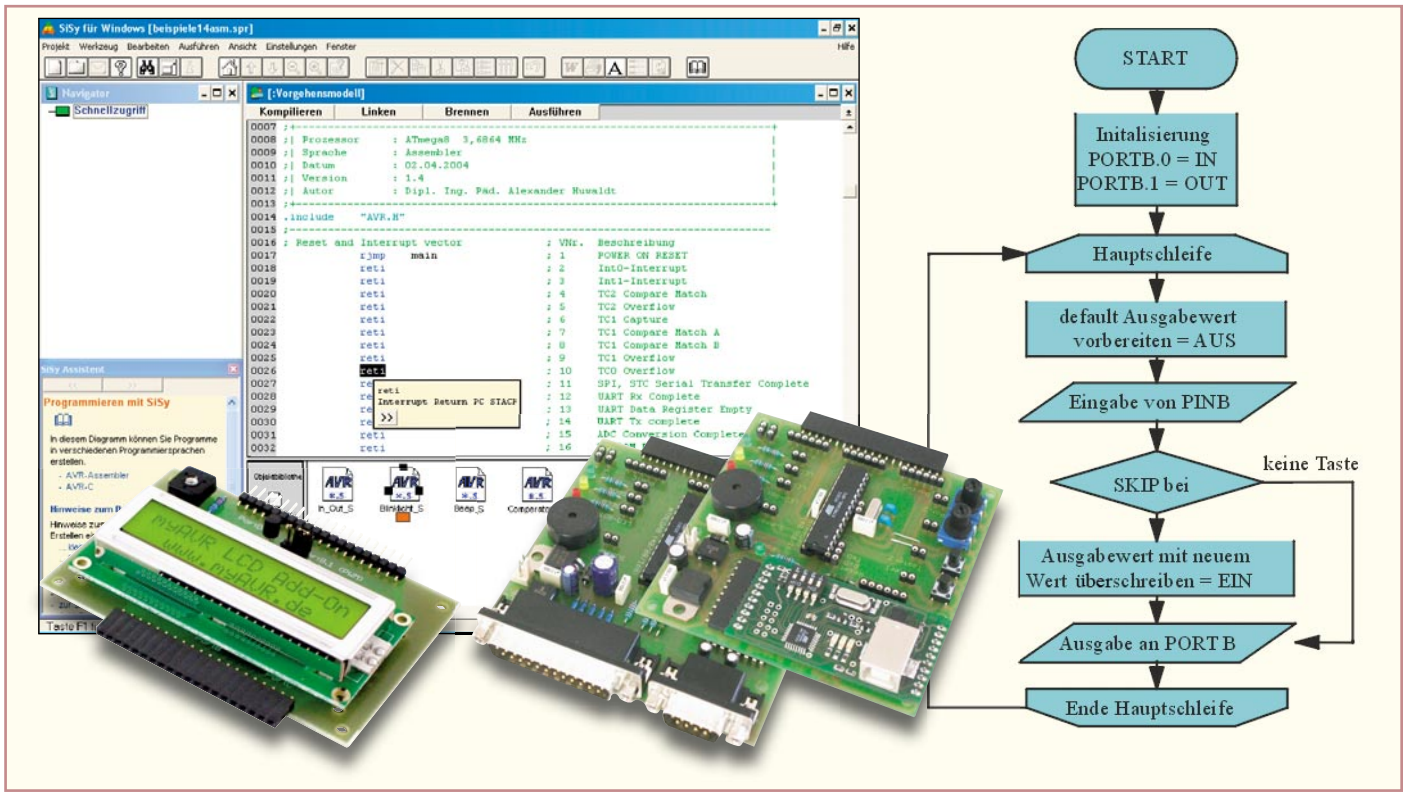

Mikrocontroller-Einstieg mit myAVR Teil 4

Keine Angst vor dem Einstieg in die Welt der Mikrocontroller-Programmierung! Die myAVR-Sets enthalten alles Nötige für den schnellen und fundierten Beginn der Programmierer-Karriere – Experimentier-Board mit ATMEL-Controller, Lehrbuch, Softwarepaket, Kabel, sämtliches Zubehör. Im vierten Teil unserer Serie zum Einstieg in die AVR-Programmierung wenden wir uns den integrierten Timern des AVR zu und nutzen diese zur Tonerzeugung.

Teilen und Zählen – der Timer

Dass Zähler ein zentraler Bestandteil nahezu jeder digitalen Schaltung sind, ist nichts Neues. Sie sorgen u. a. dafür, dass zu einer bestimmten Zeit ein bestimmtes Ereignis ausgelöst wird. Sie benötigen einen Takt, dessen Frequenz um einen bestimmten Faktor geteilt wird – so entsteht eine zeitlich bezogene Steuerung, ein so genannter Timer.

Auch die Timer des AVR-Controllers funktionieren auf dieser Basis. Und sie haben bei den meisten Aufgaben, die der Controller bekommt, eine Menge zu tun, um zeitliche Abläufe zu regeln.

Unser ATmega 8 besitzt drei Timer mit unterschiedlicher Verarbeitungsbreite und unterschiedlichen Aufgaben. Alle Timer sind interruptfähig, sind also als interne Interruptquelle über die Interrupt-Vektor-Tabelle einbindbar (vgl. dazu auch Tabelle 1 im Teil 3). Je nach Timerkonfiguration und Timerart können folgende Ereignisse einen Interrupt auslösen:

- Nulldurchlauf (Overflow, der Timer zählt herauf oder herab, bis er an den Wert 0 gelangt und löst dann einen Interrupt aus)
- Vergleichswert (Compare, der Timer zählt bis zu einem in einem Register abgelegten Zahlenwert und löst bei dessen Erreichen einen Interrupt aus)
- externer Zählimpuls (Capture, Auslösung eines Interrupts bei Erreichen einer bestimmten Anzahl von externen Zählimpulsen)

Als Taktquelle für den Timer dient in den allermeisten Fällen der interne Prozessortakt, seltener ein externer Takt.

Timer-Aufbau und Timer-Register

Wir wollen die grundsätzliche Arbeitsweise am 8-Bit-Timer/Counter 0 betrachten

(Abbildung 36). Er setzt sich aus mehreren Elementen zusammen. Da ist zunächst die eigentliche Timerlogik, die die Taktauswahl und -generierung, darunter auch aus externen Taktquellen, realisiert und einen 10-Bit-Vorteiler beherbergt, der je nach Aufgabe den Eingangstakt um die Faktoren 8, 64, 256 oder 1024 teilt. Der „Rest“ sind wiederum Register, die nach entsprechendem Laden die Aufgaben des Timers festlegen:

- TCNT0: Timer/Counter 0, das Zählerregister, in dem das eigentliche Zählen stattfindet. Bei Überlauf (Nulldurchgang) wird ein Signal an die Kontroll-Logik abgegeben.
- TIMSK: Timer Interrupt Mask Register; dient der Konfiguration der Interrupt-Ausgabe (Interrupt-Maskierung, siehe Teil 3: Interruptbefehle und -register).
- TCCR0: Timer Counter Control Register, dient als Steuerregister der Konfiguration, z. B. des gewünschten Teilerfaktors.
- TIFR: Timer Interrupt Flag Register, dient zum Auswerten des Interrupt-Status. Sind hier z. B. das Flag TOV0 (Timer/Counter 0 Overflow Flag) und im Register TIMSK das Flag TOIE0 (Timer/Counter 0 Overflow Interrupt Enable, Interrupt aktiviert) gesetzt, erfolgt die Ausführung des Timer-Interrupts.

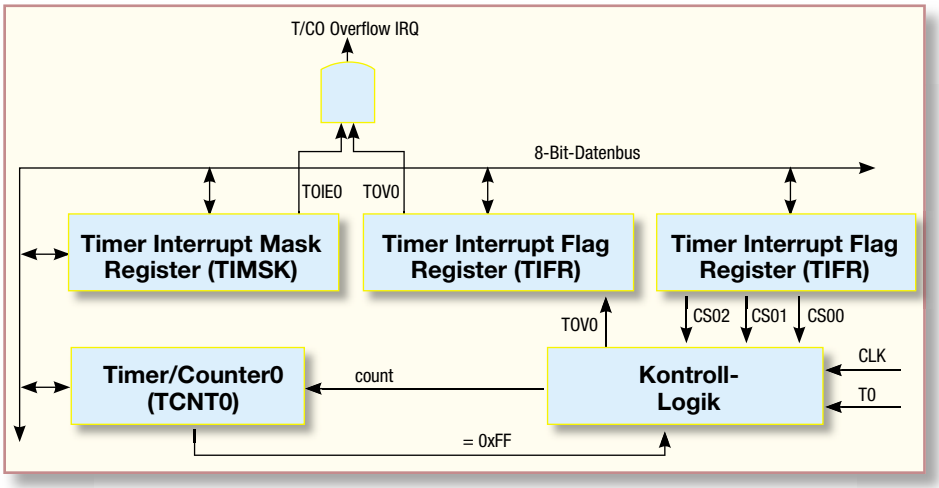

Bild 36: Der Grundaufbau des Timers/Zählers TCNT0

7	6	5	4	3	2	1	0	Bit
OCIE2	TOIE2	TICIE2	OCIE1A	OCIE1B	TOIE2	–	TOIE0	TIMSK

7	6	5	4	3	2	1	0	Bit
–	–	–	–	–	CS02	CS01	CS00	TCCR0

CS02	CS01	CS00	Beschreibung
0	0	0	Keine Taktquelle (Timer/Counter gestoppt)
0	0	1	CLK/0 (Vorteiler aus)
0	1	0	CLK/8 (Vorteilerfaktor 8)
0	1	1	CLK/64 (Vorteilerfaktor 64)
1	0	0	CLK/256 (Vorteilerfaktor 256)
1	0	1	CLK/1024 (Vorteilerfaktor 1024)
1	1	0	Externer Takt an T0, Auslösung: fallende Flanke
1	1	1	Externer Takt an T0, Auslösung: steigende Flanke

7	6	5	4	3	2	1	0	Bit
OCF2	TOV2	ICF1	OCF1B	OCF1B	TOV1	–	TOV0	TIFR

Bild 37: Die Register TIMSK, TCCR0 und TIFR

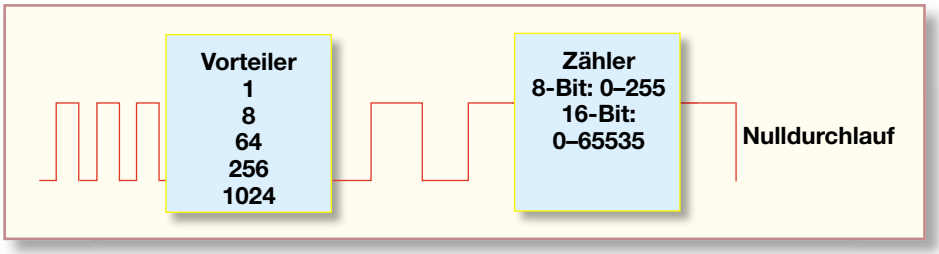

Bild 38: Die Zusammenarbeit Vorteiler/Zähler

Die Register werden für alle drei Timer gemeinsam genutzt, während die Kontroll-Logik und das Zählerregister sich entsprechend der Aufgabe des jeweiligen Timers unterscheiden. So verfügt z. B. Timer 1 über eine Verarbeitungsbreite von 16 Bit (Zählumfang bis 65.535 für längere und präzisere Zeitmessung als mit Timer 0) und einen so genannten Waveform Generation Mode, über den eine Pulsweiten-Modulation möglich ist. So kann man z. B. die Helligkeit einer LED modulieren oder die Drehzahl eines Motors steuern.

In Abbildung 37 ist der Aufbau aller drei beschriebenen Register zur Übersicht dargestellt, dazu für das TCCR0-Register die Tabelle zur Programmierung der Taktauswahl (Clock-Select). Auf diese werden wir noch zurückkommen.

Abbildung 38 zeigt die grundsätzliche Funktion des Teilers/Zählers. Der Prozessortakt (oder externe Takt) wird zunächst durch den programmierbaren Vorteiler auf den gewünschten Wert, der für den 8-Bit-Zähler im Bereich von 1 bis 255 liegen muss, herabgeteilt. Die Ausgangsimpulse des Vorteilers werden nun vom Zähler von einem programmierbaren Punkt an (Differenz zu 255) herauf- oder herabgezählt, bis ein Nulldurchgang auftritt. Dieses Ereignis wird an die Kontroll-Logik gemeldet, und der Zählerlauf beginnt von vorn.

Doch zunächst müssen wir die Voraussetzungen betrachten, um den Timer überhaupt in ein Programm einzubinden.

Vorbereitungen

Um die Timer-Aktivitäten als Interrupt ausnutzen zu können, ist natürlich, wie wir es bereits im Teil 3 kennengelernt haben, der passende Interrupt-Vektor auf eine zu erstellende Interrupt-Service-Routine zu setzen.

Nach bekanntem Schema wird also eine Interrupt-Service-Routine (ISR) mit den Befehlen cli, sei und reti erstellt, und in der Vektortabelle ist der zugehörige Interrupt-Vektor (Timer 0, Nulldurchlauf) auf die ISR zu setzen.

Weiterhin ist dem Steuerwerk des Controllers über das Timer Interrupt Mask Register TIMSK mitzuteilen, welches Ereignis als Interrupt akzeptiert werden soll (Maskierung). Wenn wir also den Nulldurchgang unseres Zählers 0 als Interrupt-Ereignis definieren wollen, ist im Register TIMSK das zugehörige Bit 1 zu setzen. Der myAVR-Lehrgang gibt zur gesamten Interrupt-Programmierung des Timers noch weitergehende Erläuterungen.

Das klingt bisher alles noch etwas theoretisch, wir werden aber gleich anhand unseres Beispielprogramms in der Praxis sehen, wie es funktioniert.

Der Prozessor ruft „A“

Wir wollen, wie angekündigt, über die Timerprogrammierung unserem AVR einen Ton entlocken. Dabei ist zunächst zu betrachten, wie eine Tonfrequenz prinzipiell entsteht. Sie ist vereinfacht als Folge von High-Low-Wechseln eines digitalen Signals zu betrachten. Je höher die Tonfrequenz sein soll, desto dichter müssen die High-Low-Wechsel aufeinander folgen. Ein Zyklus eines Tons besteht jeweils aus zweien dieser Wechsel („Halbwellen“), dies muss man beim späteren Berechnen der so genannten Wartezeit beachten.

Wir wollen dazu ein Beispiel des myAVR-Lehrgangs betrachten, das sich zum Ziel stellt, den Kammerton „A“ mit einer Frequenz von 440 Hz aus der Taktfrequenz des Controllers via Timersteuerung und Interruptauswertung des Timers zu erzeugen und mittels des kleines Lautsprechers auf dem myAVR-Board auszugeben.

Für die Signalausgabe ist es also nötig, einen High-Low-Signalwechsel per Timer-Overflow-Interrupt zu erzeugen. Dazu gibt es zunächst etwas Rechenarbeit. Die Zykluszeit des 440-Hz-Signals beträgt zunächst 1/440 s, also 2,28 ms. Für zwei „Halbwellen“ sind dies 1/880 s, also 1,14 ms. Dies ist die so genannte Wartezeit von einer zur nächsten Halbwelle, die vom Zähler zu generieren ist. Bei der Taktfrequenz des AVR von 3,6864 MHz entspricht diese Wartezeit etwa 4189 Taktzyklen des Systemtakts (ca. 0,24 µs).

Um nun aus diesem Systemtakt 440 Hz zu erzeugen, ist mittels der Möglichkeiten, die der Vorteiler des Zählers und der Zähler selbst bieten, der Systemtakt herabzuteilen. Dabei muss beachtet werden, dass der Vorteiler-Faktor so zu wählen ist, dass die gewünschte Wartezeit im 8-Bit-Zählbereich des Zählers von 1 bis 255 liegt. Die Wartezyklen werden dann bei jedem Interrupt des Timers als Differenz zu 255 in den Zähler geladen. Eine Re-Initialisierung bei diesem Differenzwert bewirkt, dass die Zyklen bis zum nächsten Nulldurchlauf wieder herabgezählt werden.

Bei der relativ geringen Auflösung unseres 8-Bit-Zählers und der ganzzahligen Teilung wird nicht jede gewünschte Frequenz exakt erreicht werden, so auch unsere 440 Hz. Denn wenn wir die Taktzyklen exakt durch die Vorteiler-Faktoren teilen, ergeben sich immer Nachkommastellen, die mit dem ganzzahligen Zähler nicht zu realisieren sind. Wir vernachlässigen jedoch hier die geringe Abweichung, die unter einem Prozent liegt.

Wir teilen also die o. g. 4189 Taktzyklen durch 64, 256 und 1024 und erhalten als nächstliegende ganze Zählerwerte entsprechend 65, 16 und 4. Rechnen wir zurück:

64 x 65 = 4160
256 x 16 = 4096
1024 x 4 = 4096

4160 liegt dem Ziel 4189 am nächsten, also wählen wir die Kombination: Vorteiler 64, und Re-Initialisierungswert 190 (255 – 65).

Auf diese Weise kann man nahezu jede beliebige Frequenz bis herauf zur Taktfrequenz (dann ist der Vorteiler über das Register TCCR0 abzuschalten) erzeugen. Nachdem wir nun alle benötigten Voraussetzungen besprochen haben, was zeigt, wie man eine zu lösende Aufgabe vorab durchdenken muss, können wir nun an das Schreiben des Programms gehen.

Das Programm

Im Quellcode (Abbildung 39) sehen wir zunächst den Zeiger in der Vektortabelle,

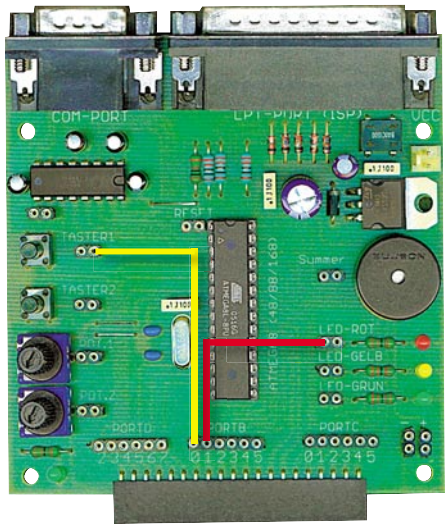

Bild 40: Die Verdrahtung des myAVR-Boards

```
;+----------------------------------------------------------------------+
;| Titel         : Beispiel Timer0 für SiSy AVR-Board                   |
;+----------------------------------------------------------------------+
;| Funktion      : generiert eine Tonfrequenz per Timer-Interrupt       |
;| Schaltung     : Port B.0 an Speaker                                  |
;+----------------------------------------------------------------------+
;| Prozessor     : ATmega8  3,6864 MHz                                  |
;| Sprache       : Assembler                                            |
;| Datum         : 05.09. 2006                                          |
;| Version       : 1.0                                                  |
;| Autor         : Dipl. Ing. Päd. Alexander Huwaldt & ELV              |
;+----------------------------------------------------------------------+
.include    "AVR.H"
;----------------------------------------------------------------------
; Reset and Interrupt vector              ; VNr.  Beschreibung
            rjmp    main                  ; 1     POWER ON RESET
            reti                          ; 2     Int0-Interrupt
            reti                          ; 3     Int1-Interrupt
            reti                          ; 4     TC2 Compare Match
            reti                          ; 5     TC2 Overflow
            reti                          ; 6     TC1 Capture
            reti                          ; 7     TC1 Compare Match A
            reti                          ; 8     TC1 Compare Match B
            reti                          ; 9     TC1 Overflow
            rjmp    onTC0                 ; 10    TC0 Overflow
            reti                          ; 11    SPI, STC Serial Transfer Complete
            reti                          ; 12    UART Rx Complete
            reti                          ; 13    UART Data Register Empty
            reti                          ; 14    UART Tx complete
            reti                          ; 15    ADC Conversion Complete
            reti                          ; 16    EEPROM Ready
            reti                          ; 17    Analog Comparator
            reti                          ; 18    TWI (I²C) Serial Interface
            reti                          ; 19    Store Program Memory Ready
;----------------------------------------------------------------------
; Start, Power ON, Reset
main:       ldi     r16,    hi8(RAMEND)   ;Stack Initialisierung
            out     SPH,    r16
            ldi     r16,    lo8(RAMEND)
            out     SPL,    r16
            sbi     DDRB,   0             ;Out Speaker
            sbi     DDRB,   1             ;Out LED
            ldi     r16,    0b00000011    ;Vorteiler setzen
            out     TCCR0,  r16           ;Timer0 setup  3686400/64
            ldi     r16,    0b00000001    ;Interrupt maskieren
            out     TIMSK,  r16           ;Interrupt Timer0 bei overflow
            sei
;----------------------------------------------------------------------
mainloop:   wdr
            rjmp    mainloop
;----------------------------------------------------------------------
onTC0:      cli                           ;Interrupts sperren
            com     r16                   ;Einerkomplement
            out     PORTB,  r16           ;Port B umschalten (Signalwechsel)
            ldi     r17,    190           ;berechneter Re-Init-Wert
            out     TCNT0,  r17           ;Timer Re-Init
            sei                           ;Interrupt wieder freigeben
            reti                          ;Rücksprung
;----------------------------------------------------------------------
```

Bild 39: Der Quellcode für die Tonerzeugung via Timer-Interrupt-Steuerung

der den Interrupt durch Overflow von Zähler 0 auslöst.

Das Hauptprogramm beginnt, wie bekannt, mit der Initialisierung, wobei hier Port B als Ausgang für den Anschluss des Signalgebers und der LED festgelegt wird.

Dem folgen die Festlegung des Teilerfaktors 64 laut TCCR0-Tabelle aus Abbildung 37 sowie die Konfiguration des Timer Interrupt Mask Registers (Interrupt erlauben).

In der nach der Interrupt-Auslösung aufgerufenen Interrupt-Service-Routine „onTC0“ finden wir einen neuen Befehl: com – das Einerkomplement. Der bedeutet hier, dass mit dem Interrupt ein Signalwechsel an Port B stattfindet und so ein Ton ausgegeben wird (Einerkomplement: jede Zahl wird durch ihr Gegenteil ersetzt, also 0 wird 1 und die 1 wird 0).

Beispiel:

Zahl 00010011 (19)

Einer-Komplement 11101100 (236)

Additionsergebnis 11111111 (255)

Als Re-Init-Wert finden wir zum Rückstellen des Zählers unseren berechneten Wert 190 vor.

Danach werden die allgemeinen Interrupts wieder freigegeben und es erfolgt der Rücksprung ins Hauptprogramm.

Als Ergebnis wird man nun nach Compilieren, Linken und Brennen den Kammerton A aus dem Signalgeber des myAVR-Boards vernehmen. Die Abweichung lag tatsächlich unter 1 %, wir haben 437 Hz gemessen. Abbildung 40 zeigt die Verdrahtung. Die ebenfalls mit verdrahtete LED zeigt die hohe Frequenz für das Auge mit Dauerleuchten an und dient als optische Anzeige für Variationen des Timings.

```
;+----------------------------------------------------------------------+
;| Titel         : Beispiel Timer1 für SiSy AVR-Board                   |
;+----------------------------------------------------------------------+
;| Funktion      : Blinkfrequenz per Timer-Interrupt          |
;| Schaltung     : Port B.0 an Speaker                                  |
;+----------------------------------------------------------------------+
;| Prozessor     : ATmega8  3,6864 MHz                                  |
;| Sprache       : Assembler                                            |
;| Datum         : 05.09.2006                                           |
;| Version       : 1.0                                                  |
;| Autor         : Dipl. Ing. Päd. Alexander Huwaldt & ELV                |
;+----------------------------------------------------------------------+
.include    "AVR.H"
;----------------------------------------------------------------------
; Reset and Interrupt vector            ; VNr.  Beschreibung
            rjmp    main                ; 1     POWER ON RESET
            reti                        ; 2     Int0-Interrupt
            reti                        ; 3     Int1-Interrupt
            reti                        ; 4     TC2 Compare Match
            reti                        ; 5     TC2 Overflow
            reti                        ; 6     TC1 Capture
            reti                        ; 7     TC1 Compare Match A
            reti                        ; 8     TC1 Compare Match B
            rjmp    onTC1               ; 9     TC1 Overflow
            reti                        ; 10    TC0 Overflow
            reti                        ; 11    SPI, STC Serial Transfer Complete
            reti                        ; 12    UART Rx Complete
            reti                        ; 13    UART Data Register Empty
            reti                        ; 14    UART Tx complete
            reti                        ; 15    ADC Conversion Complete
            reti                        ; 16    EEPROM Ready
            reti                        ; 17    Analog Comparator
            reti                        ; 18    TWI (I²C) Serial Interface
            reti                        ; 19    Store Program Memory Ready
;----------------------------------------------------------------------
; Start, Power ON, Reset
main:       ldi     r16,    hi8(RAMEND)
            out     SPH,    r16
            ldi     r16,    lo8(RAMEND)     ; Stack Initialisierung
            out     SPL,    r16
            sbi     DDRB,   0
            sbi     DDRB,   1
            ldi     r16,    0b00000101
            out     TCCR1B, r16             ;Timer1 setup 3686400/1024
            ldi     r16,    0b00000100
            out     TIMSK, r16              ;Interrupt Timer1 bei overflow
            sei
;----------------------------------------------------------------------
mainloop:   wdr
            rjmp    mainloop
;----------------------------------------------------------------------
onTC1:      cli                         ;Interrupts sperren
            com     r16                 ;Einerkomplement
            out     PORTB, r16          ;Port B umschalten (Signalwechsel)
            ldi     r17,    0xFE        ;Low-Byte von 65175 setzen
            ldi     r18,    0x97        ;High-Byte von 65175 setzen
            out     TCNT1H, r17         ;Low-Byte für Re-Init auslesen
            out     TCNT1L, r18         ;High-Byte für Re-init auslesen
            sei                         ;Interrupts wieder freigeben
            reti                        ;Rücksprung
;----------------------------------------------------------------------
```

Bild 41: Der Quellcode für die Erzeugung des 5-Hz-Blinkimpulses

Experimente mit 16 Bit

Neugierig geworden, kann man nun mit den Rechenwerten experimentieren und so sicherer in der Handhabung der Optionen werden, die der Timer bietet.

Dabei wird man früher oder später die Möglichkeiten des 16-Bit-Timers ausprobieren wollen. Der hat bekanntlich einen Zählbereich bis 65.535. Das wollen wir gleich testen und stellen dem Controller im Prinzip die gleiche Aufgabe wie vorher, allerdings soll nun die LED mit 5 Hz deutlich sichtbar blinken. Ergo ergibt sich die Wartezeit bei 3,6864 MHz Takt mit 368.640 Zyklen, da ja wieder beide „Halbwellen“ zu berücksichtigen sind, der Takt also durch 10 zu teilen ist. Teilen wir diese Zyklenzahl durch 1024, so ergibt sich der Zählerwert von 360 – für Timer 1 kein Problem, und zurückgerechnet ergibt sich genau der Wert der Wartezeit. In bewährter Manier ziehen wir diese 360 von 65.535 ab und erhalten einen Re-Init-Wert von 65.175.

Jetzt ergibt sich ein Problem. Der 16-Bit-Wert passt nicht zur 8-Bit-Busstruktur des Controllers, wir könnten also die sich ergebende 16-stellige Binärzahl nicht als Re-Init-Wert in der ISR eintragen.

Das Datenblatt des ATmega8 gibt auf Seite 79 Aufschluss über die Lösung – man teilt die Zahl in zwei Bytes auf, das so genannte Low-Byte und das High-Byte, schreibt die beiden Bytes in ein Register-Paar und liest sie dann nacheinander wieder aus.

Also trägt man den Hex-Zahlenwert von 65.175, FE97, entsprechend in zwei Register ein, gefolgt von den entsprechenden Out-Befehlen. Nicht vergessen, den Eintrag in der Vektortabelle entsprechend zu modifizieren, wir nutzen nun den Timer 1 und dessen Overflow-Meldung als Interruptquelle!

Das Ergebnis ist im Listing in Abbildung 41 zu sehen. Die Verdrahtung ist identisch mit der in Abbildung 40.

Das soll es zunächst zu den Timern gewesen sein. Im Schulungsmaterial zu myAVR finden Sie noch mehrere weitere Nutzungsbeispiele, darunter auch eine ausführliche Erklärung zur Nutzung der PWM-Funktion des Timers.

In der nächsten Folge geht es um die Kommunikation mit dem PC über die serielle Schnittstelle – wir lernen die UART kennen.