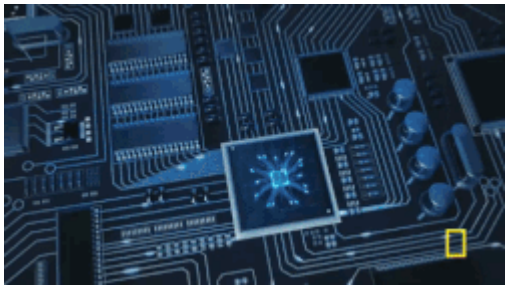
Timer und Interrupts

Parameter	Kursinformationen
Veranstaltung:	`Digitale Systeme / Eingebettete Systeme`
Semester	`Wintersemester 2022/23`

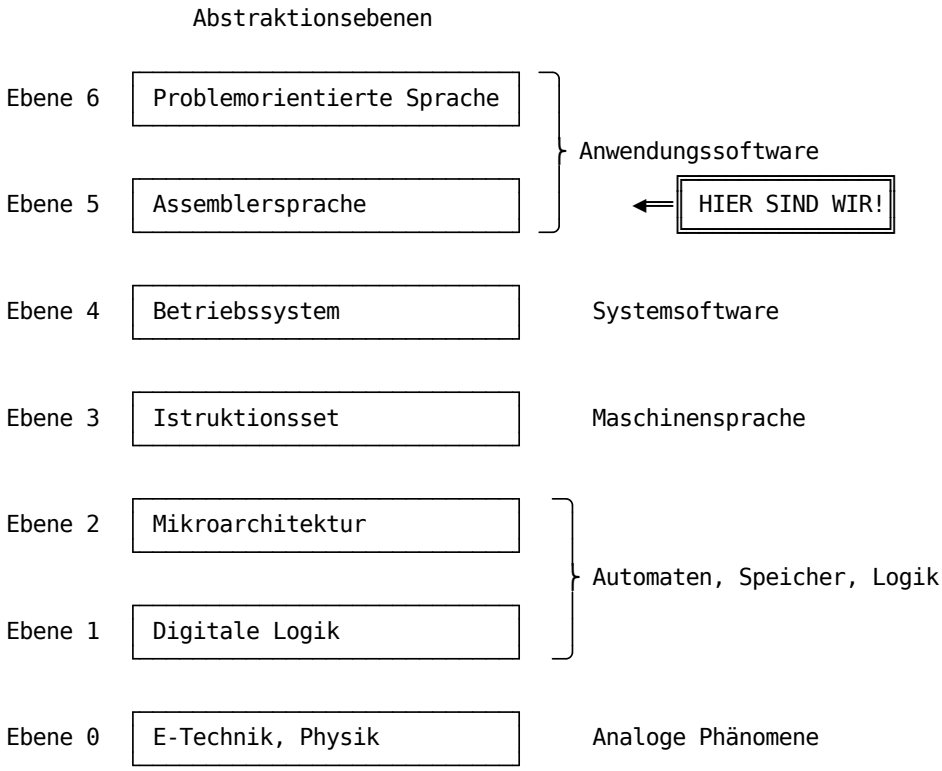
| Hochschule: | `Technische Universität Freiberg`

Inhalte:	`Konzepte und Nutzung von Timern und Interrupts`
Link auf GitHub:	https://github.com/TUBAF-IfI-LiaScript/VL_Softwareentwicklung/blob/master/15_TimerUndInterrupts.md
Autoren	Sebastian Zug & André Dietrich & Fabian Bär



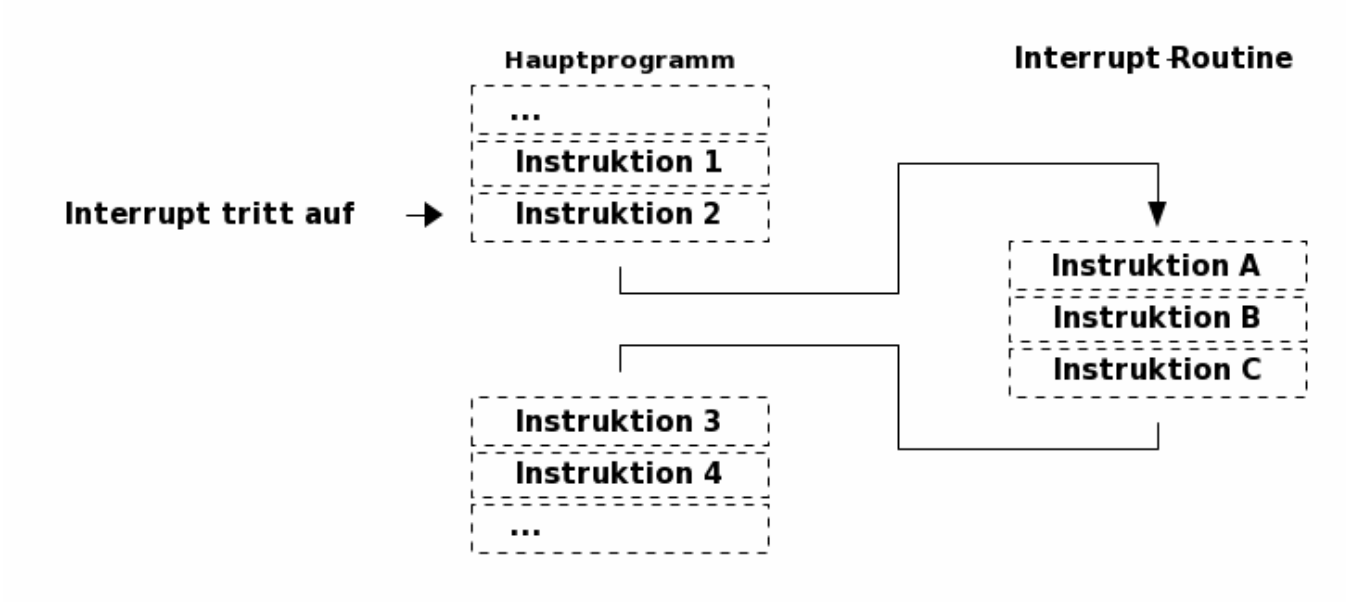
Fragen an die Veranstaltung

- Für welche Aufgaben werden die Timerfunktionen des Mikrocontrollers herangezogen?
- Wie sind ein Timer grundlegend aufgebaut (Capture&Compare)?
- Was ist PWM? Welche Arten der PWM Generierung gibt es?
- Welche Schritte werden bei der Abarbeitung von Interrupts durchlaufen?
- Welche Folgen können verpasste Interrupts haben? Beziehen Sie sich dabei insbesondere auf verschiedene Sensortypen, mit denen Sie in der Übung gearbeitet haben.
- Woraus ergibt sich die Prioritätenfolge der Interrupts beim AVR?
- Nennen Sie Interruptquellen beim AVR?
- Wie viele Interrupts können sich beim Atmega stauen, ohne dass einer verloren geht?



Interrupts

Ein Interrupt beschreibt die kurzfristige Unterbrechung der normalen Programmausführung, um einen, in der Regel kurzen, aber zeitlich kritischen, Vorgang abzuarbeiten.



Beispiele

Trigger	Beispiel
Pin Zustandswechsel	Drücken des Notausbuttons
Kommunikationsschnittstelle	Eintreffen eines Bytes
Analog Comperator	Resultat einer Analog-Comperator Auswertung
Analog-Digital-Wandler	Abschluss einer Wandlung
Timer	Übereinstimmung von Vergleichswert und Zählerwert

	Polling (zyklisches Abfragen)	Interrupts
	[1]	[1]
Vorteile	• Kein zusätzlicher Hardwareaufwand • Deterministisches Zeitverhalten	• Effiziente Abarbeitung bezogen auf die Auftretenshäufigkeit
Nachteile	• Auslastung des Prozessors • Verzögerung der Reaktion durch periodisches Abarbeitungskonzept	• Zusätzlicher Hardwareaufwand

[1] Firma Intel, Manual Intel 8259, Seite 3, Link

Formen der Unterbrechungsbehandlung

- Traps ... *a programmer initiated and expected transfer of control to a special handler routine (software interrupt).*
  - auf x86 Architekturen - INT
  - für AVR – kein eigner OP-Code
- Exceptions ... *is an automatically generated trap (coerced rather than that occurs in response to some exceptional condition. requested)*
  - auf x86 – für Division durch 0, fehlerhaften Speicherzugriff, illegal OP Code
  - für AVR – alle Resetquellen
- Interrupts (hardware interrupts) ... *are program control interruption based on an external hardware event (external to the CPU)*

Interupt Service Routine = {Trap handling, Exception handling, ...}

Ablauf

Schritt	**Beschreibung**
	Normale Programmabarbeitung ...
Vorbereitung	• Beenden der aktuelle Instruktion • (Deaktivieren der Interrupts) • Sichern des Registersatzes auf dem Stack
Ausführung	• Realisierung der Interrupt-Einsprung-Routine • Sprung über die Interrupt-Einsprungtabelle zur Interrupt-Behandlungs-Routine • Ausführung der Interrupt-Behandlungs-Routine
Rücksprung	• Wiederherstellen des Prozessorzustandes und des Speichers vom Stack • Sprung in den Programmspeicher mit dem zurückgelesenen PC
	Fortsetzung des Hauptprogrammes ...

Notwendige Funktionalität und Herausforderungen:

- Erkenne die Interruptquelle
- Bewerte die Relevanz für das aktuelle Programm
- Unterbreche den Programmablauf transparent und führe die ISR aus
- Beachte unterschiedliche Prioritäten für den Fall gleichzeitig eintreffender Interrupts

Merke ISRs sollten das Hauptprogramm nur kurz unterbrechen! Ein blockierendes Warten ist nicht empfehlenswert! Zudem darf die Abarbeitungsdauer nicht länger sein als die höchste Wiederauftretensfrequenz des Ereignisses.

Umsetzung auf dem AVR

Interrupts müssen individuell aktiviert werden. Dazu dient auf praktisch allen Mikrocontrollern ein zweistufiges System.

- Die Globale Interruptsteuerung erfolgt über ein generelles CPU-Statusbit, für den AVR Core is dies das I-Bit (Interrupt) im Statusregister (SREG).
- Die jeweilige lokale Interruptsteuerung erlaubt die individuelle Aktivierung über ein Maskenbit jeder Interruptquelle.

Damit können wahlweise Interrupts generell deaktiviert werden während die einzelne Konfiguration unverändert bleibt. Umgekehrt lassen sich spezifische Funktionalitäten ansprechen.

Eine ISR wird demnach nur dann ausgeführt, wenn

- die Interrupts global freigeschaltet sind
- das individuelle Maskenbit gesetzt ist
- der Interrupt (in dem konfigurierten Muster) eintritt

Analog Digital Converter, Seite 243 [megaAVR]

ADC Konfigurationsregister, Seite 258 [megaAVR]

Interrupt-Vector-Tabelle, Seite 74 [megaAVR]

[1]: Firma Microchip, megaAVR® Data Sheet, Seite 74, Link

Im Anschluss wird die Integration der Interrupt-Vektortabelle im Speicher dargestellt.

Der AVR deaktiviert die Ausführung von Interrupts, wenn er eine ISR aktiv ist. Dies kann aber manuell überschrieben werden.

[3]

[3]: Firma Microchip, megaAVR® Data Sheet, Seite 258, Link

ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-Write on ADCSRA, a pending interrupt can be disabled.

> **Merke:** Der "Speichermechanismus" ein und des selben Interrupts ist ein Bit groß. Von anderen Interrupts, die zwischenzeitlich eintreffen kann jeweils ein Auftreten gespeichert werden.

[megaAVR]: Firma Microchip, megaAVR® Data Sheet, Link

Praktische Interruptprogrammierung

Atomarer Datenzugriff

> **Merke:** Das Hauptprogramm kann grundsätzlich an jeder beliebigen Stelle unterbrochen werden, sofern die Interrupts aktiv sind.

Das bedeutet, dass entsprechende Variablen und Register, die sowohl im Hauptprogramm als auch in Interrupts verwendet werden, mit Sorgfalt zu behandeln sind.

Dies ist umso bedeutsamer, als das ein C Aufruf `port |= 0x03;` in drei Assemblerdirektiven übersetzt wird. An welcher Stelle wäre ein Interrupt kritisch?

```
  IN  r16, port
  ORI r16, 0x03
  OUT port, r16
```

```
// Manuelle Methode
cli(); // Interrupts abschalten
port |= 0x03;
sei(); // Interrupts wieder einschalten

// Per Macros aus dem avr gcc, schöner lesbar und bequemer
// siehe Doku der avr-libc, Abschnitt <util/atomic.h>
ATOMIC_BLOCK(ATOMIC_FORCEON) {
  port |= 0x03;
}
```

Mit Blick auf die Wiederverwendbarkeit des Codes sollte geprüft werden, ob die globalen Interrupts überhaupt aktiviert waren! Die avr-libc hält dafür die Methode `ATOMIC_BLOCK(ATOMIC_RESTORESTATE)` bereit.

Volatile Variablen

`volatile` Variabeln untersagen dem Compiler Annahmen zu deren Datenfluss zu treffen. Mit

volatile.c

```
volatile uint8_t i;

ISR( INT0_vect ){
  i++;
}

int main(){
  ...
  i = 0;
  while( 1 ) {
      Serial.println(i);
  }
}
```

Einführungsbeispiele

Externe Interrupts

Wenden wir das Konzept mal auf einen konkreten Einsatzfall an und lesen die externen Interrupts in einer Schaltung. Dabei sollen Aktivitäten an einem externen Interruptsensiblen Pin überwacht werden.

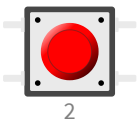
Aufgabe: Ermitteln Sie mit die PORT zugehörigkeit und die ID der Externen Interrupt PINs `INT0` und `INT1`. Welche Arduino Pin ID gehört dazu?

main.cpp

```
#define F_CPU 16000000UL

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

ISR(INT0_vect) {
  PORTB |= (1 << PB5);
}

int main (void) {
    DDRB |= (1 << PB5);
    DDRD &= ~(1 << DDD2);        // Pin als Eingang
    PORTD |= (1 << PD2);      // Pullup-Konfiguration
    EIMSK |= ( 1 << INT0);
    EICRA |= ( 1 << ISC01);
    sei();
    while (1);
    return 0;
}
```

Im Unterschied dazu sei im nachfolgenden Beispiel eine Implementierung des Polling-Mechanismus vorgestellt. Damit soll der Unterschied zwischen dem Interrupt-Modus und der kontinuierlichen Abfrage verdeutlicht werden.

Simulation time: 00:12.054



PORTB: 00000000 DDRB: 00100000 PINB: 00000000

ToggleLED.cpp

```
1  // preprocessor definition
2  #define F_CPU 16000000UL
3
4  #include <avr/io.h>
5  #include <util/delay.h>
6
7  int main (void) {
8     DDRB |= (1 << PB5);
9     int state = 0;
10    // aktives überwachen des Pins
11    while(1) {
12        if (state ^ ((PIND >> PD2) & 1)) {
13           state ^= 1;
14           if (state) {
15             PORTB ^= ( 1 << PB5 );
16           }
17        }
18        _delay_ms(1);
19    }
20    return 0;
21 }
```

```
starting simmulation...
compiling...
ready!
```

Frage: Erklären Sie den Ablauf der Aktualisierung der Zustandsvariablen in Zeile 12 und 13.

Analog Digitalwandler

Das folgende Beispiel nutzt den Analog-Digital-Wandler in einem teilautonomen Betrieb. Innerhalb der Interrupt-Routine wird das Ergebnis ausgewertet und jeweils eine neue Wandlung aktiviert.

Simulation time: 00:07.613

1.0 V

AnalogViaInterrupts.cpp

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

// Interrupt subroutine for ADC conversion complete interrupt
ISR(ADC_vect) {
    //Serial.println(ADCW);
    if(ADCW >= 600)
        PORTB |= (1 << PB5);
    else
        PORTB &=~(1 << PB5);
    ADCSRA |= (1<<ADSC);
}

int main(void){
    Serial.begin(9600);
    DDRB |= (1 << PB5);
    ADMUX = (1<<REFS0);
    ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1<<ADPS0) | (1<<ADEN)|(1
      );
    ADCSRA |= (1<<ADSC);                // Dummy read
    while(ADCSRA & (1<<ADSC));
    ADCSRA |= (1<<ADSC);                // Start conversion "chain"
    (void) ADCW;
    sei();                              // Set the I-bit in SREG

    int i = 0;
    while (1) {
        // Ich rechner fleißig!
       _delay_ms(5000);
    }

    return 0;
}
```

```
starting simmulation...
compiling...
ready!
```

Interrupt-Vektortabelle im AVR Programm

Und warum funktioniert das Ganze? Lassen Sie uns ein Blick hinter die Kulissen werfen.

Folgendes Programm, dass den externen Interrupt 0 auswertet wurde disassembliert.

```
#define F_CPU 16000000UL
```

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

ISR(INT0_vect) {
  PORTB |= (1 << PB5);
}

int main (void) {
   DDRB |= (1 << PB5);
   DDRD &= ~(1 << DDD2);
   PORTD |= (1 << PD2);
   EIMSK |= ( 1 << INT0);
   EICRA |= ( 1 << ISC01);
   sei();
   while (1);
   return 0;
}
```

main.asm

```
; Interrupt Vector Tabelle mit zugehörigen Sprungfunktionen
00000000 <__vectors>:
   0: 0c 94 34 00   jmp 0x68  ; 0x68 <__ctors_end>
   4: 0c 94 40 00   jmp 0x80  ; 0x80 <__vector_1>
   8: 0c 94 3e 00   jmp 0x7c  ; 0x7c <__bad_interrupt>

  60: 0c 94 3e 00   jmp 0x7c  ; 0x7c <__bad_interrupt>
  64: 0c 94 3e 00   jmp 0x7c  ; 0x7c <__bad_interrupt>

; Initialisierungsroutine für den Controller
00000068 <__ctors_end>:
  68: 11 24         eor r1, r1    ;       0 in R1
  6a: 1f be         out 0x3f, r1  ;       SREG = 0
  6c: cf ef         ldi r28, 0xFF ; 255
  6e: d8 e0         ldi r29, 0x08 ; 8     SP -> 0x8FF
  70: de bf         out 0x3e, r29 ; 62    SPH (0x3e)
  72: cd bf         out 0x3d, r28 ; 61    SPL (0x3d)
  74: 0e 94 4b 00   call  0x96  ; 0x96 <main>
  78: 0c 94 56 00   jmp 0xac  ; 0xac <_exit>

; Restart beim Erreichen eines nicht definierten Interrupts
0000007c <__bad_interrupt>:
  7c: 0c 94 00 00   jmp 0 ; 0x0 <__vectors>

; Interrupt Routine für External Interrupt 0
00000080 <__vector_1>:
  80: 1f 92         push  r1
  82: 0f 92         push  r0
  84: 0f b6         in  r0, 0x3f  ; 63
  86: 0f 92         push  r0
  88: 11 24         eor r1, r1
  ; PORTB |= (1 << PB5);
  8a: 2d 9a         sbi 0x05, 5 ; 5
  8c: 0f 90         pop r0
  8e: 0f be         out 0x3f, r0  ; 63
  90: 0f 90         pop r0
  92: 1f 90         pop r1
  94: 18 95         reti

00000096 <main>:
  ;DDRB |= (1 << PB5);
  96: 25 9a         sbi 0x04, 5 ; 4
  ; DDRD &= ~(1 << DDD2);
  98: 52 98         cbi 0x0a, 2 ; 10
  ; PORTD |= (1 << PORTD2);
  9a: 5a 9a         sbi 0x0b, 2 ; 11
  ; EIMSK |= ( 1 << INT0);
  9c: e8 9a         sbi 0x1d, 0 ; 29
```

```
  ; EICRA |= ( 1 << ISC01);
  9e: 80 91 69 00   lds r24, 0x0069 ; 0x800069 <__DATA_REGION_ORIGIN__+0x9>
  a2: 82 60         ori r24, 0x02 ; 2
  a4: 80 93 69 00   sts 0x0069, r24 ; 0x800069 <__DATA_REGION_ORIGIN__+0x9>
  ; sei();
  a8: 78 94         sei
  aa: ff cf         rjmp  .-2       ; 0xaa <main+0x14>

000000ac <_exit>:
  ac: f8 94         cli

000000ae <__stop_program>:
  ae: ff cf         rjmp  .-2       ; 0xae <__stop_program>
```

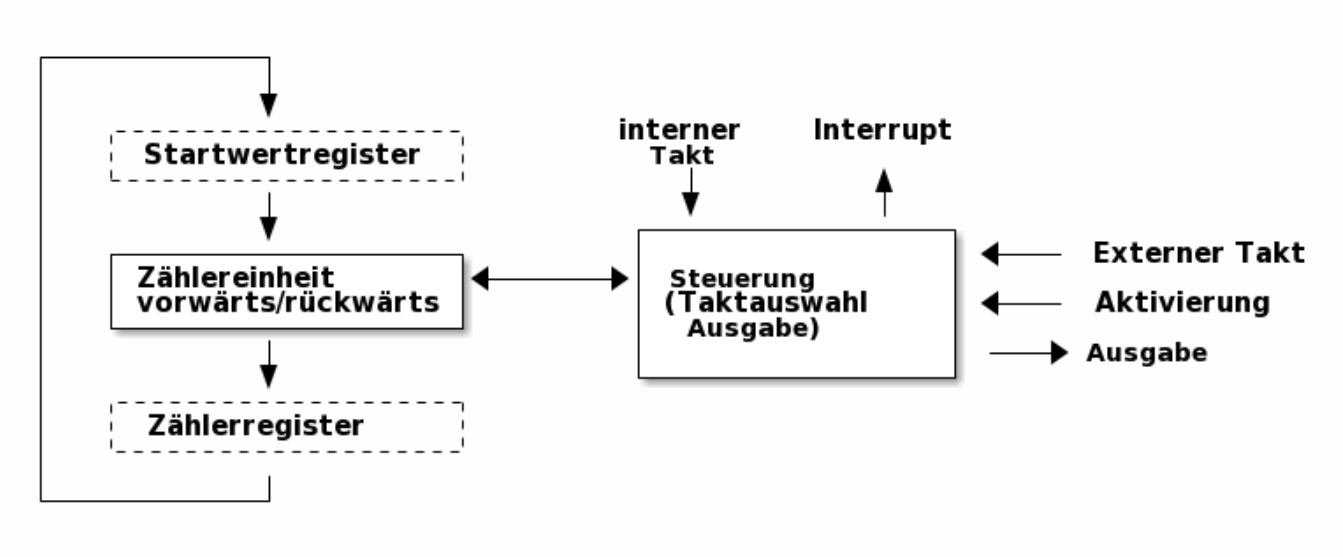
Timer

Aufgaben von Timer/Counter Lösungen in einem Mikrocontroller:

- Zählen von Ereignissen
- Messen von Zeiten, Frequenzen, Phasen, Perioden
- Erzeugen von Intervallen, Pulsfolgen, Interrupts
- Überwachen von Ereignissen und Definition von Zeitstempeln

Basisfunktionalität

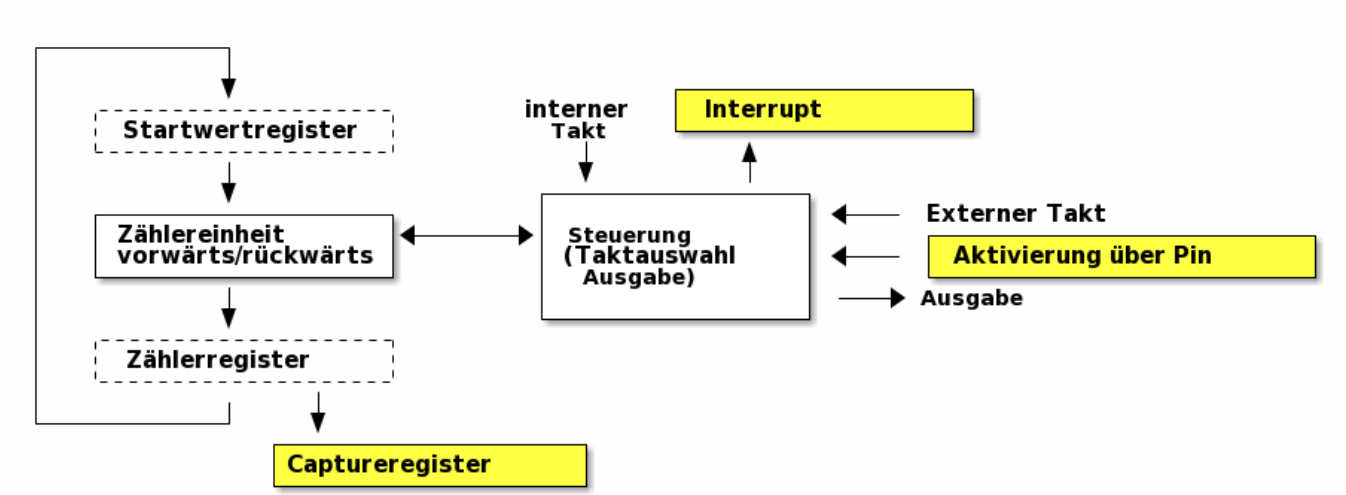
Die Grundstruktur eines Zählerbaustein ergibt sich aus folgenden Komponenten:



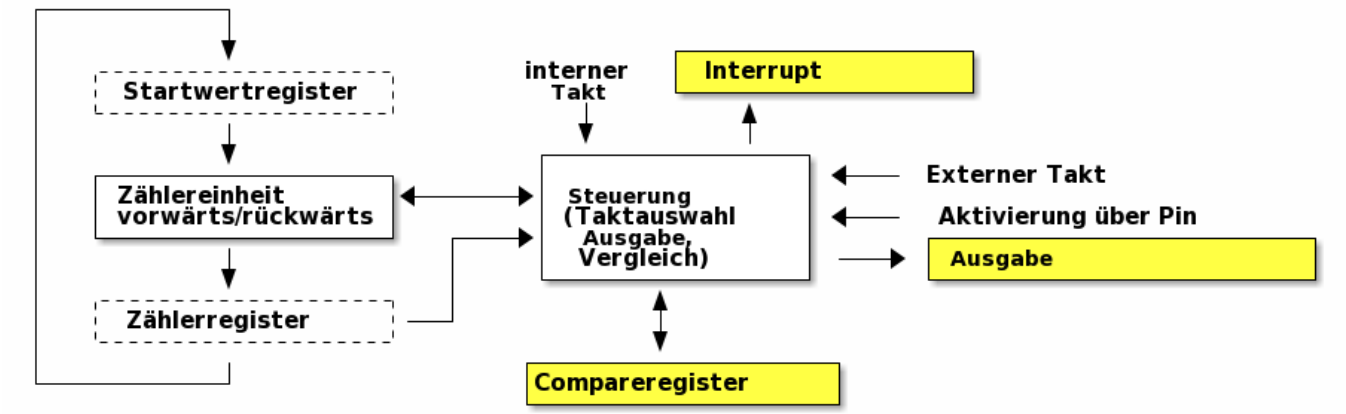
Capture/Compare-Einheiten Nutzen die Basis-Timerimplementierung. Sie können externe Signale aufnehmen und vergleichen, aber beispielsweise auch Pulsmuster erzeugen.

Sie besitzt meist mehrere Betriebsmodi:

- Timer/Zähler-Modus: Aufwärtszählen mit verschiedenen Quellen als Taktgeber. Bei Zählerüberlauf kann ein Interrupt ausgelöst werden.
- Capture-Modus: Beim Auftreten eines externen Signals wird der Inhalt des zugeordneten (laufenden) Timers gespeichert. Auch hier kann ein Interrupt ausgelöst werden.



- Compare-Modus: Der Zählerstand des zugeordneten Timers wird mit dem eines Registers verglichen. Bei Übereinstimmung kann ein Interrupt ausgelöst werden.



Merke: Diese Vorgänge laufen ausschließlich in der Peripherie-Hardware ab, beanspruchen also, abgesehen von eventuellen Interrupts, keine Rechenzeit.

Entsprechend ergibt sich für den Compare-Modus verschiedene Anwendungen und ein zeitliches Verhalten entsprechend den nachfolgenden Grafiken.

Umsetzung im AVR

Der AtMega328 bringt 4 unabhängige Timersysteme mit:

- *Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode*
- *One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode*
- *Real Time Counter with Separate Oscillator*
- *Six PWM Channels*

Dabei soll die Aufmerksamkeit zunächst auf dem 16-bit Timer/Zähler liegen.

Timer Baustein im AVR Controller, Seite 250 [megaAVR]

Folgende Fragen müssen wir die Nutzung des Timers beantwortet werden:

- Wer dient als Trigger?
- Wie groß wird der Prescaler gewählt?
- Welche maximalen und die minimalen Schranken des Zähler werden definiert?
- Wird ein Ausgang geschaltet?

[megaAVR] Firma Microchip, megaAVR® Data Sheet, Link

Timer Modi

Timer-Modi bestimmen das Verhalten des Zählers und der angeschlossenen Ausgänge / Interrupts. Neben dem als Normal-Mode bezeichneten Mechanismus existieren weitere Konfigurationen, die unterschiedliche Anwendungsfelder bedienen.

Clear to Compare Mode (CTC)

CTC mode, Seite 141 [megaAVR]

Die Periode über eine OCnA Ausgang ergibt sich entsprechend zu

$$f_{OCnA} = \frac{f_{clk_i/o}}{2 \cdot N \cdot (1 + OCRnA)}$$

Der Counter läuft zwei mal durch die Werte bis zum Vergleichsregister OCRnA. Die Frequenz kann durch das Setzen eine Prescalers korrigiert werden.

Fast PWM

Fast PWM generation, Seite 141 [megaAVR]

Die Periode des Signals an OCRnA wechselt während eines Hochzählens des Counters. Damit kann eine größere Frequenz bei gleicher Auflösung des Timers verglichen mit CTC erreicht werden.

$$f_{OCnA} = \frac{f_{clk_i/o}}{N \cdot (1 + TOP)}$$

Phase Correct PWM

Phase correct PWM generation, Seite 141 [megaAVR]

$$f_{OCnA} = \frac{f_{clk_i/o}}{2 \cdot N \cdot (TOP)}$$

[megaAVR] Firma Microchip, megaAVR® Data Sheet, Link

Timer-Funktionalität (Normal-Mode)

Für die Umsetzung eines einfachen Timers, der wie im nachfolgenden Beispiel jede Sekunde aktiv wird, genügt es einen entsprechenden Vergleichswert zu bestimmen, den der Zähler erreicht haben muss.

Counter implementation, Seite 126 [megaAVR]

Simulation time: 00:00.830

TCNT1: 0x84,2 bytes, littleendian: 0011001010101000

avrlibc.cpp

```cpp
#ifndef F_CPU
#define F_CPU 16000000UL // 16 MHz clock speed
#endif

//16.000.000 Hz / 1024 = 15.625

int main(void)
{
  DDRB |= (1 << PB5);  // Ausgabe LED festlegen
  // Timer 1 Konfiguration
  //         keine Pins verbunden
  TCCR1A  = 0;
  TCCR1B  = 0;
  // Timerwert
  TCNT1   = 0;
  TCCR1B |= (1 << CS12) | (1 <<CS10);  // 1024 als Prescaler-Wert

  while (1) //infinite loop
  {
    if (TCNT1>15625){
      TCNT1 = 0;
      PINB = (1 << PB5); // LED ein und aus
    }
  }
}
```

```
starting simmulation...
compiling...
ready!
```

Was stört Sie an dieser Umsetzung?

Wir lassen den Controller den Vergleichswert kontinuierlich auslesen. Damit haben wir noch nichts gewonnen, weil der Einsatz der Hardware unser eigentliches System nicht entlastet. Günstiger wäre es, wenn wir ausgehend von unseren Zählerzuständen gleich eine Schaltung des Ausganges vornehmen würden.

[megaAVR] Firma Microchip, megaAVR® Data Sheet, Link

Compare Mode

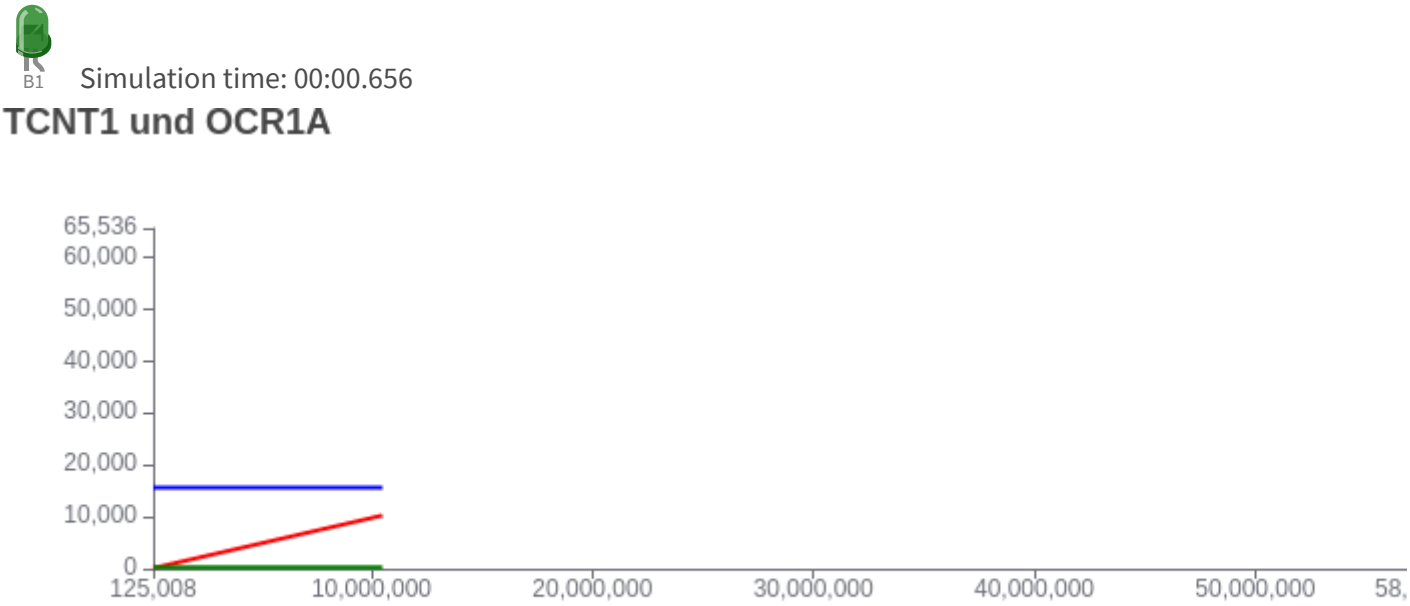
Wir verknüpfen unseren Timer im Comparemodus mit einem entsprechenden Ausgang und stellen damit sicher, dass wir die Ausgabe ohne entsprechende Ansteuerung im Hauptprogramm aktivieren.

Outputmode des Timerbausteins, Seite 126, [megaAVR]

> **Frage:** Welchen physischen Pin des Controllers können wir mit unserem Timer 1 ansteuern?

Setzen wir also die Möglichkeiten des Timers vollständig ein, um das Blinken im Hintergrund ablaufen zu lassen. Ab Zeile 13 läuft dieser Prozess komplett auf der Hardware und unser Hauptprogramm könnte eigenständige Aufgaben wahrnehmen.

Normal Mode Konfiguration



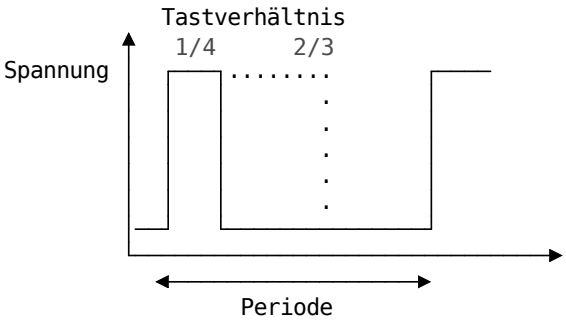
avrlibc.cpp

```cpp
#ifndef F_CPU
#define F_CPU 16000000UL // 16 MHz clock speed
#endif

int main(void)
{
  DDRB |= (1 << PB1);  // Ausgabe LED festlegen
  TCCR1A = 0;
  TCCR1B = 0;
  TCCR1B |= (1 << CS12) | (1 <<CS10);  // 1024 als Prescale-Wert
  TCCR1A |= (1 << COM1A0);
  OCR1A = 15625;

  while (1) _delay_ms(500);
}
```

```
starting simmulation...
compiling...
ready!
```

Was passiert, wenn die Aktivierung und Deaktivierung mit einer höheren Frequenz vorgenommen wird? Die effektiv wirkende Spannung wird durch den Mittelwert repräsentiert. Damit ist eine Quasi-Digital-Analoge Ausgabe ohne eine entsprechende Hardware möglich.

> **Merke:** Reale Analog-Digital-Wandler würden ein Ergebnis zwischen 0 und 2^n projiziert auf eine Referenzspannung ausgeben. PWM generiert diesen Effekt durch ein variierendes Verhältnis zwischen an und aus Phasen.



Im folgenden wird der **Fast PWW Mode** genutzt, um auf diesem Wege die LED an PIN 9 zu periodisch zu dimmen. Dazu wird der Vergleichswert, der in OCR1A enthalten ist kontinuierlich verändert.

avrlibc.cpp

```
#ifndef F_CPU
#define F_CPU 16000000UL // 16 MHz clock speed
#endif

int main(void){
  DDRB |=  (1<<PORTB1); //Define OCR1A as Output
  TCCR1A |= (1<<COM1A1) | (1<<WGM10);  //Set Timer Register
  TCCR1B |= (1<<WGM12) | (1<<CS11);
  OCR1A = 0;
  int timer;
  while(1) {
      while(timer < 255){ //Fade from low to high
         timer++;
         OCR1A = timer;
         _delay_ms(50);
      }
      while(timer > 1){ //Fade from high to low
         timer--;
         OCR1A = timer;
         _delay_ms(50);
      }
   }
}
```

[megaAVR] Firma Microchip, megaAVR® Data Sheet, Link

Capture Mode

Capture Mode, Seite, 126, [megaAVR]

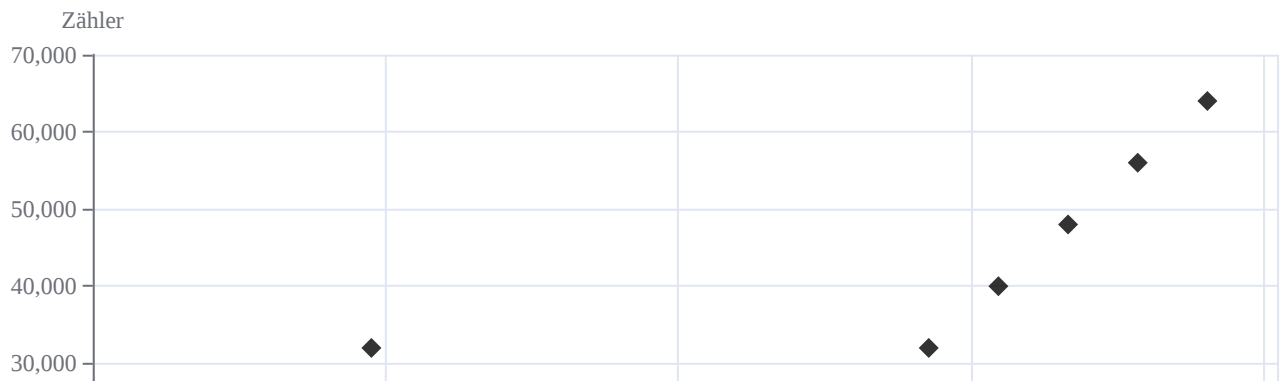
avrlibc.cpp

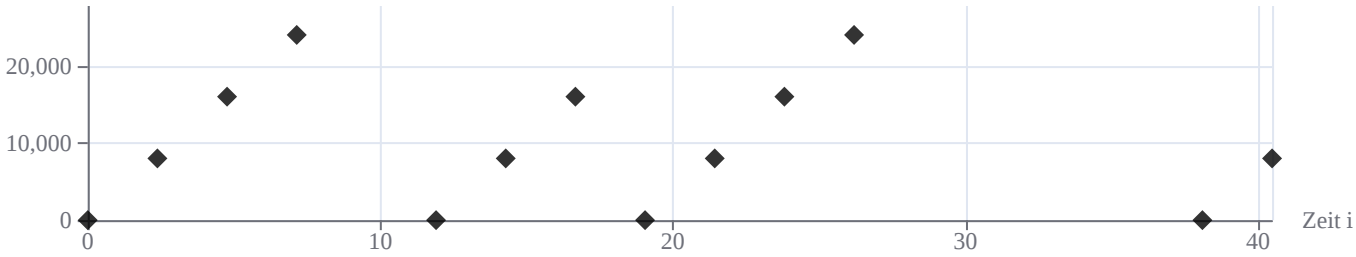
```
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
  Serial.begin(9600);
  DDRB = 0;
  PORTB = 0xFF;

  TCCR1A = 0; // Normal Mode
  TCCR1B = 0;
  //         1024 als Prescale-Wert     Rising edge
  TCCR1B  = (1 << CS12) | (1 <<CS10) | (1 << ICES1);
  TIFR1 = (1<<ICF1);  // Löschen des Zählerwertes

  while (1)
  {
      Serial.println("Waiting for Button push");
      Serial.flush();
      TCNT1 = 0;  // Löschen des Zählerwertes
      while ((TIFR1 & (1<<ICF1)) == 0);
      TIFR1 = (1<<ICF1);
      Serial.println(ICR1);
      Serial.flush();
      _delay_ms(500);
  }
  return  0;
}
```

Mit dem Schreiben des Flags `ICF1` kann der Eintrag gelöscht werden.

_This flag is set when a capture event occurs on the ICP1 pin. When the Input Capture Register (ICR1) is set by the WGM13:0 to be used as the TOP value, the ICF1 Flag is set when the counter reaches the TOP value. ICF1 is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICF1 can be cleared by writing a logic one to its bit location.

Entwicklung des Timerinhalts





Frage: Sie wollen die Ausgabe in Ticks in eine Darstellung in ms überführen. Welche Kalkulation ist dafür notwendig?

Problem: Wie groß ist das maximal darstellbare Zahlenintervall?

[megaAVR] Firma Microchip, megaAVR® Data Sheet, Link

Anwendungen

Eingangszähler

Übersicht Timermodul Seite 250, [megaAVR]

Wir wollen einen Eingangszähler entwerfen, der die Ereignisse als Zählerimpulse betrachtet und zusätzlich mit einem Schwellwert vergleicht.

avrlibc.cpp

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

ISR (TIMER1_COMPA_vect)
{
  Serial.print("5 Personen gezählt");
  TCNT1 = 0;
}

int main(void)
{
  Serial.begin(9600);
  TCCR1A = (1 << WGM12);
  // CTC Modus (Fall 4)
  TCCR1B |= (1 << CS12) | (1 << CS11) | (1 << CS10);
  OCR1A = 5;
  TIMSK1 |= (1<<OCIE1A);
  sei();

  while (1)
  {
    Serial.print(TCNT1);
    Serial.print(" ");
    _delay_ms(500);
  }
  return  0;
}
```

Glimmeffekt einer LED

Simulation time: 00:00.031

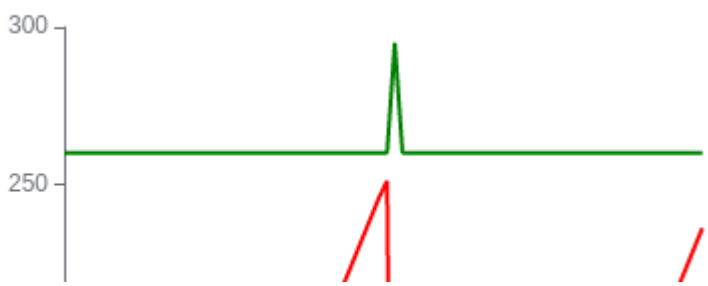
PORTB: 0x25: 00000000 **DDRB:** 0x24: 00000010 **PINB:** 0x23: 00000000

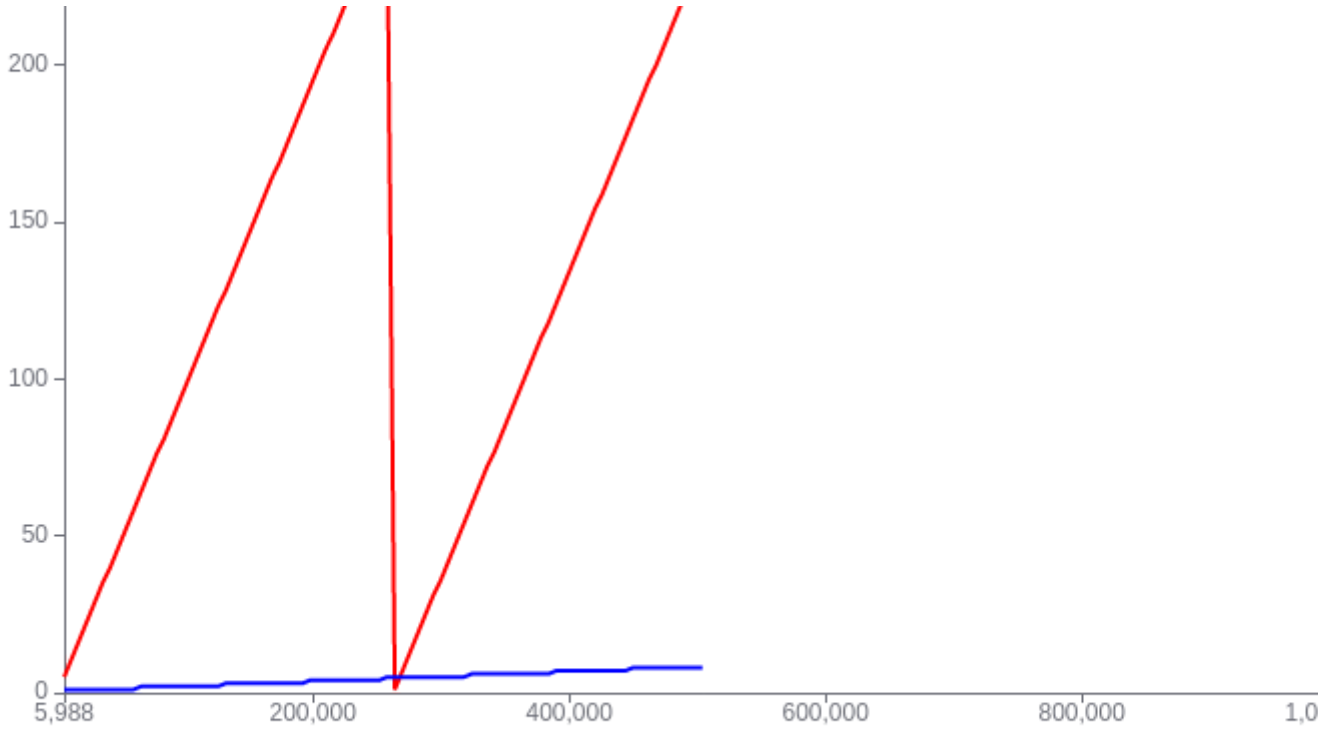
TCCR1: 0x80,2 bytes, littleendian: 0000110110000001

TCNT1: 0x84,2 bytes, littleendian: 0000000011101100

OCR1A: 0x88,2 bytes, littleendian: 0000000000001000

TCNT1 und OCR1A





Dimmer.cpp

```
#ifndef F_CPU
#define F_CPU 16000000UL // 16 MHz clock speed
#endif

int main(void){
  DDRB |=  (1 << PORTB1); //Define OCR1A as Output
  TCCR1A |= (1 << COM1A1) | (1 << WGM10);  //Set Timer Register
  TCCR1B |= (1 << CS12) | (1 << CS10) | (1 << WGM12);
  OCR1A = 0;
  int timerCompareValue = 0;
  while(1) {
   while(timerCompareValue < 255){ //Fade from low to high
    timerCompareValue++;
    OCR1A = timerCompareValue;
    _delay_ms(4);

  }
   while(timerCompareValue > 0){ //Fade from high to low
    timerCompareValue--;
    OCR1A = timerCompareValue;
    _delay_ms(4);
    }
   }
}
```

```
starting simmulation...
compiling...
ready!
```

[megaAVR] Firma Microchip, megaAVR® Data Sheet, Link