Ralf Pongratz

The Cookbook

Reactive Lights with the ATtiny13A

www.reaktivlicht.de

Ralf Pongratz

The Cookbook

Reactive Lights with the ATtiny13A

23. Dezember 2013

www.reaktivlicht.de

Inhaltsverzeichnis

1

Basic Information

1.1 Programming

1.1.1 Programming Device

To flash the program to the microcontroller, a port at the computer is needed. The easiest way is to use the parallel port for programming. Figure 1.1 shows a simple pogramming device.

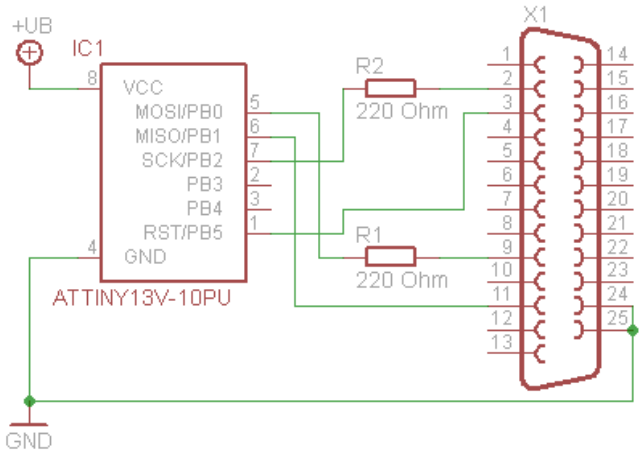

Abb. 1.1. Programming device for the parallel port.

X1 is a 25 pin SUB-D plug that is connected to the parallel port of the computer. IC1 is the microcontroller to be programmed. Moreover two resistors with a value of 220 Ohm are needed. The power comes from an external power

supply. The cable to the computer has to be as short as possible and must be shielded to prevent noise from coupling into the line.

As long as pin 1 is not used and the pins 5, 6 and 7 are not connected to a fixed voltage or among each other, the programming device can be integrated into the reactive light. The advantage is that you don't have to switch the microcontroller during programming.

1.1.2 Programming Software

Several programs exist for programming microcontrollers. In this tutorial the program „Bascom AVR" with the parallel programming device (chapter 1.1.1) is used. A demo version with a limited code size (4 kB) is available in the internet[1] for free.

Requirements

A computer with a parallel port (printer port), the parallel programming device and the programming software, that can be used for writing the programs and transfering them to the controller, are needed.

The parallel port must be set in the BIOS of the computer to ECP+EPP (In- and Output). Here an example for the settings:
Onboard Parallel Port: 378/IRQ7
Parallel Port Mode: ECP+EPP
ECP Mode Ust DMA: 3
Parallel Port EPP Type: EPP1.7

Handling of „Bascom AVR"

All instructions are based on the program version 1.11.8.3. Newer versions can be different.

At the beginning „Bascom AVR" has to be started and in the menu „File" with „New" a new program window created. After that the programming device and the chip parameters are set. Do this by selecting „Programmer" in the menu „Options".

Now select the tab „Compiler" and then „Chip" (figure 1.2). Choose the entry „attiny13.dat" from the drop down menu „Chip" and insert these values in the text fields below:

[1] `http://www.mcselec.com/index.php?option=com_docman&task=cat_view&gid=99&Itemid=54`

HW Stack = 2
Soft Stack = 8
Framesize = 24
After that save the values by clicking „Default“.

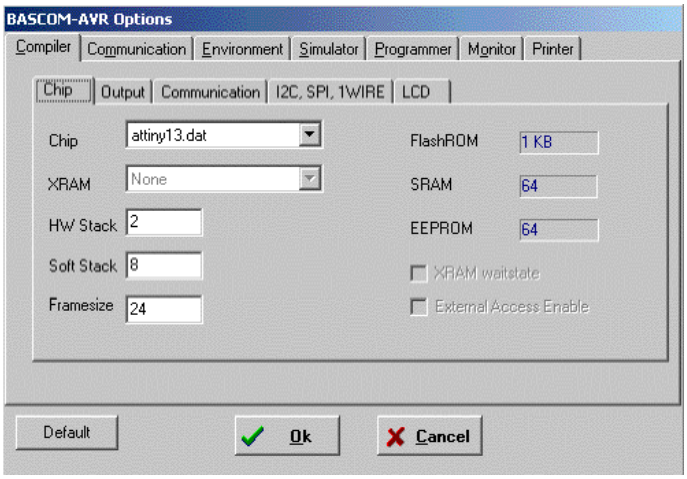

Abb. 1.2. Setting the chip parameters.

Then go to the tab „Programmer“ and select the entry „Universal MCS Interface“ from the drop down menu „Programmer“ (figure 1.3). In the tab „Universal“ the programmer „WinAVR and SP12“ has to be selected. Then close the dialog by clicking „OK“.

Next are the fuse bits. That are memory cells that define the basic behavior of the microcontroller. Click on the small green IC socket in the menu bar and select „Manual Program„ (Figure 1.4).

In the dialog select the tab „Lock and Fuse Bits“. Now Bascom reads the settings of the fuse bits out of the controller and shows them (Figure 1.5).

To change the fuse bits click on the according row that contains the bit to be edited. Select the desired value in the drop down menu. These settings has to be done:
Fusebit DCBA to „1011:Int. Osc. 128 kHz; start-up time: 14 CK + 64 ms“
Fusebit E to „1:Divide clock by 8, OFF“

Please check the settings carefully. If everything is right, click „Write FS“. The changes wll be written to the controller.

Now the program can be written. When it is done, it has to be compiled. Select „Compile“ from the menu „Program“ (Figure 1.6). The compiler starts to work. If the program contains errors, an error message will be shown in

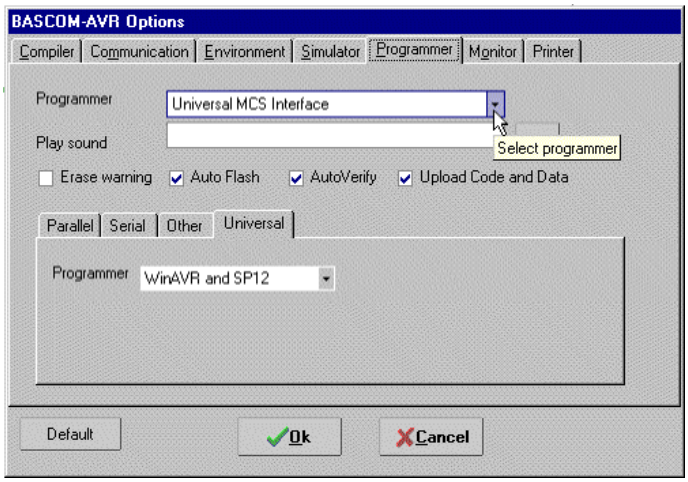

Abb. 1.3. Setting the programming device.

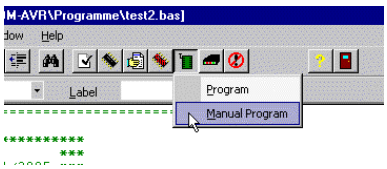

Abb. 1.4. Setting the fuse bits.

the footer. After being compiled successfully, the program can be transfered to the microcontroller. Go to the menu „Program“ and select „Send to Chip“. In the dialog click to „Autoprogram“ in the menu „Chip“ (Figure 1.7). The program will be transfered to the controller now.

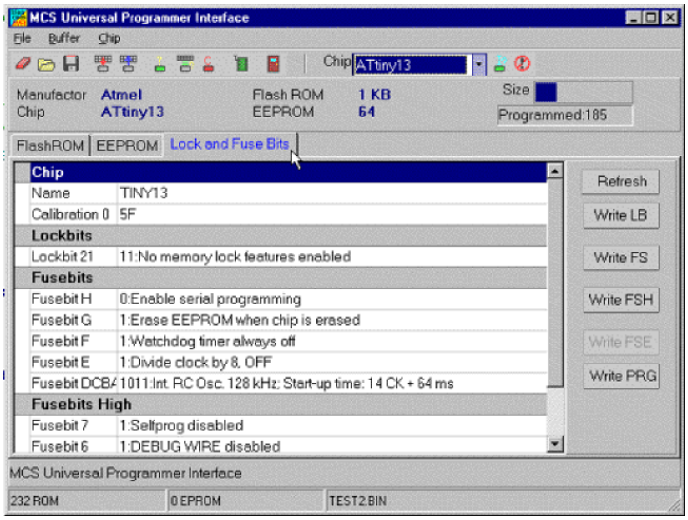

Abb. 1.5. Setting the fuse bits.

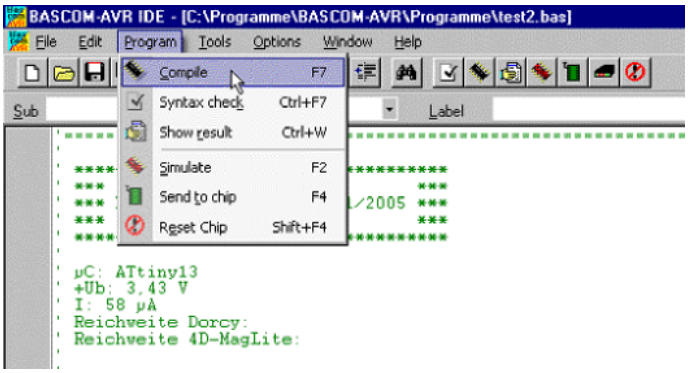

Abb. 1.6. Compiling the program.

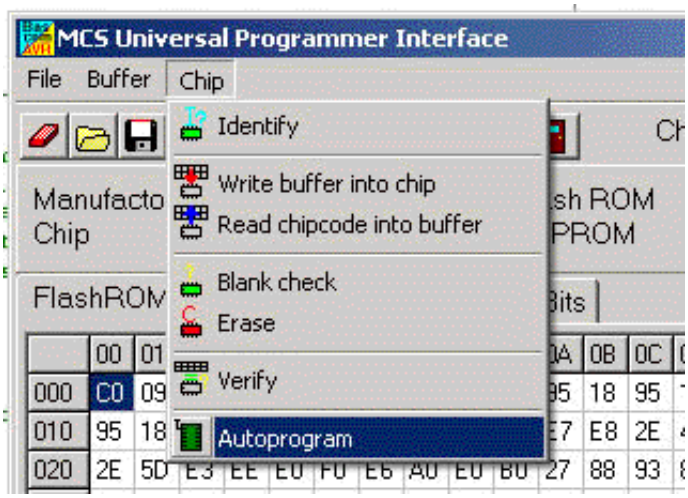

Abb. 1.7. Transfering the program.

2

Device and Program Structure

A reactive light generally consists of three different kinds of elements. The central part is the microcontroller that processes the input signals of one or more sensors and controls the actors (Figure 2.1). In seldom cases several microcontrollers that communicate to each other are used. But that is rather the exception. The individual elements are explained in the following chapters.

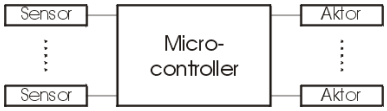

Abb. 2.1. Schematic structure of a reactive light.

2.1 Sensors

The use of sensors is to collect data that are processed in the microcontroller. These data can trigger events or change states of the program (toggle between day and night mode, activation during certain times).

2.1.1 Potential divider with a photoresistor

A photoresistor (Figure 2.2) is the easiest way to get information about the brightness. Its resistance is low if light falls onto the sensor, otherwise high. Using a potential divider this change of resistance can be converted into a voltage that can be read into the microcontroller by the analog digital converter. While the resistance in bright situation is about several kiloohm, it increases

in dark situation to several megaohm[1]. One has to be aware of the response time. In the data sheet generally the resistance after one minute darkness (R_{01}) and after five minutes darkness (R_{05}) are listed.

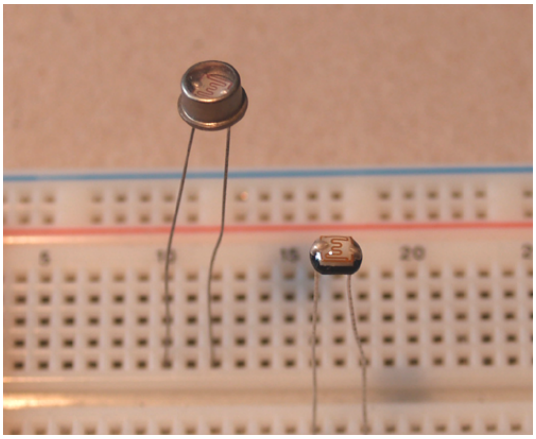

Abb. 2.2. Photoresistors. Left type A 1060, right A 9050 14.

Connected with a second (fixed value) resistor R_2 a potential divider can be contructed (Figure 2.3). Using a supply voltage U_B an output voltage

$$U_S = U_B \cdot \frac{R_2}{R_1 + R_2}. \tag{2.1}$$

results.

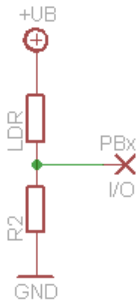

Abb. 2.3. Potential divider with a photoresistor. PBx is the input of the microcontroller.

[1] The data sheet of the used photoresistor A 9050 14 can be downloaded at `http://www.perkinelmer.de/CMSResources/Images/44-3571DTS_PhotocellsA9050.pdf`.

Hence a bright photoresistor results into a high output voltage, a dark one into a low. To have a high sensitivity the value of R_2 should be in the middle of the resistance in bright and dark situations of R_1.

```
1 Config Adc = Single , Prescaler = Auto
2 Config Portb = &B00x0xxxx
3 Portb = &Bxxx0xxxx
4
5 Dim Ldr As Integer
6
7 Start Adc
8 Ldr = Getadc(2)
9 Stop Adc
```

Line 1 configures the analog digital converter that is used to measure the output voltage of the potential divider. In this example it is connected to PB4 (pin 3). Therefore in line 2 and 3 PB4 is defined as input. The bits marked with a 'x' can have arbitrary values depending on the further wiring. In line 5 the integer variable is defined that will contain the value of brightness.

The reading of the brightness happens in line 8. `Ldr` will have a value between 0 and 1023 depending on the brightness. In line 7 and 9 the analog digital converter is switched on and off again. If the power consumption is not important, these commands can be left.

2.1.2 Capacitive Character of a Light Emitting Diode

A light emitting diode (LED) used in inverse direction takes effect like a small capacitor whose capacity depends on the brightness. In bright situations it is smaller than in dark ones[1]. The capacity underlies a self-discharge.

For measuring the brightness the capacity is loaded and the power supply interrupted. After a certain time the remaining voltage is measured. Because of the self-discharge it will be smaller than the originally loaded voltage. The bigger the capacity (hence the lower light falls onto the LED) the higher is the remaining voltage.

Because of the small angle of radiation it is guaranteed that the signal of the sensor just changes its value if the light falls directly onto the sensor. Super bright red LEDs with a clear body turned out to be the best ones for this purpose.

```
1 Config Portb = &B00x11xxx
2 Portb = &Bxxx00xxx
3
4 Dim Led As Bit
```

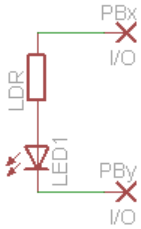

Abb. 2.4. Measuring the brightness using the capacitiy of a light emitting diode. PBx and PBy are two in- and outputs of the microcontroller.

```
 5
 6  Portb.3 = 0
 7  Portb.4 = 1
 8  Waitus 1
 9  Config Portb.4 = Input
10  Portb.4 = 0
11  Waitus 1500
12  Led = Pinb.4
13  Config Portb.4 = Output
14  Portb.4 = 0
```

Line 1 and 2 define PB3 and PB4 (pin 2 and 3) as outputs and set them to low. The variable `Led` in line 4 contains the information whether 'bright' or 'dark' was detected.

The lines 6 to 8 are loading the capacity of the LED: PB3 is set to ground, PB4 to $+U_B$. After some time the capacity is loaded. Now PB4 is defined as an input and set to 0 to stop loading. After a delay (line 11) thw status of the input is read and stored in `Led`. Afterwards PB4 is configured as an output again and set to ground. Now the LED can be switched on by setting PB3. Changing the delay time in line 11 changes the threshold.

2.2 Actuators

Using actuators the reactive light sends out signals. The most common one is a light emitting diode. An infrared LED or the activation of external devices is possible as well.

2.2.1 Light signals with a LED

The easiest way for the microcontroller to send out signals is a LED. Together with a resistor it is connected to one of the outputs of the microcontroller (Figure 2.5). The value of the resistor R_1 is

$$R_1 = \frac{U_B - U_F}{I_F}. \tag{2.2}$$

Thereby U_B is the supply voltage of the microcontroller, U_F the voltage drop across the LED and I_F the maximum current[2]. The maximum current an output provides is 40 mA.

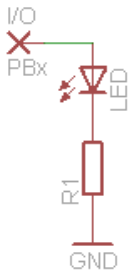

Abb. 2.5. Output of light signals by a LED. PBx is the output of the microcontroller.

```
1 Config Portb = &B00xx1xxx
2 Portb = &Bxxxx0xxx
3
4 Portb.3 = 1
5 Portb.3 = 0
```

In the first line PB3 (pin 2) is configured as an output. By setting the output bit to 1 (line 4) the LED is switched on. Setting it to 0 (line 5) switches it off again.

2.2.2 Switching loads

The maximum current that a output provides is 40 mA. In order to switch loads that need higher current, components to amplify the current has to be used.

[2] U_F and I_F can be found in the data sheet of the LED. Typically I_F is 20 mA, U_F depends on the type of LED.

Switching by a relay

The easiest way is to switch the load by a relay. Reedrelays should be preferred to normal ones because of the lower current needed for switching. But the current that can be used by the load is lower as well. In Figure 2.6 a schematic is shown. On the left is the switching circuit with the coil of relay K1, on the right the load and the switch of the relay.

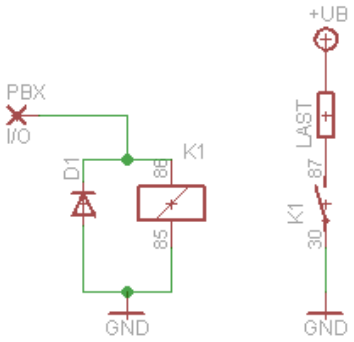

Abb. 2.6. Switching loads by a relay. PBx is the output of the microcontroller.

In parallel to the relay a diode in inverse direction has to be placed. When switching off the relay the collapsing magnetic field of the relay coil inducts a voltage that can destroy the output of the microcontroller. The diode releases the voltage riskless. A common diode (e. g. 1N4148) suffices.

Keep in mind that the current to switch the relay is not allowed to be greater than 40 mA. Futher the current the relay is able to switch has to be greater than the current of the load.

Switching by a transistor

A complicated but space saving possibility is to switch the load by a transistor. Figure 2.7 shows the schematic. An arbitrary NPN transistor can be used whose parameters fit to the load.

The current that has to go into the base of the transistor has to be

$$I_B = \frac{I_{Load}}{h_{FE}}, \tag{2.3}$$

in which I_{Load} the load current and h_{FE} the amplification factor is. The amplification factor can be found in the data sheet. For safety the shown

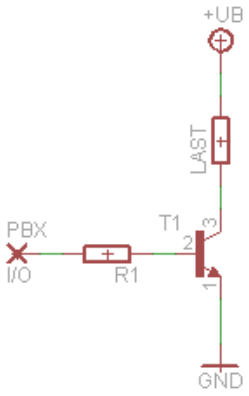

Abb. 2.7. Switching loads by a transistor. PBx is the output of the microcontroller.

minimum value should be used. It is important that the base current I_B is less than 40 mA. Otherwise a transistor with a bigger amplification factor has to be chosen.

For setting the base current a resistor R_1 is needed. Its value is

$$R_1 = \frac{U_B - U_{BE}}{I_B}. \tag{2.4}$$

U_B is the supply voltage of the microcontroller and U_{BE} the saturation voltage between base and emitter of the transistor[3].

When choosing the transistor one has to take care that the maximum collector current I_C has to be greater than the load current. Further each transtor has a leak current I_{CES}, that flows permanently between collector and emitter. If the device will be powered by batteries, it should be as small as possible.

2.3 Verarbeitung

Um die Signale der Sensoren mit den Ausgängen zu verknüpfen, muss ein Programm für den Microcontroller geschrieben werden.

[3] A silicon transistor normally has a U_{BE} of 0.7 V.

2.3.1 Grundgerüst

Die einfachste Möglichkeit besteht aus einer Endlosschleife, in der eine Fallunterscheidung steht. Dieses Grundgerüst stellt schon ein lauffähiges Programm dar, das auch in jeder umfangreicheren Variante wiederzufinden ist.

```
1  Dim hell As Bit
2
3  Do
4    Gosub Sensorabfrage
5
6    If hell = 1 Then
7      ' Hier kann z. B. eine LED aufblinken
8    End If
9  Loop
10
11 Sensorabfrage:
12   ' Hier wird der Beleuchtungszustand des
13   ' Sensors abgefragt. Ist er beleuchtet,
14   ' wird die Variable hell auf true gesetzt,
15   ' andernfalls auf false.
16   return
```

In Zeile 1 wird eine Variable definiert, in der der Zustand des Sensors gespeichert wird. Die Abfrage findet in der Unterroutine `Sensorabfrage` statt. Je nach Sensor ist hier eine andere Routine zu wählen. Welche Routine zu welchem Sensor passt, ist dem Kapitel 2.1 zu entnehmen.

Zeile 3 und 9 stellen die Endlosschleife dar. Der dazwischenstehende Code wird kontinuierlich ausgeführt. Zuerst wird in Zeile 4 der Zustand des Sensors abgefragt. Wenn er beleuchtet ist, leitet die if-Abfrage (Zeile 6) weiter zur Aktion. Verschiedene Möglichkeiten sind in Kapitel 2.2 aufgelistet.

2.3.2 Nachtaktives Programm

Damit das Reaktivlicht nur nachts ausgelöst werden kann und tagsüber nicht dauerhaft blinkt, soll Tageslicht erkannt werden. Dies geschieht, indem die Beleuchtungsdauer gemessen wird. Überschreitet sie eine festgelegte Dauer, so wird angenommen, dass Tag ist und das Licht blinkt nicht.

```
1  Dim cntH As Byte
2  Dim cntD As Byte
3  Dim lst As Bit
4
5  Do
```

```
 6   Gosub Sensorabfrage
 7
 8   If hell = 1 And cntH < 255 Then
 9     cntH = cntH + 1
10   End If
11
12   If hell = 0 And cntD < 255 Then
13     cntD = cndD + 1
14   End If
15
16   If hell = 0 Then
17     If lst = 1 And cntH < 50 And cntD > 5 Then
18       ' Hier kann z. B. eine LED
19       ' aufblinken.
20       cntH = 0
21       lst = 0
22     End If
23   Else
24     cntD = 0
25     lst = 1
26   End If
27 Loop
```

Es werden drei weitere Variablen `cntH`, `cntD` und `lst` eingeführt. In der ersten werden die Zyklen gezählt, in der der Sensor den Zustand hell detektiert. Sie wird in den Zeilen 8-10 inkrementiert, wenn der Sensor hell detektiert hat und die Variable noch nicht an ihrer Überlaufgrenze[4] angekommen ist. Die zweite Variable zählt die Zyklen, in denen der Sensor den Zustand dunkel detektiert. Sie wird in den Zeilen 12-14 inkrementiert. Die dritte Variable speichert den Beleuchtungszustand des vorangegangenen Durchlaufes, sofern er hell war. Ein dunkeler Durchlauf wird nur nach einer Aktion gespeichert.

Geht der Zustand des Sensors auf dunkel (Zeile 16), wird geschaut, ob im vorangegangenen Durchlauf noch der Zustand hell erkannt wurde (Zeile 12). Außerdem wird überprüft, ob die Länge der Hellphase kürzer als ein Schwellwert (hier 50) und die Länge der Dunkelphase größer als ein Schwellwert (hier 5) war. Die letzte Bedingung sorgt dafür, dass in der Dämmerung das Licht nicht dauerhaft angetriggert wird. Falls alle Bedingungen zutreffen, wird eine Aktion eingeleitet. Außerdem wird der Hellzähler zurückgesetzt.

Aus diesem Programm ergibt sich, dass die Aktion erst eintritt, wenn die Beleuchtung des Sensors wieder aufhört.

[4] Eine Bytevariable kann Werte zwischen 0 und 255 annehmen.

2.4 Spezialitäten

In diesem Abschnitt werden einige Besonderheiten vorgestellt, mit denen das Programm verfeinert werden kann.

2.4.1 Watchdog-Abschaltung

Anstatt den Microcontroller mit dem Befehl `Waitms` eine Wartezeit einlegen zu lassen, ist es möglich, ihn während dieser Zeit in einen Betriebszustand zu versetzen, in dem nur sehr wenig Strom verbraucht wird. Realisiert wird dies über den Watchdog-Timer. Im Ruhezustand wird der Stromverbrauch so auf wenige μA senken.

Funktionsweise des Watchdog-Timers

Der Watchdog-Timer ist ein im Microcontroller integrierter Baustein und wird über Register (Abb. 2.8) konfiguriert. Er ist ein Zähler, der die Zyklen eines eingebauten 128 kHz-Oszillators zählt. Ist der eingestellte Zielwert erreicht, wird ein Interrupt und/oder ein Reset ausgelöst. In diesem Fall wird der Interrupt genutzt, um den Microcontroller aus dem Schlafmodus zu wecken. Die Interrupts müssen dafür aktiviert sein.

Bit	7	6	5	4	3	2	1	0	
	WDTIF	WDTIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

Abb. 2.8. Konfigurationsregister des Watchdog-Timers.

Benutzung des Watchdog-Timers

```
1 Wdtcr = &B11010011
2 Enable Interrupts
3
4 Reset Watchdog
5 Powerdown
```

In der ersten Zeile wird der Watchdog-Timer konfiguriert. Der Timeout-Wert liegt bei 0,125 s. Beim Erreichen dieses Wertes wird ein Interrupt ausgelöst. In Zeile 2 werden die Interrupts aktiviert. Diese Zeilen müssen im Programm nur einmal eingefügt werden unabhängig davon, wie häufig der Watchdog-Timer

WDP3	WDP2	WDP1	WDP0	Anzahl der Oszillator-Zyklen	Typische Timeout-Zeit bei $V_{CC} = 5,0$V
0	0	0	0	2K (2048) Zyklen	16 ms
0	0	0	1	4K (4096) Zyklen	32 ms
0	0	1	0	8K (8192) Zyklen	64 ms
0	0	1	1	16K (16384) Zyklen	0,125 s
0	1	0	0	32K (32768) Zyklen	0,25 s
0	1	0	1	64K (65536) Zyklen	0,5 s
0	1	1	0	128K (131072) Zyklen	1 s
0	1	1	1	256K (262144) Zyklen	2 s
1	0	0	0	512K (524288) Zyklen	4 s
1	0	0	1	1024K (1048576) Zyklen	8 s

Tabelle 2.1. Timeout-Zeit des Watchdog-Timers

WDTON	WDE	WDTIE	Modus	Aktion nach Timeout
0	0	0	Gestoppt	keine
0	0	1	Interrupt-Modus	Interrpt
0	1	0	System-Reset-Modus	Reset
0	1	1	Interrupt- und System-Reset-Modus	Interrupt, danach Reset
1	x	x	System-Reset-Modus	Reset

Tabelle 2.2. Modus des Watchdog-Timers

genutzt wird. Nur wenn im Programm die Wartezeit verändert werden soll, muss die Zeile 1 entsprechend angepasst erneut eingefügt werden.

Im Programm selber wird an der Stelle, an der die Wartezeit eingelegt wird, die Zeilen 4 und 5 eingefügt. Zuerst wird der Timer zurückgesetzt, sodass er beginnt zu zählen. Danach geht der Microcontroller in den Schlafmodus. Ist der Timer abgelaufen, wird der Controller durch den Interrupt wieder geweckt.

2.4.2 Umschalter zwischen Betriebsmodi

Gerade für Serienfertigungen oder Wartungszwecke ist es schön, wenn man zwschen verschiedenen Betriebsmodi wechseln kann. Aufgrund der Wetterbeständigkeit der Verpackung scheiden normale Schalter aber aus. Besser geeignet sind Reedkontakte. Sie bestehen aus zwei Metallzungen, die mit einem kleinen Abstand parallel in einem Glasröhrchen befestigt sind. Wird der Reedkontakt in ein Magnetfeld gebracht, schließen sich diese Kontakte. Somit ist es möglich, durch die Gießharzumhüllung der Schaltung hindurch mittels eines kleinen Magneten den Schalter zu betätigen.

Der Schaltungsaufbau ist identisch mit dem des Spannungsteilers mit einem Fotowiderstand (Abb. 2.3). Allerdings wird anstelle des Fotowiderstandes der Reedkontakt eingefügt.

```
1  Config Portb = &B00XX10XX
2  Portb = &BXXXX00XX
3
4  Wdtcr = &B11101110
5  Enable Interrupts
6
7  Const Maxmodus = 4
8  Dim Modus As Integer
9  Dim Old As Bit
10
11 Old = 0
12
13 Do
14   If Portb.2 = 1 And Old = 0 Then
15     Portb.3 = 1
16     Modus = Modus + 1
17     Modus = Modus Mod Maxmodus
18     Reset Watchdog
19     Powerdown
20     Portb.3 = 0
21   End If
22   Old = Portb.2
23
24   If Modus = 0 Then
25     ' Aktion in Modus 0
26   Elseif Modus = 1 Then
27     ' Aktion in Modus 1
28   Elseif Modus = 2 Then
29     ' Aktion in Modus 2
30   Elseif Modus = 3 Then
31     ' Aktion in Modus 3
32   End If
33 Loop
```

Für die Abfrage des Schaltzustandes muss lediglich der Eingangswert ausgelesen werden. In dem Beispielcode ist der Spannungsteiler mit dem Reedkontakt an Port 2 (Pin 7) gehängt. Dieser Port wird in den ersten beiden Zeilen als Eingang definiert. Die Zeilen 4 und 5 konfigurieren den Watchdog-Timer (Kapitel 2.4.1), der zur Entprellung des Signals benötigt wird. Die Anzahl der Betriebsmodi wird in der Konstanten `maxModus` festgelegt. Die Variable `modus` dient der Speicherung des aktuellen Modus. In Zeile 9 wird eine Variable definiert, die den Schaltzustand aus dem letzten Scheifendurchlauf speichert. Sie wird vor dem Beginn der Schleife auf 0 initialisiert.

Die Zeilen 14 bis 21 werden ausgeführt, wenn der Reedkontakt geschlossen ist nachdem er im vorherigen Durchlauf noch offen war. Zur optischen Ausgabe, dass ein Moduswechsel stattfindet, wird eine an Port 3 angeschlossene Leuchtdiode in Zeile 15 eingeschaltet. Sie wird am Ende des Wechsels in Zeile 20 wieder ausgeschaltet. In Zeile 16 wird der Modus gewechselt. Ist der höchste Modus erreicht, wird der Zähler automatisch auf 0 und damit den Modus 0 zurückgesetzt. Danach wird der Controller für 1 s in den Schlafmodus versetzt. Ohne dies würden mehrere Moduswechsel in Folge durchgeführt werden. Der Grund liegt darin, dass der Reedkontakt prellt. Er schaltet nicht nur einmal ein, sondern während des Schaltens springt er mehrfach zwischen offen und geschlossen hin und her. Dies wird durch die Wartezeit ausgeblendet. Nach dem Moduswechsel wird der aktuelle Zustand des Reedkontaktes in der Variablen `old` gespeichert.

In den Zeilen 24 bis 32 wird der Modus ausgewertet. Je nach aktuellem Stand werden unterschiedliche Blöcke der if-Schleife ausgeführt.

Beim Eingießen der Schaltung in Kunstharz ist auf die Zerbrechlichkeit der Reedkontakte zu achten. Sie dürfen niemals nur zum Teil im Gießharz sein, da sie die Spannungen beim Aushärten nicht aushalten, sondern müssen immer komplett bedeckt werden.

2.4.3 Parametrierbare Tagschwelle

Viele Programmvarianten enthalten einen Tagmodus, bei dem der Microcontroller in einen stromsparenden Schlafmodus verfällt, wenn dauerhaft ein Helligkeitswert über einer bestimmten Schwelle liegt. Dieser Schwellwert ist normalerweise fest einprogrammiert. Mit Hilfe eines Reedkontaktes (Kapitel 2.4.2) lässt er sich als frei parametrierbarer Parameter realisieren.

```
1  Dim Tagschwelle As Integer
2  Tagschwelle = 800
3
4  Do
5
6    ' Hier steht der normale Code des Reaktivlichts
7
8     If Portb.2 = 1 Then
9      Portb.3 = 1
10     Tagschwelle = Getadc(2)
11     Wdtcr = &B11010101
12     Reset Watchdog
13     Powerdown
14     Portb.3 = 0
15   End If
```

```
16
17 Loop
```

Zeile 1 und 2 definieren eine Variable, die anstelle der Konstanten als Tagschwelle genutzt wird. Diese Zeilen müssen in den Kopfbereich des Programms geschrieben werden. Innerhalb der Schleife steht dann in den Zeilen 8 bis 15 der eigentliche Programmcode. An Port 2 (Pin 7) hängt der Spannungsteiler mit dem Reedkontakt. Er muss entsprechend als Eingang parametriert werden. Wenn der Kontakt geschlossen ist, wird dieses Codestück ausgeführt. Zuerst schaltet es die LED an Port 3 an. Dies geschieht als Rückmeldung an den Benutzer. Danach wird der aktuelle Helligkeitswert als neue Tagschwelle gespeichert. Anschließend wird eine halbe Sekunde gewartet, bevor die LED wieder ausgeschaltet wird. Der Parametriervorgang ist damit abgeschlossen.

Falls die Schaltung sich im Tagmodus befindet, ist darauf zu achten, dass der Reedkontakt lange genug geschlossen bleibt, da das Parametrieren nur funktioniert, wenn die Schaltung aus dem Schlafmodus erwacht. Dies geschieht in der Regel nur alle acht Sekunden.

2.4.4 TeachIn-Modus

Manchmal ist es gewünscht, nach Fertigstellung der Schaltung die Blinksequenz ändern zu können. Dies erfüllt der TeachIn-Modus. Nach dem Anlegen der Versorgungsspannung oder einem Reset des Microcontrollers kann eine neue Blinksequenz eingelernt werden. Dies geschieht folgendermaßen:

1. Die LED blinkt zehn Mal kurz und einmal lang. Dies kennzeichnet den Beginn der TeachIn-Phase.
2. Nun muss mit Hilfe einer Taschenlampe die Blinksequenz eingegeben werden. Sie wird mit einem Abtastintervall von ca. 80 ms im EEPROM gespeichert. Aus der Größe des EEPROMs resultiert eine Gesamtdauer der Blinksequenz von 5 s. Wird die LED während der gesamten TeachIn-Phase nicht beleuchtet, wird zum Blinken die Standradsequenz (zehn Mal blinken) genutzt.
3. Die LED blinkt drei Mal kurz. Dies kennzeichnet das Ende der TeachIn-Phase.

```
1 Dim codeValid As Bit
2 Dim code As Bit
3 Dim i As Byte
4 Dim hell As Bit
5
6 codeValid = 0
7 Gosub BlinkStartTeachIn
```

```
For i = 1 To 63
  Gosub Sensorabfrage
  If hell = 1 Then
    code = 255
    codeValid = 1
  Else
    code = 0
  End If
  Writeeeprom code, i
  Waitms 65
Next i
Gosub BlinkEndTeachIn

' Hier das Grundgerüst für das Programm
' einfügen. Dabei die Methode Blink
' verwenden, die unten definiert wird.

Blink:
  Portb.3 = 0
  If codeValid = 0 Then
    ' Standard-Blinksequenz
  Else
    For i = 1 To 63
      Readeeprom code, i
      If code = 255 Then
        Portb.3 = 1
      Else
        Portb.3 = 0
      End If
      Waitms 90
    Next i
  End If
  return

BlinkStartTeachIn:
  ' zehn Mal kurz und einmal lang
  ' blinken
  return

BlinkEndTeachIn:
  ' drei Mal kurz blinken
  return

Sensorabfrage:
  ' Hier wird der Beleuchtungszustand des
```

```
53    ' Sensors abgefragt. Ist er beleuchtet,
54    ' wird die Variable hell auf true gesetzt,
55    ' andernfalls auf false.
56    return
```

Bevor das Programm in die Endschlosschleife des Grundgerüsts springt, erfolgt in den Zeilen 6 - 19 die TeachIn-Sequenz. Zu Beginn wird der Merker `codeValid` auf 0 gesetzt. Er zeigt an, ob ein gültiger Code im EEPROM gespeichert ist. Danach wird der Beginn der TeachIn-Phase durch das Startsignal signalisiert. In den Zeilen 8 - 18 wird die Blinksequenz abgefragt. Im EEPROM stehen 63 Byte zur Verfügung, die nacheinander gefüllt werden. Ist die LED beleuchtet, so wird der Wert auf 255 gesetzt, sonst auf 0. Danach wird das Byte im EEPROM geschrieben und eine kurze Wartezeit eingelegt. Durch Veränderung der Wartezeit kann die Abtastfrequenz und die maximale Länge der Blinksequenz beeinflusst werden.

In der Prozedur `Blink` wird in Zeile 27 abgefragt, ob das EEPROM eine gültige Blinksequenz enthält. Falls nicht, erfolgt eine Standardblinksequenz. Ansonsten wird byteweise das EEPROM ausgelesen und ausgegeben (Zeile 30 - 37). Das Programm ist so angelegt, dass die LED an Pin 2 angeschlossen ist. Falls dem nicht so ist, muss es noch entsprechend geändert werden.

2.5 Verpackung

2.6 Beispiele

Hier sind einige Beispiele von Reaktivlicht-Programmen aufgeführt, die aus den Bausteinen, die in diesem Kapitel beschrieben wurden, zusammengebaut werden können.

2.6.1 Nachtaktiver Blinker mit A/D-Wandler und Watchdog-Abschaltung

Dieses Reaktivlicht besitzt als Sensor einen Fotowiderstand, dessen Wert über den A/D-Wandler eingelesen wird (Kapitel 2.1.1). Der Spannungsteiler ist an Port 4 (Pin 3) angeschlossen. Als Ausgabe dient eine einfache Leuchtdiode (Kapitel 2.2.1), die an Port 3 (Pin 2) angeschlossen ist.

```
1  $regfile = "ATtiny13.DAT"
2  $crystal = 16000
3  $hwstack = 2
4
```

```
5  Config Adc = Single , Prescaler = Auto
6  Config Portb = &B00001000
7  Portb = 0
8  Stop Ac
9  Wdtcr = &B11010011
10 Enable Interrupts
11
12 Const Schwelle = 50
13 Const Tagschwelle = 800
14 Const Zwangsimpuls = 8
15 Dim A As Byte
16 Dim Tagzaehler As Byte
17 Dim Schlafzaehler As Byte
18 Dim Ldr As Integer
19 Dim Alt As Integer
20 Dim Merker As Integer
21
22 Do
23   Reset Watchdog
24   Powerdown
25   Start Adc
26   Ldr = Getadc(2)
27   Stop Adc
28   Merker = Ldr - Alt
29   Alt = Ldr
30   If Merker > Schwelle Then
31     Gosub Blinken
32   End If
33   If Ldr > Tagschwelle Then
34     If Tagzaehler < 255 Then
35       Tagzaehler = Tagzaehler + 1
36     End If
37   Else
38     Tagzaehler = 0
39   End If
40   If Tagzaehler > 200 Then
41     Gosub Pause
42   End If
43 Loop
44
45 Blinken:
46   For A = 0 To 10
47     Portb.3 = 1
48     Reset Watchdog
49     Powerdown
```

```
50      Portb.3 = 0
51      Reset Watchdog
52      Powerdown
53    Next A
54    Alt = 1023
55  Return
56
57  Pause:
58    Wdtcr = &B11110001
59    Reset Watchdog
60    Powerdown
61    Wdtcr = &B11010011
62    Schlafzaehler = Schlafzaehler + 1
63    If Schlafzaehler = Zwangsimpuls Then
64      Portb.3 = 1
65      Reset Watchdog
66      Powerdown
67      Portb.3 = 0
68      Schlafzaehler = 0
69    End If
70  Return
71
72  End
```

In den Zeilen 1 bis 3 werden allgemeine Einstellungen an der Hardware vorgenommen. Zuerst wird dem Compiler mitgeteilt, um welchen Prozessortyp es sich handelt. Danach wird die Frequenz des internen Oszillators gesetzt. Zum Schluss wird der Stack auf 2 gesetzt, damit für die Variablen des Programms genügend Platz zur Verfügung steht. Dies hat allerdings zur Folge, dass die Verschachtelungstiefe von Funktionsaufrufen maximal zwei betragen darf, was für dieses Programm jedoch ausreichend ist.

Danach wird der Microcontroller über die Register konfiguriert. Zuerst wird der Analog-Digital-Wandler eingestellt, mit dem der Helligkeitsgrad eingestellt wird. Danach wird der Port 3 als Ausgang konfiguriert und auf low gesetzt. Die restlichen Port sind Eingänge. In Zeile 8 wird der Analog-Komparator abgestellt, um Strom zu sparen. Er wird in diesem Programm nicht benötigt. Zeile 9 und 10 stellen den Watchdog-Timer, über den die Wartezeiten realisiert werden, auf 0,125 s und in den Interrupt-Modus.

Nun werden die Konstanten und Variablen definiert, die im Programm benötigt werden. `Schwelle` gibt die Mindestgröße der Helligkeitsänderung zwischen zwei Durchläufen an, damit das Reaktivlicht ausgelöst wird. Hiermit kann die Empfindlichkeit der Schaltung eingestellt werden. `Tagschwelle` gibt den Wert vor, oberhalb dessen die Schaltung in den Tagmodus wechselt. Um den Tagmodus anzuzeigen, kann mit der Konstante `Zwangsimpuls` fest-

gelegt werden, nach wieviel Schlafzyklen (hier etwa 8 s) ein Impuls auf der LED ausgegeben werden soll. `A` ist eine Zählvariable, die in der Unterprozedur `Blinken` genutzt wird. `Tagzaehler` zählt während des Programmablaufs die aufeinanerfolgenden Zyklen, in denen der Helligkeitswert über der Tagschwelle liegt. `Schlafzaehler` zählt die Schlafzyklen für die Ausgabe des Kontrollimpulses auf der LED. Der aktuelle Helligkeitswert wird in der Variablen `Ldr`, derjenige des vorangegangenen Durchlauf in `Alt` gespeichert. `Merker` ist eine Variable, in der die Differenz zwischen diesen beiden Werten gespeichert wird.

Die Zeilen 22 bis 43 enthalten das Hauptprogramm. Zu Beginn wird der Controller für 0,125 s in den Schlafmodus versetzt. Dadurch wird die Wahrscheinlichkeit einer Fehlauslösung verringert, da beim Wechsel zwischen unbeleuchtetem und beleuchtetem Sensor ein größerer Unterschied ist als bei kontinuierlicher Abfrage. Danach wird in Zeile 25 bis 27 der Analog-Digital-Wandler gestartet, der Helligkeitswert eingelesen und der Wandler wieder ausgeschaltet. Es wird die Differenz zum vorangegangenen Zyklus berechnet und anschließend der jetzige Helligkeitswert für den nächsten Durchlauf in der Variablen `Alt` gespeichert. Ist der Helligkeitsunterschied größer als die eingestellte Schwelle, wird die Unterroutine `Blinken` aufgerufen. Ab Zeile 33 werden die Bedingungen für den Tagmodus überprüft. Ist der Helligkeitswert über der Tagschwelle, wird der Tagzaehler hochgezählt, ansonsten zurück auf 0 gesetzt. Sollte der Tagzähler den Wert 200 erreichen, was bedeutet, dass über 200 Zyklen a 0,125 s Tag detektiert wurde, wird die Unterprozedur `Pause` aufgerufen.

Die Unterprozedur `Blinken` befindet sich in den Zeilen 45 bis 54. Hier wird zehnmal in Folge die Port 3, an dem die LED angeschlossen ist, angeschaltet, mittels des Watchdog-Timers 0,125 s gewartet, der Ausgang wieder ausgeschaltet und erneut gewartet. Im Anschluss an diesen Vorgang wird der Wert der gemessenen Helligkeit auf den Maximalwert gesetzt, sodass eine Doppelauslösung verhindert wird.

Die Zeilen 57 bis 70 enthalten die Prozedur `Pause`. Damit wird der Microcontroller während des Tagmodus für 8 s ausgeschaltet. Zu Beginn wird die Zeit des Watchdog-Timers auf den Maximalwert gesetzt. Anschließend wird der Controller in den Schlafmodus versetzt. Wenn er daraus wieder erwacht ist, wird die Zeit des Watchdog-Timers wieder zurückgesetzt und der Schlafzähler inkrementiert. Erreicht der Schlafzähler den Wert, der in der Konstanten `Zwangsimpuls` angegeben ist, wird ein einzelner Lichtblitz auf der LED ausgegeben und der Schlafzähler auf 0 zurückgesetzt.

Literaturverzeichnis

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