Skriptum Mikrocontroller

Teil 6 Timer/Counter

DI Wolfgang Zukrigl, HTL Litec
DI(FH) Andreas Pötscher, HTL Litec

Inhaltsverzeichnis

Datum	Beschreibung der durchgeführten Änderung	Autor
12.05.2014	V1.0 . . . Korrektur kleinerer Fehler	Wolfgang Zukrigl, HTL Linz–Paul-Hahn-Straße (LiTec)

Datum	Beschreibung der durchgeführten Änderung	Autor
19.09.2014	V1.1 ... Übernahme in Markdown und allgemeine Überarbeitung	Wolfgang Zukrigl, HTL Linz–Paul-Hahn-Straße (LiTec)
24.02.2019	V2.0 ...: Einfügen von Merksätzen, Beispielen, Aufgaben. Viele Teile neu überarbeitet.	Wolfgang Zukrigl, HTL Linz–Paul-Hahn-Straße (LiTec)
01.04.2020	V2.1 ... Fehler bei Beispiel ausgebessert (PE6/INT6 auf PE5/INT5 geändert)	Wolfgang Zukrigl, HTL Linz–Paul-Hahn-Straße (LiTec)
19.03.2025	V2.2 ... Übernahme in Markdown und allgemeine Überarbeitung	Andreas Pötscher, HTL Linz–Paul-Hahn-Straße (LiTec)

Anwendung von Timer-Counter-Betrieb

Die meisten Mikrocontroller und Prozessoren enthalten einen oder mehrere Timer/Counter als Peripheriekomponenten. Ein Timer/Counter kann entweder als Timer oder als Counter verwendet werden:

Verwendung als Timer:

In dieser Betriebsart lassen sich exakte Zeitvorgaben realisieren. So kann zum Beispiel alle 100 Millisekunden ein Interrupt ausgelöst werden, womit sich etwa die Zeitmessung für eine Stoppuhr umsetzen lässt. Diese Zeitspanne wird vom CPU-Takt abgeleitet, der bei AVR-Mikrocontrollern meist von einem Quarzoszillator erzeugt wird. Die Frequenztoleranz eines 16-MHz-Quarzes beträgt typischerweise zwischen 10 ppm und 30 ppm (0,001 % bis 0,003 %). Eine auf dieser Basis arbeitende Stoppuhr würde somit pro Tag lediglich um etwa 0,8 bis 2,6 Sekunden abweichen.

Verwendung als Counter:

In dieser Betriebsart zählt der Counter automatisch mit, wie oft ein bestimmtes Hardware-Ereignis auftritt. Ein typisches Beispiel ist ein Magnet, der in die Speichen eines Fahrradreifens integriert ist und bei jeder Umdrehung einen elektrischen Impuls auslöst – also eine steigende oder fallende Flanke erzeugt. Bei einem Counter führt eine dieser Flanken automatisch zu einer Erhöhung des Zählerstands um eins (Inkrementierung). Der aktuelle Zählerstand entspricht somit der bislang zurückgelegten Strecke in Radumfängen. (Der Umfang eines 28Reifens eines Rennrads beträgt beispielsweise etwa 2,10 m.) Das Hauptprogramm in der CPU muss diesen Zählerstand lediglich auslesen, in Kilometer umrechnen und auf einem Display anzeigen.

> **Wichtig!**
>
> Timer/Counter werden als Timer verwendet, um exakt zeitgesteuerte Aufgaben zu erledigen. Sie werden als Counter verwendet, um das Auftreten von Hardware-Ereignissen zu zählen.

Die Timer/Counter des ATmega 2560

Der ATmega2560 verfügt insgesamt über sechs voneinander unabhängige Timer/Counter. Das Herzstück jedes Timer/Counters ist ein Zählerstandsregister, dessen Wert durch bestimmte Zählimpulse inkrementiert wird (d. h. um eins erhöht). Nach einem Reset des Mikrocontrollers steht der Zählerstand eines solchen Registers auf 0. Beim ersten eintreffenden Zählimpuls wird der Registerinhalt automatisch – also ohne Eingreifen des CPU-Programms – von 0 auf 1 erhöht, beim nächsten Impuls von 1 auf 2 usw.

Im Counter-Betrieb entstehen diese Zählimpulse durch steigende oder fallende Spannungsflanken an einem bestimmten Pin des Mikrocontrollers. Durch Auslesen des Zählerstandsregisters kann das Hauptprogramm ermitteln, wie viele dieser Flanken bisher aufgetreten sind.

Beim Timer/Counter1 trägt das Zählerstandsregister die Bezeichnung TCNT1, beim Timer/Counter3 entsprechend TCNT3, und so weiter.

Die sechs Timer/Counter des ATmega2560 tragen die Bezeichnungen Timer/Counter0 bis Timer/Counter5:

- Timer/Counter0 und Timer/Counter2 verwenden 8 Bit breite Register. Das bedeutet, dass das Zählerstandsregister Werte von 0 bis 255 annehmen kann. Solche Timer/Counter werden als 8-Bit-Timer/Counter bezeichnet.

- Timer/Counter1, Timer/Counter3, Timer/Counter4 und Timer/Counter5 besitzen 16 Bit breite Register. Die möglichen Werte des Zählerstandsregisters liegen daher zwischen 0 und 65535.

Die 8-Bit-Timer/Counter werden typischerweise für spezielle Aufgaben eingesetzt und werden in diesem Zusammenhang nicht weiter behandelt.

Die 16-Bit-Timer/Counter sind untereinander identisch aufgebaut. Im Folgenden wird exemplarisch der Timer/Counter1 behandelt; die anderen 16-Bit-Timer/Counter können auf die gleiche Weise verwendet werden.

> **Wichtig!**
>
> Der ATmega2560 hat zwei 8-Bit-Timer/Counter und vier 16-Bit-Timer/Counter namens Timer/Counter1, 3, 4 und 5. Herzstück jedes Timer/Counters ist ein 8 bzw. 16 Bit breites Zählerstands-Register, das durch bestimmte Zähl-Impulse ständig inkrementiert wird.

Auslesen des aktuellen Timer/Counter-Zählerstands im C-Programm

Um die Special-Function-Register (SFR) direkt manipulieren zu können, ist – wie üblich – folgender include nötig:

```
#include <avr/io.h>
```

Der aktuelle Zählerstand ist im Register TCNTn gespeichert, wobei n die Nummer des verwendeten Timer/Counters ist. Für den Timer/Counter 1 also.

```
uint16_t cnt;
cnt = TCNT1;
```

Clock-Select-Logic - Auswahl der Zähl-Impulse

Im folgenden Bild ist dargestellt, welche Möglichkeiten zur Auswahl der Zählimpulse für den Timer/Counter1 bestehen (Clock Select Logic). Jeder Zählimpuls bewirkt eine Inkrementierung des Zählerstands, das heißt: Der Zählerstand des Timer/Counters wird bei jedem Impuls um 1 erhöht.

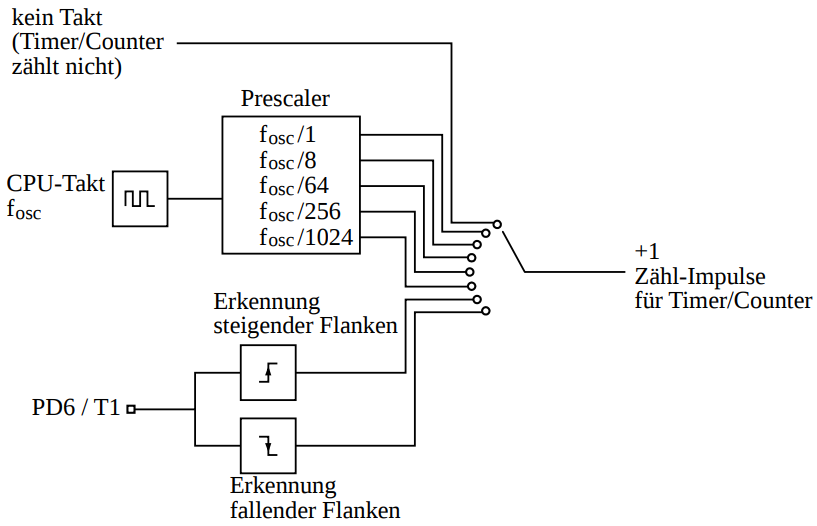

Abbildung 1: Clock-Select-Logic von Timer/Counter 1

Der Schalter im obigen Bild symbolisiert eine von acht Auswahlmöglichkeiten:

- Oberste Stellung: Der Timer/Counter ist gestoppt; der Zählerstand bleibt unverändert.
- Nächste fünf Stellungen: Es liegt Timer-Betrieb vor. Der Zählerstand wird durch die CPU-Taktfrequenz, geteilt durch einen von fünf möglichen Prescaler-Werten, erhöht.
- Unterste zwei Stellungen: Hier handelt es sich um Counter-Betrieb. Der Zählerstand wird durch steigende oder fallende Flanken am Pin PD6, der die alternative Funktion T1 besitzt, erhöht.

Für die anderen Timer/Counter muss im Datenblatt des Mikrocontrollers nachgeschlagen werden, welche Pins für den Counter-Betrieb vorgesehen sind.

Der Prescaler

Im Timer-Betrieb wird die CPU-Taktfrequenz verwendet, um den Zählerstand eines Timer/Counters zu erhöhen.

Bei Arduino-Boards wird die CPU-Taktfrequenz von einem $f_{osc} = 16\,\text{MHz}$ Schwingquarz erzeugt. Dieses Taktsignal durchläuft eine spezielle Hardware-Schaltung, den sogenannten Prescaler (auch Vorteiler genannt). Dabei handelt es sich um eine einfache Zähler- bzw. Frequenzteilerschaltung.

Die Timer Frequenz f_T berechnet sich einfach aus folgender Formel. Wobei N der Prescaler Wert ist.

$f_T = \frac{f_{osc}}{N}$

Der Prescaler kann fünf verschiedene Ausgangs-Taktsignale mit unterschiedlichen Frequenzen f_T und somit auch Periodendauern T_T erzeugen:

Prescaler-Wert	Timerfrequenz f_T	Periodendauer T_T
1	16 MHz	0,0625 μs
8	2 MHz	0,5 μs
64	250 kHz	4 μs
256	62,5 kHz	16 μs

Prescaler-Wert	Timerfrequenz f_T	Periodendauer T_T
1024	15,625 kHz	64 μs

> **Beispiel:**
>
> Nach einem Reset beträgt der Zählerstand aller Timer/Counter gleich 0. Angenommen, in der Initialisierungsphase des Hauptprogramms wird beim Timer/Counter1 ein Prescaler-Wert von 256 ausgewählt. Dadurch startet der Timer/Counter1 und sein Zählerstand beginnt sich von null weg zu erhöhen. Wie lange dauert es, bis ein Zählerstand von 32000 erreicht ist?

Lösung:

Die Frequenz des Taktsignals, das den Zählerstand erhöht, hat eine Frequenz von

$f_T = \frac{f_{osc}}{256} = \frac{16MHz}{256} = 62,5kHz$

Die Periodendauer ist der Kehrwert daraus:

$T_T = \frac{1}{f_T} = \frac{1}{62,5kHz} = 16\mu s$

Direkt nach dem Starten des Timer/Counter1 durch Auswahl des Prescaler-Werts von 256 beträgt der Zählerstand noch 0. Eine Zeitspanne von $16\mu s$ später (T_T) wird der Zählerstand von 0 auf 1 inkrementiert. Weitere $16\mu s$ später, also $32\mu s$ nach dem Starten des Timer/Counters wird der Zählerstand von 1 auf 2 inkrementiert.

Der Zählerstand erhöht sich genau alle $16\mu s$ um 1. Er ist somit eine Stufenfunktion der Zeit.

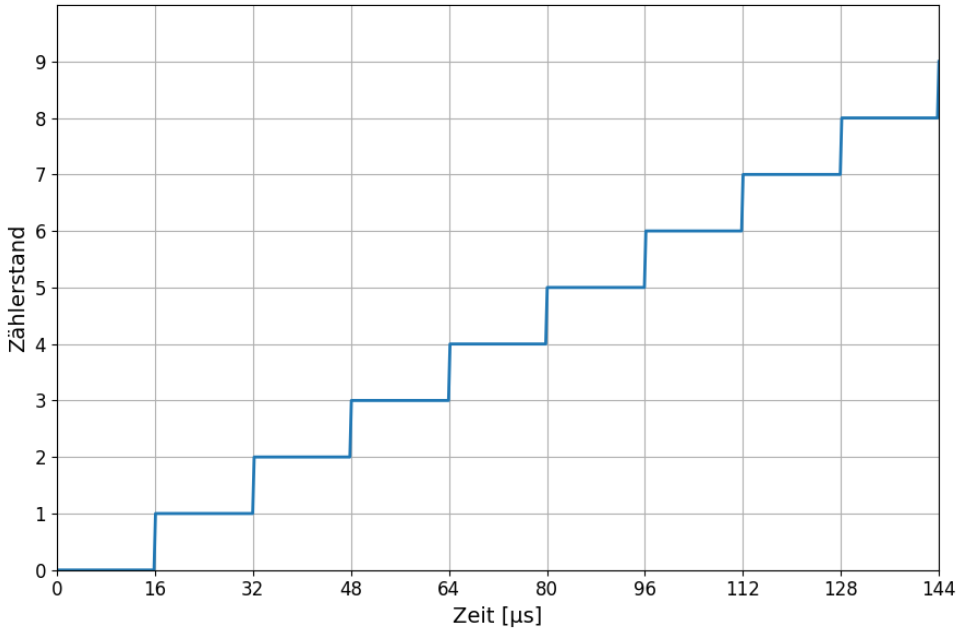

Abbildung 2: Zählerstand am Anfang

Insgesamt dauert es bis ein Zählerstand von 32000 erreicht wird dann

$32000 * T_T = 32000 * 16\mu s = 512ms$

Bei einem Zählerstand von 32000 ist die Stufenfunktion Aufgrund der vielen Punkte nicht mehr zu sehen.

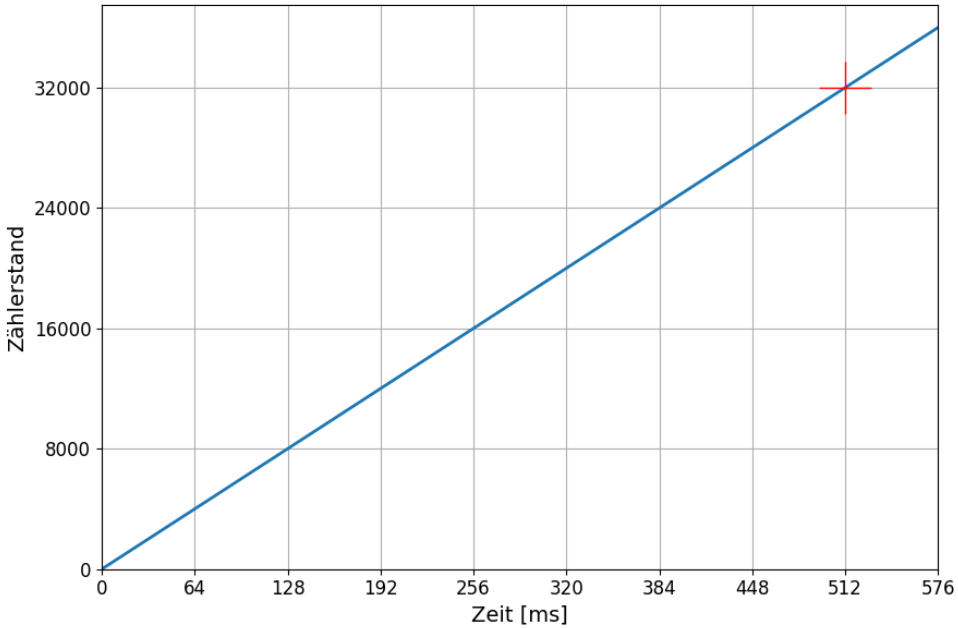

Abbildung 3: Nach 512 ms wird ein Wert von 32000 erreicht

> **Wichtig!**
>
> Im Timer-Betrieb wird die CPU-Taktfrequenz durch einen Prescaler mit einem wählbaren Teilungsfaktor reduziert. Das so erzeugte Taktsignal dient als Zählimpulsquelle für das Zählerstandsregister. Im Counter-Betrieb entstehen die Zählimpulse durch steigende oder fallende Flanken an einem festgelegten Pin des Mikrocontrollers.

Programmieren der Clock-Select-Logik

Die Auswahl einer Taktquelle startet den Timer/Counter sofort. In der Praxis erfolgt dieser Schritt während der Initialisierungsphase des Programms – nachdem der Timer/Counter vollständig konfiguriert wurde, typischerweise in der Funktion `main()`, und zwar vor dem Eintritt in die Endlosschleife.

Die Auswahl der Taktquelle geschieht durch das Setzen der Bits `CSn2`, `CSn1` und `CSn0` (CS steht für Clock Select) im Timer/Counter Control Register B (`TCCRnB`). Dabei steht das „n“ für die Nummer des jeweiligen Timer/Counters und ist entsprechend zu ersetzen.

Als Beispiel wird dieses Special-Function-Register für Timer/Counter1 gezeigt (`TCCR1B`):

Bit	7	6	5	4	3	2	1	0
TCCR1B:	—	—	—	WGM13	WGM12	CS12	CS11	CS10

Die Bedeutung der WGMnx-Bits wird weiter unten erklärt. Die Bits Nr. 7 bis 5 sind hier nicht weiter relevant, und sollten beim Beschreiben des Registers mit 0 beschrieben werden. Mit den drei Bits CSn[2..0] kann für jeden Timer/Counter eine der acht möglichen Taktquellen gewählt werden:

CSn2	CSn1	CSn0	Taktquelle für Timer/Counter
0	0	0	keine Taktquelle der Timer ist gestoppt.
0	0	1	Prescaler 1 $f_T = 16MHz$
0	1	0	Prescaler 8 $f_T = 2MHz$
0	1	1	Prescaler 64 $f_T = 250kHz$
1	0	0	Prescaler 256 $f_T = 62,5kHz$
1	0	1	Prescaler 1024 $f_T = 15,625kHz$
1	1	0	Fallende Flanke am GPIO Pin mit der Funktion "Tn"
1	1	1	Steigende Flanke am GPIO Pin mit der Funktion "Tn"

> **Beispiel:**
>
> Timer/Counter3 soll als Timer mit einem Prescaler-Wert von 256 betrieben werden.

Lösung: Im Register `TCCR3B` müssen die Bits `CS32`, `CS31`, `CS30` auf den Binärcode `100` gesetzt werden, damit der Timer mit einer Zählfrequenz von $16MHz/256$ zu zählen beginnt:

```
TCCR3B |= (0x01<<CS32); //Bit Nr. 2 (CS32) setzen
TCCR3B &= ~( (0x01<<CS31)|(0x01<<CS30) ); //Bits Nr. 1 und 0 (CS31, CS30) löschen
```

Timer-Overflow

Eine Variable vom Datentyp `uint16_t` (also ein 16-Bit breiter, vorzeichenloser Integer) kann Werte im Bereich von 0 bis $2^{16} - 1$ annehmen, also von 0 bis 65.535.

Was passiert, wenn eine solche Variable ihren maximalen Wert hat und anschließend inkrementiert wird? Welchen Wert enthält die Variable a nach Ausführung des folgenden Programmcodes?

```
uint16_t a;
a = 65535;
a++;
```

Der Wert $65535 + 1 = 65536$ (also 2^{16}) liegt außerhalb des darstellbaren Wertebereichs einer `uint16_t`-Variablen. Man spricht in diesem Fall von einem sogenannten Überlauf (Overflow). Die Variable erhält dann wieder den Wert 0.

Dieser Mechanismus ähnelt dem eines mechanischen Zählwerks, wie es früher in Kilometerzählern von Tachos verwendet wurde – und heute noch in Wasseruhren oder Gaszählern zum Einsatz kommt: Stellen Sie sich ein vierstelliges Zählwerk vor, das die Werte von „0000“ bis „9999“ anzeigen kann. Wenn es bereits „9999“ anzeigt und um eins weitergedreht wird, springt es wieder auf „0000“ zurück.

Ein ähnlicher Überlauf tritt auch bei den 16-Bit-Timer/Countern der AVR-Mikrocontroller auf: Wenn der Wert des Zählerstandsregisters den maximalen Wert 65535 erreicht hat (binär: 1111111111111111, hexadezimal: 0xFFFF), wird er beim nächsten Zählimpuls wieder auf 0 zurückgesetzt – ein Overflow tritt ein.

Gleichzeitig wird bei einem Overflow ein Interrupt-Flag gesetzt. Beim Timer/Counter1 heißt dieses Flag TOV1 (Timer Overflow 1). Ist zusätzlich das zugehörige Interrupt-Enable-Flag gesetzt,

wird das Hauptprogramm unterbrochen und die zugehörige Interrupt Service Routine (ISR) ausgeführt – in diesem Fall die Routine mit dem Vektor TIMER1_OVF_vect.

Die Overflow Frequenz f_{OVF} kann mit folgender Formel berechnet werden:

$f_{OVF} = \frac{f_{osc}}{N*2^{16}}$

> **Wichtig!**
>
> Wenn der höchstmögliche Zählerstand (255 bei 8-Bit-Timer/Countern, bzw. 65535 bei 16-Bit-Timer/Countern) erreicht ist, erzeugt der nächste Zähl-Impuls einen Overflow: Der Zählerstand beginnt wieder bei 0, und ein Interrupt-Flag wird gesetzt.

> **Beispiel:**
>
> Timer/Counter1 verwendet als Clock-Source die mit einem Prescaler-Wert von 8 geteilte CPU-Taktfrequenz (16MHz). In welchem Zeitabständen erfolgen die Overflows des Zählerstand-Registers?

Lösung:

Der Zählerstand wird wegen des Prescaler-Werts von 8 mit einer Frequenz von

$f_T = \frac{f_{osc}}{8} = 2MHz$

inkrementiert.

Die Periodendauer ist der Kehrwert daraus:

$T_T = \frac{1}{f_T} = \frac{1}{2MHz} = 0,5\mu s$

Die Zeitdauer zwischen zwei Overflows beträgt 6553**6** bzw. 2^{16} mal dieser Zeitspanne (Zählerstände 0 bis 6553**5** bzw. 0 bis $2^{16} - 1$).

Die Zeitdauer zwischen den Overflows T_{OVF} berechnet sich dann aus

$T_{OVF} = T_T * 2^{16} = 32,768ms$

Die Frequenz der Overflows

$f_{OVF} = \frac{1}{T_{OVF}} = \frac{1}{32,768ms} = 30,52Hz$

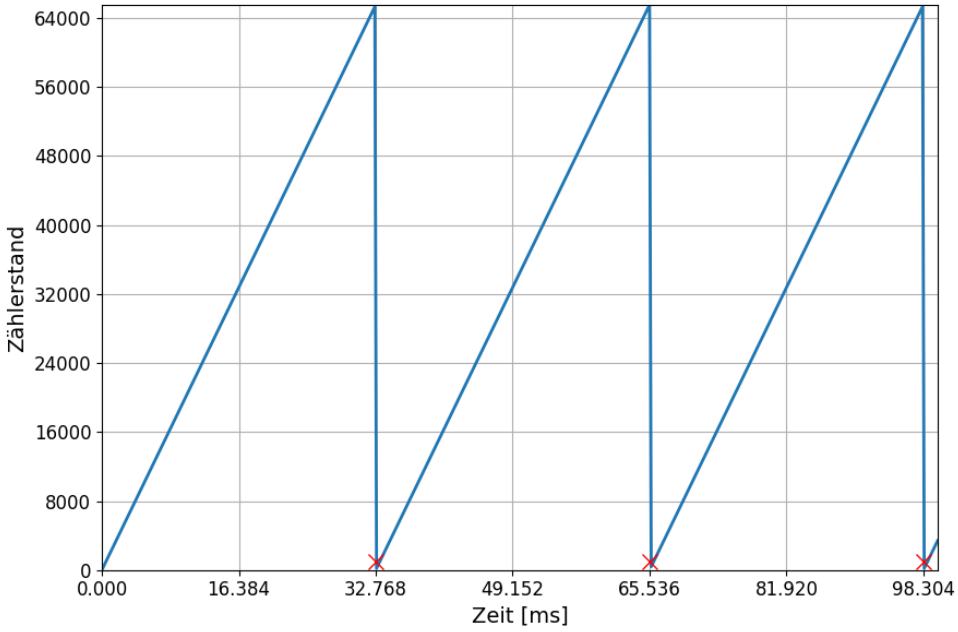

Abbildung 4: Alle 32,768 ms gibt es einen Timer overflow. Das Overflow Interrupt Flag wird gesetzt.

Die Compare Match Units

Pro Timer/Counter gibt es mehrere sogenannte Output-Compare-Register. Diese werden vom Programm beschrieben und vom der Timer Hardware verwendet.

Beim ATmega2560 existieren pro Timer/Counter drei Output-Compare-Register mit den Bezeichnungen OCRnA, OCRnB und OCRnC, wobei „n“ wiederum die Nummer des jeweiligen Timer/Counters ersetzt.

Das Output-Compare-Register A (OCRnA) nimmt unter den drei Output-Compare-Registern jedes Timer/Counters eine Sonderstellung ein. Diese wird jedoch erst im weiteren Verlauf bei der Beschreibung der Timer/Counter-Betriebsarten näher erläutert.

Der aktuelle Wert des Zählerstandsregisters wird kontinuierlich mit den drei Werten der Output-Compare-Register verglichen, wie in der folgenden Abbildung dargestellt. Immer dann, wenn der Wert des Zählerstandsregisters mit dem eines Output-Compare-Registers übereinstimmt, wird **beim darauffolgenden Zählimpuls** ein entsprechendes Interrupt-Flag gesetzt. Das heißt wenn OCRnA auf 10000 gestellt ist wird das Interruptflag erst bei einem Zählerstand von 10001 gesetzt. Ist zudem das zugehörige Interrupt-Enable-Flag aktiviert, so wird das Hauptprogramm unterbrochen und eine Interrupt-Service-Routine ausgeführt.

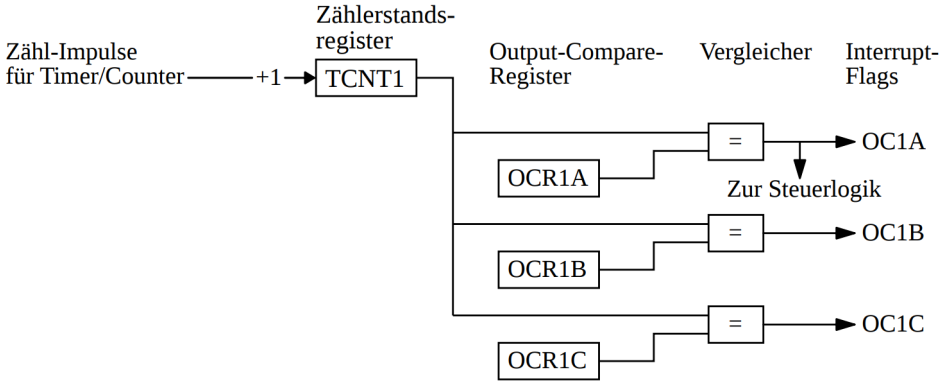

Abbildung 5: Die Compare Match Units beim ATMega 2560

Den Registern OCRnA, OCRnB und OCRnC kann direkt ein Wert zugewiesen werden.

> **Beispiel:**
>
> Jedes Mal, wenn der Zählerstand des Timer/Counter4 den Wert von 9999 erreicht, soll ein Interrupt ausgelöst werden.

Lösung:

Neben der Freigabe der Interrupts (diese wird erst weiter unten erklärt), ist folgende Zeile nötig. Es wird Output-Compare-Match-B-Register verwendet (allerdings könnte man genauso die anderen beiden Register verwenden):

```
OCR1B = 9998;
```

Der Wert muss auf 9998 = 9999 - 1 gesetzt werden. Da der Interrupt erst beim nächsten Zählimpuls ausgelöst wird.

Programmieren der Interrupts

Die zu einem Interrupt-Flag gehörende Interrupt-Service-Routine muss mit dem ISR-Makro unter Angabe des entsprechenden Interrupt-Vektors programmiert werden. Für den Overflow-Interrupt des Timer/Counters 1 verwendet man beispielsweise den Interrupt-Vektor TIMER1_OVF_vect. Die Interrupt-Vektoren für die Compare-Match-Interrupts des Timer/Counters 1 lauten:

`TIMER1_COMPA_vect` (für den Compare-Match zwischen `TCNT1` und `OCR1A`), `TIMER1_COMPB_vect`, und `TIMER1_COMPC_vect`.

Bei anderen Timer/Countern muss die „1“ im Vektornamen entsprechend durch die jeweilige Nummer des gewünschten Timers ersetzt werden.

Die folgende Tabelle zeigt die vier relevanten Interrupt-Flags, die zugehörigen Interrupt-Enable-Flags sowie die Interrupt-Vektoren für den Timer/Counter1. Für x ist A,B oder C einzusetzen. Auch hier gilt: Für andere Timer/Counters ist lediglich die „1“ im Namen entsprechend auszutauschen.

Interrupt-Ereignis:	**TCNT1-Overflow**	**Compare-Match x (`TCNT1 == OCR1x+1`)**
Interupt Flag	TOV1	OCF1x
Interupt Enable Flag	TOIE1	OCIE1x
Interrupt Vektor	TIMER1_OVF_vect	TIMER1_COMPx_vect

Die Interrupt-Flags selbst befinden sich im Special Function Register `TIFR1`, werden jedoch in der Praxis meist nicht benötigt. Sie können jedoch verwendet werden, um während der Initialisierungsphase versehentlich ausgelöste Overflow- oder Compare-Match-Interrupt-Ereignisse zu löschen, bevor die Interrupts global freigegeben werden.

Die Interrupt-Enable-Flags für den Timer/Counter1 befinden sich im Special Function Register `TIMSK1`. Diese Flags müssen auf 1 gesetzt werden, um den jeweiligen Interrupt zu aktivieren:

Bit	7	6	5	4	3	2	1	0
TIMSK1:	—	—	—	—	OCIE1C	OCIE1B	OCIE1A	TOIE1

> **Beispiel:**
>
> Bei Timer/Counter1 sollen sowohl ein Overflow als auch ein Compare-Match mit dem Register OCR1A bei einem Wert von 10000 einen Interrupt auslösen.

Lösung:

Ein Code-Gerüst für ein Programm mit diesen beiden Interrutps sieht so aus:

```
int main()
{
  TCCR1B |= (1 << CS12); // Prescaler = 1024
  TCCR1B |= (1 << CS10); // Prescaler = 1024

  OCR1A = 9999; // Vergleichswert.

  TIMSK1 |= 0x01 << OCIE1A; // Compare-Match-A Interrupt aktivieren
  TIMSK1 |= 0x01 << TOIE1;

  sei(); // Interrupts aktivieren
  while(1);
}

ISR(TIMER1_COMPA_vect)
{
    //Interrupt bei einem Zählerstand von 10000 (TCNT1 == OCR1A+1)
}

ISR(TIMER1_OVF_vect)
{
  //Interupt bei einem Overflow (TCNT1 == 0)
}
```

Timer Modi

Die 16-Bit-Timer/Counter arbeiten in einer von insgesamt 15 Betriebsarten (Timer/Counter-Modi). Es werden nur zwei davon behandelt: Normal-Mode und einer der beiden CTC-Modes.

Normal Mode

Im Normal-Mode wird das Zählstandsregister TCNTn bei jedem Zählimpuls um 1 erhöht. Wird der Höchstwert von 65535 erreicht, kommt es beim nächsten Zählimpuls zu einem Überlauf (Overflow). Der Zählerstand beginnt wieder bei null, bis es zum nächsten Überlauf kommt, und so weiter. Neben dem Overflow-Interrupt können während jedes Hochzählvorgangs von 0 bis 65535 mithilfe der drei Output-Compare-Register bis zu drei weitere Interrupts ausgelöst werden.

Diese Modus ist standardmäßig bei den 16bit Timern eingestellt.

Clear-Timer-on-Compare-Match – Mode (CTC-Mode)

Im CTC-Modus (Clear Timer on Compare Match) wird der Wert des Zählerstands nur so lange inkrementiert, bis er dem Wert des Output-Compare-Registers A entspricht (Compare Match). Sobald dies der Fall ist setzt die Steuerlogik des Timer/Counters den Zählerstand beim nächsten auftretenden Zählimpuls auf 0 zurück. Anschließend wird der Zählerstand erneut durch die Zählimpulse erhöht, bis es wieder zu einem Compare Match kommt – und so weiter.

Der größte Zählwert ist also nicht 65535 sondern der Wert der im `OCR1A` Register steht.

Die Frequenz der Interrupts kann durch Ändern des Werts im Output-Compare-Register A beeinflusst werden. Dieser Modus eignet sich daher besonders gut, um bestimmte Zykluszeiten für Interrupts präzise festzulegen oder um Rechtecksignale mit einstellbarer Frequenz zu erzeugen. Die Interrupt Frequenz berechnet sich nach folgender Formel:

$f_{OCR1A} = \frac{f_{osc}}{N*(1+OCR1A)}$

> **Beispiel:**
>
> Timer/Counter1 wird als Timer mit einem Prescaler-Wert von 64 im CTC-mode betrieben. Die CPU-Taktfrequenz ist 16MHz. In das OCR1A-Register wurde ein Wert von 6 geschrieben. Die Compare-Match-A-Interupts sind enabled, und in der zugehörigen Interrupt-Service-Routine wird der Zustand eines GPIO-Pins getoggelt. Welche Frequenz hat das Rechtecksignal, das dieser GPIO-Pin ausgibt?

Lösung:

Als erstes kann die Timer Frequenz berechnet werden.

$f_{T1} = \frac{f_{osc}}{N} = \frac{16MHz}{64} = 250kHz$

Daraufhin die Periodendauer.

$T_{T1} = \frac{1}{f_{T1}} = \frac{1}{250kHz} = 4\mu s$

Es werden immer wieder die Zählerstände von 0 bis 6 durchlaufen. Also insgesamt 7 Zählerstände.

$T_{OCR1A} = T_{T1} * (OCR1A + 1) = 4\mu s * (6 + 1) = 28\mu s$

Die Interrupt Frequenz beträgt dann.

$f_{OCR1A} = \frac{1}{T_{OCR1A}} = \frac{1}{28\mu s} = 35,714kHz$

Da das Signal bei jedem Interrupt getoggelt wird beträgt die Signalfrequenz die Hälfte der Interruptfequenz.

$f_{sig} = \frac{f_{OCR1A}}{2} = 17,86kHz$

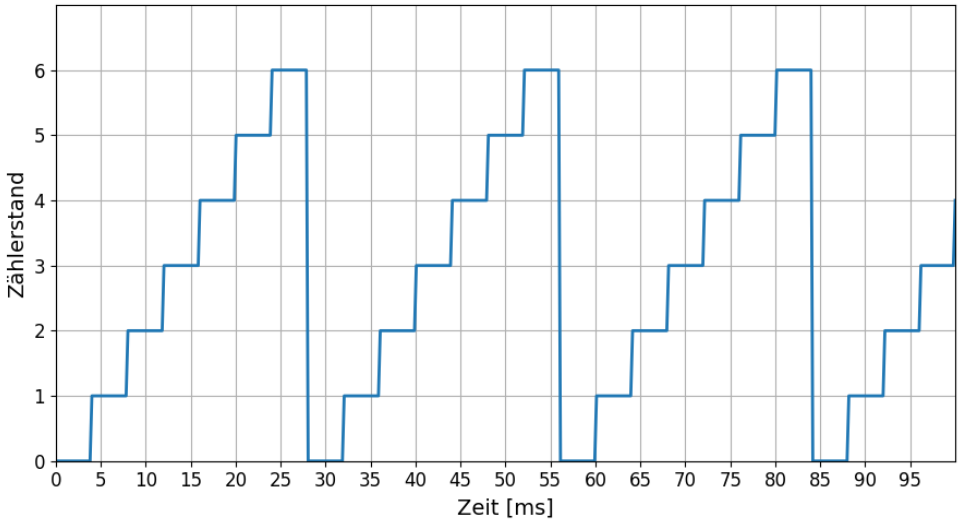

Abbildung 6: Zählerstand des Timer 1 im CTC Mode bei OCR1A = 6

> **Wichtig!**
>
> Im CTC-Mode wird der Zählerstand auf 0 zurückgesetzt, nachdem er gleich dem Wert im Output-Compare-Register A war („Clear Timer on Compare-Match“). Ergibt eine Berechnung, dass z.B. alle 20000 Zähl-Impulse ein Interrupt ausgelöst werden soll, so muss ein Wert von 19999 in das entsprechende Output-CompareRegister-A geschrieben werden. Bei nächsten Zählimpuls nach 19999 wird das Zählstandsregister auf 0 zurückgesetzt. Overflow-Interrupts treten nicht auf, da es zu keinem Overflow kommt. Es können aber alle drei Compare-Match-Interrupts verwendet werden

Programmieren der Timer Modi

Der gewünschte Timer/Counter-Mode (normal, oder CTC-Mode) wird in der Initialisierungsphase des Programms eingestellt, und bleibt meistens während der gesamten Programmlaufzeit unverändert. Zur Auswahl eines von insgesamt 15 Timer-Modi werden vier Bits mit den Namen `WGMn3`, `WGMn2`, `WGMn1` und `WGMn0` mit entsprechenden Werten beschrieben. Dabei ist „n“ wieder die Nummer des verwendeten Timer/Counters (also 1, 3, 4 oder 5). Die folgende Tabelle zeigt die zwei für dieses Skriptum relevanten Timer-Modi:

WGMn3	WGMn2	WGMn1	WGMn0	Timer Mode
0	0	0	0	Normal Mode
...	...	...	...	
0	1	0	0	CTC Mode

WGM bedeutet „Wave-Form-Generation“. Mit „Wave-Form“ ist damit der Zeitverlauf des TCNTn Register gemeint. Die vier eigentlich zusammengehörenden WGM-Bits jedes 16-BitTimer/Counters sind leider auf die beiden Register TCCRnA und TCCRnB verteilt. Für Timer/Counter 1 sehen diese beiden Register wie folgt aus:

Bit	7	6	5	4	3	2	1	0
TCCR1A:	—	—	—	—	—	—	WGM11	WGM10

Bit	7	6	5	4	3	2	1	0
TCCR1B:	—	—	—	WGM13	WGM12	CS12	CS11	CS10

> **Beispiel:**
>
> Timer/Counter1 soll im CTC-Mode betrieben werden. Dabei soll er als Timer mit einem Prescaler-Wert von 8 arbeiten

Lösung:

Da verschiedene Bits im selben Register gesetzt werden müssen sollten die Bits auf jeden Fall mit Bitschiebeoperatoren gesetzt werden. Für den CTC-Modus muss nur das Bit WGM12 gesetzt werden.

```
TCCR1B |= 0x01 << WGM12;
```

Für den Prescaler von 8 muss das Bit CS11 gesetzt werden.

```
TCCR1B |= 0x01 << CS11;
```

Timer Anwendungen

In diesem Kapitel werden typische Timer Anwendungen in einem Microcontrollerprogramm beschrieben.

Interrupts mit genauer Frequenz auslösen

Oftmals ist es notwendig eine Funktion mit einer genauen Frequenz zu wiederholen. Z.B:

- Auslesen eines Messwertes immer zum gleichen Zeitpunkt.
- Berechnen von Stellgrößen bei einem digitalen Regler.
- Erzeugen von Signalen mit konstanter Frequenz.
- ....

Dazu wird am besten der CTC Mode verwendet. Damit kann eine Interruptserviceroutine zeitlich sehr fein abgestuft aufgerufen werden. Die Interruptfequenz wird durch die Formel

$f_{OCR1A} = \frac{f_{osc}}{N*(1+OCR1A)}$

berechnet. In den meisten Fällen ist aber die Frequenz f_{OCR1A} bekannt und es soll daraus der Prescaler N und der OCR1A Wert bestimmt werden. Die umgestellte Formel sind so aus.

$OCR1A = \frac{f_{osc}}{N * f_{OCR1A}} - 1$

Da es immer noch 2 unbekannte Größen gibt muss eine davon konstant angeommen werden.

> **Beispiel:**
>
> Timer/Counter1 soll im CTC-Mode betrieben werden. Es soll dabei eine Interruptfrequenz f_{OCR1A} von 100 Hz einstellt werden.

Lösung

Für die 5 möglichen Prescaler ergeben sich folgende OCR1A Werte.

Prescaler	OCR1A berechnet	OCR1A möglich
1	159999	——
8	19999	19999
64	2499	2499
256	624	624
1024	156,25	156

Der OCR1A für der Prescaler von 1 kann nicht eingestellt werden. Da er über 65535 liegt. Mit den Prescalern 8,64 oder 256 kann der Wert von 100 Hz genau eingestellt werden. Mit einem Prescaler von 1024 kann der Wert nur gerundet eingestellt werden. Es sollten also als Prescaler entweder 8, 64 oder 256 verwendet werden.

Für manche Frequenzen kann der Wert aber mit keinem einzigen Prescaler genau eingestellt werden. Z.B. für eine Frequenz von 300 Hz

Prescaler	OCR1A berechnet	OCR1A möglich	f_{OCR1A} berechnet
1	53332,33	53332	300,0018 Hz
8	6665,66	6666	299,9850 Hz
64	832,33	832	300,1200 Hz
256	207,33	207	300,4807 Hz
1024	51,0833	51	300,4807 Hz

In der Tabelle für 300 Hz sieht man auch die berechnete Frequenz aus dem gerundeten OCR1A Wert. Für solche Fälle ist es am besten den kleinsten möglichen Prescaler zu verwenden, da auch dort der Fehler zur gewünschten Frequenz am kleinsten ist.

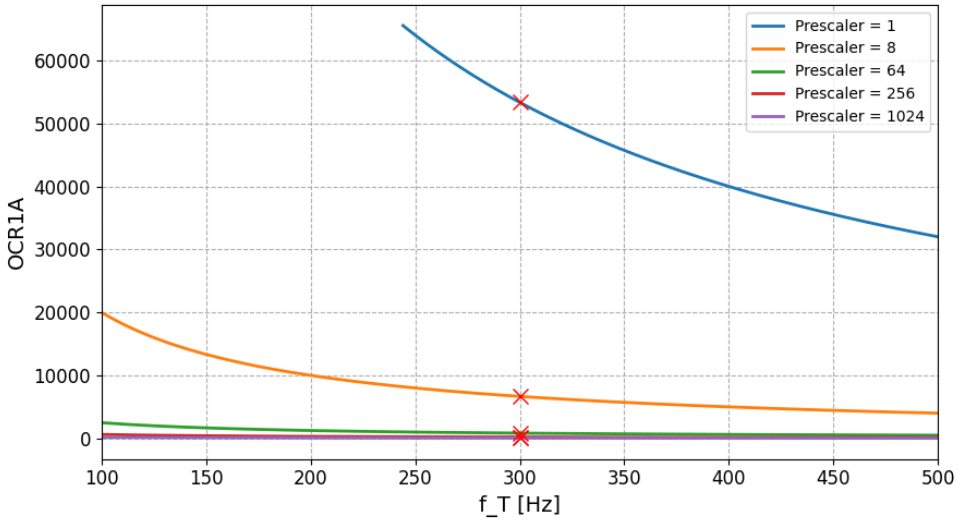

Abbildung 7: Die berechneten OCR1A Werte für alle möglichen Prescaler. Die Werte für 300 Hz sind hervorgehoben

Als Software sieht das Programm für eine Interrupt Frequenz von 300 Hz wie folgt aus. In der ISR wird eine LED getoggelt.

```
#define LED_PIN 0

int main()
{
  TCCR1B |= 0x01 << CS10; //Prescaler 1
  TCCR1B |= 0x01 << WGM12; //CTC Mode
  OCR1A = 53332; //Max Timer Value
  TIMSK1 |= 0x01 << OCIE1A; // Compare-Match-A Interrupt aktivieren
  sei(); // Interrupts global
  DDRA |= 0x01 << LED_PIN; //PA0 as Output
  while(1);
}

ISR(TIMER1_COMPA_vect)
{
  //Wird mit einer Frquenz von 300 Hz aufgerufen
  PORTA ^= 0x01 << LED_PIN;
}
```

Soft PWM

Soft PWM steht für Software Pulsweitenmodulation. Theoretisch stehen für PWM auch mehrere Timermodi zur Verfügung. In diesem Skriptum wird aber die Software PWM behandelt.

Mit einer Pulsweitenmodulation kann im weitesten Sinne ein analoger Ausgang nachgebildet werden. Dabei wird ein digitaler Ausgang mit einer fixen Frequenz ein- und ausgeschaltet. Das Verhältnis von Einschalt- und Periodendauer bestimmt dann den Ausgangswert.

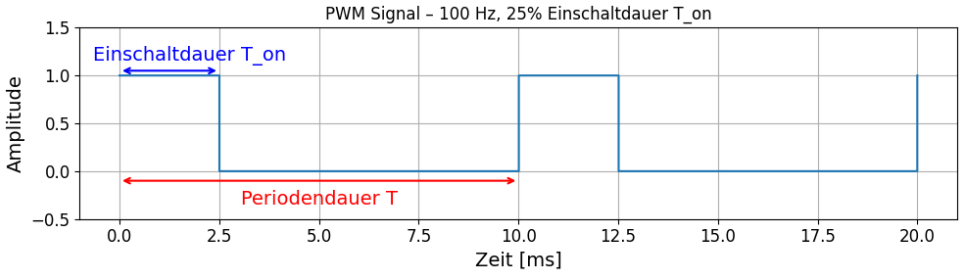

Abbildung 8: Pulsweitenmodulation

Für eine Software PWM kann sehr gut ein Timer im CTC Mode verwendet werden. In folgendem Beispiel soll eine Led soll mit einer PWM gedimmt werden. Dazu soll eine PWM mit einer Frequenz von 1kHz verwendet werden. Als ersten Schritt muss der Prescaler und der OCR1A Wert bestimmt werden.

$OCR1A = \frac{f_{osc}}{N * f_{OCR1A}} - 1$

Wir nehmen den kleinsten Prescaler von 1 an.

$OCR1A = \frac{16000kHz}{1 * 1kHz} - 1 = 15999$

Für die PWM sind dann zwei Interrupts notwendig. Einer zum Einschalten und einer zum Ausschalten der LED. Es werden der Output Compare A und Output Compare B Interrupt verwendet. Im OCA Interrupt wird die Led eingeschaltet und im OCB ausgeschaltet. Für 25% Einschaltdauer ergibt sich dann für OCR1B ein Wert von 4000;

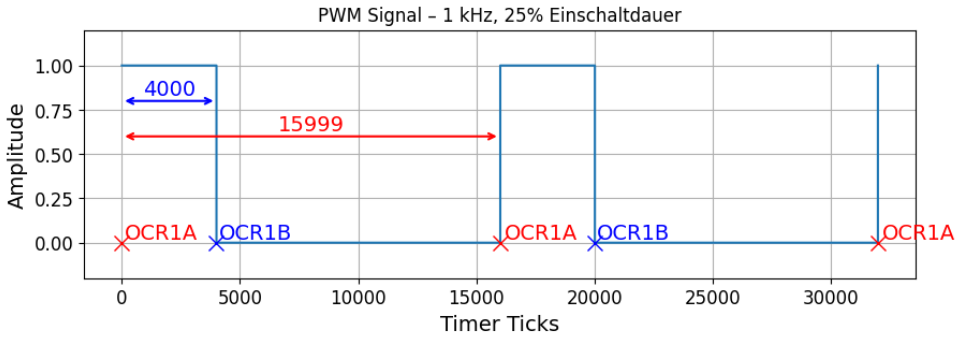

Abbildung 9: Pulsweitenmodulation mit Timer

In der Software sieht das ganz dann so aus.

```
#define LED_PIN 0

int main()
{
  TCCR1B |= 0x01 << CS10; //Prescaler 1
  TCCR1B |= 0x01 << WGM12; //CTC Mode
  OCR1A = 15999; //Max Timer Value -> 1kHz
  OCR1B = 4000; //25% von 15999
  TIMSK1 |= 0x01 << OCIE1A; // Compare-Match-A Interrupt aktivieren
  TIMSK1 |= 0x01 << OCIE1B; // Compare-Match-B Interrupt aktivieren
  sei(); // Interrupts global
```

```
  DDRA |= 0x01 << LED_PIN; //PA0 as Output
  while(1);
}

ISR(TIMER1_COMPA_vect)
{
  PORTA |= 0x01 << LED_PIN;
}

ISR(TIMER1_COMPB_vect)
{
  PORTA &= ~(0x01 << LED_PIN);
}
```