

Es gibt eine neuere Version des Kochbuchs.

Ihr findet sie unter

<http://www.reaktivlicht.de/atmel.html>

*Das Kochbuch*

# **Reaktivlicht mit dem ATtiny13(V)**

Zusammengeschrieben von Ralf Pongratz  
Nach einem Thread im Forum [www.geoclub.de](http://www.geoclub.de)

Stand: 20. September 2009

# Inhaltsverzeichnis

<b>Inhaltsverzeichnis</b>	<b>iii</b>
<b>1 Warnhinweise</b>	<b>1</b>
<b>2 Microcontroller</b>	<b>2</b>
2.1 Merkmale des ATtiny 13 . . . . .	2
2.2 Pin-Belegung . . . . .	2
<b>3 Prinzip der Helligkeitsmessung mit Hilfe einer LED</b>	<b>5</b>
<b>4 Schaltungsaufbau</b>	<b>6</b>
4.1 Grundschialtung "Lichtmessung mittels LED" . . . . .	7
4.2 Grundschialtung "Lichtmessung mittels eine Fotowiderstandes (LDR)" . . . . .	7
4.2.1 Variante 1: Spannungsteiler abschaltbar . . . . .	7
4.2.2 Variante 2: Spannungsteiler nicht abschaltbar . . . . .	8
4.3 Weitere Variationen der Grundschialtungen . . . . .	8
4.4 Programmieradapter für die parallele Schnittstelle . . . . .	9
4.5 Programmieradapter für die serielle Schnittstelle . . . . .	9
<b>5 Programmiersoftware</b>	<b>11</b>
5.1 Voraussetzungen . . . . .	11
5.2 Bedienung von „Bascom AVR“ . . . . .	11
<b>6 Programme</b>	<b>15</b>
6.1 Programme für die Grundschialtung aus Kapitel 4.1 . . . . .	15
6.1.1 Grundprogramm . . . . .	15
6.1.2 Nachtaktiver Blinker . . . . .	16
6.1.3 Verbesserter nachtaktiver Blinker . . . . .	18

6.1.4	Verbesserter nachtaktiver Blinker mit TeachIn-Modus . . . . .	20
6.1.5	Verbesserter nachtaktiver Blinker mit TeachIn-Modus mit Lauflängenspeicherung	22
6.1.6	Verbesserter nachtaktiver Blinker mit Morsezeichenausgabe . . . . .	24
6.1.7	Verbesserter nachtaktiver Blinker mit Watchdog-Abschaltung . . . . .	25
6.2	Programme für die Grundschtung aus Kapitel 4.2.1 . . . . .	27
6.2.1	Verbesserter Nachtaktiver Blinker mit Watchdog-Abschaltung . . . . .	27
6.3	Programme für die Grundschtung aus Kapitel 4.2.2 . . . . .	28
6.3.1	Nachtaktiver Blinker mit A/D-Wandler . . . . .	28
<b>7</b>	<b>Problemlösungen</b>	<b>31</b>
<b>8</b>	<b>Bestelldaten</b>	<b>34</b>
	<b>Literatur</b>	<b>36</b>

# Kapitel 1

## Warnhinweise

Dieses Kochbuch erhebt keinen Anspruch auf Vollständigkeit und Richtigkeit. Jeder, der die hier angegebenen Schaltungen nachbaut und programmiert, tut dies auf eigene Verantwortung. Was vielen wahrscheinlich nicht klar ist: Ihr bastelt an einem offenen PC rum. Eure Schaltung ist direkt mit der Hauptplatine verbunden. Von der Einkopplung von Störungen und unangepassten Leitungswiderständen abgesehen (ist bei diesen älteren Schnittstellen nicht so wild) kann ein Kurzschluss zwischen einer Signal- und Masseleitung der Parallelen Schnittstelle ernsthaftere Folgen für den Rechner haben. Also sollte jeder, der keine Erfahrungen im Gebiet der Elektrotechnik hat, im eigenen Interesse sich zuerst eine Tasse Tee und einen bequemen Sessel besorgen und das Kochbuch komplett durchlesen. Die meisten Probleme lösen sich dadurch automatisch und für den Rest sind eine Menge nette Leute im Forum zu finden, die jede Frage beantworten.

# Kapitel 2

## Microcontroller

In der Schaltung wird überwiegend der ATtiny 13V von Atmel als Microcontroller eingesetzt. Dieses Kapitel gibt grundlegende Informationen über den Controller, seine Ausstattung und Funktionsweise.

### 2.1 Merkmale des ATtiny 13

- 8 Bit RISC Mirocontroller
- 1 kB Flash-Programmspeicher
- 64 Byte EEPROM
- 64 Byte SRAM
- 8-Bit Zähler / Timer mit Prescaler und zwei PWM-Kanälen
- 4 kanaliger 10 Bit Analog-Digital-Wandler mit interner Spannungsreferenz
- Programmierbarer Watchdog-Timer mit internem Oszillator
- Analogwertvergleicher
- In-System programmierbar über SPI-Port
- Externe und interne Interrupts
- 6 programmierbare I/O-Ports
- 1,8 - 5,5 V Versorgungsspannung für ATtiny 13V  
2,7 - 5,5 V Versorgungsspannung für ATtiny 13

### 2.2 Pin-Belegung

Abbildung 2.2 zeigt die Register, die für die Konfiguration der I/O-Ports benötigt werden.

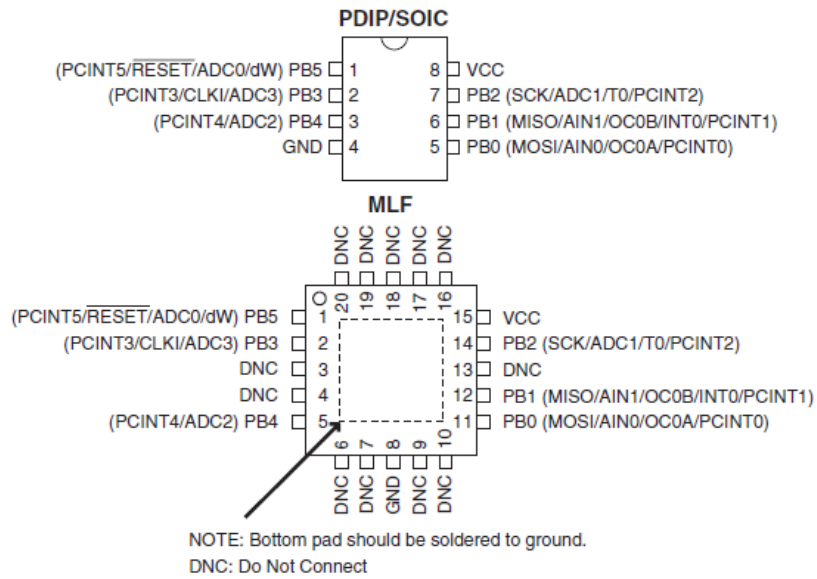


Abbildung 2.1: Pinbelegung des ATtiny 13.

DD <sub>xn</sub>	PORT <sub>xn</sub>	PUD (in MCUCR)	I/O	Pull-up	Kommentar
0	0	X	Eingang	Nein	Tri-state (Hi-Z)
0	1	0	Eingang	Ja	P <sub>xn</sub> liefert Strom, wenn extern auf low gezogen
0	1	1	Eingang	Nein	Tri-state (Hi-Z)
1	0	X	Ausgang	Nein	Ausgang Low
1	1	X	Ausgang	Nein	Ausgang High

Tabelle 2.1: Konfigurationsmöglichkeiten der I/O-Ports.

Um einen Pin als Eingang zu definieren, muss das zugehörige Bit DD<sub>xn</sub> (x steht hierbei für die Bezeichnung des Ports, n für die Nummer des Bits innerhalb des Ports) auf low gesetzt werden. Umgekehrt führt ein Setzen von DD<sub>xn</sub> auf high dazu, dass der Pin als Ausgang zu nutzen ist (vergleiche Tabelle 2.1).

In Bascom AVR können die I/O-Pins über den Befehl

```
Config Portb = &B00011000
```

gesetzt werden. Der Befehl setzt das Registerbyte DDRB. Dieses Beispiel konfiguriert die Pins PB3 und PB4 als Eingänge, den Rest als Ausgänge.

Auf die Zustände der Ein- und Ausgänge kann wie folgt zugegriffen werden:

```
Portb = &B00000000      ' Alle Ausgänge eines Ports werden auf low gesetzt
Portb.3 = 1             ' Nur Ausgang 3 wird auf high gesetzt
v = Portb.2             ' Der Zustand des Eingangs 2 wird in die Variable v geschrieben
```

## Port B Data Register – PORTB

Bit	7	6	5	4	3	2	1	0	
	–	–	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

## Port B Data Direction Register – DDRB

Bit	7	6	5	4	3	2	1	0	
	–	–	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0	DDRB
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

## Port B Input Pins Address – PINB

Bit	7	6	5	4	3	2	1	0	
	–	–	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	N/A	N/A	N/A	N/A	N/A	

Abbildung 2.2: Konfigurationsregister der I/O-Ports.



## Kapitel 3

# Prinzip der Helligkeitsmessung mit Hilfe einer LED

Die Kapazität einer Leuchtdiode (LED) in Sperrrichtung ist abhängig von der Beleuchtung. Im beleuchteten Zustand ist sie kleiner als im unbeleuchteten [1].

Bei der Grundschaltung im Kapitel 4.1 wird die Kapazität der LED regelmäßig aufgeladen und nach einer bestimmten Zeit die verbleibende Spannung nachgemessen. Aufgrund der Selbstentladung wird sie kleiner als die ursprünglich aufgeladene Spannung sein. Je größer die Kapazität der LED, desto größer ist die verbleibende Spannung. Somit ist die Restspannung über die LED ein Indiz für die Helligkeit der Beleuchtung der LED.

Durch den schmalen Abstrahlwinkel der LED ist gewährleistet, dass die Schaltung nur bei direkter Beleuchtung aktiviert wird. Es hat sich herausgestellt, dass am Besten superhelle rote LEDs mit transparentem Gehäuse geeignet sind.

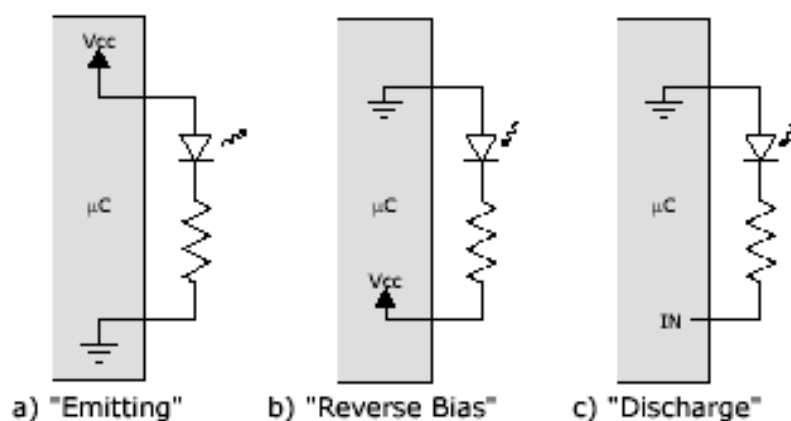


Abbildung 3.1: Prinzip der Lichtmessung.

# Kapitel 4

## Schaltungsaufbau

Das zentrale Element der Schaltung ist der Mikrocontroller ATtiny13(V) von Atmel.

Wenn die Kerbe im Gehäuse (manchmal auch ein Punkt auf der Oberfläche) nach links zeigt, ist links unten der Pin 1. Weiter geht es gegen den Uhrzeigersinn bis zu Pin 8 links oben. An Pin 8 wird die Versorgungsspannung (Pluspol der Batterie), an Pin 4 die Masse (Minuspol der Batterie) angeschlossen. Pin 1 ist der Reset-Eingang und muss im Betrieb auf High gehalten werden.

Als minimale Versorgungsspannung ist für den ATtiny13(V) 1,8 V festgelegt. Der ATtiny13 und der ATtiny12L benötigen mindestens 2,7 V. Für Versuchszwecke ist ein 5 V-Netzteil aber vollkommen ausreichend. Je geringer die Versorgungsspannung und je geringer die Taktfrequenz ist, desto weniger Strom wird verbraucht (Abbildung 4.1). So ist im normalen Betrieb ein Verbrauch im unteren  $\mu\text{A}$ -Bereich möglich, da viele interne Komponenten nicht benötigt und abgeschaltet werden können.

Figure 68. Active Supply Current vs.  $V_{CC}$  (Internal WDT Oscillator, 128 kHz)

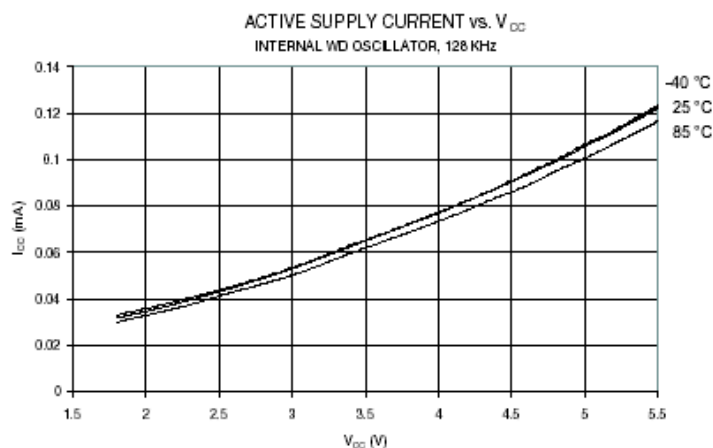


Abbildung 4.1: Zusammenhang zwischen Versorgungsspannung und Stromverbrauch bei 128 kHz.

Zwischen Pin 4 und Pin 8 kann optional ein Kondensator eingefügt werden. Dieser wird eigentlich zum Verringern von Störungen eingesetzt, hat hier aber den positiven Effekt, die Batterien besser auszunutzen. Im Idealfall werden ein Folien- oder keramischer Kondensator mit einer Kapazität von

100 nF und ein Elektrolytkondensator (Elko) mit einer Kapazität von 100  $\mu$ F parallel geschaltet. Allerdings können die Werte sehr flexibel an vorhandene Bauteile angepasst werden.

## 4.1 Grundschtaltung "Lichtmessung mittels LED"

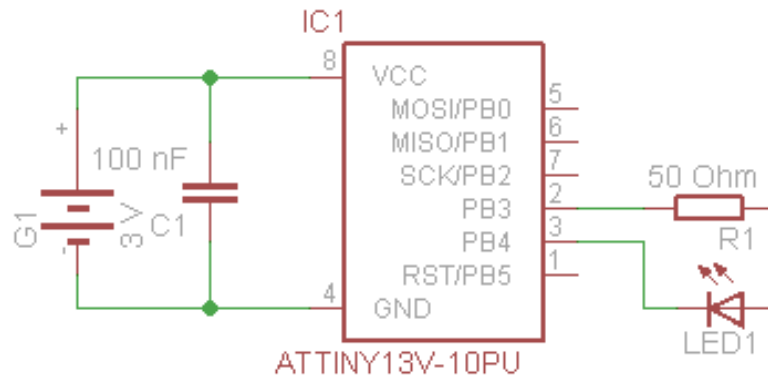


Abbildung 4.2: Aufbau der Schaltung.

Zwischen Pin 2 und Pin 3 sind in Reihe eine Leuchtdiode LED<sub>1</sub> und ein passender Vorwiderstand R<sub>1</sub> geschaltet. Dabei ist zu beachten, dass die Kathode der LED (kurzes Beinchen, flache Seite) an Pin 3 angeschlossen sein muss. Der passende Wert des Vorwiderstandes R<sub>1</sub> kann mit der Formel 4.1 berechnet werden.

$$R = \frac{U_B - U_D}{I_D} \quad (4.1)$$

Dabei ist  $U_B$  die Versorgungsspannung der Schaltung,  $U_D$  die Durchlassspannung der LED und  $I_D$  der Durchlassstrom der LED. Die Werte der LED können dem passenden Datenblatt entnommen werden. Typischerweise liegen sie bei  $U_D = 2$  V und  $I_D = 20$  mA. Daraus ergibt sich für  $U_B = 3$  V für R<sub>1</sub> ein Wert von 50 Ohm, für  $U_B = 5$  V ein Wert von 150 Ohm.

## 4.2 Grundschtaltung "Lichtmessung mittels eine Fotowiderstandes (LDR)"

Der Einsatz eines LDR lässt eine zuverlässigere und empfindlichere Helligkeitsmessung zu als bei der Grundschtaltung in Kapitel 4.1. Die höhere Empfindlichkeit geht geringfügig zu Lasten eines höheren Stromverbrauchs.

### 4.2.1 Variante 1: Spannungsteiler abschaltbar

Der LDR R<sub>3</sub> und der Widerstand R<sub>2</sub> bilden zusammen einen hochohmigen Spannungsteiler. Über PB0 wird der Spannungsteiler immer nur dann über den Ausgang PB4 mit Strom versorgt, wenn

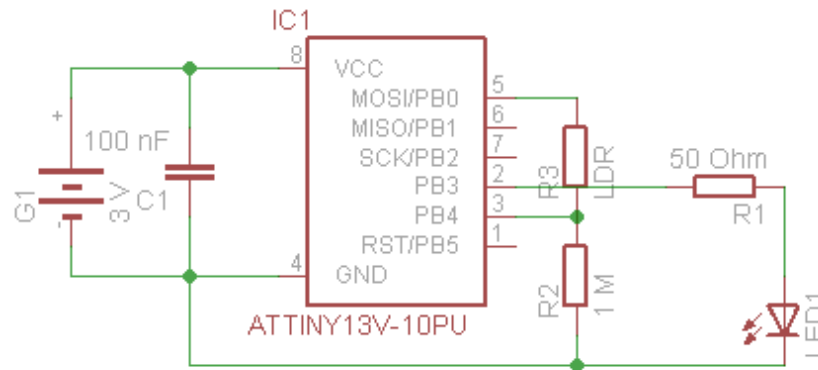


Abbildung 4.3: Aufbau der Schaltung.

tatsächlich eine Messung stattfindet. Fällt kein Licht auf den LDR, so ist sein Widerstand deutlich höher als der des Widerstandes, sodass an PB4 Low anliegt. Wird der LDR hingegen beleuchtet, sinkt sein Widerstand und am Eingang liegt High an. Die Grenze zwischen Low und High verschiebt sich proportional zur Versorgungsspannung.

#### 4.2.2 Variante 2: Spannungsteiler nicht abschaltbar

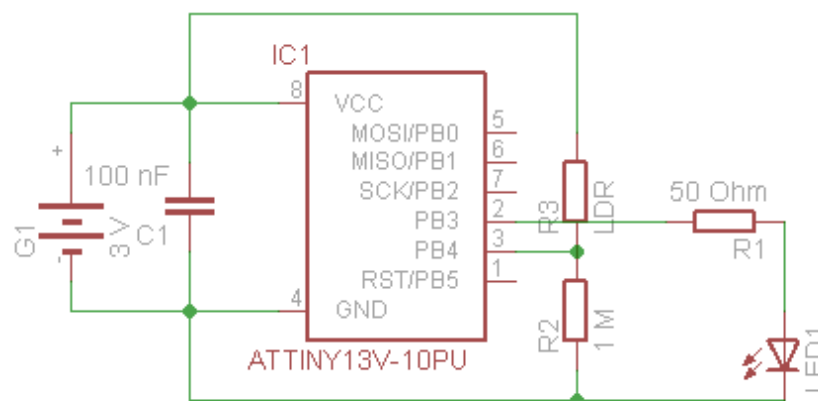


Abbildung 4.4: Aufbau der Schaltung.

Aufgrund der hochohmigkeit des Spannungsteilers ist die Stromaufnahme nur sehr gering. Insofern kann auf das Abschalten zwischen den Messungen verzichtet werden.

### 4.3 Weitere Variationen der Grundschaltungen

Der Vorwiderstand  $R_1$  kann weggelassen werden, wenn man die LED im Leuchtbetrieb mit 100...1 kHz moduliert. Allerdings kann es dann bei einer Fehlfunktion im Programm zu einer Zerstörung der LED kommen.

Durch Parallelschalten eines Kondensators zur LED in der Grundschaltung aus Kapitel 4.1 kann die Zeit, die die LED zur Entladung benötigt, verlängert werden, um evtl. zwischendurch den Prozessor in den Ruhezustand zu versetzen.

#### 4.4 Programmieradapter für die parallele Schnittstelle

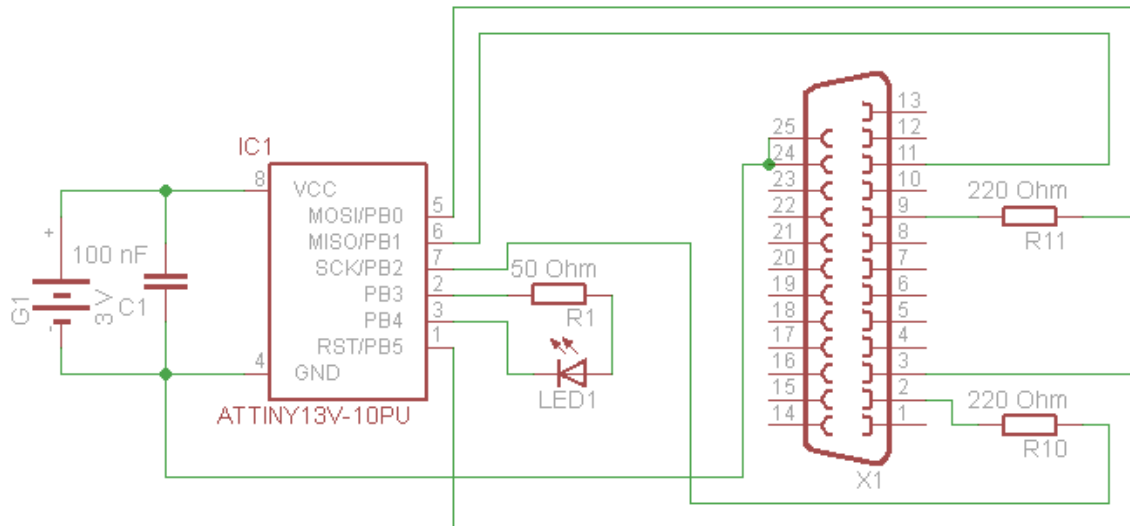


Abbildung 4.5: Aufbau des parallelen Programmieradapters (hier in Kombination mit der Grundschaltung aus Kapitel 4.1).

Für den Programmieradapter wird ein 25-poliger SUB-D-Stecker und ein Stück abgeschirmtes Kabel benötigt. Am einfachsten ist es, ein altes Druckerkaabel abzuschneiden. Pin 2 des Steckers wird über einen Widerstand  $R_{10} = 220 \text{ Ohm}$  an Pin 7 des ICs, Pin 3 des Steckers an Pin 1 des ICs, Pin 9 des Steckers über einen Widerstand  $R_{11} = 220 \text{ Ohm}$  an Pin 5 des ICs und Pin 11 des Steckers an Pin 6 des ICs und Pin 24 und Pin 25 des Steckers an Pin 4 des ICs (Masse) angeschlossen.

In Abbildung 4.5 ist der Programmieradapter mit der Grundschaltung aus Kapitel 4.1 kombiniert. Dies hat den Vorteil, dass der Controller für das Programmieren und das Testen nicht jeweils umgesetzt werden muss.

#### 4.5 Programmieradapter für die serielle Schnittstelle

Der Microcontroller kann auch über die serielle Schnittstelle programmiert werden. Abbildung 4.6 zeigt den dafür benötigten Programmieradapter. Programmiert werden kann mit der Software „PonyProg“. Erfahrungen zu „Bascom AVR“ liegen nicht vor. Die Baudrate wird automatisch ausgewählt. Man muss nur die COM-Schnittstelle wählen und „auto calibration“ aufrufen.

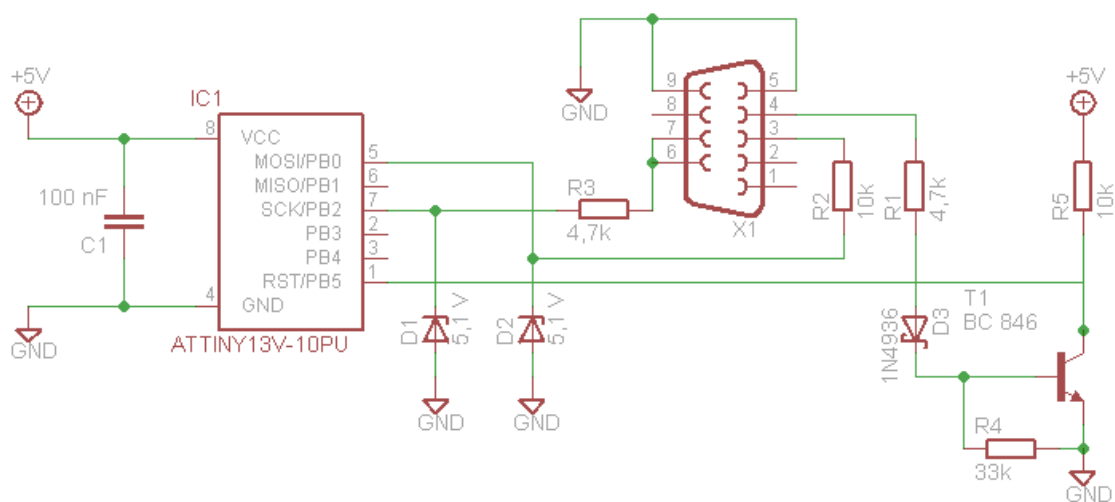


Abbildung 4.6: Aufbau des seriellen Programmieradapters.

# Kapitel 5

## Programmiersoftware

Für die Programmierung des Microcontrollers gibt es verschiedene Programme. Hier wird beispielhaft das Programm „Bascom AVR“ mit dem parallelen Programmieradapter (Kapitel 4.4) benutzt. Weitere Programme finden sich in der Linksammlung in Kapitel ...

### 5.1 Voraussetzungen

Benötigt wird ein PC mit einer parallelen Schnittstelle (Druckerschnittstelle), der parallele Programmieradapter und die Programmierumgebung, mit der die Programme geschrieben und auf den Prozessor übertragen werden können.

Die parallele Schnittstelle muss im BIOS des Rechners auf ECP+EPP (In- und Output) eingestellt werden. Hier eine Beispielseinstellung:

Onboard Parallel Port: 378/IRQ7

Parallel Port Mode: ECP+EPP

ECP Mode Use DMA: 3

Parallel Port EPP Type: EPP1.7

### 5.2 Bedienung von „Bascom AVR“

Anmerkung: Alle Angaben basieren auf die Programmversion 1.11.8.3. Neuere Versionen können davon abweichen.

Zu Beginn wird „Bascom AVR“ gestartet und im Menü „File“ mittels „New“ ein neues Programmfenster erstellt. Danach wird der Programmieradapter und die Chip-Parameter eingestellt. Dazu wird im Menü „Options“ der Eintrag „Programmer“ ausgewählt.

Nun wird der Reiter „Compiler“ aufgerufen und darin der Reiter „Chip“ (Abbildung 5.1). Im Drop-Down-Menü „Chip“ wird der Eintrag „attiny13.dat“ ausgewählt und in die darunterliegenden Felder folgende Werte eingetragen:

HW Stack = 2

Soft Stack = 8

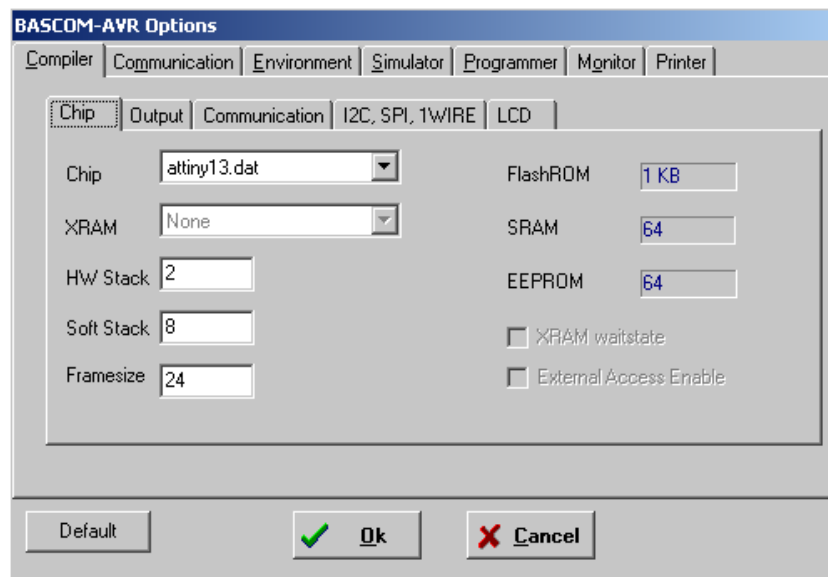


Abbildung 5.1: Einstellen der Chip-Parameter.

Framesize = 24

Anschließend werden die Werte durch Klicken auf „Default“ gespeichert.

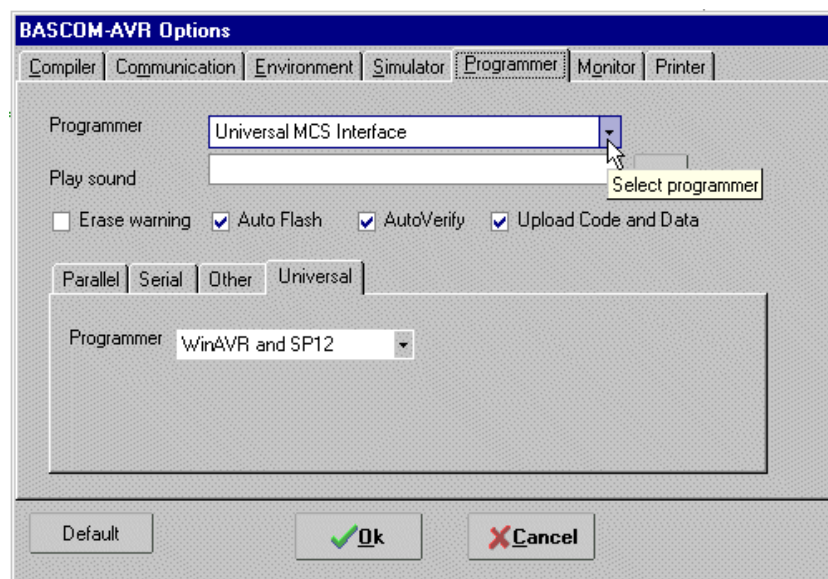


Abbildung 5.2: Einstellen des Programmieradapters.

Danach wird im Reiter „Programmer“ im Drop-Down-Menü „Programmer“ der Eintrag „Universal MCS Interface“ ausgewählt (Abbildung 5.2). Im Reiter „Universal“ wird als Programmier „WinAVR and SP12“ ausgewählt. Anschließend wird der Dialog durch Klicken auf „OK“ geschlossen.

Als nächstes werden die Fuse-Bits eingestellt. Dies sind Speicherzellen, die das Grundverhalten des Tinys festlegen. Dazu klickt man in der Icon-Leiste auf den kleinen, grünen IC-Sockel und wählt den Punkt „Manual Program“ aus (Abbildung 5.3).

Im sich öffnenden Dialogfenster klickt man auf den Reiter „Lock and Fuse Bits“. Nun liest Bascom



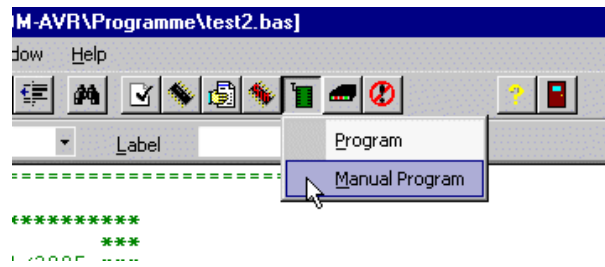


Abbildung 5.3: Einstellen der Fuse-Bits.

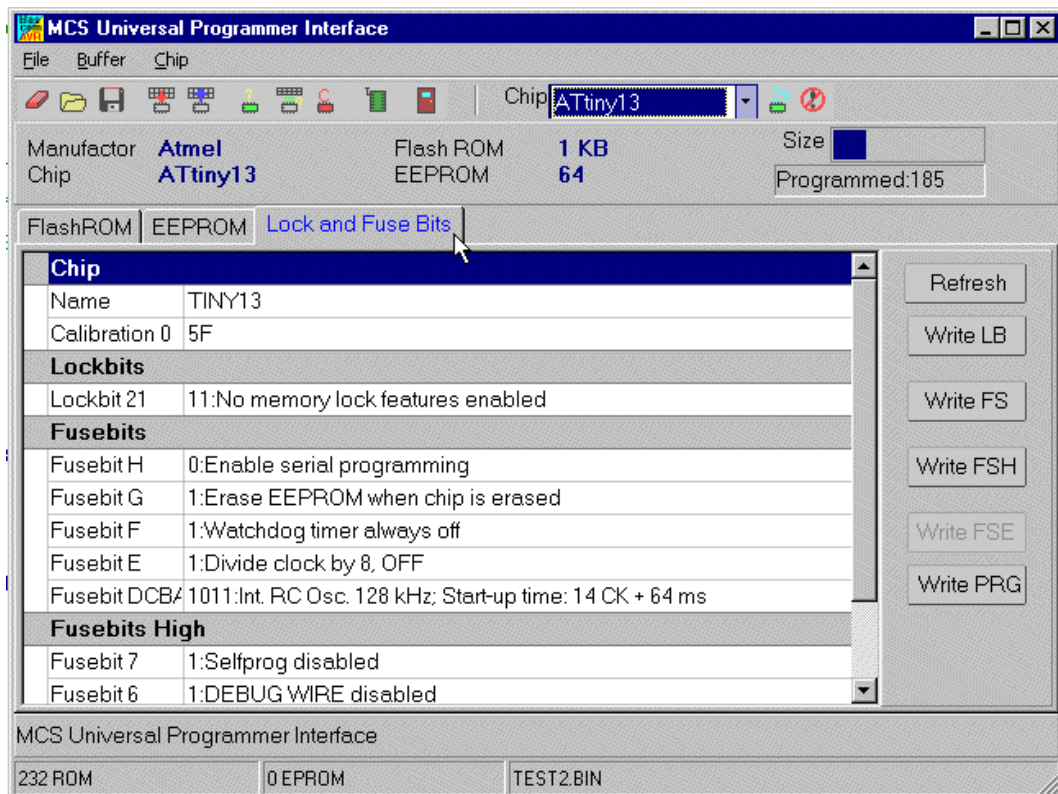


Abbildung 5.4: Einstellen der Fuse-Bits.

die Einstellung der Fuse-Bits aus dem Tiny aus und zeigt sie an (Abbildung 5.4).

Um Veränderungen an den Fuse-Bits vorzunehmen, klickt man auf die entsprechende Zeile, in der sich das gewünschte bzw. zu ändernde Bit befindet. In dem sich öffnenden Drop-Down-Menü kann daraufhin der gewünschte Wert ausgewählt werden. Folgende Einstellungen müssen hier vorgenommen werden:

Fusebit DCBA auf „1011:Int. RC Osc. 128 kHz; start-up time: 14 CK + 64 ms“

Fusebit E auf „1:Divide clock by 8, OFF“

Bitte die Einstellungen noch einmal sehr genau kontrollieren. Ist alles richtig, rechts auf „Write FS“ klicken. Die geänderten Einstellungen werden daraufhin in den Tiny geschrieben.

Anschließend wird der Quellcode des Programms geschrieben. Ist dies erledigt, erfolgt die Kompilierung. Dazu im Menü „Program“ den Punkt „Compile“ auswählen (Abbildung 5.5). Der Compiler

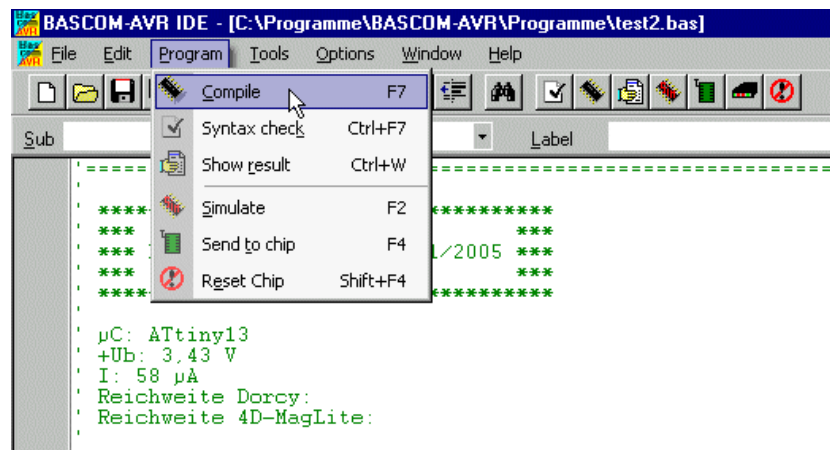


Abbildung 5.5: Kompilieren eines Programms.



Abbildung 5.6: Übertragen eines Programms.

starten nun. Enthält das Programm Fehler, wird eine entsprechende Meldung in der Fußzeile ausgegeben. Wurde das Programm erfolgreich kompiliert, kann es auf den Tiny übertragen werden. Dazu wird im Menü „Programm“ auf den Punkt „Send to Chip“ geklickt. Im sich öffnenden Fenster muss dann aus dem Menü „Chip“ der Eintrag „Autoprogram“ aufgerufen werden (Abbildung 5.6). Das zuvor kompilierte Programm wird nun in den Programmspeicher des Tiny geschrieben.

# Kapitel 6

## Programme

### 6.1 Programme für die Grundsaltung aus Kapitel 4.1

#### 6.1.1 Grundprogramm

Dieses Programm ist für die Variante „Lichtmessung mittels LED“. Die Schaltung gibt zehn kurze Lichtblitze zurück, wenn die LED angeleuchtet wird.

```
1  ' *****
2  ' *** ***
3  ' *** Reaktiv-Tiny v0.1 20/11/2005 ***
4  ' *** ***
5  ' *****
6
7  $regfile = "ATtiny13.DAT"
8  $crystal = 113000                                'Reale Frequenz des internen 128kHz-Oszillators
9
10 Config Portb = &B00011000                        'Pinb.3 und .4 auf 'Ausgang', Rest auf 'Eingang' schalten
11 Portb = &B11100111                                'Pullups zuschalten, außer für Pinb.3 und .4
12 Stop Adc                                          'A/D-Wandler abschalten, um Strom zu sparen
13 Stop Ac                                           'Analog-Komparator abschalten, um Strom zu sparen
14
15 Dim A As Byte
16 Dim Hell_dunkel As Bit
17
18 Do
19     Gosub Led_abfrage
20
21     If Hell_dunkel = 0 Then
22         For A = 1 To 10
23             Portb.3 = 1
24             Waitms 50
25             Portb.3 = 0
26             Waitms 500
27         Next A
28     End If
29
30 Loop
31
32 Led_abfrage:
33     Portb.3 = 0                                    'Portb.3 auf Masse schalten
34     Portb.4 = 1                                    'Portb.4 auf +Ub schalten, um die LED zu 'laden'
35     Waitus 1                                       'Ladezeit, kann ggf. noch verkleinert werden
36     Config Portb.4 = Input                          'Portb.4 nun zwecks Abfrage der LED-Ladung auf 'Eingang' schalten
37     Portb.4 = 0                                    'Pullup abschalten, sonst geht's nicht!
38     Waitus 1500                                    'Entladezeit - je kleiner, je unempfindlicher
39     Hell_dunkel = Pinb.4                            'Ladezustand einlesen
40     Config Portb.4 = Output                          'Portb.4 wieder auf Ausgang schalten
41     Portb.4 = 0                                    'Portb.4 auf Masse schalten
```

```

42 Return
43
44 End

```

Zeile 18 - 30 enthält das Hauptprogramm. In einer Endlosschleife wird die Helligkeit der Beleuchtung der LED abgefragt (Prozedur `Led_abfrage`) und der Beleuchtungszustand in der Variablen `Hell_dunkel` gespeichert. Ist die Variable auf 0 gesetzt (entspricht dem beleuchteten Zustand), so werden die Zeilen 23 - 26 zehnmal aufgerufen. Sie schalten den Ausgang PB3, an dem die LED angeschlossen ist, für 50 ms auf High und warten danach 500 ms.

In der Prozedur `Led_abfrage` (Zeilen 32 - 42) wird der Beleuchtungszustand der LED abgefragt. Zu Beginn wird Pin PB3 auf Low, Pin PB4 auf High gesetzt, wodurch die Kapazität der LED in Sperrrichtung aufgeladen wird. Nach 1  $\mu$ s Ladezeit wird der Pin PB4 als Eingang umkonfiguriert (Zeilen 36 - 37). Nun beginnt die Wartezeit für die Entladung der Kapazität. Nach 1500  $\mu$ s wird der Zustand der Kapazität eingelesen und in der Variablen `Hell_dunkel` gespeichert. Anschließend wird Pin PB4 für den nächsten Zyklus wieder als Ausgang konfiguriert. Über die Wartezeit in Zeile 38 wird die Empfindlichkeit der Schaltung eingestellt. Je kleiner die Wartezeit, desto unempfindlicher ist die Schaltung und desto stärker muss die LED beleuchtet werden, um auszulösen.

Alternativ gibt es das Grundprogramm auch in C. Die Funktion ist identisch.

```

1  #define F_CPU 8000000 // Quarzfrequenz 8MHz
2  #include <avr/io.h>
3  #include <avr/delay.h>
4  unsigned char f, i;
5
6  unsigned char led_abfrage(unsigned char zeit) {
7      PORTB &= ~(1<<PB3); // Portb.3 auf Masse schalten
8      PORTB |= (1<<PB4); // Portb.4 auf +Ub schalten, um die LED zu 'laden'
9      _delay_us(10); // Ladezeit, kann ggf. noch verkleinert werden
10     DDRB &= ~(1<<PB4); // Portb.4 für Abfrage der LED-Ladung auf 'Eingang' schalten
11     PORTB &= ~(1<<PB4); // Pullup abschalten, sonst geht's nicht!
12     _delay_ms(zeit); // Entladezeit zeit_ms - je kleiner, je unempfindlicher
13     i = (PINB & (1<<PINB4)); // Ladezustand einlesen
14     DDRB |= (1<<PB4); // Portb.4 wieder auf Ausgang schalten
15     PORTB &= ~(1<<PB4); // Portb.4 auf Masse schalten
16     return i;
17 }
18
19 int main(void) {
20     DDRE = 0b00011000; // Pinb.3 und .4 auf 'Ausgang', Rest auf 'Eingang' schalten
21     PORTB = 0b11100111; // Pullups zuschalten, außer für Pinb.3 und .4
22     while (1) {
23         if ( led_abfrage(6) == 0) { // LED durch Licht entladen?
24             for (f = 0; f < 10 ; f++) { // 10x blinken
25                 PORTB |= (1<<PB3); // PB3 auf Vcc schalten
26                 _delay_ms(32); // 32 ms Blitz
27                 PORTB &= ~(1<<PB3); // PB3 auf GND schalten
28                 for (i = 0; i < 10 ; i++)
29                     _delay_ms(32); // ca. 320ms Pause (workaround wegen 8MHz Quarz)
30             }
31         }
32     }
33     return 0;
34 }

```

## 6.1.2 Nachtaktiver Blinker

Dieses Programm ist eine Variante des Programms aus Kapitel 6.1.1. Bei lang anhaltender Beleuchtung lässt sich das Licht nicht mehr antriggern. Dadurch wird eine Auslösung bei Tage verhindert.

```

1  ' *****
2  ' *** ***
3  ' *** Reaktiv-Tiny v0.1 24/11/2005 ***
4  ' *** ***
5  ' *****
6
7  $regfile = "ATtiny13.DAT"
8  $crystal = 113000          'Reale Frequenz des internen 128kHz-Oszillators
9
10 Config Portb = &B00011000  'Pinb.3 und .4 auf 'Ausgang', Rest auf 'Eingang' schalten
11 Portb = &B11100111        'Pullups zuschalten, außer für Pinb.3 und .4
12 Stop Adc                  'A/D-Wandler abschalten, um Strom zu sparen
13 Stop Ac                    'Analog-Komparator abschalten, um Strom zu sparen
14
15 Dim A As Byte
16 Dim B As Byte
17 Dim Led_ladezustand As Bit
18 Dim Hell As Bit
19
20 Do
21     Gosub Led_abfrage
22
23     If Led_ladezustand = 0 Then
24         Hell = 1          'Bei Licht Merker setzen
25     End If
26     If Hell = 1 And B < 255 Then
27         B = B + 1        'Wenn Merker gesetzt wurde, Zähler für Licht dauer erhöhen (max. 255)
28     End If
29     If Led_ladezustand = 1 And Hell = 1 And B < 50 Then
30         Gosub Blinken    'Wenn es wieder dunkel ist und der Lichtimpuls nur kurz war, dann blinken
31     End If
32     If Led_ladezustand = 1 Then
33         Hell = 0 'löschen
34         B = 0
35     End If
36 Loop
37
38 Led_abfrage:
39     Portb.3 = 0          'Portb.3 auf Masse schalten
40     Portb.4 = 1          'Portb.4 auf +Ub schalten, um die LED zu 'laden'
41     Waitus 1            'Ladezeit, kann ggf. noch verkleinert werden
42     Config Portb.4 = Input 'Portb.4 nun zwecks Abfrage der LED-Ladung auf 'Eingang schalten
43     Portb.4 = 0          'Pullup abschalten, sonst geht's nicht!
44     Waitms 20           'Entladezeit 20 ms - je kleiner, je unempfindlicher
45     Led_ladezustand = Pinb.4 'Ladezustand einlesen: '1' -> dunkel, '0' -> hell
46     Config Portb.4 = Output 'Portb.4 wieder auf Ausgang schalten
47     Portb.4 = 0          'Portb.4 auf Masse schalten
48 Return
49
50 Blinken:
51     For A = 1 To 10
52         Portb.3 = 1
53         Waitms 50
54         Portb.3 = 0
55         Waitms 500
56     Next A
57 Return
58
59 End

```

Der Aufbau des Programms ist identisch mit dem Grundprogramm. Änderungen finden sich nur im Hauptprogramm in den Zeilen 20 - 36. Wenn die LED beleuchtet wird, wird ein Merker (**Hell**) gesetzt (Zeile 23 - 25). Solange dieser Merker gesetzt ist, wird in jedem Durchlauf die Variable **B** inkrementiert (Zeile 26 - 28). Die Schaltung gibt Blinksignale aus, wenn die sobald die Leuchtdiode nicht mehr beleuchtet wird, allerdings nur, wenn die voangegangene Beleuchtungsphase kurz genug gewesen ist (Zeile 29 - 31). Zum Schluss wird, wenn die LED unbeleuchtet ist, der Zähler **B** und der Merker **Hell** wieder zurückgesetzt.

Auch dieses Programm gibt es in C. Die Funktion ist identisch.

```

1  #define F_CPU 128000 // Oszillatorfrequenz in Hz
2  #include <avr/io.h>
3  #include <avr/delay.h>
4  #define LED_A PB4 // LED Pin Anode
5  #define LED_K PB3 // LED Pin Kathode
6  #define DISCHG 150 // Entladezeit der LED in ms. Je größer, desto empfindlicher.
7  #define DARKCNT 16 // Verhindert blinken bei Dauerbeleuchtung (Tag).
8 // Je größer der Wert, desto unempfindlicher gegen Taglicht.
9
10 unsigned char led_abfrage(void) {
11     unsigned char i;
12     PORTB &= ~(1<<LED_A); // Port LED_A auf Masse schalten
13     PORTB |= (1<<LED_K); // Port LED_K auf Vcc schalten, um die LED zu 'laden'
14     _delay_ms(1); // Ladezeit 1 ms
15     DDRB &= ~(1<<LED_K); // Port LED_K für Abfrage der LED-Ladung auf 'Eingang' schalten
16     PORTB &= ~(1<<LED_K); // Pullup abschalten
17     _delay_ms(DISCHG); // Entladezeit abwarten
18     if (PINB & (1<<LED_K)) // Ladezustand einlesen
19         i=0;
20     else
21         i=1;
22     DDRB |= (1<<LED_K); // Port LED_K wieder auf Ausgang schalten
23     PORTB &= ~(1<<LED_K); // Port LED_K auf Masse schalten
24     return i;
25 }
26
27 int main(void) {
28     unsigned char k;
29     DDRB = (1<<LED_A) | (1<<LED_K); // LED_A und LED_K als Ausgang definieren
30     PORTB = 0xFF; // Pullups zuschalten
31     PORTB &= ~(1<<LED_A)|(1<<LED_K); // Pullups für LED abschalten
32     OSCCAL = 0; // Oszillator auf geringste Taktfrequenz trimmen
33     ACSR = (1<<ACD); // analogen Komperator ausschalten, spart Strom
34     while (1) { // Main Loop
35         k = 0;
36         while (k < DARKCNT) { // nur wenn k-mal hintereinander "dunkel" erkannt wurde
37 // gehts weiter und die Schaltung ist "scharf"
38             if (led_abfrage())
39                 k = 0;
40             else
41                 k++;
42         }
43         while (!(led_abfrage())) { // hier ist die Schaltung scharf und wartet auf Licht
44         }
45         for (k = 0; k < 30 ; k++) { // ab hier beginnt das Blinkspektakel
46             PORTB |= (1<<LED_A);
47             _delay_ms(60);
48             PORTB &= ~(1<<LED_A);
49             _delay_ms(80);
50         }
51     }
52     return 0;
53 }

```

### 6.1.3 Verbesserter nachtaktiver Blinker

Dieses Programm ist eine Variante des Programms aus Kapitel 6.1.1. Bei lang anhaltender Beleuchtung lässt sich das Licht nicht mehr antriggern. Dadurch wird eine Auslösung bei Tage verhindert. Fehlauslösungen im Dämmerzustand werden vermieden.

```

1  ' *****
2  ' *** ***
3  ' *** Reaktiv-Tiny v0.1 24/11/2005 ***
4  ' *** ***
5  ' *****
6
7  $regfile = "ATtiny13.DAT"
8  $crystal = 113000 //Reale Frequenz des internen 128kHz-Oszillators
9

```

```

10 Config Portb = &B00011000          'Pinb.3 und .4 auf 'Ausgang', Rest auf 'Eingang' schalten
11 Portb = &B11100111                'Pullups zuschalten, außer für Pinb.3 und .4
12 Stop Adc                          'A/D-Wandler abschalten, um Strom zu sparen
13 Stop Ac                            'Analog-Komparator abschalten, um Strom zu sparen
14
15 Dim A As Byte
16 Dim B As Byte
17 Dim C As Byte
18 Dim Led_ladezustand As Bit
19 Dim Hell As Bit
20 Dim Dunkel As Bit
21
22 Do
23     Gosub Led_abfrage
24     If Led_ladezustand = 0 Then
25         Hell = 1                    'Bei Licht Merker setzen
26     End If
27     If Hell = 1 And B < 255 Then
28         B = B + 1                  'Wenn Merker gesetzt wurde, Zähler für Lichtdauer erhöhen (max. 255)
29     End If
30     If Led_ladezustand = 1 And Hell = 1 And B < 30 Then 'Wenn es wieder dunkel ist, nachsehen wie lange
31         Dunkel = 1                'Merker Setzen
32         For C = 0 To 5             'LED-Zustand mehrmals abfragen
33             Gosub Led_abfrage
34             If Led_ladezustand = 0 Then
35                 Dunkel = 0 'Wenn wieder Hell dann Dunkel-Merker löschen
36             End If
37         Next C
38         If Dunkel = 1 Then
39             Gosub Blinken          'Wenn Dunkel-Merker gesetzt blinken
40         End If
41     End If
42     If Led_ladezustand = 1 Then    'Bei Dunkelheit Merker und Zähler für Lichtdauer löschen
43         Hell = 0
44         B = 0
45     End If
46 Loop
47
48 Led_abfrage:
49     Portb.3 = 0                    'Portb.3 auf Masse schalten
50     Portb.4 = 1                    'Portb.4 auf +Ub schalten, um die LED zu 'laden'
51     Waitus 1                       'Ladezeit, kann ggf. noch verkleinert werden
52     Config Portb.4 = Input          'Portb.4 nun zwecks Abfrage der LED-Ladung auf 'Eingang' schalten
53     Portb.4 = 0                    'Pullup abschalten, sonst geht's nicht!
54     Waitms 100                     'Entladezeit 100 ms - je kleiner, je unempfindlicher
55     Led_ladezustand = Pinb.4        'Ladezustand einlesen: '1' -> dunkel, '0' -> hell
56     Config Portb.4 = Output        'Portb.4 wieder auf Ausgang schalten
57     Portb.4 = 0                    'Portb.4 auf Masse schalten
58 Return
59
60 Blinken:
61     For A = 1 To 10
62         Portb.3 = 1
63         Waitms 10
64         Portb.3 = 0
65         Waitms 100
66     Next A
67 Return
68
69 End

```

Dieses Programm ist eine Variation des Programms „Nachtaktiver Blinker“ aus Kapitel 6.1.2. Die Änderungen finden sich in den Zeilen 31 - 40. Während beim ursprünglichen Programm eine Blinksequenz unmittelbar nach der Abdunklung der LED ausgelöst wurde, muss bei dieser Variante auch die Dunkelphase eine Mindestdauer haben. In den Zeilen 32 - 37 wird fünfmal hintereinander die Helligkeit geprüft. Wird mindestens einmal der Zustand Hell detektiert, wird der Merker Dunkel wieder zurückgesetzt und die Blinksequenz nicht ausgegeben. Somit können Fehlauflösung in der Dämmerung vermindert werden.

### 6.1.4 Verbesserter nachtaktiver Blinker mit TeachIn-Modus

Bei diesem Programm wurde das Programm aus Kapitel 6.1.3 um eine frei einlernbare Blinksequenz erweitert. Nach dem Anlegen der Versorgungsspannung oder nach einem Reset kann eine neue Blinksequenz eingelernt werden. Dies geschieht folgendermaßen:

1. LED blinkt zehnmal kurz und einmal lang. Dies kennzeichnet den Beginn der TeachIn-Phase.
2. Nun muss mit Hilfe einer Taschenlampe die Blinksequenz in die LED eingegeben werden. Sie wird mit einem Abtastintervall von ca. 80 ms im EEPROM gespeichert. Wird die LED während der gesamten TeachIn-Phase nicht beleuchtet, wird zum Blinken die Standardsequenz benutzt (zehnmal blinken).
3. LED blinkt dreimal kurz. Dies kennzeichnet das Ende der TeachIn-Phase.

```

1  ' *****
2  ' *** ***
3  ' *** Reaktiv-Tiny v0.1 28/11/2005 ***
4  ' *** ***
5  ' *****
6
7  $regfile = "ATtiny13.DAT"
8  $crystal = 113000                'Reale Frequenz des internen 128kHz-Oszillators
9
10 Config Portb = &B00011000        'Pinb.3 und .4 auf 'Ausgang', Rest auf 'Eingang' schalten
11 Portb = &B11100111              'Pullups zuschalten, außer für Pinb.3 und .4
12 Stop Adc                        'A/D-Wandler abschalten, um Strom zu sparen
13 Stop Ac                          'Analog-Komparator abschalten, um Strom zu sparen
14
15 Dim A As Byte
16 Dim B As Byte
17 Dim Led_ladezustand As Bit
18 Dim Led_lichtcode As Byte
19 Dim Lichtcode_valid As Bit
20 Dim Hell As Bit
21
22 Lichtcode_valid = 0              'Merker löschen (EEPROM-Inhalt wird beim Blinken nicht verwendet)
23 Gosub Blinken                   '10 x kurz blinken
24 Portb.3 = 1
25 Wait 1                           '1 x lang blinken (Teach-In-Anfang)
26 Portb.3 = 0
27 For A = 1 To 63                  '63 Bytes des EEPROM's haben wir zur Verfügung
28     Gosub Led_abfrage
29     If Led_ladezustand = 0 Then    'Wenn LED beleuchtet wurde, dann
30         Led_lichtcode = 255      'Lichtcode-Byte setzen
31         Lichtcode_valid = 1      'Merker setzen (EEPROM-Inhalt wird beim Blinken verwendet)
32     Else                          'andernfalls
33         Led_lichtcode = 0        'Lichtcode-Byte löschen
34     End If
35     Writeeprom Led_lichtcode , A  'Lichtcode-Byte in's EEPROM schreiben
36     Waitms 65                    '65 ms warten
37 Next A
38 For A = 1 To 3
39     Portb.3 = 1
40     Waitms 50                     '3 x kurz blinken (Teach-In-Ende)
41     Portb.3 = 0
42     Waitms 50
43 Next A
44 Do
45     Gosub Led_abfrage
46     If Led_ladezustand = 0 Then
47         Hell = 1                  'Bei Licht Merker setzen
48     End If
49     If Hell = 1 And B < 255 Then
50         B = B + 1                'Wenn Merker gesetzt wurde, Zähler für Lichtdauer erhöhen (bis max. 255)
51     End If

```



```

52     If Led_ladezustand = 1 And Hell = 1 And B < 50 Then
53         Gosub Blinken           'Wenn es wieder dunkel ist und der Lichtimpuls nur kurz war, dann blinken
54     End If
55     If Led_ladezustand = 1 Then   'Bei Dunkelheit Merker und Zähler für Lichtdauer löschen
56         Hell = 0
57         B = 0
58     End If
59 Loop
60
61 Led_abfrage:
62     Portb.3 = 0                 'Portb.3 auf Masse schalten
63     Portb.4 = 1                 'Portb.4 auf +Ub schalten, um die LED zu 'laden'
64     Waitus 1                    'Ladezeit, kann ggf. noch verkleinert werden
65     Config Portb.4 = Input       'Portb.4 nun zwecks Abfrage der LED-Ladung auf 'Eingang' schalten
66     Portb.4 = 0                 'Pullup abschalten, sonst geht's nicht!
67     Waitms 20                   'Entladezeit 20 ms - je kleiner, je unempfindlicher
68     Led_ladezustand = Pinb.4     'Ladezustand einlesen: '1' -> dunkel, '0' -> hell
69     Config Portb.4 = Output      'Portb.4 wieder auf Ausgang schalten
70     Portb.4 = 0                 'Portb.4 auf Masse schalten
71 Return
72
73 Blinken:
74     Portb.3 = 0
75     Portb.4 = 0
76     If Lichtcode_valid = 0 Then  'Standard-Blinksequenz (ohne EEPROM-Inhalt)
77         For A = 1 To 10
78             Portb.3 = 1
79             Waitms 50
80             Portb.3 = 0
81             Waitms 500
82         Next A
83     Else                          'Blinksequenz mit EEPROM-Inhalt
84         For A = 1 To 63
85             Readeeprom Led_lichtcode , A
86             If Led_lichtcode = 255 Then
87                 Portb.3 = 1
88             Else
89                 Portb.3 = 0
90             End If
91             Waitms 90
92         Next A
93     End If
94 Return
95
96 End

```

Dieses Programm ist eine Variation des Programms „Nachtaktiver Blinker“ aus Kapitel 6.1.2. Bevor das Programm in die Endlosschleife (Zeilen 44 - 59) geht, erfolgt in den Zeilen 22 - 43 die TeachIn-Prozedur. Zu Beginn wird der Merker `Lichtcode_valid`, der angibt, ob sich eine gültige TeachIn-Sequenz im Speicher befindet, zurückgesetzt. Danach wird der Start der TeachIn-Prozedur durch zehnmaliges kurzes und einmaliges langes Blinken angezeigt. In den Zeilen 27 - 37 wird die Blinksequenz abgefragt. Im EEPROM stehen 63 Byte zur Verfügung, die nacheinander abgefragt werden. Ist die LED beleuchtet, so wird das Byte auf 255 gesetzt und `Lichtcode_valid` gesetzt (Zeilen 29 - 31). Ansonsten wird das Byte auf 0 gesetzt. Hinterher wird das Byte in das EEPROM geschrieben und eine Wartezeit eingelegt.

In der Blinkprozedur (Zeilen 73 - 94) wird nun der Merker `Lichtcode_valid` abgefragt, ob eine gültige Blinksequenz im EEPROM hinterlegt ist. Falls ja, wird dieser byteweise aus dem EEPROM gelesen und ausgegeben (Zeilen 84 - 92). Andernfalls erfolgt die Standard-Blinksequenz (Zeilen 77 - 82).

## 6.1.5. Verbesserter nachaktiver Blinker mit TeachIn-Modus mit Lauflängenspeicherung

### 6.1.5 Verbesserter nachaktiver Blinker mit TeachIn-Modus mit Lauflängenspeicherung

Bei diesem Programm wurde das Programm aus Kapitel 6.1.3 um eine frei einlernbare Blinksequenz erweitert. Im Gegensatz zum Programm aus Kapitel 6.1.4 wird die Blinksequenz in komprimierter Form gespeichert. Nach dem Anlegen der Versorgungsspannung oder nach einem Reset kann eine neue Blinksequenz eingelernt werden. Dies geschieht folgendermaßen:

1. LED blinkt zehnmal kurz. Dies kennzeichnet den Beginn der TeachIn-Phase.
2. Nun muss mit Hilfe einer Taschenlampe die Blinksequenz in die LED eingegeben werden. Sie wird mit einem Abtastintervall von ca. 80 ms im EEPROM gespeichert.
3. LED blinkt dreimal kurz. Dies kennzeichnet das Ende der TeachIn-Phase.

```
1  ' *****
2  ' ***
3  ' *** Teach-In mit Lauflängencodierung
4  ' ***
5  ' *****
6
7  $regfile = "ATtiny13.DAT"
8  $crystal = 113000                'Reale Frequenz des internen 128kHz-Oszillators
9  $hwstack = 6
10
11 Config Portb = &B00011000        'Pinb.3 und .4 auf 'Ausgang', Rest auf 'Eingang' schalten
12 Portb = &B11100111              'Pullups zuschalten, außer für Pinb.3 und .4
13 Stop Adc                        'A/D-Wandler abschalten, um Strom zu sparen
14 Stop Ac                          'Analog-Komparator abschalten, um Strom zu sparen
15
16 Dim A As Byte
17 Dim B As Byte
18 Dim Led_ladezustand As Bit
19 Dim Led_lichtcode As Byte
20 Dim Hell As Bit
21 Dim Last As Bit                  'Speichern des letzten Helligkeitswertes
22 Dim Count As Byte               'Speichern der Lauflänge
23 Dim I As Byte                   'Zählvariable
24
25 For I = 1 To 10
26     Portb.3 = 1
27     Waitms 50                    '10 x kurz blinken (Teach-In-Anfang)
28     Portb.3 = 0
29     Waitms 250
30 Next I
31 Gosub Led_abfrage
32 Last = Led_ladezustand
33 B = 0
34 B = Bits(last)
35 Writeeprom B , 1                'Speichern der Starthelligkeit an Byte 1 im EEPROM
36 Count = 1
37 Waitms 65
38 Gosub Led_abfrage
39 For A = 2 To 63                  '62 Bytes des EEPROM's haben wir zur Verfügung für die Lauflängen
40     While Last = Led_ladezustand And Count < 254 'Längen werden gezählt
41         Count = Count + 1
42         Waitms 65
43         Gosub Led_abfrage
44     Wend
45     If Count = 254 Then           '254 Längen erreicht -> Abspeichern 255 -> weiter mit gleichem LED-Zustand
46         Count = 255
47     Else                          'LED-Zustand.Wechsel -> Abspeichern der Länge, Invertierung von Last
48         Last = Not Last
49     End If
50     Writeeprom Count , A
51     Count = 0
```

## 6.1.5. Verbesserter nachtaktiver Blinker mit TeachIn-Modus mit Lauflängenspeicherung

```
52 Next A
53 For A = 1 To 3
54     Portb.3 = 1
55     Waitms 150                                '3 x kurz blinken (Teach-In-Ende)
56     Portb.3 = 0
57     Waitms 150
58 Next A
59 Do
60     Gosub Led_abfrage
61     If Led_ladezustand = 0 Then
62         Hell = 1                                'Bei Licht Merker setzen
63     End If
64     If Hell = 1 And B < 255 Then
65         B = B + 1                                'Wenn Merker gesetzt wurde, Zähler für Lichtdauer erhöhen (bis max. 255)
66     End If
67     If Led_ladezustand = 1 And Hell = 1 And B < 50 Then
68         Gosub Blinken                            'Wenn es wieder dunkel ist und der Lichtimpuls nur kurz war, dann blinken
69     End If
70     If Led_ladezustand = 1 Then
71         Hell = 0
72         B = 0
73     End If
74 Loop
75
76 Led_abfrage:
77     Portb.3 = 0                                'Portb.3 auf Masse schalten
78     Portb.4 = 1                                'Portb.4 auf +Ub schalten, um die LED zu 'laden'
79     Waitus 1                                    'Ladezeit, kann ggf. noch verkleinert werden
80     Config Portb.4 = Input                      'Portb.4 nun zwecks Abfrage der LED-Ladung auf 'Eingang' schalten
81     Portb.4 = 0                                'Pullup abschalten, sonst geht's nicht!
82     Waitms 20                                  'Entladezeit 20 ms - je kleiner, je unempfindlicher
83     Led_ladezustand = Pinb.4                    'Ladezustand einlesen: '1' -> dunkel, '0' -> hell
84     Config Portb.4 = Output                    'Portb.4 wieder auf Ausgang schalten
85     Portb.4 = 0                                'Portb.4 auf Masse schalten
86 Return
87
88 Blinken:
89     Portb.3 = 0
90     Portb.4 = 0
91     Readeeprom Led_lichtcode , 1                'Blinksequenz mit EEPROM-Inhalt
92     If Led_lichtcode = 1 Then                  'Auslesen des LED-Startzustandes
93         Last = 1
94     Else
95         Last = 0
96     End If
97     For A = 2 To 63                            'Auslesen der gespeicherten Bytes
98         Readeeprom Led_lichtcode , A
99         If Led_lichtcode = 255 Then
100             Portb.3 = Last
101             For I = 1 To 254
102                 Waitms 90
103             Next I
104         Else
105             Portb.3 = Last
106             For I = 1 To Led_lichtcode
107                 Waitms 90
108             Next I
109             Last = Not Last
110         End If
111     Next A
112 Return
113
114 End
```

Dieses Programm ist eine Variation des Programms „Verbesserter nachtaktiver Blinker mit TeachIn-Modus“ aus Kapitel 6.1.4. Bevor das Programm in die Endlosschleife (Zeilen 59 - 74) geht, erfolgt in den Zeilen 25 - 58 die TeachIn-Prozedur. Zu Beginn wird Beleuchtungszustand in Byte 1 des EEPROMs gespeichert (Zeilen 31 - 35). Anschließend werden die folgenden 62 Bytes gefüllt. Dazu wird gezählt, über wieviel Abtastwerte sich der Beleuchtungszustand nicht ändert (Zeilen 40 - 44). Ändert sich der Zustand, wird die Länge des vergangenen Zustandes im EEPROM gespeichert (Zeile 50). Ist

die Länge länger als 254 Zyklen, so wird ein Byte im EEPROM auf 255 gesetzt und mit dem nächsten Byte weiter gemacht. Auf diese Weise ist eine Gesamtspeicherdauer von bis zu 20 Minuten möglich. Durch die Laufängencodierung können jedoch maximal 62 verschiedene Zustände gespeichert werden, wodurch im Allgemeinen die Speicherdauer erheblich reduziert wird.

Beim späteren Auslesen (Zeilen 91 - 111) wird zuerst der Startzustand der LED gelesen (Zeilen 91 - 96), anschließend byteweise das restliche EEPROM. Ist der Wert des Bytes kleiner als 255, so wird eine entsprechende Zeit gewartet und anschließend der Zustand der LED invertiert (Zeilen 105 - 109). Bei einem Wert von 255 entfällt das Invertieren (Zeilen 100 - 103).

## 6.1.6 Verbesserter nachtaktiver Blinker mit Morsezeichenausgabe

Bei diesem Programm wurde das Programm aus Kapitel 6.1.3 um eine Ausgabe von Morsezeichen erweitert.

```

1  ' *****
2  ' *** ***
3  ' *** Reaktiv-Tiny v0.1 24/11/2005 ***
4  ' *** ***
5  ' *****
6  $regfile = "Attiny13.DAT"
7  $crystal = 113000                'Reale Frequenz des internen 128kHz-Oszillators
8
9  $hwstack = 6                    'sram Compilerfehler vermeiden
10 Config Portb = &B00011000       'Pinb.3 und .4 auf 'Ausgang', Rest auf 'Eingang' schalten
11 Portb = &B11100111             'Pullups zuschalten, außer für Pinb.3 und .4
12 Stop Adc                       'A/D-Wandler abschalten, um Strom zu sparen
13 Stop Ac                         'Analog-Komparator abschalten, um Strom zu sparen
14
15 Dim A As Byte
16 Dim B As Byte
17 Dim C As Byte
18 Dim Led_ladezustand As Bit
19 Dim Hell As Bit
20 Dim Dunkel As Bit
21 Dim Morse$ As String * 10       '10 Takte, die gemorst werden sollen
22 Dim S$ As String * 1           'Merken zum Morsen (aktueller Takt)
23
24 Do
25     Gosub Led_abfrage
26     If Led_ladezustand = 0 Then
27         Hell = 1                'Bei Licht Merker setzen
28     End If
29     If Hell = 1 And B < 255 Then
30         B = B + 1              'Wenn Merker gesetzt wurde, Zähler für Lichtdauer erhöhen (bis max. 255)
31     End If
32     If Led_ladezustand = 1 And Hell = 1 And B < 30 Then 'Wenn es wieder dunkel ist dann nachsehen für wie lange
33         Dunkel = 1             'Merker Setzen
34         For C = 0 To 5         'LED-Zustand mehrmals abfragen
35             Gosub Led_abfrage
36             If Led_ladezustand = 0 Then
37                 Dunkel = 0     'Wenn wieder Hell dann Dunkel-Merker löschen
38             End If
39         Next C
40         If Dunkel = 1 Then
41             Gosub Morsen       'Wenn Dunkel-Merker gesetzt Morsen
42         End If
43     End If
44     If Led_ladezustand = 1 Then 'Bei Dunkelheit Merker und Zähler für Lichtdauer löschen
45         Hell = 0
46         B = 0
47     End If
48 Loop
49
50 Led_abfrage:

```

```

51     Portb.3 = 0           'Portb.3 auf Masse schalten
52     Portb.4 = 1         'Portb.4 auf +Ub schalten, um die LED zu 'laden'
53     Waitus 1            'Ladezeit, kann ggf. noch verkleinert werden
54     Config Portb.4 = Input 'Portb.4 nun zwecks Abfrage der LED-Ladung auf 'Eingang' schalten
55     Portb.4 = 0         'Pullup abschalten, sonst geht's nicht!
56     Waitms 20           'Entladezeit 20ms (100 ms) - je kleiner, je unempfindlicher
57     Led_ladezustand = Pinb.4 'Ladezustand einlesen: '1' -> dunkel, '0' -> hell
58     Config Portb.4 = Output 'Portb.4 wieder auf Ausgang schalten
59     Portb.4 = 0         'Portb.4 auf Masse schalten
60 Return
61
62 Morsen:
63     Morse$ = "--*--*--*" 'String aus 10 Zeichen * = kurz - = lang
64     For A = 1 To 10      '10 Zeichen auslesen und auswerten
65         S$ = Mid(morse$, A, 1) 'Teilstring auslesen
66         If S$ = "*" Then 'wenn kurz Blinken
67             Portb.3 = 1      'LED AN
68             Waitms 30        'kurz warten
69             Portb.3 = 0      'LED AUS
70             Waitms 1000     'lang warten
71         Else              'sonst lang blinken
72             Portb.3 = 1      'LED AN
73             Waitms 300       'lange warten
74             Portb.3 = 0      'LED AUS
75             Waitms 1000     'lang warten
76         End If
77     Next A
78 Return
79
80 End

```

Dieses Programm ist eine Variation des Programms „Verbesserter nachtaktiver Blinker“ aus Kapitel 6.1.3. Die Änderungen finden sich in den Zeilen 63 bis 78. Anstelle der normalen Blink-Prozedur wird hier ein Morsecode ausgegeben. Der Code wird in Zeile 63 in der Variablen `Morse$` gespeichert. Ein Bindestrich steht für ein langes Zeichen (dah), ein Stern für ein kurzes (dit). Die Länge der Variable kann in Zeile 21 definiert werden.

## 6.1.7 Verbesserter nachtaktiver Blinker mit Watchdog-Abschaltung

Bei diesem Programm wurde das Programm aus Kapitel 6.1.3 modifiziert, sodass die Ruhestromaufnahme auf nur  $5 \mu\text{A}$  gesenkt wurde. Das Programm wartet nicht mehr mit dem `wait`-Befehl, sondern wird mittels des Watchdog-timers für eine definierte in einen Ruhemodus versetzt. „Bascom AVR“ unterstützt allerdings (bis auf den Reset-Modus) die verschiedenen Betriebszustände nicht. Insofern müssen die Register manuell gesetzt werden.

```

1  ' *****
2  ' *** ***
3  ' *** Reaktiv-Tiny mit Watchdog-Abschaltung ***
4  ' *** ***
5  ' *****
6
7  $regfile = "ATtiny13.DAT"
8  $crystal = 16000           'Frequenz des internen Oszillators
9
10 Config Portb = &B00011000 'Pinb.3 und .4 auf 'Ausgang', Rest auf 'Eingang' schalten
11 Portb = &B11100111        'Pullups zuschalten, außer für Pinb.3 und .4
12 Stop Adc                  'A/D-Wandler abschalten, um Strom zu sparen
13 Stop Ac                   'Analog-Komparator abschalten, um Strom zu sparen
14
15 Wdtcr = &B11010011        'Watchdog definieren: 0.125 Sekunden, Interrupt auslösen, kein Reset
16 Enable Interrupts        'Interrupts freigeben
17
18 Dim A As Byte

```

```

19 Dim B As Byte
20 Dim C As Byte
21 Dim Led_ladezustand As Bit
22 Dim Hell As Bit
23 Dim Hell_2 As Byte
24
25 Do
26     Gosub Led_abfrage
27     If Led_ladezustand = 0 Then
28         Hell = 1           'Bei Licht Merker setzen
29     End If
30     If Hell = 1 And B < 255 Then
31         B = B + 1         'Wenn Merker gesetzt wurde, Zähler für Lichtdauer erhöhen (bis max. 255)
32     End If
33     If Led_ladezustand = 1 And Hell = 1 And B < 30 Then 'Wenn es wieder dunkel ist und der Lichtimpuls nur kurz war
34         Hell_2 = 0       'zweiten Hell-Merker setzen
35         For C = 0 To 5   'und 5 mal abfragen
36             Gosub Led_abfrage
37             If Led_ladezustand = 0 Then
38                 Hell_2 = Hell_2 + 1 'ob es auch wieder dunkel ist
39             End If
40         Next C
41         If Hell_2 = 0 Then
42             Gosub Blinken   'erst dann blinken
43         End If
44         Hell = 0
45         B = 0
46     End If
47     If Led_ladezustand = 1 Then 'Bei Dunkelheit Merker und Zähler für Lichtdauer löschen
48         Hell = 0           'damit sich das Programm nicht aufhängt
49         B = 0
50     End If
51 Loop
52
53 Led_abfrage:
54     Portb.4 = 1           'Portb.4 auf +Ub schalten, um die LED zu 'laden'
55     Waitus 1             'Ladezeit, kann ggf. noch verkleinert werden
56     Config Portb.4 = Input 'Portb.4 nun zwecks Abfrage der LED-Ladung auf 'Eingang' schalten
57     Portb.4 = 0         'Pullup abschalten, sonst geht's nicht!
58     Reset Watchdog     'Watchdog zurücksetzen dass Controller rechtzeitig aufwacht
59     Powerdown          'Während des Entladens den Controller Schlafen schicken
60     Led_ladezustand = Pinb.4 'Ladezustand einlesen: '1' -> dunkel, '0' -> hell
61     Config Portb.4 = Output 'Portb.4 wieder auf Ausgang schalten
62 Return
63
64 Blinken:
65     For A = 1 To 10
66         Portb.3 = 1
67         Reset Watchdog
68         Powerdown
69         Portb.3 = 0
70         Reset Watchdog
71         Powerdown
72     Next A
73 Return
74
75 End

```

Dieses Programm ist eine Variation des Programms „Verbesserter nachtaktiver Blinker“ aus Kapitel 6.1.3. Anstelle des wait-Befehls wird hier allerdings der Watchdog-Timer der Microcontroller in einen Stromsparmmodus gesetzt, aus dem er nach einer definierten Zeit aufwacht. In Zeile 15 wird der Watchdog-Timer konfiguriert. Die Betriebsart ist „Interrupt“, die Wartezeit 0,125 Sekunden. Die möglichen Wartezeiten sind in Tabelle 6.1 aufgelistet. Außerdem werden in Zeile 16 die Interrupts freigeschaltet, die benötigt werden, damit der Controller nach Ablauf der Zeit wieder aktiv wird.

Soll der Controller warten, wird zuerst der Watchdog-Timer zurückgesetzt (Zeile 67). Die eingestellte Zeit beginnt zu laufen. In Zeile 68 wird nun der Controller in den Power-Down-Modus versetzt. Ist der Timer abgelaufen, wird ein Interrupt erzeugt, das den Controller wieder in den normalen Modus

Wartezeit	Wdter
16 ms	&Bxx0xx000
32 ms	&Bxx0xx001
64 ms	&Bxx0xx010
0,125 s	&Bxx0xx011
0,25 s	&Bxx0xx100
0,5 s	&Bxx0xx101
1,0 s	&Bxx0xx110
2,0 s	&Bxx0xx111
4,0 s	&Bxx1xx000
8,0 s	&Bxx1xx001

Tabelle 6.1: Mögliche Wartezeiten für den Watchdog-Timer.

wechseln lässt.

## 6.2 Programme für die Grundschtung aus Kapitel 4.2.1

### 6.2.1 Verbesserter Nachtaktiver Blinker mit Watchdog-Abschaltung

Dieses Programm ist identisch mit dem aus Kapitel 6.1.7. Lediglich ein Schlafmodus wurde hinzugefügt, der den Prozessor für 8 Sekunden ausschaltet, wenn Tag detektiert wurde.

```

1  ' *****
2  ' *** ***
3  ' *** Reaktiv-Tiny mit Watchdog-Abschaltung und LDR ***
4  ' *** ***
5  ' *****
6
7  $regfile = "ATtiny13.DAT"
8  $crystal = 16000                'Frequenz des internen Oszillators
9
10 Config Portb = &B00001001      'Pinb.0 und 3 auf 'Ausgang', Rest auf 'Eingang' schalten
11 Portb = 0
12 Stop Adc                       'A/D-Wandler abschalten, um Strom zu sparen
13 Stop Ac                         'Analog-Komparator abschalten, um Strom zu sparen
14
15 Wdter = &B11010011             'Watchdog definieren: 0.125 Sekunden, Interrupt auslösen, kein Reset
16 Enable Interrupts              'Interrupts freigeben
17
18 Dim A As Byte
19 Dim B As Byte
20 Dim C As Byte
21 Dim Ldr As Bit
22 Dim Hell As Bit
23 Dim Hell_2 As Byte
24
25 Do
26     Gosub Ldr_abfrage
27     If Ldr = 1 Then
28         Hell = 1                'Bei Licht Merker setzen
29     End If
30     If Hell = 1 And B < 255 Then
31         B = B + 1              'Wenn Merker gesetzt wurde, Zähler für Lichtdauer erhöhen (bis max. 255)
32     End If
33     If B > 200 Then
34         Gosub Abschalten      'Schlafmodus wenn es zu lange am Stück hell ist
35     End If

```

```

36     If Ldr = 0 And Hell = 1 And B < 30 Then 'Wenn es wieder dunkel ist und der Lichtimpuls nur kurz war
37         Hell_2 = 0           'zweiten Hell-Merker setzen
38         For C = 0 To 5       'und 5 mal abfragen
39             Gosub Ldr_abfrage
40             If Ldr = 1 Then
41                 Hell_2 = Hell_2 + 1 'ob es auch wieder dunkel ist
42             End If
43         Next C
44         If Hell_2 = 0 Then
45             Gosub Blinken     'erst dann blinken
46         End If
47         Hell = 0
48         B = 0
49     End If
50     If Ldr = 0 Then           'Bei Dunkelheit Merker und Zähler für Lichtdauer löschen
51         Hell = 0             'damit sich das Programm nicht aufhängt
52         B = 0
53     End If
54 Loop
55
56 Ldr_abfrage:
57     Portb.0 = 1               'Spannung auf LDR geben
58     Reset Watchdog           'Kurz warten bis sich alles eingeschwungen hat
59     Powerdown
60     Ldr = Pinb.4              'LDR abfragen
61     Portb.0 = 0              'Spannung von LDR wieder wegnehmen
62 Return
63
64 Blinken:                      'LED blinken lassen
65     For A = 1 To 10
66         Portb.3 = 1
67         Reset Watchdog
68         Powerdown
69         Portb.3 = 0
70         Reset Watchdog
71         Powerdown
72     Next A
73 Return
74
75 Abschalten:                    'Prozessor für 8 Sekunden schlafen lassen
76     Wdtcr = &B11110001
77     Reset Watchdog
78     Powerdown
79     Wdtcr = &B11010011
80 Return
81
82 End

```

In Zeile 33 wird abgefragt, ob der Fotowiderstand länger als 200 Zyklen hell detektiert hat. Ist dies der Fall, so wird in den Zeilen 76 - 79 mittels des Watchdog-Timers der Controller für 8 Sekunden in den Ruhezustand versetzt.

Bedingt durch den anderen Sensor für den Beleuchtungszustand hat sich auch die Prozedur der Helligkeitsabfrage (Zeilen 56 - 62) geändert. Zuerst wird der Ausgang PB0 auf High gesetzt. Nach einer kurzen Wartezeit hat sich alles eingeschwungen, sodass über PB4 die Helligkeit eingelesen werden kann. Anschließend wird der Spannungsteiler wieder stromlos geschaltet.

## 6.3 Programme für die Grundschtung aus Kapitel 4.2.2

### 6.3.1 Nachtaktiver Blinker mit A/D-Wandler

Bei dieser Variante wird der Spannungsteiler dauerhaft mit Strom versorgt. Durch seine hochohmigkeit wird der Stromverbrauch nicht wesentlich erhöht. Darüber hinaus wird die Helligkeit nicht als



hell/dunkel detektiert, sondern über den A/D-Wandler eingelesen. Dadurch lassen sich bis zu 1024 unterschiedliche Helligkeitsstufen erkennen.

```

1  ' *****
2  ' *** ***
3  ' *** Tiny-Reaktivlicht mit LDR und A/D-Wandler ***
4  ' *** mit Watchdog-Energiesparmodus und Tagabschaltung ***
5  ' *** ***
6  ' *****
7
8  $regfile = "ATtiny13.DAT"
9  $crystal = 16000                                'Frequenz des internen Oszillators
10 $hwstack = 2                                    'hardwarestack herabsetzen damit genügend variablen zur verfügung stehen
11
12 Config Adc = Single , Prescaler = Auto
13 Config Portb = &B00001000                        'Pinb.3 auf 'Ausgang', Rest auf 'Eingang' schalten
14 Portb = 0                                        'Ausgänge auf Low setzen
15 Stop Ac                                          'Analog-Komparator abschalten, um Strom zu sparen
16
17 Wdtcr = &B11010011                               'Watchdog definieren: 0.125 Sekunden, Interrupt auslösen, kein Reset
18 Enable Interrupts                               'Interrupts freigeben
19
20 Const Schwelle = 50                              'je größer der Schwellwert, desto unempfindlicher
21 Const Tagschwelle = 800                          'Schwellwert für Schlafmodus
22 Const Zwangsimpuls = 8                           'LED-Impuls tagsüber alle X Schlafzyklen (á ca. 8 Sekunden)
23 Dim A As Byte                                    'Variablen definieren
24 Dim Tagzaehler As Byte
25 Dim Schlafzaehler As Byte
26 Dim Ldr As Integer                               '0 = Dunkel, 1023 = Hell
27 Dim Alt As Integer
28 Dim Merker As Integer
29
30 Do
31     Reset Watchdog
32     Powerdown                                    'prozessor bremsen da sonst lichtänderung nicht erkannt wird
33     Start Adc                                    'A/D-Wandler starten
34     Ldr = Getadc(2)                              'Helligkeitswert einlesen
35     Stop Adc                                     'A/D-Wandler zum Stromsparen wieder stoppen
36     Merker = Ldr - Alt                           'Unterschied zwischen letzter und aktueller Messung ermitteln
37     Alt = Ldr                                    'letzten LDR-Wert sichern
38     If Merker > Schwelle Then
39         Gosub Blinken                            'Bei großer Änderung Dunkel->Hell: Blinken
40     End If
41     If Ldr > Tagschwelle Then                    'prüfen ob helligkeit über tagschwelle liegt
42         If Tagzaehler < 255 Then                'int-variable geht nur bis 255
43             Tagzaehler = Tagzaehler + 1
44         End If
45     Else
46         Tagzaehler = 0                          'wenn wieder dunkel tagzähler löschen
47     End If
48     If Tagzaehler > 200 Then
49         Gosub Pause                              'wenn mehr als x zyklen hell dann schlafmodus
50     End If
51 Loop
52
53 Blinken: 'LED blinken lassen
54     For A = 0 To 10
55         Portb.3 = 1
56         Reset Watchdog
57         Powerdown
58         Portb.3 = 0
59         Reset Watchdog
60         Powerdown
61     Next A
62     Alt = 1023                                  'Doppelauslösung verhindern
63 Return
64
65 Pause:
66     Wdtcr = &B11110001                          'Watchdog auf 8 Sekunden stellen
67     Reset Watchdog
68     Powerdown
69     Wdtcr = &B11010011                          'Watchdog wieder auf 0,125 Sekunden zurückstellen
70     Schlafzaehler = Schlafzaehler + 1           'merken wie oft Schlafmodus durchlaufen wurde

```

```
71     If Schlafzaehler = Zwangsimpuls Then 'als Funktionskontrolle tagsüber LED auslösen
72         Portb.3 = 1
73         Reset Watchdog
74         Powerdown
75         Portb.3 = 0
76         Schlafzaehler = 0
77     End If
78 Return
79
80 End
```

In Zeile 12 wird der A/D-Wandler konfiguriert. Beim Einlesen der Helligkeit (Zeilen 33 - 35) wird zuerst der A/D-Wandler gestartet, danach der Helligkeitswert in eine Integer-Variable gespeichert und anschließend der A/D-Wandler wieder gestoppt. Ist die Änderung der Helligkeit im Vergleich zum letzten Zyklus größer als der Wert **Schwelle**, so wird die Prozedur Blinken aufgerufen (Zeile 39).

Ist der eingelesene Helligkeitswert größer als der Wert **Tagschwelle**, so wird angenommen, dass Tag ist, und eine Variable wird inkrementiert (Zeilen 41 - 47). Hält dieser Zustand länger als 200 Zyklen an (Zeile 48), so wird der Controller in den Ruhemodus versetzt (Zeilen 66 - 77). Wurde dies häufiger gemacht, wird ein Blinken der LED ausgegeben als Funktionskontrolle (Zeilen 70 - 77).

# Kapitel 7

## Problemlösungen

**Ist es normal, dass der ATtiny13V beim Anlegen von 3 V relativ schnell heiß wird?**

- Nein, dies ist ein Microcontroller und kein Intel-PC.
- Wenn die Schaltung auf einem Steckbrett aufgebaut ist, alles bis auf die Stromversorgung abtrennen. Wird der Chip immer noch heiß, ist er defekt. Wenn er kalt bleibt, Schritt für Schritt wieder die volle Schaltung aufbauen.
- Ist der Prozessor richtig herum eingesteckt? Pin 1 ist derjenige mit der Kerbe im Gehäuse. Danach wird gegen den Uhrzeigersinn gezählt (siehe Kapitel 4).
- Sind mehrere Beinchen, die nicht zusammengehören, innerhalb einer durchkontaktierten Reihe im Steckbrett oder ist das Steckbrett defekt?

**Der Mikroprozessor ist kaputt.**

- Schon kurzes Verpolen der Versorgungsspannung kann dem Prozessor ein schnelles Ende bereiten. Da hilft nur ein neuer.

**Fehlermeldung „The HW-Stack, SW-Stack and frame space may not exceed the chip memory“.**

- Bei einem frisch installierten Programm muss man zuerst auf „Neue Datei“ klicken. Danach funktioniert es.

**Fehlermeldung „Could not identify chip with IDE: ...“ beim Setzen der Fuse-Bits.**

- Kompilieren des Quelltextes hat nicht funktioniert. Fehler entfernen und neu kompilieren, dann wird der Chip wieder erkannt.
- Ein weiterer Grund kann in der externen Beschaltung der Pins des Chips, die für die Programmierung benötigt werden, liegen.

### Fehlermeldung „Out of SRAM space“.

- Bei „Bascom AVR“ sind die Stacks standardmäßig zu hoch eingestellt. Unter „Options“, „Programmer“ im Register den „HW Stack“ runterstellen. Diese Stack-size ist von Haus aus auf 32 Byte eingestellt. Je gosub werden allerdings nur 2 Byte benötigt. Alternativ können auch folgende Zeile in das Programm eingefügt werden:

```
$hwstack = 6
```

### Fuse-Bits.

- Nicht die Fuse-Bits vergessen! Einstellungen siehe Kapitel 5.2.

### LED leuchtet nicht.

- Ist die LED richtig herum gepolt?
- Manchmal muss das Programmierkabel abgeklemmt werden, damit das Programm anläuft.

### Wenn an den Chip vom Strom nimmt und später wieder anschließt, läuft das Programm nicht automatisch an.

- Unter <http://www.mikrocontroller.net/tutorial/equipment> kann man eine Erweiterung für einen Reset beim Start sehen. an dem Reset-Pin liegt ein 10 k $\Omega$  Widerstand gegen die Versorgungsspannung und ein 47 nF Kondensator gegen Masse. Die Ladezeit des Kondensators beträgt damit 2,35  $\mu$ s, was einen sicheren Reset gewährleisten sollte.
- Der Programmieradapter zieht zwischendurch manchmal den Reset-Pin auf Masse bzw. lässt ihn nach dem Programmieren nicht wieder los. Dagegen hilft ein 10 bis 15 k $\Omega$  Widerstand von Reset an Versorgungsspannung (Pin 1 auf Pin 8). Den braucht man dann aber später bei der autarken Schaltung nicht mehr.

### Nach dem Setzen des Fuse-Bits 3 auf „External Reset Disable“ hat man keinen Zugriff mehr auf den Chip. Der Programmer kann den Chip nicht mehr identifizieren.

- Der Chip ist nicht kaputt, bekommt aber wegen des umprogrammierten Fuse-Bits nicht mehr das Reset des Programmieradapters mit. Im High-Voltage-Modus kann die Fuse wieder zurückprogrammiert werden. Dabei werden 12 V an den Reset-Pin gelegt.

### Ist es möglich, den Tiny voll beschaltet zu programmieren?

- Ja. Das Programmierinterface nennt sich ISP (In System Programmable), d. h. man kann den Tiny programmieren, während er in der Schaltung steckt. Das funktioniert aber natürlich nur dann, wenn die Ports, an denen der Tiny programmiert wird, hardwaremäßig korrekt verschaltet sind. Hängen z. B. LEDs dran, blinken diese während des Programmiervorgangs. Wenn man diese

Ports normalerweise als Eingangsports benutzt und womöglich permanent ein Low- oder High-Signal abliegen hat, geht es natürlich nicht. Dann muss man den Tiny aus der Schaltung nehmen, um ihn zu programmieren.

## Kapitel 8

# Bestelldaten

Wer der Meinung ist, dass er trotz der Liste Hilfe benötigt, darf mich kontaktieren. Gegen Übernahme der Portokosten bin ich bereit, Euch die benötigten Bauteile zum Umkostenpreis zu überlassen. Kontaktiert mich dafür unter [impressum@reaktivlicht.de](mailto:impressum@reaktivlicht.de).

Bauteil	Reichelt
ATtiny 13V-10PU ATtiny 13V-10SU (für SMD-Bestückung) Sockel DIL 8-polig	ATTINY 13V-10PU ATTINY 13V-10SU GS6
Widerstand 56 $\Omega$ Widerstand 220 $\Omega$ Widerstand 1 M $\Omega$	1/4 W 56 $\Omega$ 1/4 W 220 $\Omega$ 1/4 W 1 M
Kondensator 100 nF	MKS-2 100N
Leuchtdiode grün 13000 mcd Leuchtdiode weiß 18000 mcd	LED 5-13000 GN LED 5-18000 WS
SUB-D Stecker 25 polig Kappe für Stecker	D-SUB ST 25 KAPPE CG25G
Fotowiderstand	A 905014 A 906014

Tabelle 8.1: Bestelldaten der Bauteile

# Literaturverzeichnis

- [1] P. Dietz, W. Yerazunis, D. Leigh, *Very Low-Cost Sensing and Communication Using Bidirectional LEDs.*