2.2 Atmega-Peripherie/Timer

2.2.1 Was sind Timer und wozu braucht man sie?

Oft müssen in Rechnersystemen bestimmte Prozesse regelmäßig im Hintergrund laufen, während im Vordergrund wichtige Benutzereingaben stattfinden.

Mit normalen externen Interrupts jedoch kann man die Hauptschleife eines Programms nur auf externe Anfrage hin unterbrechen.

Will man das periodisch immer wieder haben, müsste man an so einen externen Interrupt-Eingang einen Impulsgenerator anschließen, der regelmäßig den Sprung in den entsprechenden Interrupt-Handlder verursacht (Abbildung 1).

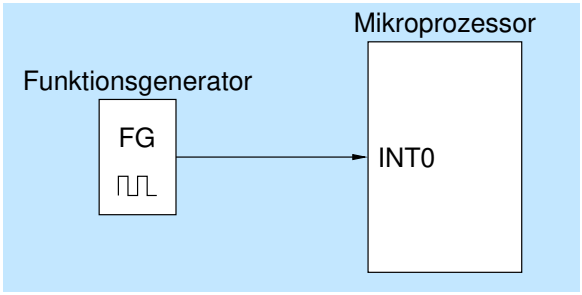

Abbildung 1: Idee eines Timers

In Mikroprozessorsystemen gibt es wirklich solche Bausteine, sie heißen dort Zeitgeber oder *Timer*. Sie können dort zum Beispiel den Refresh von dynamischen Speicherbausteinen anstoßen oder auch den Scheduler, der das Multitasking veranlasst, steuern (Abbildung 2).

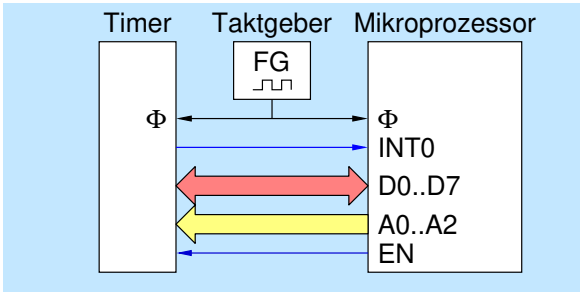

Abbildung 2: Timer in einem Mikroprozessorsystem

In Mikrocontrollern sind meistens einer oder mehrere solcher Timer integriert. Damit sie so flexibel sein können wie ihre einzeln arbeitenden Kollegen, hat man sie sehr stark konfigurierbar aufgebaut: Durch das Umsetzen bestimmter Bits in den entsprechenden Registern kann man die Arbeitsweise eines Timers völlig verändern.

2.2.2 Timer im ATmega

Am besten beschreibt man einen Timer als eine **Zähler**, der durch bestimmte **Ereignisse** hoch- oder heruntergezählt wird und bei bestimmten **Zählerständen** eine **Aktion** auslöst.

Im ATmega8 gibt es drei verschiedene Timer, die weitgehend unabhängig voneinander arbeiten können:

a) Timer 0 ist ein 8-Bit-Timer, durch Überlauf ausgelöst

b) Timer 1 ist ein 16-Bit-Timer, auf verschiedene Arten ausgelöst

CS2	CS1	CS0	Bedeutung
0	0	0	Timer steht
0	0	1	Controllertaktfrequenz zählt den Timer
0	1	0	Controllertaktfrequenz/8 zählt den Timer
0	1	1	Controllertaktfrequenz/64 zählt den Timer
1	0	0	Controllertaktfrequenz/256 zählt den Timer
1	0	1	Controllertaktfrequenz/1024 zählt den Timer
1	1	0	fallende Taktflanke am Eingang T0 zählt den Timer
1	1	1	steigende Taktflanke am Eingang T0 zählt den Timer

Tabelle 1: Bedeutung der Bits CS2 bis CS0 im Register `TCCR0`

c) Timer 2 ist noch ein 8-Bit-Timer, auf verschiedene Arten ausgelöst

Hier soll nur der Timer 0 betrachtet werden.

2.2.3 Timer 0

Der 8-Bit-Timer Timer 0 hat vier Register, die ihn steuern.

a) Das Register `TCNT0` (Timer / Counter 0) ist der eigentliche 8-Bit-Zähler, also die Ursache von Aktionen. Es kann vom Programm aus mit einem Wert geladen werden.

b) Das Register `TIFR` (Timer Interrupt Flag Register) ist die zugehörige Wirkung. In diesem Register befinden sich Flags, die angeben, dass ein bestimmter Zählerstand aufgetreten ist. Sie ermöglichen zusammen mit dem zugehörigen `TIMSK`-Bit eine Aktion. Für den Timer 0 interessiert nur das Bit Nr. 0 mit dem Namen `TOV0` (Timer Overflow 0). Dieses Bit wird auf 1 gesetzt, sobald der Zähler von 255 auf 0 überläuft[1].

c) Das Register `TIMSK` (Timer Interrupt Mask Register) dient der Konfiguration. Es maskiert, welche Aktionen ausgelöst werden. Hier interessiert nur das Bit Nr. 0 mit dem Namen `TOIE0` (Timer Overflow Interrupt Enable 0). Wenn man es auf 1 setzt, wird beim Eintreten des Überlaufs ein Interrupt `TIMER0_OVF` (Nr. 9 beim ATmega8, Nr. 10 beim ATmega16 und Nr. 12 beim ATmega32) ausgelöst (vorausgesetzt, das `I`-Bit im Statusregister `SREG` ist gesetzt worden).

d) Das Register `TCCR0` (Timer Counter Control Register) dient ebenfalls zur Konfiguration. Es legt fest, welche Ereignisse den Zähler zählen lassen. Für Timer 0 interessieren in diesem Register nur die Bits Nr. 0 (CS0), Nr. 1 (CS1) und Nr. 2 (CS2); die Bedeutung der möglichen Kombinationen zeigt Tabelle 1.

2.2.4 Maximale Laufzeit eines Zählers

Wie groß ist nun die minimale Frequenz, die man bei einer Taktfrequenz $f_{CPU} = 16\,\text{MHz}$ erreichen kann? Dazu berechnet man zunächst die Zählfrequenz f_{count}:

$$f_{count} = \frac{f_{CPU}}{N_{Teil,max}} = 16\,\text{MHz}/1024 = 15{,}625\,\text{kHz} \tag{1}$$

Da der Zähler nach 256 Impulsen überläuft, erhält man als Frequenz des Überlaufs (hier Interruptfrequenz f_{Int} genannt, weil bei jedem Überlauf ein Interrupt ausgelöst wird:

$$f_{Int} = f_{count}/256 = 15{,}625\,\text{kHz}/256 = 61{,}03\,\text{Hz} \tag{2}$$

Das folgende Programm macht die einen Ton mit der Hälfte dieser Frequenz hörbar[2] (`beisp_timer0.asm`):

[1] Falls man keinen Interrupt benutzt, muss man es anschließend von Hand wieder zurücksetzen.
[2] die Hälfte deshalb, weil alle 1/61 ms umgeschaltet wird und eine Periode 2/61 ms lang dauert

```
1  .include "/usr/share/avra/m32def.inc"
2          jmp main
3  .org ovf0addr
4          jmp timer0       ;12 Timer 0 OVF
5  main:   ldi r16, 0x80    ; PD7 als Ausgang
6          out DDRD, r16
7          ldi r16, low(RAMEND)   ; Stackpointer
8          out SPL, r16
9          ldi r16, high(RAMEND)
10         out SPH, r16
11         ldi r16, 0b0101 ; a) Timer 0 erhoehen bei
12         out TCCR0, r16  ; fCLK/1024
13         ldi r16, 0b01   ; b) Timer 0 Overflow Interrupt Enable
14         out TIMSK, r16  ; maskieren
15         sei             ; c) Interrupts erlauben
16         ldi r16, 0
17 ende:   rjmp ende       ; tu nix
18 timer0:                 ; falls Timer 0 ueberlaeuft:
19 ;  push r16             ; R16 sichern
20 ;  push r17             ; R17 sichern
21 ;  in   r16,SREG        ; SREG sichern, Teil 1
22 ;  push r16             ; SREG sichern, Teil 2
23         com r16         ; alle Bits von r16 invertieren
24         out PORTD, r16  ; Summer an/aus
25 ;  pop r16             ; SREG holen, Teil 1
26 ;  out SREG,r16        ; SREG holen, Teil 2
27 ;  pop r17             ; R17 holen
28 ;  pop r16             ; R16 holen
29         reti
```

2.2.5 Einfluss des Startwertes in `TCNT0`

Mit den bisherigen Teilerfaktoren (1, 8, 64, 256 und 1024) kann man nur eine sehr begrenzte Anzahl von Frequenzen erzeugen. Aber es gibt eine Möglichkeit, diese Grenzen zu erweitern: Mit einem geeignetem *Startwert* N_{start} im Register `TCNT0` kann man die Frequenz, in der der Timer überläuft, *nach unten hin* verändern. Der Zähler läuft dann nicht mehr nach 256, sondern nach $256 - N_{start}$ Impulsen über. Setzt man z.B. $N_{start} = 148$, dann erhält man:

$$f_{Int} = \frac{f_{count}}{N_{end} - N_{start}} = \frac{15{,}625\,\text{kHz}}{256 - 148} = 144{,}68\,\text{Hz} \tag{3}$$

N_{end} ist dabei der Wert des Timers, bei dem ein Überlauf eintritt.

2.2.6 Erzeugung einer bestimmten Frequenz

Will man eine bestimmte Frequenz erzeugen, kann man fast schon auf fertige Unterprogramme zurückgreifen (z.B. auf `timer0` in `beisp_timer0.asm`). Es sind nur zwei Dinge zu beachten:

a) Vor dem ersten Auftreten des Interrupts wird das Register `TCNT0` mit dem richtigen Startwert beschrieben.

b) Im Interrupt-Handler muss der Startwert wieder neu in das Register `TCNT0` geschrieben werden. Mit dem Verlassen des Interrupt-Handlers wird ja bekanntlicherweise das `I`-Bit im Statusregister `SREG` wieder gesetzt und der Zähler läuft von diesem Startwert aus weiter.

Im folgenden Beispiel soll eine Frequenz $f_{LS} = 440\,\text{Hz}$ aus dem Lautsprecher ausgegeben werden. Der Lautsprecher wird zweimal in der Periode umgeschaltet; die Interruptfrequenz liegt also bei $f_{Timer} = 880\,\text{Hz}$. Nun formt man einfach Gleichung 3 nach N um:

$$\Delta N = N_{end} - N_{start} = \frac{f_{count}}{f_{Int}} \tag{4}$$

Jetzt setzt man für f_{count} noch den Wert aus Gleichung 1 ein:

$$\Delta N = \frac{f_{CPU}}{f_{Int} N_{Teil}} = \frac{16\,\text{MHz}}{880\,\text{Hz} \cdot N_{Teil}} \tag{5}$$

Dann ergibt sich für die erlaubten Werte von N_{Teil}:

N_{Teil}	1	8	64	256	1024
ΔN	18181,8	2272,7	284,1	71,02	17,76

Man nimmt die größte Zahl kleiner oder gleich 256; das ist in diesem Fall $\Delta N = 71$ bei einem Teiler $N_{Teil} = 256$. Jetzt braucht man nur noch den Startwert auszurechnen:

$$N_{start} = N_{end} - \Delta N = 256 - 71 = 185 \tag{6}$$

Das fertige Programm sieht so aus (`timer0440hz.asm`):

```
1  .include "/usr/share/avra/m32def.inc"
2  .equ    STARTWERT=185
3          jmp main
4  .org ovf0addr
5          jmp timer0              ; Nr. 12: Timer 0 OVF
6  main:
7          ldi r16, 0x80           ; PD7 als Ausgang
8          out DDRD, r16
9          ldi r16, low(RAMEND)    ; Stackpointer
10         out SPL, r16
11         ldi r16, high(RAMEND)
12         out SPH, r16
13         ldi r16, 0b0100         ; a) Timer 0 erhoehen bei
14         out TCCR0, r16          ; fCLK/256
15         ldi r16, 0b01           ; b) Timer 0 Overflow Int. Enable
16         out TIMSK, r16          ; maskieren
17         ldi r16, STARTWERT      ; c)
18         out TCNT0, r16          ; Startwert setzen auf 148
19         sei                     ; d) Interrupts erlauben
20         ldi r16, 0
21 schleife:
22         rjmp schleife    ; tu nix
23 timer0:                  ; falls Timer 0 ueberlaeuft:
24         com r16          ; alle Bits von r16 invertieren
25         out PORTD, r16   ; Summer an/aus
26         ldi r17, STARTWERT ; Startwert setzen auf 148
27         out TCNT0, r17
28         reti
```