# Mikrocontroller

## Teil 2:

## Interrupt-Verarbeitung und Baugruppen

Autor: Dipl.-Ing. Edgar Hoch

(bearb.: Dipl.-Ing. J. Wemheuer, Stand: 03.2017)

# Literaturempfehlungen


- C.Kühnel Programmieren der AVR RISC Mikrocontroller mit BASCOM-AVR ISBN 3898119378 (2000) ISBN 3907857046 (2.Aufl.2004)ISBN 978-3-907857-14-4 (3. überarbeitete und erweiterte Auflage 2010)
- R.Mittermayr AVR-RISC: Embedded Software selbst entwickeln Franzis 2008 ISBN 3772341071
- F.Schäffer AVR: Hardware und C-Programmierung in der Praxis Elektor 2008 ISBN 3895762008
- G.Schmitt Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie...Oldenbourg 4.Aufl.2008 ISBN 3486587900 ISBN 3486580167 (2006) ISBN 3486577174 (2005)
- M.Schwabl-Schmidt Programmiertechniken für AVR-Mikrocontroller Elektor 2008 ISBN 3895761761
- M.Schwabl-Schmidt Systemprogrammierung für AVR-Mikrocontroller Elektor 2009 ISBN 3895762180
- W.Trampert Messen,Steuern und Regeln mit AVR Mikrocontrollern Franzis 2004 ISBN 3772342981
- W.Trampert AVR-RISC Mikrocontroller Franzis ISBN 3772354769 (2003) ISBN 3772354742 (2002) ISBN 3772354750 (2000)
- P.Urbanek Embedded Systems: Ein umfassendes Grundlagenwerk ... (2007) ISBN 3981123018
- S./F.Volpe AVR-Mikrocontroller-Praxis Elektor 2001 ISBN 3895760633
- R.Walter AVR-Mikrocontroller-Lehrbuch 3. Auflage Denkholz 2009 ISBN 9783981189445

# Einleitung

Im Teil 1 des Studienbriefes haben Sie die Grundlagen zu Mikroprozessoren und Mikrocontrollern kennen gelernt. In diesem zweiten Teil beschäftigen wir und mit der Interruptverarbeitung im Allgemeinen und mit den Baugruppen wie Analog-Digital-Umsetzer, Timer und Counter, UART/USART und Bussysteme im Besonderen.

Für die verschiedenen Steuerungsaufgaben sind diese Baugruppen von zentraler Bedeutung, wenn es darum geht, Signale von Sensoren zu verarbeiten und an Aktoren weiter zu geben. Vielfach liefern Sensoren noch analoge Signale, die erst in digitale Signale umgesetzt werden müssen, damit sie im Controller verarbeitet werden können. Müssen sehr viele Daten von Sensoren in einem Programm abgefragt werden, sind Kenntnisse der Interruptverarbeitung notwendig. Mit all diesen Themen werden Sie sich in diesem Studienbrief beschäftigen.

**Folgende Studienziele werden mit diesem Studienbrief erreicht:**

S

- Kenntnisse der Interruptverarbeitung in Mikrocontrollern
- Programmierung des Analog-Digital-Comperators
- Programmierung der UART-Schnittstelle
- Analoge Signalverarbeitung
- Umgang, Programmierung und Anwendung von Pulsweitenmodulation PWM
- Programmierung des Timers/Counters und Anwendung

# Inhaltsverzeichnis

# 1 Mikrocontroller – Interruptverarbeitung

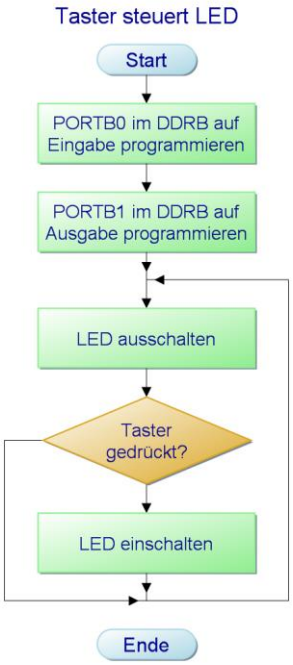



**Abbildung 1: Diese Art der Tastenabfrage wird auch "polling" genannt.**

In den bisherigen Programmentwicklungen wurden Sensoren wie zum Beispiel eine Taste durch eine Schleife abgefragt. Sie erinnern sich? Der Programmablaufplan zeigt diese Vorgehensweise noch einmal.

**Wozu Interruptverarbeitung?**

Diese Art der Abfrage, wie es im PAP gezeigt wird, wird auch „Polling“ genannt. Allerdings vergeudet der Mikrocontroller natürlich durch diese Art Abfragetechnik viel Zeit. Sinnvoller wäre es doch, nur dann einen bestimmten Programmteil abzuarbeiten, wenn dazu von einem Sensor auch Daten geliefert werden können.

Bei der Interrupt-Verarbeitung wird ein Signal nur dann eingelesen, wenn tatsächlich eine Änderung eines Signals vorliegt. Im Fall der Tastenabfrage wird in der Interrupt-Verarbeitung lediglich dann der Tastenwert eingelesen, wenn eine Signaländerung vorliegt. Solang das aber nicht der Fall ist, kann der Mikrocontroller andere Arbeiten ausführen. Hiermit wird die Prozessorleistung wesentlich besser ausgenutzt.

Und stellen Sie sich vor, dass ein Mikrocontroller in einem KFZ eingesetzt wird, in dem es viel Sensoren gibt, so würde der Mikrocontroller nur damit beschäftigt sein, die Sensoren abzufragen. Sinnvoller wäre es, die Signale der Sensoren mittels Interrupts zu verarbeiten.

Jede Interrupt-Steuerung beinhaltet Elemente, die immer vorkommen. So hat jede Interrupt-Steuerung eine

**Interruptquelle**

Damit ist die Ursache, der Auslöser des Interrupts gemeint. Auslöser für einen Interrupt kann beispielsweise ein externes Signal von einem Sensor sein – oder eine interne Baugruppe wie ein Analog-Digital-Umsetzer oder ein Timer.

Für jede Interruptquelle wird in einer Tabelle festgehalten, wo sich der Programmabschnitt befindet, der aufgrund des Interrupts ausgeführt werden soll. Diese Tabelle wird

**Interrupt Vektor Tabelle**

genannt. Der Programmteil, der aufgrund des Interrupts abgearbeitet wird, ist wie ein Unterprogramm aufgebaut. Dieser Programmteil wird

**Interrupt Service Routine**

genannt. Die Interrupt Service Routine endet – wie ein Unterprogramm – mit einem Return Befehl. Aber Vorsicht – für die ISR muss der Befehl **reti**, also „Return from Interrupt“ lauten.

## 1.1 Generelle Interrupt Befehle

Interrupts können generell zugelassen oder gesperrt werden. Das kann der Programmierer durch zwei Befehle steuern.

### 1.1.1 Interrupts zulassen mit sei

| *Befehl* | *Operand 1* | *Operand 2* |
|---|---|---|
| sei | kein | kein |
| Set Enable Interrupt<br>Interrupts generell zulassen | kein | kein |

Mit **sei** werden Interrupts generell zugelassen. Oder anders ausgedrückt: vergisst man diesen Befehl, werden Interrupts nie verarbeitet. Der Befehl setzt das Interrupt-Flag I im Statusregister SREG und signalisiert dem Mikrocontroller, dass Interrupts zugelassen sind.

### 1.1.2 Interrupts sperren mit cli

| *Befehl* | *Operand 1* | *Operand 2* |
|---|---|---|
| cli | kein | kein |
| Clear Interrupt<br>Interrupts generell sperren | kein | kein |

Gerade umgekehrt funktioniert der Befehl cli. Mit **cli** wird das Interrupt-Flag I im Statusregister SREG zurückgesetzt und Interrupts dadurch generell gesperrt. Sinnvoll ist dieser Befehl dann, wenn beispielsweise eine Interrupt Service Routine aufgerufen wurde und gleich zu Beginn dieses Programmteils ein erneuter Interrupt gesperrt werden soll.

Für die Kontrolle, die Aktivierung, Deaktivierung und die Ausführung bestimmter Interrupt-Service-Routinen sind diese Befehle einzusetzen.

## 1.2 Die Interrupt Vektor Tabelle

```
15 ;-------------------------------------------------------------------------
16 ;Reset and Interrupt vector              ;VNr.  Beschreibung
17             rjmp    main                ;1    POWER ON RESET
18             reti                        ;2    Int0-Interrupt
19             reti                        ;3    Int1-Interrupt
20             reti                        ;4    TC2 Compare Match
21             reti                        ;5    TC2 Overflow
22             reti                        ;6    TC1 Capture
23             reti                        ;7    TC1 Compare Match A
24             reti                        ;8    TC1 Compare Match B
25             reti                        ;9    TC1 Overflow
26             reti                        ;10   TC0 Overflow
27             reti                        ;11   SPI, STC Serial Transfer Complete
28             reti                        ;12   UART Rx Complete
29             reti                        ;13   UART Data Register Empty
30             reti                        ;14   UART Tx Complete
31             reti                        ;15   ADC Conversion Complete
32             reti                        ;16   EEPROM Ready
33             reti                        ;17   Analog Comparator
34             reti                        ;18   TWI (I²C) Serial Interface
35             reti                        ;19   Store Program Memory Ready
36 ;-------------------------------------------------------------------------
37 ;Start, Power ON, Reset
38 main:       ldi     r16,lo8(RAMEND)
39             out     SPL,r16
40             ldi     r16,hi8(RAMEND)
41             out     SPH,r16
42             ;Hier Init-Code eintragen.
```

**Abbildung 2: Die Interrupt Vektor Tabelle mit dem Interrupt-Vektor 1**

Diese Tabelle ist Bestandteil des „Grundgerüsts“, das wir bislang verwendet haben. Betrachten wir diese noch einmal etwas genauer:

Will man einen externen Interrupt zulassen und verarbeiten, dann muss in der Interrupt Vektor Tabelle der Befehl **reti** durch einen **rjmp**-Befehl zur Interrupt Service Routine ersetzt werden. Die Sprungmarke kann frei gewählt werden, es empfiehlt sich jedoch, eine Sprungmarke zu wählen, die „halbsprechend“ gewählt wird. Beispielsweise **onINT0** o.ä.

Für die in der Interrupt-Vektor-Tabelle gezeigten Interruptquellen wird bei allen AVR-Controllern ab der Adresse 0x001 im Programmspeicher (Flash) die Liste der Interrupt-Vektoren erwartet. Die Anzahl der Interruptquellen richtet sich nach dem Prozessortyp. Für alle Prozessoren ist jedoch gemeinsam, dass nach Einschalten der Betriebsspannung oder nach Betätigung des Reset-Signals die Adresse 0x0000 im Programmzähler angesteuert wird. Das Einschalten oder das Bedienen des Reset-Signals wird auch als Interrupt 0 bezeichnet. Wenn Sie in die Interrupt-Vektor-Tabelle sehen, erkennen Sie, dass der „erste Interrupt“ zur Interrupt-Service-Routine mit der Bezeichnung „main“ führt. Und diese Sprungmarke kennen Sie bereits. Hier beginnt jedes Hauptprogramm

Jeder Interrupt-Vektor muss einen Befehl enthalten, der ausgeführt wird, wenn ein Interrupt angefordert wird. Deshalb ist im Grundgerüst für jeden Interrupt-Vektor der Befehl RETI vorsorglich eingetragen. Dieser Befehl beendet den angeforderten Interrupt ordnungsgemäß. Wird ein Interrupt-Vektor vom Programmierer genutzt, wird der Befehl RETI durch einen Sprungbefehl RJMP zur jeweils verwendeten Interrupt-Sprungmarke ersetzt.

## 1.3 Interrupt INT0 und INT1 – externe Interrupts

Mit Interrupt-Anforderungen von extern angeschlossenen Sensoren werden wir die Behandlung von Interrupts beginnen. An zwei Stellen des Mikrocontrollers könnten externe Interrupt-Anforderungen an den Controller weitergeleitet werden. Diese werden im ATmega8 mit INT0 und INT1 bezeichnet und sind an den beiden PINs Nr4 und Nr5 mit den PORT-Pins D2 und D3 doppelt belegt:

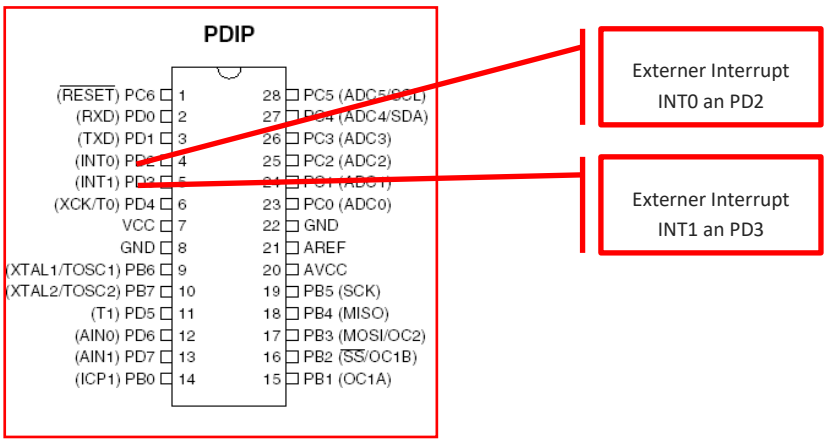



**Abbildung 3: Interrupt-Eingänge INT0 und INT1. Sie sind an den PORT-Pins D2 und D3 doppelt belegt.**

An diesen beiden Eingängen können Signale angeschlossen werden, die externe Interrupts auslösen. Im einfachsten Fall sind dies Schalter oder Taster, zum Beispiel Endschalter aus Steuerungen. Wir werden dies gleich an einem Bei-

spiel verdeutlichen. Zuvor kümmern wir uns aber um die Steuerregister, die das Handling bestimmen.

## 1.4 Steuerregister für die externen Interrupts

Jede Interrupt-Quelle wird durch ein oder mehrere spezifische Steuerregister konfiguriert. So gibt es auch für die Steuerung von externen Interrupt-Anforderungen Register, die das Handling steuern.

Externe Interrupts können einzeln zugelassen oder gesperrt werden. Man spricht auch von maskierbaren Interrupts oder Interrupt Maskierung. Für dieses zulassen oder sperren von Interrupts ist das Register GICR zuständig. Dabei ist es wichtig zu wissen, welche Bedeutung die einzelnen Bits des Registers haben. Aufschluss darüber gibt die Original-Beschreibung des atmega8. Hier ein Auszug daraus:

### 1.4.1 Das GICR – General Interrupt Control Register

**Moving Interrupts Between Application and Boot Space**

The General Interrupt Control Register controls the placement of the Interrupt Vector table.

**General Interrupt Control Register – GICR**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | INT1 | INT0 | – | – | – | – | IVSEL | IVCE | GICR |
| Read/Write | R/W | R/W | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| *Bit/Stelle* | *Bedeutung* | *Hinweis* |
|---|---|---|
| 7 | Interrupt INT1 wird durch 1-Signal aktiviert | Bit beim Einschalten = 0 |
| 6 | Interrupt INT0 wird durch 1-Signal aktiviert | Bit beim Einschalten = 0 |

Im Steuerregister GICR können externe Interrupt-Anforderungen gesperrt oder zugelassen werden. Ein weiteres Register bestimmt, wie ein Interrupt ausgelöst werden soll. Hier muss die auslösende Signalart beschrieben werden.

### 1.4.2 Das MCUCR – Micro Controller Unit Control Register

**MCU Control Register – MCUCR**

The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | SE | SM2 | SM1 | SM0 | ISC11 | ISC10 | ISC01 | ISC00 | MCUCR |
| 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 | |

Die Bedeutung der wichtigsten Einzelbits in diesem Register sind:

| *Bit/Stelle* | *Name* | *Bedeutung* | *Auslösung* |
|---|---|---|---|
| 3 | ISC11 | Flankenbestimmung für den INT1 | 00 fortlaufend<br>01 jede Änderung<br>10 fallende Flanke<br>11 steigende Flanke |
| 2 | ISC10 | | |
| 1 | ISC01 | Flankenbestimmung für den INT0 | |
| 0 | ISC00 | | |

Jeweils zwei Stellen (2/3 und 0/1) bestimmen die Auslöseart für den jeweiligen Interrupt. Werden die beiden Stellen mit zwei Nullen belegt, wird fortlaufend ein Interrupt angefordert, wird die Bitkombination 01 gewählt, führt jede Flankenänderung zum Interrupt, die Bitkombination 10 bedeutet, dass die fallende Flanke einen Interrupt auslöst und wird die Bitkombination 11 gewählt, so führt die aufsteigende Flanke die Interruptanforderung aus.

## 1.5 Praktische Anwendung

Sollen externe Interrupt-Anforderungen vom Mikrocontroller verarbeitet werden, sind also folgende Programmschritte notwendig:

1. In der Interrupt-Vektor-Tabelle RETI gegen RJMP mit der entsprechenden Sprungmarke ersetzen
2. Interrupt Maskierung im GICR einstellen
3. Interrupt Auslösung bestimmen im MCUCR
4. Interrupt generell zulassen durch sei

Die externen Interrupts werden an PORTD, Stelle 2 für INT0 und Stelle 3 für INT1 angeschlossen.

> **Noch ein wichtiger Hinweis:** die Bitmanipulationsbefehle können bei der Steuerung der Register GICR und MCUCR nicht verwendet werden. Sie liegen auf Adressen > 0x32. Bitmanipulationsbefehle können nur auf Register angewendet werden, die auf Adressen unterhalb 0x32 liegen.

Ein einfaches Beispiel veranschaulicht den Vorgang der Interrupt-Verarbeitung.

**Beispiel:**

Ein Interrupt INT0 soll durch einen Taster ausgelöst werden. In der Interrupt-Service-Routine soll dann eine Leuchtdiode angesteuert werden. Das Hauptprogramm entfällt in diesem Falle völlig. Lediglich main: muss noch bedient werden.

```
;+---------------------------------------------------------------------
;| Title            : Interruptverarbeitung
;+---------------------------------------------------------------------
;| Funktion         : ...
;| Schaltung        : ...
;+---------------------------------------------------------------------
;| Prozessor        : ATmega8
;| Takt             : 3,6864 MHz
;| Sprache          : Assembler
;| Datum            : 1.7.2011
;| Version          : ...
;| Autor            : Edgar Hoch
;+---------------------------------------------------------------------
.include    "AVR.H"
;---------------------------------------------------------------------
;Reset and Interrupt vector          ;VNr.  Beschreibung
            rjmp    main            ;1    POWER ON RESET
            rjmp    onint0          ;2    Int0-Interrupt
            reti                    ;3    Int1-Interrupt
            reti                    ;4    TC2 Compare Match
            reti                    ;5    TC2 Overflow
            reti                    ;6    TC1 Capture
            reti                    ;7    TC1 Compare Match A
            reti                    ;8    TC1 Compare Match B
            reti                    ;9    TC1 Overflow
            reti                    ;10   TC0 Overflow
            reti                    ;11   SPI, STC Serial Tr
            reti                    ;12   UART Rx Complete
            reti                    ;13   UART Data Register Empty
            reti                    ;14   UART Tx Complete
```

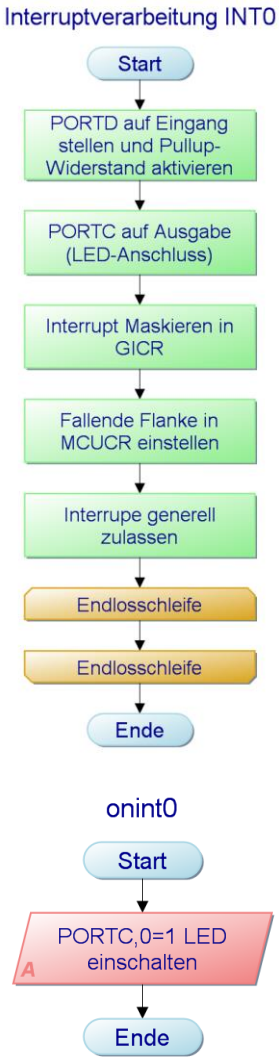



**Abbildung 4:**
**Die Interrupt-Verarbeitung, dargestellt in zwei Programmablaufplänen**

```
                reti                    ;15  ADC Conversion Complete
                reti                    ;16  EEPROM Ready
                reti                    ;17  Analog Comparator
                reti                    ;18  TWI
                reti                    ;19  Store Program Memory Ready
;-----------------------------------------------------------------

main:           ldi     r16,lo8(RAMEND)
                out     SPL,r16
                ldi     r16,hi8(RAMEND)
                out     SPH,r16
                cbi     DDRD,2              ;Datenrichtung an D2 Eingabe
                sbi     PORTD,2             ;Pullup-Widerstand für D2
                sbi     DDRC,0              ;Datenrichtung an C0 Ausgabe
                ldi     R16,0b01000000      ;Bit 6 aktiviert den INT0
                out     GICR, R16           ;Übergabe in GICR
                ldi     R16,0b00000010      ;Fallende Flanke wählen
                out     MCUCR,R16           ;und Übergabe an MCUCR
                sei                         ;Interrupt generell erlauben

;-----------------------------------------------------------------
mainloop:       wdr
                rjmp    mainloop
;-----------------------------------------------------------------

onint0:         sbi     PORTC,0             ;LED wird aktiviert
                reti
;-----------------------------------------------------------------
```

**Beschreibung der Einstellungen:**

In der Interrupt-Vektor-Tabelle wird in der Zeile, in der der Interrupt Vektor2 definiert wird, der **reti**-Befehl durch einen **rjmp** onint0 ersetzt.

In main wird zunächst die Datenflussrichtung im DDRD, Stelle 2 eingestellt. Diese muss auf Dateneingabe gestellt werden und anschließend der Pullup-Widerstand gesetzt werden.

Die nächsten Befehle bedienen die Register GICR und MCUCR. Im GICR wird der externe Interrupt aktiviert; im MCUCR wird die fallende Flanke als Auslöser an INT0 bestimmt. Das ist auch sinnvoll, weil ein offener Taster durch den Pullup-Widerstand als 1-Signal interpretiert wird. Wird nun die Taste betätigt, wird dadurch ein 0-Signal geliefert. Das bedeutet eine abfallende Flanke bei Tastendruck.

Zum Schluss der Definitionen müssen durch den Befehl **sei** Interrupts generell zugelassen werden.

In der mainloop geschieht eigentlich nichts. Hier wird durch den Befehl rjmp mainloop lediglich eine Endlosschleife durchlaufen.

Die Interrupt-Service-Routine beginnt bei der Sprungmarke onint0:. Der **sbi**-Befehl aktiviert die LED. Danach erfolgt der Rücksprung ins Hauptprogramm durch **reti**.

### 1.5.1 Übung

Eine Taste soll den Motor eines Fensterhebers im KFZ steuern. Der Motor läuft so lange, bis ein zweites Signal eintrifft, das vom Endschalter geliefert wird.

Die beiden Signale werden von den Tastern des Entwicklungsboards geliefert und für die beiden Interrupts INT0 und INT1 verwendet.

Ü

Eine Leuchtdiode soll den Motor simulieren. Die LED soll an PORTC, Stelle 0 angeschlossen werden.

### 1.5.2 Zusammenfassung

- Durch externe Interrupts kann ein Hauptprogramm unterbrochen und eine Interrupt-Service-Routine abgearbeitet werden. Der Mikrocontroller merkt sich die Stelle im Hauptprogramm, an der er unterbrochen wurde, und arbeitet nach Abarbeitung der Interrupt-Service-Routine an genau dieser Stelle wieder weiter.
- Der Rücksprung aus der Interrupt-Service-Routine erfolgt durch den Befehl **reti**.
- Externe Interrupts werden im Steuerregister GICR aktiviert. Die Interrupts werden mit INT0 und INT1 bezeichnet.
- Die Signalformen für die Auslösung der Interrupts werden im Steuerregister MCUCR definiert.
- Interrupts müssen durch den Befehl **sei** generell zugelassen werden. Generell ist der Microcontroller auf **cli** (clear interrupt) gesetzt.
- Die Sprungmarken zu den Interrupt-Service-Routinen müssen in der Interrupt-Vektor-Tabelle eingetragen werden.

Z

A

### 1.5.3 Aufgaben

Erklären Sie kurz den Ablauf für die Programmierung eines externen Interrupts INT1 und die zugehörigen Befehle.

Im folgenden Programm sollen zwei Tasten zwei Interrupt-Service-Routinen auslösen. Allerdings scheinen die Tasten recht „unkontrolliert“ zu funktionieren. Erklären Sie den Fehler im folgenden Programm.

*Assemblerprogramm zu Aufgabe 1.5.3*

```
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            cbi     DDRD,2              ;Datenrichtung an D2 Eingabe
            sbi     PORTD,2             ;Pullup-Widerstand für D2
            cbi     DDRD,3              ;Datenrichtung an D3 Eingabe
            sbi     DDRC,0              ;Datenrichtung an C0 Ausgabe
            ldi     R16,0b11000000      ;Bit 6/7 aktivieren INT0/INT1
            out     GICR, R16           ;Übergabe in GICR
            ldi     R16,0b00001010      ;10 aktiviertabfallende Flanken
            out     MCUCR,R16           ;und Übergabe an MCUCR
            sei                         ;Interrupt generell erlauben
;-------------------------------------------------------------------------

mainloop:   wdr
            ;Hier den Quellcode eintragen.
            rjmp    mainloop
;-------------------------------------------------------------------------

onint0:     sbi     PORTC,0             ;LED wird aktiviert
            reti
;-------------------------------------------------------------------------

onint1:     cbi     PORTC,0             ;LED wird deaktiviert
            reti
;-------------------------------------------------------------------------
```

# 2 Mikrocontroller - Serielle Datenübertragung

In der Steuerungstechnik ist es außerordentlich wichtig, Daten von und zu externen Baugruppen aber auch zu anderen Controllersystemen mit möglichst wenig Leitungsaufwand zu übertragen. In KFZ-Systemen ist dies besonders leicht einzusehen, wenn man daran denkt, dass verschiedene Systeme miteinander kommunizieren müssen. In diesem Fall müssen die Daten seriell aufbereitet und übertragen werden.

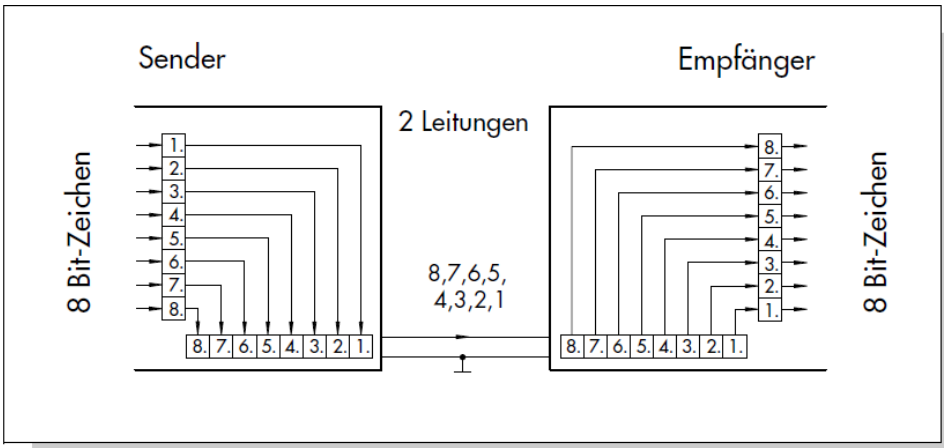



Abbildung 5: Prinzip der seriellen Datenübertragung

Daten liegen in Mikrocontrollersystemen meist in 8 Bit Parallelstruktur vor. Beispielsweise, wenn man an den Inhalt eines Registers oder eines Speicherplatzes denkt. Die Umsetzung der Paralleldaten in Serielldaten übernimmt ein Baustein, der UART oder USART genannt wird. Ausgeschrieben: **U**niversal **S**erial **A**synchronous **R**eceiver and **T**ransmitter.

Diese Baugruppe ist im ATmega8 integriert und kann per Programm aktiviert werden. Nach außen sind die beiden Anschlüsse RXD und TXD die Verbindungsstellen zu anderen Systemen.

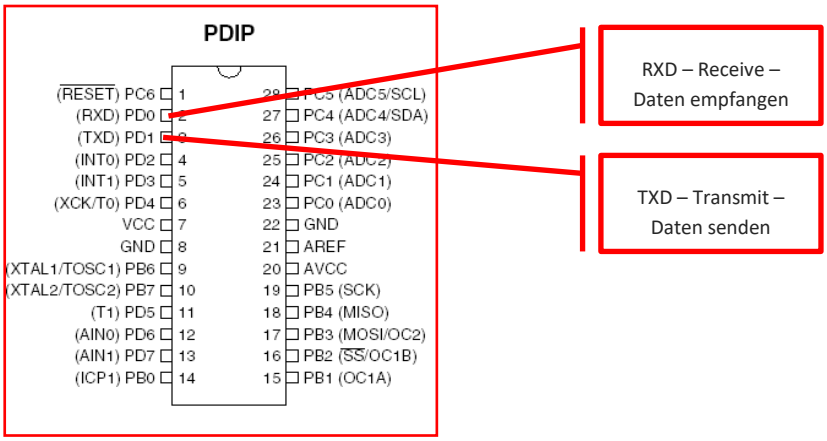



Abbildung 6: TXD ist der serielle Datenausgang und wird an PD1 gesendet. RXD ist der Dateneingang und wird an PD0 empfangen.

## 2.1 USART Aufbau und Funktion

Die USART-Baugruppe im ATmega8 übernimmt die Umsetzung der parallel vorliegenden Daten in serielle Strukturen inklusive der Anreicherung der seriellen Daten mit Steuersignalen.

Der USART-Baustein arbeitet asynchron, das bedeutet, dass keine Synchronisation zwischen Sender und Empfänger stattfindet. Das wiederum bedeutet, dass beide Baugruppen mit exakt der gleichen Übertragungsgeschwindigkeit Daten senden und empfangen müssen. Diese Übertragungsgeschwindigkeit heißt BAUD-Rate.

**Kurze Erklärung:** Die BAUD-Rate ist die Einheit für die Schrittgeschwindigkeit in der Nachrichtentechnik und Fernmeldetechnik. 1 Baud ist die Geschwindigkeit, wenn 1 Symbol pro Sekunde übertragen wird. Jedes Symbol entspricht einer definierten messbaren Signaländerung im physischen Übertragungsmedium. In der Mikrocontrollertechnik spricht man auch von einer Übertragungsgeschwindigkeit von 1 bps = 1 BAUD -> also 1 Bit pro Sekunde als Grundgröße.

Die Baudrate einer Datenübertragung muss auf Sende- und Empfangsseite gleich sein. Außerdem müssen Beginn und Ende der Kommunikation durch Start- und Ende-Signale gekennzeichnet werden (Start- und Stop-Bit).

### 2.1.1 Blockdiagramm zum USART

Der USART des ATmega8 besteht aus drei Baugruppen. Die erste Baugruppe ist der BAUD RATE GENERATOR.

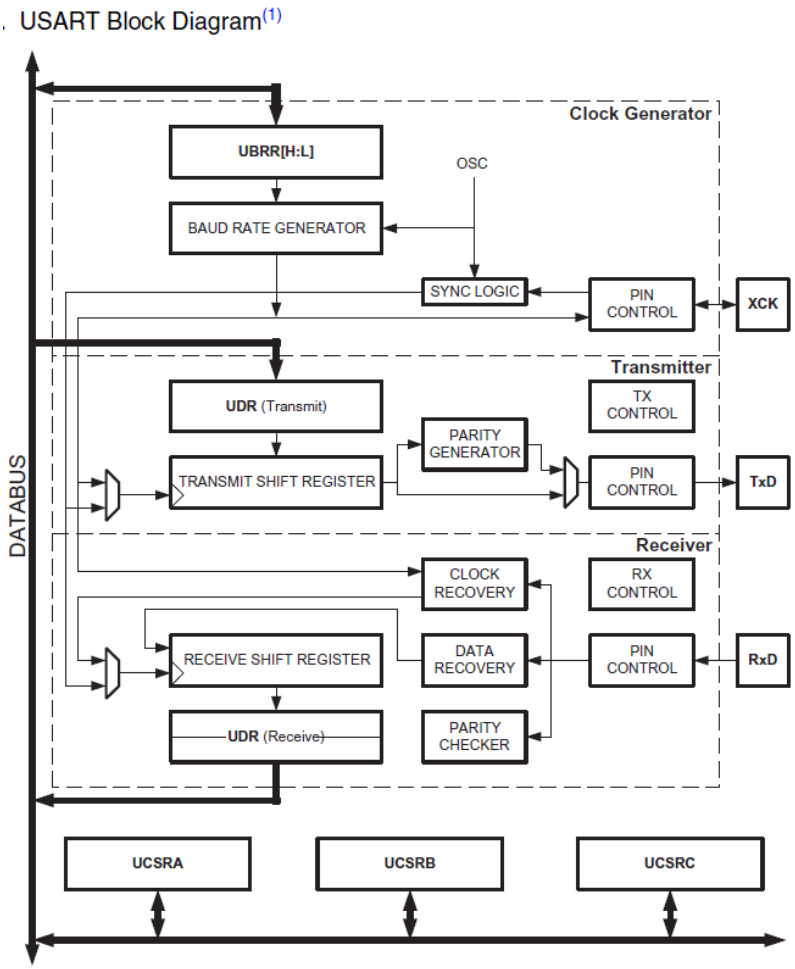



**Abbildung 7: Das USART-Blockdiagramm zeigt den Taktgenerator, den Empfangsteil und den Sendeteil**

Eine übliche Datenübertragungsgeschwindigkeit ist eine Baudrate von 9600 Baud. Diese Baudrate wird aus dem verwendeten Systemtakt generiert und

kann vom Anwender im Steuerregister UBRR eingestellt werden. Das UBRR ist ein 16-Bit-Register und wird in die 8-Bit-Anteile UBRRL und UBRRH (Low Byte / High Byte) aufgeteilt.

Für die Berechnung des Registerwertes gilt:

$$BAUD = \frac{Systemfrequenz}{16(UBRR+1)} \text{ oder umgeformt } UBRR = \frac{Systemfrequenz}{16 \cdot BAUD} - 1$$

Will man mit einer Systemfrequenz von 3,6864 MHz arbeiten und eine Baudrate von 9600 bps für die Datenübertragung nutzen, wird der Registerwert für UBRR wie folgt berechnet

$$UBRR = \frac{3686400}{16 \cdot 9600} - 1 = 23$$

Der Hersteller gibt folgende Tabelle mit Fehlerquoten und möglichen Datenübertragungsgeschwindigkeiten an:

| Baud Rate (bps) | $f_{osc}$ = 3.6864MHz | | | | $f_{osc}$ = 4.0000MHz | | | | $f_{osc}$ = 7.3728MHz | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | U2X = 0 | | U2X = 1 | | U2X = 0 | | U2X = 1 | | U2X = 0 | | U2X = 1 | |
| | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error | UBRR | Error |
| 2400 | 95 | 0.0% | 191 | 0.0% | 103 | 0.2% | 207 | 0.2% | 191 | 0.0% | 383 | 0.0% |
| 4800 | 47 | 0.0% | 95 | 0.0% | 51 | 0.2% | 103 | 0.2% | 95 | 0.0% | 191 | 0.0% |
| 9600 | 23 | 0.0% | 47 | 0.0% | 25 | 0.2% | 51 | 0.2% | 47 | 0.0% | 95 | 0.0% |
| 14.4k | 15 | 0.0% | 31 | 0.0% | 16 | 2.1% | 34 | -0.8% | 31 | 0.0% | 63 | 0.0% |
| 19.2k | 11 | 0.0% | 23 | 0.0% | 12 | 0.2% | 25 | 0.2% | 23 | 0.0% | 47 | 0.0% |
| 28.8k | 7 | 0.0% | 15 | 0.0% | 8 | -3.5% | 16 | 2.1% | 15 | 0.0% | 31 | 0.0% |
| 38.4k | 5 | 0.0% | 11 | 0.0% | 6 | -7.0% | 12 | 0.2% | 11 | 0.0% | 23 | 0.0% |
| 57.6k | 3 | 0.0% | 7 | 0.0% | 3 | 8.5% | 8 | -3.5% | 7 | 0.0% | 15 | 0.0% |
| 76.8k | 2 | 0.0% | 5 | 0.0% | 2 | 8.5% | 6 | -7.0% | 5 | 0.0% | 11 | 0.0% |
| 115.2k | 1 | 0.0% | 3 | 0.0% | 1 | 8.5% | 3 | 8.5% | 3 | 0.0% | 7 | 0.0% |
| 230.4k | 0 | 0.0% | 1 | 0.0% | 0 | 8.5% | 1 | 8.5% | 1 | 0.0% | 3 | 0.0% |
| 250k | 0 | -7.8% | 1 | -7.8% | 0 | 0.0% | 1 | 0.0% | 1 | -7.8% | 3 | -7.8% |
| 0.5M | – | – | 0 | -7.8% | – | – | 0 | 0.0% | 0 | -7.8% | 1 | -7.8% |
| 1M | – | – | – | – | – | – | – | – | – | – | 0 | -7.8% |
| Max (1) | 230.4kbps | | 460.8kbps | | 250kbps | | 0.5Mbps | | 460.8kbps | | 921.6kbps | |

**Abbildung 8: Mögliche Baudraten bei unterschiedlichen Taktfrequenzen**

Theoretisch könnte man also bei einer Systemfrequenz von 3,6864 MHz eine Datenübertragungsgeschwindigkeit von 230 kbps erreichen.

### 2.1.2 Der Transmitter (Sender)

In diesem Teil des USART wird die parallele Datenstruktur in eine serielle Struktur umgewandelt. Die Paralleldaten liegen im Register mit der Bezeichnung **UDR** vor und werden im Shiftregister in die serielle Struktur umgesetzt.

Das Senden der seriellen Datenstruktur erfolgt so, wie es das Signal-Zeit-Diagramm zeigt:

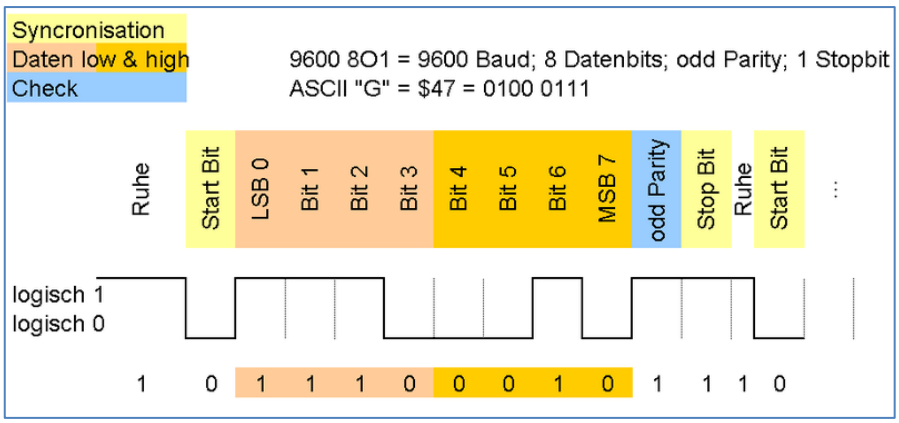



Abbildung 9: Übertragungsschema mit Start- und Stop-Signal sowie Parity-Bit

Der Beginn der Übertragung wird durch ein Start-Bit gekennzeichnet, das Ende der 8 Bits signalisiert das Stop-Bit. Die Bits 0 bis 7 werden gesendet und im Parity-Generator überprüft.

Die Parity-Prüfung ist eine Prüfung, ob die Anzahl der 1-Signale eine gerade oder eine ungerade Zahl ist. Mit Hilfe dieses Bits können Übertragungsfehler vom Empfänger erkannt und ggf. selbst „repariert“ werden.

### 2.1.3 Der Receiver (Empfänger)

Grundsätzlich funktioniert der Empfangsteil nach genau dem gleichen Prinzip wie der Sendeteil. Auch das Register **UDR** wird beim Datenempfang wieder genutzt.

Die seriellen Daten werden zunächst auf Fehler geprüft und ggf. repariert. Danach werden die Daten an das Shiftregister übergeben und in eine Parallelstruktur gebracht. Das Ergebnis der Umwandlung wird im **UDR** gespeichert.

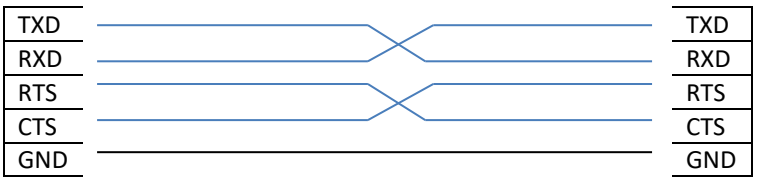
### 2.1.4 Verbindung über Nullmodemkabel

Das einfachste Verbindungskabel benötigt nur drei Leitungen: RxD, TxD und GND. Für Datenübertragungen zwischen "gleichen" Geräten müssen die Datenleitungen (RxD und TxD) natürlich gekreuzt werden. Derartige Kabel werden "Nullmodemkabel" genannt.

Soll der Datenfluss ebenfalls über diese Leitungen gesteuert werden, benötigt man zwei Codes, welche die Empfangs- bzw. Sendebereitschaft signalisieren:

CTS: clear to send
RTS: request to send

Braucht man dagegen einen Hardware-Handshake, so müssen diese Signale als eigene Steuerleitungen realisiert werden. Dazu müssen mindestens RTS und CTS verbunden werden, ebenfalls gekreuzt. Für die bidirektionale Datenübertragung zwischen zwei Geräten gilt somit folgendes Anschlussschema:



### 2.1.5 Protokoll einer Datenübertragung

Zu der reinen „Hardware-Verbindung“ zweier Geräte gehört auch eine Vereinbarung, wie die Daten übertragen werden sollen. Wir sprechen hierbei von einem Protokoll. Das Protokoll für eine klassische Seriellschnittstelle ist das RS232-Protokoll. Dieses wird vom Mikrocontroller voll unterstützt.

Das in diesem Studienbrief verwendete Entwicklungsboard nutzt die USB-Schnittstelle für die serielle Datenübertragung. Bei den Überlegungen zur Programmierung und Datenübertragung brauchen Sie das nicht berücksichtigen.

## 2.2 Programmierung des USART

Die USART-Schnittstelle muss zunächst initialisiert werden. Die Datenübertragungsgeschwindigkeit muss eingestellt werden. Das geschieht im USART-Bauraten-Register UBRR (UBRRL + UBRRH). Die Berechnung für eine Übertragungsrate von 9600 Baud haben Sie bereits kennen gelernt. In das Register UBRRL muss demnach die Dezimalzahl 23 übergeben werden.

Das Register kann durch den Befehl out angesprochen werden. Da der out-Befehl nur durch ein Register erfolgen kann, wird hierfür das Register R16 verwendet.

```
ldi     R16,23          ;Konstante für Baudrate 9600
out     UBRRL,R16       ;Übergabe an den Baudratengenerator
```

Damit ist die Übertragungsgeschwindigkeit festgelegt. Eine weitere Einstellung muss im USART-Control-Register vorgenommen werden. Das Register hat die Bezeichnung UCSRB und ist wie folgt aufgebaut:

| *USART Control and Status Register B – UCSRB* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ***Bit*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ***Name*** | RXCIE | TXCIE | UDRIE | RXEN | TXEN | UCSZ2 | RXB8 | TXB8 |
| ***R/W*** | R/W | R/W | R/W | R/W | R/W | R/W | R | R/W |
| ***Initial*** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

| ***Bit/Stelle*** | ***Bedeutung im UCSRB*** | ***Name*** | ***Beeinfl.*** |
|---|---|---|---|
| 7 | Complete Interrupt Enable, Interruptauslösung, wenn ein Byte komplett empfangen wurde | RXCIE | RX |
| 6 | Complete Interrupt Enable, Interruptauslösung, wenn ein Byte komplett gesendet wurde | TXCIE | TX |
| 5 | Data Register Empty Interrupt Enable, signalisiert, wenn das Datenregister UDR leer ist also die Daten gesendet wurden | UDRIE | |
| 4 | Receiver Enable, hiermit wird der Empfänger eingeschaltet | RXEN | |
| 3 | Transmitter Enable, hiermit wird der Sender eingeschaltet | TXEN | |
| 2 | Einstellen der Datenbreite 0=8Bit | | |
| 1 | Einstellen der Datenbreite für Empfang 0=8Bit | RXB8 | |
| 0 | Einstellen der Datenbreite für Senden 0=8Bit | TXB8 | |

Der USART kann im Polling oder durch Interrupt-Verarbeitung benutzt werden. Verzichtet man vorerst auf die Interrupt-Verarbeitung, dann reicht es, wenn die Bits 4 und 5 eingestellt werden. Mit diesen beiden Bits werden Sender und Empfänger getrennt ein-/ oder ausgeschaltet.

### 2.2.1 Aktivieren des USART

Soll der USART – genauer gesagt: Sender und der Empfänger – eingeschaltet werden, kann das mit Bitmanipulationsbefehlen erfolgen. Die Stellen Bit 3 und Bit 4 müssen dazu mit 1 belegt werden.

```
sbi     UCSRB,3             ;Transmitter einschalten
sbi     UCSRB,4             ;Receiver einschalten
```

### 2.2.2 Daten empfangen

Die Steuerung des Datenempfangs erfolgt im Register UCSRA (USART-Status-Register). Die Daten selbst werden nach der kompletten Übertragung im Register UDR für die weitere Verarbeitung bereitgehalten.

| *USART Control and Status Register A – UCSRA* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ***Bit*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ***Name*** | RXC | TXC | UDRE | FE | DOR | PE | U2X | MPCM |
| ***R/W*** | R | R/W | R | R | R | R | R/W | R/W |
| ***Initial*** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

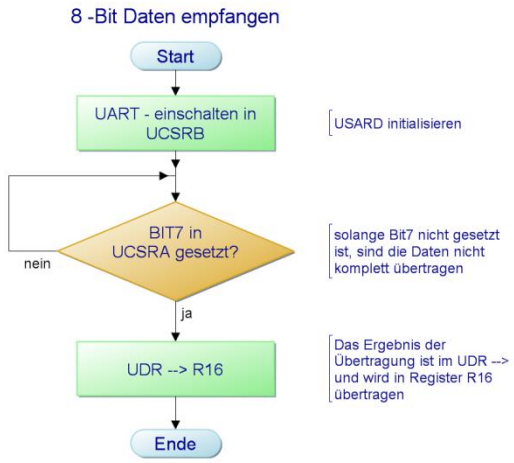
| *USART I/O Data Register – UDR* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ***Bit*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ***Name*** | RXB (UDR Read) | | | | | | | |
| ***Name*** | TXB (UDR Write) | | | | | | | |
| ***R/W*** | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ***Initial*** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

| ***Bit/Stelle*** | ***Bedeutung im UCSRA*** | ***Bezeichnung*** | ***Beeinfl.*** |
|---|---|---|---|
| 7 | Receive complete, 1=Daten werden empfangen, 0=Datenempfang komplett, UDR ist „gefüllt“ | RXC | |
| 6 | Transmit complete, 1=Daten werden gesendet, 0=Datensendung komplett, UDR ist „leer“ | TXC | |
| 5 | Data Register UDR Empty, Datenregister ist leer | UDRE | |
| 4 | Frame Error – Fehler bei Datenrahmen | FE | |
| 3 | Data overrun | DOR | |
| 2 | Parity error – Fehler beim Paritätskennung | PE | |
| 1 | Doppelte Datenübertragungsrate | U2X | |
| 0 | Multiprozessor communication mode, Mehrprozessorbetrieb | MPCM | |

Für einfache Datenübertragungen sind Bit 7 und Bit 6 von Interesse. Sie signalisieren, wenn eine Datenübertragung fertig ist. Man kann auch das Bit 5 zur Erkennung des Endes einer Übertragung benutzen. Es zeigt an, ob das Datenregister leer ist oder nicht. Ist das Datenregister beim Senden leer, ist der Prozess beendet.

Die Funktionen des USART sind recht umfangreich und bieten dem Anwender auch das Eingreifen in den Übertragungsprozess an, wenn sich Fehler ergeben. Für uns sind momentan die Bits 7,6 und 5 für die Prozesssteuerung ausreichend.

Eine kleine Programmroutine kann den Datenempfang umsetzen:



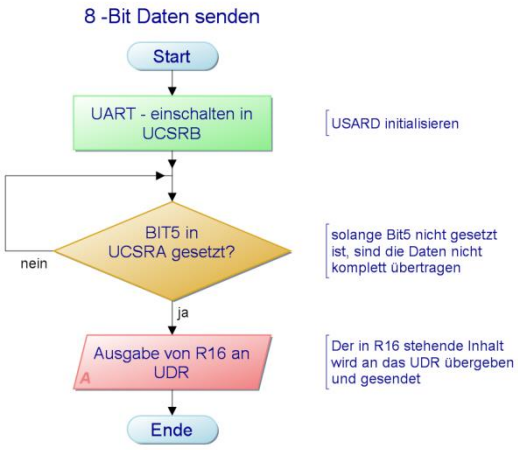

Die entsprechenden Programmzeilen sehen dann wie folgt aus:

```
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            ldi     R16,23          ;Konstante für Baudrate 9600
            out     UBRRL,R16       ;Übergabe an den Baudratengenerator
            sbi     UCSRB,3         ;Transmitter einschalten
            sbi     UCSRB,4         ;Receiver einschalten
;-----------------------------------------------------------------
mainloop:   wdr
            sbis    UCSRA,7         ;Warten, bis Ergebnis vollständig
            rjmp    mainloop
            in      R16, UDR           ;Ergebnis nach R16
Ende:       rjmp    Ende
;-----------------------------------------------------------------
```

Das Ergebnis der Datenübertragung ist im Register R16. Die Warteschleife wird durch einen sbis UCSRA, 7 Befehl realisiert, der immer wieder zum Programmanfang verzweigt, solange das Ergebnis nicht vollständig ist.

### 2.2.3 Daten Senden

Der Vorgang gleicht dem des Datenempfangs. Zuerst muss der USARD initialisiert werden und senden aktiviert werden. Das geschieht wieder im Steuerregister UCSRB. Der zu übertragene Wert wird in ein Register geladen und dann an das UDR-Register übergeben. Im Register UDSRA wird dann das Ergebnis der Übertragung überwacht, indem das Bit 5 abgefragt wird. Ist dieses Bit =1, dann ist das Register UDR bereit neue Daten aufzunehmen. Hier hätte man auch da Bit 6 zur Abfrage heranziehen können. Dieses Bit signalisiert, dass die Übertragung fertig ist.



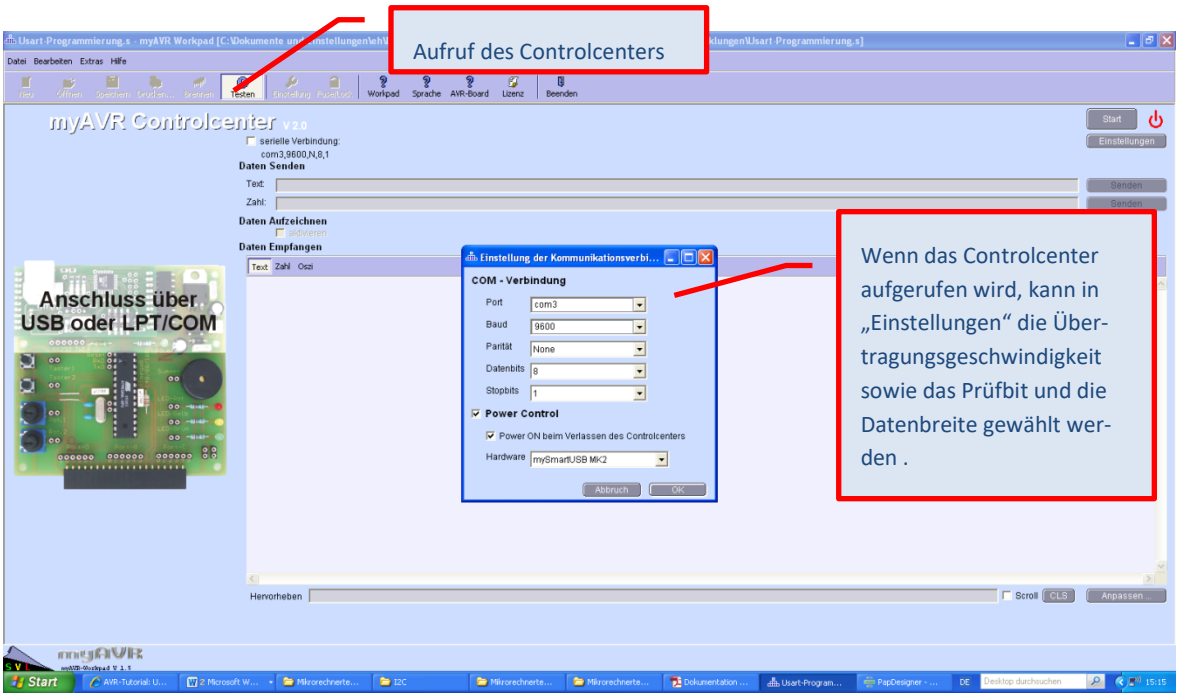

Die entsprechenden Assemblerprogrammzeilen sehen wie folgt aus:

```
mainloop:   wdr
            sbis    UCSRA,5          ;Warten, bis Ergebnis vollständig
            rjmp    mainloop
            out     UDR,R16          ;Inhalt von R16 senden
Ende:       rjmp    Ende
```

### 2.2.4 Senden einer Zeichenfolge mit dem Entwicklungsboard

Wie schon erwähnt, bietet das Entwicklungsboard die Möglichkeit, über die USB-Schnittstelle Daten zum PC zu senden oder vom PC zum Entwicklungsboard zu empfangen.

Dazu bietet das myavr-Workpad die Funktion Controlcenter bzw. in der Menüleiste die Funktion „Testen“.




Das Controlcenter bietet die Möglichkeit, Zahlen oder Text zu übertragen. Empfang und Senden der Daten erfolgt mit dem Startknopf rechts oben!

Sollen Daten vom AVR an den PC übergeben werden, erfolgt dies in der Regel als ASCII-Zeichen. Ein ASCII-Zeichen wird in Hochkommas im Assembler-Quellcode geschrieben.

**Ein kleines Programm erklärt die Vorgehensweise**

Sollen Daten an den USART übergeben und gesendet werden, müssen diese Byte für Byte an das Register UDR übergeben werden. Man kann nun Byte für Byte in einem Register die Übertragungswerte generieren. Für längere Texte ist diese Methode allerdings recht umständlich. Wir werden bei der indirekten Adressierung von Speicherplätzen noch einmal darauf eingehen.

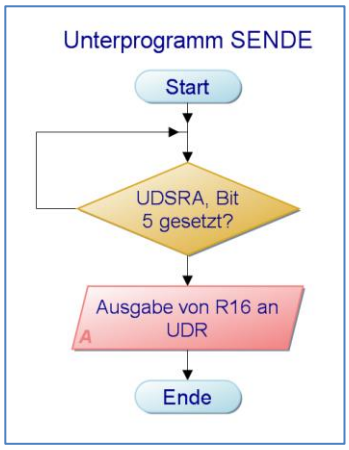



**Abbildung 11: Das Unterprogramm SENDE überträgt je Aufruf ein Byte.**

Für kurze Texte kann diese Methode durchaus verwendet werden. Will man beispielsweise den Text „oben“ ausgeben, dann muss dies durch einzelne Ladebefehle Byte für Byte generiert und an den USART übergeben werden. Nach jedem Byte müsste das Übertragungsprogramm geschrieben werden. Das realisiert man jedoch besser, wenn man das Übertragungsprogramm als Unterprogramm schreibt.

Die entsprechenden Assemblerzeilen zeigen die Umsetzung:

**Abbildung 12: Jeder Buchstabe muss geladen und gesendet werden. Das Senden übernimmt das Unterprogramm.**

```
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            ldi     R16,23              ;Konstante für Baudrate 9600
            out     UBRRL,R16           ;Übergabe an UBRRL
            sbi     UCSRB,3             ;Transmitter einschalten
            sbi     UCSRB,4             ;Receiver einschalten

;-----------------------------------------------------------------
mainloop:
            wdr
            ldi     r16,'O'             ;Wert für O ins Register R16
            rcall   SENDE               ;Unterprogramm SENDE aufrufen
            ldi     R16,'B'             ;Wert für B ins Register R16
            rcall   SENDE               ;Unterprogramm SENDE aufrufen
            ldi     R16,'E'             ;Wert für E ins Register R16
            rcall   SENDE               ;Unterprogramm SENDE aufrufen
            ldi     R16,'N'             ;Wert für N ins Register R16
            rcall   SENDE               ;Unterprogramm SENDE aufrufen
ENDE:       rjmp    ENDE                ;Endlosschleife
            rjmp    mainloop
;-----------------------------------------------------------------
SENDE:      sbis    UCSRA,5             ;Sprung, wenn Bit 5 =1
            rjmp    SENDE
            out     UDR, R16            ;Ausgabe von R16 an Sendereg.
            ret
;-----------------------------------------------------------------
```

Die Intitialisierung des USART beginnt wieder mit der Einstellung für die Datenübertragungsgeschwindigkeit und das Einschalten des Senders.

Im Hauptprogramm werden dann die einzelnen Bytes des Wortes „OBEN“ generiert und nach jedem Buchstaben, das Unterprogramm SENDEN aufgerufen, das die Werte an den USART übergibt.

Mit diesem Wissen können Sie nun eine kleine Übung selbst programmieren.

Ü

### 2.2.5 Übung

Das Programm aus der Übung 1 soll verbessert werden. Und zwar so, dass durch die Taste 1 – diese sollte den Motor einschalten – der Text „Motor ein“ ausgegeben wird. Wird die Taste 2 betätigt, soll der Text „Motor aus“ ausgegeben werden, als Zeichen dafür, dass der Endschalter den Motorstrom wieder ausgeschaltet hat.

### 2.2.6 Empfangen einer Zeichenfolge vom PC

Mit Hilfe der Funktion „Test“ des Workpads soll nun noch die Empfangsfunktion ausprobiert und programmiert werden.

Um die Empfangsfunktion ausprobieren zu können, muss das empfangene Byte gleich wieder an das Workpad zurück gesendet werden. Sonst kann die Funktion des Programms nicht überprüft werden.

Das Programm soll also ein Byte vom PC im USART empfangen und dieser sendet das empfangene Byte sofort wieder an den PC zurück.

Das Assemblerprogramm (ohne Grundgerüst) sieht nun so aus:

```
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            ldi     R16,23              ;Konstante für Baudrate 9600
            out     UBRRL,R16           ;Übergabe an Baudratengenerator
            sbi     UCSRB,3             ;Transmitter einschalten
            sbi     UCSRB,4             ;Receiver einschalten
;-----------------------------------------------------------------------
mainloop:
            wdr
            rcall   EMPFANGE
            rcall   SENDE
            rjmp    mainloop
;-----------------------------------------------------------------------
SENDE:      sbis    UCSRA,5             ;Sprung, wenn Bit 5 =1
            rjmp    SENDE
            out     UDR,R16             ;Ausgabe von R16 an Senderegister
            ret
;-----------------------------------------------------------------------
EMPFANGE:   sbis    UCSRA,7             ;Sprung, wenn Bit 7 =1
            rjmp    EMPFANGE
            in      R16,UDR             ;empfangenes Zeichen in R16
            ret
;-----------------------------------------------------------------------
```

Die gesendeten Daten werden mit diesem Programm sofort wieder an den Sender zurückgegeben und dargestellt. Das Senden übernimmt die Funktion „Test“ im Workpad.

## 2.3 Weitere serielle Datenübertragungsmöglichkeiten

Neben der seriellen Datenübertragung mit Hilfe des USARTs gibt es noch einige weit verbreitete Übertragungsarten, gerade, wenn es darum geht, Daten von mehreren Sensoren einzulesen oder mit mehreren Mikrocontrollern zu korrespondieren und zu kommunizieren.

Als markanteste sind zu nennen:

- $I^2C$, TWI –Schnittstelle und Bussystem, ein Zweidrahtsystem, das auch vom AVR ATmega8 unterstützt wird.
- CAN-Bussystem, das in der Sensorik häufig Verwendung findet.
- PROFI-BUS, wenn es sich um Systeme wie SPS mit entsprechenden Sensoren geht.
- SPI – Serial Prozessor Interface.

Auf diese Bussysteme werden wir in einem späteren Studienbrief genauer eingehen, ebenso auf die dazugehörigen Protokolle.

## 2.4 Zusammenfassung

Z

- Die serielle Datenübertragung ist bei Mikrocontrollersystemen sehr wichtig. Sie spart Leitungen und kann verschiedene Baugruppen mit Daten versorgen.
- Für die serielle Datenübertragung beinhaltet der AVR ATmega8 einen speziellen Baustein, den USART. Aufgelöst: **U**niversal **S**erial **A**synchronous **R**eceiver and **T**ransmitter.
- Dieser Baustein orientiert sich an der Arbeitsweise einer V24- bzw. RS232-Schnittstelle und kann auch die entsprechenden Protokolle generieren und verstehen.
- Die Aus- bzw. Eingangssignale der seriellen Datenübertragung sind TxD und RxD. Diese Leitungen übertragen die Daten. Zwei über diese Leitungen verbundene Geräte oder Baugruppen müssen mit einem Nullmodem-Kabel versorgt werden. Die Leitungen sind gekreuzt.
- Für den Handshakebetrieb stehen die Signale CTS und RTS zur Verfügung.
- Die Datenübertragungsgeschwindigkeit ist in seriell arbeitenden Systemen sehr wichtig. Die Übertragungsgeschwindigkeit wird in BAUD gemessen und angegeben. Für das Entwicklungsboard wird 9600 BAUD verwendet.
- Die Datenübertragungsgeschwindigkeit wird im Steuerregister UBRRL und UBRRH eingetragen.
- Der USART bzw. das Senden und das Empfangen wird im Register UCSRB ein- bzw. ausgeschaltet.
- Das Steuerregister UCSRA liefert die Information, ob eine Datenübertragung schon komplett ist oder noch läuft. Sie kann im Polling abgefragt werden – es ist aber auch eine Interrupt-Steuerung möglich.

A

### 2.4.1 Aufgabe

Erklären Sie kurz, wie Sie eine Baudrate von 9600 Baud einstellen, wenn der Systemtakt 6 MHz beträgt.

Woran kann man erkennen, ob ein gesendetes Byte schon übertragen wurde oder nicht?

Woran kann man erkennen, ob ein empfangenes Byte komplett ist oder noch nicht?

Wie kann man den Empfangsteil und wie den Sendeteil im USART aktivieren?

Jetzt sind Sie wieder einmal am Ende eines Kapitels angekommen und wissen, wie man einen USART programmiert. Als nächstes Thema widmen wir uns analogen Signalen. Diese kommen in der Mikrocontrollertechnik sehr häufig vor und müssen in digitale Signale umgewandelt werden, bevor Sie im Mikrocontroller verarbeitet werden können.

# 3 Mikrocontroller - Analoge Signalverarbeitung

In diesem Kapitel lernen Sie zunächst die prinzipielle Funktion von **A**nalog/**D**igital-**U**msetzern (kurz: A/D-Umsetzer bzw. ADU) kennen. Diese werden auch als **A**nalog/**D**igital-**C**onverter (engl. converter = Umformer) mit der Abkürzung ADC bezeichnet.

In Mikrocontrollern sind A/D-Umsetzer in der Regel integriert. Im ATmega8 steht ein ADC mit einer Auflösung von 10-Bit zur Verfügung. Aber zunächst ein wenig über A/D-Umsetzer allgemein.

## 3.1 Prinzipielle Funktion und Auflösung der Analog/Digital-Umsetzer

Bevor schaltungstechnische Realisierungen der A/D-Wandler betrachtet werden, erfolgt die Vorstellung ihrer prinzipiellen Funktion. An dieser Stelle sei auf einschlägige Lehrbriefe hingewiesen, in denen der Unterschied zwischen digitalen und analogen Signalen sowie die Bedeutung der Begriffe „Quantisierung“, „Abtastung“ und „Abtasttheorem“ erläutert werden.

Die prinzipielle Funktion von A/D-Umsetzern ist die Umwandlung einer analogen (d.h. wert- und zeitkontinuierlichen) Eingangsspannung in eine ihr entsprechende digitale (d.h. zeit- und wertdiskrete) Ausgangsspannung. Diese Digitalisierung lässt sich in die drei Schritte Abtastung, Quantisierung und Codierung unterteilen. Ein A/D-Umsetzer liefert also eine der analogen Eingangsspannung proportionale Digitalzahl. Unter Verwendung des Schaltzeichens nach Norm EN 60617 (früher DIN 40900) zeigt Abbildung 13a) diese Funktion, während in Abbildung 13b) am Beispiel des Analogsignalverlaufs die Schritte der Digitalisierung zu sehen sind.

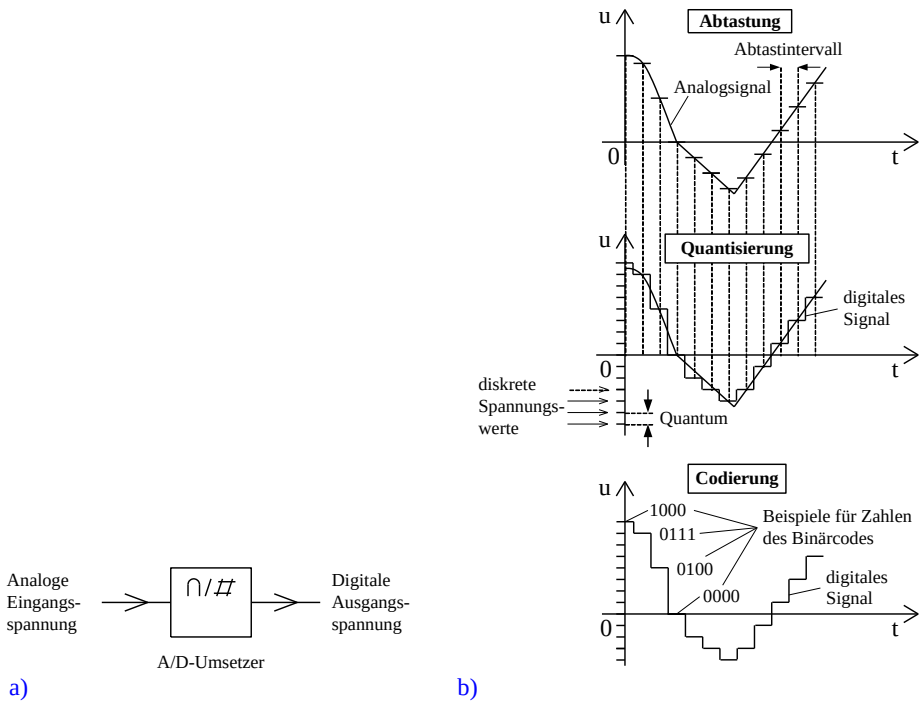



**Abbildung 13: Analog-Digital-Umsetzung**

*a) Prinzipielle Funktion und Schaltzeichen eines A/D-Umsetzers*

*b) Analogsignal mit Schritten der A/D-Umsetzung und Digitalsignal*

**Ein A/D-Umsetzer setzt analoge Eingangsspannungen in entsprechende digitale Ausgangsspannungen bzw. eine proportionale Digitalzahl um. Für die Digitalisierung des Analogsignals ist dessen Abtastung, Quantisierung und Codierung erforderlich.**

Bei der Abtastung des Analogsignals erfolgt dessen Messung in regelmäßigen Zeitabständen (Abtastintervallen) und man erhält als Ergebnis zunächst ein zeitdiskretes, aber wertkontinuierliches Signal. Dieses kommt dem Analogsignalverlauf umso näher, je kürzer das Abtastintervall bzw. je höher die Abtastfrequenz ist. Dabei ist das Abtasttheorem nach Nyquist und Shannon zu beachten, wonach die Abtastfrequenz (engl. sampling rate) mindestens doppelt so hoch wie der höchste Frequenzanteil im Analogsignal sein muss.

Bei der Quantisierung werden den Messwerten zu den diskreten Abtastzeitpunkten ganze Zahlen Z zugeordnet. Dabei gibt Z die Summe der für die möglichst exakte Darstellung des Analogwertes nötigen Quanten an. Ein Quantum Q ist die kleinste messbare bzw. darstellbare Spannung. Sie wird ULSB (engl. least significant bit = Bit mit der niedrigsten Wertigkeit, auch geringstwertiges Bit bzw. niedrigstwertiges Bit) oder LSB bezeichnet.

Ist z.B. der Analogwert +5 V und das Quantum 5 mV, so ergibt sich bei der Quantisierung die Zahl $Z = 1000$ ($1000 \cdot 5\,\text{mV} = 5\,\text{V}$)! Die Werte des Analogsignals können umso feiner dargestellt werden, je kleiner das Quantum bzw. das Raster der Quantisierungsstufen ist. So kann z.B. bei einem Quantum von 5 mV der Analogwert 5004 mV nur näherungsweise als ganzzahliges Vielfaches von 5 mV angegeben werden, nämlich zu 1001 ($1001 \cdot 5\,\text{mV} = 5005\,\text{mV}$). Diese Zahl ändert sich auch nicht, wenn der Analogwert z.B. 5003 mV ist, d.h. eine Änderung der Analogspannung um -1 mV führt zu keiner Änderung von Z!

Weisen die Quantisierungsstufen das Raster 1 mV auf, so würde die Zahl 5004 den Analogwert exakt ($5004 * 1\,\text{mV} = 5004\,\text{mV}$) darstellen. Bei 5003 mV wäre die Zahl 5003, d.h. hier bewirkt eine Änderung des Analogsignals um -1 mV eine Änderung von Z!

Je kleiner also das Quantisierungsraster ist, desto eher führt eine Änderung der Analogspannung zu einer entsprechenden Änderung von Z! Allerdings steigt die Größe der Zahlen mit der Feinheit des Quantisierungsrasters, was zu mehr Stellen bei der Codierung führt. Es können also nicht alle Amplitudenwerte des Analogsignals (dies sind ja unendlich viele) abgebildet werden, so dass sich eine wertdiskrete Darstellung ergibt. Zusammen mit der zeitdiskreten Abtastung erhält man ein wert- und zeitdiskretes Signal (Digitalsignal).

Zur Weiterverarbeitung eines Digitalsignals ist dessen Umwandlung in eine entsprechende Binär- bzw. Dualzahl nötig. Prinzipiell können auch Zahlen eines beliebigen Zahlensystems verwendet werden, das Binärsystem hat sich jedoch in der Praxis durchgesetzt. Dies erfolgt durch die Codierung, d.h. es wird jedem abgetasteten und quantisierten Wert eine Binärzahl zugeordnet und physikalisch durch entsprechende Spannungen (z.B. „0“ $\hat{=}$ 0 V und

„1“ $\hat{=}$ +5 V) ausgedrückt. Dabei steigt die Anzahl der Bits (kleinste Informationseinheit, z.B. „0“ oder „1“!) mit der Feinheit des Quantisierungsrasters (d.h. je kleiner das Quantum!). So ist z.B. zur Darstellung der Zahl 1001 im Dualcode eine 10-stellige Zahl (d.h. 10 bits!) und zur Umwandlung der Zahl 5004 eine 13-stellige Zahl (d.h. 13 bits) notwendig. Man spricht in diesem Zusammenhang von der Auflösung (engl. resolution) bzw. dem Auflösungsvermögen, welches also mit der Anzahl der Bits (d.h. der Feinheit des Quantisierungsrasters) und damit der Länge des Binärcodes steigt. Allgemein lässt sich für die Auflösung bei binären A/D-Umsetzern mit der Anzahl n der Bits angeben:

$$Auflösung\,[\%] = \frac{1}{2^n}$$

Beispiel:

Welche Auflösung in Prozent weist ein 8-bit-A/D-Umsetzer auf und um welchen Wert muss sich eine Analogspannung von 2,0 V mindestens ändern, damit sich die Zahl Z bzw. das geringstwertige Bit (LSB) des Binärcodes ändert?

Lösung:

Die Auflösung ergibt sich nach Gleichung 1.1.1 zu: $\frac{1}{2^8} = \frac{1}{256} \approx 0{,}00391 \;\hat{=}\; 0{,}391\,\%$! Sämtliche Änderungen der Analogspannung die geringer als 0,391 % ausfallen, führen zu keiner Veränderung von Z und damit des LSB. Somit muss sich der Momentanwert mindestens um $\pm 2{,}00\ \text{V} \cdot 0{,}00391 = \pm 0{,}00782\ \text{V}$ ändern.

**Die Auflösung eines A/D-Umsetzers ist eine wichtige Kenngröße, die durch die Anzahl der Bits des Binärcodes oder in Prozent angegeben werden kann. Änderungen der Analogspannung unterhalb der Auflösung führen zu keiner Veränderung des Binärcodes.**

### 3.1.1 Kenngrößen von A/D-Umsetzern

Ebenfalls in Analogie zu den Halbleiterbauelementen ist auch bei den A/D-Umsetzern die Angabe von *Kennwerten* sowie *Kennlinien* die gebräuchlichste Möglichkeit, um deren elektrisches Verhalten zu beschreiben. Einige *allgemeine* sowie jeweils bedeutende *statische* und *dynamische* Spezifikationen seien in den folgenden Ausführungen vorgestellt. Dabei handelt es sich bei den *statischen* Kenngrößen grundsätzlich um *Fehler*, welche *nach dem Abklingen* sämtlicher *Einschwingvorgäng*e vorhanden sind.

Die *dynamischen* Kenndaten beziehen sich auf Wechselsignale. Ihre Auswirkungen steigen häufig mit zunehmender Annäherung an die maximale Geschwindigkeit der Umsetzer.

Es sei an dieser Stelle angemerkt, dass eine Reihe von Parametern wie z.B. die Umsetzungsgeschwindigkeit oder die Genauigkeit *temperaturabhängig* sind, also einer *Temperaturdrift* unterliegen. Aus diesem Grund findet man in den

Datenblättern auch zahlenmäßige Angaben in Form von Temperaturkoeffizienten.

Auch der Wert und die Stabilität der *Versorgungsspannung* wirken sich auf die Kenngrößen eines A/D-Umsetzers aus. Deshalb werden entsprechende Angaben als „**P**ower **S**upply **R**ejection **R**atio“ (PSRR) oder kurz „**P**ower **S**upply **R**ejection“ (PSR) in den Datenblättern gemacht (siehe z.B. [6], [13]). Die Einheit ist dabei entweder „dB“ oder “mV/V“ bzw. “ %/V“. Die beiden zuletzt genannten Dimensionen bezeichnen dabei Ausgangsgrößenänderung im Verhältnis zur Änderung der Betriebsspannung.

**Bei den Kenngrößen von A/D-Umsetzern unterscheidet man zwischen allgemeinen, statischen und dynamischen Kenngrößen. Einige davon sind von der Temperatur und der Betriebsspannung abhängig.**

### 3.1.2 Allgemeine Kenndaten von A/D-Umsetzern

Zu den *allgemeinen* Kenngrößen der A/D-Umsetzer gehören die zahlenmäßigen Angaben bezüglich des *analogen Eingangsspannungsbereiches* bzw. des *Skalenendbereiches FSR* (engl. **f**ull **s**cale **r**ange, z.B. 0 V bis +5 V), der *Auflösung* (z.B. 12 bit), des *Eingangswiderstandes*, (z.B. 2,5 kΩ) und der *Wandlungszeit* (z.B. 50 ns, stark abhängig vom Umsetzungsverfahren). Außerdem sind Angaben in Bezug auf die *Genauigkeit* bzw. die *Fehler* (siehe folgende Abschnitte) und das *Driftverhalten* (Kenngrößenänderungen durch Alterung und Umgebungseinflüsse, wie z.B. Temperatur) von Bedeutung.

### 3.1.3 Übertragungskennlinie und statische Kenngrößen von A/D-Umsetzern

Betrachtet wird die folgende Abbildung, welche die Übertragungskennlinie eines idealen 3-bit-A/D-Umsetzers, deren Annäherung durch eine Treppenfunktion sowie den stets auftretenden, systematischen Quantisierungsfehler zeigt.

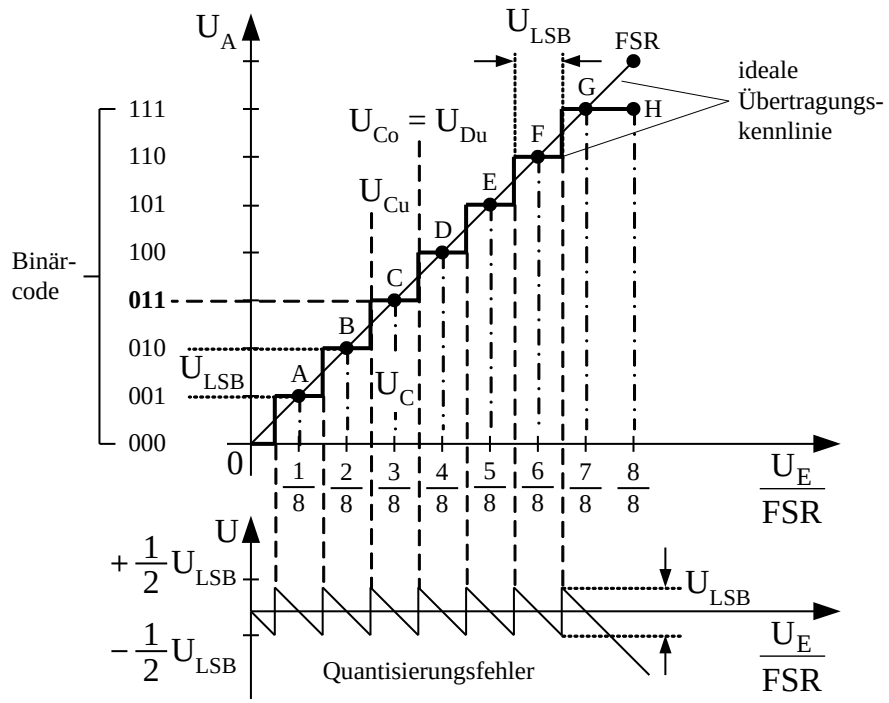



**Abbildung 14: Übertragungskennlinie eines idealen A/D-Umsetzers mit Annäherung durch eine fehlerfreie Treppenkurve sowie Quantisierungsfehler**

Mit den 3 Bits lassen sich entsprechend dem eingetragenen Binärcode $N = 2^3 = 8$ Zustände darstellen. Der gesamte Eingangsspannungsbereich bzw. der Skalenendbereich FSR kann demnach in 8 *gleich große* Quanten $Q = U_{LSB}$ unterteilt werden und man spricht von der *linearen* Quantisierung. Es ergeben sich somit die diskreten Ausgangsspannungswerte A bis G, wobei jedem Punkt eine Codezahl zugeordnet ist („Codemittelpunkte“). Verbindet man diese Spannungswerte untereinander, so erhält man die *durch den Nullpunkt gehende ideale Übertragungskennlinie mit der Steigung 1*. Diese wird mit den 8 möglichen Binärzahlen am besten durch die eingezeichnete Treppenkurve angenähert.

Beachten Sie, dass die *Ausgangsspannung den Skalenendbereich FSR (das wäre bei 8/8) - unabhängig vom Wandlertyp – nicht erreichen kann*. Mit n bits lassen sich nämlich $2^n$ Zustände bzw. Codezahlen *inklusive dem Wert Null* darstellen. Das bedeutet, dass es nur $2^n$ -1 Zustände größer Null gibt. Um den Punkt FSR der idealen Übertragungskennlinie zu erreichen, wären aber $2^n$ Schritte über Null erforderlich, so dass bei 8/8 ($\hat{=}$ FSR) der Codemittelpunkt für die Ausgangsspannung der Punkt H ist. Dennoch wird in den Datenblättern immer der Skalenendbereichswert angegeben (Nominalwert) und nicht der geringere, tatsächlich erreichbare Wert!

Liegt nun $U_E$ z.B. im Bereich von $U_{Cu}$ und $U_{Co}$ (**u**ntere und **o**bere Grenze von **C**), so werden diese Eingangsspannungswerte *alle* durch die *eine* Binärzahl 011

ausgedrückt. Dies ist jedoch nur für den einen Spannungswert $U_E = U_C$ exakt richtig. Je weiter sich $U_E$ – ausgehend von $U_E = U_C$ - dem Wert $U_{Co}$ nähert, desto größer wird der Betrag der Abweichung zwischen dem tatsächlichen Wert der Eingangsspannung und dem durch die Binärzahl 011 repräsentierten Wert der Ausgangsspannung. Ist $U_{Cu} < U_E < U_C$, so gilt dies entsprechend, jedoch mit anderem Vorzeichen, d.h. die Binärzahl ist dann zu groß. Dabei ist der Betrag der maximalen Abweichung jeweils 0,5 $U_{LSB}$ (halbes Quantum).

Trägt man die Abweichungen der durch die Binärzahl repräsentierten Ausgangsspannung von der Eingangsspannung in Abhängigkeit von der Größe der Eingangsspannung auf, so ergibt sich der dargestellte *sägezahnförmige* Verlauf des *Quantisierungsfehlers* (engl. quantization error). Hierbei handelt es sich um einen *statischen* Fehler, welcher *nur bei A/D-Umsetzern* (nicht bei D/A-Wandlern!) auftritt. Man spricht in diesem Zusammenhang auch vom *Quantisierungsrauschen*. Beachten Sie, dass dessen Mittelwert Null ist (reine Wechselgröße), der Spitze-Spitze-Wert bei $U_{LSB}$ liegt und sich für den Effektivwert $U_{LSB} / \sqrt{12}$ ergibt. Der Wert des Quantisierungsfehlers lässt sich nur durch eine Erhöhung der Auflösung (Verkleinerung von $U_{LSB}$) verringern, jedoch nicht beseitigen!

**Der Quantisierungsfehler (bzw. das Quantisierungsrauschen) stellt einen systematischen und stets vorhandenen Fehler der A/D-Umsetzer dar, der infolge der begrenzten Auflösung entsteht. Sein Wert beträgt 0,5 · $U_{LSB}$ und sinkt mit zunehmender Auflösung.**

Neben dem Quantisierungsfehler gibt es noch weitere statische Fehler, deren prinzipielle Einflüsse zusammenfassend die Abbildung 15 zeigt.

In der Abbildung) ist zunächst der *differentielle Linearitätsfehler* (engl. differential linearity error, kurz DLE), welcher auch als *differentielle Nichtlinearität* (engl. differential nonlinearity, kurz DNL) bezeichnet wird, zu sehen.

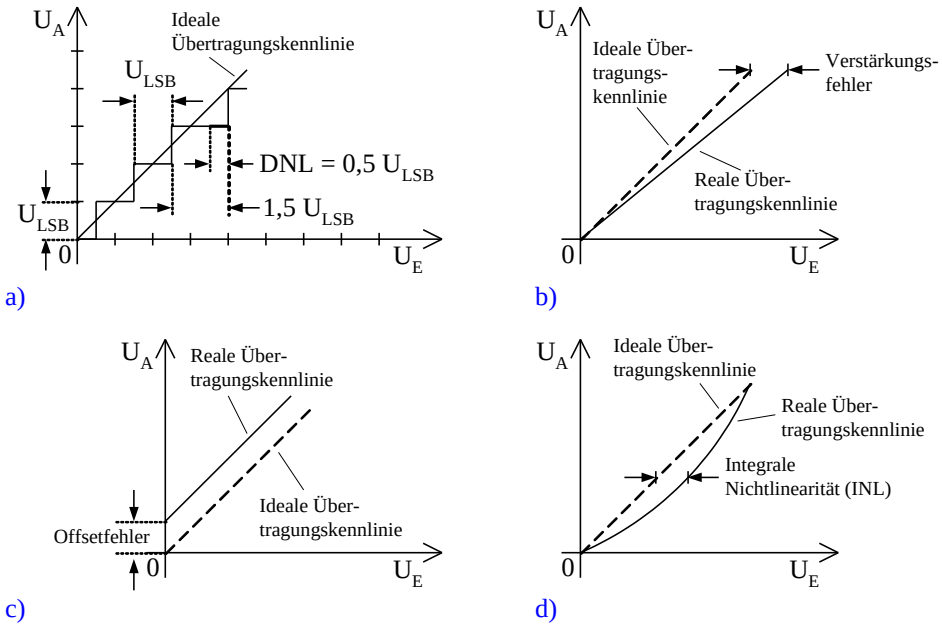



**Abbildung 15: Prinzipdarstellungen der statischen Fehler einer A/D-Umsetzung**

a) Differentieller Linearitätsfehler
b) Verstärkungsfehler (Skalenfaktorfehler)
c) Offsetfehler (Nullpunktfehler)
d) Integraler Linearitätsfehler

- Beim differentiellen Linearitätsfehler handelt es sich um eine Abweichung der tatsächlichen Größe des Quantums von dessen Sollwert.
- Als Verstärkungsfehler bezeichnet man die Abweichung der Steigung der tatsächlichen Übertragungskennlinie von der idealen Steigung (= 1).
- Der Offsetfehler gibt an, um welchen Wert die tatsächliche Übertragungskennlinie gegenüber dem Nullpunkt parallelverschoben ist.
- Der integrale Linearitätsfehler entsteht durch nicht gleich breite Quantisierungsstufen und gibt die über den gesamten Eingangsspannungsbereich maximal vorhandene Abweichung der realen Übertragungsfunktion von der idealen Transferfunktion an.

Nach diesen allgemeinen Betrachtungen zu A/D-Umsetzern wenden wir uns nun dem integrierten A/D-Umsetzer des ATmega8 zu.

## 3.2 10-Bit-Analog/Digitalumsetzer im ATmega8

Der integrierte Analog/Digitalumsetzer arbeitet in 10-Bit-Auflösung und kann per Steuerregister auf eine Auflösung von 8 Bit eingestellt werden.

Aus der allgemeinen Beschreibung sind die Grunddaten und das interne Schaltbild zu entnehmen:

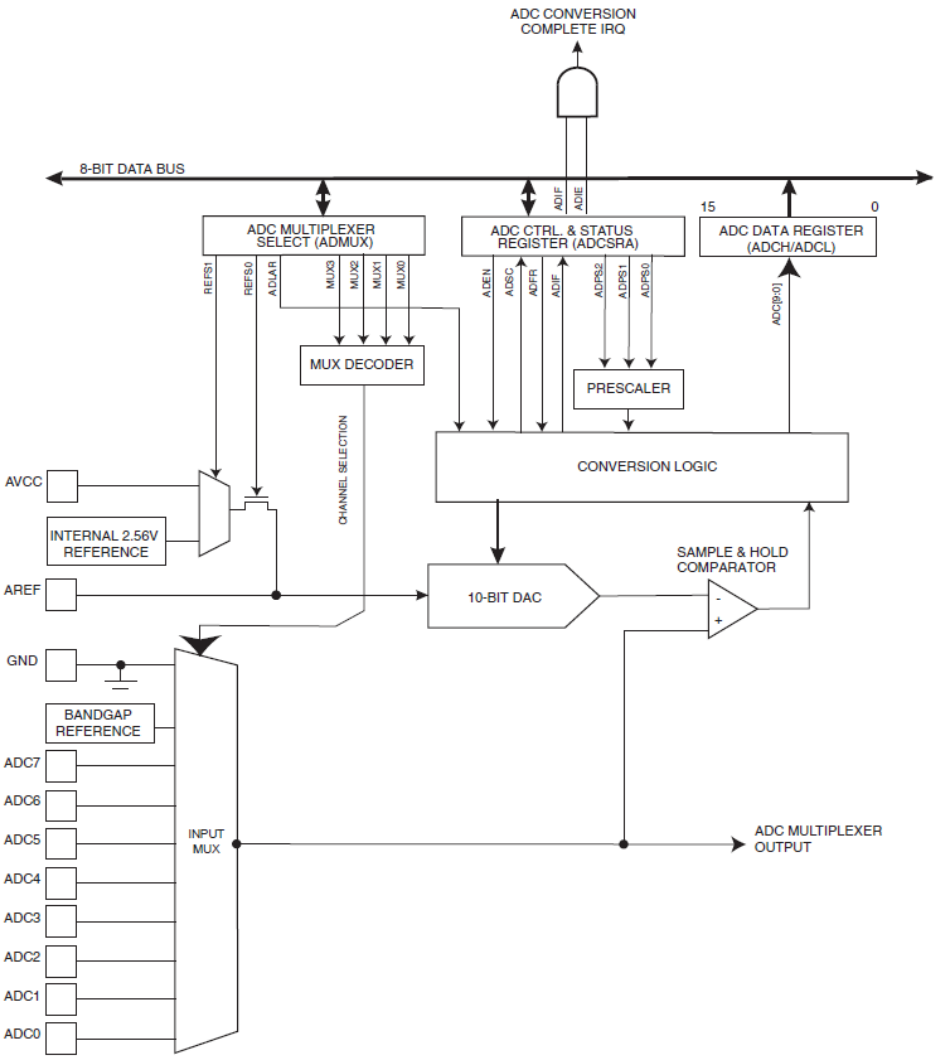



**Abbildung 16:**
**Das Schaltbild des A/D-Wandlers im ATmega8**

- **10-bit Resolution**
- **0.5 LSB Integral Non-linearity**
- **±2 LSB Absolute Accuracy**
- **13µs - 260µs Conversion Time**
- **Up to 15 kSPS at Maximum Resolution**
- **6 Multiplexed Single Ended Input Channels**
- **2 Additional Multiplexed Single Ended Input Channels (TQFP and QFN/MLF Package only)**
- **Optional Left Adjustment for ADC Result Readout**
- **0 - $V_{CC}$ ADC Input Voltage Range**
- **Selectable 2.56V ADC Reference Voltage**
- **Free Running or Single Conversion Mode**
- **Interrupt on ADC Conversion Complete**
- **Sleep Mode Noise Canceler**

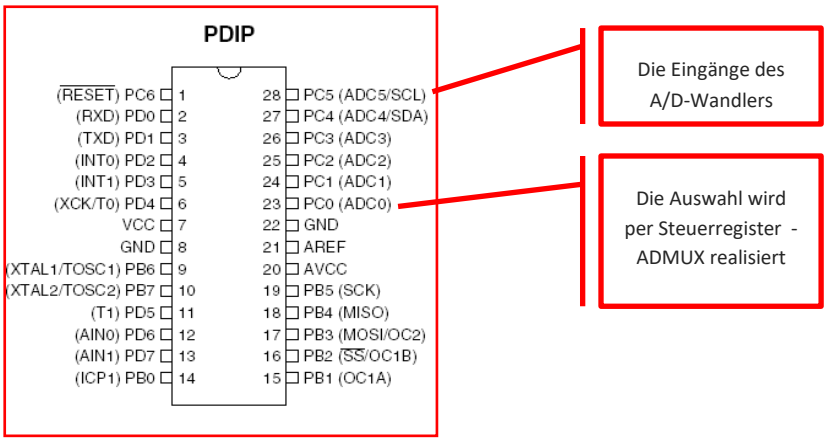



**Abbildung 17: Pinbelegung des ATmega8 für die Analog-Digital-Umsetzung**

Der A/D-Umsetzer bietet für die Analogsignale 6 Eingänge, die gemultiplext werden. Diese Eingänge stehen am PORTC zur Verfügung. Die Sockelbeschaltung zeigt die Anschlüsse.

Wie aus dem Blockschaltplan ersichtlich ist, wird der A/D-Umsetzer durch drei Steuerregister eingestellt. Diese Steuerregister und deren Bedeutung sehen wir uns etwas genauer an:

### 3.2.1 ADC Muliplexer ADMUX

| ***ADC Multiplexer Selection Register – ADMUX*** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ***Bit*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ***Name*** | REFS1 | REFS0 | ADLAR | - | MUX3 | MUX2 | MUX1 | MUX0 |
| ***R/W*** | R/W | R/W | R/W | R | R/W | R/W | R/W | R/W |
| ***Initial*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| ***Bit/Stelle*** | ***Bedeutung im ADMUX*** | ***Bezeichnung*** |
|---|---|---|
| 7 | Einstellung der Referenzspannung (s. Tabelle) | REFS1 |
| 6 | Einstellung der Referenzspannung (s. Tabelle) | REFS0 |
| 5 | Einstellung des Wandlers auf 8Bit, Ergebnis im ADCH | ADLAR |
| 4 | Keine Bedeutung | |
| 3 | Kanalauswahl | MUX3 |
| 2 | Kanalauswahl | MUX2 |
| 1 | Kanalauswahl | MUX1 |
| 0 | Kanalauswahl | MUX0 |

In diesem Steuerregister wird neben der Referenzspannung, die extern oder intern gewählt werden kann, die Einstellung für die Auflösung des AD-Umsetzers eingestellt, sie kann von 10Bit auf 8Bit umgestellt werden.

| ***REFS1*** | ***REFS0*** | ***Voltage Reference Selection*** |
|---|---|---|
| 0 | 0 | AREF, internal $V_{ref}$ turned off |
| 0 | 1 | $AV_{CC}$ with external capacitor at AREF pin |
| 1 | 0 | Reserved |
| 1 | 1 | Internal 2.56 V Reference with external capacitor at AREF pin |

Will man mit der internen Referenzspannung arbeiten, dann sind zwei 1-Signale an den Stellen REFS1 und REFS0 zu setzen.

Eine dritte Funktion haben die ersten vier Bits dieses Registers. Hier kann der Eingangskanal ausgewählt werden. Beim ATmega8 sind dies 6 Kanäle, die auf den PORTs C0 bis C5 zugänglich sind.

| *MUX3…0* | *Eingang* | *PIN* |
|---|---|---|
| 0000 | ADC 0 | PORT C0 |
| 0001 | ADC 1 | PORT C1 |
| 0010 | ADC 2 | PORT C2 |
| 0011 | ADC 3 | PORT C3 |
| 0100 | ADC 4 | PORT C4 |
| 0101 | ADC 5 | PORT C5 |

### 3.2.2 Konfiguration der Betriebsart / Wandlertakt ADCSRA

In diesem Steuerregister des AD-Wandlers wird der Umsetzer eingeschaltet, gestartet und die Interrupt-Konfiguration gewählt. Ebenfalls in diesem Register ist der Wandlertakt wählbar. Die Basis für die Taktung ist der Systemtakt. Auf unserem Entwicklungsboard beträgt die Taktfrequenz 3,6864 MHz. Diese muss verlangsamt werden, und das geschieht durch einen Vorteiler, der in diesem Register eingestellt werden kann.

| *ADC Control and Status Register A – ADCSRA* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ***Bit*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ***Name*** | ADEN | ADSC | ADFR | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 |
| ***R/W*** | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ***Initial*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| *Bit/Stelle* | *Bedeutung im ADCSRA* | *Bezeichnung* |
|---|---|---|
| 7 | ADC einschalten = 1 | ADEN |
| 6 | Start der AD-Umsetzung =1 | ADSC |
| 5 | Free Run Select | ADFR |
| 4 | ADC Interrupt Flag | ADIF |
| 3 | ADC Interrupt enable /Interrupt einschalten | ADIE |
| 2 | Frequenz-Vorteiler | ADPS2 |
| 1 | Frequenz-Vorteiler | ADPS1 |
| 0 | Frequenz-Vorteiler | ADPS0 |

Im Frequenz-Vorteiler lassen sich folgende Teilerverhältnisse einstellen:

| *ADPS2* | *ADPS1* | *ADPS0* | *Division Factor* |
|---|---|---|---|
| 0 | 0 | 0 | 2 |
| 0 | 0 | 1 | 2 |
| 0 | 1 | 0 | 4 |
| 0 | 1 | 1 | 8 |
| 1 | 0 | 0 | 16 |
| 1 | 0 | 1 | 32 |
| 1 | 1 | 0 | 64 |
| 1 | 1 | 1 | 128 |

Übliche Abtastfrequenzen liegen im Bereich 50kHz bis 200kHz. Bei einem Systemtakt von 3,6864 MHz sind somit folgende Teilerverhältnisse möglich:

$$Teilerverhältnis(\max) = \frac{Systemtakt}{Wandlertakt} = \frac{3{,}6864\ MHz}{50\ kHz} = 73{,}7$$

$$Teilerverhältnis(\min) = \frac{Systemtakt}{Wandlertakt} = \frac{3{,}6864\ MHz}{200\ kHz} = 18{,}4$$

Im Bereich „der Mitte“ würden die Verhältnisse 32 oder 64 liegen, die wir für unser Entwicklungssystem ebenfalls nutzen.

### 3.2.3 Die Ergebnisregister ADCH und ADCL

Ist eine Umsetzung der Analogspannung in einen digitalen Wert fertig, signalisiert dies der AD-Wandler, indem er das Bit 6 im ADCSRA auf 0 setzt. Das Ergebnis selbst ist in den Registern ADCH und ADCL abgelegt. Hier ist darauf zu achten, dass zuerst der Wert aus ADCH und dann der Wert aus ADCL gelesen werden muss. Weiter ist zu beachten, dass bei der Wahl der 8-Bit-Auflösung das Ergebnis nur im ADCH-Register steht und dort entnommen werden kann.

**The ADC Data Register – ADCL and ADCH**

*ADLAR = 0*

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | – | – | ADC9 | ADC8 | ADCH |
| | ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 | ADC1 | ADC0 | ADCL |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | R | R | R | R | |
| | R | R | R | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

*ADLAR = 1*

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | ADC9 | ADC8 | ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 | ADCH |
| | ADC1 | ADC0 | – | – | – | – | – | – | ADCL |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | R | R | R | R | |
| | R | R | R | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Beispiel für die Einstellung des AD-Umsetzers:**

- Der Umsetzer soll mit 8 Bit Auflösung betrieben werden.
- Der Umsetzer soll mit interner Referenzspannung betrieben werden.
- Der Umsetzer soll im Polling arbeiten.
- Der Umsetzer soll mit 115,200Hz betrieben werden.
- Die Spannung soll an PORTC, Stelle 0 eingelesen werden.

Damit müssen die Steuerregister wie folgt eingestellt werden:

| *Bit/Stelle* | *Bedeutung im ADMUX* | *Bezeichnung* | *Bit* |
|---|---|---|---|
| 7 | Einstellung der Referenzspannung (s. Tabelle) | REFS1 | 1 |
| 6 | Einstellung der Referenzspannung (s. Tabelle) | REFS0 | 1 |
| 5 | Einstellung des Umsetzers auf 8Bit, Ergebnis im ADCH | ADLAR | 1 |
| 4 | Keine Bedeutung | | 0 |
| 3 | Kanalauswahl | MUX3 | 0 |
| 2 | Kanalauswahl | MUX2 | 0 |
| 1 | Kanalauswahl | MUX1 | 0 |
| 0 | Kanalauswahl | MUX0 | 0 |
| *Bit/Stelle* | *Bedeutung im ADCSRA* | *Bezeichnung* | |
| 7 | ADC einschalten =1 | ADEN | 1 |
| 6 | Start der AD-Wandlung =1 | ADSC | 1 |
| 5 | Free Run Select | ADFR | 0 |
| 4 | ADC Interrupt Flag | ADIF | 0 |
| 3 | ADC Interrupt enable /Interrupt einschalten | ADIE | 0 |

0b11100000

0b11000101

| | | | |
|---|---|---|---|
| 2 | Frequenz - Vorteiler | ADPS2 | 1 |
| 1 | Frequenz - Vorteiler | ADPS1 | 0 |
| 0 | Frequenz - Vorteiler | ADPS0 | 1 |

**Beispiel für die Ermittlung eines Digitalwertes:**

In einer Programmschleife wird der Analogwert an PORTC0 ständig abgefragt und als Zahlenwert an den USART übergeben.

Den Analogwert soll das Potentiometer auf dem Entwicklerboard liefern.

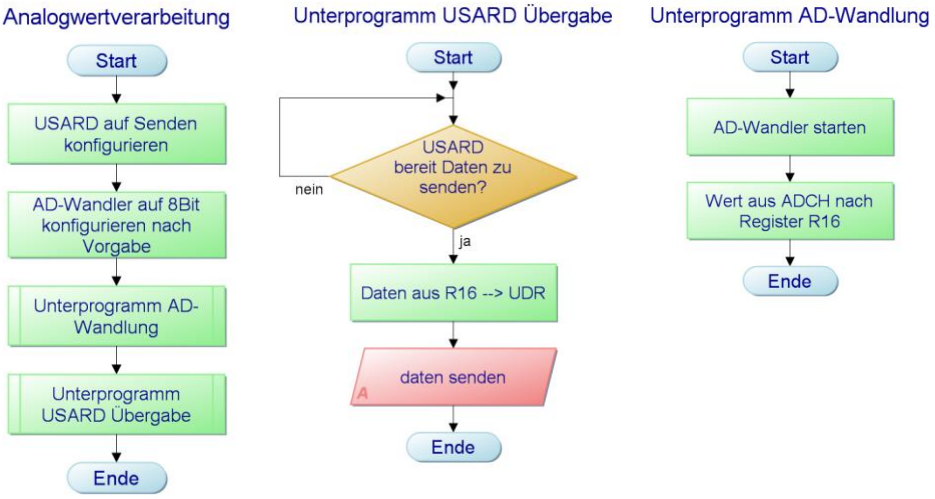



**Abbildung 18: Das Programm Analogwertverarbeitung – gegliedert in zwei Unterprogramme**

Bei der Programmumsetzung müssen zunächst in „main:“ der USART und der AD-Umsetzer konfiguriert werden:

```
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
; ------------------ADC auf 8 Bit, Interne Ref. Spannung, Kanal 0
            ldi     r16,0b11100000
            out     ADMUX,r16
; ------------------ADC aktivieren und starten, Vorteiler 32
            ldi     r16,0b11000101
            out     ADCSRA,r16

; ------------------UART Übertragungsrate und Transmit aktivieren
            ldi     r16,23              ;Wert für 9600 Baud
            out     UBRRL,R16           ;Übergabe an das Steuerregister
            sbi     UCSRB,3             ;Transmit einschalten
```

Damit sind alle Einstellungen des USART und des AD-Umsetzers getan. Bitte vergleichen Sie die entsprechenden Tabellen.

Im Hauptprogramm werden nun die zwei Unterprogramme aufgerufen und die AD-Wandlung durch Setzten des Bit 6 im ADCSRA gestartet. Die entsprechenden Programmzeilen sehen so aus:

```
;-----------------------------------------------------------------------
mainloop:   wdr
            sbi     ADCSRA,6            ;Wandler einschalten
```

```
            rcall   ADC                 ;Unterprogrammaufruf ADC
            rcall   SENDEN              ;Unterprogrammaufruf SENDEN
            rjmp    mainloop
;-----------------------------------------------------------------------
```

Das Umwandlungsprogramm als Unterprogramm sieht so aus:

```
;-----------------------------------------------------------------------
ADC:        sbic    ADCSRA,6            ;Warten bis Wandlung komplett
            rjmp    ADC
            in      R16,ADCH            ;Wert an R16 übergeben
            ret
```

An Bit 6 wird ein Null-Signal erwartet, wenn die Umwandlung komplett ist. Solange wird die Schleife durch rjmp ausgeführt. Ist die Umwandlung komplett, wird das Ergebnis aus ADCH ins Register R16 übertragen. Das Register R16 wird im Sendeprogramm dann an den PC gesendet.

```
;-----------------------------------------------------------------------
SENDEN:     sbis    UCSRA,5             ;Warten, bis Ergebnis vollständig
            rjmp    SENDEN
            out     UDR,R16             ;Inhalt von R16 senden
            ret
```

Wenn Sie das Programm mit dem Entwicklungsboard testen, dann können Sie auch gleich eine neue Funktion erproben: die Testfunktion mit der Oszilloskop-Darstellung.

Die Testfunktion haben Sie ja bei den Programmen zum USART schon kennen gelernt. Benutzen Sie diese Funktion jetzt wieder, aber stellen Sie die Funktion auf „Oszi“ um. (Achten Sie auch hier wieder auf die richtige Einstellung der COM-Schnittstelle.) Sie sollten dann einen geraden Strich sehen, den Sie durch Drehen am Potentiometer nach oben oder unten bewegen können.

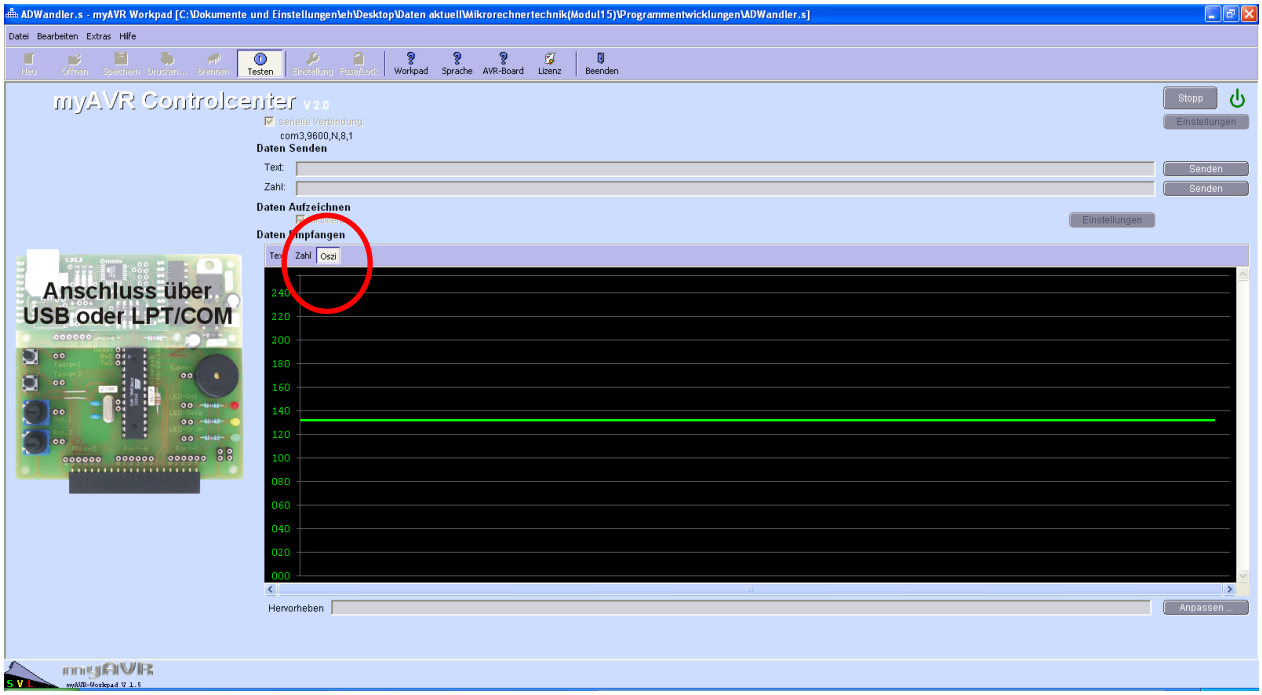



**Abbildung 19: Die Testfunktion als Oszilloskop-Ersatz. Die im ADC umgesetzten Spannungswerte können dargestellt werden**

### 3.2.4 Aufruf des AD-Umsetzers mit Interrupt-Auslösung

Statt im Polling kontinuierlich nachzufragen, ob die Umsetzung des analogen Signals in ein digitales Signal fertig ist, kann man das Ende der Umsetzung auch per Interrupt erfahren.

In diesem Fall müssen die Bits in ADCSRA wie folgt eingestellt werden:

```
            ldi     r16,0b11011101
            out     ADCSRA,r16
```

Die Bits 3 und 4 des ADCSRA schalten in den Interrupt-Modus. Alle anderen Bits können von der vorigen Einstellung übernommen werden.

Statt des Unterprogramms ADC verwenden wir nun eine Interrupt-Service-Routine, die in der Interrupt-Vektor-Tabelle eingetragen werden muss:

```
            rjmp    onADC               ;15    ADC Conversion Complete
```

Auf der Vektor-Nr. 15 muss der reti-Befehl gegen einen rjmp-Befehl zur Interrupt-Service-Routine ausgetauscht werden. Die Routine beginnt in unserem Beispiel bei der Sprungmarke onADC.

Das vollständige Programm sehen Sie hier:

```
;-------------------------------------------------------------------------
; Titel             : Spannungswert mit Interrupt
;-------------------------------------------------------------------------
; Funktion          : Wandelt ein analoges Signal in ein digitales um
;                   : dieses wird als Wert auf das Oszilloskop geben
; Schaltung         : PORTC.0=Pot1,
;-------------------------------------------------------------------------
; Prozessor         : ATmega8
; Takt              : 3,6864 MHz
; Sprache           : Assembler
; Autor             : Edgar Hoch
;-------------------------------------------------------------------------
.equ        F_CPU,  3686400
.include    "AVR.H"
;-------------------------------------------------------------------------
;Reset and Interruptvectoren            ;VNr.   Beschreibung
            rjmp    main                ; 1    POWER ON RESET
            reti                        ; 2    Int0-Interrupt
            reti                        ; 3    Int1-Interrupt
            reti                        ; 4    TC2 Compare Match
            reti                        ; 5    TC2 Overflow
            reti                        ; 6    TC1 Capture
            reti                        ; 7    TC1 Compare Match A
            reti                        ; 8    TC1 Compare Match B
            reti                        ; 9    TC1 Overflow
            reti                        ;10    TC0 Overflow
            reti                        ;11    SPI, STC Serial Transfer
            reti                        ;12    UART Rx Complete
            reti                        ;13    UART Data Register Empty
            reti                        ;14    UART Tx Complete
            rjmp    onADC               ;15    ADC Conversion Complete
            reti                        ;16    EEPROM Ready
            reti                        ;17    Analog Comperator
            reti                        ;18    TWI (IC) Serial Interface
            reti                        ;19    Strore Program Memory Read
;-------------------------------------------------------------------------
main:
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
```

```
            ldi     r16,lo8(RAMEND)     ;Stack Initialisierung
            out     SPL,r16
; -----------------USART Übertragungsrate und Transmit aktivieren
            ldi     r16,23              ;Wert für 9600 Baud
            out     UBRRL,R16           ;Übergabe an das Steuerregister
            sbi     UCSRB,3             ;Transmit einschalten
;------------------PORT Einstellung-----------------------
            sbi     DDRB,0
            sei
;------------------AD-Umsetzer-Einstellungen---------------
            ldi     r16,0b11100000      ;Start, 8Bit, ext.Spg
            out     ADMUX,r16
            ldi     r16,0b11011101      ;int Spg. Interrupt, Teiler32
            out     ADCSRA,r16
;-------------------------------------------------------
mainloop:
            wdr
            rjmp    mainloop
;-------------------------------------------------------
onADC:      cli                         ;Interrupts sperren
            in      r17,ADCH            ; Gewandelter Wert
            rcall   SENDEN              ; Unterprogamm Senden
            sbi     ADCSRA,6            ; neuer Start Wandlung
            sei                         ;Interrupts zulassen
            reti
;-------------------------------------------------------
SENDEN:     sbis    UCSRA,5             ;Warten, bis Ergebnis vollständig
            rjmp    SENDEN
            out     UDR,R17             ;Inhalt von R17 senden
            ret
```

Z

### 3.2.5 Zusammenfassung

- Analog/Digitalwandler arbeiten nach einer bestimmte Auflösung, die auch die Genauigkeit des Wandlers bestimmt. Diese Auflösung ist im ATmega8 mit 8-Bit oder 16-Bit wählbar.
- Der Analog/Digitalwandler im ATmega8 kann 6 Analogspannungen verarbeiten. Diese werden gemultiplext und am PORTC an den Stellen 0 bis 5 eingelesen.
- Der AD/Wandler wird durch zwei Register in seiner Arbeitsweise eingestellt und liefert in zwei weiteren Registern das digitale Ergebnis.
- Die Steuerregister sind ADMUX und ADCSRA.
- Der AD/Wandler kann im Polling betrieben werden oder per Interrupt.
- Das Bit 6 im ADCSRA startet die eine Wandlung. Setzt der AD-Wandler dieses Bit wieder zurück, ist die Umwandlung komplett und das Ergebnis wird in den Registern ADCH und ADCL für eine weitere Verarbeitung zur Verfügung gestellt.
- Das Bit 5 im Register ADMUX bestimmt, ob der Wandler mit einer Auflösung von 8 Bit oder mit 10 Bit arbeitet.
- Wird der Interrupt-Modus verwendet, wird die Interrupt-Service-Routine dann aufgerufen, wenn die Umwandlung komplett ist.

A

### 3.2.6 Aufgaben zum AD-Wandler

- Welche Betriebsarten des AD-Wandlers eines AVR-Controllers kennen Sie?
- Wie erfolgt die Initialisierung des AD-Wandlers im Prinzip?
- Wie wird das Einlesen eines Analogwertes im Prinzip Programmiert?

# 4 Analog-Komparator

Ein Teil des AD-Umsetzers ist eine Vergleichsstelle, die zwei Spannungswerte miteinander vergleicht oder eine Referenzspannung mit einer externen Spannung vergleicht. Oft reicht dieser Vergleich, um im Programm entsprechende Entscheidungen zu treffen, ohne gleich den AD-Umsetzer bemühen zu müssen. Außerdem ist ein Vergleich in dieser Art und Weise sehr schnell. Das Ergebnis liegt im Nanosekundenbereich vor.

Sehen wir uns die Innenbeschaltung etwas genauer an. Hier hilft wieder einmal eine Übersicht aus den Originalunterlagen zum ATmega8:

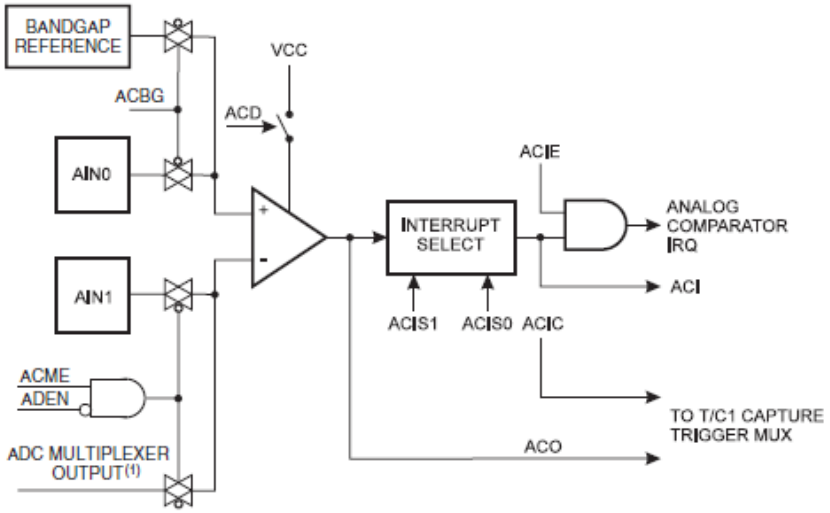



Abbildung 20: Schaltbild des Analog-Komparators im ATmega8

Die beiden zu vergleichenden Spannungen liegen an den Eingängen AIN0 und AIN1 an. Die Spannungswerte werden über den PORTD Stelle 6 und Stelle 7 für den Vergleich zugeführt. Der Vergleich kann wieder durch Polling oder durch Interruptsteuerung durchgeführt werden.

Ist die Spannung  am Eingang AIN1 kleiner als die Spannung am Eingang AIN0, dann ist das Signal an ACO=0

Ist die Spannung  am Eingang AIN1 größer als die Spannung am Eingang AIN0, dann ist das Signal an ACO=1

Die Anschlussbelegung

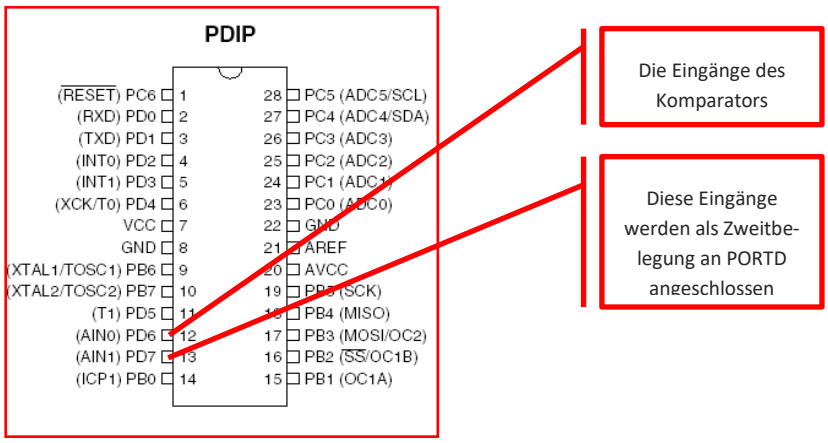



Abbildung 21: Anschluss der zu vergleichenden Spannungswerte AIN0 und AIN1

### 4.1.1 Das Konfigurationsregister ACSR

Im Konfigurationsregister ACSR werden alle Einstellungen vorgenommen, die für die Arbeit des Komparators wichtig sind.

| *Analog Comparator Control and Status Register – ACSR* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ***Bit*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ***Name*** | ACD | ACBG | ACO | ACI | ACIE | ACIC | ACIS1 | ACIS0 |
| ***R/W*** | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W |
| ***Initial*** | 0 | 0 | N/A | 0 | 0 | 0 | 0 | 0 |

| ***Bit/Stelle*** | ***Bedeutung im ACSR*** | ***Bezeichnung*** |
|---|---|---|
| 7 | Einschalten des Komparators: OFF=1 / ON=0 | ACD |
| 6 | Umschaltung von externer Spannung (Bit=0)<br>auf interne Referenzspannung 1,22V (Bit=1) | ACBG |
| 5 | Analog Comparator Output - Ausgangssignal | ACO |
| 4 | Analog Comparator Interrupt Flag | ACI |
| 3 | Analog Comparator Interrupt Enable | ACIE |
| 2 | Analog Comparator Input Capture Enable | ACIC |
| 1 | Flankenauswahl für Interrupt | ACIS1 |
| 0 | Flankenauswahl für Interrupt | ACIS0 |

**Bit 7** muss auf 0-Signal gesetzt werden, damit der Komparator aktiv ist.

**Bit 6** schaltet zwischen der internen Referenzspannung (1,22 V) und der externen Referenzspannung (an AIN0) als Vergleichsspannung um. Die interne Referenzspannung von 1,22V kann bei unterschiedlichen ATmega8-Typen verschieden sein. Hier lohnt sich ein Blick in das Datenblatt.

**Bit 5** ist das Ausgabesignal des Komparators und liefert das aktuelle Ergebnis: AIN1 < Uref: ACO=0 / AIN1 > Uref: ACO=1

**Bit 4** zeigt an, dass ein Interruptereignis des Komparators vorliegt.

**Bit 3** schaltet bei 1-Signal den Interrupt-Modus ein.

**Bit 2** regelt die Zusammenarbeit  mit dem Timer 1.

**Bit 1-0** hier kann die gewünschte Flanke für die Interrupt-Steuerung ausgewählt werden.

| ***ACIS1*** | ***ACIS0*** | ***Voltage Reference Selection*** |
|---|---|---|
| 0 | 0 | Komparator-Interrupt bei jedem Flankenwechsel |
| 0 | 1 | Reserved |
| 1 | 0 | Komparator-Interrupt bei negativer (fallender) Flanke |
| 1 | 1 | Komparator-Interrupt bei positiver (steigender) Flanke |

Neben der fallenden oder steigenden Flanke kann auch jede Änderung als Interrupt-Signal ausgewählt werden.

### 4.1.2 Beispiel:

Die Spannung aus einem optischen Sensor soll ausgewertet werden. Wenn der Sensor eine Spannung US liefert, die höher ist als eine an einem Potentiometer einzustellende Spannung UP, dann soll eine rote LED aufleuchten.

Die Schaltung des Sensors ist eine Reihenschaltung eines Festwiderstandes mit dem LDR. Das Potentiometer liegt konstant an 5V. Die Spannung UP wird über den Schleifring abgegriffen und an Port D6 angeschlossen.

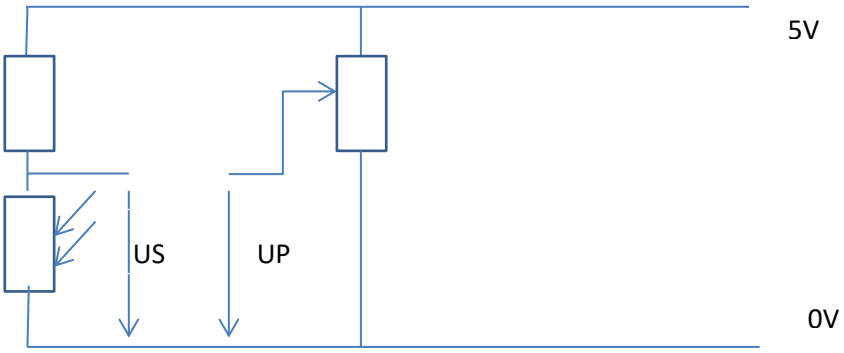



**Abbildung 22: Die Spannungen werden extern auf dem Entwicklungsboard generiert. Die Spannungen werden dann an PD6/PD7 angeschlossen**

### Programmvariante Vergleich zweier Spannungen per Polling

Wir können also mit der Initialisierung des ADC beginnen:

```
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            cbi     ACSR, 7             ;Komparator einschalten
            cbi     ACSR, 6             ;ext. Referenzspannung an AIN0
            sbi     DDRB,0              ;LED-Anschluss
;-----------------------------------------------------------------------
```

Viel ist hier nicht einzustellen. Im ACSR an der Stelle 7 wird der Komparator eingeschaltet und an der Stelle 6 wird die externe Referenzspannung ausgewählt. Das ist schon alles.

Im Hauptprogramm gilt es nun, die Stelle 5 im ACSR abzufragen. Diese Stelle gibt direkt über den Zustand des Komparator-Ausgangs Auskunft und lässt sich zu Sprungzwecken nutzen.

```
mainloop:   wdr
            cbi     PORTB,0             ;LED ausschalten
            sbis    ACSR,5              ;Stelle 5 im ACSR entscheidet
            rjmp    mainloop            ;ob es zurück zur mainloop geht
            sbi     PORTB,0             ;oder die LED eingeschaltet wird.
            rjmp    mainloop
;-----------------------------------------------------------------------
```

Eine recht einfache Konstruktion also. Will man mehrere Sensoren abfragen, kann man dies mit dem Komparator ebenfalls tun, man kann die multiplexfähigen Eingänge des AD-Umsetzers mit benutzen.

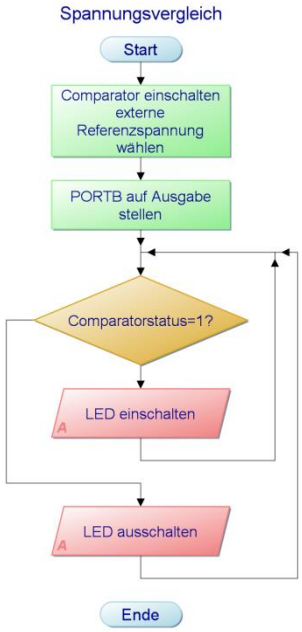



**Abbildung 23: Der Programmablaufplan zeigt die Reaktion auf den Spannungsvergleich**

Ü

### 4.1.3 Übung

Das eben gezeigte Beispiel soll in ein Programm umgewandelt werden, das mit Interrupt arbeitet. Die steigende Flanke soll den Interrupt auslösen.

Erstellen Sie das Assemblerprogramm.

Ü

### 4.1.4 Übung

Erweitern Sie das Programm so, dass folgende Logik berücksichtigt wird.

Uref > US → Rote LED ein – Grüne LED aus

Uref < US → Rote LED aus – Grüne LED ein

Z

### 4.1.5 Zusammenfassung

- Der Komparator im ATmega8 ist ein leicht zu bedienendes Instrument, wenn es darum geht, zwei Spannungen miteinander zu vergleichen.
- Der Spannungsvergleich kann extern geschehen, dann sind die Spannungswerte an PORTD Stelle 7 und Stelle 6 anzuschließen.
- Wird eine interne Referenzspannung verwendet, dann wir nur die Spannung an der Stelle 7 benötigt.
- Die Konfiguration des Komparators wird im ACSR vorgenommen.
- Der Komparator ist interruptfähig. Die Interrupt-Auslösung kann durch Flankenauswertung geschehen.

A

### 4.1.6 Aufgaben

- Was ist ein Komparator?
- Wie funktioniert ein Komparator?
- Welche unterschiedlichen Interrupts kann der Komparator des AVR-Controllers auslösen?

# 5 Timer und Counter

Wie Sie im Teil 1 in den Grundlagenbetrachtungen gesehen haben, benötigt man in Programmen Zeitverzögerungen, um bestimmte Prozesse einzuleiten oder auszuführen. Den Mikrocontroller für diese Zeitverzögerungen „mit Nichts“ zu beschäftigen, ist natürlich nicht besonders wirkungsvoll. Besser wäre es, für diesen Zweck extra ein Baustein zu bemühen. Und das können wir auch, im ATmega8 gibt es verschiedene Timer-Bausteine, die wir im Folgenden genauer ansehen möchten.

Insgesamt stehen dem Programmierer 3 Timer zur Verfügung. Ein 16-Bit Timer T1 und zwei 8-Bit Timer T0 und T2.

Diese Timer arbeiten völlig autonom im Mikrocontroller und unabhängig vom Prozessor. Das bringt natürlich viele Vorteile. Und da diese Bausteine auch noch interruptfähig sind, kann man mehrere Prozesse parallel verarbeiten.

Die Grundstruktur der Timer zeigt das folgende Bild:

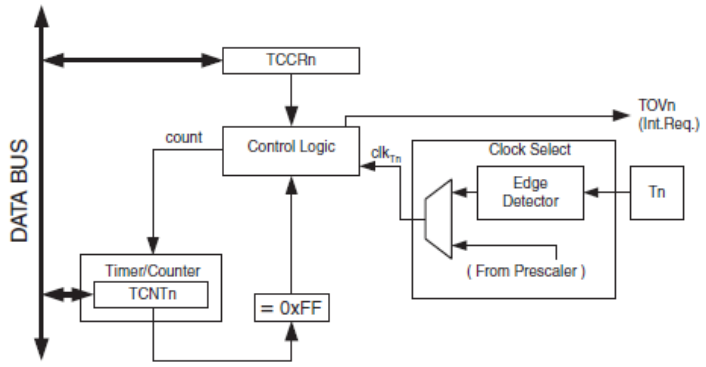



**Abbildung 24: Die Grundstruktur der Timer mit den wichtigsten Steuerregistern**

## 5.1 Timer0

Grundlage für die Zeitfunktionen ist der Systemtakt des Mikrocontrollersystems. Im Entwicklungssystem ist das der Systemtakt 3,6864 MHz. Das kennen Sie bereits.

Dieser Takt wird in einem Register, das auch Prescaler (Vorteiler) genannt wird, verlangsamt. Zuständig dafür ist das Register mit der Bezeichnung TCCR0.

Die niederwertigsten drei Bits in diesem Register bestimmen das Teilerverhältnis. Sind diese drei Bits auf Null gesetzt, wird der Timer gestoppt. Die weiteren Kombinationen sind Teilerverhältnisse von 1/1 bis 1/1024 – je nach Bitkombination.

Die letzten zwei Bitkombinationen sind sehr wichtig, denn diese Einstellungen schalten den Timer um als Zeitgeber oder Zähler. Sie werden das noch sehen.

### 5.1.1 TCCRO - Vorteiler und Zähler

Die Bedeutung der einzelnen Bits dieses Registers sehen Sie im folgenden Bild:

| *Timer/Counter Control Register – TCCR0* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ***Bit*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ***Name*** | - | - | - | - | - | CS02 | CS01 | CS00 |
| ***R/W*** | R | R | R | R | R | R/W | R/W | R/W |
| ***Initial*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| ***CS02*** | ***CS01*** | ***CS00*** | ***Resultat*** |
|---|---|---|---|
| 0 | 0 | 0 | Stopp. Der Timer/Counter wird angehalten. |
| 0 | 0 | 1 | CPU-Takt |
| 0 | 1 | 0 | CPU-Takt / 8 |
| 0 | 1 | 1 | CPU-Takt / 64 |
| 1 | 0 | 0 | CPU-Takt / 256 |
| 1 | 0 | 1 | CPU-Takt / 1024 |
| 1 | 1 | 0 | Externer Zähleingang an Pin T0, fallende Flanke |
| 1 | 1 | 1 | Externer Zähleingang an Pin T0, steigende Flanke |

Das maximale Teilerverhältnis ist mit dem Prescaler 1024 zu erreichen.

Wird mit einem Systemtakt von 3,6864 MHz gearbeitet, wäre die geringst mögliche Frequenz nach dem Prescaler demnach:

$$f = \frac{Systemtakt}{Teilerverhältnis} = \frac{3{,}6864\ MHz}{1024} = 3{,}6\ kHz$$

Oder anders ausgedrückt, mit einem Prescaler-Verhältnis von 1024 erhält man bei einem Systemtakt von 3,6864 MHz eine Zeitverzögerung von:

$$T = \frac{1}{f} = \frac{1}{3{,}6\ kHz} = 0{,}278\ ms$$

Diese Zeitverzögerung ist noch nicht allzu groß. Aber, wir haben noch eine weitere „Zeitschleife“ im Register TCNTO, die genutzt werden kann.

Zunächst muss jedoch festgelegt werden, wie ein Interrupt behandelt werden solle. Das muss im Register TIMSK vorbereitet werden. Das Register ist wie folgt aufgebaut und auch hier kommt jeder Stelle eine Bedeutung zu.

### 5.1.2 TIMSK – Timer Interrupt Mask

| *Timer/Counter Interrupt Mask Register – TIMSK* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ***Bit*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ***Name*** | OCIE2 | TOIE2 | TICIE1 | OCIE1A | OCIE1B | TOIE1 | - | TOIE0 |
| ***R/W*** | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ***Initial*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Dieses Register ist für alle drei Timer/Counter zuständig. Die Bedeutung der einzelnen Stellen ist:

| ***Bit/Stelle*** | ***Bedeutung im TIMSK*** | ***Bezeichnung*** |
|---|---|---|
| 7 | Timer 2 Output Compare Match Interrupt Enable | OCIE2 |
| 6 | Timer/Counter Overflow – Interrupt 2 | TOIE2 |
| 5 | Timer1 – Input Capture Interrupt Enable | TICIE1 |
| 4 | Timer 1 Output Compare Match A Interrupt Enable | OCIE1A |
| 3 | Timer 1 / output compare match B Interrupt Enable | OCIE1B |
| 2 | Timer/Counter Overflow – Interrupt 1 | TOIE1 |
| 1 | Ohne Bedeutung | - |
| 0 | Timer 0 – Overflow –Interrupt | TOIE0 |

In den ersten Versuchen benötigen wir nur die zu Timer0 zugehörigen Bits. Und das ist das Bit 0. Hiermit stellen wir eine Interrupt-Auslösung ein, die erfolgt, wenn das Timer Register TCNT0 einen Übergang von 255 nach 0 ausführt.

### 5.1.3 Das Zählregister TCNT0 für Timer 0

Dieses Register zählt von 0 aufwärts bis der Wert 255 erreicht ist. Beim Übergang von 255 auf 0 kann dann, wenn dies in Register TIMSK so eingestellt wurde, ein Interrupt ausgelöst werden. Wichtig ist zu wissen, dass die Differenz vom Anfangswert dieses Registers bis zum Endwert (255) der Multiplikator ist, der für die Zeitverzögerung zuständig ist.

Eine maximale Zeitverzögerung wird erreicht, wenn das TCNT0-Register mit dem Anfangswert 0 geladen wird. In diesem Fall sind es 256 Schritte, bis der Interrupt ausgelöst wird.

### 5.1.4 Zeitprogramm mit Timer0

Wenn der Vorteiler im Register TCCR0 auf den Maximalwert von 1024 eingestellt wurde, dann wird eine Frequenz von 3,85kHz an den Timer gesendet. Dividiert man nun diesem Wert durch den Maximalwert des Zählregisters (256) dann erhält man die kleinstmögliche Frequenz, die mit diesem 8-Bit Timer eingestellt werden kann.

$$f = \frac{Systemtakt}{Teilerverhältnis} = \frac{3{,}6864\ MHz}{1024} = 3{,}6\ kHz$$

$$f_{\min} = \frac{f}{256} = \frac{3{,}6\ kHz}{256} = 14{,}0625\ Hz$$

Die notwendigen Programmzeilen sind:

```
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            sbi     DDRB,0              ;LED Anschluß konfigurieren
            ldi     r16,0b00000101      ;maximal einstellbarer
            out     TCCR0,r16           ;Vorteiler in TCCR0
            ldi     r16,0b00000001      ;Interruptauslöser konfigurieren
            out     TIMSK,r16
            sei                         ;Interrupt generell zulassen
;-----------------------------------------------------------------------
mainloop:   wdr
            rjmp    mainloop
;-----------------------------------------------------------------------
TC0:        cli                         ;Interrupt sperren

            com     r16                 ;Jedes Bit wird negiert und
            out     PORTB,r16           ;der Wert an LED übergeben
            ldi     r17,00              ;Maximalwert für Timer reinit.
            out     TCNT0,r17

            sei                         ;Interrupt wieder zulassen
            reti
```

Natürlich muss auch in der Interrupt-Vektor-Tabelle der Vektor 10 für den Timer0 auf die Sprungadresse TC0 verweisen!

In der Interrupt-Service Routine wird nun der Timer-Reinit-Wert immer wieder neu generiert. Sowie der Wert für TCNT0 geladen wird, beginnt der Timer zu arbeiten. Solange, bis beim Übergang vom Wert 255 auf 0 dann wieder ein Interrupt ausgelöst wird.

Wenn Sie das Programm austesten, werden Sie sehen, die Frequenz von 14 Hz ist gerade noch vom menschlichen Auge als blinkendes Licht erfassbar.

## 5.2 Timer 1

Timer 1 ist im ATmega8 ein 16-Bit-Timer und lässt dadurch natürlich größere Zeitverzögerungen zu. Folgende Funktionen hat dieser Timer

- Overflow Interrupt
- Clear Timer on Compare Match
- Input Capture
- 2 Compare Einheiten
- diverse PWM Modi

Das allgemeine Blockschaltbild sieht so aus:

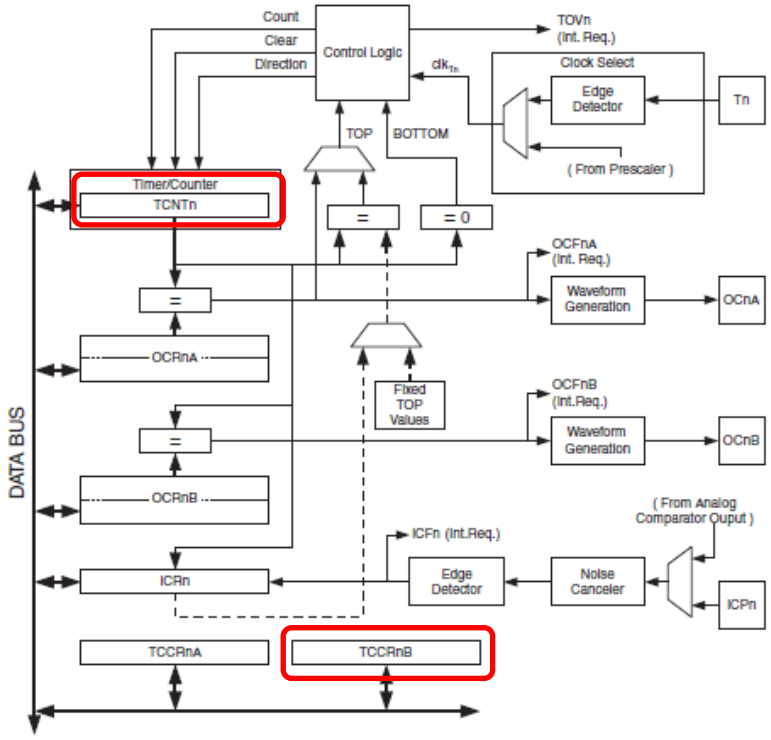



**Abbildung 25: Die Grundstruktur des Timer1 mit den wichtigsten Steuerregistern**

Zunächst einmal arbeitet er jedoch nahezu gleich wie der Timer 0. Auch hier wird wieder ein Vorteiler benutzt, dessen Einstellung wieder in einem Register zu finden ist, das für den Timer1 die Bezeichnung TCCR1B trägt.

| ***Timer/Counter 1 Control Register – TCCR1B*** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ***Bit*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ***Name*** | ICNC1 | ICES1 | - | WGM13 | WGM12 | CS12 | CS11 | CS10 |
| ***R/W*** | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W |
| ***Initial*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Momentan sind nur die drei niederwertigen Bits 0, 1 und 2 von Bedeutung und Sie sehen, die Funktionen sind die gleichen wie beim 8-Bit-Timer:

| ***CS12*** | ***CS11*** | ***CS10*** | ***Resultat*** |
|---|---|---|---|
| 0 | 0 | 0 | Stopp. Der Timer/Counter wird angehalten. |
| 0 | 0 | 1 | CPU-Takt |
| 0 | 1 | 0 | CPU-Takt / 8 |
| 0 | 1 | 1 | CPU-Takt / 64 |
| 1 | 0 | 0 | CPU-Takt / 256 |
| 1 | 0 | 1 | CPU-Takt / 1024 |
| 1 | 1 | 0 | Externer Zähleingang an Pin T1, fallende Flanke |
| 1 | 1 | 1 | Externer Zähleingang an Pin T1, steigende Flanke |

Auch hier ist das maximale Teilerverhältnis für den Vorteiler 1/1024.

In diesem Register können noch weitere Funktionen eingestellt werden, die wir aber vorerst noch nicht betrachten wollen.

### 5.2.1 TIMSK – Timer Interrupt Mask Timer 1

Das Timer Interrupt Mask Register ist ja für alle 3 Timer zuständig. Hier sehen Sie zur Erinnerung noch einmal die Belegung der Registerstellen:

| *Timer/Counter Interrupt Mask Register – TIMSK* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ***Bit*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ***Name*** | OCIE2 | TOIE2 | TICIE1 | OCIE1A | OCIE1B | TOIE1 | - | TOIE0 |
| ***R/W*** | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| ***Initial*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Für Timer 1 ist im TIMSK zunächst einmal die Stelle 2 wichtig. Hier kann der Interrupt bei Overflow für den Timer 1 ausgelöst werden.

| ***Bit/Stelle*** | ***Bedeutung im TIMSK*** | ***Bezeichnung*** |
|---|---|---|
| 7 | Timer2: Output Compare Match Interrupt Enable | OCIE2 |
| 6 | Timer/Counter2: Overflow Interrupt Enable | TOIE2 |
| 5 | Timer1: Input Capture Interrupt Enable | TCIE1 |
| 4 | Timer1: Output Compare Match A Interrupt Enable | OCIE1A |
| 3 | Timer1: Output Compare Match B Interrupt Enable | OCIE1B |
| 2 | Timer/Counter1: Overflow Interrupt Enable | TOIE1 |
| 1 | Ohne Bedeutung | - |
| 0 | Timer0: Overflow Interrupt Enable | TOIE0 |

Nun fehlt nur noch das Register TCNT, das eigentliche Zählregister. Dieses Register ist beim 16-Bit-Timer natürlich auch ein 16-Bit-Register, genauer gesagt, wieder ein zusammengesetztes Register aus zwei Einzelregistern.

### 5.2.2 Die Zählregister TCNT1L und TCNT1H für Timer 1

Diese Register zählen vom Wert 0 bis zum Wert 65535. Der eigentliche Zählvorgang ist damit wesentlich größer als beim Timer 0. Im einfachsten Fall wird dann beim Übergang von 65535 auf den Wert 0 ein Interrupt ausgelöst.

Das Laden der beiden Register TCNT1L und TCNT1H muss allerdings in zwei getrennte Vorgänge aufgeteilt werden. Die Berechnung ist ein wenig komplizierter, weil man eine Dezimalzahl in zwei Hälften teilen muss. Will man zum Beispiel nach 2500 Zählschritten einen Interrupt auslösen, dann muss in die beiden Register ein Wert von: 65535-2500=63035 geladen werden. Aber wie soll dieser Wert dann auf zwei Register verteilt werden?

Nun, am einfachsten bemüht man einen Taschenrechner, der in mehreren Zahlensystemen arbeiten kann. Beispielsweise in binär, dezimal und hexadezimal. Oder Sie rufen in Windows die Funktion „Rechner“ unter Zubehör auf, stellen den Rechner auf „wissenschaftlich“ ein und rechnen den Dezimalwert in einen Hexadezimalwert um.

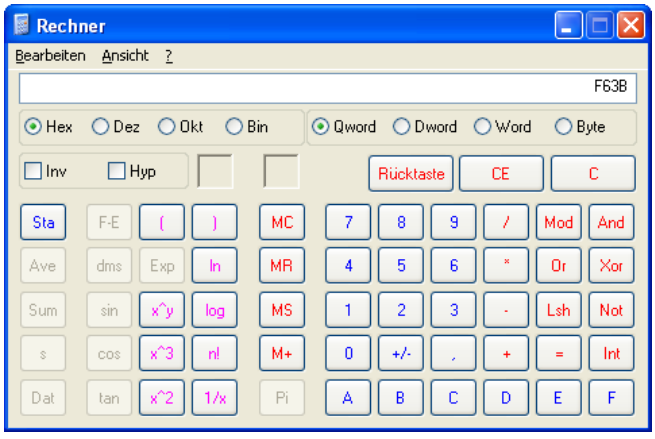



**Abbildung 26: Mit dem Windows-Rechner ist die Umrechnung des 16-Bit-Wertes recht unkompliziert**

Das Ergebnis der Umrechnung liefert den hexadezimalen Wert F63B. Damit ist dann auch klar, was man in die einzelnen Register laden muss:

**F6** muss in das Register **TCNT1H** und

**3B** muss in das Register **TCNT1L** geladen werden.

**Achtung: Die Werte für TCNT1H und TCNT1L müssen Sie in der Reihenfolge: „erst das H und dann das L-Register“ laden.**
**Sonst funktioniert diese Betriebsart nicht richtig!**

Diese Berechnung kann man in vielen Assemblern auch berechnen lassen. Damit wir aber die Darstellungen in diesem Skript so allgemein wie möglich halten wollen, verzichten wir auf diese Berechnung.

Wenn wir nun das für den Timer0 gezeigte Programm etwas „umfunktionieren“, ist es auch für den Timer1 lauffähig und bringt Zeitverzögerungen bzw. Frequenzen im Bereich

$$f = \frac{Systemtakt}{Teilerverhältnis} = \frac{3{,}6864\ MHz}{1024} = 3{,}6\ kHz$$

$$f_{min} = \frac{f}{256} = \frac{3{,}6\ kHz}{65536} = 0{,}0549\ Hz$$

Oder in eine Zeitverzögerung umgerechnet:

$$T = \frac{1}{f} = \frac{1}{0{,}0549\ Hz} = 18{,}204\ s$$

### 5.2.3 Zeitprogramm mit Timer1

Dies können Sie in einem modifizierten Programm gleich austesten.

```
;+--------------------------------------------------------------------
;| Title            : Zeitverzögerung mit Timer 1
;+--------------------------------------------------------------------
;| Funktion         : Maximale Zeitverzögerung
;| Schaltung        : ...
;+--------------------------------------------------------------------
;| Prozessor        : ATmega8
;| Takt             : 3,6864 MHz
;| Sprache          : Assembler
;| Datum            : ...
;| Version          : ...
;| Autor            : Edgar Hoch
;+--------------------------------------------------------------------
.include    "AVR.H"
;--------------------------------------------------------------------------
;Reset and Interrupt vector       ;VNr.  Beschreibung
            rjmp    main         ;1    POWER ON RESET
            reti                 ;2    Int0-Interrupt
            reti                 ;3    Int1-Interrupt
            reti                 ;4    TC2 Compare Match
            reti                 ;5    TC2 Overflow
            reti                 ;6    TC1 Capture
            reti                 ;7    TC1 Compare Match A
            reti                 ;8    TC1 Compare Match B
            rjmp    TC1          ;9    TC1 Overflow
            reti                 ;10   TC0 Overflow
```

```
            reti                    ;11  SPI, STC Serial Transfer Complete
            reti                    ;12  UART Rx Complete
            reti                    ;13  UART Data Register Empty
            reti                    ;14  UART Tx Complete
            reti                    ;15  ADC Conversion Complete
            reti                    ;16  EEPROM Ready
            reti                    ;17  Analog Comparator
            reti                    ;18  TWI (IC) Serial Interface
            reti                    ;19  Store Program Memory Ready
;-----------------------------------------------------------------------

main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            sbi     DDRB,0              ;LED Anschluß konfigurieren
            ldi     r16,0b00000101      ;maximal einstellbarer
            out     TCCR1B,r16          ;Vorteiler in TCCR1B
            ldi     r16,0b00000100       ;Interruptauslöser konfigurieren
            out     TIMSK,r16
            sei                         ;Interrupt generell zulassen
;-----------------------------------------------------------------------
mainloop:   wdr
            rjmp    mainloop
;-----------------------------------------------------------------------
TC1:        com     r16                 ;Jedes Bit wird negiert und
            out     PORTB,r16           ;Wert an LED übergeben
            ldi     r17,0x00            ;Maximalwert für Timer reinit.
            out     TCNT1L,r17          ;Niederwertige Hälfte
            ldi     r17,0x00            ;Maximalwert für Timer reinit.
            out     TCNT1H,r17          ;Höherwertige Hälfte
            reti
```

Dieses Programm liefert eine Zeitverzögerung von 18,27 Sekunden. Die Programmänderungen finden Sie in den roten Umrandungen. Die erste Änderung betrifft die Interrupt-Vektor-Tabelle. Dort muss die Interrupt-Service-Routine für TC1 eingetragen werden. Das betrifft jetzt den Vektor Nr. 9 statt zuvor den Vektor Nr.10.

Die Einstellung im Vorteiler ist gleich geblieben, das Register ist jetzt jedoch TCCR1B. Im Register TIMSK hat sich die Interrupt Definition für den Timer 1 um zwei Stellen nach links verschoben.

Und zuletzt sind in der Interrupt-Service-Routine statt TCNT0 jetzt zwei Einzelregister TCNT1L und TCNT1H zu laden. Hier jeweils mit dem Maximalwert 00.

## 5.3 Verwendung der Timer als Zähler

Sie haben nun die Timer-Funktionen des ATmega8 kennen gelernt. Zwar noch nicht alle, aber für einen ersten Überblick war das völlig ausreichend. Die bisher betrachteten Timer sind auch sehr einfach als Zähler umzustellen. Die Umstellung geschieht im TCCR-Register.

### 5.3.1 Einstellung des Timer 0 als Zähler

Zur Erklärung trägt noch einmal die Bitbedeutung dieses Registers bei.

| *Timer/Counter Control Register – TCCR0* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ***Bit*** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ***Name*** | - | - | - | - | - | CS02 | CS01 | CS00 |
| ***R/W*** | R | R | R | R | R | R/W | R/W | R/W |
| ***Initial*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| ***CS02*** | ***CS01*** | ***CS00*** | ***Resultat*** |
|---|---|---|---|
| 0 | 0 | 0 | Stopp. Der Timer/Counter wird angehalten. |
| 0 | 0 | 1 | CPU-Takt |
| 0 | 1 | 0 | CPU-Takt / 8 |
| 0 | 1 | 1 | CPU-Takt / 64 |
| 1 | 0 | 0 | CPU-Takt / 256 |
| 1 | 0 | 1 | CPU-Takt / 1024 |
| 1 | 1 | 0 | Externer Zähleingang an Pin T0, fallende Flanke |
| 1 | 1 | 1 | Externer Zähleingang an Pin T0, steigende Flanke |

Die beiden letzten Bitkombinationen für CS0 bis CS2 sind hier von Bedeutung.

Durch das Setzen der Bits CS02 und CS01 wird der Vorteiler ausgeschaltet. Stattdessen wird der Timer jetzt als Counter umgeschaltet. Die Impulse, die gezählt werden, können an der Stelle CS00 noch genauer definiert werden: entweder mit einer steigenden oder mit einer fallenden Flanke.

Im Zählregister TCNT0 werden die Impulse gezählt. Beim Übergang vom Maximalwert 255 auf den Wert 0 kann ein Overflow-Interrupt ausgelöst werden. Diese Funktion wird wie beim Timer im Register TIMSK eingestellt.

### 5.3.2 Beispielprogramm Impulszählung

Nehmen wir an, dass eine Messeinrichtung Impulse zählen soll. Nach dem 10. Impuls soll eine rote Leuchtdiode diesen Zählerstand signalisieren. Dann müssten folgende Einstellungen am Zähler gemacht werden.

1. Interrupt-Service-Routine festlegen. In der Interrupt-Tabelle ist das der Interrupt Nr.10 – wie auch schon beim Timer0
2. TCCR0 auf Zähler einstellen. Und die Impulsart festlegen (aufsteigende oder absteigende Impulsflanke).
3. Anfangswert in TCNT0 laden, damit nach 10 Impulsen ein Übergang von 255 auf 0 entsteht und damit ein Interrupt ausgelöst wird.

Mit diesen drei Einstellungen ist also der Timer auf Zähler umzustellen. Setzen wir das nun in ein Assemblerprogramm um. Zuvor aber noch ein kurzer Blick

auf die Sockelbeschaltung, damit klar ist, wo wir die Taste anschließen, die hier als Impulszähler arbeiten soll.

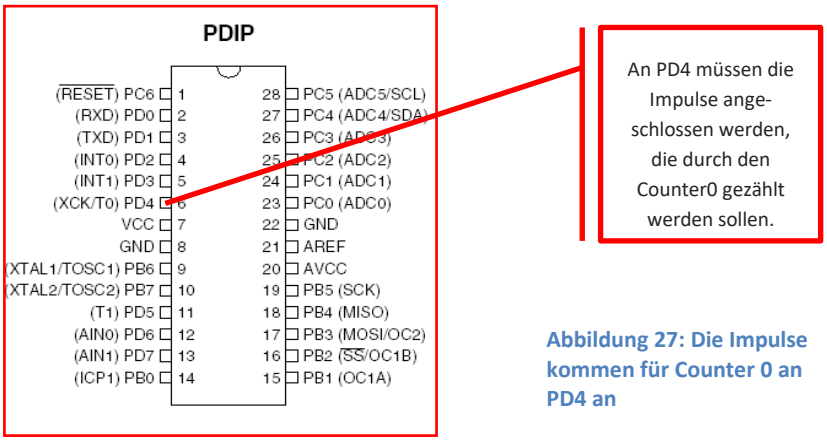



**Abbildung 27: Die Impulse kommen für Counter 0 an PD4 an**

Der Impuls für den Counter 0 ist PD4. Das ist also ein Eingang und Sie sollten diese Stelle auch im Programm als Eingang definieren und nicht den Pullup-Widerstand vergessen. Wichtig ist auch, dass der Reinit-Wert für TCNT0 schon in main einmal initialisiert wird. Ansonsten wird der erste Interrupt erst nach 256 Signalwechseln an PORTD4 wirksam.

Das Programm sieht nun so aus:

```
;+----------------------------------------------------------------------
;| Title            : Zähler mit Timer 0
;+----------------------------------------------------------------------
;| Funktion         : 10 Impulse werden gezählt und dann die LED
;| Schaltung        : an PORTB umgeschaltet
;+----------------------------------------------------------------------
;| Prozessor        : ATmega8
;| Takt             : 3,6864 MHz
;| Sprache          : Assembler
;| Datum            : ...
;| Version          : ...
;| Autor            : Edgar Hoch
;+----------------------------------------------------------------------
.include    "AVR.H"
;----------------------------------------------------------------------
;Reset and Interrupt vector       ;VNr.  Beschreibung
            rjmp    main         ;1    POWER ON RESET
            reti                 ;2    Int0-Interrupt
            reti                 ;3    Int1-Interrupt
            reti                 ;4    TC2 Compare Match
            reti                 ;5    TC2 Overflow
            reti                 ;6    TC1 Capture
            reti                 ;7    TC1 Compare Match A
            reti                 ;8    TC1 Compare Match B
            reti                 ;9    TC1 Overflow
            rjmp    TC0          ;10   TC0 Overflow
            reti                 ;11   SPI, STC Serial Transfer Complete
            reti                 ;12   UART Rx Complete
            reti                 ;13   UART Data Register Empty
            reti                 ;14   UART Tx Complete
            reti                 ;15   ADC Conversion Complete
            reti                 ;16   EEPROM Ready
            reti                 ;17   Analog Comparator
            reti                 ;18   TWI (IC) Serial Interface
            reti                 ;19   Store Program Memory Ready
;----------------------------------------------------------------------
```

```
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            sbi     DDRB,0              ;LED Anschluß konfigurieren
            cbi     DDRD,4              ;PROTD4 auf Eingabe stellen
            Sbi     PORTD,4             ;Pullup an PORTD4 setzen
            ldi     r16,0b00000110      ;maximal einstellbarer
            out     TCCR0,r16           ;Vorteiler in TCCR0
            ldi     r16,0b00000001      ;Interruptauslöser konfigurieren
            out     TIMSK,r16
            ldi     r17,245             ;Maximalwert für Timer reinit.
            out     TCNT0,r17
            sei                         ;Interrupt generell zulassen
;-----------------------------------------------------------------------
mainloop:   wdr
            rjmp    mainloop
;-----------------------------------------------------------------------
TC0:        com     r16                 ;Jedes Bit wird negiert und
            out     PORTB,r16           ;Wert an LED übergeben
            ldi     r17,245             ;Maximalwert für Timer reinit.
            out     TCNT0,r17
            reti
```

Nach jeweils 10 Zählimpulsen wird der Wert an PORTB invertiert. Man muss allerdings an dieser Stelle erwähnen, dass die im Entwicklungsboard verwendeten Tasten stark prellen. Möglicher weise kann das zu Fehlzählungen führen.

Benötigt man größere Zähler, kann man den Timer 1 als Counter einsetzen. Hier ist dann das Zählen von 65536 Zählimpulsen möglich.

### 5.3.3 Timer 1 als Zähler

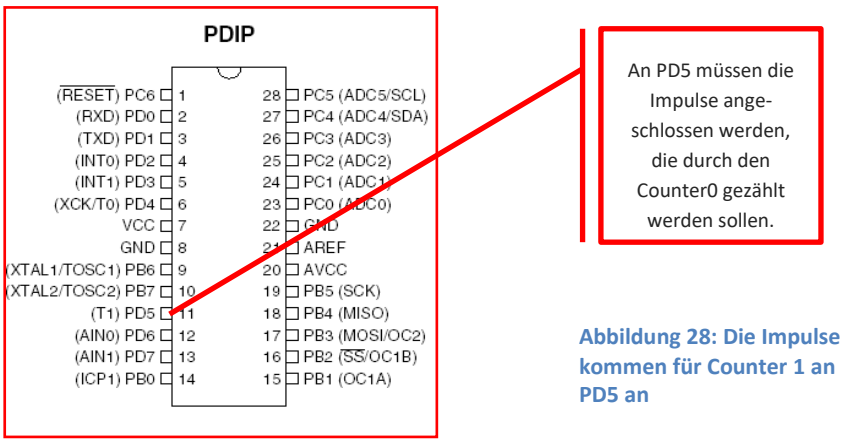



**Abbildung 28: Die Impulse kommen für Counter 1 an PD5 an**

Wird der Zähler 1 benutzt, muss beachtet werden, dass der Zählimpuls in diesem Fall an PD5 angeschlossen werden muss.

Das Register TCCR1B ist das Einstellregister für die Flankensteuerung des Zählimpulses

```
        ldi     r16,0b00000110      ;absteigende Flanke
        out     TCCR1B,r16          ;einstellen
        ldi     r16,0b00000100      ;Interruptauslöser konfigurieren
        out     TIMSK,r16
```

Ein Assemblerprogramm mit Timer 1 zeigt das folgende Listing:

```
;+---------------------------------------------------------------------
;| Title            : Zähler mit Timer 1
;+---------------------------------------------------------------------
;| Funktion         : 10 Impulse werden gezählt und dann die LED
;| Schaltung        : an PORTB umgeschaltet
;+---------------------------------------------------------------------
;| Prozessor        : ATmega8
;| Autor            : Edgar Hoch
;+---------------------------------------------------------------------
.include    "AVR.H"
;---------------------------------------------------------------------
;Reset and Interrupt vector       ;VNr.  Beschreibung
            rjmp    main         ;1    POWER ON RESET
            reti                 ;2    Int0-Interrupt
            reti                 ;3    Int1-Interrupt
            reti                 ;4    TC2 Compare Match
            reti                 ;5    TC2 Overflow
            reti                 ;6    TC1 Capture
            reti                 ;7    TC1 Compare Match A
            reti                 ;8    TC1 Compare Match B
            rjmp    TC1          ;9    TC1 Overflow
            reti                 ;10   TC0 Overflow
            reti                 ;11   SPI, STC Serial Transfer Complete
            reti                 ;12   UART Rx Complete
            reti                 ;13   UART Data Register Empty
            reti                 ;14   UART Tx Complete
            reti                 ;15   ADC Conversion Complete
            reti                 ;16   EEPROM Ready
            reti                 ;17   Analog Comparator
            reti                 ;18   TWI (IC) Serial Interface
            reti                 ;19   Store Program Memory Ready
;---------------------------------------------------------------------
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            sbi     DDRB,0              ;LED Anschluß konfigurieren
            cbi     DDRD,5              ;PROTD4 auf Eingabe stellen
            Sbi     PORTD,5             ;Pullup an PORTD5 setzen
            ldi     r16,0b00000110      ;absteigende Flanke
            out     TCCR1B,r16          ;einstellen
            ldi     r16,0b00000100      ;Interruptauslöser konfigurieren
            out     TIMSK,r16
            ldi     r17, 0xff           ;zur Auslösung des
            out     TCNT1H,r17          ;Interrupts
            ldi     r17,0xf8            ;7 Zählimpulse führen
            out     TCNT1L,r17          ;bei diesem Wert zum Interrupt
            sei                         ;Interrupt generell zulassen
;---------------------------------------------------------------------
mainloop:   wdr
            rjmp    mainloop
;---------------------------------------------------------------------
TC1:        cli
            com     r16                 ;Jedes Bit wird negiert und
            out     PORTB,r16           ;Wert an LED übergeben
            ldi     r17, 0xff           ;zur Auslösung des
            out     TCNT1H,r17          ;Interrupts
            ldi     r17,0xf8            ;7 Zählimpulse führen
            out     TCNT1L,r17          ;bei diesem Wert
            sei
            reti
```

**Achtung: Die Werte für TCNT1H und L müssen Sie in der Reihenfolge: erst das H und dann das L-Register laden. Sonst funktioniert diese Betriebsart nicht!**

Jetzt sind Sie wieder einmal am Ende eines Abschnitts dieses Studienbriefes angekommen. Natürlich konnten wir nur die momentan wichtigsten Funktionen der Timer erklären. Die Timer lassen noch einige weitere Funktionen zu, die den Rahmen dieses Studienbriefes jedoch sprengen würden.

Fassen wir noch einmal das Wichtigste über Timer und Counter zusammen:

### 5.3.4 Zusammenfassung

Z

- Der ATmega8 besitzt insgesamt drei Timer: zwei 8-Bit-Timer und einen 16-Bit-Timer.
- Timer 0 und Timer 1 können auch als Counter eingesetzt werden.
- Die wichtigsten Register sind das TIMSK, das TCCR und das TCNT-Register.
- Im TIMSK wird die Interruptbehandlung festgelegt, im TCCR wird festgelegt, ob als Zähler oder Timer gearbeitet werden soll und im Timer-Betrieb, wie groß die Vorteilung sein soll.
- In TCNT wird der Reinitialisierungswert abgelegt. Bei Timer 1 ist das Register ein 16-Bit-Register, das aus den beiden Teilregistern TCNT1L und TCNT1H besteht.
- Mit dem Timer 0 können durch Teilerfunktionen Werte von 15 Hz erreicht werden. Wird mit dem Timer 1 gearbeitet, dann sind Zeitverzögerungswerte bis ca. 18 Sekunden möglich.

### 5.3.5 Aufgabe

A

Berechnen Sie die Zeitverzögerung, die durch das Laden der Register mit folgenden Werten erreicht werden kann. Die Systemtaktfrequenz soll 12 MHz betragen.

- TCNT1L = 0x23
- TCNT1H =0xf7

### 5.3.6 Aufgabe

A

- Erklären Sie kurz, was im Register TIMSK eingestellt werden kann.
- Erklären Sie kurz, was ein Vorteiler ist und in welchem Register dieser eingestellt werden kann.
- Erklären Sie kurz die Aufgaben des Steuerregisters TCNT0 im Timer 1.

### 5.3.7 Übung

Mit Hilfe der Timer oder des Timers soll eine Frequenz von 1 Hertz erzeugt werden.

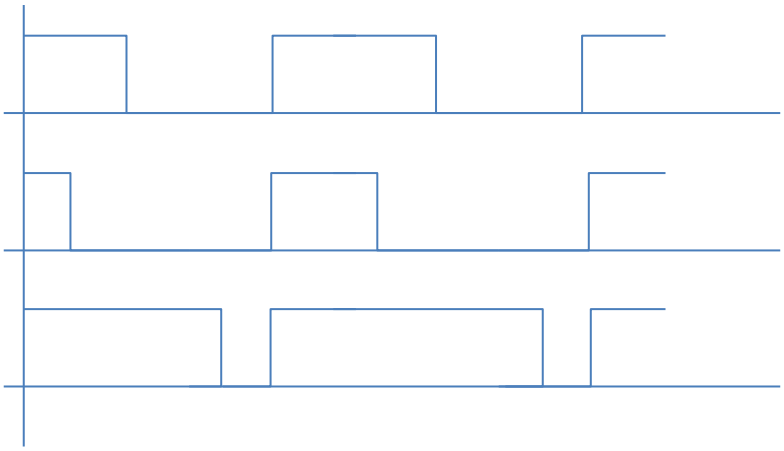
Ü

1. Entscheiden Sie, welchen Timer Sie nehmen und warum.
2. Berechnen Sie die Werte, die für die Register einzusetzen sind.
3. Schreiben Sie das Programm als Assemblerlisting.

# 6 Pulsweitenmodulation

Unter Pulsweitenmodulation versteht man schlicht die Veränderungen der Ein-Ausschaltzeiten eines Signals während einer Periode. Mit einer Veränderung dieser Ein- und Ausschaltzeiten wird die Signal-Zeitfläche vergrößert oder verkleinert.

Nehmen wir an, wir verwenden eine Frequenz von 1 Hz. Damit haben wir im Normalfall eine Einschaltzeit von 0,5 Sekunden und 0,5 Sekunden Ausschaltzeit. Wir sprechen auch von einem Tastgrad von 0,5 (Verhältnis von Einschaltzeit zur Gesamtperiodendauer).



Die Frequenz der Spannung und damit auch die Periodendauer ist in allen Fällen gleich groß.

- Im ersten Impulsdiagramm ist die Einschaltzeit gleich lang wie die Ausschaltzeit des Signals (Tastgrad 0,5).
- Im zweiten Impulsdiagramm ist die Ausschaltzeit größer als die Einschaltzeit (Tastgrad 0,3).
- Und im dritten Impulsdiagramm ist die Einschaltzeit größer als die Ausschaltzeit (Tastgrad 0,8).

Diese Veränderung wird durch die Pulsweitenmodulation erreicht.

## 6.1 Der Timer 1 als PWM

Wenn der Timer/Counter 1 in der PWM-Betriebsart betrieben wird, so bilden das Datenregister **TCNT1H/TCNT1L** und das Vergleichsregister **OCR1H/OCR1L** einen frei laufenden Pulsweitenmodulator, wobei das PWM-Signal an PORT B1 (im Mikrocontroller als OC1 bezeichnet) abgegriffen werden kann.

Das Datenregister **TCNT1H/TCNT1L** wird dabei als Auf-/Ab-Zähler betrieben, der von 0 an aufwärts zählt bis zur Obergrenze und dann wieder zurück auf 0. Die Obergrenze ergibt sich daraus, ob eine 8-, 9- oder 10-Bit-PWM verwendet wird.

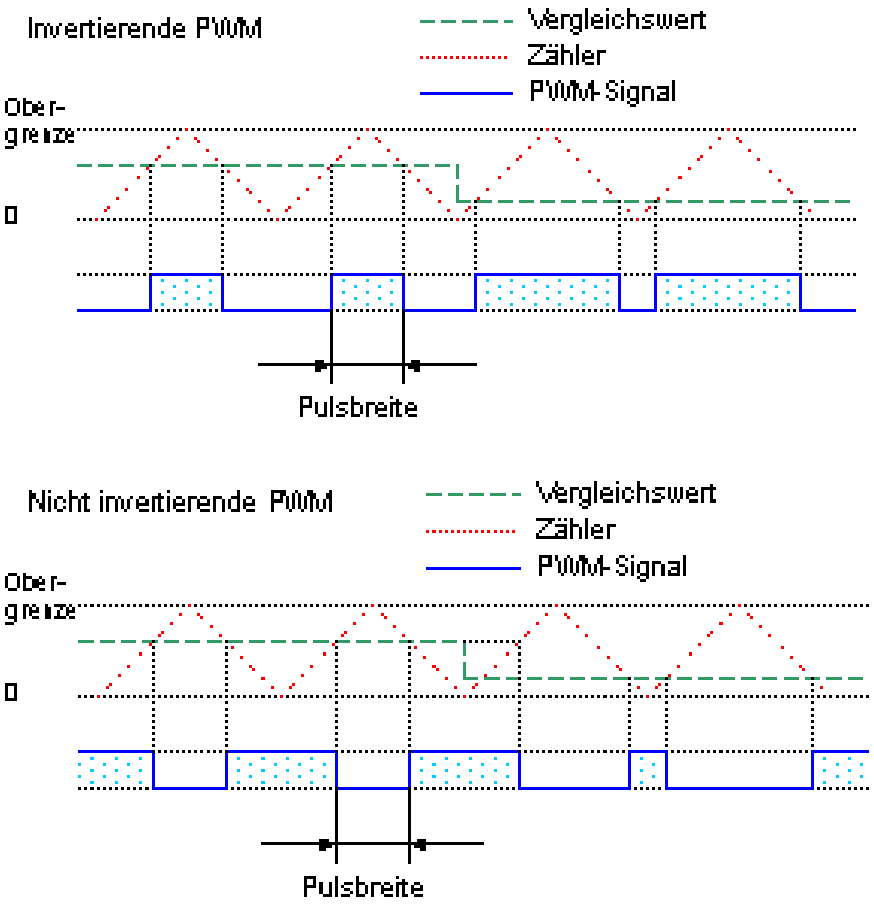



**Abbildung 29: Das Prinzipbild des Pulsweitenmodulators. Die Ein-Ausschaltzeiten werden durch den Vergleichswert bestimmt.**

**a) Invertierende PWM:**
**Ausgang ist 1, wenn der Zählerwert >= Vergleichswert ist,**
**Ausgang ist 0, wenn der Zählerwert < Vergleichswert ist**

**b) Nichtinvertierende PWM:**
**Ausgang ist 0, wenn der Zählerwert >= Vergleichswert ist**
**Ausgang ist 1, wenn der Zählerwert < Vergleichswert ist**

Wird der Vergleichswert nach „unten“ verändert, verändert sich auch die Ein-Ausschaltzeit des Ausgangssignals (im Bild rechts).

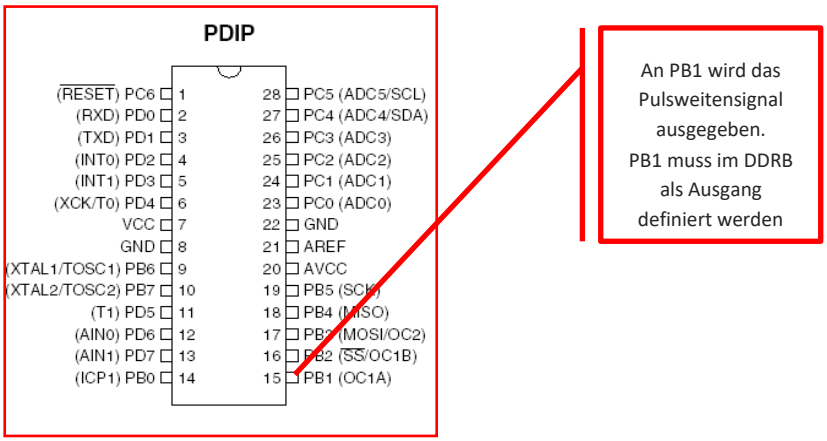



**Abbildung 30:**
**An PB1 wird das PWM-Signal ausgegeben**

### 6.1.1 Einstellung des PWM-Mode im Register TCCR1A und TCCR1B

Die Steuerung der PWM geschieht vornehmlich in den Registern TCCR1A und TCCR1B, das wir anschließend noch genauer betrachten. Zunächst sehen wir uns die einzelnen Bits des Registers TCCR1A einmal etwas näher an:

**TCCR1A - Timer/Counter 1 Control Register A**

In diesem Register stellen wir ein, wie der Timer/Counter verwendet werden soll. Das Register ist wie folgt aufgebaut:

| ***Bit*** | ***7*** | ***6*** | ***5*** | ***4*** | ***3*** | ***2*** | ***1*** | ***0*** |
|---|---|---|---|---|---|---|---|---|
| **Name** | **COM1A1** | **COM1A0** | COM1B1 | COM1B0 | FOC1A | FOC1B | **PWM11** | **PWM10** |
| **R/W** | R/W | R/W | R/W | R/W | R | R | R/W | R/W |
| **Initialwert** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**COM1A1 / COM1A0** – **C**ompare **O**utput **M**ode Control Bits – Channel **A**

Diese 2 Bits bestimmen die Aktion, die am Output-Pin **OC1** ausgeführt werden soll, wenn der Wert des Zählregisters im Timer/Counter 1 den Wert des Vergleichsregisters erreicht, also ein so genannter Compare Match auftritt. Der Pin **OC1** muss im Datenrichtungsregister als Ausgang konfiguriert werden.

| ***COM1A1*** | ***COM1A0*** | ***Resultat*** |
|---|---|---|
| 0 | 0 | Output-Pin **OC1** wird nicht angesteuert. |
| 0 | 1 | Das Signal am Pin **OC1** wird invertiert (Toggle). |
| 1 | 0 | Der Output Pin **OC1** wird auf 0 gesetzt. |
| 1 | 1 | Der Output Pin **OC1** wird auf 1 gesetzt. |

In der PWM-Betriebsart haben diese Bits eine andere Funktion.

| ***COM1A1*** | ***COM1A0*** | ***Resultat*** |
|---|---|---|
| 0 | 0 | Output-Pin **OC1** wird nicht angesteuert. |
| 0 | 1 | Output-Pin **OC1** wird nicht angesteuert. |
| **1** | **0** | **Wird beim Hochzählen der Wert im Vergleichsregister erreicht, so wird der Pin OC1 auf 0 gesetzt.**<br>**Wird beim Herunterzählen der Wert im Vergleichsregister erreicht, so wird der Pin auf 1 gesetzt.**<br>**Man nennt dies nicht invertierende PWM.** |
| 1 | 1 | Wird beim Hochzählen der Wert im Vergleichsregister erreicht so wird der Pin **OC1** auf 1 gesetzt.<br>Wird beim Herunterzählen der Wert im Vergleichsregister erreicht so wird der Pin auf 0 gesetzt.<br>Man nennt dies invertierende PWM. |

**PWM11 / PWM10** - **PWM** Mode Select Bits

Mit diesen 2 Bits wird die PWM-Betriebsart des Timer/Counter 1 gesteuert.

| ***PWM11*** | ***PWM10*** | ***Resultat*** |
|---|---|---|
| 0 | 0 | Die PWM-Betriebsart ist nicht aktiviert.<br>Timer/Counter 1 arbeitet als normaler Timer bzw. Zähler. |
| **0** | **1** | **8-Bit PWM Betriebsart aktivieren.** |
| 1 | 0 | 9-Bit PWM Betriebsart aktivieren. |
| 1 | 1 | 10-Bit PWM Betriebsart aktivieren. |

Mit folgenden Programmzeilen können wir das Register einstellen

```
ldi     r16,0b10000001      ;PWMMode und Ausgabekanal konfig.
out     TCCR1A,r16
```

Durch die Bits 0 und 1 wird also die 8-Bit Betriebsart in PWM eingestellt, durch die Bits 7 und 6 wird folgendes bewirkt:

- Wird beim Hochzählen der Wert im Vergleichsregister erreicht, so wird der Pin OC1 auf 0 gesetzt.
- Wird beim Herunterzählen der Wert im Vergleichsregister erreicht, so wird der Pin OC1 auf 1 gesetzt.

Alle weiteren Bits (**COM1A1 / COM1A0** – **C**ompare **O**utput **M**ode Control Bits – Channel **A sowie FOC1A / FOC1B** – **F**orce **O**utput **C**ompare Channel **A / B** ) werden in diesem Studienbrief nicht benötigt.

### TCCR1B - Timer/Counter 1 Control Register B

Im Register TCCR1B wird vornehmlich die Taktfrequenz der PWM bestimmt.

| ***Bit*** | ***7*** | ***6*** | ***5*** | ***4*** | ***3*** | ***2*** | ***1*** | ***0*** |
|---|---|---|---|---|---|---|---|---|
| **Name** | ICNC1 | ICES1 | - | WGM13 | WGM12 | **CS12** | **CS11** | **CS10** |
| **R/W** | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W |
| **Initialwert** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**CS12-CS10** – **C**lock **S**elect

Diese 3 Bits bestimmen die Quelle für den Timer/Counter:

| ***CS12*** | ***CS11*** | ***CS10*** | ***Resultat*** |
|---|---|---|---|
| 0 | 0 | 0 | Stopp, Der Timer/Counter wird angehalten. |
| 0 | 0 | 1 | CPU-Takt |
| **0** | **1** | **0** | **CPU-Takt / 8** |
| 0 | 1 | 1 | CPU-Takt / 64 |
| 1 | 0 | 0 | CPU-Takt / 256 |
| 1 | 0 | 1 | CPU-Takt / 1024 |
| 1 | 1 | 0 | Externer Pin T0, fallende Flanke |
| 1 | 1 | 1 | Externer Pin T0, steigende Flanke |

Wenn als Quelle der externe Pin T0 verwendet wird, so wird ein Flankenwechsel auch erkannt, wenn der Pin T0 als Ausgang geschaltet ist.

Als letztes brauchen wir noch die Vergleichswerte in den Registern OCR1AH und OCR1AL. Eigentlich haben wir ja eine 8-Bit-PWM-Steuerung ausgewählt und es würde genügen, das niederwertige Register zu bedienen. Allerdings werden diese beiden Register wieder zusammen als 16-Bit-Register angesprochen, deshalb ist es notwendig, beide Register mit den Vergleichswerten zu laden und zwar in der Reihenfolge: Erst das High-Register und dann das Low-Register.

Die Obergrenze ist durch unsere Einstellung auf den Wert 255 (8-Bit) eingestellt. Der Wert im Register OCR1AL bestimmt nun die Schaltwerte. Wird bei-

spielsweise in das Register eine große Zahl geladen, wird die LED heller leuchten. Je kleiner der Wert im OCR1AL ist, umso dunkler wird die LED.

Die PWM läuft in dieser Betriebsart völlig autonom. Wir brauchen die Programmzeilen nur einmal zu definieren und schon kann die PWM benutzt werden. Wir haben im Programm nur noch eine „main“ Definition.

```
main:   ldi     r16,lo8(RAMEND)
        out     SPL,r16
        ldi     r16,hi8(RAMEND)
        out     SPH,r16

        sbi     DDRB,1              ;Bit B1 als Ausgabe

        ldi     r16,0b10000001      ;PWMMode und Ausgabekanal konfig.
        out     TCCR1A,r16
        ldi     r16,0b00000010      ;8Bit PWM, phasenkorr. einstellen
        out     TCCR1B,r16
        ldi     r16,0x00            ;Wert für die Helligkeit
        ldi     r17,0xff            ;Wert für die Helligkeit
        out     OCR1AH,r16          ;Ausgabe an Register
        out     OCR1AL,r17          ;Ausgabe an Register
```

Das R17 bestimmt den Helligkeitsgrad der LED. Testen Sie das Programm und verändern Sie den Wert in R17.

Ü

### 6.1.2 Übung Dimmen

Die Leuchtdiode soll „sanft“ von einer großen Lichtstärke auf eine niedrige Lichtstärke gedimmt werden. Ist die Lichtstärke ganz niedrig, soll das Programm von vorne beginnen. Es wird sich also eine pulsierende Lichtstärke an der LED einstellen.

Verwenden Sie aus dem Grundgerüst das Unterprogramm Wait ms.

### 6.1.3 Zusammenfassung

Z

Für die Pulsweitenmodulation sind mehrere Register zuständig:

- TCCR1A: hier wird der Modus eingestellt – die Auflösung.
- TCCR1B: hier wird die Frequenz eingestellt.
- OCR1AL / OCR1AH: In diesen beiden Registern werden die Vergleichswerte eingestellt. Sie werden immer bedient, auch wenn der Betriebsmodus beispielsweise mit 8 Bit eingestellt ist.

A

### 6.1.4 Aufgabe

Erstellen Sie für eine Pulsweitenmodulation die Werte für die Register TCCR1A und TCCR1B für eine Motorsteuerung mit einer Grundfrequenz von f = Mikrocontrollertakt/1024. Die Auflösung soll auf 10 Bit eingestellt werden.

In den Vergleichsregistern soll ein mittlerer Wert eingestellt werden. Als Kontrollbit wird eine Leuchtdiode an PORTB.1 verwendet.

# 7 Arbeiten mit den Speichern des AVR ATmega8

Will man Daten in den Speicher des Systems ablegen oder von dort Daten auslesen, stehen dem Anwender die Speicher Flash, EEPROM oder SRAM zur Verfügung.

## 7.1 Arbeiten mit dem SRAM (Static RAM)

Das SRAM ist durch die Funktionen der Register R0 - R31 und der I/O-Register teils belegt. Genauer gesagt von den Hexadezimaladressen 0x0000 bis 0x005F.

Der Adressbereich für den Anwender beginnt somit ab der Adresse 0x0060. Dies ist die erste zugängliche Adresse. Hier können Daten abgelegt bzw. von dort gelesen werden. Die letzte erreichbare Speicherstelle ist im ATmega8 die Stelle 0x045F. Das sind somit 1024 frei zugängliche Speicherstellen im SRAM.

Die Übersicht zeigt nochmals die Speicherorganisation.

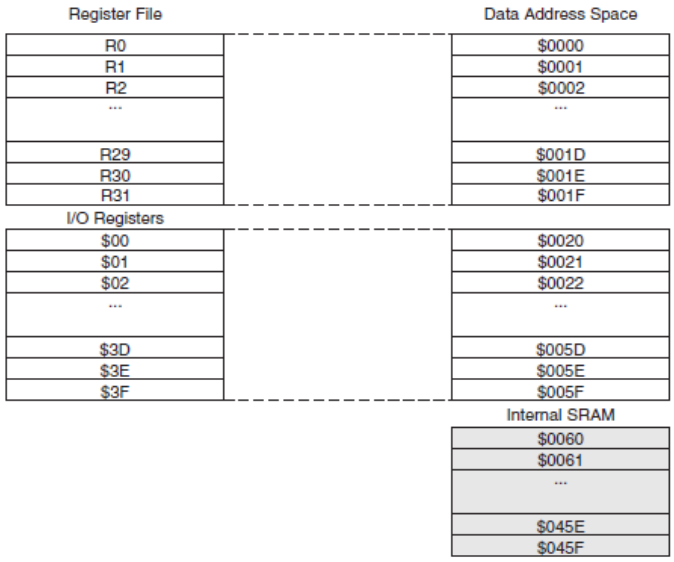



**Abbildung 31: Die Speicherorganisation des SRAMs im ATmega8**

Der Zugriff zu diesen Adressen erfolgt durch indirekte Adressierung.

### 7.1.1 Indirekte Adressierung

Damit ist gemeint, dass die Adresse eines Speicherplatzes nicht im Befehl direkt genannt wird, sondern an einem anderen Platz in Form eines 16-Bit-Zeigers abgelegt ist und dort auch eingestellt werden kann.

Für diese indirekte Adressierung stehen die Registerpaare X, Y und Z zur Verfügung. Sie erinnern sich, im Teil 1 haben wir diesen Registern folgende Register zugewiesen:

Z = R31/R30

Y = R29/R28

X = R27/R26

Diese Register werden nun also zu Registerpaaren zusammengesetzt, wobei das nach Nummer kleinere Register das lower-Byte der Adresse aufnimmt.

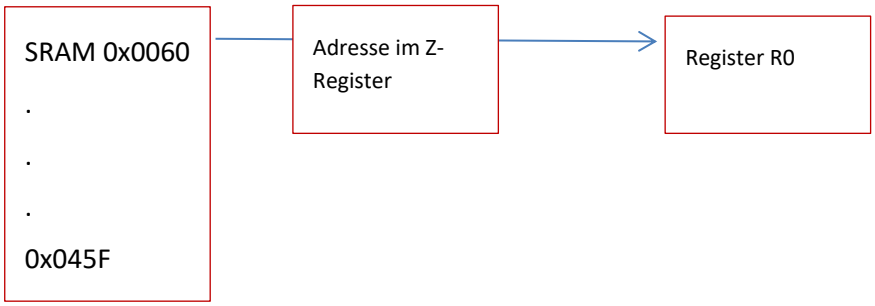



**Abbildung 32: Das Z-Register beinhaltet die Adresse des SRAMs. Diese Adressierung wird indirekte Adressierung genannt**

Welches Registerpaar Sie für die indirekte Adressierung verwenden wollen, ist Ihnen überlassen. Wir schlagen das Z-Register mit den Registern R30 / R31 vor.

Die Startadresse für die Register kann man durch einfache Ladebefehle einstellen. Beispielsweise durch:

```
ldi     R31,0x00            ;höherwertiges Byte in R31
ldi     R30,0x60            ;niederwertiges Byte in R30
```

### 7.1.2 Die Transferbefehle ST und LD

Nun wird ein Transferbefehl gebraucht, der aus der ersten eingestellten SRAM-Adresse den Wert liest oder dorthin speichert. Diese beiden Befehle lauten:

ST Z,Rxx ;Speichert den Inhalt des Registers Rxx in die SRAM-Adresse, die im Register R30/R31 angegeben ist.

LD Rxx,Z ;Lade in das Register Rxx den Wert der SRAM-Adresse, die im Register R39/31 angegeben ist.

Anhand eines Beispiels wird die Funktion deutlicher:

Die Zahlen von 1 bis 5 sollen in den ersten 5 Adressen des SRAM abgelegt werden. Das kann wie folgt geschehen

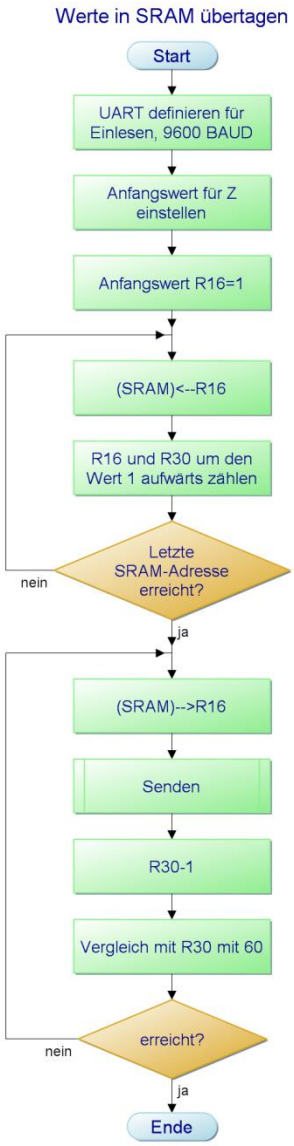



**Abbildung 33: Fünf Adressen des SRAMs werden beschrieben und später wieder ausgelesen**

```
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            ldi     R31,0x00            ;höherwertiges Byte in R31
            ldi     R30,0x60            ;niederwertiges Byte in R30
```

Zunächst wird die Anfangsadresse des SRAM in den Registern R31 und R30 abgelegt. Sie werden fortan als Register Z angesprochen.

In der mainloop wird nun das Register R16 mit dem Wert 1 geladen, der dann in den Speicherplatz im SRAM mit der Adresse 0x0060 übergeben wird.

```
;-----------------------------------------------------------------------
mainloop:   wdr
            ldi     R16,0x01            ;Anfangswert in Register R16
Loop1:      st      Z,R16
```

Ist der Speicherbefehl ausgeführt, dann kann die nächste SRAM-Adresse eingestellt werden, der nächste Wert für das Register 16 kann eingestellt werden und es folgt die Abfrage, ob die letzte Adresse schon bedient wurde, also die

Adresse 0x0065 schon bearbeitet wurde. Wenn nicht, verzweigt das Programm zur Sprungmarke Loop1 und speichert den nächsten Wert.

```
            inc     R30
            cpi     R30,0x66
            brne    Loop1
ende:       rjmp    ende
;-----------------------------------------------------------------------
```

Ist die letzte Adresse „bedient“, endet das Programm in einer „Endlosschleife“.

Leider können wir so nicht verfolgen, ob diese Befehle tatsächlich ausgeführt wurden oder nicht.

**Einbeziehung des USART zur Sichtung der Adressinhalte**

Nachdem die Adressen 0x0060 bis 0x0065 mit den Zahlenwerten 1 bis 6 belegt wurden, können wir diese durch einen zweiten Programmteil auch wieder auslesen und uns im Workpad ansehen.

Das nebenstehende PAP zeigt den Ablauf.

### 7.1.3 Übung

Ü

Erstellen Sie das komplette Assemblerprogramm für den nebenstehenden PAP und überprüfen Sie das Programm mit dem Workpad.

Für den Datentransfer von bzw. zum SRAM stehen noch weitere Befehle zur Verfügung, die sich auf ein automatisches Inkrementieren oder Dekrementieren der Adresse beziehen. Das erleichtert natürlich die Programmierarbeit.

**Die Befehle hierfür lauten:**

| | |
|---|---|
| ST Z+,Rxx | ;Speichert den Inhalt des Registers Rxx in die SRAM-Adresse, die im Register R30/R31 angegeben ist und erhöhe den Wert in R30 um den Wert 1 |
| ST -Z,Rxx | ;Speichert den Inhalt des Registers Rxx in die SRAM-Adresse, die im Register R30/R31 angegeben ist und erniedrige den Wert in R30 um den Wert 1 |
| LD Rxx,Z+ | ;Lade in das Register Rxx den Wert der SRAM-Adresse, die im Register R39/31 angegeben ist und erhöhe den Wert in R30 um den Wert 1 |
| LD Rxx,-Z | ;Lade in das Register Rxx den Wert der SRAM-Adresse, die im Register R39/31 angegeben ist und erniedrige den Wert in R30 um den Wert 1 |

## 7.2 Arbeiten mit dem EEPROM

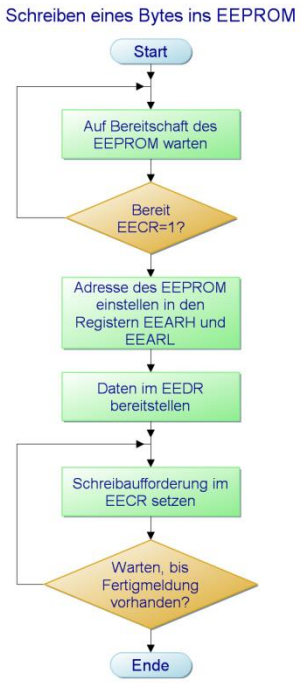



**Abbildung 34: Prinzipieller Ablauf eines Schreibvorgangs in das EEPROM**

Sollen Daten nach dem Abschalten der Betriebsspannung nicht verloren gehen, dann sollte man diese nicht im SRAM aufbewahren, sondern eher im EEPROM. Hier sieht der Datentransfer jedoch etwas anders aus als im SRAM.

Für die Datenübertragung mit dem EEPROM stehen verschiedene Register zur Verfügung. Dies sind die Register EEARH und EEARL, die eine EEPROM-Adresse aufnehmen können.

Weiter gibt es ein Daten Register mit der Bezeichnung EEDR und schließlich noch das Control Register EECR:

- Das Register EECR signalisiert die Bereitschaft des EEPROMs, Daten zu empfangen und Daten zu schreiben.
- Das Datenregister EEDR stellt die Daten für einen Transfer bereit.

Der PAP zeigt die Reihenfolge der Arbeitsschritte zum Schreiben eines Bytes.

Ein Unterprogramm zum Schreiben von Daten aus einem Register ins EEPROM würde dann so aussehen:

```
;-----------Unterprogramm EEPROM schreiben-------------------------------
EEWrite:
            out     EEARH,R17           ;Output address HI
            out     EEARL,R16           ;Output address LO
            out     EEDR,R18            ;Output data
            sbi     EECR,2              ;set EEPROM Write EnabLED
            sbi     EECR,1              ;set EEPROM Write EnabLED
waitEEW:
            sbic    EECR,1              ;wait for Handshake
            rjmp    waitEEW
            ret
```

Die Register R16 und R17 bilden den Pointer (Zeiger) auf die Adresse, mit der im EEPROM gearbeitet – in diesem Fall Daten geschrieben – werden soll.

Das Register R18 enthält die Daten, die Sie ins EEPROM schreiben wollen. Der Schreibvorgang wird durch Setzen der Bits 1 und 2 im Register EECR eingeleitet. Wird das Bit 1 vom Prozessor rückgesetzt, ist der Schreibvorgang beendet.

Ein Unterprogramm zum Lesen von Daten aus dem EEPROM in ein Register würde beispielsweise so aussehen:

```
;-----------Unterprogramm EEPROM lesen-----------------------------------
EERead:
            out     EEARH,R17           ;Adresse
            out     EEARL,R16           ;Adresse
            sbi     EECR,0              ;Leseaufforderung
waitEER:
            sbic    EECR,0              ;warte auf Fertigmeldung
            rjmp    waitEER
            in      R18,EEDR            ;Daten übernehmen
            ret
```

Auch in diesem Unterprogramm werden die Register R16 und R17 zur indirekten Adressierung benutzt. Durch den Befehl **sbi** EECR,0 wird der Lesevorgang eingeleitet. Wird dieses Bit vom Prozessor auf 0 gesetzt, ist der Lesevorgang abgeschlossen.

Für Steuerungszwecke kann man Datenstrukturen im EEPROM sehr gut ablegen und auch sehr schnell wieder ändern. Das Entwicklungsboard bietet hierfür nicht sehr viele Möglichkeiten – und doch, eine kleine Ampelschaltung mit drei Leuchtdioden können Sie damit schon programmieren.

Dazu verbinden Sie die drei Leuchtdioden mit dem PORTC des Boards und zwar die drei niederwertigsten Stellen (0:grün, 1:gelb, 2:rot).

Das Bitmuster für die Ampelschaltung:

| Zyklus | LED ROT | LED GELB | LED GRÜN |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 2 | 1 | 1 | 0 |
| 3 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 |
| | 1 | 0 | 0 |

Mit 4 Zyklen kommt man für diese kleine Steuerung aus. Es gilt nun, diese Werte ins EEPROM und zwar in die ersten 4 Adressen abzuspeichern. Das könnte wie folgt geschehen:

```
            ldi     R16,0x00            ;Anfangsadresse EEPROM
            ldi     R17,0x00            ;Anfangsadresse EEPROM
            ldi     R18,0x04            ;erster Wert für EEPROM
            rcall   EEWrite
            inc     R16                 ;nächste Adresse im EEPROM
            ldi     R18,0x06            ;nächster Wert für EEPROM
            rcall   EEWrite
            inc     R16                 ;nächste Adresse im EEPROM
            ldi     R18,0x01            ;nächster Wert für EEPROM
            rcall   EEWrite
            inc     R16                 ;nächste Adresse im EEPROM
            ldi     R18,0x02            ;nächster Wert für EEPROM
            rcall   EEWrite
```

Zugegeben, nicht das eleganteste Programm, aber für diese kleine Steuerungsaufgabe doch vertretbar. Die Konstanten, die im EEPROM abgelegt werden sollen, ergeben sich aus der Tabelle.

Nun fehlt nur noch ein kleines Zeitverzögerungsprogramm, das Sie aber am Anfang dieses Studienbriefes schon geschrieben haben, und das wir wieder verwenden wollen.

```
;-----------Unterprogramm Zeit-----------------------------------------
Zeit:       push    r16                 ;r16 retten
            push    r17                 ;r17 retten
            push    r18                 ;r18 retten
            ldi     r16,20              ;Laufvariable
loop1:      ldi     r17,255             ;Laufvariable loop1
loop2:      ldi     r18,255             ;Laufvariable loop3
loop3:      dec     r18                 ;Zähler 3
            brne    loop3               ;Solange nicht NULL
            dec     r17                 ;Zähler 2
            brne    loop2               ;Solange nicht NULL
            dec     r16                 ;Zähler1
            brne    loop1               ;Solange nicht NULL
            pop     r18                 ;r18 wiederherstellen
            pop     r17                 ;r17 wiederherstellen
            pop     r16                 ;r16 wiederherstellen
            ret                         ;Rücksprung
;----------------------------------------------------------------------
```

Das Programm ist als Unterprogramm angelegt. Mit den Push-Befehlen werden die Inhalte der Register auf den Stack gerettet und am Schluss des Unterprogramms wieder von dort zurückgeholt. So wird das Unterprogramm recht universell einsetzbar, weil sich der Programmierer nicht um die eventuell woanders verwendeten Register kümmern muss. Die Werte werden zwar überschrieben, nach Abarbeitung des Unterprogramms ist alles wieder so hergestellt wie vor dem Unterprogrammablauf.

Das Auslesen und die Ausgabe der Werte aus dem EEPROM auf PORTC erledigt dieser Programmteil:

```
;-----------Ausleseteil-----------------------------------------------
Anfang:     ldi     R16,0x00            ;Anfangsadr EEPROM
Marke1:     rcall   EERead              ;Wert aus 1. Adresse lesen
            out     PORTC,R18           ;Ausgabe Wert R18 an PORTC
            rcall   Zeit                ;Zeitverzögerung ca. 1 Sekunde
            inc     R16                 ;nächste Adresse einstellen
            cpi     R16,4               ;letzte Adresse erreicht?
            brne    Marke1              ;nein dann weiter bei Marke1
            rjmp    Anfang
```

Das vollständige Programm können Sie in Ihr AVR-Board eingeben und testen.

```
;+--------------------------------------------------------------------
;| Title            : EEPROM als Steuerung
;+--------------------------------------------------------------------
;| Prozessor        : ATmega8
;| Takt             : 3,6864 MHz
;| Sprache          : Assembler
;| Autor            : Edgar Hoch
;+--------------------------------------------------------------------
.include    "AVR.H"
;---------------------------------------------------------------------------
;Reset and Interrupt vector          ;VNr.  Beschreibung
            rjmp    main            ;1    POWER ON RESET
            reti                    ;2    Int0-Interrupt
            reti                    ;3    Int1-Interrupt
            reti                    ;4    TC2 Compare Match
            reti                    ;5    TC2 Overflow
            reti                    ;6    TC1 Capture
            reti                    ;7    TC1 Compare Match A
            reti                    ;8    TC1 Compare Match B
            reti                    ;9    TC1 Overflow
            reti                    ;10   TC0 Overflow
            reti                    ;11   SPI, STC Serial Transfer Compl.
            reti                    ;12   UART Rx Complete
            reti                    ;13   UART Data Register Empty
            reti                    ;14   UART Tx Complete
            reti                    ;15   ADC Conversion Complete
            reti                    ;16   EEPROM Ready
            reti                    ;17   Analog Comparator
            reti                    ;18   TWI (IC) Serial Interface
            reti                    ;19   Store Program Memory Ready
;---------------------------------------------------------------------------
;Start, Power ON, Reset
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            ldi     R16,0xff        ; PORTC auf Ausgabe
            out     DDRC,r16
```

```
;-----------Einleseteil-----------------------------------------------
mainloop:   wdr
            ldi     R16,0x00        ;Anfangsadr EEPROM
            ldi     R17,0x00        ;Anfangsadr EEPROM
            ldi     R18,0x04        ;erster Wert für EEPROM
            rcall   EEWrite
            inc     R16             ;nächste Adresse im EEPROM
            ldi     R18,0x06        ;nächster Wert für EEPROM
            rcall   EEWrite
            inc     R16             ;nächste Adresse im EEPROM
            ldi     R18,0x01        ;nächster Wert für EEPROM
            rcall   EEWrite
            inc     R16             ;nächste Adresse im EEPROM
            ldi     R18,0x02        ;nächster Wert für EEPROM
            rcall   EEWrite
;-----------Ausleseteil-----------------------------------------------
Anfang:     ldi     R16,0x00        ;Anfangsadresse EEPROM
Marke1:     rcall   EERead          ;Wert aus 1. Adresse lesen
            out     PORTC,R18       ;Ausgabe Wert R18 an PORTC
            rcall   Zeit            ;Zeitverzögerung ca. 1 Sekunde
            inc     R16             ;nächste Adresse einstellen
            cpi     R16,4           ;letzte Adresse erreicht?
            brne    Marke1          ;nein dann weiter bei Marke1
            rjmp    Anfang
            rjmp    mainloop
;---------------------------------------------------------------------
;-----------Unterprogramm EEPROM schreiben--------------------------
EEWrite:
            out     EEARH,R17       ;Output address
            out     EEARL,R16       ;Output address
            out     EEDR,R18        ;Output data
            sbi     EECR,2          ;set EEPROM Write EnabLED
            sbi     EECR,1          ;set EEPROM Write EnabLED
waitEEW:
            sbic    EECR,1          ;wait for Handshake
            rjmp    waitEEW
            ret
;---------------------------------------------------------------------
;-----------Unterprogramm EEPROM lesen------------------------------
EERead:
            out     EEARH,R17       ;Adresse
            out     EEARL,R16       ;Adresse
            sbi     EECR,0          ;Leseaufforderung
waitEER:
            sbic    EECR,0          ;warte auf Fertigmeldung
            rjmp    waitEER
            in      R18,EEDR        ;Daten übernehmen
            ret
;---------------------------------------------------------------------
;-----------Unterprogramm Zeit----------------------------------------
Zeit:       push    r16             ;r16 retten
            push    r17             ;r17 retten
            push    r18             ;r18 retten
            ldi     r16,20          ;Laufvariable
loop1:      ldi     r17,255         ;Laufvariable loop1
loop2:      ldi     r18,255         ;Laufvariable loop3
loop3:      dec     r18             ;Zähler 3
            brne    loop3           ;Solange nicht NULL
            dec     r17             ;Zähler 2
            brne    loop2           ;Solange nicht NULL
            dec     r16             ;Zähler1
            brne    loop1           ;Solange nicht NULL
            pop     r18             ;r18 wiederherstellen
            pop     r17             ;r17 wiederherstellen
            pop     r16             ;r16 wiederherstellen
            ret                     ;Rücksprung
;---------------------------------------------------------------------
```

Z

### 7.2.1 Zusammenfassung

- Sollen Daten im Speicher des ATmega8 für Steuerungszwecke aufbereitet werden, stehen EEPROM und SRAM zur Verfügung.
- Der Umgang mit dem SRAM ist recht einfach und ist eine typische Variante der Indirekten Adressierung.
- Bei der indirekten Adressierung wird die Adresse des Speicherplatzes in der Regel in zwei Registern bereitgehalten. In zwei Registern deshalb, weil die Adresse 16 Bit umfasst.
- Für die indirekte Adressierung stehen die Registerpaare X, Y und Z bereit. Das sind die Register mit den Bezeichnungen R31 bis R26.

# 8 Lösung der Aufgaben und Übungen

## 8.1 Lösung zu Übung 1.5.1

```
;+----------------------------------------------------------------------
;| Title            : Übung 1 Taster steuert Fensterheber
;+----------------------------------------------------------------------
;| Funktion         : ...
;| Schaltung        : ...
;+----------------------------------------------------------------------
;| Prozessor        : ATmega8
;| Takt             : 3,6864 MHz
;| Sprache          : Assembler
;| Datum            : 1.7.2011
;| Version          : ...
;| Autor            : Edgar Hoch
;+----------------------------------------------------------------------
.include    "AVR.H"
;----------------------------------------------------------------------
;Reset and Interrupt vector             ;VNr.  Beschreibung
            rjmp    main                ;1   POWER ON RESET
            rjmp    onint0              ;2   Int0-Interrupt
            rjmp    onint1              ;3   Int1-Interrupt
            reti                        ;4   TC2 Compare Match
            reti                        ;5   TC2 Overflow
            reti                        ;6   TC1 Capture
            reti                        ;7   TC1 Compare Match A
            reti                        ;8   TC1 Compare Match B
            reti                        ;9   TC1 Overflow
            reti                        ;10  TC0 Overflow
            reti                        ;11  SPI, STC Serial Transfer Complete
            reti                        ;12  UART Rx Complete
            reti                        ;13  UART Data Register Empty
            reti                        ;14  UART Tx Complete
            reti                        ;15  ADC Conversion Complete
            reti                        ;16  EEPROM Ready
            reti                        ;17  Analog Comparator
            reti                        ;18  TWI (IC) Serial Interface
            reti                        ;19  Store Program Memory Ready
;----------------------------------------------------------------------

main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            cbi     DDRD,2              ;Datenrichtung an D2 Eingabe
            sbi     PORTD,2             ;Pullup-Widerstand für D2
            cbi     DDRD,3              ;Datenrichtung an D3 Eingabe
            sbi     PORTD,3             ;Pullup-Widerstand für D3
            sbi     DDRC,0              ;Datenrichtung an C0 Ausgabe
            ldi     R16,0b11000000      ;Bit 6/7 aktivieren INT0/INT1
            out     GICR, R16           ;Übergabe in GICR
            ldi     R16,0b00001010      ;10 aktiviertabfallende Flanken
            out     MCUCR,R16           ;und Übergabe an MCUCR
            sei                         ;Interrupt generell erlauben

;----------------------------------------------------------------------
mainloop:   wdr
            ;Hier den Quellcode eintragen.
            rjmp    mainloop
;----------------------------------------------------------------------

onint0:     sbi     PORTC,0             ;LED wird aktiviert
            reti
;----------------------------------------------------------------------

onint1:     cbi     PORTC,0             ;LED wird deaktiviert
            reti
;----------------------------------------------------------------------
```

## 8.2 Lösung der Aufgabe 1.3.4

Folgende Schritte sind notwendig:

- PORTD,2 muss auf Eingabe und der Pullup-Widerstand aktiviert sein.
- Im MCUCR muss die Flankenart eingestellt werden und im GICR muss der Interrupt generiert (aktiviert) werden.
- In der Interrupt-Verktor-Tabelle müssen die Sprungmarken der Interrupt-Service-Routinen eingetragen werden und reti gegen rjmp ausgetauscht werden.
- Interrupts müssen durch **sei** generell zugelassen werden.
- Im Programm wurde vergessen, den Pullup-Widerstand für den PORTD3 zu aktivieren. Deshalb kommt es an PORTD3 zu undefinierten Signalen. Der Mikrocontroller reagiert scheinbar unkontrolliert.

## 8.3 Lösung der Übung 2.2.5

```
;+---------------------------------------------------------------------
;| Title            : Übung 2 Interrupt mit Textausgaben
;+---------------------------------------------------------------------
;| Funktion         : Taster 1 und Taster 2 steuern Text
;+---------------------------------------------------------------------
;| Prozessor        : ATmega8
;| Takt             : 3,6864 MHz
;| Autor            : Edgar Hoch
;+---------------------------------------------------------------------
.include    "AVR.H"
;---------------------------------------------------------------------
;Reset and Interrupt vector              ;VNr.  Beschreibung
            rjmp    main                ;1    POWER ON RESET
            rjmp    onint0              ;2    Int0-Interrupt
            rjmp    onint1              ;3    Int1-Interrupt
            reti                        ;4    TC2 Compare Match
            reti                        ;5    TC2 Overflow
            reti                        ;6    TC1 Capture
            reti                        ;7    TC1 Compare Match A
            reti                        ;8    TC1 Compare Match B
            reti                        ;9    TC1 Overflow
            reti                        ;10   TC0 Overflow
            reti                        ;11   SPI, STC Serial
            reti                        ;12   UART Rx Complete
            reti                        ;13   UART Data Register Empty
            reti                        ;14   UART Tx Complete
            reti                        ;15   ADC Conversion Complete
            reti                        ;16   EEPROM Ready
            reti                        ;17   Analog Comparator
            reti                        ;18   TWI (IC) Serial Interface
            reti                        ;19   Store Program Memory Ready
;---------------------------------------------------------------------
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            cbi     DDRD,2              ;Datenrichtung an D2 Eingabe
            sbi     PORTD,2             ;Pullup-Widerstand für D2
            cbi     DDRD,3              ;Datenrichtung an D3 Eingabe
            sbi     PORTD,3             ;Pullup-Widerstand für D3
            sbi     DDRC,0              ;Datenrichtung an C0 Ausgabe
            ldi     R16,0b11000000      ;Bit 6/7 aktivieren INT0/INT1
            out     GICR,R16            ;Übergabe in GICR
            ldi     R16,0b00001010      ;10 aktiviert fallende Flanken
```

```
            out     MCUCR,R16           ;und Übergabe an MCUCR
            ldi     r16,23              ;Wert für 9600 Baud
            out     UBRRL,R16           ;Übergabe an das Steuerregister
            sbi     UCSRB,3             ;Transmit einschalten
            sei                         ;Interrupt generell erlauben
;---------------------------------------------------------------------
mainloop:   wdr
            rjmp    mainloop
;---------------------------------------------------------------------
onint0:     cli                         ;Interrupt sperren
            sbi     PORTC,0             ;LED einschalten
            ldi     R16,'M'             ;Text für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'o'             ;Text für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'t'             ;Text für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'o'             ;Text für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'r'             ;Byte für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,' '             ;Byte für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'e'             ;Byte für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'i'             ;Byte für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'n'             ;Byte für Motor ein
            rcall   SENDE               ;Byte senden
            sei                         ;Interrupt zulassen
            reti
;---------------------------------------------------------------------
onint1:     cli                         ;Interrupt sperren
            cbi     PORTC,0             ;LED einschalten
            ldi     R16,'M'             ;Text für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'o'             ;Text für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'t'             ;Text für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'o'             ;Text für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'r'             ;Byte für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,' '             ;Byte für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'a'             ;Byte für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'u'             ;Byte für Motor ein
            rcall   SENDE               ;Byte senden
            ldi     R16,'s'             ;Byte für Motor ein
            rcall   SENDE               ;Byte senden
            sei                         ;Interrupt zulassen
            reti
;---------------------------------------------------------------------
SENDE:      cli                         ;Interrupt sperren
            sbis    UCSRA,5             ;warten bis UDR leer
            rjmp    SENDE
            out     UDR,R16             ;Byte senden
            sei                         ;Interrupt zulassen
            ret
```

Anmerkung: Es kann durchaus sein, dass die Taste des Entwicklungsboards etwas prellen und Texte deshalb zweimal versendet werden.

## 8.4 Lösung der Aufgabe 2.3.2

Erklären Sie kurz, wie Sie eine Baudrate von 9600 Baud einstellen, wenn der Systemtakt 6MHz beträgt.

Zunächst wird die Konstante für das Register UBRRL berechnet. Es kann die Formel

$$UBRR = \frac{Systemfrequenz}{16 \cdot BAUD} - 1$$

verwendet werden. Eingesetzt ergibt sich folgender Wert für das Register.

$$UBRR = \frac{6000000}{16 \cdot 9600} - 1 = 38$$

Das Register kann durch den out-Befehl angesprochen bzw. geladen werden

Die Konstante wird zuvor in ein Register geladen

Woran kann erkannt werden, ob ein gesendetes Byte schon übertragen wurde oder nicht?

Das Bit 6 TXC im UCSRA-Register zeigt das an. Es kann zyklisch abgefragt werden. Alternativ kann das Bit 5 abgefragt werden. Dieses Bit zeigt an, ob das Datenregister UDR leer ist oder nicht.

Woran kann erkannt werden, ob ein empfangenes Byte komplett ist oder noch nicht?

Das Bit 7 RXC im UCSRA-Register zeigt das an.

Wie kann der Empfangsteil und wie kann der Sendeteil im USART aktiviert werden.

Das geschieht durch die Bits im UCSRB. Speziell das Bit 3 ist für das Senden, das Bit 4 für das Empfangen zuständig. Ein 1-Signal an diesen Stellen aktiviert die beiden Funktionen.

**Lösung der Aufgabe 3.2.7**

Welche Betriebsarten des AD-Wandlers eines AVR-Controllers kennen Sie?

- Der AD-Wandler kann per Polling oder Interrupt gesteuert werden.
- Dabei kann der AD-Wandler im Einzelschrittbetrieb oder vollständig parallel arbeiten.

Wie erfolgt die Initialisierung des AD-Wandlers im Prinzip?

- Kanal auswählen
- Betriebsart auswählen
- Vorteiler für Geschwindigkeit festlegen
- Interrupt erlauben
- Sample starten

Wie wird das Einlesen eines Analogwertes im Prinzip Programmiert?

- AD-Wandler muss initialisiert sein
- Sample starten
- Sample-Time abwarten (polling oder interrupt)
- Wert auslesen

## 8.5 Lösung zur Übung 3.3.3

```
;+----------------------------------------------------------------------
;| Title            : Lösung zur Übung 3.3.3
;+----------------------------------------------------------------------
;| Prozessor        : ATmega8 / Takt             : 3,6864 MHz
;+----------------------------------------------------------------------
.include    "AVR.H"
;----------------------------------------------------------------------
;Reset and Interrupt vector              ;VNr.  Beschreibung
            rjmp    main                ;1   POWER ON RESET
            reti                        ;2   Int0-Interrupt
            reti                        ;3   Int1-Interrupt
            reti                        ;4   TC2 Compare Match
            reti                        ;5   TC2 Overflow
            reti                        ;6   TC1 Capture
            reti                        ;7   TC1 Compare Match A
            reti                        ;8   TC1 Compare Match B
            reti                        ;9   TC1 Overflow
            reti                        ;10  TC0 Overflow
            reti                        ;11  SPI, STC Serial Transfer
            reti                        ;12  UART Rx Complete
            reti                        ;13  UART Data Register Empty
            reti                        ;14  UART Tx Complete
            reti                        ;15  ADC Conversion Complete
            reti                        ;16  EEPROM Ready
            rjmp    oncomp              ;17  Analog Comparator
            reti                        ;18  TWI (IC) Serial Interface
            reti                        ;19  Store Program Memory Ready
;----------------------------------------------------------------------
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            CBI     ACSR, 7             ;Comparator einschalten
            cbi     ACSR, 6             ;ext. Referenzspannung an AIN0
            ldi     r16,0b00001111      ;ACSR initialisieren
            out     ACSR,r16
            sbi     DDRB,0              ;LED-Anschluss
            sei                         ;Interrupts zulassen
;----------------------------------------------------------------------
mainloop:   wdr
            rjmp    mainloop
;----------------------------------------------------------------------
oncomp:     cli                         ;Interrupts sperren
            cbi     PORTB,0             ;LED ausschalten
            sbis    ACSR,5              ;Stelle 5 im ACSR eintscheidet
            rjmp    oncomp              ;ob es zurück zur mainloop geht
            sbi     PORTB,0             ;oder die LED eingeschaltet wird.
            sei                         ;Interrupts zulassen
            reti
```

## 8.6 Lösung der Aufgabe 3.3.6

Was ist ein Komparator?

Zwei „beliebige“ analoge Spannungen können mit einem Komparator verglichen werden. Als Ergebnis erhält man, welche der beiden Spannungen die größere ist.

Wie funktioniert ein Komparator?

Der Komparator vergleicht die Vergleichsspannung (IST) mit der Referenzspannung (SOLL). Ist die Vergleichsspannung kleiner als die Referenzspannung, liefert der Komparator das Ergebnis logisch 0. Liegt sie über der Referenzspannung liefert der Komparator das Ergebnis logisch 1.

Welche unterschiedlichen Interrupts kann der Komparator des AVR-Controllers auslösen?

Der Komparator eines AVR-Controllers kann folgende Interrupts auslösen:

- steigende Flanke
- fallende Flanke
- Low-Pegel

## 8.7 Lösung der Aufgabe 5.3.3

Berechnen Sie die Zeitverzögerung, die durch das Laden der Register mit den Werten

TCNT1L = 0x23

TCNT1H =0xf7

erreicht werden kann. Die Systemtaktfrequenz soll 12 MHz betragen

Der Gesamtwert aus den beiden Registern beträgt in einen Dezimalwert umgerechnet

63267

$$f = \frac{Systemtakt}{Teilerverhältnis} = \frac{12\ MHz}{1024} = 11{,}718\ kHz$$

$$f_{\min} = \frac{f}{256} = \frac{11{,}718\ kHz}{63267} = 0{,}185\ Hz$$

$$T = \frac{1}{f} = \frac{1}{0{,}17\ Hz} = 5{,}34\ s$$

## 8.8 Lösung der Aufgabe 5.3.6

Erklären Sie kurz, was im Register TIMSK eingestellt werden kann

In TIMSK wird die Interrupt-Behandlung festgelegt.

Erklären Sie kurz, was ein Vorteiler ist und in welchem Register dieser eingestellt werden kann

Der Vorteiler teilt die Systemfrequenz durch die Werte 1,8,64,256 oder 1024. Diese Werte sind an den einzelnen Bits des Registers TCCR0 oder beim Timer 1 im Register TCCR1B einzustellen.

Erklären Sie kurz die Aufgaben des Registers TCNT0 im Timer0

In diesem Register wird der Anfangswert des eigentlichen Zählers festgelegt. Je höher der Wert ist, umso schneller wird der Wert 255 erreicht. Mit dem nächsten Impuls wird dann ein Interrupt ausgelöst.

## 8.9 Lösung zur Übung 5.3.7

Mit Hilfe der Timer oder des Timers soll eine Frequenz von 1 Hertz erzeugt werden.

1. Entscheiden Sie, welchen Timer Sie nehmen und warum
2. Berechnen Sie die Werte, die für die Register einzusetzen sind
3. Schreiben Sie das Programm als Assemblerlisting.

Der Timer 1 wird verwendet, weil Timer 0 keine so großen Werte erlaubt.

Der Reinitialisierungswert für die beiden Register ist wie folgt zu berechnen:

$$f = \frac{Systemtakt}{Teilerverhältnis} = \frac{3{,}6864\ MHz}{1024} = 3{,}6\ kHz$$

$$Teilerwert = \frac{f}{fsoll} = \frac{3{,}6\ kHz}{1Hz} = 3600$$

Dieser Wert ist in zwei Register aufzuteilen. Am einfachsten ist das, wenn man den Dezimalwert in einen Hexadezimalwert umrechnet. Das ergibt den Wert:

0x0E0A

Das Assemblerlisting sieht so aus:

```
;+----------------------------------------------------------------------
;| Title            :Übung 5-3-7
;+----------------------------------------------------------------------
;| Funktion         : Zeitverzögerung 1 Sekunde
;| Schaltung        : ...
;+----------------------------------------------------------------------
;| Prozessor        : ATmega8
;| Takt             : 3,6864 MHz
;| Sprache          : Assembler
;| Datum            : ...
;| Version          : ...
;| Autor            : Edgar Hoch
;+----------------------------------------------------------------------
.include    "AVR.H"
;----------------------------------------------------------------------
;Reset and Interrupt vector       ;VNr.  Beschreibung
            rjmp    main         ;1    POWER ON RESET
            reti                 ;2    Int0-Interrupt
            reti                 ;3    Int1-Interrupt
```

```
            reti                    ;4   TC2 Compare Match
            reti                    ;5   TC2 Overflow
            reti                    ;6   TC1 Capture
            reti                    ;7   TC1 Compare Match A
            reti                    ;8   TC1 Compare Match B
            rjmp    TC1             ;9   TC1 Overflow
            reti                    ;10  TC0 Overflow
            reti                    ;11  SPI, STC Serial Transfer Complete
            reti                    ;12  UART Rx Complete
            reti                    ;13  UART Data Register Empty
            reti                    ;14  UART Tx Complete
            reti                    ;15  ADC Conversion Complete
            reti                    ;16  EEPROM Ready
            reti                    ;17  Analog Comparator
            reti                    ;18  TWI (IC) Serial Interface
            reti                    ;19  Store Program Memory Ready
;-----------------------------------------------------------------------
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            sbi     DDRB,0              ;LED Anschluß konfigurieren
            ldi     r16,0b00000101      ;maximal einstellbarer
            out     TCCR1B,r16          ;Vorteiler in TCCR1B
            ldi     r16,0b00000100      ;Interruptauslöser konfigurieren
            out     TIMSK,r16
            sei                         ;Interrupt generell zulassen
;-----------------------------------------------------------------------
mainloop:   wdr
            rjmp    mainloop
;-----------------------------------------------------------------------
TC1:        com     r16                 ;Jedes Bit wird negiert und
            out     PORTB,r16           ;Wert an LED übergeben
            ldi     r17,0x0A            ;Maximalwert für Timer reinit.
            out     TCNT1L,r17          ;Niederwertige Hälfte
            ldi     r17,0x0E            ;Maximalwert für Timer reinit.
            out     TCNT1H,r17          ;Höherwertige Hälfte
            reti
```

Hinweis: Eventuell könnte man mit einem kleineren Vorteilerverhältnis noch genauere Werte erhalten. Das muss man jedoch von Fall zu Fall austesten, weil die Werte ja nur Ganzzahlwerte sein können. Einen Rundungsfehler wird man allerdings immer machen.

## 8.10 Lösung zur Übung 6.1.2

Die Leuchtdiode soll „sanft“ von einer großen Lichtstärke auf eine niedrige Lichtstärke gedimmt werden. Ist die Lichtstärke ganz niedrig, soll das Programm wieder von vorne beginnen. Es wird sich also eine pulsierende Lichtstärke an der LED verfolgen lassen.

Verwenden Sie aus dem Grundgerüst das Unterprogramm Wait ms.

```
;+---------------------------------------------------------------------
;| Title            : Gedimmtes Licht Übung 5-3-8
;+---------------------------------------------------------------------
;| Funktion         : Helligkeitssteuerung
;| Schaltung        : ...
;+---------------------------------------------------------------------
;| Prozessor        : ATmega8
;| Takt             : 3,6864 MHz
;| Sprache          : Assembler
;| Datum            : ...
```

```
;| Version          : ...
;| Autor            : Edgar Hoch
;+---------------------------------------------------------------------
.include    "AVR.H"
;---------------------------------------------------------------------
;Reset and Interrupt vector       ;VNr.  Beschreibung
            rjmp    main         ;1    POWER ON RESET
            reti                 ;2    Int0-Interrupt
            reti                 ;3    Int1-Interrupt
            reti                 ;4    TC2 Compare Match
            reti                 ;5    TC2 Overflow
            reti                 ;6    TC1 Capture
            reti                 ;7    TC1 Compare Match A
            reti                 ;8    TC1 Compare Match B
            reti                 ;9    TC1 Overflow
            reti                 ;10   TC0 Overflow
            reti                 ;11   SPI, STC Serial Transfer Complete
            reti                 ;12   UART Rx Complete
            reti                 ;13   UART Data Register Empty
            reti                 ;14   UART Tx Complete
            reti                 ;15   ADC Conversion Complete
            reti                 ;16   EEPROM Ready
            reti                 ;17   Analog Comparator
            reti                 ;18   TWI (IC) Serial Interface
            reti                 ;19   Store Program Memory Ready
;---------------------------------------------------------------------
;Start, Power ON, Reset
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            sbi     DDRB,1           ;Port B1 als Ausgabe
            ldi     r16,0b10000001   ;PWM-Mode und Ausgabekanal
            out     TCCR1A,r16
            ldi     r16,0b00000010   ;8Bit PWM, phasenkorr. einstellen
            out     TCCR1B,r16
            ldi     r16,0x00         ;Wert für die Helligkeit
            ldi     r17,0xff         ;Wert für die Helligkeit
;---------------------------------------------------------------------
mainloop:   wdr
            rcall   waitMs
            out     OCR1AH,r16    ;Ausgabe an Register
            out     OCR1AL,r17    ;Ausgabe an Register
            dec     r17
            rjmp    mainloop
;---------------------------------------------------------------------
; UP WaitMs, Warteroutine im Millisekundenbereich
waitMs:     push    r16           ;r16 retten
            push    r17           ;r17 retten
            push    r18           ;r18 retten
            ldi     r16,1         ;Laufvariable ca x ms bei 3,6 MHz
loop1:      ldi     r17,0xFF      ;Laufvariable loop1
loop2:      ldi     r18,13        ;Laufvariable loop3
loop3:      dec     r18           ;Zähler 3-1, Kalibrierung auf MCU-Takt
            brne    loop3         ;Solange nicht NULL
            dec     r17           ;Zähler 2-1
            brne    loop2         ;Solange nicht NULL
            dec     r16           ;Zähler 1-1
            brne    loop1         ;Solange nicht NULL
            pop     r18           ;r18 wiederherstellen
            pop     r17           ;r17 wiederherstellen
            pop     r16           ;r16 wiederherstellen
            ret                   ;Rücksprung
;---------------------------------------------------------------------
```

## 8.11 Lösung der Aufgabe 6.1.4

Erstellen Sie für eine Pulsweitenmodulation die Werte für die Register TCCR1A und TCCR1B für eine Motorsteuerung die mit einer Grundfrequenz von f=Mikrocontrollertakt/1024. Die Bitbreite soll auf 10Bit eingestellt werden.

In den Vergleichsregistern soll ein mittlerer Wert eingestellt werden. Als Kontrollbit wird eine Leuchtdiode an PORTB1 verwendet.

In TCCR1A wird der Binärwert 0b10000011 eingestellt.

Die ersten beiden Bits bestimmen die Auflösung. 11 bedeutet 10Bit-Auflösung. Die letzten beiden Bits bestimmen, was aufgrund eines Vergleichsergebnisses getan wird. In diesem Fall beim Hochzählen ein 1-Signal an PORTB1, beim Runterzählen ein 0-Signal an PORTB1.

In TCCR1B wird der Binärwert 0b00000010 geladen. Die ersten beiden Bits bestimmen den Teilerfaktor für den Systemtakt. In diesem Fall 1024.

Wenn in OCR1AL und OCR1AH ein mittlerer Wert eingestellt werden soll, dann muss man bedenken, dass der Maximalwert in 10Bit Auflösung der Hexadezimalwert 0x03ff ist.

Oder binär ausgedrückt: 0b0000 0011 1111 1111

Die Hälfte dieser Zahl ist dann 0000 0001 1111 1111 also hexadezimal 0x01ff

Somit muss in OCR1Al der Hexadezimalwert 0xff und in OCR1AH der Wert 0x01 geladen werden.

## 8.12 Lösung zu Übung 7.1.3

```
;+----------------------------------------------------------------------
;| Title            :Übung 7-1-3
;+----------------------------------------------------------------------
;| Prozessor        : ATmega8
;| Takt             : 3,6864 MHz
;| Sprache          : Assembler
;+----------------------------------------------------------------------
.include    "AVR.H"
;----------------------------------------------------------------------
;Reset and Interrupt vector       ;VNr.  Beschreibung
            rjmp    main         ;1    POWER ON RESET
            reti                 ;2    Int0-Interrupt
            reti                 ;3    Int1-Interrupt
            reti                 ;4    TC2 Compare Match
            reti                 ;5    TC2 Overflow
            reti                 ;6    TC1 Capture
            reti                 ;7    TC1 Compare Match A
            reti                 ;8    TC1 Compare Match B
            reti                 ;9    TC1 Overflow
            reti                 ;10   TC0 Overflow
            reti                 ;11   SPI, STC Serial Transfer Complete
            reti                 ;12   UART Rx Complete
            reti                 ;13   UART Data Register Empty
            reti                 ;14   UART Tx Complete
            reti                 ;15   ADC Conversion Complete
            reti                 ;16   EEPROM Ready
            reti                 ;17   Analog Comparator
            reti                 ;18   TWI (IC) Serial Interface
            reti                 ;19   Store Program Memory Ready
;----------------------------------------------------------------------
;Start, Power ON, Reset
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16
            ldi     r16,hi8(RAMEND)
            out     SPH,r16
            ldi     r16,23
            out     UBRRL,R16    ;9600Baud einstellen
            sbi     UCSRB,3      ;Transmit einschalten
            ldi     R31,0x00     ;höherwertiges Byte in R31
            ldi     R30,0x60     ;niederwertiges Byte in R30
;----------------------------------------------------------------------
mainloop:   wdr
            ldi     R16,0x00     ;Anfangswert in Register R16
Loop1:      st      Z,R16
            inc     R30
            inc     R16
            cpi     R30,0x66
            brne    Loop1
;----------------------------------------------------------------------
Loop2:      ld      R16,Z        ;Inhalt SRAM lesen
            rcall   SENDE
            dec     R30
            cpi     R30,0x5F
            brne    Loop2
ende:       rjmp    ende         ;=Endlosschleife
;----------------------------------------------------------------------
; UP: uartPutChar, Zeichen per UART senden
; PE: r16 = Zeichen
;----------------------------------------------------------------------
SENDE:      sbis    UCSRA,5      ;USR Bit5 =  UDRC
            rjmp    SENDE        ;warten, bis UDR bereit
            out     UDR,r16
            ret                  ;zurück zum Hauptprogramm
```