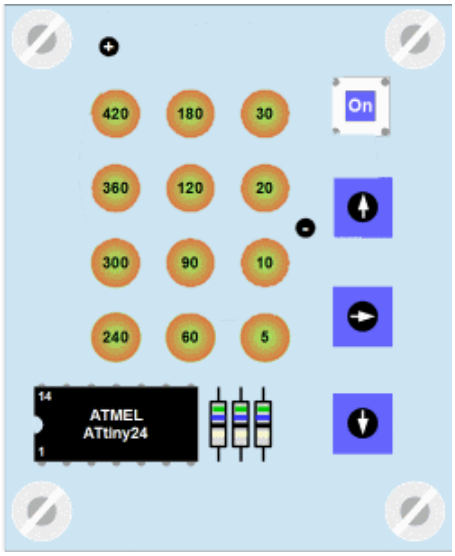


AVR-Anwendungen

Vielfachtimer mit ATtiny24

Vielfachtimer mit ATtiny24 und 12 LEDs

Hier wird ein Timer beschrieben, mit dem Zeiten zwischen 5 Sekunden und sieben Minuten in zwölf Stufen ausgewählt werden können (grüne LEDs) und auf Knopfdruck gestartet werden (rote LEDs). Auswahl und Start/Stop des Timers werden mit drei Tasten kontrolliert.

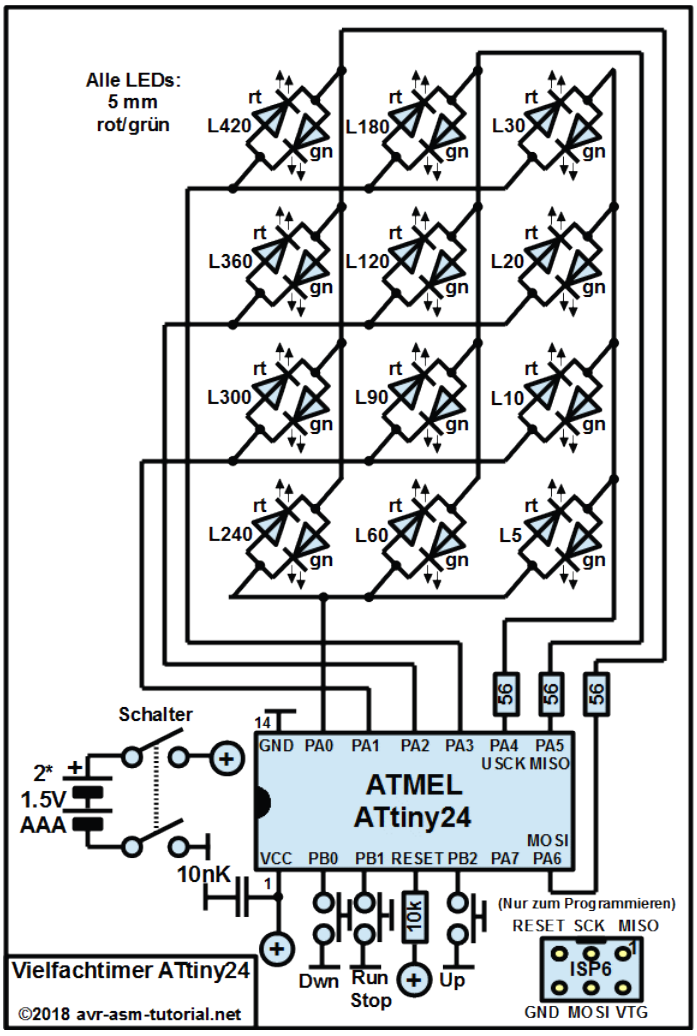
Abspielen eines Videos.

Seitenanfang | Hardware | Aufbau | Software

1 Hardware

Das ist es:

- Ein ATtiny24 kümmert sich um das Timing, liest die Schalter schaltet die LEDs.
- 12 rot/grüne Duo-LEDs zeigen den Zustand an und sind in einer 4-mal-drei-Matrix an den Port A des Controllers angeschlossen.
- Drei Taster sind an Port B des Controllers angeschlossen und Schalten die mit den internen Pull-Up-Widerständen auf Eins gezogenen Eingänge auf Null.
- Das Gerät wird mit zwei AAA-Batterien mit 3 V betrieben.



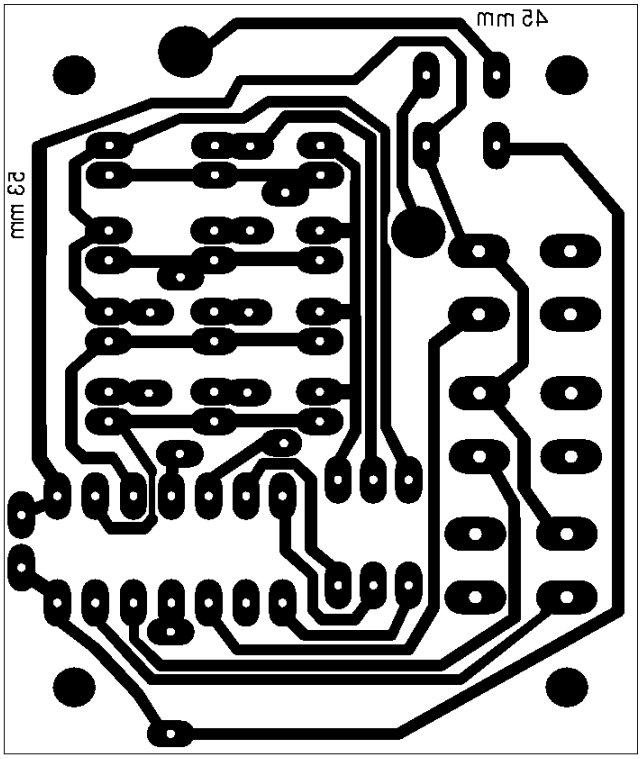
Die LED-Ströme in dieser Schaltung wurden gemessen und liegen zwischen 10 mA für eine grüne LED und 12 mA für eine rote LED. Das reicht unter normalen Umständen völlig aus. Durch den Blinkmodus wird der durchschnittliche Strombedarf zusätzlich halbiert. Der Be-

triebsstrom für den Controller liegt bei 0,1 mA, weil er interrupt-gesteuert angesteuert wird und die überwiegende Betriebszeit im Schlafmodus auf Timer- und Tasten-Ereignisse wartet.

2 Aufbau

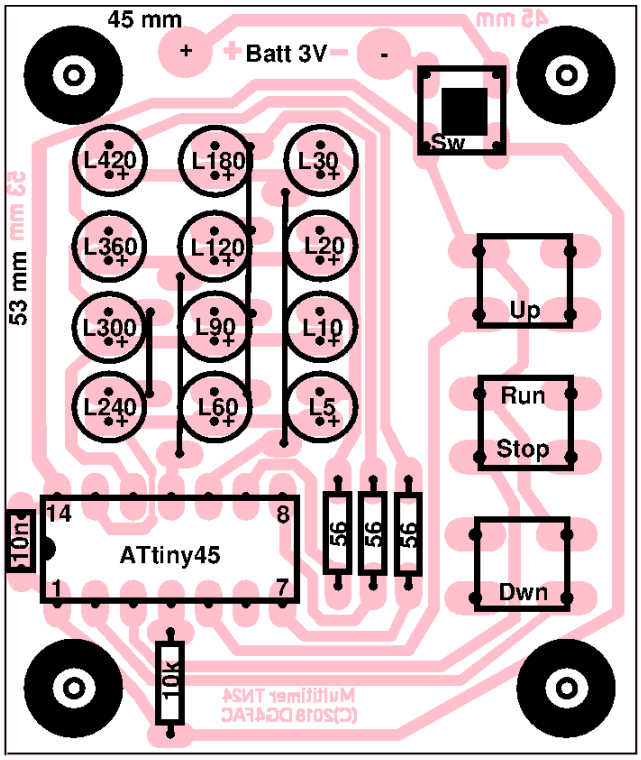
2.1 Gedruckte Platine

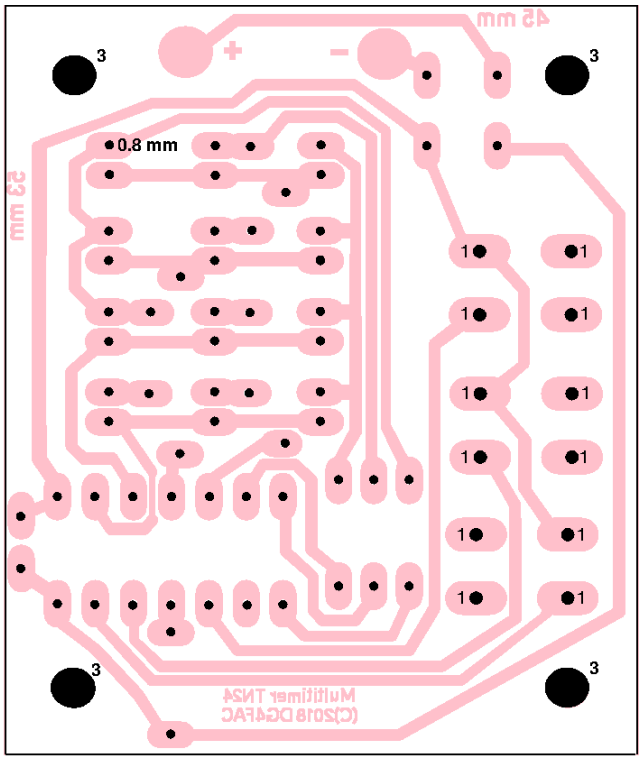
Das ist das Layout der 45-mal-53 mm großen gedruckten Platine. Die Maße wurden so gewählt, damit sechs davon auf eine Euro-Platine mit 160-mal-100 mm passen.



Das ist der Bohrplan für die Platine. Alle unbeschrifteten Löcher sind vom Durchmesser 0,8 mm.

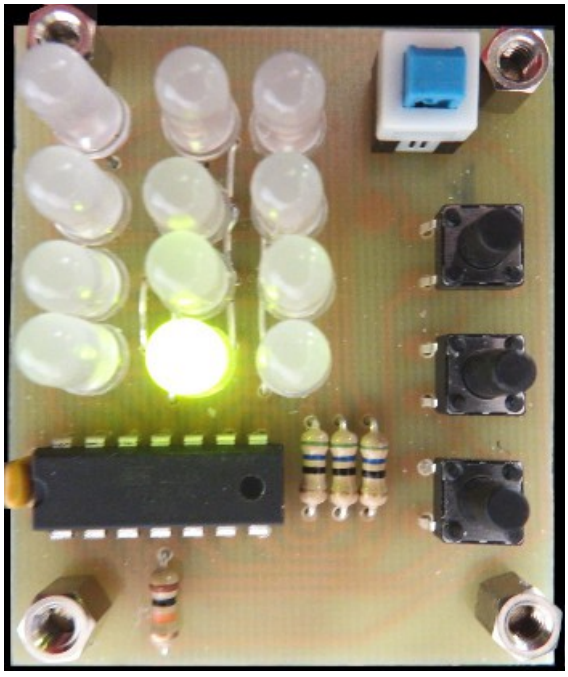
So sind die Bauteile auf der Platine zu bestücken.

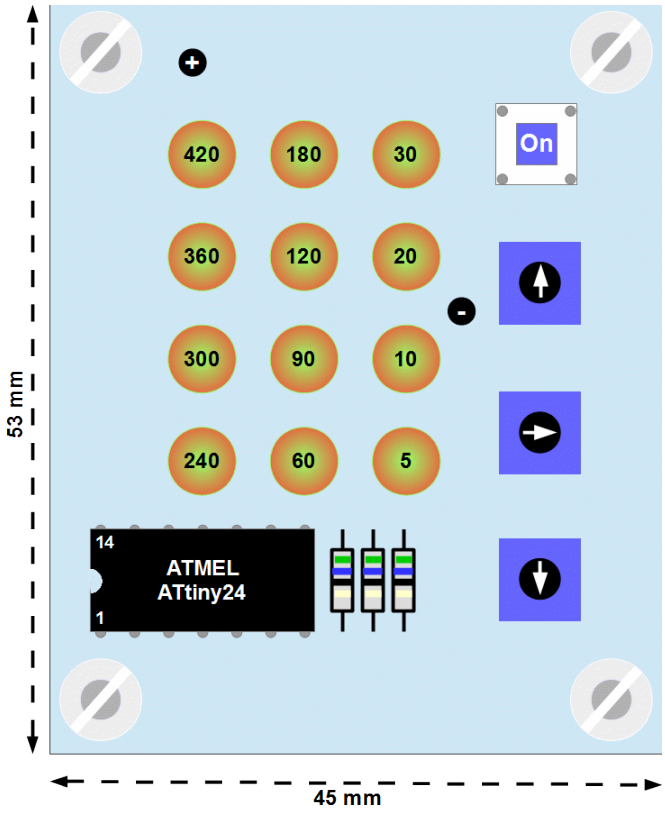




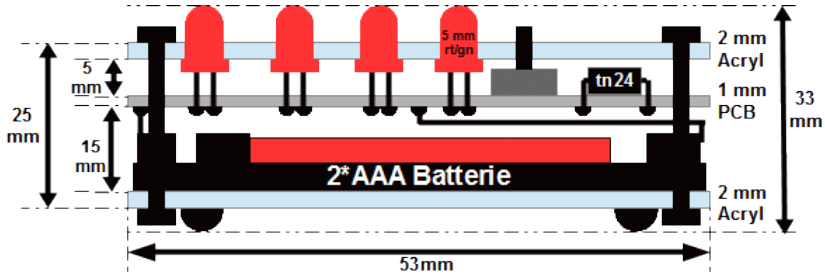
2.2 Montage

Das ist der Blick auf das Teil, wie es aus der Platzierung der Bauelemente auf der Platine resultiert. Die Taster und der Schalter sind für die Bedienung mit der rechten Hand ausgelegt (sorry, Linkshänder).



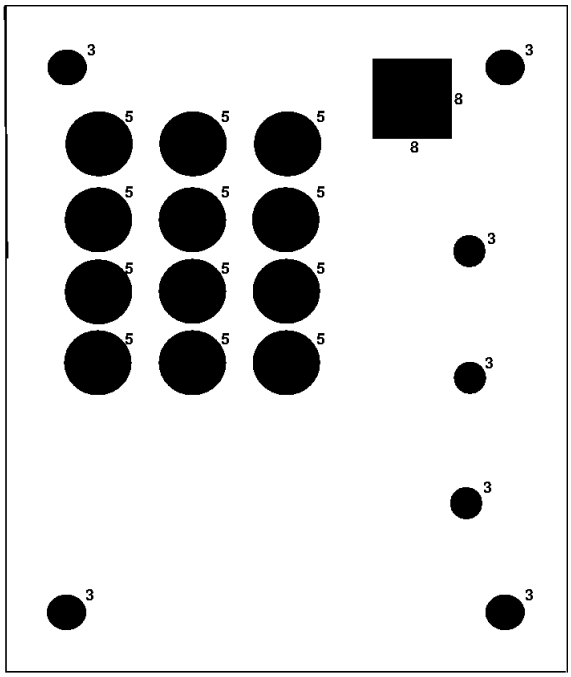


Das ist die Seitenansicht mit Bemaßung. Es zeigt die Plexiglas®-Platten auf der Ober- und Unterseite und wie der Controller, die LEDs, die Taster und die beiden AAA-Batterien reinpassen. Das Batteriegehäuse ist mit einem kurzen Stück Zwillingslitze von unten her an die Platine gelötet. Durch Abschrauben der unteren Plexiglas®-Platte kann die Batterie leicht gewechselt werden.



So sieht die Leiterplatte mit den montierten Komponenten aus.

Das ist der Bohrplan für die Abdeckung.



Seitenanfang | Hardware | Aufbau | Software

3 Software

Die Software ist komplett in Assembler geschrieben, um vollen Zugriff auf die zeitlichen Abläufe zu erreichen und minimalen Batteriestromverbrauch. Der Controller wird zu mehr als 99,9% im Schlafmodus betrieben, wird nur beim 0,1-Sekunden-Interrupt und bei Tastendrücken aus dem Schlafmodus geholt. Das Gerät schaltet nach 10 Sekunden die Leuchtdioden ganz aus. Danach liegt der Strombedarf ungefähr bei der Batterie-Selbstentladungsrate.

Die folgenden Kapitel zeigen die Struktur der Software und wie sie funktioniert:

1. Kapitel 3.1 liefert Links zum Quellcode im Assembler- und im HTML-Format.
2. Kapitel 3.2 stellt die Hardware-Testoptionen vor, die im Quellcode eingeschaltet werden können.
3. In Kapitel 3.2 wird der Tasteninterrupt PCINT1 dargestellt.
4. Kapitel 3.3 zeigt die Erzeugung des 0,1-Sekunden-Interrupts.
5. Kapitel 3.4 demonstriert wie der 0,1-Sekunden-Interrupt behandelt wird. und
6. Kapitel 3.5 zeigt, wie der besondere Blinkrhythmus erzeugt wird.

Die Entscheidung, alle Abläufe innerhalb der beiden Interrupt-Service-Routinen zu bearbeiten, ist etwas ungewöhnlich, ist aber wegen der raschen Bearbeitung der Interrupts und der gemächlichen Abläufe ohne Probleme machbar.

3.1 Quellcode

Den Assembler-Quellcode gibt es hier zum Download und kann hier im Anhang angeschaut werden.

3.2 Hardware-Testoptionen

Für die Inbetriebnahme der fertig aufgebauten Schaltung enthält der Quellcode Testoptionen.

Mit dem Schalter Debug_Leds = 1 werden die LEDs reihum nacheinander in grün und rot beleuchtet.

Mit dem Schalter Debug_Switches = 1 werden die Schalter abgefragt und, falls gedrückt, auf drei der LEDs dargestellt.

3.3 Tasten-Interrupt PCINT1

Dieser Interrupt tritt ein, sobald eine Taste gedrückt oder losgelassen wird. Die Interrupt-Service-Routine muss

- die gedrückte Taste (Down, Start/Stop, Up) erkennen und
- die damit verbundene Aktion ausführen, und
- die Prellunterdrückung der Tasten durch Setzen des Prellunterdrückungsregisters (rTgl) auf seinen eingestellten Ausgangswert setzen (bei jedem Tastendruck) und darf
- weitere Tastendrücke innerhalb der Karenzzeit nicht und darf sie erst erst dann wieder akzeptieren wenn rTgl auf Null gezählt ist.

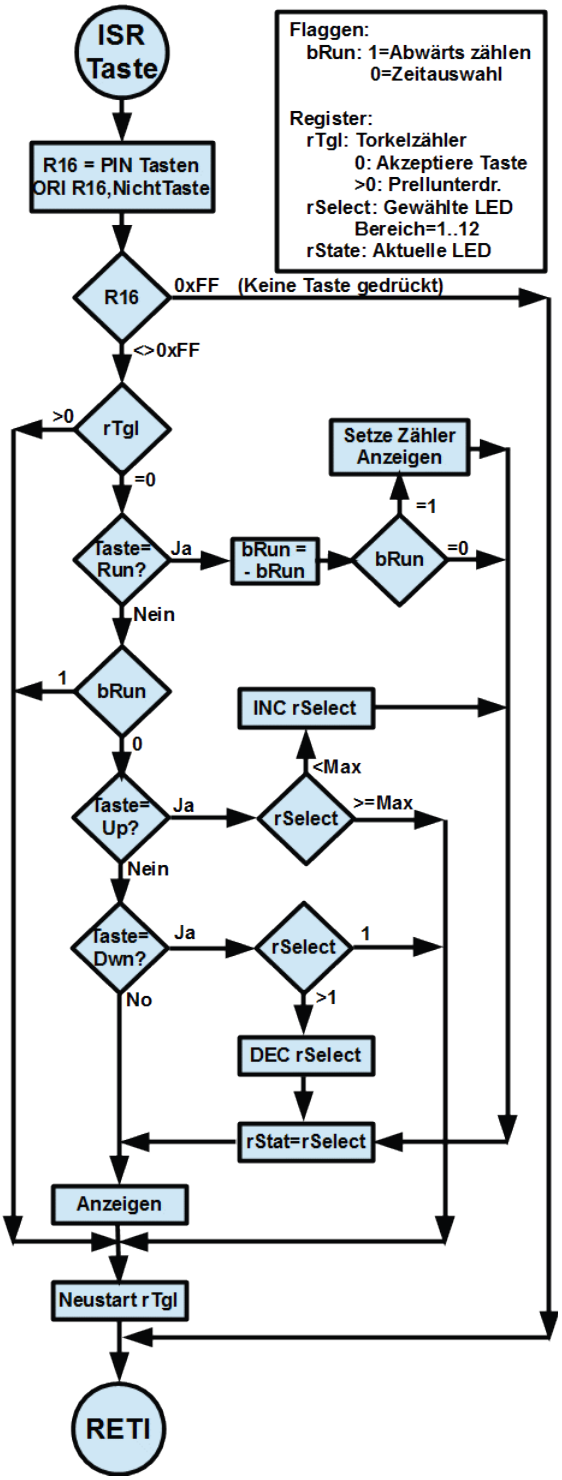
Zuerst wird der Eingabeport, an den die Tasten angeschlossen sind, gelesen. Alle nicht mit Tasten verbundenen Eingangsbits werden mit Einsen ausgeodert und festgestellt, ob mindestens einer der drei Tasteneingänge auf Null ist. Ist das der Fall wird überprüft, ob das Torkelregister rTgl auf Null ist. Ist dies nicht der Fall, wird der Tastendruck nicht weiter ausgewertet (Prellunterdrückung, erst wenn die Karenzzeit in der 0,1-Sekunden-Routine abgelaufen ist).

Dann wird die Taste Run abgefragt. Ist diese gedrückt, wird die bRun-Flagge invertiert. Ist bRun danach gesetzt, wird der Zählerwert in rSecH:rSecL auf den gewählten Zeitwert gesetzt. In beiden Fällen wird die gewählte LED mit der Nummer in rSelect in die aktuell angezeigt LED rState kopiert und angezeigt.

Ist bRun aktiv (Zähler zählt abwärts) erfolgt keine Auswertung der beiden ausstehenden Tasten Down und Up. Lählt der Timer nicht, wird bei den Tasten Down und Up überprüft ob der ausgewählte Wert der LED-Nummer bei Eins (Down-Taste) oder beim Maximalwert (Up-Taste) steht. Ist das nicht der Fall, wird der rSelect-Wert erhöht bzw. erniedrigt.

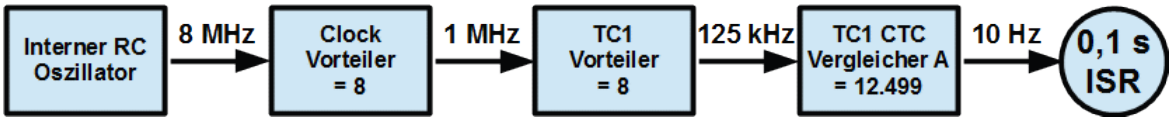
In allen beiden Fällen wird die gewählte LED in rSelect angezeigt. In allen Fällen mit aktivem Tastendruck wird das Prellunterdrückungsregister rTgl auf seinen Neustartwert gesetzt und damit bis zum Ende der Karenzzeit Tastenreaktionen unterbunden.

Der Assembler-Quellcode der Routine kann auf der oben gelinkten Webseite oder im Quellcode selbst eingesehen werden.



3.4 0,1-Sekunden-Takterzeugung

Werksseitig läuft der ATtiny24 mit dem internen 8-MHz-RC-Oszillator, teilt dessen Frequenz mit dem Taktvorteiler CLKPR durch 8 und taktet den



Controller und alle Timer mit 1 MHz Takt.

Der 16-Bit-Timer TC1 wird von dieser Software mit einem Vorteilerwert von 8 gestartet und in den CTC-Modus gebracht: beim Erreichen des Wertes im Vergleichsport A wird der Timer mit dem nächsten Taktimpuls auf Null gesetzt. Mit dem Vergleichswert von 12.499 ergibt sich ein Takt von 10 Hz.

Beim Erreichen des Compare-Werts in A wird der Compare-A-Interrupt ausgelöst.

3.4 0,1-Sekunden-Interrupt

Sieht kompliziert aus, besteht aber überwiegend aus einfachen Verzweigungen: das Ablaufdiagramm der 0,1-Sekunden-Interrupt-Service-Routine.

Zum Programmstart ist die Flagge bLedTest gesetzt und die Nummer der LED in rState wird einfach erhöht und die LED eingeschaltet. Das erfolgt bis zur 12-ten LED, dann wird bLedTest ausgeschaltet und die LED-Nummer wird auf den voraugewählten Zustand eingestellt. Ab jetzt beginnt der normale Ablauf.

Er beginnt mit der Abfrage, ob das Entprellregister rTgl Null ist. Wenn nicht, wird es um Eins vermindert.

Dann erfolgt die Abfrage, ob der Zähler gerade abwärts zählt oder steht (Flagge bRun). Ist das Zählen inaktiv, wird das 16-Bit-Sekundenregister in rSecH:rSecL um Eins vermindert. Erreicht es Null, wird die LED ausgeschaltet (Timeout).

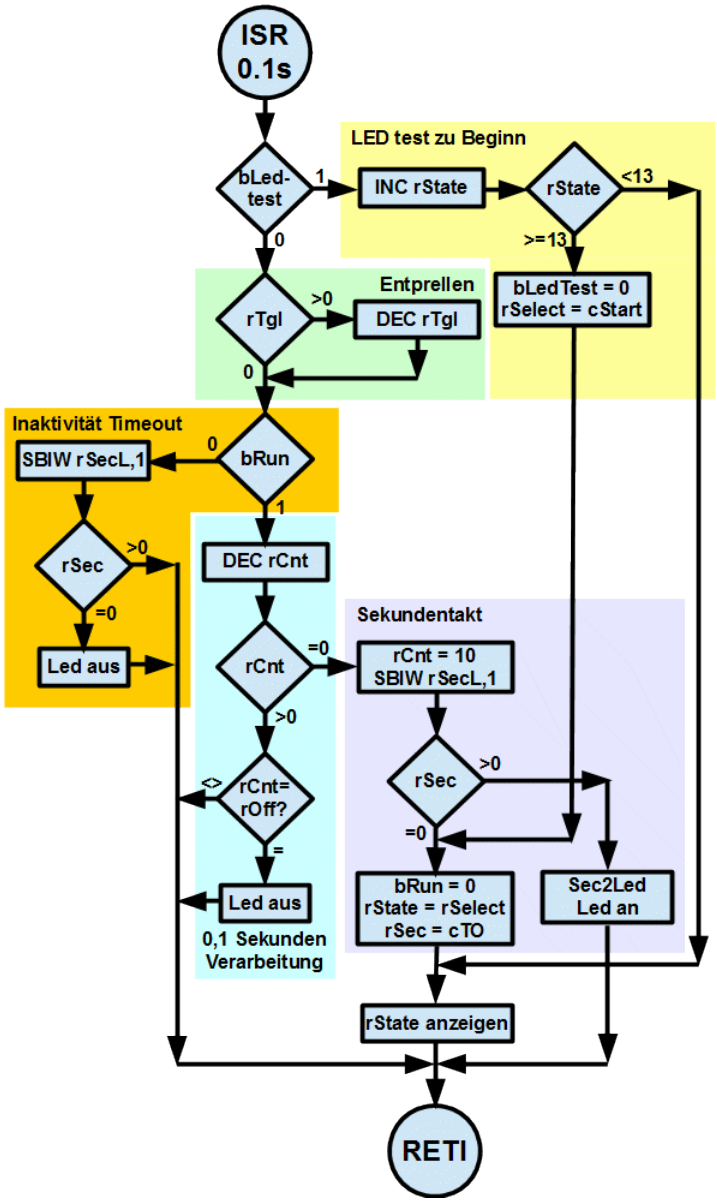
Ist Zählen aktiv, wird der Zehntelsekunden-Zähler rCnt abwärts gezählt. Ist der nicht Null, wird der Zählerstand in rCnt mit dem Abschaltwert in rOff verglichen. Sind beide gleich, wird die LED ausgeschaltet.

Mit dem Erreichen von Null bei rCnt ist eine Sekunde um und der 16-Bit-Sekundenzähler in rCntH:rCntL wird um eine Sekunde vermindert. Ist dieser nicht Null, wird der Sekundenstand in

- die anzuzeigende LED in rState, und
- in den Abschaltwert rOff (siehe unten)

umgewandelt und die erreichte LED aktiviert.

Wenn die Sekunden Null erreichen, wird die bRun-Flagge ausgeschaltet, die anzuzeigende LED-Nummer aus der Nummer der ausgewählten LED in rSelect entnommen, der Inaktivitäts-Abschaltwert in das Sekundenregister geladen und die LED in rState angezeigt.



Der gesamte Ablauf, einschließlich der Software-Multiplikation zum Berechnen des rOff-Wertes, bleibt unter einer halben Millisekunde und ist daher mit dem Zeitablauf verträglich, auch wenn während des Ablaufs der 0,1-Sekunden-Routine nicht auf Tastendruck-Interrupts reagiert werden kann.

Der resultierende Assembler-Code für diesen Ablauf kann im Quellcode besichtigt werden.

3.6 Blinkrhythmus

Das hier erläutert die LED-Ansteuerung am Beispiel der LED5. Diese LED ist mit der grünen Anode am Portpin PA0 (über einen Widerstand), mit der roten Anode an Portpin PA4 angeschlossen. Um zu leuchten, müssen beide Richtungsbits (DDA0 und DDA4) als Ausgang (=1) geschaltet sein. Der Ausgangsport PORTA bestimmt dann die Farbe der LED: Mit PORTA0 High und PORTA4 Low leuchtet die LED grün, sind beide Bits umgekehrt ist sie rot.

Ansteuerung der LED (LED5)

LED an: DDRA	0	0	0	1	0	0	0	1
LED gn: PORTA		x	x	0	x	x	x	1
LED rt: PORTA		x	x	1	x	x	x	0
		Rote Kathode Grüne Anode			Rote Anode Grüne Kathode			

Soll die LED ausgeschaltet werden, z. B. beim Blinken, wird einfach Null in den Richtungsport geschrieben. Soll die gleiche LED wieder eingeschaltet werden, wird der zuvor in das Register rDdr geschriebene Wert in den Richtungsport ausgegeben.

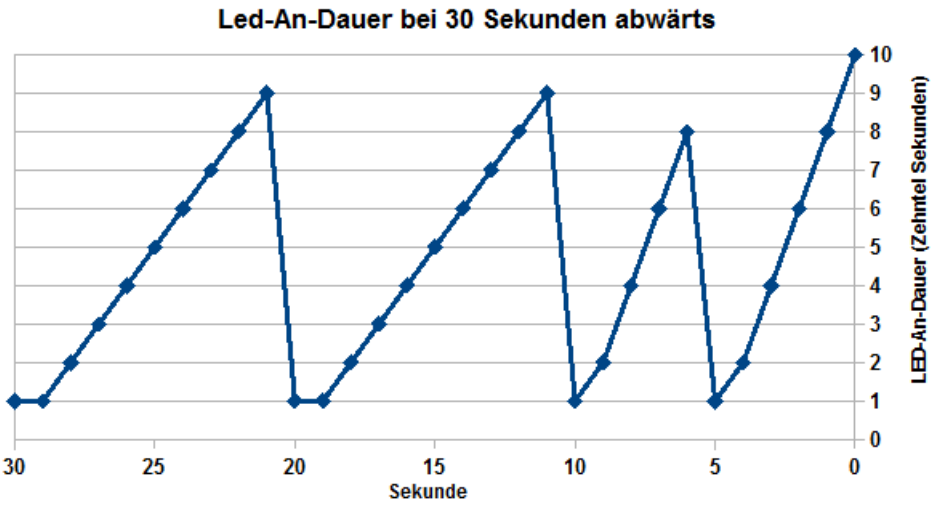
Das ist der nötige Assemblercode, um die LED in Abhängigkeit vom Zählzustand in der Flagge bRun entweder grün (bRun=0) oder grün (bRun=1) leuchten zu lassen: alle Lampen-Bits werden mit einem Exklusiv-Oder mit 0x7F einfach umgedreht (aus Nullen werden Einsen, aus Einsen werden Nullen).

Assembler-Beispiel:

```
ldi R16,0x01 ; Lade R16 fuer PORTA mit grüner LED
ldi R17,0x7F ; Lade R17 mit Farb-Inverter
sbrc rFlag,bRun ; Laeuft die Zeit?
  ; Ueberspringe naechste Instruktion wenn nicht
eor R16,R17 ; Ja, invertiere Farbe zu rot
out PORTA,R16 ; Zum Output-Treiber
ldi R16,0x11 ; Schalte Richtungsbits LED5 an
out DDRA,R16 ; Setze Richtungsport fuer LED5
```

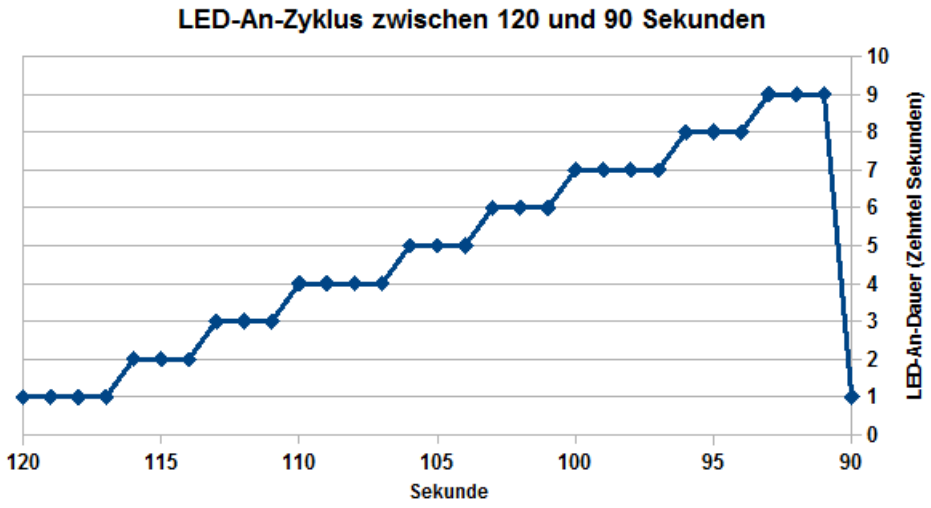
Sind die 12 LED-Kombinationen aus Richtungs- und Output-Werten in einer Tabelle abgelegt, kann sogar für den Fall, dass einzelne LEDs falsch herum eingebaut wurden, einfach der Grün-Wert anders definiert werden und die Farben der LED sind wieder korrekt.

Im Zählmodus (hier für 30 Sekunden abwärts) zeigt die An-Dauer der LED an, wie nahe der Zählerstand zur nächsten LED ist:



- Die 30-Sekunden-LED ist für eine zehntel Sekunde lang an und für neun zehntel Sekunden aus.
- Je näher die Zeit an die 20-Sekunden-Marke heranrückt, desto länger wird die An-Zeit und desto kürzer die Aus-Zeit, so dass man einen visuellen Eindruck von den erreichten Zwischenzeiten erhält.
- Wenn 20 Sekunden erreicht sind, geht LED20 an und beginnt mit einer zehntel Sekunde An-Zeit. Je näher 10 Sekunden rücken, desto länger wird die An-Zeit.
- Das Gleiche passiert, wenn fünf Sekunden erreicht werden.
- Letztlich, bei Null Sekunden Rest-Zeit, wird LED5 ausgeschaltet und LED30 wird grün.

Das ist die LED-An-Dauer zwischen 120 und 90 Sekunden.



Die Berechnung der An-Zeit der LED geht folgendermaßen (hier am Beispiel der Sekunden bei 13).

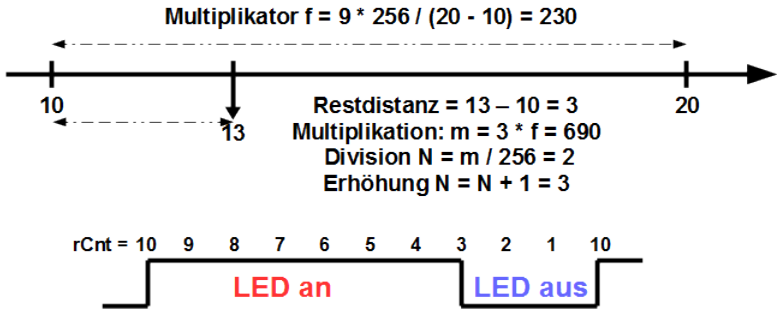
Der gesamte Abstand zwischen 20 und 10 Sekunden wird in 9 Abschnitte geteilt und mit 256 multipliziert (resultierender Faktor f - hier 230 - wird im Quellcode als Konstante abgelegt). f wird nach folgender Formel berechnet:

$$f = 9 * 256 / (N_{oben} - N_{unten})$$

und ist für die verschiedenen Zeitperioden

- zwischen 420 und 121: 38,
- zwischen 120 und 31: 77,
- zwischen 30 und 11: 230,

und in jedem Fall kleiner als 256 (8-Bit).



Die zwischen der aktuellen Zeit (13 Sekunden) und der nächstniedrigen Zeit (10 Sekunden) Zeitraum (3 Sekunden) wird mit diesem Faktor multipliziert, ergibt in diesem Fall 690 oder hexadezimal 0x02B2. Durch Wegstreichen des niedrigen Bytes (0xB2) wird der Wert durch 256 geteilt. Das MSB wird um Eins erhöht (2 ==> 3) und bildet den Wert rOff. Die Formel für diese Berechnung lautet also:

$$N = (T - N_{unten}) * f / 256 + 1$$

Man beachte, dass bei der Division in jedem Fall abgerundet wird.

Erreicht der Wert der Zehntelsekunden (in Register rCnt) diesen Abschaltwert, wird die LED abgeschaltet.

Im Quellcode ist diese Berechnung in der Unterroutine Sec2Led zu finden. Sie berechnet aus der Zeitinformation in rSecH:rSecL

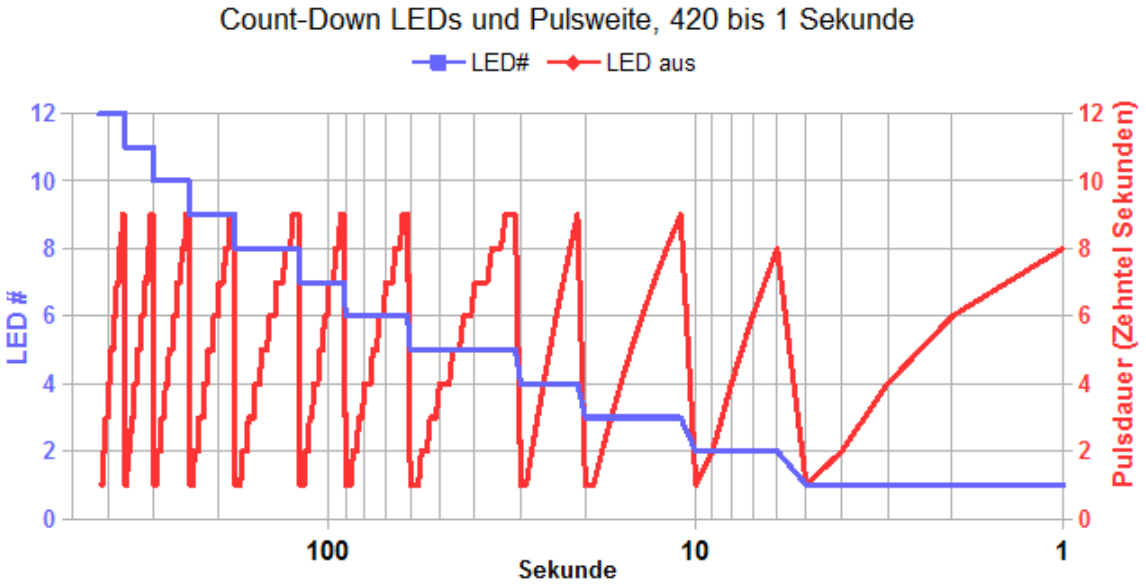
- die einzuschaltende LED in rState, und
- den Abschaltwert rOff in Zehntelsekunden.

Die dafür nötigen Werte sind als Tabellen im Flash abgelegt, z. B. die Dauern bei den verschiedenen LEDs:

```
;
; Dauertabelle
LedDur:
.dw 0,5,10,20,30
.dw 60,90,120,180
```

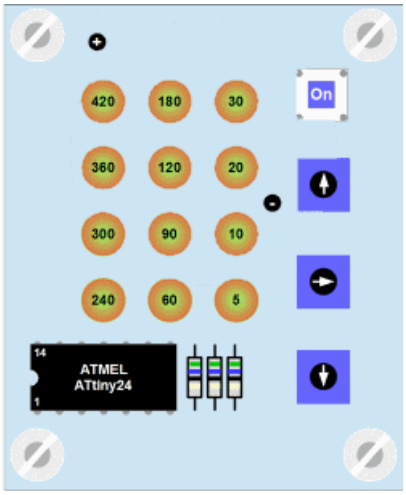
```
.dw 240,300,360,420
.dw 65535 ; Ende der Tabelle
;
; Multiplikatorentabelle
MultTab:
.db 0,0,0,230,230,77,77,77,38,38,38,38,38,1
;
; Zehnertabelle (fuer 10..1)
TenTable:
.db 0,2,4,6,8,9
;
; Led-Tabelle der Ports
;   1. Byte: PORTA, 2. Byte: DDRA
LedTable:
.db 0b00000000,0b00000000 ; #0, aus
.db 0b00010000,0b00010001 ; #1, LED 5 gruen
.db 0b00010000,0b00010010 ; #2, LED 10 gruen
.db 0b00010000,0b00010100 ; #3, LED 20 gruen
.db 0b00010000,0b00011000 ; #4, LED 30 gruen
.db 0b00100000,0b00100001 ; #5, LED 60 gruen
.db 0b00100000,0b00100010 ; #6, LED 90 gruen
.db 0b00100000,0b00100100 ; #7, LED 120 gruen
.db 0b00100000,0b00101000 ; #8, LED 180 gruen
.db 0b01000000,0b01000001 ; #9, LED 240 gruen
.db 0b01000000,0b01000010 ; #10, LED 300 gruen
.db 0b01000000,0b01000100 ; #11, LED 360 gruen
.db 0b01000000,0b01001000 ; #12, LED 420 gruen
;
```

Das Ergebnis der Berechnungen für den gesamten Zeitraum zeigt das Diagramm.



Angezeigt sind die Nummern der LEDs (linke Skala, in blau) und der Abschaltwert in Zehntelsekunden (rechte Skala, rot).

Seitenanfang	Hardware	Aufbau	Software



AVR-Anwendungen

Vielfachtimer mit ATtiny24 Assembler Quellcode

Quellcode für den Multitimer tn24

HTML-formatierter Assembler-Quellcode, der Original-Quellcode kann hier heruntergeladen werden.

```
;
; ************************************
; * Multitimer mit ATtiny24 und 12 LEDs *
; * Version 1.0 August 2018            *
; * (C)2018 www.avr-asm-tutorial.net   *
; ************************************
;
.nolist
.include "tn24def.inc"
.list
;
; *********************************
;        H A R D W A R E
; *********************************
;
; Device: ATtiny24, Package: 14-pin-PDIP_SOIC
;
;                 _______
;              1 /       |14
;       +3V o--|VCC GND|--o GND
; Taste Dwn o--|PB0 PA0|--o Led An 0
;  Taste Go o--|PB1 PA1|--o Led An 1
;     RESET o--|RES PA2|--o Led An 2
;  Taste Up o--|PB2 PA3|--o Led An 3
;        NC o--|PA7 PA4|--o Led Cat 0
; Led Cat 2 o--|PA6 PA5|--o Led Cat 1
;               |_______|
;
; ***********************************
;    H A R D W A R E   T E S T I N G
; ***********************************
;
; Alle Hardware Testcodes beginnen bei
;   000000 und enden in einer unendlichen
;   Schleife
;
; Testen der LEDs
;   Die LEDs LED5 bis LED420 werden
```

```
;     reihum eingeschaltet
;   Erste Runde: alle in gruen
;   Zweite Runde: alle in rot
.equ Debug_Leds = 0 ; 1 = Test, 0=Normal
;
.if Debug_Leds == 1
.equ cDelay=50000
  ldi R16,0 ; LEDs aus
  clr R18
Debug_Led1:
  ldi ZH,High(2*LedTable)
  ldi ZL,Low(2*LedTable)
  mov R17,R16
  lsl R17
  add ZL,R17
  ldi R17,0
  adc ZH,R17
  lpm R17,Z+
  eor R17,R18
  out PORTA,R17
  lpm R17,Z
  out DDRA,R17
  ldi ZH,High(cDelay)
  ldi ZL,Low(cDelay)
Debug_Led2:
  sbiw ZL,1
  brne Debug_Led2
  inc R16
  cpi R16,13
  brcs Debug_Led1
  clr R16
  ldi R17,0x7F
  eor R18,R17
  rjmp Debug_Led1
  .endif
;
; Testen der Taster
;   Liest die drei Taster
;     Status der Schalter wird angezeigt
;       LED5 (Down), LED60 (Run/Stop) und LED240 (Up)
;     solange der Taster gedrueckt ist
.equ Debug_Switches = 0 ; 1=Test, 0=Normal
;
.if Debug_Switches == 1
Debug_Sw:
  ldi R16,0x07
  out PORTB,R16
  in R17,PINB
  ldi R16,1
  sbrs R17,0
  rjmp Debug_Sw_Nmbr
  ldi R16,5
  sbrs R17,1
  rjmp Debug_Sw_Nmbr
  ldi R16,9
  sbrs R17,2
  rjmp Debug_Sw_Nmbr
  clr R16
Debug_Sw_Nmbr:
  ldi ZH,High(2*LedTable)
  ldi ZL,Low(2*LedTable)
  lsl R16
  add ZL,R16
  ldi R16,0
```

```
  adc ZH,R18
  lpm R16,Z+
  out PORTA,R16
  lpm R16,Z+
  out DDRA,R16
  clr R16
Debug_Sw_Nmbr1:
  dec R16
  brne Debug_Sw_Nmbr1
  rjmp Debug_Sw
  .endif
;
; ************************************
;  E I N S T E L L B A R E   K O N S T
; ************************************
;
.equ clock=1000000 ; Taktfrequenz
;
; Lednummer bei Programmbeginn
.equ cStart = 5 ; Kann zwischen 1 und 12 sein
;
; Entprellzyklen, in 0,1 Sekunden
.equ cDebounce = 2 ; Anzahl Perioden
;
; Inaktivitaetsdauer bis LED aus
;   in Zehntelsekunden
.equ cAutoOff = 100 ; Automatisch aus
;
; ************************************
;  F E S T E  &  A B G E L.  K O N S T
; ************************************
;
; TC1 erzeugt 0,1 s Takt
.equ cTc1Presc = 8 ; TC1 Vorteiler
.equ cTc1Div = clock / cTc1Presc / 10 ; TC1-Teiler
.equ cTc1CmpA = cTc1Div-1 ; Compare-A-Wert
;
; Tasten-Entprell-Wert
.equ cTgl = cDebounce + 1
;
; ********************************
;         R E G I S T E R
; ********************************
;
; frei: R0 bis R9
.def rOff = R10 ; Abschaltwert LED
.def rSelect = R11 ; Ausgewaehlte LED Nummer
.def rState = R12 ; Aktive LED
.def rPort = R13 ; LED Output-Wert PORTA
.def rDdr = R15 ; LED Richtungs-Wert DDRA
.def rmp = R16 ; Vielzweckregister
.def rFlag = R17 ; Flaggenregister
  .equ bRun = 0 ; Zaehler laeuft
  .equ bTimeOut = 1 ; Ende der LED-Anzeige
  .equ bLedTest = 7 ; LED-Test zu Beginn
.def rCnt = R18 ; Zaehler 0,1 Sekunden
.def rTgl = R19 ; Entprellzaehlerregister
; frei: R20 bis R23
.def rSecL = R24 ; Sekundenzaehler, LSB
.def rSecH = R25 ; dto., MSB
; benutzt: R27:R26 = X fuer Multiplikation
; frei: R29:R28 = Y
; benutzt: R31:R30 = Z fuer diverse Zwecke
;
```

```
; **********************************
;            C O D E
; **********************************
;
.cseg
;
; ************************************
; R E S E T  &  I N T - V E K T O R E N
; ************************************
  rjmp Main ; Reset Vektor
  reti ; EXT_INT0, unbenutzt
  reti ; PCI0, unbenutzt
  rjmp Pcint1Isr ; PCI1
  reti ; WATCHDOG, unbenutzt
  reti ; ICP1, unbenutzt
  rjmp Tc1CmpAIsr ; OC1A
  reti ; OC1B, unbenutzt
  reti ; OVF1, unbenutzt
  reti ; OC0A, unbenutzt
  reti ; OC0B, unbenutzt
  reti ; OVF0, unbenutzt
  reti ; ACI, unbenutzt
  reti ; ADCC, unbenutzt
  reti ; ERDY, unbenutzt
  reti ; USI_STR, unbenutzt
  reti ; USI_OVF, unbenutzt
;
; **********************************
;  I N T - S E R V I C E   R O U T .
; **********************************
;
; PCINT1 Externer Int
;   wird bei jeder Aenderung an den
;     Tasteneingaengen ausgefuhrt
;   Identifiziert die gedrueckte Taste und
;     fuehrt entsprechende Aktionen aus
Pcint1Isr:
  tst rTgl ; Pruefe Entprellregister
  brne PcInt1Isr9 ; Nicht Null, ignoriere Taste
  in rmp,PINB ; Lese Tasten
  ori rmp,0b11111000 ; Setze alle Nicht-Tasten-Bits
  cpi rmp,0xFF ; Keine Taste gedrueckt?
  breq Pcint1Isr9 ; Ja, Ende
  ldi ZL,cTgl ; Entprellregister auf Startwert
  mov rTgl,ZL
  sbrs rmp,1 ; Run/Stop-Taste?
  rjmp KeyRun ; Ja
  sbrc rFlag,bRun ; Laeuft Zaehler?
  reti ; Ja, keine weitere Tastenauswertung
  ldi rSecH,High(cAutoOff) ; Setze Inaktivitaetszaehler
  ldi rSecL,Low(cAutoOff)
  sbrs rmp,0 ; Down-Taste?
  rjmp KeyDown ; Ja
  sbrs rmp,2 ; Up-Taste?
  rjmp KeyUp ; Ja
PcInt1Isr9:
  reti
;
KeyDown:
  mov rmp,rSelect ; Gewaehlte LED Nummer
  cpi rmp,1 ; Bei Eins?
  breq KeyDown1 ; Ja, ignoriere Taste
  dec rSelect ; Gewaehlte LED Nummer abwaerts
  mov rState,rSelect
```

```
  rcall SetLed ; Zeige LED an
KeyDown1:
  reti
;
KeyUp:
  mov rmp,rSelect ; Gewaehlte LED Nummer
  cpi rmp,12 ; Unter 12?
  brcs KeyUp1 ; Ja
  ldi rmp,11 ; Lade Nummer 11
  mov rSelect,rmp ; in gewaehlte LED Nummer
KeyUp1:
  inc rSelect ; Erhoehe gewaehlte LED Nummer
  mov rState,rSelect ; Kopiere in angezeigte
  rcall SetLed ; Zeige LED an
  reti
;
KeyRun:
  ldi rmp,1<<bRun ; Invertier bRun-Flagge
  eor rFlag,rmp
  sbrs rFlag,bRun ; Ueberspringe naechste Instruktion wenn bRun Ein
  rjmp KeyStop ; Stoppen
  ldi ZH,High(2*LedDur) ; Z auf Tabelle mit Zeitdauer
  ldi ZL,Low(2*LedDur)
  mov rmp,rSelect ; Gewaehlte Nummer kopieren
  lsl rmp ; Mal zwei
  add ZL,rmp ; und zur Tabellenadresse addieren, LSB
  ldi rmp,0
  adc ZH,rmp ; Ueberlauf zu MSB addieren
  lpm rSecL,Z+ ; Sekundenzaehler aus Tabelle laden, LSB
  lpm rSecH,Z ; dto., MSB
  mov rState,rSelect ; In aktuelle LED kopieren
  ldi rmp,9 ; Abschaltwert ganz kurz
  mov rOff,rmp
  ldi rmp,10 ; Zehntelsekundenzaehler auf 10
  mov rCnt,rmp
  rcall SetLed ; LED anzeigen
  reti
;
KeyStop:
  ldi rSecH,High(cAutoOff) ; Lade Inaktivitaetszeit
  ldi rSecL,Low(cAutoOff)
  mov rState,rSelect ; Gewaehlte LED in angezeigte
  rcall SetLed ; LED anzeigen
  reti
;
; TC1 Compare A Interrupt
;   wird alle 0,1 s ausgefuehrt
;   Wenn Zaehlen nicht aktiv: Zehntelsekundenzaehler abwaerts
;     Wenn Null: LED abschalten
;   Wenn Zaehlen aktiv: Zehntelsekundenzaehler abwaerts
;     Wenn nicht Null: Zehntelsekunde = Abschaltregister?
;       Wenn gleich: LED aus
;     Wenn Null: Sekundenzaehler abwaerts
;       Wenn Null: Zaehlen aus, gewaehlte LED anzeigen
;       Wenn nicht Null: Sekunde in LED umrechnen und
;         Abschaltwert ausrechnen, LED anzeigen
Tc1CmpAIsr:
  sbrs rFlag,bLedTest ; LED Testphase?
  rjmp Tc1CmpAIsrRun ; Nein
  inc rState ; Naechste LED
  ldi rmp,13 ; Letzte LED?
  cp rState,rmp ; Vergleiche
  brcs Tc1CmpAIsrLed ; Nein, weiter LED anzeigen
  rjmp Tc1CmpAIsrStart ; Ja, starte
```

```
Tc1CmpAIsrRun:
  tst rTgl ; Pruefe Prellregister
  breq Tc1CmpAIsrRun1 ; Ist null, nicht vermindern
  dec rTgl ; Prellregister vermindern
Tc1CmpAIsrRun1:
  sbrs rFlag,bRun ; Zaehlen aktiv?
  rjmp Tc1CmpAIsrAuto ; Nein, Inaktivitaet testen
  dec rCnt ; Zahler 0,1 s abwaerts
  breq Tc1CmpAIsrSec ; Null, Sekunden zaehlen
  cp rCnt,rOff ; Ende LED an?
  brne Tc1CmpAIsrReti ; Nein
  clr rmp ; Loesche LED
  out DDRA,rmp
  reti
Tc1CmpAIsrSec:
  ldi rCnt,10 ; Neustart 0,1 s Zaehler
  sbiw rSecL,1 ; Sekunden abwaerts
  breq Tc1CmpAIsrStart ; Ende Sekunden, starte neu
  rcall Sec2Led ; Umrechnen Sekunden in LED und rOff
  rjmp Tc1CmpAIsrLed ; LED anzeigen
Tc1CmpAIsrAuto:
  sbrc rFlag,bTimeOut ; Timeout Inaktivitaet erreicht?
  reti ; Nein
  sbiw rSecL,1 ; Inaktivitaetszaehler abwaerts
  brne Tc1CmpAIsrBlink ; Nicht bei Null
  clr rmp ; Schalte LED aus
  out DDRA,rmp
  sbr rFlag,1<<bTimeOut ; Setze Timeout-Flagge
  reti
Tc1CmpAIsrBlink:
  ldi rmp,0 ; Blinke gruene LED
  sbrs rSecL,0 ; Zehntelsekunde Bit 0 = 1?
  out DDRA,rmp ; Nein, aus
  sbrc rSecL,0 ; Zehntelsekunde Bit 0 = 0?
  out DDRA,rDdr ; Nein, LED an
  reti
Tc1CmpAIsrStart:
  ; Starte Ablauf neu
  cbr rFlag,(1<<bRun)|(1<<bLedtest) ; bRun und bLedtest aus
  mov rState,rSelect ; Angezeigt = Gewaehlt
  ldi rSecH,High(cAutoOff) ; Setze Inaktivitaetszaehler
  ldi rSecL,Low(cAutoOff)
Tc1CmpAIsrLed:
  rcall SetLed ; Zeige LED an
Tc1CmpAIsrReti:
  reti

; ***********************************
;  I S R   U N T E R P R O G R A M E
; ***********************************
;
;
; LED aus Sekundenzaehler berechnen
;   Sekunden in rSecH:rSecL in Aktiv-LED
;     in rState und LED-Aus-Zeit in rOff
Sec2Led:
  ; Zeit in LED wandeln
  ldi ZH,High(2*LedDur+2)
  ldi ZL,Low(2*LedDur+2)
  clr rState ; rState ist LED # Zaehler
Sec2Led1:
  inc rState ; Erhoehe Zahler
  lpm XL,Z+ ; Lese LSB Dauer aus Tabelle in X
  lpm XH,Z+
```

```
  sec ; Carry auf Eins
  cpc rSecL,XL ; Vergleiche Zeit LSB
  cpc rSecH,XH ; Vergleiche Zeit MSB
  brcc Sec2Led1 ; Wiederhole mit naechstem Tabellenwert
  ; Lese niedrigeren Tabellenwert nach X
  sbiw ZL,4 ; Auf vorletzten Tabellenwert
  lpm XL,Z+ ; Lese LSB Tabellenwert
  lpm XH,Z ; dto., MSB
  ; Differenz Zeit - Tabellenwert
  mov ZH,rSecH ; Kopiere Zeit nach Z
  mov ZL,rSecL
  sub ZL,XL ; Subtrahiere Tabellenwert
  sbc ZH,XH
  mov XH,ZH ; Kopiere nach X
  mov XL,ZL
  ; Hole Multiplikator fuer LED Nummer
  ldi ZH,High(2*MultTab) ; Z auf Tabelle Multiplikatoren
  ldi ZL,Low(2*MultTab)
  add ZL,rState ; LED Nummer addieren, LSB
  ldi rmp,0
  adc ZH,rmp ; Ueberlauf MSB
  lpm rmp,Z
  ; Teste Multiplikator = 0
  tst rmp
  brne Sec2Led2 ; Nicht Null, multipliziere
  ; LED5 or LED10
  ldi ZH,High(2*TenTable) ; Zehnertabelle
  ldi ZL,Low(2*TenTable)
  add ZL,XL ; Addiere Zeitdifferenz LSB
  adc ZH,XH ; dto., MSB
  lpm rOff,Z ; Lese rOff-Wert aus Tabelle
  ret
Sec2Led2:
  ; Ab LED20, Multiplikator nicht Null, multipliziere
  clr ZL ; Z ist Ergebnis
  clr ZH
Sec2Led3:
  tst rmp ; Fertig multipliziert?
  breq Sec2Led5 ; Ja, Ende Multiplikation
  lsr rmp ; Dividiere Multiplikator durch 2, niedrigstes Bit in Carry
  brcc Sec2Led4 ; Carry Null, nicht zum Ergebnis addieren
  add ZL,XL ; Addiere Multiplikator
  adc ZH,XH
Sec2Led4:
  lsl XL ; Multipliziere mit 2
  rol XH
  rjmp Sec2Led3 ; Weiter multiplizieren
Sec2Led5:
  inc ZH ; Plus Eins
  mov rOff,ZH ; In rOff
  ret
;
; Zeitdauertabelle
LedDur:
.dw 0,5,10,20,30
.dw 60,90,120,180
.dw 240,300,360,420
.dw 65535 ; End of table
;
; Multiplikatortabelle
MultTab:
.db 0,0,0,230,230,77,77,77,38,38,38,38,38,1
;
; Fuer die Sekunden zwischen 10 und 1 ist es einfacher
```

```
;   die Abschaltzeit aus einer eigenen Tabelle abzulesen
TenTable:
.db 0,2,4,6,8,9
;
; Schalte die LED in rState an
SetLed:
  mov rmp,rState ; Aktuelle LED Nummer
  lsl rmp ; Multipliziere mit 2
  ldi ZH,High(2*LedTable) ; Zeige auf LED-Tabelle
  ldi ZL,Low(2*LedTable)
  add ZL,rmp ; Addiere zu LSB
  ldi rmp,0
  adc ZH,rmp ; und Ueberlauf zu MSB
  lpm rPort,Z+ ; Port lesen
  lpm rDdr,Z ; Richtung lesen
  ldi rmp,0x7F ; Invertieren in rot
  sbrc rFlag,bRun ; Ueberspringe wenn nicht zaehlen
  eor rPort,rmp ; Invertiere Port
  out PORTA,rPort ; Output schreiben
  out DDRA,rDdr ; Richtung schreiben
  cbr rFlag,1<<bTimeOut ; Timeout-Flagge loeschen
  ret
;
; LED-Tabelle der Ports
;   1. Byte: PORT, 2. Byte: DDR
LedTable:
.db 0b00000000,0b00000000 ; #0, LED aus
.db 0b00010000,0b00010001 ; #1, LED 5 gruen
.db 0b00010000,0b00010010 ; #2, LED 10 gruen
.db 0b00010000,0b00010100 ; #3, LED 20 gruen
.db 0b00010000,0b00011000 ; #4, LED 30 gruen
.db 0b00100000,0b00100001 ; #5, LED 60 gruen
.db 0b00100000,0b00100010 ; #6, LED 90 gruen
.db 0b00100000,0b00100100 ; #7, LED 120 gruen
.db 0b00100000,0b00101000 ; #8, LED 180 gruen
.db 0b01000000,0b01000001 ; #9, LED 240 gruen
.db 0b01000000,0b01000010 ; #10, LED 300 gruen
.db 0b01000000,0b01000100 ; #11, LED 360 gruen
.db 0b01000000,0b01001000 ; #12, LED 420 gruen
;
; **********************************
;  H A U P T P R O G R A M   I N I T
; **********************************
;
Main:
  ; Stapel initiieren
  .ifdef SPH
    ldi rmp,High(RAMEND) ; Setze SPH bei ATtiny44/84
    out SPH,rmp
    .endif
  ldi rmp,Low(RAMEND)
  out SPL,rmp ; Initiiere LSB Stapelzeiger
  ; LED = 0
  clr rState
  rcall SetLed ; LED anzeigen
  ; Setze Startwerte
  ldi rmp,cStart
  mov rSelect,rmp ; Start mit vorgewaehltem Wert
  ldi rSecH,High(cAutoOff) ; Inaktivitaetswert
  ldi rSecL,Low(cAutoOff)
  ; Initiiere PCINT fuer Tasten
  ldi rmp,(1<<PORTB0)|(1<<PORTB1)|(1<<PORTB2) ; Pull-Ups an
  out PORTB,rmp ; und in Output-Port
  clr rmp ; Konfigurieren als Inputs
```

```
  out DDRB,rmp
  ldi rmp,(1<<PCINT8)|(1<<PCINT9)|(1<<PCINT10) ; Maskiere Tasteneingaenge
  out PCMSK1,rmp
  ldi rmp,1<<PCIE1 ; Enable Interrupt PCINT1
  out GIMSK,rmp
  ; Initiiere TC1
  ldi rFlag,(1<<bLedTest)|(1<<bRun) ; LED Testphase an
  ldi rmp,High(cTc1CmpA) ; Setze Vergleichswert CTC
  out OCR1AH,rmp ; MSB
  ldi rmp,Low(cTc1CmpA)
  out OCR1AL,rmp ; LSB
  clr rmp ; Mode TC1 Port A
  out TCCR1A,rmp ; Kontrollport TC1 A
  ldi rmp,(1<<WGM12)|(1<<CS11) ; CTC Compare A, Vorteiler=8
  out TCCR1B,rmp ; Kontrollport TC1 B
  ldi rmp,1<<OCIE1A ; TC1 Compare A Interrupt enable
  out TIMSK1,rmp ; in TC1 Interruptmaske
  ; Enable Sleep
  ldi rmp,1<<SE ; Schlafmodus Idle
  out MCUCR,rmp
  ;
  ; Enable Enterrupts
  sei ; Enable Interrupts
;
; *********************************
;   P R O G R A M M S C H L E I F E
; *********************************
;
Loop:
  sleep
  rjmp loop
;
; Ende Quellcode
;
```

Lobpreisungen, Fehlerberichte, Geschimpfe und Spam bitte via Kommentarseite an mich.

Zum Seitenanfang