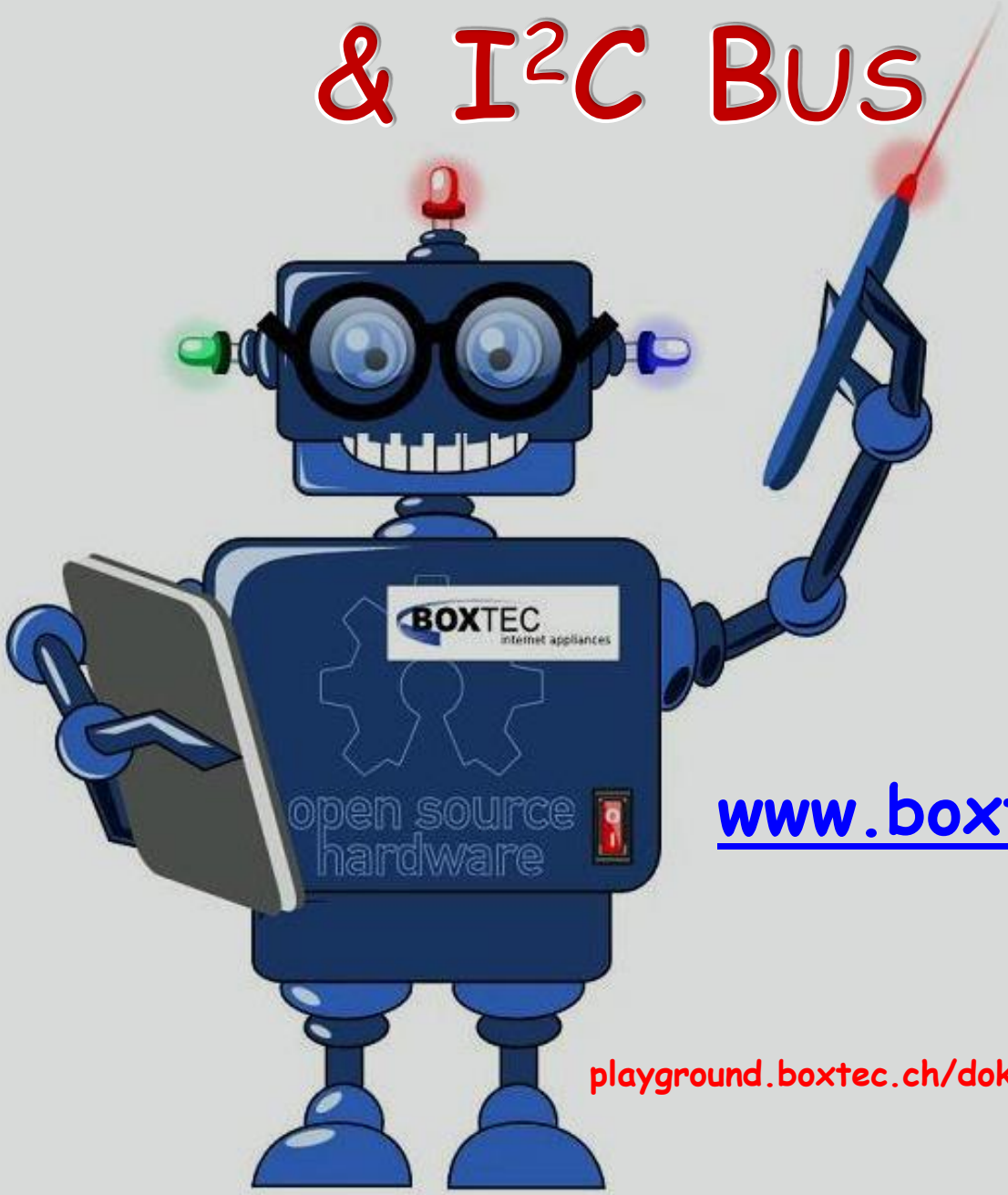
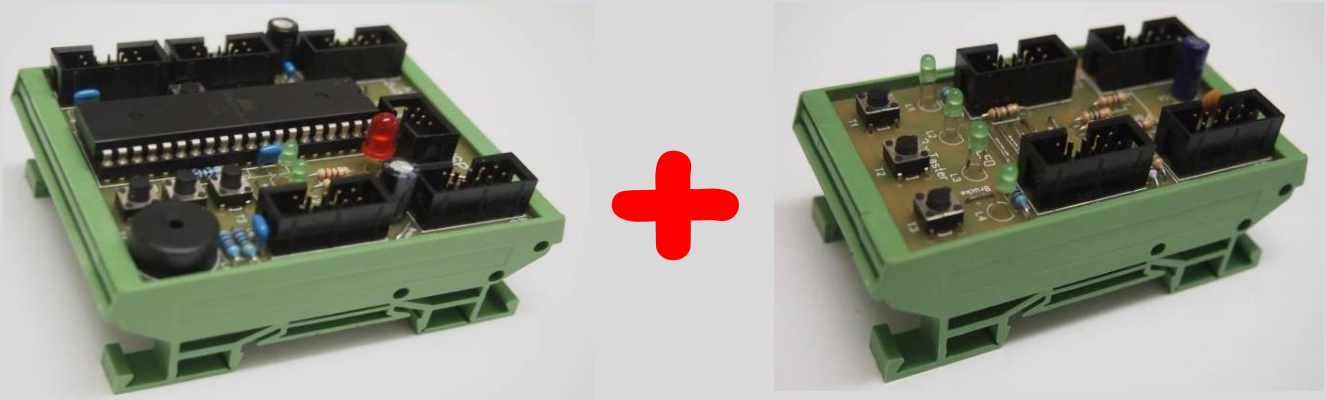
Mikrokontroller & I²C Bus



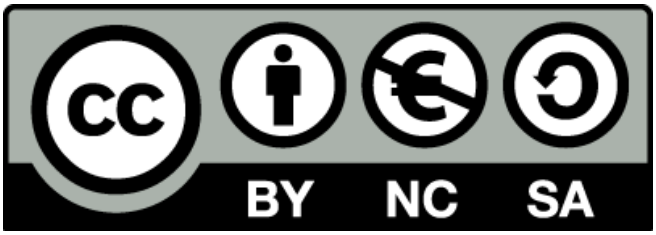
www.boxtec.ch

playground.boxtec.ch/doku.php/tutorial



Multitasking 4

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 4 (mit Timer)

12. Was sind Timer ?

Timer sind selbstständige Zähler im Prozessor. Man braucht sie dort, wo Zeitkritische und genaue Aufgaben gefordert werden, z.B. Zeitmessung.
Ein Timer ist ein Zähler, der anfängt von Null an hochzuzählen, bis er seinen Höchstwert oder einen eingestellten Vergleichswert erreicht hat. Dieser Höchstwert oder Vergleichswert ist abhängig von der Menge der sogenannten Bits, die für die Anzahl der Speicherstellen stehen.
Wir haben in unserem ATmega 1284 8- und 16- Bit Timer.

- **Timer mit 8 Bit – gehen von 0 bis 255 (256)**
- **Timer mit 16 Bit – gehen von 0 bis 65535 (65536)**

Bei jedem Überschreiten des Höchstwertes oder Erreichen eines Vergleichswertes hat man die Möglichkeit ein sogenannten Interrupt auszulösen. Man kann ihn also anweisen in ein Unterprogramm zu springen und dieses abzuarbeiten.
Es gibt verschiedene Betriebsarten. So kann er z.B. zwei Werte vergleichen, oder hoch zählen bis zu einem bestimmten Wert und dann wieder rückwärts zählen. Die Art (Mode) wie der Timer zählt, wird ihm durch ein Befehl mitgeteilt.
Der Prozessor wird durch einen Quarz mit 16 MHz getaktet. Das sind 16 000 000 Takte in der Sekunde. Bei dieser Frequenz würde der Timer sehr schnell zum Ende kommen und einen Interrupt auslösen.
Damit der Timer bei einer Taktfrequenz von 16 MHz nicht zu schnell einen Interrupt auslöst, haben wir „Prescaler". Das sind Vorteiler, die den Systemtakt durch festgelegte Werte teilen und den Timer langsamer laufen lassen. Es sind z.B. Teilungen von 8, 64, 256 und 1024 möglich.

13. Welche Timer hat der ATmega 1284p ?

Im ATmega 1284p habe ich die folgenden Timer:

- **Timer 0 – 8 Bit – 0 bis 255 (256)**
- **Timer 1 – 16 Bit – 0 bis 65535 (65536) (unklare Angabe)**
- **Timer 2 – 8 Bit – 0 bis 255 (256)**

Die Angaben habe ich dem Datenblatt von 2013 entnommen

14. Betriebsmodi

Die AVR-Timer können in unterschiedlichen Betriebsmodi betrieben werden. Diese sind:

- **Normaler Modus**
- **CTC Modus**
- PWM (gehen wir nicht drauf ein)

14.1. Mode 0 (Normal Modus)

In diesem Mode zählt der Timer immer von Null an aufwärts bis 255 beim 8-Bit Timer oder 65535 beim 16-Bit Timer. Nachdem er seinen Höchstwert erreicht hat fängt er wieder bei null an. Bei Bedarf kann bei jedem Überlauf ein Interrupt ausgelöst werden. Das heißt, dem Timer kann gesagt werden, das er bei jedem Überlauf eine Routine anspringen soll um diese dann abzuarbeiten. Das hat Priorität vor dem Ablauf der eigentlichen Programmschleife.

14.2. Mode 2 (CTC – Modus)

Hier zählt der von null an aufwärts. Es besteht die Möglichkeit einen Vergleichswert vorzugeben, bis zu dem der Timer zählen soll. Dann stellt er sich selbstständig wieder auf null und beginnt wieder von null an zu zählen. Er muss nicht zwangsläufig bis 255 oder 65535 zählen.

15. Unser erster Timer

```
/* ATB_Multi_11.c Created: 22.08.2014 10:59:55 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
#include <avr/interrupt.h>

int16_t Zled7=0;                // Variable für PA 7 - LED 7
volatile int8_t flag_1ms;       // Globale Variable flag_1ms

ISR (TIMER0_COMPA_vect)         // ISR
 {
   flag_1ms=1;                  // setzt flag_1ms auf 1
 }

void led_blinken1()             // Unterprogramm 1
 {
   Zled7++;
   if(Zled7==500)               // Angabe Zeit 500ms
   PORTA &= ~(1<<PA7);          // Schaltet PA7 ein
   else
    {
      if(Zled7==1000)           // Angabe Zeit 500 ms
        {
          PORTA |= (1<<PA7);    // Schaltet PA7 aus
          Zled7=0;              // setzt Zled7 auf 0
        }
    }
 }
```

```
void timer_init()                    // Timer 0 konfigurieren
 {
  TCCR0A = 0;                        // Es werden keine Bits gesetzt
  TCCR0B = (1<<WGM01)|(1<<CS01)|(1<<CS00);   // Einstellung CTC Modus, Prescaler 64
  TCNT0=1;                           // Initialisiert Timer
  OCR0A=249;                         // Laden des Vergleichswertes
  TIMSK0|=(1<<OCIE0A);               // Interrupt erlauben
 }

int main(void)
 {
  timer_init();                      // Initiiert Timer (erster Aufruf)
  DDRA=0b10000000;                   // setzt Port A, Pin PA7 auf Ausgang
  sei();                             // gibt Interrupts frei
  while(1)                           // Beginn Programmschleife while
   {
    if(flag_1ms)                     // Abfrage ob flag_1ms wahr (1) ist
      {                              // Wenn Abfrage wahr ist, dann …
       flag_1ms=0;                   // setzt flag_1ms auf 0
       led_blinken1();               // Aufruf Unterprogramm 1
      }
   }
 }
```

Im oberen Programm schaltet der Timer 0 eine LED mit einer Frequenz von 1 Hz ein und aus (je 500 ms an und 500 ms aus). Sehen wir uns die Funktion genauer an.

15.1. Unser Timer

Um unseren Timer einzustellen bedienen wir uns der sogenannten Register. Das sind Speicherplätze in denen wir die Art und Funktion der Timer konfigurieren können. Wir haben die folgenden Register zur Verfügung:

- **TCCR0A** – Timer/Counter Control Register (Gibt die Funktionsweise des Zählers an)
- **TCCR0B** – Timer/Counter Control Register (Gibt die Funktionsweise des Zählers an)
- **TCNT0** – Timer/Counter Register (Initialisiert Timer 0)
- **OCR0A** – Output Compare Register (Laden des Vergleichswertes)
- **TIMSK** – Timer/Counter Interrupt Mask (Interrupts einschalten)

15.2. Einstellung Timer

Mit dieser Erklärung können wir nun die einzelnen Funktionen unseres Timers ermitteln.

```
void timer_init()                    // Timer 0 konfigurieren
 {
  TCCR0A = 0;                        // Es werden keine Bits gesetzt
  TCCR0B = (1<<WGM01)|(1<<CS01)|(1<<CS00);   // Einstellung CTC Modus, Prescaler 64
  TCNT0=1;                           // Initialisiert Timer
  OCR0A=249;                         // Laden des Vergleichswertes
  TIMSK0|=(1<<OCIE0A);               // Interrupt erlauben
 }
```

In unserem Unterprogramm **timer_init()** wollen wir unseren Timer mit den gewünschten Einstellungen initiieren.
Im Register **TCCR0A** nehmen wir keine Einstellungen vor. Mit dem Register **TCCR0B** stellen wir den **CTC-Modus** ein und setzen den **Prescaler** auf **64**. Durch **TCNT0** wird der Timer initialisiert. Mit **OCR0A** setzen wir den Vergleichswert auf **249**. Und als letztes erlauben wir die Interrups mit **TIMSK0**.

15.3. Änderungen im RegisterTCCR0B

Bit	7	6	5	4	3	2	1	0
Name	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Wert	0	0	0	0	1	0	1	1

Wenn wir die Bits 0, 1 und 3 verändern, indem wir die 1 eintragen, so werden die 16 MHz durch 64 geteilt und CTC ausgeführt.

15.4. Welche Zeit / Einstellung habe ich / brauche ich ?

Quarzfrequenz: 16 MHz (16 000 000 Hz)
Gesuchte Zeit: 1 ms (entspricht 1000 Hz)
Vorteiler/Prescaler: 64

Mit der Quarzfrequenz von 16 MHz können wir unseren Timer nicht direkt ansteuern. Das hätte zur Folge, dass unser Programm viel zu schnell läuft. Deshalb müssen wir die Zeit ein bisschen verändern.

Zeitrechnung: Gesuchte Frequenz / Zeit = **1000** Hz entspricht **1** ms

$$\frac{16\ MHz\ (\ 16\ 000\ 000\ Hz\ )}{64\ (\ Prescaler\ )} = 250\ 000$$

Bei einer Quarzfrequenz von 16 MHz (16 000 000 Hz) und einem eingestellten Prescaler (Vorteiler) von 64 ergeben sich 250 000.

Als nächsten wird der errechnete Wert durch unsere Frequenz geteilt.

$$\frac{250\ 000}{1000\ Hz} = 250$$

Damit ergibt sich ein Wert von 250. Eingestellt wird 249, denn der Zähler läuft von 0 bis 249, das sind 250 Takte. Durch unseren Timer wird jede Millisekunde ein Interrupt ausgelöst. Da unser errechneter Wert kleiner 256 ist, können wir einen 8-Bit Timer nehmen.

16. ISR – Was ist das ?

Bei Mikrocontrollern werden Interrupts z. B. ausgelöst wenn:

- sich der an einem bestimmten Eingangs-Pin anliegende Pegel ändert
- eine vorher festgelegte Zeitspanne abgelaufen ist (Timer)
- eine serielle Übertragung abgeschlossen ist
- eine Messung des Analog-Digital-Wandlers abgeschlossen ist

Bei der Auslösung des Interrupts wird das Anwendungsprogramm unterbrochen, das auslösende Interruptflag gelöscht und ein Unterprogramm, die sogenannte **I**nterrupt **S**ervice

Routine (ISR), aufgerufen. Wenn dieses beendet ist, läuft das Anwendungsprogramm ganz normal weiter. Für die Auswertung des Interrups nutze ich die folgenden Zeilen

```
ISR (TIMER0_COMPA_vect)      // ISR
 {
  flag_1ms=1;                // setzt flag_1ms auf 1
 }
```

In diesem **ISR – Programm** wird **Globale Variable flag_1ms** auf **1 (wahr)** gesetzt.
Die Globale Variable muss am Anfang des Programmes definiert werden.

```
volatile int8_t flag_1ms;    // Globale Variable flag_1ms
```

Sie bewirkt, dass sie auch außerhalb der ISR vom Programm gesehen wird.

17. Die Auswertung

Innerhalb der while Schleife werte ich die Variable **flag_1ms** aus. Dabei erfolgt eine Abfrage ob **flag_1ms** wahr (**1**) ist. Wenn ja, wird **flag_1ms** auf **0** gesetzt und die entsprechenden Unterprogramme ausgeführt. Wenn **flag_1ms** nicht wahr (0) ist, wird die Schleife wiederholt.

```
while(1)                     // Beginn Programmschleife while
 {
  if(flag_1ms)               // Abfrage ob flag_1ms wahr (1) ist
    {                        // Wenn Abfrage wahr ist, dann …
     flag_1ms=0;             // setzt flag_1ms auf 0
     led_blinken1();         // Aufruf Unterprogramm 1
    }
 }
```

Das Unterprogramm **led_blinken1** wird danach ausgeführt. Innerhalb dieses Programmes darf es zu keiner Verzögerung kommen.

18. Anwendungen

Den Timer und das Unterprogramm **led_blinken1** kann ich bei fast allen Gelegenheiten nutzen. Durch das gleich mäßige Blinken habe ich immer eine Anzeige, dass mein Programm läuft. Es kann mir sofort eine Verzögerung des Durchlaufes anzeigen. Jeder Stopp wird durch eine Unterbrechung des Blinkens angezeigt.
Die Funktion des Unterprogrammes **led_blinken1** habe ich schon in einem anderen Teil beschrieben.

19. Erweiterung

Es kann beim Testen neuer Unterprogramme vorkommen, dass der Durchlauf unterbrochen wird oder eine falsche Funktion ausgeführt wird. Zur eigentlich Fehlersuch kommen wir später. Zur Erkennung kann das folgende Code Stück genommen werden.
Es muss nur innerhalb der while Schleife ausgetauscht werden. Es müssen allerdings ein paar Änderungen an den Einstellungen vorgenommen werden.

```
DDRA=0b11000000;             // Port A auf Ausgang schalten
                             // Pin 6 und 7 freigeben
```

Es muss zusätzlich der Pin 6 freigegeben werden

```
( 1 )        while(1)                          // Programmschleife
( 2 )         {
( 3 )          if(flag_1ms)                    // Abfrage flag_1ms
( 4 )           {
( 5 )            flag_1ms=0;                   // Setzt flag_1ms auf 0
( 6 )            PORTA |= (1<<PA6);            // Schaltet Pin 6 aus bei korrekten Betrieb
( 7 )            led_blinken1();               // Aufruf Unterprogramm 1
( 8 )            if (flag_1ms)                 // Abfrage flag_1ms
( 9 )              {
( 10 )               PORTA &= ~(1<<PA6);       // Schaltet PA6 ein
( 11 )               while(1);                 // Programmschleife
( 12 )             }
( 13 )          }
( 14 )        }
```

In der **Zeile 3** erfolgt die Abfrage von **flag_1ms**. Wenn diese **wahr (1)** ist wird die Programmabarbeitung in der **Zeile 5** fortgesetzt und **flag_1ms** auf **0** gesetzt. In der **Zeile 6** wird **PA 6** ausgeschaltet, damit sie bei korrektem Betrieb nicht leuchtet. Anschliessend wird das Unterprogramm in der **Zeile 7** ausgeführt.
In der **Zeile 8** erfolgt die zusätzlich Abfrage, ob **flag_1ms** wirklich auf **0** gesetzt wurde. Ist er noch **wahr (1)** wird **Zeile 10** ausgeführt und **PA 6** geschaltet. Das Programm wird durch die **Zeile 11** unterbrochen.

20. Fehlersuche

Je mehr Unterprogramme ich in meinem Programm verwende, umso höher ist die Wahrscheinlichkeit eines Fehlers. Doch in welchem Teil meines Programmes ist er versteckt?
Da hilft nur suchen oder ich gewöhne mir gleich einen bestimmten Ablauf an.

```
while(1)                          // Beginn Programmschleife while
    {
     if(flag_1ms)                 // Abfrage ob flag_1ms wahr (1) ist
       {                          // Wenn Abfrage wahr ist, dann …
        flag_1ms=0;               // setzt flag_1ms auf 0
        Unterprogramm 1;
        Unterprogramm 2;
        Unterprogramm 3;
        Unterprogramm 4;
        Unterprogramm 5;
        led_blinken1();           // Aufruf led_blinken1
       }
    }
```

So könnte es aussehen. Doch welches Unterprogramm verursacht den Fehler?
Eigentlich ganz einfach. Ich kommentiere alle Unterprogramme aus und verwende nur die minimal Version. Auskommentieren bedeutet:

```
// Unterprogramm 1;
// Unterprogramm 2;
```

Vor jedes Unterprogramm setze ich die Zeichen //. Dadurch wird der Aufruf und das eigentliche Unterprogramm nicht beachtet. Es verbleibt nur

```
if(flag_1ms)                // Abfrage ob flag_1ms wahr (1) ist
  {                         // Wenn Abfrage wahr ist, dann …
   flag_1ms=0;              // setzt flag_1ms auf 0
   led_blinken1();          // Aufruf Unterprogramm 1
  }
```

Damit müsste ein Durchlauf erfolgen und eine LED blinken. Danach kann ich die Unterprogramme einzeln wieder in Betrieb nehmen und testen was geht. Habe ich das nicht laufende Unterprogramm gefunden, hilft nur Fehlersuche.
Das wichtigste ist, ein funktionierender Timer und eine Blinkroutine.

21. Ein weiteres Programm

```
/* ATB_Multi_12.c Created: 23.08.2014 08:38:12 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
#include <avr/interrupt.h>
#include <stdint.h>

int16_t led1=0;
int16_t led2=0;
int taster1;
int taster2;

volatile int8_t flag_1ms;

ISR (TIMER0_COMPA_vect)
 {
   flag_1ms=1;
 }

int taste_lesen1()
 {
   if(PINA & 1<<PA3)
   return 1;
   else return 0;
 }

int taste_lesen2()
 {
   if(PINA & 1<<PA1)
   return 1;
   else return 0;
 }

void led_taster1(int taster1)
 {
   if(taster1==1)
```

```
    {
      PORTA &= ~(1<<PA4);          // Schaltet Pin  A4
      PORTA |= (1<<PA5);           // Schaltet Pin  A5
    }
   else
    {
      PORTA |= (1<<PA4);           // Schaltet Pin  A4
      PORTA &= ~(1<<PA5);          // Schaltet Pin  A5
    }
 }
void led_taster2(int taster2)
 {
   if(taster2==1)
    {
      PORTC &= ~(1<<PC5);          // Schaltet Pin
      PORTC |= (1<<PC6);           // Schaltet Pin
    }
   else
    {
      PORTC |= (1<<PC5);           // Schaltet Pin
      PORTC &= ~(1<<PC6);          // Schaltet Pin
    }
 }
void led_blinken1()
 {
   led1++;
   if(led1==300)
    {
      PORTA &= ~(1<<PA6);          // Schaltet Pin
      PORTA |= (1<<PA7);           // Schaltet Pin
    }
   else
    {
      if(led1==600)
        {
          PORTA |= (1<<PA6);       // Schaltet Pin
          PORTA &= ~(1<<PA7);      // Schaltet Pin
          led1=0;
        }
    }
 }
void timer_init()
 {                                 // Timer 0 konfigurieren
   TCCR0A = 0;                     // CTC Modus
   TCCR0B = (1<<WGM01)|(1<<CS01)|(1<<CS00);          // Prescaler 64
```

```
  TCNT0=1;
  OCR0A=249;
  TIMSK0|=(1<<OCIE0A);          // Compare Interrupt erlauben
 }

int main(void)
 {
  timer_init();
  DDRA=0b11110000;              // Port A auf Ausgang schalten
  DDRC=0b01100000;              // Port C auf Ausgang schalten
  sei();                        // Global Interrupts aktivieren
  while(1)                      // Programmschleife
   {
    if(flag_1ms)
      {
       flag_1ms=0;
       taster1 = taste_lesen1();  // Aufruf Unterprogramm 1
       led_taster1 (taster1);           // Aufruf Unterprogramm 2
       taster2 = taste_lesen2();  // Aufruf Unterprogramm 3
       led_taster2 (taster2);           // Aufruf Unterprogramm 4
       led_blinken1();                        // Aufruf Unterprogramm 5
      }
   }
 }
```

22. Ein-Knopf-Bedienung

Damit kommen wir zu einem recht anspruchsvollen Programm. Es läuft wieder mit Multitasking, mit einem 1 ms Timer und hat eine Tasterentprellung nach Peter Dannegger drin.
Das schönste daran ist aber die Bedienung.
Mit dem Taster T1 (PA1) schalte ich eine LED durch einen kurzen Druck auf den Taster ein.
Mit dem gleichen Taster T1 (PA1) schalte ich dieselbe LED mit einem langen Druck auf den Taster wieder aus.
Das ist eine echte **Ein-Knopf-Bedienung**. Ein- und Ausschalten mit demselben Taster !!!
Zusätzlich kann ich mit dem Taster T2 (PA2) die LED L3 einschalten und mit dem Taster T3 (PA3) wieder ausschalten. Also mit einem Taster einschalten und mit einem anderen Taster ausschalten.

```
/* ATB_Multi_13.c Created: 24.08.2014 08:38:22 Author: AS */

#define F_CPU 16000000UL      // Angabe der Quarzfrequenz, wichtig für die Zeit
#include <avr/io.h>           // Einbindung Datein
#include <avr/interrupt.h>
#include <stdint.h>

volatile int16_t led1=0;
volatile int8_t wait;
volatile int8_t flag_1ms;
volatile uint8_t key_state;
volatile uint8_t key_press;
```

```
volatile uint8_t key_rpt;
#define KEY_DDR    DDRA                 // Datenrichtung A
#define KEY_PORT   PORTA                // Angabe Port A
#define KEY_PIN    PINA                 // Angabe PIN A
#define KEY_1      1                    // PA 1
#define KEY_2      2                    // PA 2
#define KEY_3      3                    // PA 3
#define ALL_KEYS (1<<KEY_1|1<<KEY_2|1<<KEY_3)
#define REPEAT_MASK (1<<KEY_1|1<<KEY_2)
#define REPEAT_START 50                 // after 500ms
#define REPEAT_NEXT 20                  // every 200ms

ISR (TIMER0_COMPA_vect)                 // wait1=1ms,
 {
   static uint8_t ct0,ct1,rpt;
   uint8_t i;
   flag_1ms=1;
   if(wait<=9)                          // bei 9 sind es 10ms
    {
      wait++;
    }                                   // erhöht
   else                                 // wenn dann ...
    {
      wait=0;                           // setzt wait auf 0
      i=key_state ^~KEY_PIN;
      ct0=~(ct0&i);
      ct1=ct0^(ct1&i);
      i&=ct0&ct1;
      key_state^=i;
      key_press|=key_state&i;
      if((key_state & REPEAT_MASK)==0)
      rpt=REPEAT_START;
      if(--rpt==0)
       {
         rpt=REPEAT_NEXT;
         key_rpt|=key_state & REPEAT_MASK;
         }
      }
 }
uint8_t get_key_press(uint8_t key_mask)
 {
   cli();
   key_mask &=key_press;
   key_press^=key_mask;
   sei();
   return key_mask;
 }
```

```
uint8_t get_key_rpt(uint8_t key_mask)
 {
   cli();
   key_mask &=key_rpt;
   key_rpt^=key_mask;
   sei();
   return key_mask;
 }
uint8_t get_key_short(uint8_t key_mask)
 {
   cli();
   return get_key_press(~key_state & key_mask);
 }
uint8_t get_key_long(uint8_t key_mask)
 {
   return get_key_press(get_key_rpt(key_mask));
 }
void led_blinken1()
 {
   led1++;
   if(led1==500)
    {
      PORTA &= ~(1<<PA7);                  // Schaltet Pin
    }
   else
    {
      if(led1==1000)
        {
          PORTA |= (1<<PA7);               // Schaltet Pin
          led1=0;
        }
     }
   }
void timer_init()
 {                                         // Timer 0 konfigurieren
   TCCR0A = 0;                             // CTC Modus
   TCCR0B = (1<<WGM01)|(1<<CS01)|(1<<CS00);   // Prescaler 64
   TCNT0=1;
   OCR0A=249;
   TIMSK0|=(1<<OCIE0A);                    // Interrupt erlauben
 }
int main(void)
 {
   timer_init();
   DDRA=0b11110000;                        // Port A auf Ausgang schalten
   KEY_DDR&=~ALL_KEYS;
   KEY_PORT|=ALL_KEYS;
```

```
  PORTA |= (1<<PA4);
  PORTA |= (1<<PA5);
  PORTA |= (1<<PA6);
  sei();
  while(1)                          // Programmschleife
   {
     if(get_key_press(1<<KEY_2))    // nur Taste press
      {                             // LED an
        PORTA &= ~(1<<PA5);
      }
     if(get_key_press(1<<KEY_3))    // nur Taste press
      {                             // LED aus
        PORTA |= (1<<PA5);
      }
     if(get_key_short(1<<KEY_1))    // kurz schalten immer zusammen mit long
      {
        PORTA &= ~(1<<PA4);         // LED an
      }
     if(get_key_long(1<<KEY_1))     // Lang schalten immer mit short
      {
        PORTA |= (1<<PA4);          // LED aus
      }
     if(flag_1ms)
      {
        flag_1ms=0;
        led_blinken1();             // Aufruf Unterprogramm
      }
   }
}
```

Damit haben wir diesen Teil geschafft. Auf eine genaue Erklärung möchte ich verzichten. Habe zwar eine schriftlich mehrseitige Erklärung zu liegen, diese möchte ich euch ersparen. Versteh sie selber nicht, wie Peter das gemacht hat. Sorry, bin auch nur ein Mensch (mit Fehlern). Am besten Peter selber fragen.
Für mich ist die Funktion am wichtigsten. Wünsche viel Spass damit.

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