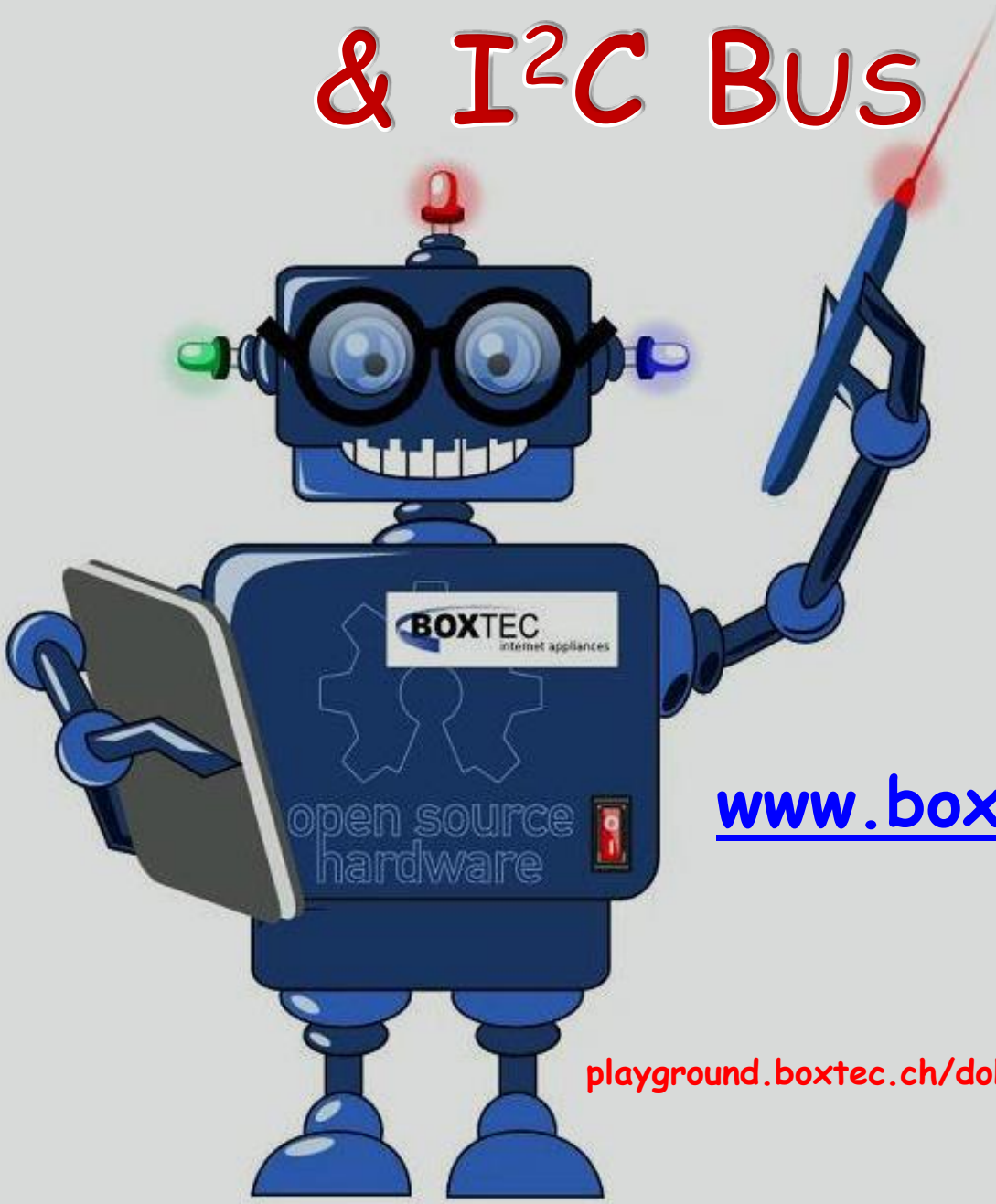
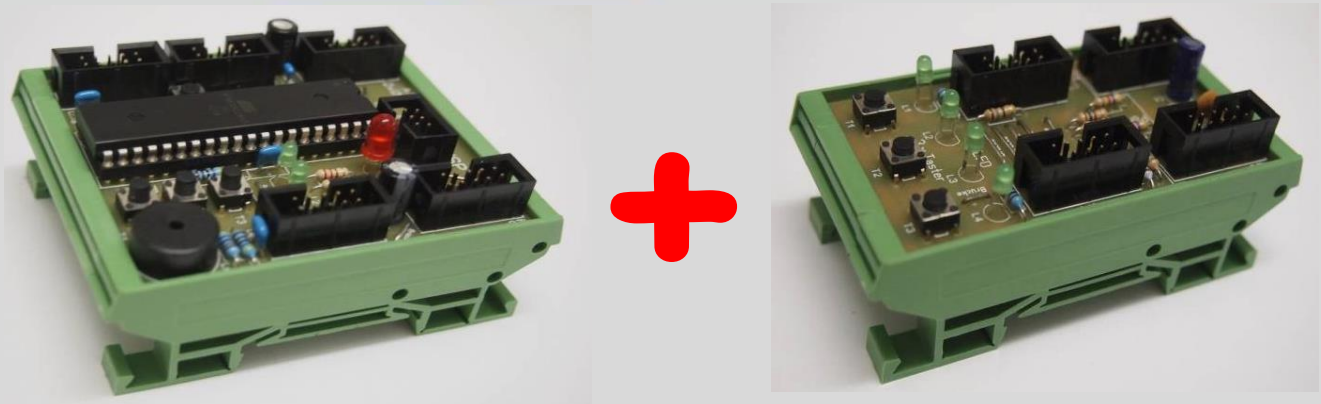
Mikrokontroller & I²C Bus



www.boxtec.ch

playground.boxtec.ch/doku.php/tutorial



Multitasking 5

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 5 (… Kontrolle ist besser)

23. Kontrolle - Warum ?

Wie ich es schon oben geschrieben habe, Vertrauen ist gut, Kontrolle ist besser.

Warum eigentlich?

In den ersten Beispielen war das Zeitraster die Summe aus Laufzeit aller Funktionen/Tasks und dem _delay_ms(1).
In den letzten Beispielen ist das Zeitraster der Hauptschleife exakt, unabhängig von der Laufzeit der Aufgaben, weil der Timer unabhängig eine feste Interrupt Frequenz generiert. Dadurch wird die CPU-Rechenleistung wird 100% in der Abarbeitung der Task verwendet und nicht für nutzlose Warteschleifen verschwendet.
Trotzdem kann es zu Fehlern in der Abarbeitung kommen.

- Die Summe der Funktionen/Task und _delay_ms(1) ist in Wirklichkeit >> 1ms. Dadurch können Zeiten nicht eingehalten werden
- Durch den Timer wird unabhängig eine feste Zeit von 1 ms generiert. Obwohl die erste Zeit noch nicht fertig ist, wird eine zweite Zeit generiert

Im realen System müssen wir prüfen, ob die Laufzeit der Tasks klein genug ist, um den Anforderungen des Timers zu genügen. Diese Überprüfung kann an zwei Stellen durchgeführt werden.

- Am Ende der Hauptschleife nach Abarbeitung aller Ausgaben. Wenn hier die Variable flag_1ms schon wieder aktiv ist, dauerte die Abarbeitung länger als 1ms. Wenn man ein sehr strenges Timing sicherstellen möchte, ist das ein Fehler, der erkannt und signalisiert werden kann
- In der ISR. Wenn hier die Variable immer noch aktiv ist, wurde sie von der Hauptschleife noch nicht erkannt und zurückgesetzt. Das ist definitiv ein Fehler, denn jetzt würde ohne Fehlererkennung ein Timerdurchlauf von der Hauptschleife verschluckt werden. Diese Prüfung ist etwas nachgiebiger, weil zwischenzeitlich ein Durchlauf der Hauptschleife mehr als 1ms, jedoch nicht länger als 2ms dauern darf

24. Timer mit 1ms

Zum Betrieb benötigen wir einen Timer. Dieser erzeugt aus den 16 MHz einen Takt von 1 ms.

```
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
 }
```

Dieser Takt wird durch den Timer vollkommen unabhängig erzeugt.

25. Die ISR

Zum Timer gehört eine ISR. In dieser setzt der Timer den flag_1ms auf 1. Es erfolgt hier kein Rücksetzen auf 0.

```
ISR (TIMER0_COMPA_vect)                   // wait1=1ms,
 {
  flag_1ms=1;
 }
```

Dieser Takt mit 1ms wird vollkommen unabhängig vom restlichen Programm erzeugt und verursacht keine Verzögerung.

26. Das Programm (Version 1)

Bisher könnte das Programm so aussehen:

```
(1)     if (flag_1ms)                   // Abfrage flag_1ms
(2)      {                              // wenn ja, dann ...
(3)        flag_1ms=0;                  // setzt flag_1ms auf 0
(4)        taste = taste_lesen();       // Unterprogramm taste _lesen
(5)        led_blinken(taste);          // Unterprogramm led_blinken
(6)      }
```

In der Zeile 1 wird abgefragt, ob flag_1ms = 1 (vorhanden) ist. Wenn ja, dann werden die Zeilen 3 bis 5 ausgeführt und die Unterprogramme aufgerufen. In der Zeile 3 wird flag_1ms auf 0 gesetzt.
Im nächsten Teil wollen wir testen, ob der flag_1ms bereits wieder gesetzt ist. Dazu wollen wir diese Zeile einfügen:

```
if (flag_1ms)                 // Laufzeit der Tasks >1ms
 {                            // Fehlersignalisierung
  PORTA &= ~(1<<PA6);         // PA6 schalten
  while(1);                   // Programm stoppen
 }
```

Beide Teile zusammen könnten so aussehen:

```
(1)     if (flag_1ms)                   // Abfrage flag_1ms
(2)      {                              // wenn ja, dann ...
(3)        flag_1ms=0;                  // setzt flag_1ms auf 0
(4)        taste = taste_lesen();       // Unterprogramm taste _lesen
(5)        led_blinken(taste);          // Unterprogramm led_blinken
(6)        if (flag_1ms)                // Laufzeit der Tasks >1ms
(7)         {                           // Fehlersignalisierung
(8)           PORTA &= ~(1<<PA6);       // PA6 schalten
(9)           while(1);                 // Programm stoppen
```

```
(10)        }
(11)    }
```

Die Zeilen 1 bis 5 sind identisch. In der Zeile 6 wird flag_1ms wieder abgefragt, ob flag_1ms = 1 (vorhanden) ist. In der Zeile 3 wurde flag_1ms auf 0 gesetzt.
Wenn in der Zeile 6 flag_1ms wieder 1 ist, dann betrug die Laufzeit unserer beiden Unterprogramme mehr als 1ms.

Dieses Programm hat für mich ein paar Nachteile

- Es hat ein sehr strenges Timing
- Auch sehr kurze Überschreitungen lösen ein Programmstop aus
- Keinerlei Toleranz
- Es muss immer als letztes im Programm stehen

Daher werde ich es nicht nutzen.

27. Das Programm (Version 2)

Warum müssen wir die Abfrage extra ins Programm nehmen? Es geht auch einfacher.
Wir nutzen bereits die ISR :

```
ISR (TIMER0_COMPA_vect)                    // wait1=1ms,
 {
  flag_1ms=1;
 }
```

Nehmen wir doch einfach diese Abfrage zusätzlich in die ISR:

```
if (flag_1ms)   Timer Frage
 {              // Laufzeit der Tasks >2ms, Fehlersignalisierung auf PA6, Programm stoppen
  PORTA &= ~(1<<PA6);          // Schaltet Pin
  while(1);                    // Endlosschleife, stoppt Programm
 }
```

Damit könnte unser Programm so aussehen:

```
ISR (TIMER0_COMPA_vect)      // wait1=1ms,
 {
  if (flag_1ms)
   {          // Laufzeit der Tasks >2ms, Fehlersignalisierung auf PA6, Programm stoppen
    PORTA &= ~(1<<PA6);          // Schaltet Pin
    while(1);                    // Endlosschleife, stoppt Programm
   }
  flag_1ms=1;
 }
```

Die Abfrage in der ISR hat mehrere Vorteile:

- Ist unempfindlich gegen kurze Zeitüberschreitungen bis 2 ms
- Ist immer in der ISR und kann dadurch nicht vergessen werden

Nicht vergessen – flag_1ms auf 0 setzen !

```
if (flag_1ms)                 // Abfrage flag_1ms
 {                            // wenn ja, dann …
   flag_1ms=0;                // setzt flag_1ms auf 0
   taste = taste_lesen();     // Unterprogramm taste _lesen
   led_blinken(taste);        // Unterprogramm led_blinken
 }
```

Innerhalb des Hauptprogrammes ist keine Abfrage notwendig.

29. Vollständiges Programm mit Tasterentprellung und Test

```
/* ATB_Multi_14.c Created: 18.03.2015 18:17:52 Author: AS */

// ori Teile P30 und P 31

#define F_CPU 16000000UL          // Angabe der Quarzfrequenz, wichtig für die Zeit
#include <avr/io.h>               // Einbindung Datein
#include <avr/interrupt.h>
#include <stdint.h>
#include <util/delay.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;
   if (flag_1ms)
    {
      // Laufzeit der Tasks >2ms, Fehlersignalisierung PA6, Programm stoppen
      PORTA &= ~(1<<PA6);         // Schaltet Pin
      while(1);                   // Endlosschleife, stoppt Programm
    }
   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(flag_1ms)
     {
       flag_1ms=0;
       led_blinken1();                     // Aufruf Unterprogramm
     }
    ////////////////////////////////        delay zum testen einfügen
    //_delay_ms(2);
    ////////////////////////////////

   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
    }
  }
}
```

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