

## Beispiel ohne Interrupt

Blinkt mit 0,5 Hz bei 1 MHz Taktfrequenz

```

int main(void) {
    DDRB = 0xFF;
    PORTB = 0xFF;
    TCCR0 |= (1<<CS00)|(1<<CS02);
    int count = 0;
    while (1) {
        if (TIFR & (1<<TOV0)) {
            TIFR |= (1<<TOV0);
            count++;
            if (count == 4) {
                count = 0;
                PORTB |= 1; // EXOR
            }
        }
    }
}

```

x	y	z
0	0	
0	1	
1	0	
1	1	

PORTB 11111111

EXOR 00000001

=> 11111110

EXOR 00000001

11111111

In einer Schleife wird TOV0 auf den Zustand 1 abgelegt  
Danach muss dieser Flag wieder gelöscht werden (durch schreiben einer 0)

## Beispiel mit Interrupt

Um nicht ständig das TOVO Flag abfragen zu müssen, empfiehlt sich der Einsatz des Queueflow Interrupts.

ISR (Timer 0 - OVF\_vect) {

static int count = 0; // static: Wert bleibt erhalten

count++;

if (count >= 4) {

count = 0;

PORTB ^ = 1;

}

}

int main(void) {

DDRB = 0xFF; // PORTB als Ausgang

PORTB = 0xFF; // alle LEDs aus

TCCR0 |= (1<<CS00) | (1<<CS02); // Prescaler auf 1024

TIMSK |= (1<<TOIE0) // Enable Interrupt für Timer 0

sei(); // General Interrupt enable overflow

while(1);

}

Beispiel CTC

Jede 1ms soll Interrupt ausgelöst werden

```
int main (void) {
```

    TCCR0 |= (1<< WGM01) | (1<< CS01); // CTC Modus + Prescaler 8

    OCR0 = 125 - 1; // 1ms / 1μs \* 8

// Compare Interrupt erlauben

    TIMSK = (1<< OCIE0);

    sei(); // Globale Interrupts erlauben

    while(1); {

        ISR(Timer0\_COMPA\_vect);

    }

}

Der Compare Interrupt - Handler wird aufgerufen, wenn  $TCCR0 = OCR0 =$

= 125 - 1

(125 Schritte)

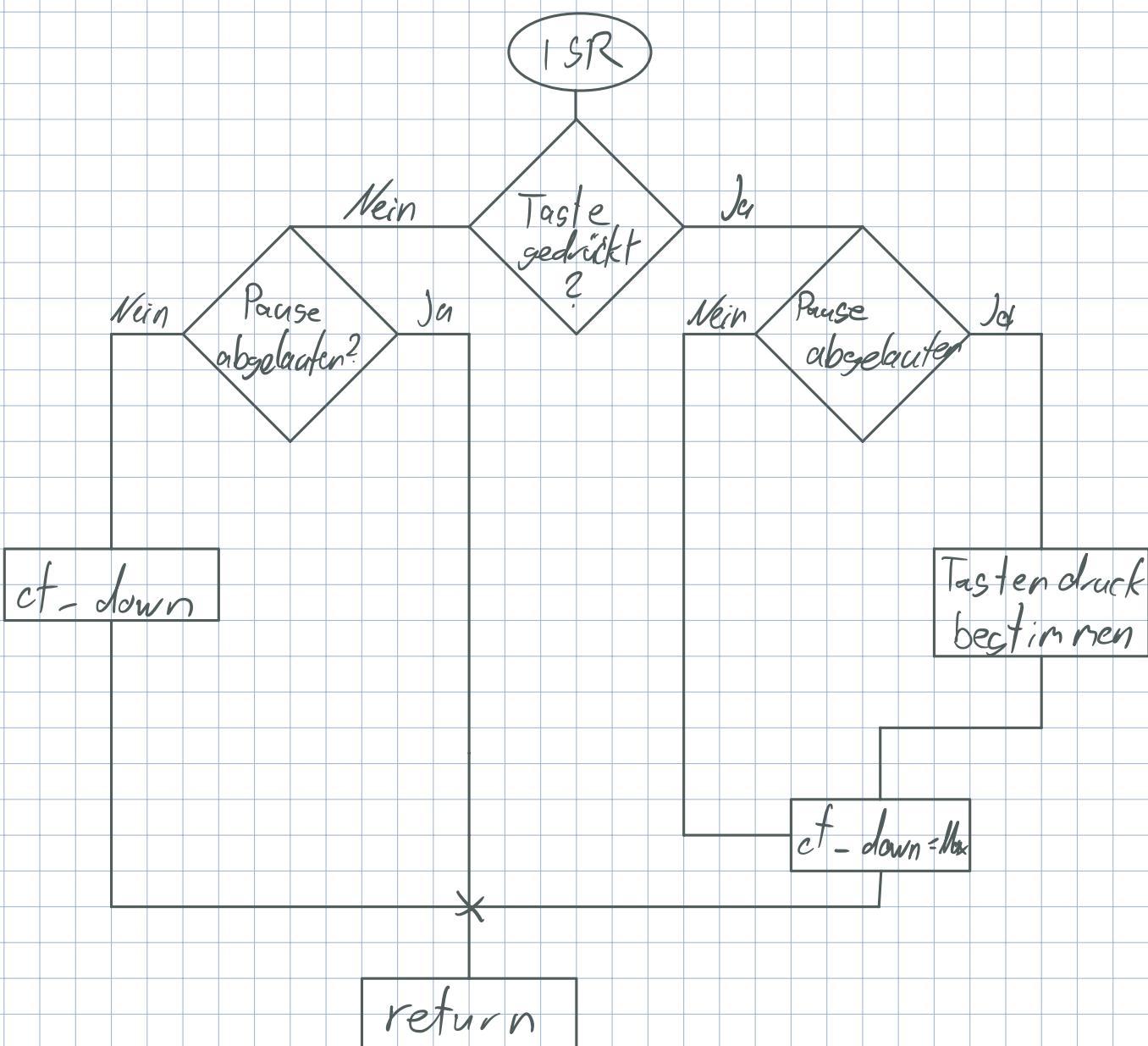
⇒ genau alle 1ms

# Tastenentprellung mit Timer (TC)

Hauptprogramm:

- Timer konfigurieren
- Warten auf validen Tastendruck  
↳ Ausgang leggen

ISR (Interrupt Service Routine) alle Sons aufrufen



static volatile byte key = 0xFF; // Globale Variable (static: nur in diesem File sichtbare)  
// volatile: Compiler optimiert nichts weg

int main(void) {

DDR B = 0xFF;

TCCR0 = (1<< WGM01) | (1<< CS02); // CTC Mode setzen + Prescaler

OCR0 = 20 - 1;

TIMSK |= (1<< OIE0),  
sei(1);

auf 256

// eine Taktflanke 1μs → alle 256μs ein Timerstep

⇒ 20 Steps für

while(1) {

PORTE B = (1<< getKey());

}

ISR(TIMER0\_COMR\_vect) {

// alle Spins aufrufen

test\_keys();

}

void test\_keys(void) {

byte mask = 1;

static byte ct..down = 10;

// Wert wird nur beim ersten mal initialisiert

if(PIND != 0xFF)

// Wert bleibt beim nächsten Aufruf erhalten

if(ct..down == 0) {

key = 0;

while((PIND & mask) != 0) {

mask << 1;

key +=;

} ct..down = 10;

else if(ct..down != 0)

ct..down --;

```

byte gefkey(void) {
    byte k;
    while (key == 0x FF) { }
    clr(); // Interrupt disable, damit key während der Speicherung
    k = key; // nicht geändert wird
    key = 0xFF;
    sei();
    return k;
}

```

## PWM (Pulse Width Modulation)

Viele elektronische Verbraucher können dadurch in ihrer Leistung reguliert werden, dass die Versorgungsspannung verändert wird.

Ein Gleichstrommotor wird zum Beispiel langsamer laufen wenn er mit geringerer Spannung versorgt wird.

Diese Art der Ansteuerung ist digitalen Systemen aber schwierig.

(Ein Controller kann nicht keine analogen Spannungen ausgeben)

Eine einfache Art die zugeführte Energie zu verringern, ist die volle Versorgungsspannung periodisch schnell ein- und auszuschalten. Man muss nur dafür sorgen, dass die einzelnen Impulse nicht mehr reihlauf sind (durch Filtrierung oder hoher Frequenz)

Bei PWM wird das Verhältnis zwischen Einschaltzeit und Periodendauer eines Rechtecksignals bei fester Grundfrequenz variiert.

Test Uc:

D1C

Übungsaufg.

- 1) Welche Periodendauern ergibt sich, wenn der μC mit 8MHz Taktfrequenz betrieben wird\* und bei jedem Overflow ein Ausgangspin invertiert wird. Prescalerwert = 64  
\*, der Timer 0 in Normal Mode programmiert wird!

$$T = \frac{1}{8\text{MHz}} = 125\text{ ns} \rightarrow 125 \cdot 64 = 8\mu\text{s} \cdot 23,6872,098\text{ ms} \cdot 2 = 4,096\text{ ms}$$
$$\frac{1}{8\text{MHz}} = 125\text{ kHz}$$

- 2) Mit dem Timer 0 soll am Normal Mode bei einer Taktfrequenz von 4MHz, Prescaler 8 Ausgang PortB bit 7 jede Ms gestoßt werden. schreibe ein vollständiges Programm

$$\frac{1}{4\text{MHz}} = 250\text{ ns}$$

$$\text{Prescaler } 8 \rightarrow 2\mu\text{s} \quad \text{Overflow: } 256 \cdot 2\mu\text{s} = 512\mu\text{s}$$

variable main () {

DPRB = 0x00;

int count != 0;

TCCR0I = (1<<SO1);

while (1) {

if (TIFR & (1<<TOV0)) {

TIFR |= (1<<TOV0);

count++;

if (count == 2) {

PORTR |= (1<<PB7);

3) wie 2 nur mit Interrupt & Service Routine

int main ()

{ DDRB = 0x FF;

TCCR0 = (1<< CS01); // Timer 0 Normal Mode, Prescale = 8

TIMSK = (1<< TOIE0); // Enable Interrupt für Timer 0 Overflow

sei(); // Generate Interrupt enable

while(1){

}

ISR (TIMERO\_OVF\_vect) {

static int count = 0;

count++;

if (count == 2) {

count = 0;

PORTB = (1<< PB7);

}

}

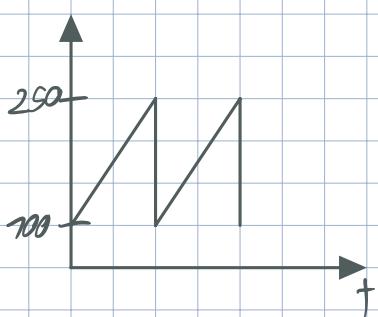
4) In welchen Zeitabsständen tritt ein Überlauf auf, wenn der Timer 0 mit einer Taktfrequenz von 1 MHz, einem Prescaler von 1024 und einem Verladerwert von 100 beliefert wird.

Schreibe auch das Programm (ohne Interrupt)

$$f_{CPU} = 1 \text{ MHz} \Rightarrow T = 1 \mu\text{s}$$

$$\text{PRE} = 1/1024$$

Verladezeit 100



Overflow

$\Rightarrow$  Anzahl an Zählschritten  
756

Zählschritte 100, 101, ... 255, 101, 100, 101,

$\Rightarrow$  Dauer eines Zählschritts 1024 μs

0 kommt nur kurz vor, Software setzt Timer auf 100  $\Rightarrow T_{Op} = 156 \cdot \frac{1}{MHz}$   
= 160ns

int main() {

TCCR0 |= (1 << CS00) | (1 << CS02); // Prescaler auf 1024

while(1) {

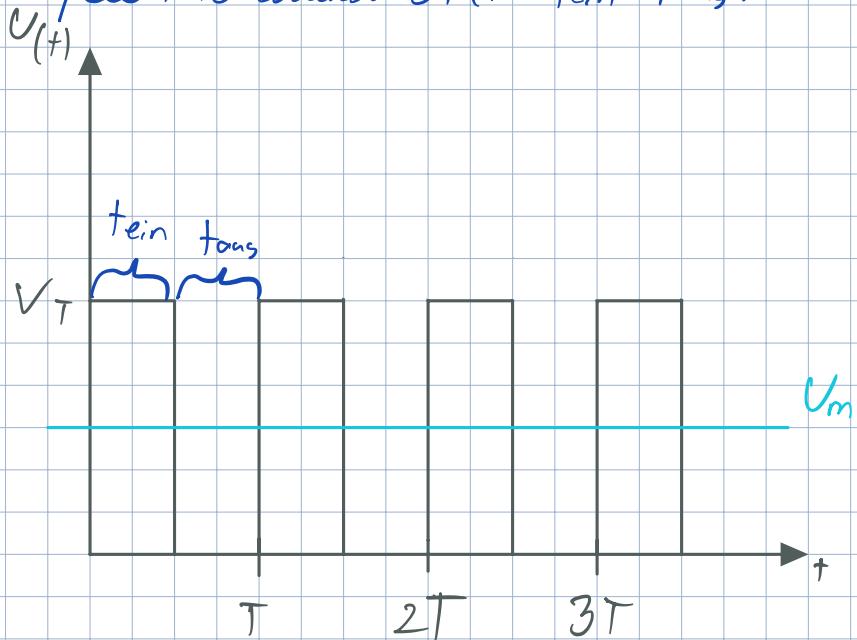
if (TIFR & (1 << TOV0)) {

TCNT0 = 100; // Vorladen auf 100  
TIFR |= (1 << TOV0);

}  
}  
}

weiter mit Stoff: (PWM)

Das Verhältnis zwischen der Einschaltzeit ( $t_{ein}$ ) und der freien Periodendauer ( $T = t_{ein} + t_{aus}$ ) wird als Duty Cycle bezeichnet



Bsp: Duty Cycle 50%.

$$U_m = U_f \frac{t_{ein}}{T} = U_i$$

$$0,5 = \frac{U_f}{2}$$

Der Mittelwert der Spannung ist direkt vom Duty-Cycle abhängig.

Träge Verbraucher (Motor, Sägen) können direkt an ein PWM Signal angeschlossen werden.

Bei LEDs erfolgt die Demodulation durch das Kugelelement. Wird die Kurzgangspannung als reine Abgangsspannung benötigt, dann kann die Mittelung des Signals mit einem RC-Glied erfolgen.

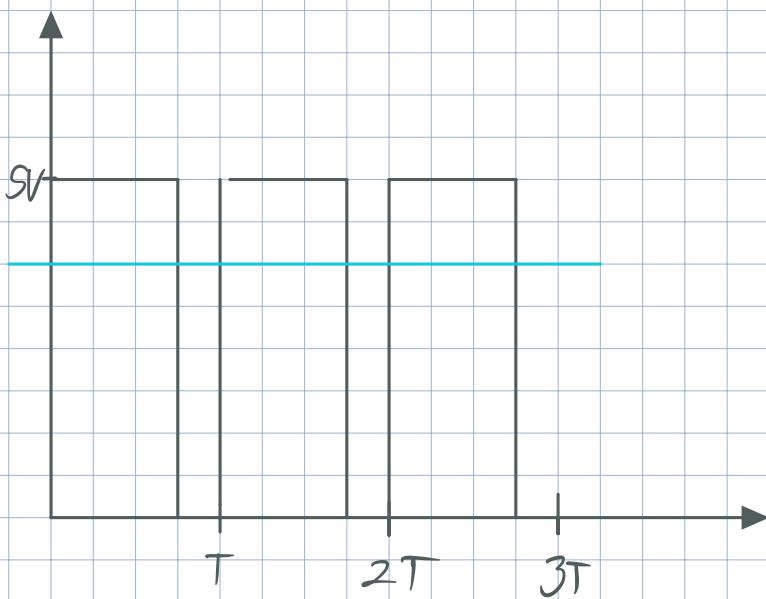
Bsp.:

Berechne & zeichne:

Es soll mittels PWM eine mittlere Spannung von 3,75V ausgegeben werden. Der µ-Controller wird mit 5V versorgt, die PWM-Frequenz beträgt 250Hz.

$$DC = \frac{3,75V}{5V} = 0,75 = 75\%$$

$$T = \frac{1}{f} = \frac{1}{20\text{Hz}} = 40\text{ms}$$



## Fast PWM mit Timer0

Mit der Fast PWM wird am OC0-Re ein PWM-Signal generiert.

Das Signal wird bei jedem Timer-Overflow auf High gesetzt.

Beim erreichen des Vergleichwerts im OCR0 Register wird es auf Null gesetzt. Mit dem Wert im OCR0 Register kann also der Duty-Cycle eingestellt werden, mit dem Prescaler die PWM-Frequenz.

Dadurch, dass das PWM-Signal pro Timer-Interval zweimal toggelt, sind höhere Frequenzen, als mit dem CTC-Modus möglich.

Um symmetrische PWM-Signale zu generieren, kann die Phasenverschiebung PWM verwendet werden. (siehe ATMEGA DDL S. 79)

Bsp:

Wie müssen TCCR0 und OCR0 programmiert werden, damit man aus Ausgang OC0 ein PWM Signal mit  $DC = \frac{1}{3}$  und eine  $f_{pwm} = 2\text{ kHz}$  hat?  $f_{cpu} = 4\text{ MHz}$

$$OCR0 = \frac{1}{3} \rightarrow \frac{256}{3} = 85,3$$

$$f_{pwm} = \frac{f_{cpu}}{N \cdot 256}$$

$$\frac{f_{cpu}}{f_{pwm} \cdot 256} = N \quad N = 8$$

$$TCCR0 = (1 \ll WGM00) | (1 \ll WGM01);$$

$$TCCR0 = (1 \ll CS01);$$

$$OCR0 = 85,$$

## Bsp CTC

Timer 0 in CTC Modus + Interrupt

$$f_{CPU} = 1 \text{ MHz}$$

$$\text{Prescaler} = 1024$$

Schreibe ein Programm, das eine LED an Port B mit einer Blinkfrequenz von ca. 10 Hz blinken lässt  $\Rightarrow$  In der ISR soll Port B Bits runden getestet werden.

$$f = 10 \text{ Hz} \Rightarrow T = 100 \text{ ms}$$

$$\text{Timerstep: } 1 \mu\text{s} \cdot 1024 = 1,024 \text{ ms} \quad T_{CPU} = 1 \mu\text{s}$$

↑  
Prescaler

$$OCR0 = \frac{50 \text{ ms}}{1,024 \text{ ms}} - 1 = 48,8 - 1 = 47,8 = \underline{\underline{48}}$$

```
int main(void) {
```

$$DDRB = 0 \times FF$$

$$PORTB = 0 \times FF$$

$$TCCR0 = (1 \ll WGMD1) | (1 \ll CS02) | (1 \ll CS00)$$

$$OCR0 = 48;$$

$$TIMSK = 1 = (1 \ll OCIE0);$$

sei();

while(1);

3

```
ISR(Timer0_comp_vect) {
```

$$PORTD = (1 \ll PB0);$$

3

Bsp: Timer 0 in CTC-Modus Prescaler = 8, f<sub>CPU</sub> = 8 MHz  
 Alle 100 µs soll ein Interrupt erzeugt werden. In den ISR wird die Variable count um erhöht, bis sie den Wert 10000 erreicht ( $\Rightarrow 1\text{s}$ ). Dann soll die Variable sek für Sekunden erhöht werden. (sek muss auch im Hauptprogramm verfügbar sein)

$8\text{MHz} = f_{\text{CPU}}$ , Prescaler = 8  $\Rightarrow f_{\text{timer}} = 1\text{MHz} \Rightarrow \text{Timerstep} = 1\mu\text{s}$   
 Es müssen also 100 Zählschritte sein

$$T_{\text{CPU}} = \frac{1}{f_{\text{CPU}}} = \frac{1}{8\text{MHz}} = 0,125\mu\text{s}$$

$$\text{Timerstep} = 0,125\mu\text{s} \cdot 8 = 1\mu\text{s}$$

$$OCR0 = 100 - 1$$

#include <avr/interrupt.h>

```
static volatile int sec; // static: nur aus main.c erreichbar
                        // volatile: Variable wird von Compiler nicht
                        // "optimiert"  $\Rightarrow$  wichtig wenn Zugriff von ISR
void main() {           und Hauptprogramm erfolgen
    TCCR0 |= (1<<CS01)|(1<<WGM01); // kann
    OCR0 = 99;
    TIMSK |= (1<<OCIE0);
    sei();
    while(1);
}
```

}

ISR (Timer0\_comp\_vect) {

```
static unsigned int count = 0; // Variable wird nur beim ersten mal initialisiert
if(count < 10000) {
    count++;
} else {
    sec++;
}
refarn count;
```

Bsp: Timer 0 im PWM Modus

$$f_{CPU} = 16 \text{ MHz}$$

$$\text{Prescaler} = 1024$$

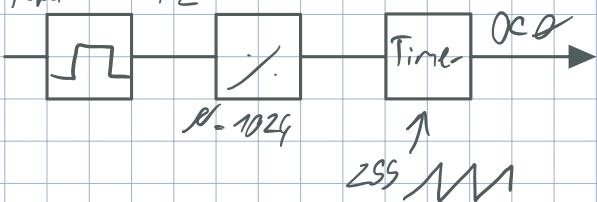
a) Berechne die PWM-Frequenz

b) Berechne OCRO für Duty Cycle = 1/10

c) Zeichne das Ausgangssignal für 2 Perioden und beschreibe das Diagramm

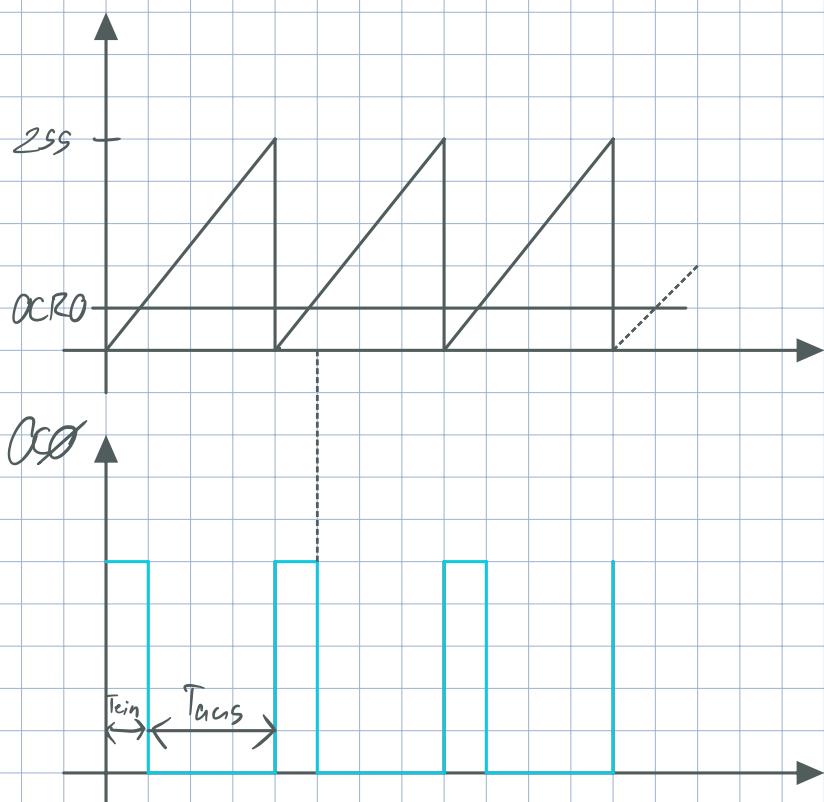
a)

$$f_{CPU} = 16 \text{ MHz} \quad \text{Prescaler} = 1024 \text{ Hz}$$



$$f_{PWM} = \frac{f_{CPU}}{N \cdot 256} = \frac{16 \cdot 10^6}{1024 \cdot 256} = 61,03 \text{ Hz}$$

b)



$$DC = \frac{T_{ein}}{T} \quad T = T_{ein} + T_{aus}$$

$$DC = \frac{1}{10} = \frac{OCR0}{256} \Rightarrow OCR0 = \frac{256}{10}$$

$$= 25,6 \rightarrow \text{Auffrunken} - 2G$$

