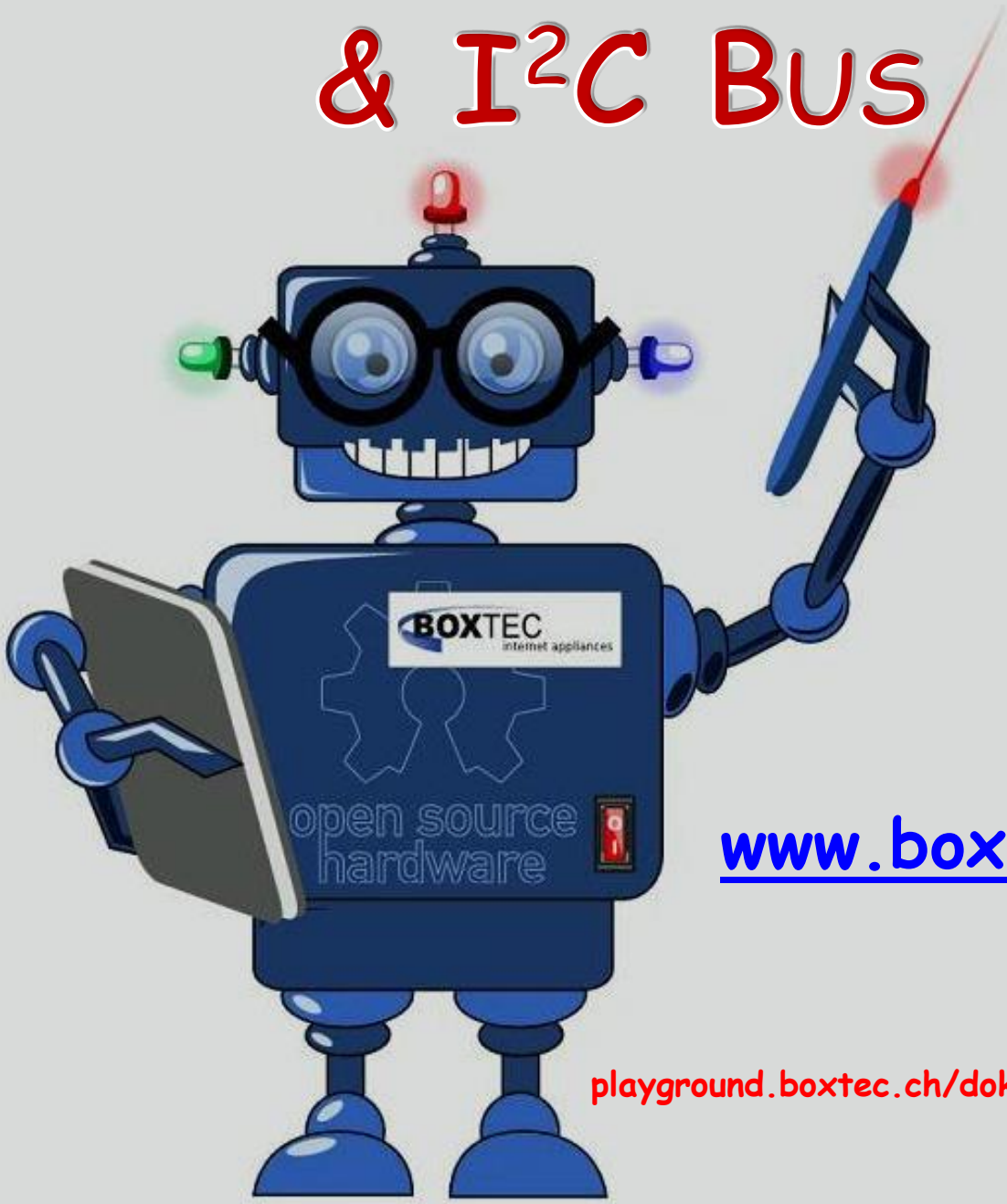
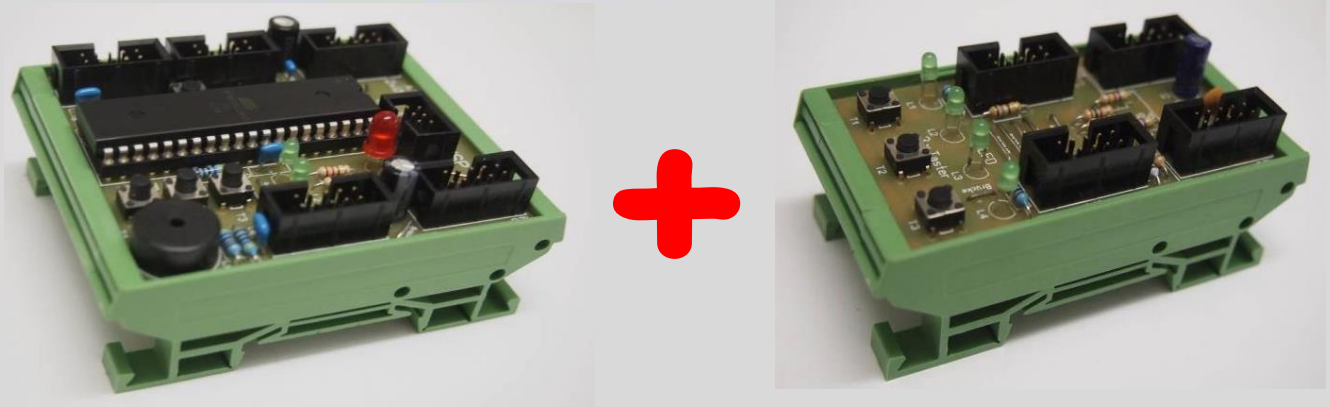
Mikrokontroller & I²C Bus



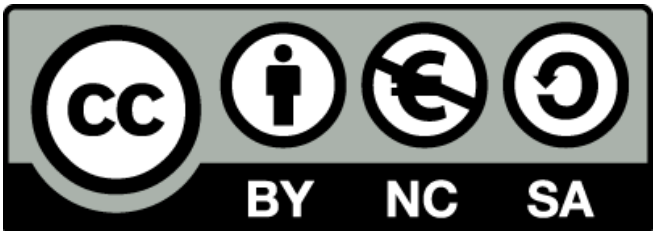
www.boxtec.ch

playground.boxtec.ch/doku.php/tutorial



Multitasking 1

Copyright



Sicherheitshinweise

Lesen Sie diese Gebrauchsanleitung, bevor Sie diesen Bausatz in Betrieb nehmen und bewahren Sie diese an einem für alle Benutzer jederzeit zugänglichen Platz auf. Bei Schäden, die durch Nichtbeachtung dieser Bedienungsanleitung verursacht werden, erlischt die Gewährleistung/Garantie. Für Folgeschäden übernehmen wir keine Haftung! Bei allen Geräten, die zu ihrem Betrieb eine elektrische Spannung benötigen, müssen die gültigen VDE-Vorschriften beachtet werden. Besonders relevant sind für diesen Bausatz die VDE-Richtlinien VDE 0100, VDE 0550/0551, VDE 0700, VDE 0711 und VDE 0860. Bitte beachten Sie auch nachfolgende Sicherheitshinweise:

- Nehmen Sie diesen Bausatz nur dann in Betrieb, wenn er zuvor berührungssicher in ein Gehäuse eingebaut wurde. Erst danach darf dieser an eine Spannungsversorgung angeschlossen werden.
- Lassen Sie Geräte, die mit einer Versorgungsspannung größer als 24 V- betrieben werden, nur durch eine fachkundige Person anschließen.
- In Schulen, Ausbildungseinrichtungen, Hobby- und Selbsthilfewerkstätten ist das Betreiben dieser Baugruppe durch geschultes Personal verantwortlich zu überwachen.
- In einer Umgebung in der brennbare Gase, Dämpfe oder Stäube vorhanden sind oder vorhanden sein können, darf diese Baugruppe nicht betrieben werden.
- Im Falle eine Reparatur dieser Baugruppe, dürfen nur Original-Ersatzteile verwendet werden! Die Verwendung abweichender Ersatzteile kann zu ernsthaften Sach- und Personenschäden führen. Eine Reparatur des Gerätes darf nur von fachkundigen Personen durchgeführt werden.
- Spannungsführende Teile an dieser Baugruppe dürfen nur dann berührt werden (gilt auch für Werkzeuge, Messinstrumente o.ä.), wenn sichergestellt ist, dass die Baugruppe von der Versorgungsspannung getrennt wurde und elektrische Ladungen, die in den in der Baugruppe befindlichen Bauteilen gespeichert sind, vorher entladen wurden.
- Sind Messungen bei geöffnetem Gehäuse unumgänglich, muss ein Trenntrafo zur Spannungsversorgung verwendet werden
- Spannungsführende Kabel oder Leitungen, mit denen die Baugruppe verbunden ist, müssen immer auf Isolationsfehler oder Bruchstellen kontrolliert werden. Bei einem Fehlers muss das Gerät unverzüglich ausser Betrieb genommen werden, bis die defekte Leitung ausgewechselt worden ist.
- Es ist auf die genaue Einhaltung der genannten Kenndaten der Baugruppe und der in der Baugruppe verwendeten Bauteile zu achten. Gehen diese aus der beiliegenden Beschreibung nicht hervor, so ist eine fachkundige Person hinzuzuziehen

Bestimmungsgemäße Verwendung

- Auf keinen Fall darf 230 V~ Netzspannung angeschlossen werden. Es besteht dann Lebensgefahr!
- Dieser Bausatz ist nur zum Einsatz unter Lern- und Laborbedingungen konzipiert worden. Er ist nicht geeignet, reale Steuerungsaufgaben jeglicher Art zu übernehmen. Ein anderer Einsatz als angegeben ist nicht zulässig!
- Der Bausatz ist nur für den Gebrauch in trockenen und sauberen Räumen bestimmt.
- Wird dieser Bausatz nicht bestimmungsgemäß eingesetzt kann er beschädigt werden, was mit Gefahren, wie z.B. Kurzschluss, Brand, elektrischer Schlag etc. verbunden ist. Der Bausatz darf nicht geändert bzw. umgebaut werden!
- Für alle Personen- und Sachschäden, die aus nicht bestimmungsgemäßer Verwendung entstehen, ist nicht der Hersteller, sondern der Betreiber verantwortlich. Bitte beachten Sie, dass Bedien- und /oder Anschlussfehler außerhalb unseres Einflussbereiches liegen. Verständlicherweise können wir für Schäden, die daraus entstehen, keinerlei Haftung übernehmen.
- Der Autor dieses Tutorials übernimmt keine Haftung für Schäden. Die Nutzung der Hard- und Software erfolgt auf eigenes Risiko.

Multitasking 1 (einfacher Ansatz)

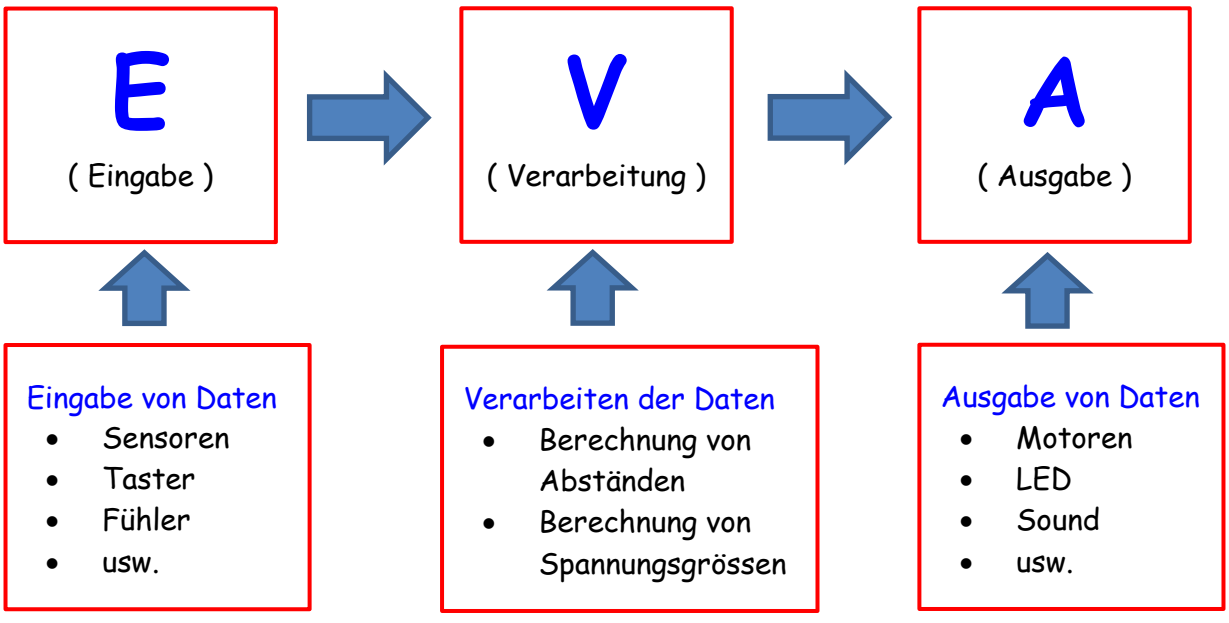
Diese Tutorial entstand unter Verwendung von Beiträgen von Falk. Dabei werden Textstellen wörtlich zitiert. http://www.mikrocontroller.net/articles/Multitasking

1. Einleitung

Da eine echte parallele Ausführung von mehreren Prozessen (Programmen, Funktionen) auf einem einzelnen CPU-Kern nicht möglich ist, wird ein "Trick" verwendet. Dabei werden die einzelnen Prozesse jeweils nur für kurze Zeit (1..50 ms) bearbeitet und danach auf einen anderen Prozess umgeschaltet.
Man spricht auch von einer verschachtelten Bearbeitung (**interleaving**).
Das Herz jedes Multitasking-Systems ist der Scheduler. Dieses Programm beinhaltet einen Algorithmus, der überprüft, welcher Prozess als nächstes die CPU (also Rechenzeit) zugeteilt bekommt. Leider ist das auf unserem ATmega 128 nicht so vorgesehen. An Hand von Beispielen möchte ich euch zeigen, was möglich ist.

2. Das EVA – Prinzip

Sehen wir uns einmal die Verarbeitung von Daten auf einem Rechner an:



Das EVA-Prinzip bedeutet also, die Eingabe, Verarbeitung und Ausgabe der Daten. Da wir nur einen CPU-Kern haben, können wir auch Eingabe, Verarbeitung oder Ausgabe nicht gleichzeitig durchführen, sondern nur streng sequentiell. Sehen wir uns mal ein kleines Stück Programm dazu an:

```
/* ATB_Multi_1.c  Created: 17.08.2014 11:05:43 Author: AS */

#define F_CPU 16000000UL      // Angabe der Quarzfrequenz, wichtig für die Zeit
#include <util/delay.h>       // Einbindung Datei Pause
#include <avr/io.h>           // Einbindung Datei Ausgänge

int main(void)
 {
   DDRC=0b00100000;           // Port C auf Ausgang schalten
   while(1)                   // Programmschleife
    {
      PORTC &= ~(1<<PC5);     // Schaltet Pin aus
      _delay_ms(500);         // Pause 500 ms
      PORTC |= (1<<PC5);      // Schaltet Pin ein
      _delay_ms(500);         // Pause 500 ms
    }
 }                            // Ende Programm
```

Zur Erzeugung einer Blinkfrequenz von 1 Sekunde benutzen wir den Befehl **_delay_ms(500).** Damit erfolgt jeweils 0,5 s (500ms) LED an und 0,5s (500ms) LED aus. Das entspricht einer Blinkzeit von 1 Sekunde.
Was macht der Prozessor eigentlich während dieser Zeit?
Die Antwort ist sehr einfach – eigentlich gar nichts. Ist nicht ganz korrekt, er zählt fleissig vor sich hin und führt „Nichts-Befehle" (nop) aus. Er tritt sozusagen auf der Stelle, mit Blick auf die Uhr und wartet bis die Zeit um ist.

3. Warum kann „_delay_ms (..)" gefährlich sein?

Nehmen wir doch einfach mal die folgende Situation an.
Unser Prozessor ist damit beschäftigt, die Zeit zum Einschalten einer LED zu berechnen bzw. zu zählen. Während dieser Zeit können viele andere Informationen an den Ports/Pins anliegen. Da der Prozessor mit zählen beschäftigt ist, erfolgt keine Abfrage dieser Pins.
In unserem Beispiel kann z.B. einer der 3 Taster auf unserem Board betätigt werden, ohne dass eine Reaktion erfolgt. Das kann fatale Folgen haben. Sicherheitseingänge werden nicht abgefragt, Abstandssensoren und Bodensensoren nicht beachtet usw. Bei Fahrmodellen kann es zu einem Absturz kommen und damit vielleicht zu einem Totalschaden.

4. Wie können wir Anweisungen fast gleichzeitig ausführen?

Wollen wir mehrere Sachen fast gleichzeitig ausführen, gibt es Probleme mit der Zeit.
Nehmen wir als Beispiel:

- **Eine Taste abfragen und damit eine LED schalten**
- **Eine LED mit ca. 2 s blinken lassen**
- **Eine LED mit ca. 0,6 s blinken lassen**

4.1. Einfacher Ansatz

Versuchen wir als nächstes 2 LED mit unterschiedlichen Zeiten blinken zu lassen.
Unten sehen wir das Programmes **ATB_Multi_2**. Wie man sieht, werden innerhalb der Hauptschleife die einzelnen Unterprogramme aufgerufen.

```
/* ATB_Multi_2.c Created: 17.08.2014 12:07:45 Author: AS */

#define F_CPU 16000000UL        // Angabe der Quarzfrequenz, wichtig für die Zeit
#include <util/delay.h>         // Einbindung Datei Pause
#include <avr/io.h>             // Einbindung Datei Ausgänge

void led_blinken1()             // Unterprogramm 1
 {
  PORTC &= ~(1<<PC5);           // Schaltet Pin aus
  _delay_ms(2000);              // Pause 2000 ms
  PORTC |= (1<<PC5);            // Schaltet Pin ein
  _delay_ms(2000);              // Pause 2000 ms
 }

void led_blinken2()             // Unterprogramm 2
 {
  PORTC &= ~(1<<PC6);           // Schaltet Pin aus
  _delay_ms(600);               // Pause 600 ms
  PORTC |= (1<<PC6);            // Schaltet Pin ein
  _delay_ms(600);               // Pause 600 ms
 }

int main(void)
 {
  DDRC=0b01100000;              // Port C auf Ausgang schalten
  while(1)                      // Programmschleife
   {
    led_blinken1();             // Aufruf Unterprogramm 1
    led_blinken2();             // Aufruf Unterprogramm 2
   }
 }
```

Wenn man das Programm nun laufen lässt, wird man feststellen dass (bitte mit **ATB_Multi_2** testen)

- die Abarbeitung der Unterprogramme für das blinken nacheinander erfolgt und die LED damit sehr komisch laufen

Dieser Ansatz ist also untauglich für uns. Egal wie schnell unser AVR auch ist, er reagiert sehr langsam. Sehen wir uns einmal den Grund für unser Problem an. Es werden innerhalb der **while** Schleife immer wieder die folgenden Unterprogramme ausgeführt:

led_blinken1();	→	LED blinkt, verzögert 2 x 2 s
led_blinken2();	→	LED blinkt, verzögert 2 x 0,6 s

Damit haben wir beiden jedem Durchlauf des Programmes eine Verzögerung von ca. 5 bis 6 s

Diese Zeit ist für den Prozessor eine Ewigkeit.

Sehen wir uns einmal die betreffende Stelle im Programm an.

```
void led_blinken1()                 // Unterprogramm 1
 {
   PORTC &= ~(1<<PC5);              // Schaltet Pin aus
   _delay_ms(2000);                 // Pause 2000 ms
   PORTC |= (1<<PC5);               // Schaltet Pin ein
   _delay_ms(2000);                 // Pause 2000 ms
 }
```

In diesem Unterprogramm finden wir unsere grössten Bremsen, 2 x _delay_ms (2000). Das sind allein rund 4 Sekunden. In den anderen Teilen finden wir noch mehr. Damit ist diese Version für uns nicht geeignet. Man kann im Grund sagen:

Warten verboten!

(Die Anweisung delay sollten wir nur selten verwenden !)

In Abhängigkeit des Programmes kann man die Anweisung „delay" nutzen. Dabei ist es aber notwendig genau abzuwägen, wo diese sinnvoll ist. Solange ich keine Zeitkritischen Prozesse habe oder Stromsparen muss, kann ich diese Funktion verwenden.

4.2. Verbesserter Ansatz (*)

Will man mehrere Dinge gleichzeitig bearbeiten, muss man die Aufgaben in kleinste Häppchen zerteilen. Diese kleinsten Häppchen werden dann verschachtelt abgearbeitet, also ein Häppchen von Aufgabe A, ein Häppchen von Aufgabe B, ein Häppchen von Aufgabe C, usw.
Sehen wir uns einmal das Programm **ATB_Multi_3** an. Eigentlich macht es dasselbe wie das Programm **ATB_Multi_2**, eigentlich …
Richtig gesehen, läuft einiges ganz anders.

```
/* ATB_Multi_3.c Created: 17.08.2014 16:30:50 Author: AS */

#define F_CPU 16000000UL        // Angabe der Quarzfrequenz, wichtig für die Zeit
#include <util/delay.h>         // Einbindung Datei Pause
#include <avr/io.h>             // Einbindung Datei Ausgänge

int16_t led1=0;
int16_t led2=0;

void led_blinken1()
 {
   led1++;
   if(led1==1999)
   PORTC &= ~(1<<PC5);          // Schaltet Pin PC5
   else
    {
      if(led1==3999)
```

```
        {
          PORTC |= (1<<PC5);          // Schaltet Pin PC5
          led1=0;
        }
    }
 }
void led_blinken2()
 {
   led2++;
   if(led2==599)
   PORTC &= ~(1<<PC6);               // Schaltet Pin PC6
   else
    {
      if(led2==1199)
        {
          PORTC |= (1<<PC6);         // Schaltet Pin PC6
          led2=0;
        }
    }
 }
int main(void)
 {
   DDRC=0b01100000;                  // Port C auf Ausgang schalten
   while(1)                          // Programmschleife
    {
      led_blinken1();                // Aufruf Unterprogramm 1
      led_blinken2();                // Aufruf Unterprogramm 1
      _delay_ms(1);                  // Pause 1 ms
    }
 }
```

Innerhalb der **while** Schleife ist die Anweisung **_delay_ms (1)** dazu gekommen. Ich kann schon einige Leute hören, was die so sagen. Habe doch gesagt: **Warten verboten !** Stimmt genau, hab ich gesagt. Es ist aber notwendig, sich das gesamte Programm anzusehen, bevor man etwas sagt.
Eines vorweg dazu. Das Programm **ATB_Multi_3** läuft ganz hervorragend. Beide LEDs werden entsprechend den gewählten Zeiten ein- und ausgeschaltet.
Warum eigentlich?
Fangen wir mit der **while** Schleife an. In ihr werden die einzelnen Unterprogramme aufgerufen.

```
   while(1)
    {
       led_blinken1();                  // Aufruf UPrg led blinken 1 mit 2 s
       led_blinken2();                  // Aufruf UPrg led blinken 2 mit 0,6 s
       _delay_ms(1);                    // einziges delay im Programm
    }
```

Es erfolgt dadurch ein Aufruf von Unterprogrammen, ein Rücksprung zum Hauptprogramm und wieder der Aufruf eines Unterprogrammes usw. Sehen wir uns mal ein Unterprogramm an z.B. **led blinken1()**

```
( 1 ) // *** led blinken 1 ***
( 2 ) void led_blinken1 ()                  // UPrg led blinken 1
( 3 )  {                                    // Klammer Anfang
( 4 )      led1++;                          // Zähler led 1
( 5 )      if (led1==1999)                  // Abfrage Zähler
( 6 )      PORTC &= ~(1<<PC5);              // setzt PIN PC5
( 7 )       else
( 8 )         {
( 9 )           if (led1==3999)            // Abfrage Zähler
( 10 )          {
( 11 )            PORTC |= (1<<PC5);       // setzt Pin PC5
( 12 )            led1=0;                  // setzt Zähler led 1 auf 0
                }
             }
        }                                   // Klammer Ende
```

In der **Zeile 2** wird das Unterprogramm gestartet. Bei jedem Durchlauf wird in der **Zeile 4** zum Zähler **led1** ein Betrag von 1 addiert. In der **Zeile 5** wird die Höhe des Zählers **led1** mit dem eingestellten Wert von 1999 verglichen. Entspricht der Zähler dem eingestellten Wert, wird **Zeile 6** ausgeführt und die **PC5** geschaltet. Wenn nicht wird durch die **Zeile 7** die **Zeile 9** ausgeführt. Wieder wird der Zähler **led1** mit dem eingestellten Wert von 3999 verglichen. Stimmen Zähler und Wert überein, wird **Zeile 11** ausgeführt, die **PC5** geschaltet und der Zähler **led1** auf 0 gesetzt. Durch die beiden (1999 + 3999) Zahlen wird die Länge der Zeit bestimmt.
Wird eine andere Zeit gewählt, muss die Grösse der Zahlen entsprechen angepasst werden.

Und damit kommen wir zu **_delay_ms(1)**. Mit dieser kurzen Pause wird der gesamte Durchlauf eines Programm Zyklus auf mindestens 1 ms gesetzt. Innerhalb einer Zeit von 1 ms und der notwendigen Zeit für die Anweisungen wird das gesamte Programm durchlaufen. Während dieser Zeit erfolgt das schalten der Zähler (+/-), das Einlesen der Eingänge und das setzen der Ausgänge.
Damit kann ich, bei einer eingestellten Zeit von 1999, von einer Laufzeit von z.B. 1999 ms ausgehen. Das entspricht ca. 2 Sekunden und damit meiner gewünschten Blinkzeit.
Bei jedem Durchlauf kann ich mehrere Zähler setzen und damit verschiedene Aufgaben (fast) gleichzeitig ausführen.
Die einzelnen kleinen Häppchen sind verdaulicher als die grossen. Die maximale Durchlaufzeit der einzelnen Funktionen ist drastisch reduziert. Anstatt in der LED-Ausgabe einmal 2000 ms zu warten wird nun 2000 x 1ms gewartet. Zwischendurch werden aber 2000 mal die anderen Prozesse bearbeitet. Echte Demokratie sozusagen.
Das ist eigentlich der ganze "Trick" eines kooperativen Multitaskings. Auch wenn die Verwendung von _delay_ms(1) noch ein kleiner Schönheitsfehler ist, den die Profis lieber mit einem **Timer** erledigen, so wird das Prinzip klar.

- **Prozesse eines kooperativen Multitaskingsystems warten nicht auf das Eintreten von Ereignissen, sondern bearbeiten nur bereits eingetretene Ereignisse.**
- **Grössere Aufgaben werden in kleine Teilaufgaben zerlegt, welche nur durch mehrfaches Aufrufen der Funktion abgearbeitet werden.**
- **Prozesse eines kooperativen Multitaskings haben eine garantierte, maximale Durchlaufzeit, welche möglichst klein ist.**

Damit ähneln die Prozesse einem **Interrupt,** auch wenn sie als ganz normale Funktionen ausserhalb eines Interrupts ausgeführt werden. An diesem Beispiel erkennt man die Vor- und Nachteile des kooperativen Multitaskings

Vorteile

- einfacher Scheduler mit geringster CPU Belastung und Speicher
- Deterministische Arbeitsweise, damit einfach prüfbar und strenges Timing möglich

Nachteile

- eine andere Programmierweise zur Zerlegung größerer Aufgaben in kleine Teilaufgaben muss manuell vorgenommen werden
- Alle Häppchen werden gleich oft aufgerufen und nicht nur bei Bedarf

Wichtigster Grundsatz ist die Herangehensweise! Viele Programmieranfänger haben damit Schwierigkeiten (ich auch), was u.a. an den schlecht vermittelten (gelernten) Grundlagen liegt.

Verwendet Hardware: Board 1 (ATmega 1284) und NT2 im System 72

Ich hoffe, dass dieser Tut zu einer anregenden Diskussion führt. Bin jeder Zeit offen für Erweiterungen oder sinnvolle Ergänzungen.
Es werden noch weitere Tuts zu diesem Thema folgen. Man kann sehr viel damit machen. Besonders Sachen, die vorher unmöglich erschienen z.B.

- **Tasterentprellung**
- **Mehrfaches Schalten mit einem Taster (kurz/lang)**
- **Darstellung eines Menue und Auswahl mit Taster**
- **Aufruf von Notfunktionen**
- **… und viele andere Sachen**

Einige Teile des Textes wurden zur besseren Übersicht farblich gestaltet.
Die Nutzung erfolgt auf eigenes Risiko.
Ich wünsche viel Spaß beim Bauen und programmieren
Achim

myroboter@web.de