0x07) + 48;  // Ergebnis in ASCII
                                        // konvertieren
    }

if(SEK_TICK==29)          // Bit von Sekunde 29 gibt Stunden-Einer
{  if(ZER_ONE==1)                  // Falls "1" gesendet:
   {  STD_EINS=STD_EINS|0x01;    // Einer-Stelle setzen
   }
   else                            // sonst: löschen
   {  STD_EINS=STD_EINS&0x0e;
   }
    }

    if(SEK_TICK==30)         // Bit von Sekunde 30 gibt Stunden-Zweier
{  if(ZER_ONE==1)                  // Falls "1" gesendet:
   {  STD_EINS=STD_EINS|0x02;    // Zweier-Stelle setzen
   }
   else                            // sonst: löschen
   {  STD_EINS=STD_EINS&0x0d;
   }
    }

    if(SEK_TICK==31)         // Bit von Sekunde 31 gibt Stunden-Vierer
{  if(ZER_ONE==1)                  // Falls "1" gesendet:
   {  STD_EINS=STD_EINS|0x04;    // Vierer-Stelle setzen
   }
   else                            // sonst: löschen
   {  STD_EINS=STD_EINS&0x0b;
   }
    }
```

```
    if(SEK_TICK==32)        // Bit von Sekunde 32 gibt Stunden-Achter
{  if(ZER_ONE==1)                 // Falls "1" gesendet:
   {  STD_EINS=STD_EINS|0x08;    // Achter-Stelle setzen
   }
   else                           // sonst: löschen
   {  STD_EINS=STD_EINS&0xf7;
   }
    STD_EINS=(STD_EINS&0x0f) + 48;  // Ergebnis in ASCII konvertieren
    }

    if(SEK_TICK==33)         // Bit von Sekunde 33 gibt Stunden-Zehner
{  if(ZER_ONE==1)                 // Falls "1" gesendet:
   {  STD_ZEHN=STD_ZEHN|0x01;    // Zehner-Stelle setzen
   }
   else                           // sonst: löschen
   {  STD_ZEHN=STD_ZEHN&0x0e;
   }
    }

    if(SEK_TICK==34)       // Bit von Sekunde 34 gibt Stunden-Zwanziger
{  if(ZER_ONE==1)                 // Falls "1" gesendet:
   {  STD_ZEHN=STD_ZEHN|0x02;    // Zwanziger-Stelle setzen
   }
   else                           // sonst: löschen
   {  STD_ZEHN=STD_ZEHN&0x0d;
   }
   STD_ZEHN=(STD_ZEHN&0x03) + 48;   // Ergebnis nach ASCII
                     // konvertieren
    }

if(SEK_TICK==36)            // Bit von Sekunde 36 gibt Tag-Einer
{  if(ZER_ONE==1)                 // Falls "1" gesendet:
   {  TAG_EINS=TAG_EINS|0x01;    // Einer-Stelle setzen
   }
   else                           // sonst: löschen
   {  TAG_EINS=TAG_EINS&0x0e;
   }
    }

    if(SEK_TICK==37)            // Bit von Sekunde 37 gibt Tag-Zweier
   {  if(ZER_ONE==1)            // Falls "1" gesendet:
   {  TAG_EINS=TAG_EINS|0x02;    // Zweier-Stelle setzen
   }
   else                           // sonst: löschen
   {  TAG_EINS=TAG_EINS&0x0d;
   }
    }

    if(SEK_TICK==38)            // Bit von Sekunde 38 gibt Tag-Vierer
{  if(ZER_ONE==1)                // Falls "1" gesendet:
   {  TAG_EINS=TAG_EINS|0x04;   // Vierer-Stelle setzen
   }
   else                          // sonst: löschen
   {  TAG_EINS=TAG_EINS&0x0b;
   }
    }
```

```
    if(SEK_TICK==39)             // Bit von Sekunde 39 gibt Tag-Achter
{  if(ZER_ONE==1)                // Falls "1" gesendet:
   {  TAG_EINS=TAG_EINS|0x08;   // Achter-Stelle setzen
   }
   else                          // sonst: löschen
   {  TAG_EINS=TAG_EINS&0x07;
   }

   TAG_EINS=(TAG_EINS&0x0f) + 48;   // Ergebnis in ASCII
                     // konvertieren
    }

if(SEK_TICK==40)            // Bit von Sekunde 40 gibt Tag-Zehner
{  if(ZER_ONE==1)                // Falls "1" gesendet:
   {  TAG_ZEHN=TAG_ZEHN|0x01;   // Zehner-Stelle setzen
   }
   else                          // sonst: löschen
   {  TAG_ZEHN=TAG_ZEHN&0x0e;
   }
    }

    if(SEK_TICK==41)          // Bit von Sekunde 41 gibt Tag-Zwanziger
{  if(ZER_ONE==1)                // Falls "1" gesendet:
   {  TAG_ZEHN=TAG_ZEHN|0x02;   // Zwanziger-Stelle setzen
   }
   else                          // sonst: löschen
   {  TAG_ZEHN=TAG_ZEHN&0x0d;
   }
   TAG_ZEHN=(TAG_ZEHN&0x03) + 48;   // Ergebnis in ASCII
                     // konvertieren
    }

    if(SEK_TICK==42)       // Bit von Sekunde 42 gibt Wochentag-Einer
{  if(ZER_ONE==1)                // Falls "1" gesendet:
   {  WOCH_TAG=WOCH_TAG|0x01;   // Einer-Stelle setzen
   }
   else                          // sonst: löschen
   {  WOCH_TAG=WOCH_TAG&0x0e;
   }
    }

    if(SEK_TICK==43)       // Bit von Sekunde 43 gibt Wochentag-Zweier
{  if(ZER_ONE==1)                // Falls "1" gesendet:
   {  WOCH_TAG=WOCH_TAG|0x02;   // Zweier-Stelle setzen
   }
   else                          // sonst: löschen
   {  WOCH_TAG=WOCH_TAG&0x0d;
   }
    }

    if(SEK_TICK==44)      // Bit von Sekunde 44 gibt Wochentag-Vierer
{  if(ZER_ONE==1)                // Falls "1" gesendet:
   {  WOCH_TAG=WOCH_TAG|0x04;   // Vierer-Stelle setzen
   }
   else                          // sonst: löschen
   {  WOCH_TAG=WOCH_TAG&0x0b;
   }
    WOCH_TAG=WOCH_TAG&0x07;        // Ergebnis in ASCII konvertieren
    }
```

```
if(SEK_TICK==45)           // Bit von Sekunde 45 gibt Monat-Einer
{  if(ZER_ONE==1)               // Falls "1" gesendet:
   {  MON_EIN=MON_EIN|0x01;     // Einer-Stelle setzen
   }
   else                          // sonst: löschen
   {  MON_EIN=MON_EIN&0x0e;
   }
    }

    if(SEK_TICK==46)           // Bit von Sekunde 46 gibt Monat-Zweier
{  if(ZER_ONE==1)               // Falls "1" gesendet:
   {  MON_EIN=MON_EIN|0x02;    // Zweier-Stelle setzen
   }
   else                          // sonst: löschen
   {  MON_EIN=MON_EIN&0x0d;
   }
    }

    if(SEK_TICK==47)           // Bit von Sekunde 47 gibt Monat-Vierer
{  if(ZER_ONE==1)               // Falls "1" gesendet:
   {  MON_EIN=MON_EIN|0x04;    // Vierer-Stelle setzen
   }
   else                          // sonst: löschen
   {  MON_EIN=MON_EIN&0x0b;
   }
    }

    if(SEK_TICK==48)          // Bit von Sekunde 48 gibt Monat-Achter
{  if(ZER_ONE==1)               // Falls "1" gesendet:
   {  MON_EIN=MON_EIN|0x08;    // Achter-Stelle setzen
   }
   else                          // sonst: löschen
   {  MON_EIN=MON_EIN&0x07;
   }
    MON_EIN=(MON_EIN&0x0f) + 48; // Ergebnis in ASCII konvertieren

    }

if(SEK_TICK==49)          Bit von Sekunde 49 gibt Monat-Zehner
{  if(ZER_ONE==1)               // Falls "1" gesendet:
   {  MON_ZEH=MON_ZEH|0x01;    // Zehner-Stelle setzen
   }
   else                          // sonst: löschen
   {  MON_ZEH=MON_ZEH&0x0e;
   }
    MON_ZEH=(MON_ZEH&0x01) + 48; // Ergebnis in ASCII konvertieren
    }

if(SEK_TICK==50)          // Bit von Sekunde 50 gibt Jahr-Einer
{  if(ZER_ONE==1)               // Falls "1" gesendet:
   {  JAHR_EIN=JAHR_EIN|0x01;  // Einer-Stelle setzen
   }
   else                          //  sonst: löschen
   {  JAHR_EIN=JAHR_EIN&0x0e;
   }
    }
```

```
    if(SEK_TICK==51)          // Bit von Sekunde 51 gibt Jahr-Zweier
{  if(ZER_ONE==1)               // Falls "1" gesendet:
   {  JAHR_EIN=JAHR_EIN|0x02;  // Zweier-Stelle setzen
   }
   else                         // sonst: löschen
   {  JAHR_EIN=JAHR_EIN&0x0d;
   }
    }

    if(SEK_TICK==52)         // Bit von Sekunde 52 gibt Jahr-Vierer
{  if(ZER_ONE==1)               // Falls "1" gesendet:
   {  JAHR_EIN=JAHR_EIN|0x04;  // Vierer-Stelle setzen
   }
   else                         // sonst: löschen
   {  JAHR_EIN=JAHR_EIN&0x0b;
   }
    }

    if(SEK_TICK==53)         // Bit von Sekunde 53 gibt Jahr-Achter
{  if(ZER_ONE==1)               // Falls "1" gesendet:
   {  JAHR_EIN=JAHR_EIN|0x08;  // Achter-Stelle setzen
   }
   else                         // sonst: löschen
   {  JAHR_EIN=JAHR_EIN&0x07;
   }
    JAHR_EIN=(JAHR_EIN&0x0f) + 48;  // Ergebnis in ASCII konvertieren
    }

if(SEK_TICK==54)             // Bit von Sekunde 54 gibt Jahr-Zehner
{  if(ZER_ONE==1)                // Falls "1" gesendet:
   {  JAHR_ZEH=JAHR_ZEH|0x01;    // Zehner-Stelle setzen
   }
   else                          //sonst: löschen
   {  JAHR_ZEH=JAHR_ZEH&0x0e;
   }
    }

    if(SEK_TICK==55)          // Bit von Sekunde 55 gibt Jahr-Zwanziger
{  if(ZER_ONE==1)                // Falls "1" gesendet:
   {  JAHR_ZEH=JAHR_ZEH|0x02;    // Zwanziger-Stelle setzen
   }
   else                          // sonst: löschen
   {  JAHR_ZEH=JAHR_ZEH&0x0d;
   }
    }

    if(SEK_TICK==56)          // Bit von Sekunde 56 gibt Jahr-Vierziger
{  if(ZER_ONE==1)                // Falls "1" gesendet:
   {  JAHR_ZEH=JAHR_ZEH|0x04;    // Vierziger-Stelle setzen
   }
   else                          // sonst: löschen
   {  JAHR_ZEH=JAHR_ZEH&0x0b;
   }
    }
```

```
        if(SEK_TICK==57)        // Bit von Sekunde 57 gibt Jahr-Achtziger
     {  if(ZER_ONE==1)               // Falls "1" gesendet:
        {  JAHR_ZEH=JAHR_ZEH|0x08;    // Achtziger-Stelle setzen
        }
        else                         // sonst: löschen
        {  JAHR_ZEH=JAHR_ZEH&0x07;
        }
         JAHR_ZEH=(JAHR_ZEH&0x0f) + 48; // Ergebnis nach ASCII konvertieren
         }
}
//-------------------------------------------------------

void Conv_Second(unsigned char a) // Konvertiert Sekunden nach ASCII
{  unsigned char x;
   x=a;                  // Kopie anfertigen
   time_array[13]=x%10 +0x30;   // Sekunden-Einer konvertieren
   x=x/10;
     time_array[12]=x%10 +0x30;     // Sekunden-Zehner konvertieren
}
//-------------------------------------------------------

void Conv_Date(void)    // Legt das Datum-Array für das Display an
{  date_array[15]=JAHR_EIN;       // Jahres-Einer
   date_array[14]=JAHR_ZEH;       // Jahres-Zehner
   date_array[10]=MON_EIN;        // Monats-Einer
   date_array[9]=MON_ZEH;         // Monats-Zehner
   date_array[7]=TAG_EINS;        // Tages-Einer
   date_array[6]=TAG_ZEHN;        // Tages-Zehner
   date_array[0]=wochent[(WOCH_TAG-1)*2];     // Wochentage von Mo bis Fr
   date_array[1]=wochent[(WOCH_TAG-1)*2+1];
}
//-------------------------------------------------------

void Conv_Time(void)              // Legt das Zeit-Array für das Display an
{  time_array[10]=MIN_EINS;       // Minuten-Einer
   time_array[9]=MIN_ZEHN;        // Minuten-Zehner
   time_array[7]=STD_EINS;        // Stunden-Einer
   time_array[6]=STD_ZEHN;        // Stunden-Zehner
}
//-------------------------------------------------------

void Zeitanzeige(void)       // zeigt die Zeit auf dem Display an
{    switch_z1();
     show_text(time_array);
}
//-------------------------------------------------------

void Datumsanzeige(void)       // zeigt das Datum auf dem Display an
{    switch_z2();
     show_text(date_array);
}
//-------------------------------------------------------
```

Anhang 1: Neuer, selbstgeschriebener Header für ATMEL AT89C51AC3

```
/*****************************************************************************
* NAME: AT89C51AC3.h   =  Header for ATMEL AT89C51AC3
*----------------------------------------------------------------------------
* PURPOSE: include file for KEIL
*
* Modification of <t89c51ac2.h> by Gunthard Kraus, Elektronikschule Tettnang
*****************************************************************************/
#ifndef _AT89C51AC3_H_

#define _AT89C51AC3_H_

#define Sfr(x, y)        sfr x = y
#define Sbit(x, y, z)   sbit x = y^z
#define Sfr16(x,y)      sfr16 x = y

/*----------------------------------------*/
/* Include file for 8051 SFR Definitions  */
/*----------------------------------------*/

/*  BYTE Register  */
Sfr (P0 , 0x80);
Sfr (P1 , 0x90);

Sbit (P1_7, 0x90, 7);
Sbit (P1_6, 0x90, 6);
Sbit (P1_5, 0x90, 5);
Sbit (P1_4, 0x90, 4);
Sbit (P1_3, 0x90, 3);
Sbit (P1_2, 0x90, 2);
Sbit (P1_1, 0x90, 1);
Sbit (P1_0, 0x90, 0);

Sfr (P2 , 0xA0);

Sbit (P2_7 , 0xA0, 7);
Sbit (P2_6 , 0xA0, 6);
Sbit (P2_5 , 0xA0, 5);
Sbit (P2_4 , 0xA0, 4);
Sbit (P2_3 , 0xA0, 3);
Sbit (P2_2 , 0xA0, 2);
Sbit (P2_1 , 0xA0, 1);
Sbit (P2_0 , 0xA0, 0);


Sfr (P3 , 0xB0);

Sbit (P3_7 , 0xB0, 7);
Sbit (P3_6 , 0xB0, 6);
Sbit (P3_5 , 0xB0, 5);
Sbit (P3_4 , 0xB0, 4);
Sbit (P3_3 , 0xB0, 3);
Sbit (P3_2 , 0xB0, 2);
Sbit (P3_1 , 0xB0, 1);
Sbit (P3_0 , 0xB0, 0);

Sbit (RD , 0xB0, 7);
Sbit (WR , 0xB0, 6);
Sbit (T1 , 0xB0, 5);
Sbit (T0 , 0xB0, 4);
Sbit (INT1, 0xB0, 3);
Sbit (INT0, 0xB0, 2);
Sbit (TXD , 0xB0, 1);
Sbit (RXD , 0xB0, 0);

Sfr (P4 , 0xC0);
```

```
Sfr (PSW , 0xD0);

Sbit (CY  , 0xD0, 7);
Sbit (AC  , 0xD0, 6);
Sbit (F0  , 0xD0, 5);
Sbit (RS1 , 0xD0, 4);
Sbit (RS0 , 0xD0, 3);
Sbit (OV  , 0xD0, 2);
Sbit (UD  , 0xD0, 1);
Sbit (P   , 0xD0, 0);

Sfr (ACC , 0xE0);
Sfr (B , 0xF0);
Sfr (SP , 0x81);
Sfr (DPL , 0x82);
Sfr (DPH , 0x83);

Sfr (PCON , 0x87);
Sfr (CKCON0 , 0x8F);
Sfr (CKCON1 , 0x9F);

/*------------------ TIMER registers ----------------------*/
Sfr (TCON , 0x88);
Sbit (TF1 , 0x88, 7);
Sbit (TR1 , 0x88, 6);
Sbit (TF0 , 0x88, 5);
Sbit (TR0 , 0x88, 4);
Sbit (IE1 , 0x88, 3);
Sbit (IT1 , 0x88, 2);
Sbit (IE0 , 0x88, 1);
Sbit (IT0 , 0x88, 0);

Sfr (TMOD , 0x89);

Sfr  (T2CON , 0xC8);
Sbit (TF2   , 0xC8, 7);
Sbit (EXF2  , 0xC8, 6);
Sbit (RCLK  , 0xC8, 5);
Sbit (TCLK  , 0xC8, 4);
Sbit (EXEN2 , 0xC8, 3);
Sbit (TR2   , 0xC8, 2);
Sbit (C_T2  , 0xC8, 1);
Sbit (CP_RL2, 0xC8, 0);

Sfr (T2MOD , 0xC9);
Sfr (TL0 , 0x8A);
Sfr (TL1 , 0x8B);
Sfr (TL2 , 0xCC);
Sfr (TH0 , 0x8C);
Sfr (TH1 , 0x8D);
Sfr (TH2 , 0xCD);
Sfr (RCAP2L , 0xCA);
Sfr (RCAP2H , 0xCB);
Sfr (WDTRST , 0xA6);
Sfr (WDTPRG , 0xA7);


/*------------------- UART registers ------------------------*/
Sfr (SCON , 0x98);
Sbit (SM0  , 0x98, 7);
Sbit (FE   , 0x98, 7);
Sbit (SM1  , 0x98, 6);
Sbit (SM2  , 0x98, 5);
Sbit (REN  , 0x98, 4);
Sbit (TB8  , 0x98, 3);
Sbit (RB8  , 0x98, 2);
Sbit (TI   , 0x98, 1);
Sbit (RI   , 0x98, 0);
```

```
Sfr (SBUF , 0x99);
Sfr (SADEN , 0xB9);
Sfr (SADDR , 0xA9);


/*----------------------Serial Port Interface SPI------------*/
Sfr (SPCON , 0xD4);
Sfr (SPSCR , 0xD5);
Sfr (SPDAT , 0xD6);


/*--------------------- ADC registers ----------------------*/
Sfr (ADCLK , 0xF2);
Sfr (ADCON , 0xF3);
#define MSK_ADCON_PSIDLE 0x40
#define MSK_ADCON_ADEN   0x20
#define MSK_ADCON_ADEOC  0x10
#define MSK_ADCON_ADSST  0x08
#define MSK_ADCON_SCH    0x07
Sfr (ADDL , 0xF4);
#define MSK_ADDL_UTILS  0x03
Sfr (ADDH , 0xF5);
Sfr (ADCF , 0xF6);

/*--------------------- FLASH EEPROM registers ------------*/
Sfr (FCON  , 0xD1);
#define MSK_FCON_FBUSY 0x01
#define MSK_FCON_FMOD  0x06
#define MSK_FCON_FPS   0x08
#define MSK_FCON_FPL   0xF0
Sfr (EECON  , 0xD2);
#define MSK_EECON_EEBUSY 0x01
#define MSK_EECON_EEE    0x02
#define MSK_EECON_EEPL   0xF0

Sfr (AUXR  , 0x8E);
Sfr (AUXR1 , 0xA2);
#define MSK_AUXR_M0      0x20

#define MSK_AUXR1_ENBOOT 0x20

Sfr (FSTA , 0xD3);


/*--------------------- Interrupt registers ----------------------*/
Sfr (IPL1 , 0xF8);
Sfr (IPH1 , 0xF7);
Sfr (IEN0 , 0xA8);
Sfr (IPL0 , 0xB8);
Sfr (IPH0 , 0xB7);
Sfr (IEN1 , 0xE8);


/*  IEN0  */
Sbit (EA   , 0xA8, 7);
Sbit (EC   , 0xA8, 6);
Sbit (ET2  , 0xA8, 5);
Sbit (ES   , 0xA8, 4);
Sbit (ET1  , 0xA8, 3);
Sbit (EX1  , 0xA8, 2);
Sbit (ET0  , 0xA8, 1);
Sbit (EX0  , 0xA8, 0);


/*  IEN1  */
Sbit (ESPI , 0xE8, 3);
Sbit (EADC , 0xE8, 1);
```

```
/*  IPL0  */
Sbit (PPC  , 0xB8, 6);
Sbit (PT2  , 0xB8, 5);
Sbit (PS   , 0xB8, 4);
Sbit (PT1  , 0xB8, 3);
Sbit (PX1  , 0xB8, 2);
Sbit (PT0  , 0xB8, 1);
Sbit (PX0  , 0xB8, 0);


/*  IPL1  */
Sbit (SPIL   , 0xF8, 3);
Sbit (PADCL  , 0xF8, 1);



/*---------------------- PCA registers ----------------------*/
Sfr (CCON , 0xD8);
Sbit(CF  , 0xD8, 7);
Sbit(CR  , 0xD8, 6);
Sbit(CCF4, 0xD8, 4);
Sbit(CCF3, 0xD8, 3);
Sbit(CCF2, 0xD8, 2);
Sbit(CCF1, 0xD8, 1);
Sbit(CCF0, 0xD8, 0);

Sfr (CMOD , 0xD9);
Sfr (CH , 0xF9);
Sfr (CL , 0xE9);
Sfr (CCAP0H  , 0xFA);
Sfr (CCAP0L  , 0xEA);
Sfr (CCAPM0  , 0xDA);
Sfr (CCAP1H  , 0xFB);
Sfr (CCAP1L  , 0xEB);
Sfr (CCAPM1  , 0xDB);
Sfr (CCAP2H  , 0xFC);
Sfr (CCAP2L  , 0xEC);
Sfr (CCAPM2  , 0xDC);
Sfr (CCAP3H  , 0xFD);
Sfr (CCAP3L  , 0xED);
Sfr (CCAPM3  , 0xDD);
Sfr (CCAP4H  , 0xFE);
Sfr (CCAP4L  , 0xEE);
Sfr (CCAPM4  , 0xDE);

#endif
```

Anhang 2: Neues Control – File „LCD_Ctrl_ATMEL.c“ zur Verwendung des LCD-Displays

```
/*------------------------------------------------------------------
LCD_CTRL_ATMEL.c
C-Programm-Modul  zur Ausgabe von Texten auf dem LCD-Display
02.11.2007 / G. Kraus
------------------------------------------------------------------*/
#include <t89c51ac2.h>       // Header für AT89C51AC3
#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=0xA0;                  // Port 2 (Adresse 0xA0) als Display-Ausgang


sbit enable=ausg_lcd^4;             // Portpin P2.4 ist "enable"
sbit RW_LCD=ausg_lcd^5;             // Portpin P2.5 ist "READ - WRITE"
sbit RS_LCD=ausg_lcd^6;             // Portpin P2.6 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
}
```

Anhang 3: Neues Control – File „LCD_CTRL_ATMEL_7pins.c“ mit freiem Pin 8 beim Einsatz des LCD-Displays

```
/*-------------------------------------------------------------------------------------------------------------------
LCD_CTRL_ATMEL_7pins.c
C-Programm-Modul  zur Ausgabe von Texten auf dem LCD-Display, wobei der freie achte Pin P2^7 des Display-
Ausgangs für andere Zwecke verwendet werden kann.
22.11.2007 / G. Kraus
-------------------------------------------------------------------------------------------------------------------*/
#include <t89c51ac2.h>                  // Header für AT89C51AC3
#include <stdio.h>                      // Standard-Eingabe-Ausgabe-Header

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);

void conv_lcd_ausg(unsigned char a);

sfr ausg_lcd=0xA0;                      // Port 2 als Display-Ausgang

unsigned char temp_1;
unsigned char temp_2;

sbit enable=ausg_lcd^4;                 // Portpin P2.4 ist "enable"
sbit RW_LCD=ausg_lcd^5;                 // Portpin P2.5 ist "READ - WRITE"
sbit RS_LCD=ausg_lcd^6;                 // Portpin P2.6 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();
        }

        temp_1=0x03;                    // Dreimal nacheinander dasselbe Codewort
        conv_lcd_ausg(temp_1);

        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
{       temp_1=0x02;                    // "0010B"  = 0x02 ergibt 4-Bit-Darstellung beim Display
        conv_lcd_ausg(temp_1);
        lcd_com();                      // Code senden

        temp_1=0x02;                    // Oberes Nibble für "4 Bit / 2 Zeilen / 5x7 dots"
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken

        temp_1=0x08;                    // Unteres Nibble für "4 Bit / 2 Zeilen / 5x7 dots"
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken

        temp_1=0x00;                    // Oberes Nibble von "Display OFF"
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken

        temp_1=0x08;                    // Unteres Nibble von "Display OFF"
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken

        temp_1=0x00;                    // Oberes Nibble von "Clear all"
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken

        temp_1=0x01;                    // Unteres Nibble von "Clear all"
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken

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

        temp_1=0x00;                    // Oberes Nibble von "Entry mode" (Cursor steht, Display wandert
        conv_lcd_ausg(temp_1);          // nach rechts
        lcd_com();                      // losschicken

        temp_1=0x06;                    // Unteres Nibble von "Entry mode" (Cursor steht, Display wandert
        conv_lcd_ausg(temp_1);          // nach rechts
        lcd_com();                      // losschicken

        temp_1=0x00;                    // Oberes Nibble von "Display ON, Cursor OFF, Kein Blinken"
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken

        temp_1=0x0c;                    // Unteres Nibble von "Display ON, Cursor OFF, Kein Blinken"
        conv_lcd_ausg(temp_1);
        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
{       temp_1=0x08;                    // Erforderliches Codewort hat als oberes Nibble "1000B" = 0x08
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken

        temp_1=0x00;                    // Erforderliches Codewort hat als unteres Nibble "0000B = 0x00
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken
}
```

```
void switch_z2(void)                    // Schalte auf Zeile 2 um und stelle den Cursor an den Anfang der Zeile
{       temp_1=0x0c;                    // Erforderliches Codewort hat als oberes Nibble "1100B" = 0x0C
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken

        temp_1=0x00;                    // Erforderliches Codewort hat als oberes Nibble "0000" = 0x00
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken
}

void switch_z3(void)                    // Schalte auf Zeile 3 um und stelle den Cursor an den Anfang der Zeile
{       temp_1=0x09;                    // Erforderliches Codewort hat als oberes Nibble "1001B" = 0x09
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken

        temp_1=0x04;                    // Erforderliches Codewort hat als unteres Nibble "0100B = 0x04
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken
}



void switch_z4(void)                    // Schalte auf Zeile 4 um und stelle den Cursor an den Anfang der Zeile
{       temp_1=0x0D;                    // Erforderliches Codewort hat als oberes Nibble "1101B" = 0x0D
        conv_lcd_ausg(temp_1);
        lcd_com();                      // losschicken

        temp_1=0x04;                    // Erforderliches Codewort hat als oberes Nibble "0100" = 0x04
        conv_lcd_ausg(temp_1);
        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. */
        {
        temp_1=(*ptr/0x10)|0x40;        /*  Oberes Nibble des Zeichen-Bytes wird um 4 Stellen nach rechts
                                            geschoben und anschließend um die erforderlichen Steuersignale
                                            ergänzt. */
        conv_lcd_ausg(temp_1);
        lcd_com();                      // Oberes Nibble losschicken

        temp_1=(*ptr&0x0f)|0x40;        /*  Unteres Nibble des Zeichen-Bytes wird durch Löschen des
                                            oberen Nibbles gewonnen und anschließend um die
                                            erforderlichen Steuersignale ergänzt. */
        conv_lcd_ausg(temp_1);
        lcd_com();                      //  Unteres Nibble losschicken


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


void show_char(char *ptr)               //  Anzeige eines einzigen ASCII-Zeichens. Adresse wird übergeben.
{       temp_1=(*ptr/0x10)|0x40;        /*  Oberes Nibble des Zeichen-Bytes wird um 4 Stellen nach
                                            rechts geschoben und anschließend um die erforderlichen
                                            Steuersignale (für Datenübertragung) ergänzt. */
        conv_lcd_ausg(temp_1);
        lcd_com();                      //  Oberes Nibble losschicken

        temp_1=(*ptr&0x0f)|0x40;        /*  Unteres Nibble des Zeichen-Bytes wird durch Löschen des
                                            oberen Nibbles gewonnen und anschließend um die erforderlichen
                                            Steuersignale (für Datenübertragung) ergänzt. */
        conv_lcd_ausg(temp_1);
        lcd_com();                      // Unteres Nibble losschicken
```

```
}


void conv_lcd_ausg(unsigned char a)
{       temp_1=(ausg_lcd)&(0x80);       // Pin p5^7 - Pegel speichern
        temp_2=(a)&(0x7F);              // dieses Bit bei ausg_lcd auf Null setzen
        ausg_lcd=(temp_1)|(temp_2);     // alten Bitwert einkopieren
}
```