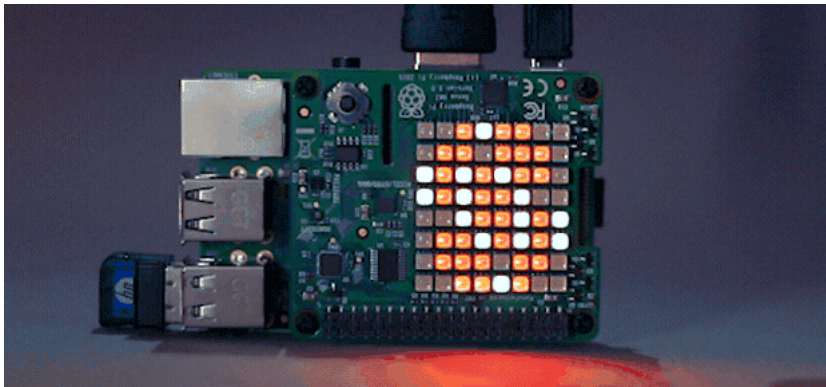


Interrupts und Timer

Parameter	Kursinformationen
Veranstaltung:	Vorlesung Digitale Systeme
Semester	Sommersemester 2022
Hochschule:	Technische Universität Freiberg
Inhalte:	Interrupt und Timerkonzepte des AVR
Link auf den GitHub:	https://github.com/TUBAF-lfi-LiaScript/VL_DigitaleSysteme/blob/main/lectures/05_InterruptsTimer.md
Autoren	Sebastian Zug, Karl Fessel & Andr�� Dietrich

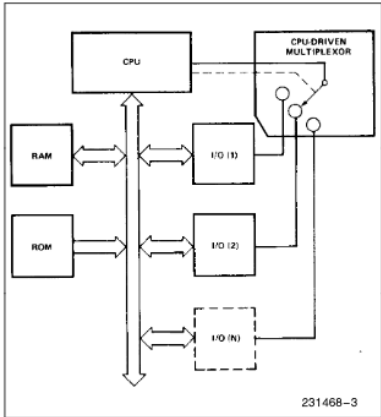
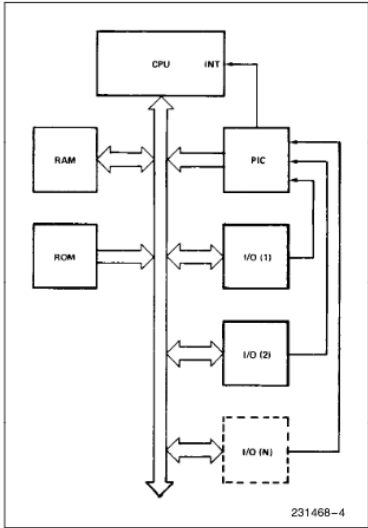


Interrupts

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

Beispiele

Trigger	Beispiel
Pin Zustandswechsel	Dr��cken des Notausbuttons
Kommunikationsschnittstelle	Eintreffen eines Bytes
Analog Comperator	Resultat einer Analog-Comperator Auswertung
Analog-Digital-Wandler	Abschluss einer Wandlung
Timer	��bereinstimmung von Vergleichswert und Z��hlerwert

	Polling (zyklisches Abfragen)	Interrupts
	 <p>Figure 3a. Polled Method</p> <p>[1]</p>	 <p>Figure 3b. Interrupt Method</p> <p>[1]</p>
Vorteile	<ul style="list-style-type: none"> Kein zusätzlicher Hardwareaufwand Deterministisches Zeitverhalten 	<ul style="list-style-type: none"> Effiziente Abarbeitung bezogen auf die Auftretenshäufigkeit
Nachteile	<ul style="list-style-type: none"> Auslastung des Prozessors Verzögerung der Reaktion durch periodisches Abarbeitungskonzept 	<ul style="list-style-type: none"> 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 eigener 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 Speicherezugriff, illegal OP Code
 - für AVR – alle Resetquellen
- Interrupts (hardware interrupts) ... *are program control interruption based on an external hardware event (external to the CPU)*

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

Ablauf

Schritt	Beschreibung
	Normale Programmabarbeitung ...
Vorbereitung	<ul style="list-style-type: none"> • Beenden der aktuelle Instruktion • (Deaktivieren der Interrupts) • Sichern des Registersatzes auf dem Stack
Ausführung	<ul style="list-style-type: none"> • Realisierung der Interrupt-Einsprung-Routine • Sprung über die Interrupt-Einsprungtabelle zur Interrupt-Behandlungs-Routine • Ausführung der Interrupt-Behandlungs-Routine
Rücksprung	<ul style="list-style-type: none"> • 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.

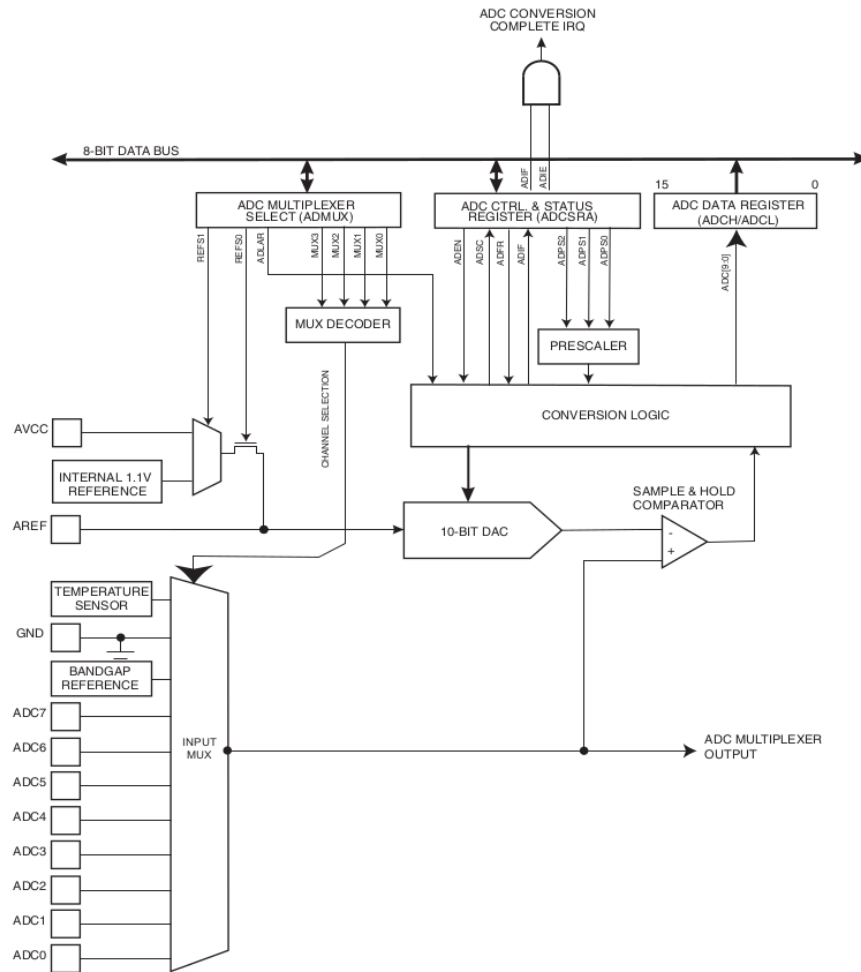
- Die Globale Interruptsteuerung erfolgt über ein generelles CPU-Statusbit, für den AVR Core ist dies das I-Bit (Interrupt) im Statusregister (SREG).
- Die jeweilige lokale Interruptsteuerung erlaubt die individuelle Aktivierung über ein Maskenbit jeder Interruptquelle.

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

Figure 24-1. Analog to Digital Converter Block Schematic Operation



12.4 Interrupt Vectors in ATmega328 and ATmega328P

Table 12-6. Reset and Interrupt Vectors in ATmega328 and ATmega328P

VectorNo.	Program Address ⁽²⁾	Source	Interrupt Definition
1	0x0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

Interrupt-Vektortabelle des AVR ^[AVR328] Seite 74

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.

24.9.2 ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Controllregister ADCSR des Analog-Digital-Wandlers ^[AVR328] Seite 258

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.

[AVR328] 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` Variablen 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 << DD2); // Pin als Eingang
    PORTD |= (1 << PORTD2); // Pullup-Konfiguration
    EIMSK |= (1 << INT0);
    EICRA |= (1 << ISC01);
    sei();
    while (1);
    return 0;
}
```

Merke: Der AtMega328 unterscheidet zwei Modi des externen Interrupts - eine 1:1 Zuordnung für die "echten" externen Interrupts und die sogenannten "Pin Change Interrupts" `PCINT23...0`.

Controllregister ADCSR des Analog-Digital-Wandlers ^[AVR328] Seite 79

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.

Als Demonstrator dient ein Spannungsteiler über einen lichtabhängigen Widerstand.

```
#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 << ADIF);

    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) {
        Serial.print("Ich rechne fleißig ... ");
        Serial.println(i++);
        _delay_ms(50);
    }

    return 0;
}
```

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 << PORTD2);
    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>
```

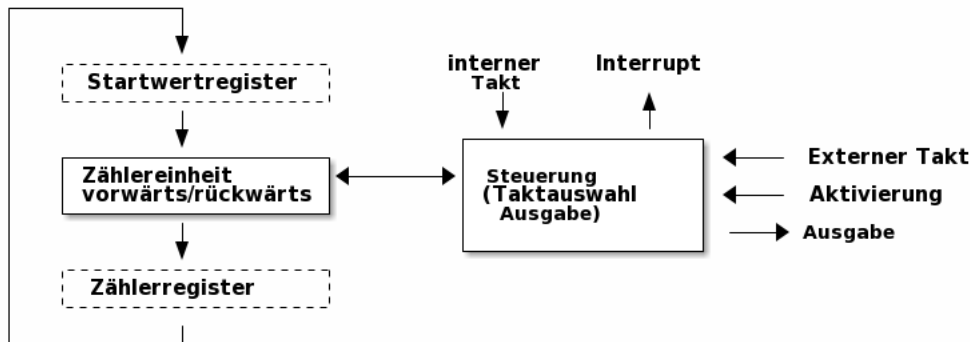
Timer

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

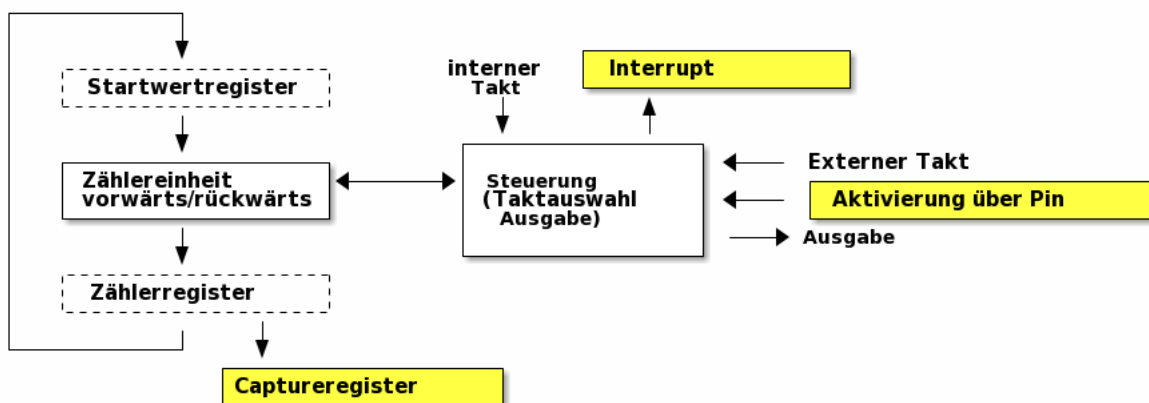
Die Grundstruktur eines Zählerbaustein ergibt sich aus folgenden Komponenten:



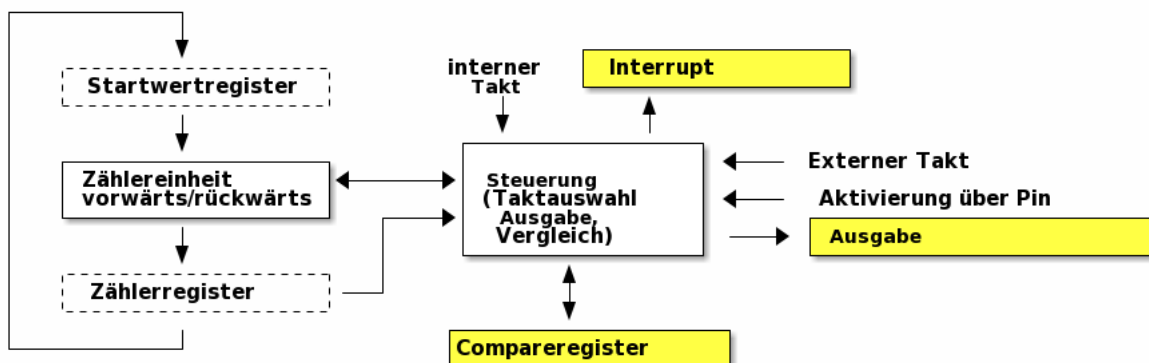
Capture/Compare-Einheiten nutzen die Basis-Timerimplementierung. Sie können externe Signale aufnehmen und vergleichen, aber beispielsweise auch Pulsmuster erzeugen.

Sie besitzen 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.

Generierung von Interrupts in bestimmten Zeitintervallen

Generierung eines PWM Signals

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.

Aufbau des 16 Bit Timers [\[AVR328\]](#)

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 Zählers werden definiert?
- Wird ein Ausgang geschaltet?

[AVR328] 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 Modus des AVR [\[AVR328\]](#) Seite 141

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 eines Prescalers korrigiert werden.

Fast PWM

FastPWM Modus des AVR [\[AVR328\]](#) Seite 143

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

PhaseCorrectPWM Modus des AVR [\[AVR328\]](#) Seite 145

$$f_{OCnA} = \frac{f_{clk_{i/o}}}{2 \cdot N \cdot TOP}$$

[AVR328] 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.

FastPWM Modus des AVR ^[AVR328] Seite 126



13 Simulation time: 00:18.000

avrlibc.cpp

```
1  #ifndef F_CPU
2  #define F_CPU 16000000UL // 16 MHz clock speed
3  #endif
4
5  //16.000.000 Hz / 1024 = 15.625
6
7  int main(void)
8  {
9      Serial.begin(9600);
10     DDRB |= (1 << PB5); // Ausgabe LED festlegen
11     // Timer 1 Konfiguration
12     //     keine Pins verbunden
13     TCCR1A = 0;
14     TCCR1B = 0;
15     // Timerwert
16     TCNT1 = 0;
17     TCCR1B |= (1 << CS12) | (1 << CS10); // 1024 als Prescale-Wert
18
19     while (1) //infinite loop
20     {
21         if (TCNT1 > 15625) {
22             TCNT1 = 0;
23             PINB = (1 << PB5); // LED ein und aus
24         }
25     }
26 }
```

Sketch uses 1150 bytes (3%) of program storage space. Maximum is 32256 bytes.
Global variables use 175 bytes (8%) of dynamic memory, leaving 1873 bytes for local variables. Maximum is 2048 bytes.

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 Ausgangs vornehmen würden.

[AVR328] 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.

Compare Modus des AVR ^[AVR328] Seite 126

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

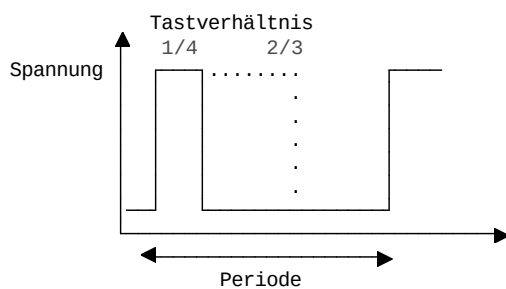
```
1 #ifndef F_CPU
2 #define F_CPU 16000000UL // 16 MHz clock speed
3 #endif
4
5 int main(void)
6 {
7     DDRB |= (1 << PB1); // Ausgabe LED festlegen
8     TCCR1A = 0;
9     TCCR1B = 0;
10    TCCR1B |= (1 << CS12) | (1 << CS10); // 1024 als Prescale-Wert
11    TCCR1A |= (1 << COM1A0);
12    OCR1A = 15625;
13
14    while (1) _delay_ms(500);
15 }
```

Sketch uses 194 bytes (0%) of program storage space. Maximum is 32256 bytes.

Global variables use 0 bytes (0%) of dynamic memory, leaving 2048 bytes for local variables. Maximum is 2048 bytes.

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 PWM 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);
        }
    }
}
```

[AVR328] Firma Microchip, megaAVR® Data Sheet, [Link](#)

Capture Mode

Capture Unit des AVR ^[AVR328] Seite 126

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      Filter
    TCCR1B = (1 << CS12) | (1 << CS10) | (1 << ICES1) | (1 << ICNC1);
    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

Zähler

70,000

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?

[AVR328] Firma Microchip, megaAVR® Data Sheet, [Link](#)

Anwendungen

Zähler von Aktivitäten

Capture Unit des AVR [\[AVR328\]](#) Seite 250

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();

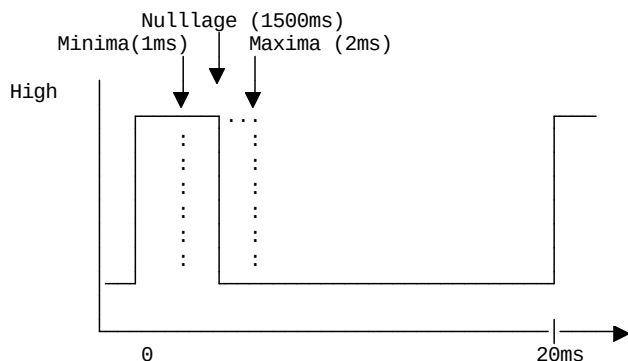
    while (1)
    {
        Serial.print(TCNT1);
        Serial.print(" ");
        _delay_ms(500);
    }
    return 0;
}
```

[AVR328] Firma Microchip, megaAVR® Data Sheet, [Link](#)

Servomotoren

Als Servomotor werden Elektromotoren bezeichnet, die die Kontrolle der Winkelposition ihrer Motorwelle sowie der Drehgeschwindigkeit und Beschleunigung erlauben. Sie integrieren neben dem eigentlichen Elektromotor, eine Sensorik zur Positionsbestimmung und eine Regelelektronik. Damit kann die Bewegung des Motors entsprechend einem oder mehreren einstellbaren Sollwerten – wie etwa Soll-Winkelposition der Welle oder Solldrehzahl – bestimmt werden.

Servomotor ^{[[wikimedia:Servo](#)]}



Diese Funktionalität lässt sich mit einem Timer entsprechend umsetzen.

$$20ms = 2500 \times 0.008ms$$

```
#define F_CPU 1000000UL
#include <avr/io.h>
#include <avr/interrupt.h>

ISR( TIMER1_COMPA_vect ){
    OCR1A = 2500-OCR1A;
}

int main (void){
    TCCR1A = (1<<COM1A0); // Togglen bei Compare Match
    TCCR1B = (1<<WGM12) | // CTC-Mode; Prescaler 8
            (1<<CS11);
    TIMSK = (1<<OCIE1A); // Timer-Compare Interrupt an
    OCR1A = 2312; // Neutralposition
    sei(); // Interrupts global an
    while( 1 ) {
        ...
        OCR1A = OCR1A + 3;
        _delay_ms(40);
        ...
    }
    return 0;
}
```

[[wikimedia:Servo](#)] Wikipedia, Autor Bernd vdB, Servo and receiver connections, [Link](#)

Gleichstrommotor

<https://www.tinkercad.com/things/lu1Gt48hNsl-gleichstrommotor-mit-encoder/editel>

Aufgaben

- ☐ Variieren Sie die Helligkeit der boardinternen LED mit dem zugehörigen Timer. Ermitteln Sie dessen Pin Belegung.
- ☐ Integrieren Sie PWM Funktionen in unsere Simulationsumgebung und evaluieren Sie diese. Stellen Sie die Impulse dar und Messen Sie die Genauigkeiten der Zeitvorgaben.