C2 Timer

Einführung

Die drei Timer des ATmega32 bieten sehr viele unterschiedliche Betriebsarten. Im folgenden Kapitel sollen nur einige der Betriebsarten kennen gelernt werden.

Was ist ein Timer?

Ein Timerbaustein oder eine Timereinheit ist ein dualer Aufwärtszähler mit der Schrittweite Eins. Er ist zu jeder Zeit lese- und schreibbar und kann vielfältig als Zeitgeber (*timer*) oder Zähler (*counter*) eingesetzt werden.
Als **Timer** wird der Systemtakt zum Zählen verwendet. Mittels Vorteiler kann dieser falls nötig verringert werden.
Als **Counter** erhöhen externe Flanken am Timereingang (z.B. **T0**) den Zählstand.

Der Timer[11] ist eigenständig und läuft immer unabhängig vom Controllerkern und Programm. Nur mit ihr kann man zum Beispiel zeitliche Abstände präzise bestimmen oder mittels Interrupt in genauen Zeitabständen Ereignisse auslösen.

Im ATmega32 bzw. ATmega8 sind drei unabhängige Timereinheiten mit teils unterschiedlichen Eigenschaften verfügbar.

Was kann man mit Timern tun?

Timer kann man einsetzen um Zeitverzögerungen zu bewirken (anstatt von Zeitschleifen), als Frequenzgenerator oder -zähler (-messer), als Ereigniszähler für externe Signale, um Zeitabstände externer Signale zu messen, als Zeitgeber (Uhr), oder zur Pulsweitenmodulation zum Beispiel zum Ansteuern von Motoren,

Der ATmega32 enthält drei Universal-Timer mit folgenden Eigenschaften:

Timer/Counter 0: 8 Bit (0-255)

- Aufwärtszähler mit externem oder internem Takt mit Vorteiler (10 Bit)
- Überlauf-Interrupt (**T0V0**) und Vergleichswertinterrupt (**OCF0**)
- Frequenzgenerator
- Externer Ereigniszähler
- Pulsweitenmodulation PWM (*fast* PWM und *phase correct* PWM)
- Vergleichswertauswertung mit der Möglichkeit den Timer zurückzusetzen (*clear timer on compare match*, CTC-Modus)

11 Der Einfachheit halber wird im Text nicht immer zwischen Timer und Counter unterschieden. Die Bezeichnung "Timer" wird allgemein für die Timereinheit verwendet.

Timer/Counter 1: 16 Bit (0-65535)
zusätzlich zu Timer 0:
- Zwei unabhängige Vergleichswertauswertungen
- Input-Capture-Einheit (mit Rauschunterdrückung). Sie ermöglicht die Zeitmessung externer Signale (Stoppuhr)
- 4 Interruptquellen (Überlauf (**T0V1**), 2 mal Vergleichswert (**OCF1A** und **OCF1B**) und Input-Capture (**ICF1**))

Timer/Counter 2: 8 Bit (0-255)
ähnlich Timer 0, allerdings ohne externen Ereigniszähler!
zusätzlich zu Timer 0:
- Timer 2 kann als Echtzeituhr (*real time clock*, RTC) verwendet werden

Grundsätzliche Funktionsweise des Timers

Ein Timer (Zeitgeber) oder Counter (Zähler) wird gesetzt (initialisiert) und gestartet. Danach läuft[12] (zählt) er unabhängig und unbeeinflusst vom Controllerkern und der anderen Peripherie bis man ihn stoppt. Hat der Timer seinen Höchstwert (255 bzw. 65535 für 8 und 16 Bit) erreicht, so beginnt er wieder bei Null. Bei diesem Überlauf, oder bei einem anderen vorgegebenem Wert wird ein Flag gesetzt, das durch Polling abgefragt werden kann, oder es wird, was meist sinnvoller ist, ein Interrupt ausgelöst.

Man kann den Timer auch beim Erreichen eines Überlaufs oder eines Vergleichswertes ein Signal an einem Pin ausgeben lassen, oder zum Beispiel die Zählrichtung umkehren.

Als Takt kann der AVR-Takt verwendet werden. Ein Vorteiler ermöglicht die Änderung der Taktfrequenz. Es kann aber auch ein externer Takt verwendet werden.

Zum Kennenlernen wird in den folgenden Kapiteln der 8-Bit Timer 0 verwendet. Alle Beispiele lassen sich auch auf die beiden anderen Timer übertragen indem man die Bezeichnung der Register anpasst.

Der Timer als Zeitgeber

Überlauf mit dem Timer 0

Nach der Aktivierung des Timers beginnt dieser gleich mit dem Zählen. Bei jedem Überlauf des Timers wird automatisch das Flag **T0V0** im SF-Register **TIFR** gesetzt. Bei Timer 0 also jedes Mal wenn dieser von 0 (Default-Startwert nach Reset) bis 255 gezählt hat (nach 256 Zählschritten).

Der Timer 0 soll nun bei jedem Überlauf periodisch einen Interrupt auslösen (Timer 0 Overflow Interrupt).

12 Der Timer wird automatisch inkrementiert oder dekrementiert.

Initialisierung des Timer 0 für den Überlauf-Interrupt

Zuerst muss geklärt werden mit welcher Taktrate gezählt werden soll. Diese Einstellung erfolgt mit den Bits 0-2 (**CS00-CS02**) des Timer/Counter Controll Registers **TCCR0**. Folgende Tabelle bietet einen Überblick:

Bit 2	Bit 1	Bit 0	Taktquelle
CS02	CS01	CS00	
0	0	0	Timer Stopp (verbraucht keinen Strom)
0	0	1	Systemtakt (:1)
0	1	0	Systemtakt / 8
0	1	1	Systemtakt / 64
1	0	0	Systemtakt / 256
1	0	1	Systemtakt / 1024
1	1	0	externer Takt: fallende Flanke an **T0** (**PB0**)
1	1	1	externer Takt: steigende Flanke an **T0** (**PB0**)

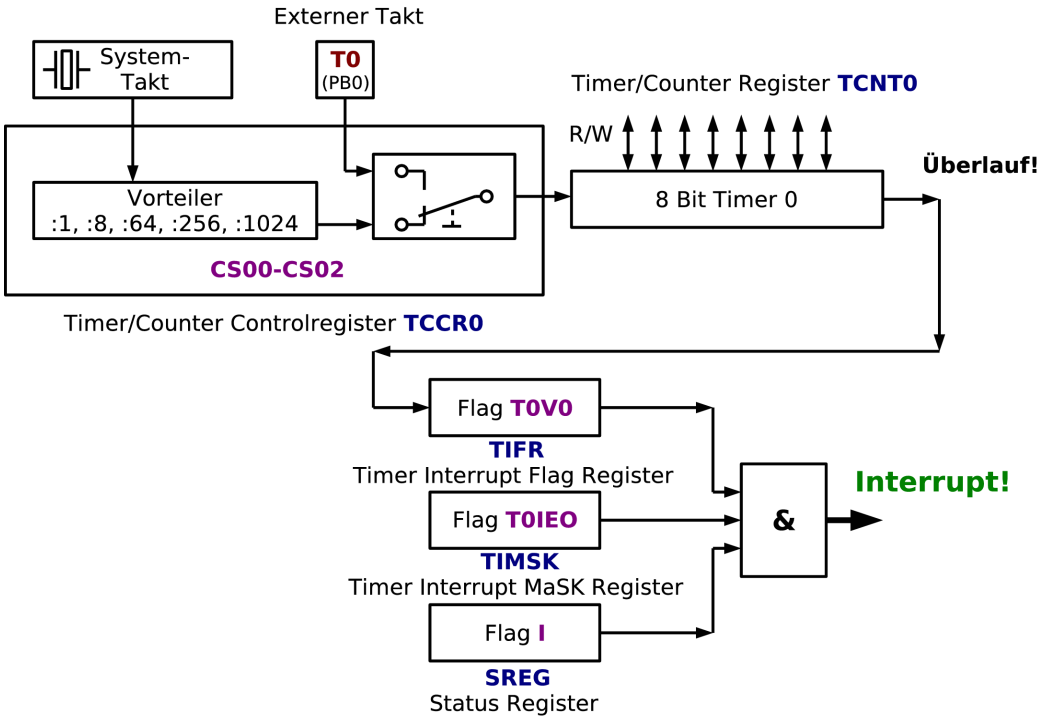
Bemerkung: Durch Setzen der drei Bits **CS00-CS02** auf einen Wert größer als Null wird der Timerbaustein eingeschaltet. Er beginnt nach dieser Initialisierung gleich mit dem Zählen ab Null, wenn kein Wert voreingestellt wurde.

Die Frequenz errechnet sich mit:

$$f = \frac{\text{Systemtakt}}{\text{Vorteiler} \cdot \text{Zählschritte}}$$

Beispiel: Als Vorteiler wurde 1024 gewählt bei einer Quarzfrequenz von 16 MHz. Damit ergibt sich eine Zählfrequenz des Timers von 16 MHz / 1024 = 15625 Hz. Da jedes Interrupt nach 256 Zählschritten erfolgt, wird der Interrupt also mit einer Frequenz von 15625 / 256 = 61,03515625 Hz aufgerufen.

Hinweis: Die obige Tabelle gilt ähnlich für Timer 1. Für Timer 2 ist sie allerdings nicht gültig da dieser keine externen Signale verarbeiten kann. Dafür kann aber zusätzlich ein Vorteiler von 32 und 128 eingestellt werden (siehe Datenblatt).



Damit das Interrupt ausgelöst wird müssen drei Bedingungen erfüllt sein:

1. Das Flag **T0V0** im Timer Interrupt Flag Register **TIFR** muss Eins sein (**T0V0** = 1). Dies wird **automatisch gesetzt** wenn ein Überlauf erfolgt!
2. Der **T0**-Interrupt muss erlaubt sein (**TOIE0** = 1 im Register **TIMSK**).
3. Interrupts müssen global freigegeben sein (**I** in **SREG** = 1 mit "**sei**")

Um den Timer als Zeitgeber im Interrupt-Betrieb zu benutzen, müssen außer der üblichen Initialisierung also noch folgende Schritte erledigt werden:

- Sprungadresse für den Interrupt organisieren (aus der Definitionsdatei: **OVF0addr = 0x0016**) und der Interruptroutine einen Namen zuweisen (z.B. **INTT0**).

```
;-----------------------------------------------------------------------------
;       Sprungadressen fuer die Interrupts organisieren (ISR VECTORS)
;-----------------------------------------------------------------------------
      ;Vektortabelle (im Flash-Speicher)
.ORG    OVF0addr                ;Vektor Nr 12 (Adresse 0x16) fuer
        rjmp INTT0              ;INTT0 (Timer0-Overflow Interrupt)

;-----------------------------------------------------------------------------
;       Initialisierungen und eigene Definitionen
;-----------------------------------------------------------------------------
.ORG    INT_VECTORS_SIZE        ;Platz fuer ISR Vektoren lassen
INIT:
```

- Initialisierung des Stapels nicht vergessen!
- Die Timereinheit mit den Flags **CS00-CS02** im SF-Register **TCCR0** einschalten. Gleichzeitig wird der Vorteiler festgelegt. Da die oberen fünf Bit dieses Registers nur für die PWM-Steuerung zuständig sind, und per default Null sind, muss hier nicht zwingend eine Maskierung eingesetzt werden

```
        ;Timer einschalten und Vorteiler festlegen
        ldi     Tmp1,0x02       ;TCCR0 = 0b00000 010 (Systemtakt/8)
        out     TCCR0,Tmp1      ;
```

- **T0IE0** im SF-Register **TIMSK** mittels Maskierung setzen

```
        ;Timer Interrupt erlauben
        in      Tmp1,TIMSK      ;
        ori     Tmp1,0x01       ;Mit einer ODER-Maskierung (00000001b)
                                ;Bit 0 (TOIE0) im TIMSK-Register auf eins setzen
        out     TIMSK,Tmp1      ;Timer 0-Overflow-Interrupt ein
```

- Interrupts global freigeben (sind bei Reset des Controllers per default gesperrt)

```
         ;Interrupts freigeben
         sei                    ;Interrupts global freigeben (I-Bit im SREG)
```

- Interruptroutine schreiben. Innerhalb der Routine müssen die Flags (**SREG**) und alle!, innerhalb der Routine benutzten Register, auf den Stapel gerettet werden.

✎ **C200** Es soll am Pin **PA0** mittels Timer 0-Interrupt ein Rechtecksignal durch Toggeln des Ausgangs erzeugt werden. Es wird ein Frequenzzähler oder Oszilloskop an das Pin angeschlossen. Um zu zeigen, dass die CPU (das Hauptprogramm) dadurch nicht belastet wird, soll das Hauptprogramm im Sekundentakt von Null an aufwärts zählen und den Zählerstand am Sieben-Segment-Display ausgeben. Der Code für das Hauptprogramm ist vorgegeben (Unterprogrammbibliotheken für Display und Zeitschleifen müssen eingebunden werden und die entsprechenden Ports initialisiert werden!):

```
;-------------------------------------------------------------------------------
;       Hauptprogramm
;-------------------------------------------------------------------------------
MAIN:   rcall   NDISP           ;Darstellungsunterprogramm aufrufen
        adiw    YL,1            ;Zaehler inkrementieren
        rjmp    MAIN            ;Weiter
```

a) Errechne die zu erwartende Frequenz, wenn bei der Initialisierung des Timers der Vorteiler auf 256 eingestellt wird (Quarzfrequenz 16 MHz).
b) Zeichne das Flussdiagramm und erstelle das Programm.
Nenne das Programm "**C200_8bit_timer0_1.asm**"
c) Überprüfe die Frequenz mittels Zähler oder Oszilloskop.

Überlauf mit dem Timer 0 mit Voreinstellung

Das Timerzählregister **TCNT0** kann nicht nur ausgelesen werden, sondern auch beschrieben werden. Dies ermöglicht es die Zählschritte bis zum Überlauf zu verringern. Wird das Register zum Beispiel mit dem Startwert 200 voreingestellt, so zählt der Timer von 200 bis 255, also nur 56 Schritte.

Soll jetzt, um zum Beispiel eine höhere Frequenz zu erzeugen, der Timer nach jedem Überlauf mit einem festen Startwert versehen werden, so muss das gleich am Anfang der Interruptroutine (vor den Push-Befehlen!) geschehen um die Genauigkeit des Timers nicht zu gefährden. Am einfachsten wird der Startwert in einer globalen Variablen bei der Initialisierung festgelegt.

✎ **C201** Ändere das Programm **C200** so um, dass der Timer 0 immer mit dem Startwert 150 beginnt.
Nenne das neue Programm "**C201_8bit_timer0_2.asm**".
a) Ermittle rechnerisch und per Zähler (Oszilloskop) die erzeugte Frequenz.
b) Errechne den Startwert, der benötigt wird um eine Frequenz von 440 Hz (Kammereton A (LA)) zu erzeugen. Überprüfe das Ergebnis.

✎ **C202** Ändere das Programm **C201** so um, dass im Hauptprogramm im Sekundenrythmus der Startwert des Timers ab Null jeweils um 5 erhöht wird.
Nenne das neue Programm "**C202_8bit_timer0_3.asm**".

Bemerkung: Der Timer 0 hat auch eine Funktion um Frequenzen ohne Interrupt zu erzeugen (Frequenzgenerator)! Ein fest zugeordnetes Pin (**OC0** (**PB3**)) wird bei entsprechender Initialisierung getoggelt sobald ein im Output Compare Register **OCR0** eingetragener Wert erreicht wurde (siehe PWM).

Der Timer als Zähler (Counter)

Über ein reserviertes Pin kann die fallende oder die steigende Flanke eines externen Signals dazu verwendet werden den Zähler zu erhöhen. So ist es möglich den Timer als Zähler externer Ereignisse einzusetzen.

Timer 0 als Zähler (Counter)

Um den Timer als Zähler einzusetzen, reicht es, über die Bits **CS00-CS02** im Timer/Counter Control Register **TCCR0** statt einem internen Takt einen externen Takt auszuwählen. Dabei kann die steigende Flanke oder die fallende Flanke des externen Signals zum Zählen genutzt werden. Es ist sinnvoll, da meist mit Pull-Up Widerständen gearbeitet wird (negative Logik), die fallende Flanke zum Zählen zu verwenden. Das externe Signal muss an Pin **T0** (**PB0**) angelegt werden.

✎ **C203** Ein nicht entprellter Taster soll als Signalquelle für den Zähler dienen (**T0** als Eingang initialisieren und Pull-Up einschalten!). Der Zählstand (**TCNT0**) soll im Hauptprogramm im Sekundentakt angezeigt werden. Der Zähler ist auf den Wert 246 voreingestellt, und beim Überlauf soll eine LED (**PA0**) umgeschaltet werden.
Nenne das neue Programm "**C203_8bit_counter0_1.asm**".

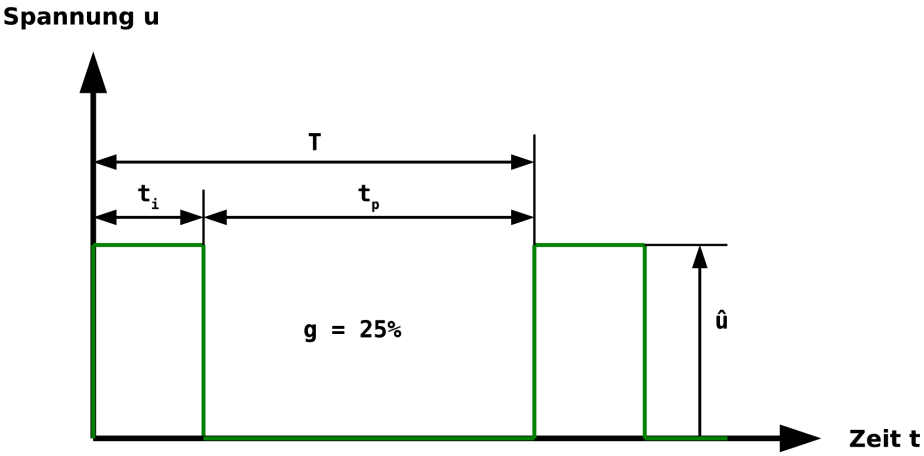
Pulsweitenmodulation mit dem Timer

Die Leistung kann bei vielen elektrischen Geräten durch Veränderung der Spannung gesteuert werden. Es gibt aber auch Geräte wo dies nicht zufriedenstellend funktioniert, da diese Geräte zum Beispiel eine Mindestspannung zum Arbeiten benötigen (LED, Motor). Die Pulsweitenmodulation bietet hier Abhilfe und lässt sich (anders als eine variable Spannung) sehr leicht mit einem Mikrocontroller realisieren.

Bei der Pulsweitenmodulation wird eine Rechteckspannung mit fester Frequenz verwendet, bei der der Tastgrad[13] (*duty cycle*) verändert wird. Die Impulsweite im Verhältnis zur Periodendauer wird meist in Prozent ausgedrückt.

$$\text{Impulsdauer } t_i \qquad \text{Pausendauer } t_p$$

$$\text{Tastgrad } g = \frac{t_i}{T} \qquad \text{Periodendauer } T = t_i + t_p$$



Pulsweitenmodulation mit Timer 0

Die Pulsweitenmodulation lässt mit dem Timer relativ einfach softwaremäßig realisieren. Bei dieser Variante mit Überlauf- und Vergleichswert-Interrupt kann dann jedes beliebige Ausgangspin genutzt werden.

Noch einfacher geht es allerdings wenn der Timer alle Aufgaben selber übernimmt. Dieser von ATMEL® genannte "Fast PWM"-Modus benötigt keine Interruptroutinen und schont somit die Ressourcen des Controllers. Als Ausgang muss für Timer 0 der **OC0**-Pin (**PB3**) genutzt werden.

Funktionsweise des "Fast PWM"-Modus:

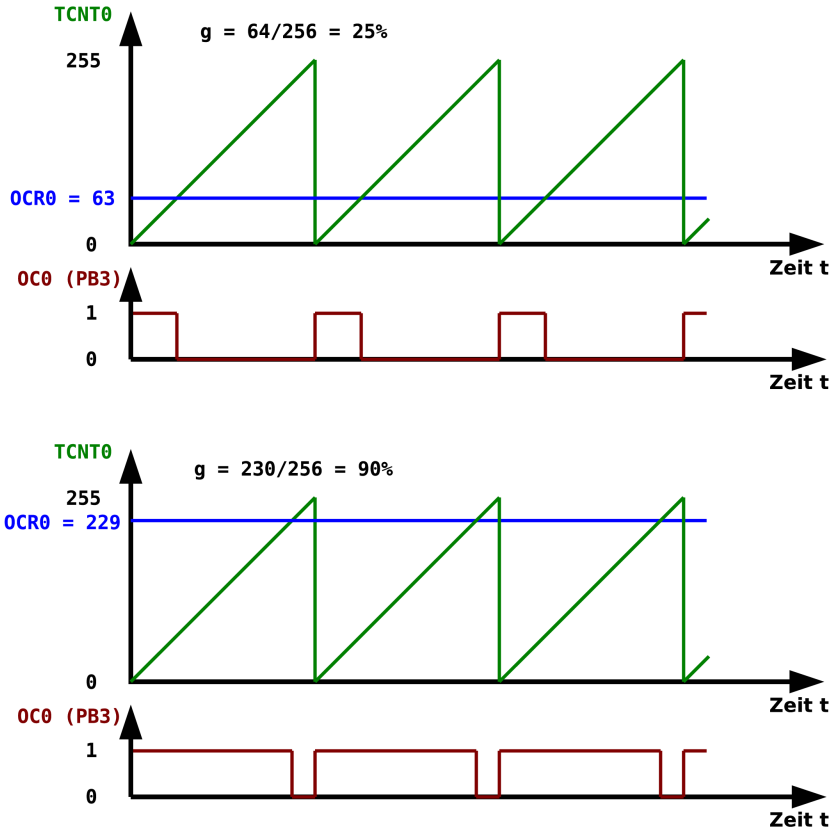
Der Timer 0 zählt durchgehend von 0 bis 255 und beginnt dann wieder bei 0 (Zählregister **TCNT0**). Ein gesamter Durchlauf entspricht der Periodendauer des PWM-Signals. Die Frequenz wurde durch den Systemtakt und den Vorteiler bestimmt.

13 Oft findet man auch den Begriff "Tastverhältnis". Dieser Begriff ist allerdings nicht einheitlich definiert. Oft findet man ihn als Kehrwert des Tastgrads, manchmal ist er gleich wie der Tastgrad definiert, oder auch als Verhältnis zwischen Impuls- und Pausendauer.

$$f = \frac{\text{Systemtakt}}{\text{Vorteiler} \cdot 256}$$

Am Ausgangspin **OC0** liegt High-Pegel an (nicht-invertierender Modus).

In einem zweiten Register, dem sogenannten Vergleichsregister **OCR0**, wird ein Vergleichswert festgelegt. Erreicht der Zählwert diesen Vergleichswert, so kippt der Pegel am Ausgangspin **OC0** auf Low-Pegel. Der Zähler zählt ungerührt weiter um beim Überlauf das Ausgangspin wieder auf High-Pegel zu setzen. Der Vergleichswert bestimmt den Tastgrad!



Initialisierung des Timer 0 für den "Fast-PWM"-Modus

Um den "Fast PWM"-Modus einzuschalten müssen die Bits **WGM00** und **WGM01** im Kontrollregister **TCCR0** beide auf Eins gesetzt werden. In diesem Modus wird mit **COM01** = 1 und **COM00** = 0 (beide ebenfalls im Kontrollregister **TCCR0**) der nicht invertierende PWM-Modus ausgewählt (invertierender Modus: Modus: **COM01** = 1 und **COM00** = 1). Über die Bits **CS00-CS02** wird über den Vorteiler die Frequenz eingestellt. Das letzte freie Bit (**FOC0**) im Register **TCCR0** muss in diesem Modus auf Null gesetzt werden.

Zusätzlich zur Initialisierung des Kontrollregisters **TCCR0** muss nur noch das Pin **OC0** (**PB3**) als Ausgang initialisiert werden, und ein Vergleichswert muss ins Vergleichsregister **OCR0** Register geschrieben werden.

Die ganze Initialisierung besteht zum Beispiel aus folgenden Zeilen:

```
;viertes Pin von Port B (PB3) als Ausgang initialisieren
sbi     DDRB,3          ;Ausgang

;Timer einschalten, Vorteiler festlegen und Fast PWM waehlen
ldi     Tmp1,0x6C       ;TCCR0 = 0b01101100 (Fast PWM,nicht inv., 1/256)
out     TCCR0,Tmp1      ;
;Vergleichswert fuer PWM festlegen
ldi     Tmp1,63         ;Vergleichswert festlegen
out     OCR0,Tmp1       ;
```

✎ **C204** Die Helligkeit einer LED soll im Sekundenrhythmus erhöht werden. Dazu soll im Hauptprogramm einmal pro Sekunde der Vergleichswert ab 0 um 5 erhöht werden. Die Siebensegmentanzeige gibt gleichzeitig den Vergleichswert aus.

a) Schreibe das Programm und nenne es "**C204_8bit_timer0_pwm_1.asm**".
b) Betrachte die Spannung mit einem Oszilloskop. Interessant ist auch die Ausgabe mittels Summer.
c) Welchen Einfluss hat die Frequenz?

Bemerkung: Im normalen Modus kann mit Hilfe des Vergleichswertregisters ein zusätzlicher Interrupt bei Übereinstimmung des Zählwertes mit dem Vergleichswert ausgelöst werden (*Output Compare Match Interrupt*). Im sogenannten CTC-Modus (*Clear Timer on Compare Match Mode)* kann der Timer bei Erreichen des Vergleichswertes auch zurückgesetzt werden.

Die SF-Register des Timer 0

Das Timer/Counter Control Register TCCR0

Das **Timer/Counter Control Register 0** befindet sich auf der SRAM-Adresse **0x0053** (SF-Register-Adresse **0x33**) und wird mit der Abkürzung "**TCCR0**" angesprochen (Definitionsdatei). Die Befehle **sbi**, **cbi**, **sbic** und **sbis** können nicht verwendet werden.

TCCR0 = Timer/Counter Control Register 0

Bit	7	6	5	4	3	2	1	0
TCCR0 0x33	**FOC0**	**WGM00**	**COM01**	**COM00**	**WGM01**	**CS02**	**CS01**	**CS00**
Startwert	0	0	0	0	0	0	0	0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

WGM0n ***Waveform Generation Mode*** ***WGM01, WGM00***

Mit diesen zwei Bit wird der Operationsmodus des Timer festgelegt:

WGM01 WGM00 2^1 2^0	Modus:
00	Normaler Modus
01	Phasenkorrekter PWM-Modus
10	CTC-Modus (clear timer on compare match)
11	Fast PWM-Modus

COM0n ***Compare Match Output Mode*** ***COM01, COM00***

Mit diesen zwei Bit wird das Verhalten des Ausgangspin **OC0** festgelegt.
Je nach Modus ändert das Verhalten:

COM01 COM00 2^1 2^0	Verhalten im normalen Modus:
00	**OC0** abgeschaltet (normales Port-Pin)
01	Toggele **OC0** bei Vergleichsübereinstimmung
10	Lösche **OC0** bei Vergleichsübereinstimmung
11	Setze **OC0** bei Vergleichsübereinstimmung
COM01 COM00 2^1 2^0	Verhalten im Fast-PWM Modus:
00	**OC0** abgeschaltet (normales Port-Pin)
01	Reserviert
10	nicht-invertierender Fast-PWM-Modus
11	invertierender Fast-PWM-Modus

CS0n ***Clock Select Timer 0*** ***CS02, CS01, CS00***

Mit diesen drei Bit wird die Taktquelle für Timer 0 festgelegt.

CS02 CS01 CS00 2^2 2^1 2^0	Taktquelle:
000	Timer Stopp (verbraucht keinen Strom, default nach RESET!)
001	Systemtakt (:1)
010	Systemtakt / 8 (
011	Systemtakt / 64
100	Systemtakt / 256
101	Systemtakt / 1024
110	externer Takt: fallende Flanke an **T0** (**PB0**)
111	externer Takt: steigende Flanke an **T0** (**PB0**)

Weitere Informationen findet man im Datenblatt.

Das Timer/Counter Interrupt Mask Register TIMSK

Das **Timer/Counter Interrupt Mask Register** befindet sich auf der SRAM-Adresse **0x0059** (SF-Register-Adresse **0x39**) und wird mit der Abkürzung "**TIMSK**" angesprochen (Definitionsdatei). Die Befehle **sbi**, **cbi**, **sbic** und **sbis** können nicht verwendet werden.

TIMSK = Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
TIMSK 0x39	**OCIE2**	**TOIE2**	**TICIE1**	**OCIE1A**	**OCIE1B**	**TOIE1**	**OCIE0**	**TOIE0**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

OCIE0 ***Timer/Counter 0 Output Compare Match Interrupt Enable***

0 kein **OCF0**-Interrupt erlaubt.

1 Das Setzen dieses Bit **ermöglicht das Auslösen eines Interrupts** in dem Moment, wo das **OCF0**-Flag (**TIFR**) gesetzt wird, also eine Übereinstimmung zwischen dem Zählregister **TCNT0** und dem Vergleichsregister **OCR0** stattgefunden hat, oder wenn das Flag manuell gesetzt wurde. Interrupts müssen dazu global frei gegeben sein (**I**=1 im **SREG** mit "**sei**").

TOIE0 ***Timer/Counter 0 Overflow Interrupt Enable***

0 kein **TOV0**-Interrupt erlaubt.

1 Das Setzen dieses Bit **ermöglicht das Auslösen eines Interrupts** in dem Moment, wo das **TOV0**-Flag (**TIFR**) gesetzt wird, also ein Überlauf auftrat oder wenn das Flag manuell gesetzt wurde. Interrupts müssen dazu global frei gegeben sein (**I**=1 im **SREG** mit "**sei**").

Das Timer/Counter Interrupt Flag Register TIFR

Das **Timer/Counter Interrupt Flag Register** befindet sich auf der SRAM-Adresse **0x0058** (SF-Register-Adresse **0x38**) und wird mit der Abkürzung "**TIFR**" angesprochen (Definitionsdatei). Die Befehle **sbi**, **cbi**, **sbic** und **sbis** können nicht verwendet werden.

TIFR = Timer/Counter Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
TIFR 0x38	**OCF2**	**TOV2**	**ICF1**	**OCF1A**	**OCF1B**	**TOV1**	**OCF0**	**TOV0**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

OCF0 ***Output Compare Flag 0***

0 Keine Übereinstimmung des Inhalts des Zählregister mit dem Inhalt des Vergleichsregister.

1 Bei Übereinstimmung (*compare match*) des Wertes im im Zählregister **TCNT0** mit dem Wert im Vergleichsregister **OCR0** wird das Flag gesetzt. Das Flag wird automatisch gelöscht, wenn der **OC**-Interrupt ausgeführt wird. Manuell kann das Flag durch das **Schreiben einer Eins!** gelöscht werden.

TOV0 ***Timer/Counter 0 Overflow Flag***

0 Keine Überlauf eingetreten.

1 Tritt ein Überlauf des Zählregisters **TCNT0** auf so wird das Flag gesetzt. Das Flag wird automatisch gelöscht, wenn der **OV**-Interrupt ausgeführt wird. Manuell kann das Flag durch das **Schreiben einer Eins!** gelöscht werden.

Das Timer/Counter Register 0 TCNT0

Das **Timer/Counter Register 0** befindet sich auf der SRAM-Adresse **0x0052** (SF-Register-Adresse **0x32**) und wird mit der Abkürzung "**TCNT0**" angesprochen (Definitionsdatei). Die Befehle **sbi**, **cbi**, **sbic** und **sbis** können nicht verwendet werden. Das Register ist les- und schreibbar.

TCNT0 = Timer/Counter Register 0

Bit	7	6	5	4	3	2	1	0
TCNT0 0x32	**TCNT07**	**TCNT06**	**TCNT05**	**TCNT04**	**TCNT03**	**TCNT02**	**TCNT01**	**TCNT00**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Das Output Compare Register 0 OCR0

Das **Output Compare Register 0** befindet sich auf der SRAM-Adresse **0x005C** (SF-Register-Adresse **0x3C**) und wird mit der Abkürzung "**OCR0**" angesprochen (Definitionsdatei). Die Befehle **sbi**, **cbi**, **sbic** und **sbis** können nicht verwendet werden. Das Register ist les- und schreibbar.

OCR0 = Output Compare Register 0

Bit	7	6	5	4	3	2	1	0
OCR0 0x3C	**OCR07**	**OCR06**	**OCR05**	**OCR04**	**OCR03**	**OCR02**	**OCR01**	**OCR00**
Startwert	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Weitere Aufgaben

✎ **C205** Ändere das Programm **C200** so um, dass es mit dem 16-Bit-Timer 1 arbeitet. Es soll die gleiche Frequenz erzeugt werden! Ermittle die für Timer 1 benötigten Register (Datenblatt).
Nenne das neue Programm "**C205_16bit_timer1_1.asm**".

(Achtung!! Wenn mit 16 Bit (SF Doppelregister) gearbeitet wird soll bei beiden in oder out Befehlen cli vor und sei hinten angestellt werden damit Interrupts das Lesen nicht beeinflussen können!!)