

## **Proseminar**

Timer/Counter und PWM beim ATmega16 Mikrocontroller

Marcel Jakobs

September 2006

# Inhaltsverzeichnis

<b>1 Was ist ein Timer/Counter</b>	<b>2</b>
1.1 Pulsweitenmodulation PWM	2
1.1.1 RC-Glied zur Glättung	3
1.2 Die Timer des ATmega16	3
1.2.1 Timer0	4
1.2.2 Timer1	4
1.2.3 Timer2	4
<b>2 Der Prescaler</b>	<b>5</b>
<b>3 Funktion des Timers</b>	<b>6</b>
3.1 Aufbau des Timers	6
3.2 Register	8
3.3 Interrupts	8
3.3.1 Output Compare Match Interrupt (OCI)	8
3.3.2 Timer Overflow Interrupt (TOC)	9
3.3.3 Timer Input Capture Interrupt (TICI)	9
3.4 Funktionen	9
3.4.1 Compare Output Mode	9
3.4.2 Input Capture Mode	9
3.4.3 Asynchronous Operation	10
3.5 Timer Modi	10
3.5.1 Normaler Modus	10
3.5.2 Clear Timer on Compare (CTC) Modus	10
3.5.3 Force Output Compare (FOC) Modus	10
3.5.4 Fast PWM Modus	11
3.5.5 Phase Correct PWM Modus	11
3.5.6 Phase and Frequency Correct PWM Modus	11
3.5.7 Waveform Generation Mode (WGM)	11
<b>4 Beispiele</b>	<b>12</b>
4.1 PWM/RC-Glied Berechnung	12
4.2 Elektromotor per PWM	12
4.2.1 Zum Schaltbild	13
4.2.2 Zum Programm	13
4.2.3 Spannungswerte bei verschiedenen Werten von OCR0	15
<b>5 Zusammenfassung</b>	<b>16</b>
5.1 Registerübersicht	16
5.2 Quellenangaben	17
5.2.1 Abbildungen	18

# Kapitel 1

## Was ist ein Timer/Counter

Ein Timer ist ein Counter (Abbildung1.1), der mit jedem Takt das Spezialregister TCNTx hochzählt. Er führt periodisch einen Interrupt aus, wenn das Register überläuft oder ein eingestellter Wert erreicht wird. Man kann als Takt auch eine externe Quelle benutzen. Dann sagt man Counter, da die steigenden bzw fallenden Taktflanken gezählt werden.

Anwendungsmöglichkeiten finden sich beim Zählen von Ereignissen, beim zeitabhängigen oder periodischen Ausführen von Programmteilen oder Messen von Zeitabständen.

### 1.1 Pulsweitenmodulation PWM

Pulsweitenmodulation (PWM) wird oft als Ersatz zu einem D/A-Wandler oder als Funktionsgenerator benutzt. Dabei erzeugt man eine Spannung, indem ein Ausgang schnell ein- und ausgeschaltet wird. Beim Dimmen von LEDs dient das menschliche Auge als Filter, ansonsten sollte man einen Filter durch ein RC-Glied hinter den Ausgang schalten. Der Mittelwert der Spannung liegt dann am Ausgang an. Berechnet wird der Mittelwert durch

$$U_m = U_{aus} + (U_{ein} - U_{aus}) \cdot \frac{t_{ein}}{t_{ein} + t_{aus}}$$

$U_{aus}$  ist dabei normalerweise 0V,  $U_{ein}$  die Betriebsspannung, z.B. 5V.

Die Leistung muss während der Ein- und Ausschaltzeit getrennt betrachtet werden nach der Formel:

$$P = \frac{U_{ein}^2}{R} \cdot \frac{t_{ein}}{t_{ein} + t_{aus}} + \frac{U_{aus}^2}{R} \cdot \frac{t_{aus}}{t_{ein} + t_{aus}}$$

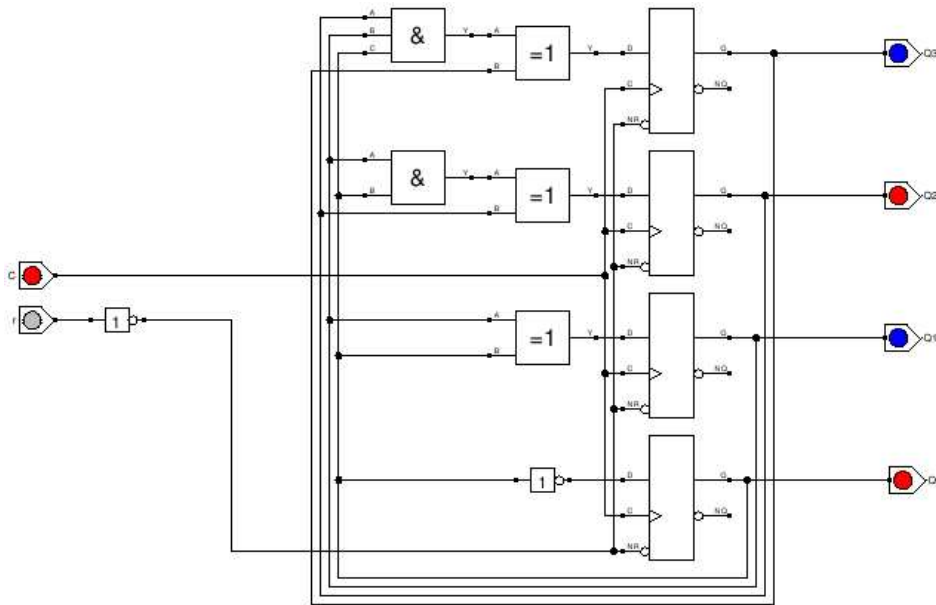


Abbildung 1.1: 4Bit Counter

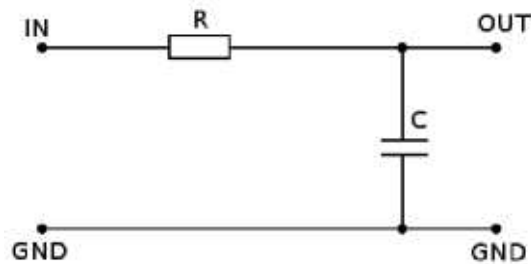


Abbildung 1.2: Schaltung RC-Glied

### 1.1.1 RC-Glied zur Glättung

Ein RC-Glied (Abbildung 1.2) filtert Frequenzen oberhalb der Grenzfrequenz heraus. Der Kondensator dieses sogenannten Tiefpass nimmt Strom bei einer Spannungsspitze auf, um ihn beim Spannungsabfall wieder abzugeben. Dadurch wird das Signal oberhalb der Grenzfrequenz geglättet.

Berechnet wird das RC-Glied mit der Formel

$$f_g = \frac{1}{2\pi \cdot R \cdot C}$$

## 1.2 Die Timer des ATmega16

Der ATmega 16 hat 3 Timer. Zwei 8 Bit Timer und einen 16 Bit Timer.

### 1.2.1 Timer0

Der Timer0 ist ein 8 Bit Timer. Er besitzt neben einem 10 Bit Prescaler, einen Frequenzgenerator und PWM. Zudem unterstützt er das Zählen von Taktflanken. Der CTC-Modus (Clear Timer on Compare Match) löscht bei Bedarf das Timerregister wenn es mit dem Vergleichsregister übereinstimmt. Er ist ein 8 Bit Timer ohne spezielle Funktionen und somit für simple Anwendungen gedacht.

### 1.2.2 Timer1

Der Timer1 ist ein 16 Bit Timer mit den gleichen Funktionen wie der Timer0. Darüber hinaus verfügt er über ein zusätzliches unabhängiges Vergleichsregister. Des weiteren beherrscht der Timer1 PWM mit variabler Periode und somit Pulsfrequenzmodulation. Er besitzt eine Input Capture Unit um externe Ereignisse zu zählen oder um auf sie zu reagieren.

### 1.2.3 Timer2

Der Timer2 ist ein 8 Bit Timer, der alle Funktionen des Timer0 besitzt, außer dem Zählen von Taktflanken.

Dafür verfügt er über die Möglichkeit einen 32Khz Uhrenquarz als Taktquelle für den Timer zu nutzen. Damit wird ein asynchroner Timer möglich, der unabhängig vom Takt ist. Dieser ist mindestens 4 mal langsamer und ermöglicht so auch das Messen weitaus längerer Zeitabstände.

Funktion	Timer0	Timer1	Timer2
Anzahl Bits	8	16	8
Zählen von Taktflanken	X	X	-
Prescaler	10 Bit	10 Bit	10Bit
CTC möglich	X	X	X
Frequenzgenerator	X	X	X
PWM	X	X	X
PWM mit variabler Periode	-	X	-
Zusätzliches unabhängiges Vergleichsregister	-	X	-
Input Capture Unit	-	X	-
Erlaubt 32Khz Uhrenquarz als Taktquelle	-	-	X

Timerfunktionen im Überblick

X Ja bzw. vorhanden

- Nein bzw. nicht vorhanden

CTC Funktion zum Löschen des Zählregisters, wenn es gleich dem Vergleichsregister ist

## Kapitel 2

# Der Prescaler

Der Prescaler (Vorteiler) teilt den Takt. Damit kann man den Timer dann z.B. nur jeden 2. oder 32. Takt zählen lassen. So können auch größere Zeitabstände (z.B. Die Sekunde einer Uhr) noch gezählt werden.

Die Bits CSx0, CSx1 und CSx2 im Register TCCRx steuern das Verhalten des Prescalers, wobei x durch den entsprechenden Timer ersetzt wird (0,1 oder 2)

Mit den CSxx Bits wird auch die Funktion für das Zählen von Taktflanken konfiguriert.

Beim Timer0 wer-

den die Taktflanken am Pin T0 (Pin PB0) registriert, beim Timer1 am Pin T1 (Pin PB1).

CSx2	CSx1	CSx0	Funktion Timer0/1	Funktion Timer 2
0	0	0	Keine Taktquelle (Timer hält)	gleich
0	0	1	Prozessortakt wird benutzt (kein prescaling)	gleich
0	1	0	Prozessortakt / 8	gleich
0	1	1	Prozessortakt / 64	Prozessortakt / 32
1	0	0	Prozessortakt / 256	Prozessortakt / 64
1	0	1	Prozessortakt / 1024	Prozessortakt / 128
1	1	0	Takt bei fallender Flanke	Prozessortakt / 256
1	1	1	Takt bei steigender Flanke	Prozessortakt / 1024

# Kapitel 3

## Funktion des Timers

### 3.1 Aufbau des Timers

Beschreibung des Blockschaltbilds (Abbildung 3.1)

Das TCCR<sub>x</sub>-Register konfiguriert die Steuereinheit.

Der Takt wird durch die Clock Select Einheit der Steuereinheit zugeführt. Dieser kann entweder vom Edge Detector (Flankenerkennung) oder vom Prescaler kommen. Von der Steuereinheit des Counters gehen drei Signalverbindungen zum Zählregister TCNT<sub>x</sub>. Diese sind count zum Zählen, clear zum Löschen des Registers und direction um die Richtung (herauf- oder herunterzählen) anzugeben.

Das Zählregister kann zwei Signale an die Steuereinheit zurückgeben: BOTTOM signalisiert die 0 und TOP den höchsten erreichbaren Wert.

Das Vergleichsregister und das Zählregister werden durch ein Äquivalenzglied auf Gleichheit geprüft, dessen Ausgang auf den Funktionsgenerator und von dort auf den Ausgabepin OC<sub>x</sub> geleitet wird. Gleichzeitig wird ein Output Compare Interrupt Request gesendet. (Bei eingeschaltetem Output Compare Match Interrupt wird dieser dann ausgelöst.) Des weiteren kann die Ausgabe des Äquivalenzgliedes auch als TOP-Signal genutzt werden, also das Erreichen des maximalen Wertes signalisieren. Ein Multiplexer entscheidet, ob eine Gleichheit von Zählregister und Vergleichsregister das TOP-Signal auslöst oder die feste Grenze des Zählregisters. Der Multiplexer wird über das CTC-Bit (Clear Timer on Compare) des TCCR-Registers gesteuert.

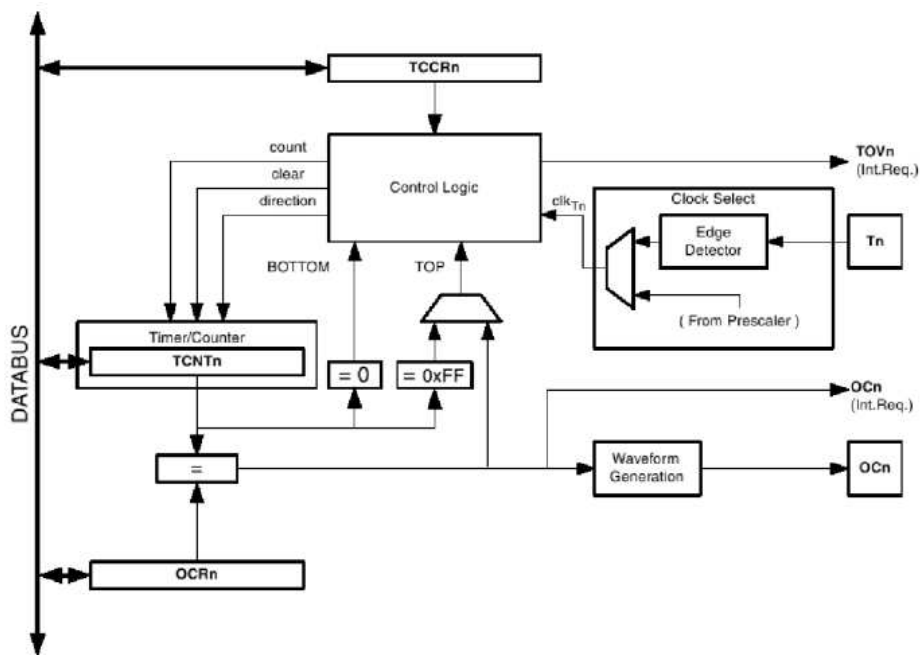


Abbildung 3.1: Blockschaltbild 8 Bit Timer0

count	erhöhe oder verringere TCNT0 um 1
direction	wählen zwischen Erhöhen und Verringern
clear	Löscht das Timer-Register TCNT0 (setzt alle Bits auf 0)
$clk_{tn}$	Timer/Counter Takt (hier $clk_{t0}$ )
TOP	Signalisiert, dass TCNT0 den maximalen Wert erreicht hat
BOTTOM	Signalisiert, dass TCNT0 den minimalen Wert erreicht hat (alle Bits 0)

## 3.2 Register

Die wichtigsten Register der Timer sind:

1. TCCR<sub>x</sub>[A | B] - Time/Counter Control Register (A oder B nur Timer1)  
Dient zur Auswahl des Betriebsmodus und zur Konfiguration des Prescalers.
2. TCNT<sub>x</sub> - Timer/Counter Register  
In diesem Register wird der aktuelle Zählwert gespeichert.
3. OCR<sub>x</sub>[A | B][H | L] - Output Compare Register (A oder B und H oder L nur Timer1)  
Wenn TCNT<sub>x</sub> den Wert dieses Registers erreicht, wird ein Output Compare Interrupt Request ausgelöst.
4. ICR1[H | L] - Input Capture Register 1 Hi oder Low (nur Timer1)  
Wird bei jedem Input Capture Event mit TCNT abgeglichen. Kann zur Festlegung des Maximalwertes genutzt werden.
5. TIMSK - Timer/Counter Interrupt Mask Register  
Zur Auswahl, welche Interrupts aktiviert werden sollen.
6. TIFR - Timer/Counter Interrupt Flag Register  
Hier werden bestimmte Bits gesetzt, wenn ein Interrupt ausgeführt werden soll.
7. ASSR - Asynchronous Status Register (nur Timer2)  
Register zur Steuerung des asynchronen Modus
8. SFIOR - Special Function IO Register  
PSR2 Bit setzt den Prescaler des Timer2 zurück

## 3.3 Interrupts

Die Timer des ATmega16 können verschiedene Interrupts auslösen um auf verschiedene Timerereignisse (z.B. wenn vorher bestimmter Wert erreicht wurde) zu reagieren.

Um die Interrupts zu aktivieren, muss das entsprechende Bit im TIMSK (Timer/Counter Interrupt Mask Register) gesetzt werden. Bei einem Interrupt Request wird das entsprechende Flag (Bit) im TIFR (Timer Interrupt Flag Register) gesetzt, welches dann den Interrupt auslöst. Man kann also auch durch setzen des entsprechenden Bits im TIFR einen Interrupt von Hand auslösen.

### 3.3.1 Output Compare Match Interrupt (OCI)

Erreicht das Register TCNT<sub>x</sub> den Wert, der im Register OCR<sub>x</sub> voreingestellt wurde, so tritt ein Timer Compare Match ein. Dabei wird ein Output Compare Interrupt ausgelöst. Auf Wunsch wird der Counter dabei zurückgesetzt. Dazu dient das CTC-Bit (Clear Timer on Compare) im TCCR<sub>x</sub>-Register.

Das OCR<sub>x</sub> Register kann jederzeit verändert werden, dadurch kann die Anzahl der Takte zwischen zwei OCI bei Bedarf angepasst werden.

### **3.3.2 Timer Overflow Interrupt (TOC)**

Ist das Register TCNTx voll, so wird ein Timer/Counter Overflow und damit ein Timer/Counter Overflow Interrupt ausgelöst. Der Timer zählt dabei von 0 aufwärts weiter.

So kann periodisch eine Interruptroutine ausgeführt werden.

Bei PWM kann bei Bedarf der Timer Overflow Interrupt aktiviert werden um z.B. das OCRx Register zu ändern, wodurch man einen sinusförmigen Ausgang generieren kann.

### **3.3.3 Timer Input Capture Interrupt (TICI)**

Der TICI wird bei einem Input Capture Event ausgelöst, welches eintritt, wenn am ICP1-Pin eine steigende/fallende Flanke (je nach Einstellung) gemessen wird. Es besteht die Möglichkeit einen Noise Canceler einzuschalten, der dafür sorgt, dass der TICI erst ausgelöst wird, wenn das Signal 4 Takte lang stabil anliegt. Dieser Modus kann genutzt werden um Ereignisse zu zählen oder auf sie zu reagieren.

## **3.4 Funktionen**

Die Timer haben verschiedene Spezialfunktionen, die es ermöglichen den Timer nicht nur mikrocontrollerintern zu nutzen, sondern ihn mit Hilfe von Pins auch mit externen Komponenten einer Schaltung zu verbinden.

### **3.4.1 Compare Output Mode**

Der Compare Output Mode ist in alle Timer integriert. Die Bits COMx0 und COMx1 steuern das Verhalten des OCx-Pins (der Ausgabepin des Compare Output Modes). Wenn beide Bits auf 0 gesetzt sind, behält der OCx-Pin seine Standardfunktion im Mikrocontroller. Ist das COMx1 Register gesetzt wird der OCx-Pin bei einem Compare Match auf 0 gesetzt (bei PWM wird er bei Erreichen vom maximalen Wert wieder auf 1 gesetzt). Sind beide Register gesetzt ist diese Funktion invertiert (also bei jedem Compare Match wird OCx auf 1 gesetzt).

Ist nur das COMx0 Bit gesetzt, so wird der Zustand des OCx-Pins bei jedem Compare Match invertiert. (Nur für nicht-PWM Modi)

### **3.4.2 Input Capture Mode**

Die Input Capture Unit ist nur im Timer1 enthalten. Eine steigende bzw fallende Flanke löst ein Input Capture Event aus, bei dem der Wert des TCNT1 Registers in das ICR1 Register geschrieben wird. Im selben Takt wird das entsprechende Bit im TIFR geschrieben. Bei eingeschaltetem Noise Canceler wird der Input Capture erst ausgelöst, wenn das Signal 4 Takte lang stabil anliegt. Input Capture ist auch mit dem Analogcomparator statt dem ICP1 (Input Capture Pin) möglich.

### 3.4.3 Asynchronous Operation

Nur der Timer2 besitzt eine Möglichkeit um asynchrone Timerfunktionen zu benutzen. Dieser Modus dient zur Verwendung eines Uhrenquarzes an den TOSC1-Pin als Takt für den Timer. Dafür muss der CPU-Takt mindestens 4-mal höher sein als der Quarz. Der Asynchrone Modus dient so dem Umgang mit längeren Zeitabständen. Zum Wechseln vom normalen in den Asynchronen Modus und zurück müssen alle Interrupts des Timers deaktiviert sein und nach dem Wechsel müssen alle Register neu gesetzt werden.

## 3.5 Timer Modi

Folgende Modi stehen mit den verschiedenen Timern zur Verfügung:

### 3.5.1 Normaler Modus

Der normale Modus ist in alle Timer/Counter integriert. In diesem Modus wird ganz normal hochgezählt. Das TCNTx Register wird nicht gelöscht, bei einem Überlauf startet das Register von 0. Bei aktiviertem Timer/Counter Overflow Interrupt wird dieser ausgelöst. Man kann den Wert, bei dem ein Interrupt ausgelöst werden soll, auch durch das Vergleichsregister OCRx einstellen. Dafür aktiviert man den Compare Match Interrupt, der ausgelöst wird, sobald  $TCNTx = OCRx$  ist. In diesem Modus kann man einen Timer Overflow Interrupt periodisch auslösen. Zusätzlich kann bei einem bestimmten Wert des Zählregisters ein Output Compare Match Interrupt ausgelöst werden.

### 3.5.2 Clear Timer on Compare (CTC) Modus

Alle Timer/Counter des ATmega16 besitzen diesen Modus. Stimmt das TCNTx Register mit dem Vergleichsregister OCRx überein, wird das TCNTx Register auf 0 gesetzt und bei aktiviertem Timer/Counter Overflow Interrupt wird dieser ausgelöst. Zum aktivieren des CTC-Modus müssen die WGM-Bits (Waveform Generation Mode) im TCCRx Register entsprechend gesetzt sein. Dieser Modus ermöglicht es periodisch einen Timer Overflow Interrupt auszulösen, wobei die Zeit zwischen 2 Interrupts verändert werden kann

### 3.5.3 Force Output Compare (FOC) Modus

Der Force Output Compare Modus ist in alle Timer/Counter integriert. Der FOC Modus kann nur bei nicht-PWM Modi genutzt werden. Beim Schreiben einer logischen 1 auf FOCx wird ein sofortiger Compare Match des Funktionsgenerators erzwungen. Der OCx Ausgang wird auf den Status des COMx1 Bits gesetzt. Deshalb bestimmt der Wert aus COMx1 den Wert des OCx Pins beim Vergleich.

### **3.5.4 Fast PWM Modus**

Alle Timer/Counter sind mit dem Fast PWM Mode ausgestattet. Der Counter zählt immer vom minimalen aufwärts zum maximalen Wert und beginnt wieder beim minimalen Wert. Dies ermöglicht einen schnellen Funktionsgenerator. Der Ausgang wird beim Starten auf 1 gesetzt und beim maximalen Wert auf 0.

### **3.5.5 Phase Correct PWM Modus**

Alle Timer/Counter beherrschen diesen Modus. Der Phasenkorrekte PWM Modus ermöglicht genaue Funktionsgeneratoren. Hier wird erst hoch gezählt und danach wieder herunter, wobei der Ausgang OC beim Hochzählen auf 0 und beim Herunterzählen auf 1 gesetzt ist. Da das Vergleichsregister OCR1A (bzw. ICR1) den maximalen Zählwert bestimmt, kann so Pulsfrequenzmodulation implementiert werden.

### **3.5.6 Phase and Frequency Correct PWM Modus**

Dieser Modus ist nur im Timer1 integriert. Er ist genau wie der Phase Correct PWM Mode, allerdings wird das OCRx Register öfter updated.

### **3.5.7 Waveform Generation Mode (WGM)**

Im TCCRx Register gibt es einige Bits, die WGMxx heißen. Damit kann man die PWM-Modi und den CTC-Modus steuern. Für die PWM Modi gilt: Die Funktionen können am Pin OCx ausgegeben werden. Dafür muss das entsprechende Bit COMxx gesetzt sein. Der Pin muss als Output konfiguriert sein (PullUps).

# Kapitel 4

## Beispiele

### 4.1 PWM/RC-Glied Berechnung

Berechnung 3V per Fast PWM ohne Prescaler mit Timer0 (8 Bit)

$$\begin{aligned}U_m &= U_{aus} + (U_{ein} - U_{aus}) \cdot \frac{t_{ein}}{t_{ein} + t_{aus}} \\ \Leftrightarrow 3V &= 0V + (5V - 0V) \cdot \frac{t_{ein}}{256} \\ \Leftrightarrow \frac{3V}{5V} \cdot 256 &= t_{ein} \\ \Rightarrow t_{ein} &= 153.6 \text{ also ca } 153 \text{ von } 256 \text{ Takten}\end{aligned}$$

RC-Glied Berechnung für 8Bit PWM ohne Prescaler

$$\begin{aligned}f_g &= \frac{1}{2\pi \cdot R \cdot C} \\ f_g &= \frac{1000000}{256} = 3906,25\text{Hz} \\ \text{Widerstand } &100\Omega \\ \Leftrightarrow 3906,25 \cdot \pi \cdot 100 &= \frac{1}{C} \\ \Leftrightarrow 2453906 &= \frac{1}{C} \\ \Leftrightarrow C &= 4,075 \cdot 10^{-7} = 0,4\mu F\end{aligned}$$

### 4.2 Elektromotor per PWM

Vergleiche Abbildung 4.1 (SG1 ist der Elektromotor). Als praktisches Beispiel habe ich eine Motoransteuerung mit PWM erstellt.

Per FastPWM gibt der 8Bit Timer0 die Betriebsspannung des Motors aus, die durch ein RC-Glied geglättet wird. (Berechnung siehe oben). Der Prescaler wurde deaktiviert, so dass der normale CPU-Takt genutzt wurde. Der Status des Vergleichsregisters OCR0 wird auf zwei 7-Segmentanzeigen als HEX-Wert ausgegeben und kann durch zwei Taster incrementiert bzw. decrementiert werden.

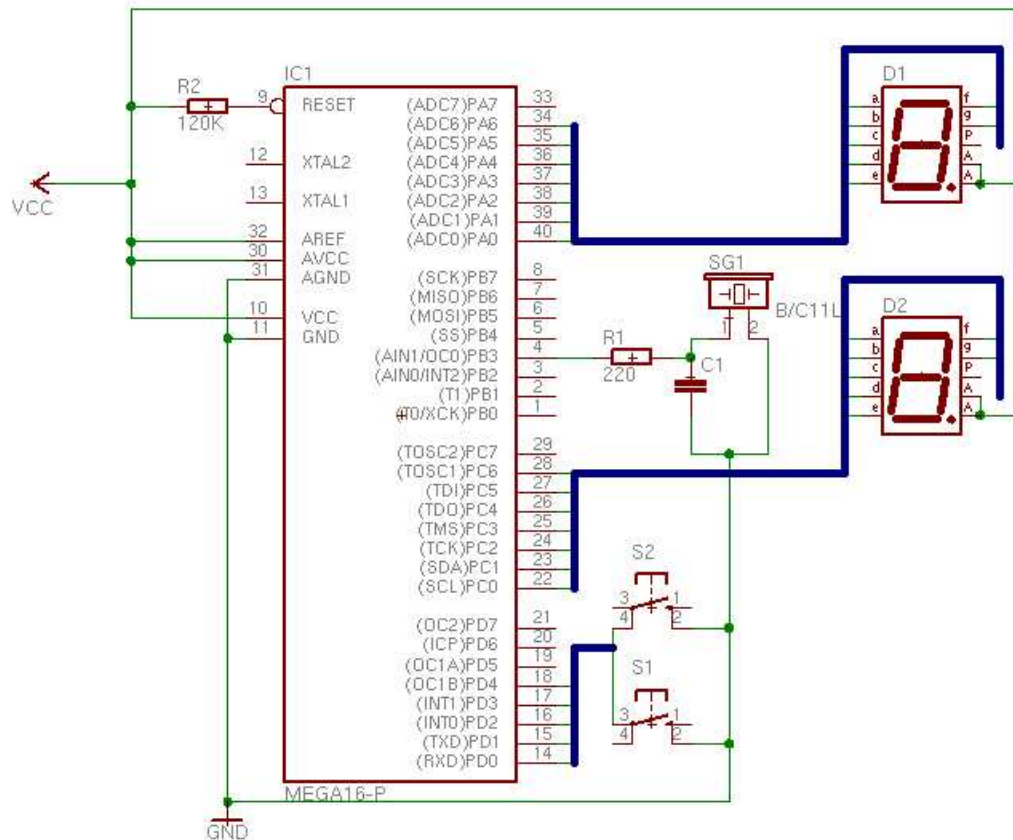


Abbildung 4.1: Schaltung

#### 4.2.1 Zum Schaltbild

(Abbildung 4.1) Die beiden 7-Segmentanzeigen hängen an den Ports A und C. Die Taster sind an Port D angeschlossen und werden gegen Masse geschlossen. Der OC0 Pin ist der PIN3 des Ports B, dort ist das RC-Glied und darauf folgend der Elektromotor angeschlossen. Der Resetpin ist mit einem 120K Widerstand mit VCC (5 Volt) verbunden.

#### 4.2.2 Zum Programm

Vergleiche Abbildung 4.2. Ein Chararray (X) wird definiert, welches die dem Index entsprechenden Werte enthält, um die 7-Segmentanzeigen anzusteuern. Eine Wartefunktion soll die Taster entprellen, damit ein einzelner Tastendruck erkannt werden kann. Ihr wird ein Wert übergeben, der angibt, wie oft bis 100 gezählt wird bevor zurück zum Hauptprogramm gesprungen wird. Im Hauptprogramm werden zuerst die Ports definiert:

- Port A und C als Ausgabe für die 7-Segmentanzeigen
- Port B Pin 3 (OC0) wird als Ausgang definiert (zur PWM-Ausgabe)

```

#include <avr/io.h>

// Array für 7-Segment Anzeige
unsigned char X[16]={1,0x4f,0x12,6,0x4c,0x24,0x20,0xf,0,4,8,0x60,0x31,0x42,0x30,0x38};

// wait wartet ein paar Takte, damit ein einzelner Tastendruck moeglich ist (polling)
wait(unsigned int i){
    unsigned int j,k;
    for(k=i;k>0;k--){
        {
            j=100;
            while(j>0){ j--; }
        }
    }
}

int main(void)
{
    DDRA=255; // Anzahl der ausgeschalteten Takte auf 7-Segment
    DDRC=255; // Anzahl der eingeschalteten Takte auf 7-Segment
    DDRB=8; // PWM Ausgang
    PORTD=255; // PORTD ist Eingang fuer die Taster
    TCCR0=(1<<CS00)|(1<<WGM00)|(1<<WGM01)|(1<<COM01); // Timer0 ohne Prescaler als FastPWM
    OCR0=0; // Vergleichswert für pwm

    while(1) // Endlosschleife
    {
        //Ausgabe auf 7-Segment
        PORTC=X[OCR0/16];
        PORTA=X[OCR0%16];

        // INC und DEC durch Taster
        if(bit_is_set(PIND,0)==0){
            OCR0++;
            wait(30);
        }
        if(bit_is_set(PIND,1)==0){
            OCR0--;
            wait(30);
        }
    }
}

```

Abbildung 4.2: Quelltext

- Port D wird als Eingabe definiert (Taster)

Im TCCR0 Register wird das CS00 Bit gesetzt um den Clock Selector auf CPU-Takt zu stellen.

Die WGM00 und WGM01 Bits werden gesetzt um FastPWM einzuschalten.

Das COM01 Bit ebenfalls um das PWM-Signal am Ausgabepin OC0 auszugeben (Compare Output Mode).

Das Vergleichsregister OCR0 wird mit 0 initialisiert.

Nun folgt die Endlosschleife, um die Taster abzufragen und die 7-Segmentanzeigen zu aktualisieren:

Port C wird (mittels des Arrays X) auf den Wert OCR0/16, also die höherwertige Stelle gesetzt.

Port A wird durch OCR0%16 (mittels der Arrays) auf die niederwertige Stelle gesetzt.

Dann werden mit Hilfe der Funktion bit\_is\_set(PINx,y) (x ist der entsprechende Port und y die PIN-Nummer) beide Taster abgefragt. Ist einer gedrückt, so wird das OCR0 Register incrementiert/decrementiert und danach die wait-Funktion mit dem Parameter 30 aufgerufen.

Dadurch wird bei jedem Tastendruck ca. 6000 Takte gewartet.

### 4.2.3 Spannungswerte bei verschiedenen Werten von OCR0

Bei  $VCC=5V$  und 1MHz Oszillator kann man folgende Spannungen am Ausgang berechnen und messen:

Wert OCR0	Spannung (V)
0	0
10	0,19
50	0,98
75	1,47
100	1,96
128	2,5
150	2,94
175	3,43
200	3,92
225	4,41
255	5

Dies gilt für die Einstellungen, wie im Beispiel 4.2 beschrieben. Berechnen lassen sich diese Werte durch die in 1.1 beschriebene Formel. Dabei ist  $t_{ein} = OCR0$ ,  $t_{ein} + t_{aus} = 255$ ,  $U_{aus} = 0V$  und  $U_{ein} = VCC$  (also 5V). Da  $0 + (5 - 0) = 5$  kann man die Formel vereinfachen zu:

$$U_m = 5 \cdot \frac{OCR0}{255}$$

# Kapitel 5

## Zusammenfassung

### 5.1 Registerübersicht

TCCR<sub>x</sub> (Timer/Counter Control Register):

Beschreibung:	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Name Timer0:	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Name Timer1A:	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
Name Timer1B:	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
Name Timer2:	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20

CS<sub>xx</sub> (Clock Select): konfigurieren den Prescaler (siehe Tabelle Prescaler)

WGM<sub>xx</sub> (Waveform Generation Mode): die Register WGM<sub>xx</sub> steuern die verschiedenen PWM-Modi

COM<sub>xx</sub> (Compare Match Output Mode): diese Register steuern den Output Compare Pin OC<sub>x</sub>. Das DDR Register muss entsprechend gesetzt sein.

FOC<sub>x</sub> (Force Output Compare): Beim Schreiben einer logischen 1 auf FOC<sub>x</sub> wird ein sofortiger compare match des Funktionsgenerators erzwungen.

ICES1 (Input Capture Edge Select): Bestimmt, welche Flanke ein Capture event auslöst. Bei 0 löst eine fallende Flanke ein Capture event aus, bei 1 eine steigende.

ICNC1 (Input Capture Noise Canceler): Filtert den Eingang des ICP1 Pins. Die Filterfunktion benötigt 4 Takte lang ein stabil anliegendes Signal am ICP1 Pin um den Wert anzuerkennen.

TIMSK (Timer/Counter Interrupt Mask Register):

Beschreibung:	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Name:	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

TOIE<sub>x</sub> (Timer Overflow Interrupt Enable): Aktiviert den Timer Overflow Interrupt, der ausgelöst wird, wenn das TCNT<sub>x</sub> Register überläuft

OCIE<sub>x</sub> (Output Compare Interrupt Enable): Aktiviert den Output Compare Interrupt, der ausgelöst wird, wenn das TCNT<sub>x</sub> Register mit den Wert des OCR<sub>x</sub> Registers übereinstimmt

TICIE1: (Timer Input Capture Interrupt Enable): Aktiviert den TIC1, der ausgelöst

wird, wenn eine Taktflanke erkannt wird

TIFSK (Timer/Counter Interrupt Mask Register):

Beschreibung:	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Name:	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

TOV<sub>x</sub> (Timer Overflow Flag): Wenn dieses Flag gesetzt wird, wird ein TOI ausgelöst

OCF<sub>x</sub> (Output Compare Match Flag): Wenn dieses Flag gesetzt wird, wird ein OCI ausgelöst

ICF<sub>1x</sub> (Timer Input Capture Flag): Wenn dieses Flag gesetzt wird, wird ein TICI ausgelöst

ASSR (Asynchronous Status Register):

Beschreibung:	Bit3	Bit2	Bit1	Bit0
Name:	AS2	TCN2UB	OCR2UB	TCR2UB

AS2 (Asynchronous Timer/Counter 2): Wenn dieses Flag gesetzt wird, bezieht der Timer2 seinen Takt von einem externen Oszillator

TCN2UB (Timer/Counter2 Update Busy): ist gesetzt, während TCNT2 beschrieben wird

OCR2UB (Output Compare Register2 Update Busy): ist gesetzt während OCR2 beschrieben wird

TCR2UB (Timer/Counter Control Register2 Update Busy) ist gesetzt, während TCCR2 beschrieben wird

## 5.2 Quellenangaben

- Informationen zur Berechnung von PWM habe ich von <http://www.mikrocontroller.net/articles/PWM>
- Sonstige Informationen über PWM habe ich von [http://www.mikrocontroller.net/articles/AVR\\_PWM](http://www.mikrocontroller.net/articles/AVR_PWM) und aus dem Datenblatt (siehe unten)
- Informationen zum RC-Glied und dessen Berechnung habe ich von <http://www.mikrocontroller.net/articles/Filter>
- Für die Definition eines Timers habe ich diese als Grundlage genommen: <http://www.mikrocontroller.net/articles/Timer>
- Die Definition der PFM habe ich von <http://de.wikipedia.org/wiki/Pulsfrequenzmodulation>
- Alle Weiteren Informationen (Timer des ATmega16, Interrupts, Funktionen, Modi, Register und Prescaler) habe ich aus dem Datenblatt des ATmega16 unter [http://www.atmel.com/dyn/resources/prod\\_documents/doc2466.pdf](http://www.atmel.com/dyn/resources/prod_documents/doc2466.pdf)

### 5.2.1 Abbildungen

- Abbildung 1.1 (Counter) ist aus dem Script "Technische Informatik C" von Dipl-Inform. Dr. Merten Joost
- Abbildung 1.2 (RC-Glied) habe ich mit Gimp erstellt
- Abbildung 3.1 (Timer0) ist aus dem Datenblatt des Atmega16
- Abbildung 4.1 (Schaltbild) habe ich mit Eagle erstellt
- Abbildung 4.1 (Quelltext) habe ich aus Gedit (Texteditor) per Screenshot kopiert.