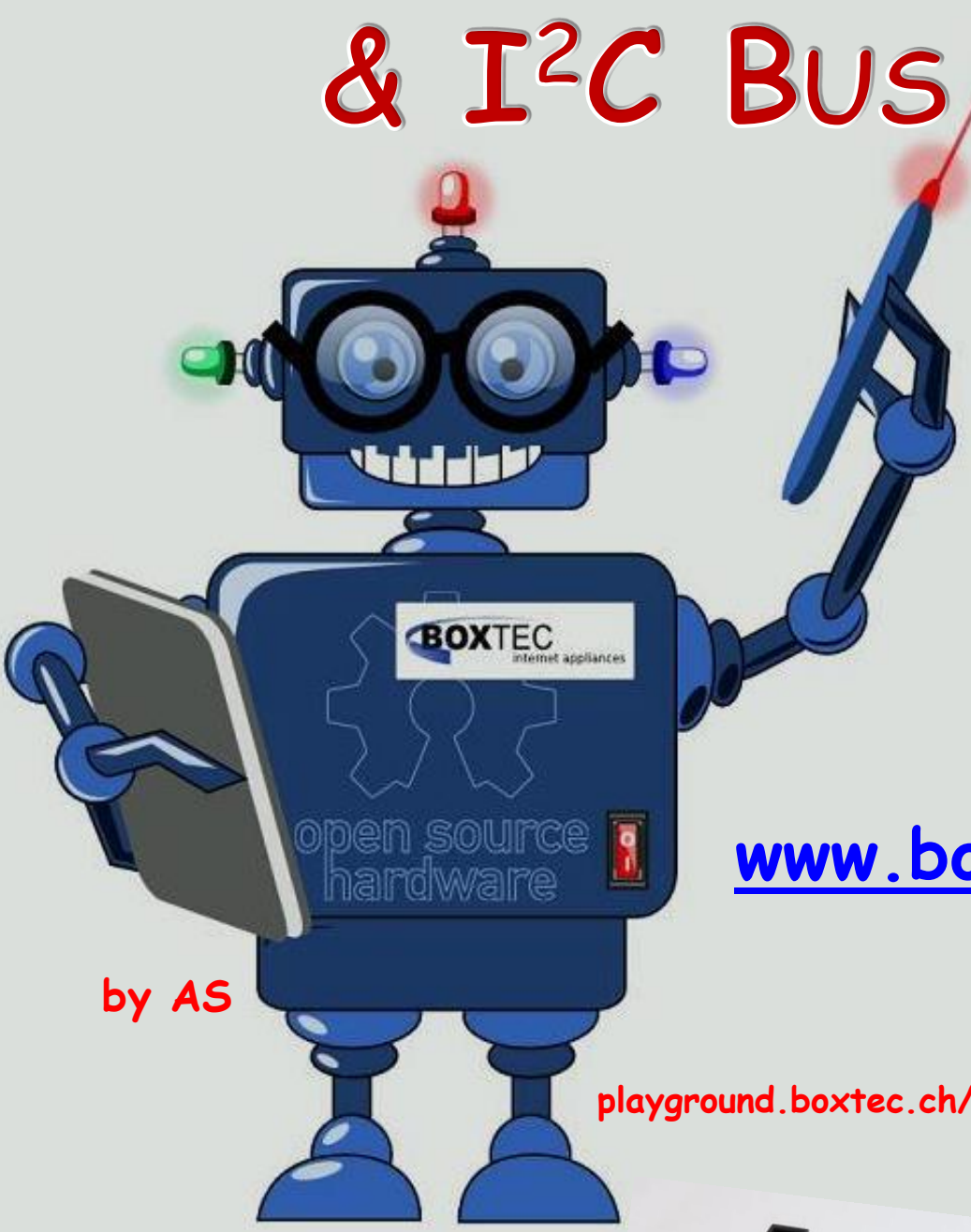
Mikrokontroller & I²C Bus

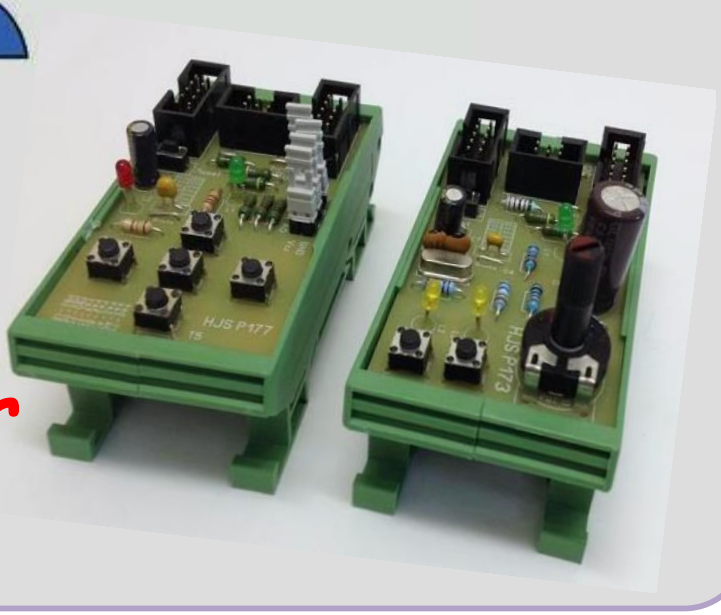


www.boxtec.ch

by AS

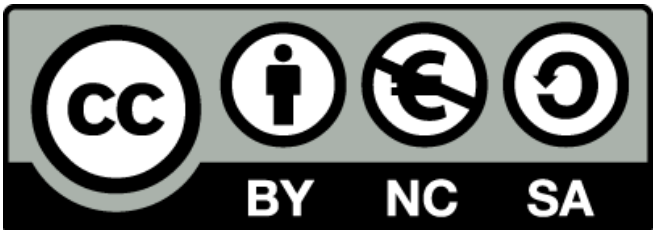
playground.boxtec.ch/doku.php/tutorial

Attiny 841 – ein IC und 5 verschiedene Module
Teil 8 – Taster entprellen



I²C Bus und der Attiny 841

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 Nichtbe-achtung 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 die-ser 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 Versor-gungsspannung 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 Spannungsver-sorgung verwendet werden
- Spannungsführende Kabel oder Leitungen, mit denen die Baugruppe verbunden ist, müssen immer auf Isolationsfehler oder Bruchstellen kontrolliert werden. Bei einem Fehler muss das Gerät un-verzüglich 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 Baugrup-pe 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 Ge-fahren, 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.

Attiny 841 – Taster entprellen

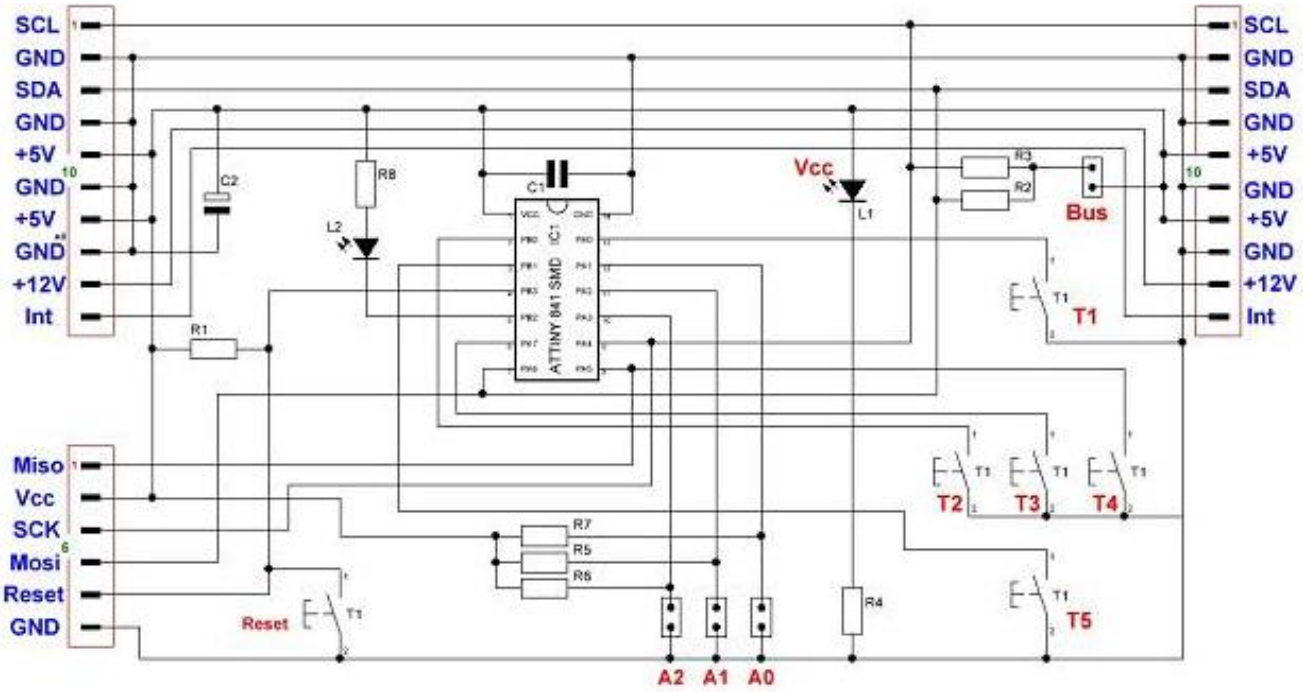
Vor über 10 Jahren hat Peter Dannegger eine Universelle Tasterabfrage veröffentlicht. Das Original kann jeder unter

https://www.mikrocontroller.net/topic/48465?page=single

ansehen. Im Laufe der Zeit wurde einiges angepasst oder Erweitert. In diesem Beitrag werde ich nicht auf die Funktion oder Besonderheiten des Codes eingehen. Im Netz sind viele Beiträge zu finden die über die genaue Funktion Auskunft geben. Im Grunde sind sich alle einig, das Programm ist super in der Funktion und Anwendung.
In diesem Tutorial möchte ich das Programm auf einen Attiny 841 adaptieren und seine Anwendung zeigen. Da der Programmplatz auf einem Attiny 841 relativ gering ist, kommt es besonders auf seine Grösse an. Das Atmel Studio zeigt mir einen Speicherverbrauch von

Program Memory Usage : 394 bytes 4,8 % Full
Data Memory Usage : 6 bytes 1,2 % Full

an. Da bleibt noch einiges für andere Anwendungen frei. Ich werde das Programm auf meiner Platine P177 betreiben. Dazu wieder die Schaltung:



Belegung:

PA0 – Taster T1
PA1 – A0
PA2 – A1
PA3 – A2
PA5 – Taster T4
PA7 – Taster T3
PB0 – Taster T2
PB1 – Taster T5
PB2 – LED 2

Die Pins PA1, PA2 und PA3 sind für eine Angabe von Adressen mit dem I^2C Bus gedacht und werden in diesem Tut nicht verwendet. Am Pin PB2 liegt eine LED und dient zur Anzeige. An den Pins PA0, PB0, PA7, PA5 und PB1 liegen die Taster nach GND (negative Logik) und haben kein Widerstand nach Vcc. Die Widerstände könne im IC zugeschaltet werden.

Als nächste möchte ich euch das eigentliche Programm vorstellen. Wenn ihr eine andere Version dieses Programm aus dem Internet nehmt achtet besonders auf den Timer und die ISR. Es haben sich einige Bezeichnungen geändert und müssen angepasst werden. Den Attiny 841 betreibe ich hierbei ohne Quarz. Bei Verwendung eines Quarzes bitte die Fuses anpassen.

```
/* ATB_Ati841_P177_Prg8.c */

// Programm für Attiny 841 ohne Quarz mit Platine 177
// 5 x Taster als Bedienkreuz ohne Widerstand nach Vcc und Kontroll LED
// 3 x Wahl für Busadresse A0, A1, A2
// mit Tasterentprellung nach Peter Dannegger, mit Timer 10ms

// Fuse Einstellung ohne Quarz
// Ex - 0xFF
// Hi - 0XDF
// Lo - 0xC2

// Funktion und Belegung der Taster                    Anordnung der Taster
// KEY_LINKS_PORTA          Taster 1 - PA0                   T1
// KEY_read_down            Taster 2 - PB0
// KEY_RECHTS_PORTA         Taster 3 - PA7            T2     T3     T4
// KEY_ENTER_PORTA          Taster 4 - PA5
// KEY_read_up              Taster 5 - PB1                   T5

#define F_CPU 8000000UL                  // Angabe Frequenz 8000000
#include "avr/io.h"                      // Einbindung Datein
#include "avr/interrupt.h"
#include "stdint.h"

volatile uint8_t key_state;
volatile uint8_t key_press;
volatile uint8_t key_rpt;

#define KEY_DDRA     DDRA                // Datenrichtung A
#define KEY_PORTA    PORTA               // Angabe Port A
#define KEY_PINA     PINA                // Angabe PIN A

#define KEY_LINKS_PORTA    0             // PA 0 - Taster links    Taster 1
#define KEY_RECHTS_PORTA   7             // PA 7 - Taster rechts   Taster 3
#define KEY_ENTER_PORTA    5             // PA 5 - Taster enter    Taster 4

#define KEY_DDRB     DDRB                // Datenrichtung B
#define KEY_PORTB    PORTB               // Angabe Port B
#define KEY_PINB     PINB                // Angabe PIN B

#define KEY_DOWN_PORTB 0                 // PB 0 - Taster down    Taster 2
#define KEY_UP_PORTB   1                 // PB 1 - Taster up      Taster 5
#define KEY_read_links   0               // wie Port Hilfsbyte Bit 0 - Taster links
#define KEY_read_rechts  7               // wie Port Hilfsbyte Bit 7 - Taster rechts
#define KEY_read_enter   5               // wie Port Hilfsbyte Bit 5 - Taster enter
#define KEY_read_down    1               // nach shiften Hilfsbyte Bit 1-Taster down
#define KEY_read_up      2               // nach shiften Hilfsbyte Bit 2-Taster up
```

```
#define ALL_KEYS_PA (1<<KEY_LINKS_PORTA|1<<KEY_RECHTS_PORTA|
        1<<KEY_ENTER_PORTA)
#define ALL_KEYS_PB (1<<KEY_DOWN_PORTB|1<<KEY_UP_PORTB)

#define REPEAT_MASK (1<<KEY_read_links|1<<KEY_read_rechts|1<<KEY_read_enter|
        1<<KEY_read_down| 1<<KEY_read_up)
#define REPEAT_START 50                 // after 500ms
#define REPEAT_NEXT 20                  // every 200ms

ISR (TIMER1_COMPA_vect)                 // Timer 10ms
 {
  static uint8_t ct0,ct1,rpt;
  uint8_t i;
  uint8_t k;
  TCNT0 = (uint8_t)(int16_t)-(F_CPU / 1024 * 10e-3 + 0.5);
  k = PINA & (1<<PINA0 | 1<<PINA5 | 1<<PINA7);       // einlesen PORTA unveraendert
  i = PINB &(1 << PINB0|1 << PINB1);      // einlesen PORTB unveraendert
  i = i << 1;                             // schiebe das von PORTB eingelesene um 1 nach
  k = i | k;                              // verodern von i (PORTB geschoben) und PORTA
  i = key_state ^ ~k;               // i = key_state ^ ~KEY_PIN; ->
  ct0=~(ct0&i);                     // i = key_state ^ KEY_PIN; Taster positiv oder negativ
  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 timer1_init()                      // Timer 1, 16 Bit, 8MHz, 10ms
 {                                      // Timer 1 konfigurieren
  TCCR1A = (1<<WGM01);                  // Auswahl CTC Modus
  TCCR1B = (1<<CS01)|(1<<CS00);         // Prescaler auf 64 setzen
  OCR1A=1249;                           // Wert für 10ms
  TIMSK1|=(1<<OCIE1A);                  // Interrupt erlauben
 }
int main(void)
 {                                      // Programmschleife main
  timer1_init();                        // Timer 1 init
  PUEA=0b10100001;                      // R auf Taster PORT A
  PUEB=0b00000011;                      // R auf Taster PORT B
  DDRB=0b00000100;                      // Port B auf Ausgang schalten
  PORTB = (1<<PINB2);
  KEY_DDRA&=~ALL_KEYS_PA;
  KEY_PORTA|=ALL_KEYS_PA;
  KEY_DDRB&=~ALL_KEYS_PB;
  KEY_PORTB|=ALL_KEYS_PB;
  sei();

  // Auswahl der Taster
  // if(get_key_press(1<<KEY_LINKS_PORTA))      // Taster 1
  // if(get_key_press(1<<KEY_read_down))        // Taster 2
  // if(get_key_press(1<<KEY_RECHTS_PORTA))     // Taster 3
  // if(get_key_press(1<<KEY_ENTER_PORTA))      // Taster 4
  // if(get_key_press(1<<KEY_read_up))          // Taster 5

  while(1)
   { // Mögliche Taster und Funktionen, Beispiel mit Taster 1
    if(get_key_press(1<<KEY_LINKS_PORTA))      // Schaltet bei kurzem drücken
    // if(get_key_rpt(1<<KEY_LINKS_PORTA))     // Schaltet press aus (länger drücken)
    // if(get_key_short(1<<KEY_LINKS_PORTA))   // Bei kurzem drücken einschalte Kombi
                                               mit long
    // if(get_key_long(1<<KEY_LINKS_PORTA))    // Bei langem drücken ausschalten Kombi
                                               mit short
```

```
      {
        PORTB &= ~(1<<PINB2);                  // LED einschalten
        // PORTB ^= (1<<PINB2);                // LED toggelt, Mögliche Version für press
      }
     // Mögliche Taster und Funktionen, Beispiel mit Taster 2
     // if(get_key_press(1<<KEY_read_down))  // Schaltet bei kurzem drücken
     if(get_key_rpt(1<<KEY_read_down))       // Schaltet press aus (länger drücken)
     // if(get_key_short(1<<KEY_read_down)) // Bei kurzem drücken einschalten Kombi mit
                                             long
     // if(get_key_long(1<<KEY_read_down))  // Bei langem drücken ausschalten Kombi mit
                                             short
      {
        PORTB |= (1<<PINB2);                 // LED ausschalten
      }
   }                                         // Ende while
}                                            // Ende main
```

Im Programm habe ich viele Kommentare angegeben. Insbesondere für die Verwendung der Taster und deren Zuordnung. Innerhalb der ISR habe ich die Änderung bei Verwendung einer positiven Taster Logik ebenfalls angegeben. Im Programm selber verwende ich nur die Taster T1 und T2. Unterschiedliche Funktionen der Taster habe ich auskommentiert.

Welche unterschiedliche Kombinationen und Funktionen sind denn möglich?

Es gibt insgesamt 4 verschiedene Funktionen:

- get_key_press()
- get_key_rpt()
- get_key_short()
- get_key_long_r()

Jede dieser 4 Funktionen hat dabei die folgende Aufgabe:

- get_key_press() - schaltet die LED beim Drücken ein. Jeder Druck wird nur einmal gemeldet
- get_key_rpt() – schaltet die LED beim Drücken aus. Reagiert nicht auf kurzes drücken
- get_key_short() - schaltet die LED ein bei kurzem drücken
- get_key_long() - schaltet LED bei langem drücken aus

Die Funktionen

get_key_press und get_key_rpt gehören zusammen und
get_key_short und get_key_long gehören zusammen.

Alle get_.... Funktionen liefern dir die Info ob eine Taste niedergedrückt wurde. Im Falle von get_key_short und get_key_long erst dann, wenn die Taste wieder losgelassen wird.

Auch in diesem Programm sind Verbesserungen möglich. Als Timer verwende ich den Timer 1 und dieser arbeitet mit 16 Bit. Ist eigentlich eine Verschwendung. Besser dazu geeignet ist der Timer 0 mit 8 Bit. In der ISR wird aber eine Zeit zwischen ca. 5ms und 50ms angegeben. Diese Zeit ist aber mit dem Timer 0 schlecht zu erreichen. Der Timer 1 mit 16 Bit wird besser für PWM verwendet. Für zukünftige Anwendungen ist es besser **Time Slots** zu verwenden.
Das bedeutet, dass ein einheitlicher Timer für allgemeine Anwendungen verwendet wird und die

notwendigen Zeiten „zusammengebaut" werden. Sehen wir uns den Timer 0 dazu an:

```
void timer0_init()                          // 8MHz, 8 Bit, 1ms
 {                                          // Timer 0 konfigurieren
   TCCR0A = (1<<WGM01);                     // Auswahl CTC Modus
   TCCR0B = (1<<CS01)|(1<<CS00);            // Prescaler auf 64 setzen
   OCR0A=124;                               // Wert für 1ms
   TIMSK0|=(1<<OCIE0A);                     // Interrupt erlauben
 }
```

Mit dem Timer 0 erzeuge ich hierbei einen 1ms Impuls. Um ihn in der ISR zu verwenden muss er verzögert werden auf 10ms. Das mache ich mit diesem Zähler.

```
ISR (TIMER0_COMPA_vect)                     // Timer 1ms
 {
   millisekunden++;                         // Zähler für 10ms
   if(millisekunden == 9)
    {
        static uint8_t ct0,ct1,rpt;
        ......
        millisekunden = 0;
    }
 }
```

Ich muss millisekunden als Variable angeben. Innerhalb der der Klammer { … } mit static uint8_t ct0,ct1,rpt; ist dasselbe Programm wie oben angegeben. Am Ende muss millisekunden wieder auf 0 gesetzt werden, damit der Timer auch beim nächsten Aufruf funktioniert.

Bitte die Einstellung kontrollieren:

Fuse Einstellung ohne Quarz

Ex - 0xFF
Hi - 0XDF
Lo - 0xC2

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