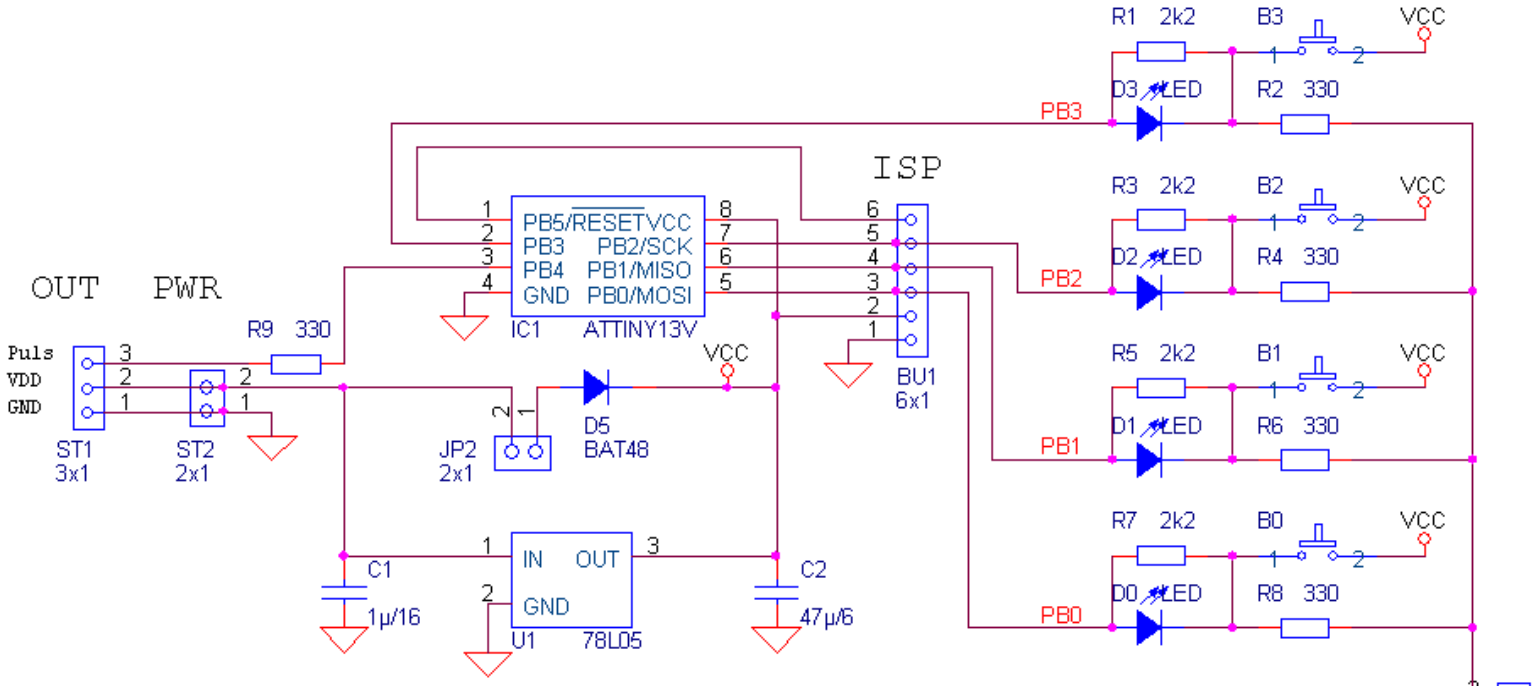
B3
R1
R2
D3
IC1
C2
B2
D2
R3
R4
B1
D1
R5
R6
BU1
B0
D0
R7
R8
U1
D5
ST1
ST2
R9
C1
JP2
JP1

Circuit

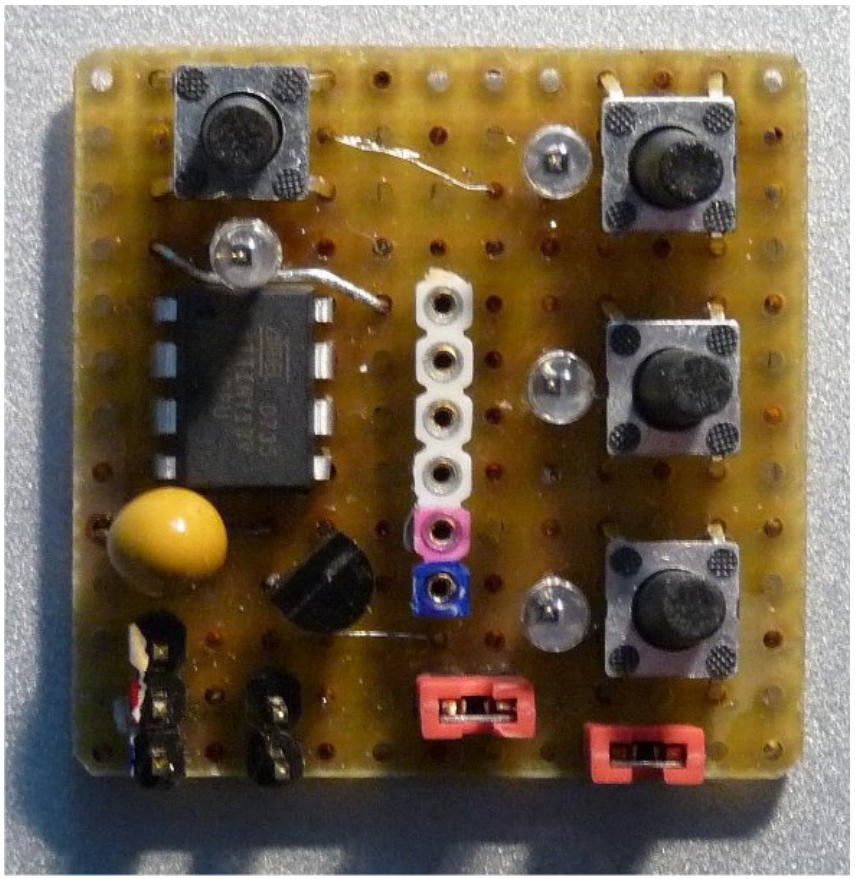
- Open JP1 while programming
- JP2: open break for 7,4 Volt
- ST2: power input
- ST1: puls-output

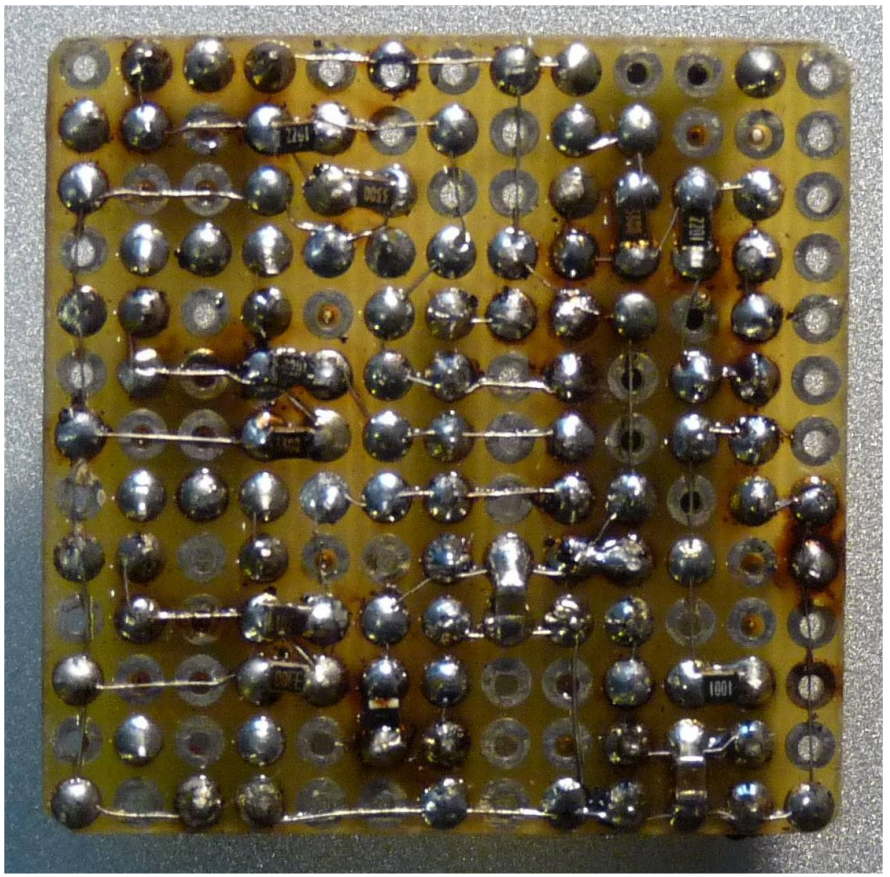


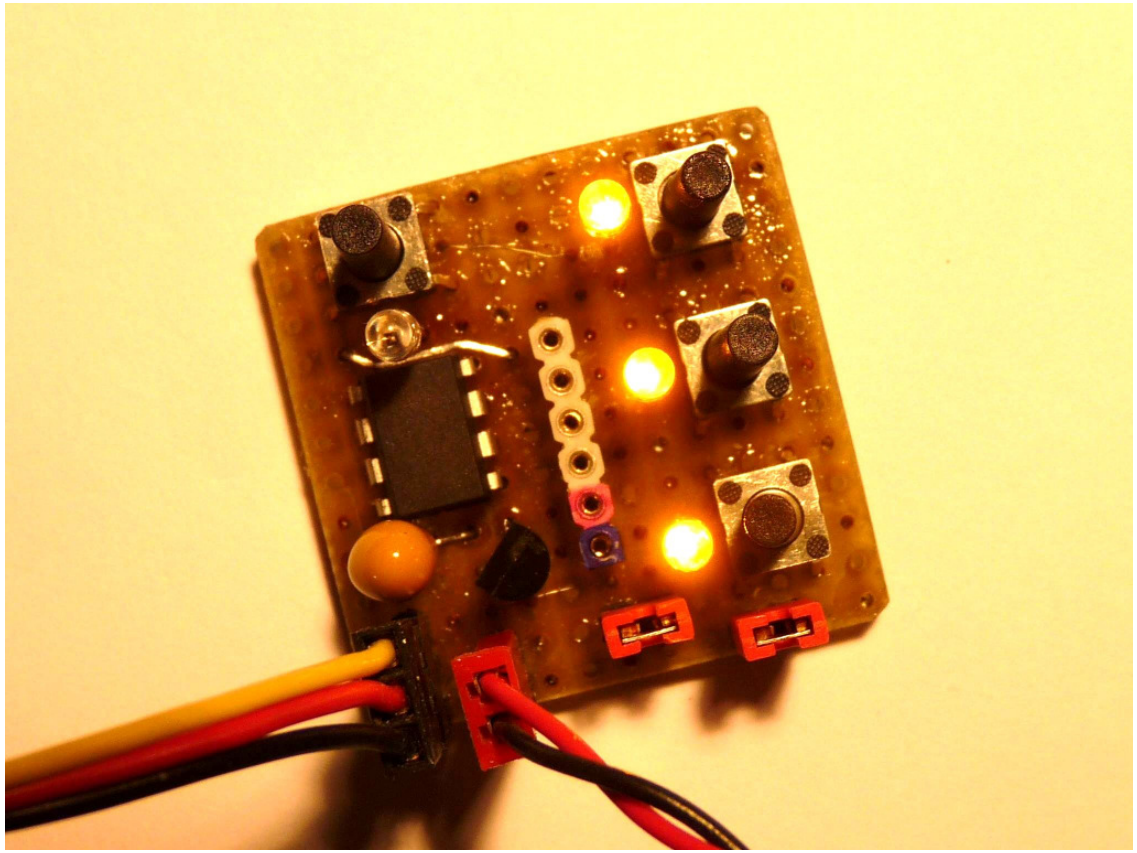
ST1: Pulsout
ST2: Power-In (5...10V)
JP1: Programmieren: off
JP2: Power: 5V: on, >5V: off
BU1: AVR-ISP Atmel

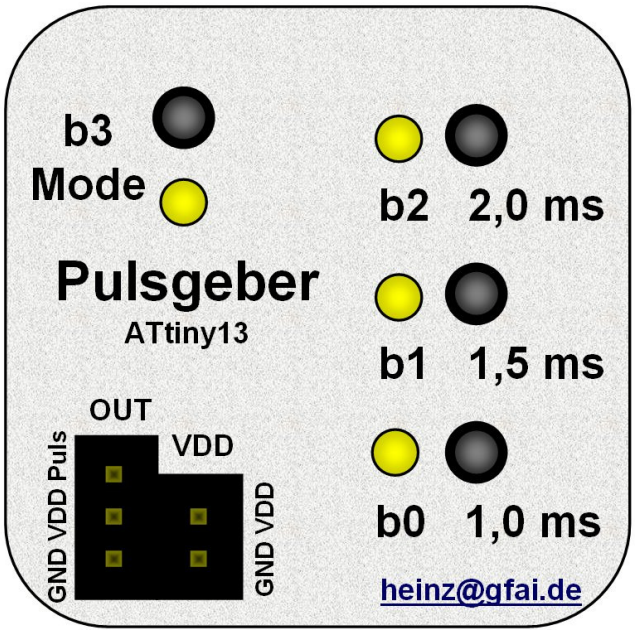
Funktion:
B0: Puls 1ms
B1: Tuls 1.5ms
B2: Puls 2ms
B3: Vari-Mode 1ms-2ms-1ms-2ms
Pause 1ms-0.9ms-0.8ms...0.1ms

Layout








b3
Mode
b2 2,0 ms
Pulsgeber
ATtiny13
b1 1,5 ms
OUT
VDD
GND VDD Puls
GND VDD
b0 1,0 ms
heinz@gfai.de


b3
Mode
b2 2,0 ms
Pulsgeber
b1 1,5 ms
OUT
VDD
GND VDD Puls
GND VDD
b0 1,0 ms
heinz@gfai.de

```
; Servotester
; Board mit Atmel ATtiny13
; dr.g.heinz@web.de     08/2010
; Entwickelt auf AVR-Studio 4.15.6.23

/*
Funktion
Der Servotester hat vier Tasten und vier zugeordnete LEDs an PB0...3.
An PB4 wird der Servopuls (1...2ms) ausgegeben.

Tasten/LEDs PB0...2 sind für manuelle Vorgabe der Pulslänge da:
 0: 1,0 ms
 1: 1,5 ms
 2: 2,0 ms

Taste 3 startet den Automatikmode, dabei wird zwischen Endlagen
des Servos schnell hin- und hergeschaltet (1ms -> 2ms -> 1ms ->...)

Die Zeitdauer ist über Mode einstellbar.

Am verminderten Servoausschlag läßt sich erkennen, wann die
Geschwindigkeitsgrenze des Servos erreicht ist.

Mit Taste PB3 (Mode) wird die Umschaltzeit vorgegeben
(Start mit 1 Sekunde, Ende 0,1 Sekunde):
Einmal drücken -> Pulsdauer 1,02s
Zweimal drücken -> 0,9 s
...
Beim schritte-maligen Druck Rücksetzen auf Handbetrieb
Die Anzahl der Tastendrücke wird binär auf LED0...3 blinkend angezeigt
*/
;
;   Mode Handbetrieb (startup):
;       BTN0    1 ms    LED0 an
;       BTN1    1,5 ms  LED1 an
;       BTN2    2 ms    LED2 an
;       BTN3    n-mal drücken für auto-hin/zurück
;               Binärwert wird auf LEDs 0...3 blinkend angezeigt
;
; Automatikmode (hin/zurück):
;   Dauer = schritte * 4 * 25,5 ms =  schritte * 102 ms
; Beispiel:
;   Anzeige binär 0x0a (zehn): Dauer = 10*4*25,5ms = 1,02 sec

.equ schritte = 0x0b    ; Maximale Zahl von Tastendrücken bei auto plus Eins

; Prozessor: ATMEL ATtiny13A mit 5V und Fuse 9,6MHz interner RC-Osz.
.include "tn13def.inc"  ; Prozessordaten

;
; Pinout 8-PDIP/SOIC:
;                 --------
;   /RESET  PB5 |1  `´  8| VCC
;           PB3 |2      7| PB2  SCK
;           PB4 |3      6| PB1  MISO
;           GND |4      5| PB0  MOSI
;                 --------
; Ports:
;   PB0:    LED0    BTN0    1.0 ms
;   PB1:    LED1    BTN0    1.5 ms
;   PB2:    LED2    BTN0    2.0 ms
;   PB3:    LED3    BTN3    Mode
;   PB4:    Ausgabe Servopuls
;
;   Beschaltung Ports 0 bis 3 (4x LED/Button an PB0...3):
;
;           LED         ___330
;   PBx>--o---|>|---o---___-----| GND
;         |   ___    |
;         +---___---o---o___o---+ +5V
;               2k2       BTN
```

```
;
;   BTNx ziehen auf +5V - aktiv high !!!
;   LEDx leuchten bei 1 - aktiv high !!!

;   Test von LED und Button:    PBx auf H -> LED leuchtet
;                               BTNx drücken -> LED aus

; Timing:
;   Puls variabel 1...2ms, Pause 25,5ms fix
;   PWM-Timer: clocks laufen von 10 bis 20 (mal 0,1 sec)
;   Pause 25,5 ms anhängen
;   OCR0A erzeugt Puls  10...20
;   Timer-Overflow erzeugt 255 ~> 25,5 ms Pause


; Ziel Pulsdauer (0,1000 ms) für clocks:
;   interner clock  104,166 ns ~ 9,6 MHz (RC-Oszillator)
;   Vorteiler 1024: 104,166ns x 1024 = 106,66 µs -> zu lang:
;
; Grundtakt 0,100 ms erzeugen:
;   Oscillator von 9,6 auf 10,240 MHz setzen
;   dazu Oszillator höher trimmen
;
; Frequenzerhöhung um  106,666 % von 9,6 auf 10,24 MHz
; dazu oscal zunächst aus ATtiny auslesen, dann soft überschreiben:
;
;       MHz     9,6     10,24       Typ
; oscal         0x57    0x5c        ATtiny13A
; oscal         0x68    0x6e        ATtiny13
.equ oscal = 0x6e   ; oscillator calibration (maximum frequency 0x7f)
; nachmessen und oscal ggf. leicht korrigieren
;
; Stellbereich 1,0 ... 1,5 ... 2,0 ms:
.equ unten = 10 ;       1,0 ms
.equ mitte = 15 ;       1,5 ms
.equ oben  = 20 ;       2,0 ms

.equ led0 = 0x01            ;   LED0 einschalten
.equ led1 = 0x02            ;   LED1 einschalten
.equ led2 = 0x04            ;   LED2 einschalten
.equ led3 = 0x08            ;   LED3 einschalten

.equ maxcount = 4   ; Delay für einen Zyklus (mal 25,5 ms)

; Set prescaler TCCR0B 2:0 = CS0 2:0
        ;   /1:     001     1
        ;   /8:     010     2
        ;   /64:    011     3
        ;   /256:   100     4
        ;   /1024:  101     5

.equ prescal = 5    ; prescal=5: 10,24 MHz /1024 = 10000 Hz ~ 0,100 ms
                    ; für Simulation = 1 setzen

; Register dynamisch
.def    A1  = r16   ; Arbeit
.def    A2  = r17   ; Arbeit

; Register statisch
.def dauer  = r18   ; Vorgabewert Anzahl der Pausenzyklen
.def auto   = r19   ; Zähler-Register für Automatic-Mode
.def leds   = r20   ; LED-Ausgabe
.def zyklus = r21   ; Pausenzähler bei Automatik-hoch/runter
.def clocks = r22   ; Anzahl Takte für Timer Interrupt
.def schalt = r23   ; Umschaltregister auto hoch/runter
.def butn   = r24   ; Tastenentprellung
.def buta   = r25   ; Tastenentprellung
.def butb   = r26   ; Tastenentprellung
.def btnmod = r27   ; Tastenspeicher (0x00...0x03)


; ############ Inits ########################################
```

```
rjmp Reset

; Interrupt-Adresse reservieren (Compiler-Anweisung)
;################ Adressen ################

;   0x0000 rjmp RESET       ; Reset Handler
;   0x0001 rjmp EXT_INT0    ; IRQ0 Handler
;   0x0002 rjmp PCINT0      ; PCINT0 Handler
;   0x0003 rjmp TIM0_OVF    ; Timer0 Overflow
;   0x0004 rjmp EE_RDY      ; EEPROM Ready Handler
;   0x0005 rjmp ANA_COMP    ; Analog Comparator Handler
;   0x0006 rjmp TIM0_COMPA  ; Timer0 Compare A
;   0x0007 rjmp TIM0_COMPB  ; Timer0 Compare B
;   0x0008 rjmp WATCHDOG    ; Watchdog Interrupt Handler
;   0x0009 rjmp ADC         ; ADC Conversion Handler
                            ; siehe Tab.9-1, S.44

; Timer Overflow S.44
.org 0x0003 rjmp Timer_overflow     ; Pausenende erreicht

; Timer/Counter Compare Match A, S.44
.org 0x0006 rjmp Timer_compa        ; Pulsende erreicht

.cseg   ; Codesegment folgt

Reset:
; Stack setzen:
    ldi A1,low(RAMEND)      ; Set Stack Pointer to top of RAM
    out SPL,A1              ; (SPH gibt es beim ATtiny13 nicht)


rjmp Main       ; zum Hauptprogramm

;#############################################################
;######################## Unterprogramme #####################
;#############################################################


Init_Timerinterrupts:       ; Timer-Interrupt alle 0,1 ms = 100 µs
                        ; Timer-Interrupt Init compare match A
                        ; siehe S.60, 11.5

  ; Vorteiler einstellen, S.73, Tab.11-9

    ldi A1,prescal      ; Timer-Prescaler laden (CS0)
    out TCCR0B,A1       ;

  ; Fast PWM mode einstellen:

    ; S.64, 11.7.3;  13A: S.73, 11-8
    ; WGM2:0 = 011 dazu WGM01:0 = 11 setzen
    ldi A1,0x03         ; 0b0000_0011, Seite 72, Tab. 11-8
    out TCCR0A,A1

  ; Zeit einstellen für Pulslänge:

    ; Zeitdauer für Interrupt in Timer-Register OCR0A
    ; nach Anzahl "clocks" Interrupt auslösen
    ldi clocks,mitte    ; 0x0e = dez.15 ~ 1,5 ms (neutral)
    out OCR0A,clocks    ; Timer-Register A laden

  ; Interrupts maskieren:
    ; Timer comp-A Interrupt OCIE0A freigeben (Puls-Ende)
    ; Timer Overflow Interrupt TOIE0 freigeben (Pausen-Ende)
    in    A1,TIMSK0             ; read
    sbr   A1,(1<<OCIE0A)+(1<<TOIE0) ;(use masks, not bit#)
    ;+(1<<OCIE0B)
    out   TIMSK0,A1             ; write back

    ; so gehts nicht:
    ; Timer comp-A Interrupt OCIE0A freigeben (Puls-Ende)
    ;sbi A1,OCIE0A      ;
    ;out TIMSK0,A1      ; Timer Interrupt freigeben S.64, S.74
```

```
    ; Timer Overflow Interrupt TOIE0 freigeben (Pausenende)
    ;sbi A1,TOIE0       ;
    ;out TIMSK0,A1      ; Timer Interrupt freigeben S.64, S.74

ret

Transfer:   ; i: clocks, leds,   o:-
    ; aktuelle Parameter an LEDs und Timer übergeben
    rcall Pulsdauer
    rcall Portwrite
ret


Pulsdauer:  ; i:clocks
            ; Pulsdauer in Timer-Register schreiben
            ; nach Anzahl "clocks" Interrupt auslösen
            ; Zeitdauer für Interrupt in Timer-Register OCR0A
    out OCR0A,clocks    ; Timer-Register A laden
ret


Portwrite:  ; i:leds o:PBx
            ; Datenrichtung S.51    0: input, 1: output
    ldi A1,0b_1111_1111 ; PB7...0 auf out
    out DDRB,A1         ; PB4 auf output, PB0...3 output
    out PORTB,leds      ; Register leds auf LEDs ausgeben
ret


Portread:   ; i:PBx o:butn
    ; Datenrichtung S.51    0: input, 1: output
    ldi A1,0b_1111_0000 ; PB7...0
    out DDRB,A1         ; PB0...3 input
    nop                 ; nötig, um Zeit für Umladung zu gewinnen
    nop
    nop
    nop
    in butn,PINB        ; read pinb, auf butn kopieren
    nop                 ; high, wenn Taste gedrückt
    nop
    ; zurücksetzen auf out:
    ldi A1,0b_1111_1111 ; PB7...0, PB4 pulsout
    out DDRB,A1         ; PB0...3 output
ret


Buttonread: ; i:butn,   o:btnmod
    ; Entprellen der Tasten über drei Zyklen a 25,5 ms

    ; über zwei Timer-Zyklen schieben:
    mov butb,buta       ; schieben
    mov buta,butn       ; nochmal schieben

    ; neuen Wert für butn holen
    rcall Portread  ; Port einlesen auf Register butn

    ; Test, ob alle gleich sind:

    mov A1,butn     ; Wert landet auf erstem Operanden A1
    eor A1,buta     ; EXOR: bei Gleichheit wird Zero-Flag gesetzt (Z=1)
    eor A1,butb     ; EXOR
    breq buttzero   ; Zero-Flag gesetzt, alle sind gleich
    ; EXOR-Ausgang ist 1, wenn die Eingänge verschieden sind
    ret             ; raus, weil ungleich

    buttzero:       ; Wert erkannt?
    ; hier sind alle gleich
    ; sind sie auch von null verschieden?
    tst butb        ; verschieden von null?
    brne buttwei    ; nicht Null
    ret             ; raus, weil null

    buttwei:        ; butb ist von null verschieden
```

```
    ; wir setzen buta zurück, damit Tastendruck nicht nochmal ausgewertet wird
    ldi buta,0

    ; wir können den Wert akzeptieren
    mov btnmod,butb     ; Taste sichern
    mov leds,btnmod     ; und anzeigen
    rcall Transfer

    ; wir können den Key untersuchen
    rcall Chbutton  ; Mode auswerten
ret


Chbutton:   ; button has changed, auswerten
    ;   i:btnmod,   o: leds,clocks,auto

    cpi btnmod,led0 ; Vergleich mit led0
    breq chunten    ; butn0 was pressed (unten)

    cpi btnmod,led1
    breq chmitte    ; buta was pressed (mitte)

    cpi btnmod,led2
    breq choben     ; butb was pressed (oben)

    cpi btnmod,led3
    breq chmode     ; butn3 was pressed (mode)

    ret

    ; Puls-Parameter und LEDs setzen
    chunten:
    ldi clocks,unten    ; unten
    rjmp chende

    chmitte:
    ldi clocks,mitte    ; Mittelstellung
    rjmp chende

    choben:
    ldi clocks,oben     ; oben
    rjmp chende

    chmode:             ; hoch/runter-Automatik
    rcall Modechanged   ; Automatic Mode untersuchen
    ret

    chende:             ; reset auf Handbetrieb
    mov leds,btnmod     ; btnmod anzeigen
    ldi auto,schritte   ; rücksetzen auf Handbetrieb
    rcall Transfer      ; clock und leds aktualisieren

ret


Modechanged:    ; Mode has changed - wenn butn3 - MODE - gedrückt wurde
                ; i: btnmod     o: zyklus, auto, leds, clocks

    ldi zyklus,0    ; Zyklenzähler rücksetzen, von vorn beginnen

    ; dauer = auto * 4;      Dauer = Anz. der PB3-Drücke mal 102ms
    mov dauer,auto      ; dauer berechnen
    lsl dauer           ; mit zwei multiplizieren
    lsl dauer           ; mit zwei multiplizieren

    ; Anzeige auf leds erzeugen
    mov leds,auto       ; Binärwert auto nach leds kopieren
    rcall Transfer      ; Werte übertragen

    ; auto soll nicht Null werden
    subi auto,1         ; auto minus eins
    cpi auto,0          ; ist auto abgelaufen? Reset auf Handbetrieb
    breq modechend      ; ja, Ende
```

```
    ret                 ; hier gehts raus, wenn auto ungleich null

    ; Rückschaltung auf Handbetrieb:
    modechend:
    ldi auto,schritte   ; wenn auto kleiner/gleich Null war, neu laden

    ; Reset auf Handbetrieb:
    ldi leds,led1       ; led1 einschalten
    ldi clocks,mitte    ; in Neutralstellung gehen
    rcall Transfer      ; Werte übertragen

ret


Automatic:  ; Automatikbetrieb hoch/runter durchführen
            ; i: auto, zyklus       o: leds, clocks
    cpi auto,schritte   ; testen (Handbetrieb)
    brne autowei
    ret                 ; raus, falls

    autowei:
    ldi A1,1
    add zyklus,A1       ; Zyklenzähler erhöhen

    cp dauer,zyklus     ; Zyklus Ende erreicht?
    brne autoende       ; nein

    ; neuer Zyklus
    ldi zyklus,0        ; Reset Zyklenzähler, von vorn

    ; hoch/runter-Umschaltung
    subi schalt,1       ; schalt speichert nur das hoch/runter-Bit

    ; odd/even für hoch/runter
    sbrs schalt,0       ; Test Bit0 gerade oder ungerade
    rjmp autonocar      ;

    ; gerade
    ldi clocks,oben     ; hoch an Timer übergeben
    mov leds,auto       ; LEDs einschalten
    rcall Transfer      ; Werte übertragen
    ret

    ; ungerade
    autonocar:
    ldi clocks,unten    ; runter an Timer übergeben
    ldi leds,0          ; LEDs ausschalten
    rcall Transfer      ; Werte übertragen
    ret

    autoende:
ret




;*********************************************
;           Interrupt-Behandlung
;*********************************************

; Puls/Pause erzeugen

Timer_compa:        ; Timer Interrupt bei compare match A
                    ; Pulsende und Pausenbeginn
    cbi PORTB,4         ; Puls auf PB4 rücksetzen
    rcall Buttonread    ; Knopfdruck abfragen
    rcall Automatic     ; Automatikbetrieb abfragen
reti


Timer_overflow:     ; Timer-Overflow erkannt (Pausenende)
                    ; Pausenende und Pulsbeginn
```

```
    sbi PORTB,4     ; Puls auf PB4 setzen
reti



;***************************************************************
;************************ MAIN *********************************
;***************************************************************


Main:

; Inits
    ldi A1, oscal       ; Osc. calibration byte lesen
    out osccal,A1       ; in calibration register ablegen

    ldi auto,0x00       ; set Handbetrieb, Bitposition 0 = 0x01

    ldi butb,0      ; Tastenentprellung reset

    ldi btnmod,led1 ; Button 1 init
    ldi leds,led1       ; LED PB1 an, Neutralstellung
    ldi butn,led1       ; butn init irgendwas
    ldi clocks,mitte    ; Neutralstellung
    rcall Transfer      ; Werte übertragen

    rcall Init_Timerinterrupts
    sei                 ; Interrupt freigeben

mainloop:           ; auf Timer-Interrupt warten
rjmp mainloop       ; Rest findet in Timer_Interrupts statt
```