

# Grundlagen Rechnertechnik

## Versuch 3: Interrupts

**Anleitung zum Praktikumsversuch**  
**Autoren: Marco Schmidt und Jan Weber**

# 1 Einführung

*Die Einführung bitte aufmerksam lesen, da in diesem Versuch mit Hardware gearbeitet wird!*

Dieses Dokument beschreibt das erste Laborpraktikum zur Vorlesung "Grundlagen Rechnertechnik". In diesem Versuch geht es um die Verarbeitung von Interrupts auf einem AVR-Mikrocontroller in Assembler. Die Themen wurden inhaltlich in der Vorlesung besprochen, nun sollen die theoretischen Grundlagen in die Praxis überführt werden. Da Sie in diesem Praktikum mit Hardware arbeiten, lesen Sie die folgenden Hinweise bitte aufmerksam durch. Wenn Sie Fragen bezüglich der Hardware haben oder sich nicht sicher sind wie Sie die Hardware anschließen müssen, fragen sie *zuerst* den Praktikumsleiter und versuchen *danach* Ihre Lösung umzusetzen.

## 1.1 Hinweise zur Hardware

Sie werden in diesem Praktikum mit elektronischen Bauteilen arbeiten, diese sind sehr empfindlich. Es gilt folgendes zu beachten:

- Bevor Sie das Praktikum beginnen, müssen Sie an der Sicherheitseinweisung in das Labor teilgenommen haben.
- Alle Bauteile sind sorgfältig zu behandeln, elektronische Bauteile sollten niemals mit Gewalt in ein Steckbrett gedrückt werden.
- Bei Anschluss der Bauteile an eine Spannungsquelle achten Sie immer darauf, dass Sie + und – Pol nicht vertauschen.
- Einige Bauteile dürfen nicht direkt an die Spannungsquelle angeschlossen werden, sondern müssen mit einem Vorwiderstand versehen werden. Dies ist in der Versuchsbeschreibung explizit angegeben. Diese Bauteile werden zerstört, wenn Sie keinen Vorwiderstand einsetzen. Eine mutwillige Zerstörung führt zum Ausschluss vom Praktikum.
- Die Bauteile reagieren empfindlich auf statische Aufladung. Stellen Sie sicher dass Sie nicht statisch aufgeladen sind, wenn Sie die Bauteile in die Hand nehmen.
- Der verwendete Mikrocontroller auf dem Arduino-Board kann nur sehr kleine Ströme schalten. Schon kürzeste Kurzschlüsse führen zur Zerstörung dieses Boards. Überprüfen Sie die Verschaltung also mehrfach vor der Inbetriebnahme! Ändern Sie die Schaltung niemals im eingeschalteten Zustand.

## 2 Versuch: Toggeln einer LED

### 2.1 Beschreibung des Versuchs

In dieser Aufgabe soll eine LED durch das Betätigen eines Tasters ein- und ausgeschaltet werden. Jedes mal, wenn der Taster gedrückt wird, soll der Leuchtzustand der LED geändert werden, sodass der Benutzer durch den ersten Tastendruck die LED einschaltet und durch einen weiteren ausschaltet.

Für diese Funktion sind externe Interrupts gut geeignet.

### 2.2 Beschreibung der Hardware

In diesem Versuch kommt das Arduino-Nano-Board, ein Taster und ein  $10\text{k}\Omega$  zum Einsatz. Diese Bauteile haben Sie bereits in den vergangenen Praktika kennengelernt.

Um den Taster mit dem Mikrocontroller auslesen zu können ist er wie in Bild 1 zu verschalten. Wenn der Taster nicht gedrückt ist, sind seine beiden Anschlüsse nicht miteinander verbunden. In diesem Fall liegt am Ausgang über den  $10\text{k}\Omega$  Widerstand ein eindeutiger Zustand von  $0\text{V}$  an. Falls der Taster gedrückt wird ist der Ausgang direkt mit  $5\text{V}$  verbunden.

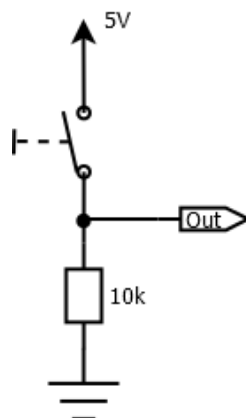


Abbildung 1: Taster mit Pulldown-Widerstand

### 2.3 Bearbeiten von Registern

Um den Mikrocontroller zu steuern müssen einzelne Bits in verschiedenen Registern des Mikrocontrollers manipuliert werden. Hierbei kann dem ganzen Register ein neuer Wert zugewiesen werden, sodass alle Bits im Register überschrieben werden. Alternativ können auch nur einzelne Bits verändert werden, sodass die restlichen Bits bestehen bleiben. Listing 1 zeigt die übliche Vorgehensweise in Assembler.

Listing 1: Register manipulieren

```
; das 5. Bit von PORTD wird gesetzt, der Rest nicht veraendert
```

```

in      r16 ,PORTD
sbr     r16 ,(1<<5)
out     PORTD,r16

;das Bit COM0B0 im Register TCCR0A wird gesetzt, der Rest nicht veraendert
in      r16 ,TCCR0A
sbr     r16 ,(1<<COM0B0)
out     TCCR0A,r16

;0x0d(Hex) = 0b00001101(Binaer) = Bits 0,2,3 in PORTD werden gesetzt
ldi r16 ,0b00001101;
out PORTD,r16

```

## 2.4 IO-Register

Jeder IO-Pin des Mikrocontrollers hat eine eindeutige Bezeichnung der Form P[Port][Pin]. PC0 bedeutet also, dass es sich um den Pin 0 des Ports C handelt. Jeder Port besteht in der Regel aus den acht Pins 0..7. Um die IO-Pins zu benutzen sind die im Folgenden beschriebenen Register wichtig.

**DDRx** Das Data Direction Register bestimmt, welche Pins des Mikrocontrollers Ein- oder Ausgänge sind. Das x gibt den Port an. Wenn das DDRC Register auf den Wert 0b00000011 gesetzt wird, bedeutet das, dass die Pins 0 und 1 des Ports C als Ausgang definiert sind und die restlichen Pins dieses Ports als Eingang.

**PORTx** Mit dem PORT-Register können die als Ausgang definierten Pins des Ports auf HIGH oder LOW gesetzt werden, also ein- und ausgeschaltet werden.

**PINx** Das PIN-Register enthält die aktuell an den Pins anliegenden Zustände. Hiermit kann also überprüft werden, ob ein Eingang gerade HIGH oder LOW ist.

Listing 2: Umgang mit IOs

```

;PB0 wird als Ausgang definiert
in      r16 ,DDRB
sbr     r16 ,(1<<PB0)
out     DDRB,r16

;PB0 wird auf High gesetzt
in      r16 ,PORTB
sbr     r16 ,(1<<PB0)
out     PORTB,r16

;Lese Pegel an PC3 ein.
;r16 enthaelt dann eine 0 bei Low und einen Wert >0 bei High
in      r16 ,PINC
sbr r17 ,(1<<PC3)
and     r16 ,r17

```

## 2.5 Konfiguration von externen Interrupts

Um auf dem verwendeten Mikrocontroller Interrupts verarbeiten zu können, müssen diese zunächst konfiguriert werden. Dies geschieht durch das setzen einzelner Bits in verschiedenen Registern des Mikrocontrollers, die im Datenblatt näher beschrieben sind. Das Datenblatt für den Mikrocontroller *AtMega328*, der auf dem Arduino-Nano-Board zum Einsatz kommt, finden Sie auf dem Labor-PC `~/rechnertechnik/atmega328.pdf`. Die für diese Aufgabe notwendigen Register sind im Folgenden kurz beschrieben.

**EIMSK** Der *AtMega328* verfügt über die zwei externen Interrupts INT0 (verbunden mit Arduino Pin D2) und INT1 (Arduino Pin D3). An diese Pins können digitale Signale angelegt werden, die dann Interrupts auf dem Controller auslösen. Diese externen Interrupts werden mit dem *External Interrupt Mask Register* **EIMSK** ein- und ausgeschaltet (vgl. Datenblatt Abschnitt 12.2.2 S. 72).

**EICRA** Mit dem *External Interrupt Control Register A* kann konfiguriert werden, ob ein Interrupt bei einer steigenden Flanke, einer fallenden Flanke oder einem logischen Wechsel des anliegenden Signals ausgelöst werden soll (vgl. Datenblatt Abschnitt 12.2.1 S. 71).

Nachdem die Interrupts konfiguriert wurden, müssen Interrupts noch allgemein mit dem Assembler Befehl **sei** erlaubt werden. Der Befehl **cli** verbietet Interrupts wiederum.

**ISR** Sobald nun beispielsweise der Interrupt INT0 ausgelöst wird, springt der Controller zur Adresse INT0addr. An dieser Stelle kann nur ein Befehl stehen. Zur Verarbeitung des Interrupts reicht meistens ein einzelner Befehl nicht aus, deswegen wird hier in der Regel ein Sprungbefehl hinterlegt, der in ein Unterprogramm springt. Dieses Unterprogramm wird auch *Interrupt Service Routine*, kurz *ISR*, genannt.

Mittels *.org* ist es möglich, Befehle an die explizite Speicherstellen zu schreiben. Listing 3 zeigt dieses Vorgehen. Zunächst wird an die Adresse *0x000*, an der der Controller nach dem Einschalten startet, ein Sprungbefehl zu einem Unterprogramm *main* gesetzt. Dies gewährleistet, dass der Controller nach dem Einschalten mit *main* beginnt. Außerdem wird an die Adresse *INT0addr* ein Sprungbefehl zur *ISR* von INT0 gesetzt.

Am Ende der *ISR* muss der Befehl *reti* stehen, damit der Controller da weiterarbeitet, wo er von dem Interrupt unterbrochen wurde.

Listing 3: ISR für den externen Interrupt INT0

```
.org 0x000
    rjmp    main        ; Sprung nach main
.org INT0addr
    rjmp    ISR_int0    ; Sprung zur ISR von INT0

main:                                ; hier beginnt das Hauptprogramm

ISR_int0:                            ; hier beginnt die ISR von INT0
    reti                ; Rueckkehr
```

## 2.6 Kompilieren und Programmieren

Auf dem Labor-PC finden Sie im Arbeitsverzeichnis `~/rechnertechnik/ASM` ein Makefile, mit dem Ihr Code kompiliert und auf das per USB angeschlossene Arduino-Board geladen werden kann. In diesem Verzeichnis finden Sie auch die Datei *main.S*, die Sie als Grundlage für Ihre Programmierarbeit verwenden können. Wenn Sie den Befehl **make** in diesem Verzeichnis im Terminal ausführen, wird der Code kompiliert. Mit dem Befehl **make program** laden Sie das Programm dann auf den Mikrocontroller.

## 2.7 Aufgabenstellung

Die Bauteile für diesen Versuch finden Sie in Ihrem Sortimentkasten. Bearbeiten Sie die Aufgaben bitte mit größter Sorgfalt, da es sonst zu einer Zerstörung der Hardware kommen kann. Im Zweifelsfall fragen sie bitte einen Betreuer.

1. Schließen Sie den Taster gemäß Bild 1 an den Arduino-Pin D2 an. Dieser ist mit dem INT0 am Pin PD2 des Mikrocontrollers verbunden.
2. Als LED wird die onboard LED L benutzt. Diese ist auf dem Arduino-Board bereits über einen Vorwiderstand mit D13=PB5 verbunden.
3. Konfigurieren Sie den Pin PB5, an dem die LED angeschlossen ist als Ausgang und den Pin PD2 des Tasters als Eingang.
4. Konfigurieren Sie den externen Interrupt INT0 so, dass er bei einer steigenden Flanke (rising edge) des Tastersignals ausgelöst wird.
5. Füllen Sie die ISR mit Code, sodass bei jedem Tasterdruck der Leuchtzustand der LED geändert wird. Führen Sie dies dem Praktikumsleiter vor!

## 3 Versuch: Dimmen einer LED

### 3.1 Beschreibung des Versuchs

In dieser Aufgabe soll eine LED gedimmt werden, sodass sie in verschiedenen Helligkeiten leuchten kann. Dies soll jedoch nicht durch eine analoge Ansteuerung, wie beispielsweise eine Veränderung des Stroms, geschehen, sondern mit einem digitalen PWM-Signal.

### 3.2 Beschreibung der Hardware

Die Hardware für diesen Versuch ist bereits aus den vorherigen Versuchen bekannt, es ist lediglich das Arduino-Board mit einer LED und einem Vorwiderstand notwendig.

### 3.3 PWM-Signal

Bei der Pulsweitenmodulation handelt es sich um eine digitale Modulation. Es ist ein Rechtecksignal, dessen Verhältnis der eingeschalteten Zeit zur ausgeschalteten Zeit variiert wird.

In der Praxis ermöglicht das PWM Signal unter anderem LEDs zu dimmen. Blinkt eine LED mit einer Frequenz von beispielsweise 1kHz, so nimmt das träge menschliche Auge das Blinken nicht mehr wahr. Allerdings nimmt das Auge eine Dimmung der LED wahr. Bei einem Tastverhältnis von 50% leuchtet die LED mit etwa halber Intensität.

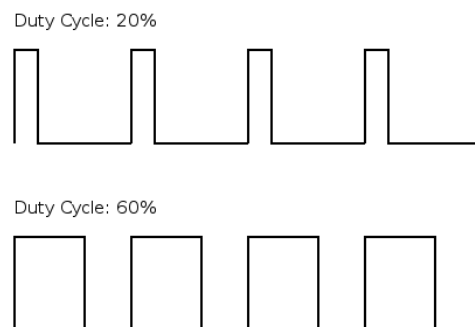


Abbildung 2: PWM-Signal mit 20% und 60% Tastverhältnis

### 3.4 Konfiguration von PWM-Signalen

Für die Generierung von PWM-Signalen verfügt der *AtMega328* über mehrere Timer, die jeweils als eigenes Hardwaremodul unter anderem PWM-Signale erzeugen können. Im Folgenden sind die Register von Timer0 beschrieben, der eine Auflösung von 8 Bit hat.

**TCCR0A** Das Timer/Counter Control Register A ermöglicht es, den Arbeitsmodus von Timer0 festzulegen. Die Bits WGM00 bis WGM02 geben an, in welchem Mode dieser Timer betrieben wird. Unter anderem kann hier gemäß Tabelle 14-8 auf Seite 108 des Datenblatts der *Fast PWM Mode 3* konfiguriert werden.

Außerdem beeinflussen die Bits COM0A1 bis COM0A0 die Funktion des Timers. Diese sind je nach gewähltem Arbeitsmodus nach den Tabellen in Abschnitt 14.9.1 zu konfigurieren.

**TCCR0B** Das Timer/Counter Control Register B beinhaltet unter anderem die Bits CS00 bis CS02. Hiermit kann die Zeitbasis für das PWM Signal eingestellt werden. Der sogenannte Prescaler bestimmt, durch welchen Faktor die Taktfrequenz des Mikrocontrollers geteilt wird, bevor diese Taktquelle als Zeitbasis für den PWM-Generierung genutzt wird. Im Rahmen dieses Praktikums wird bei einem Prescaler von 64 eine PWM-Frequenz von etwa  $\frac{f_{CPU}}{\text{Prescaler} \cdot \text{Timerauflösung}} \approx 980\text{Hz}$  erreicht. (vgl. Tabelle 14-9 S.110)

**OCR0A** Das Output Compare Register A ermöglicht es, das Tastverhältnis des PWM-Signals des PWM-Ausgangs A einzustellen. In dieses Register kann ein Wert im Bereich von 0 bis 255 (256 Schritte= $2^8$ =Timerauflösung) eingetragen werden. 0 bedeutet dabei 0% Tastverhältnis und 255 bedeutet 100% Tastverhältnis.

### 3.5 Aufgabenstellung

1. Schließen Sie eine LED an den Arduino-Pin D6 Boards. Dieser ist mit dem Pin PD6 des Mikrocontrollers verbunden, der gleichzeitig der PWM Ausgang A von Timer0 ist. Vergessen Sie dabei nicht den 390Ω Vorwiderstand!
2. Konfigurieren Sie den Pin PD6, an dem die LED angeschlossen ist als Ausgang.
3. Konfigurieren Sie den Timer0 folgendermaßen:
  - Mode 3 (vgl. Tabelle 14-8 S.108)
  - non-inverting mode (vgl. Tabelle 14-3 S.106)
  - Prescaler=64 (vgl. Tabelle 14-9 S.110)
4. Stellen Sie verschiedene Tastverhältnisse ein und prüfen Sie, ob die LED mit der erwarteten Intensität leuchtet. Führen Sie dies dem Praktikumsleiter vor!



## 4 Versuch: Auswerten eines Inkrementalgebers

### 4.1 Beschreibung des Versuchs

In dieser Aufgabe soll ein Inkrementalgeber ausgewertet werden und mit ihm eine LED gedimmt werden. Der Code für das Dimmen der LED kann aus der vorherigen Aufgabe übernommen werden.

### 4.2 Beschreibung der Hardware

In dieser Aufgabe kommt unter anderem ein inkrementeller Drehgeber zum Einsatz. Dieser Drehgeber ist ein Sensor, der eine Drehbewegung in ein digitales Signal umwandelt. Der Drehgeber gibt ein sogenanntes Quadratursignal aus, das aus den beiden Rechtecksignalen A und B besteht. Ein Mikrocontroller kann mittels Interruptverarbeitung diese beiden Signale auswerten und feststellen, in welche Richtung und wie weit der Drehgeber gedreht wird.

Der in diesem Praktikum verwendete Drehgeber vom Typ *Panasonic EVEQDBRL416B* hat eine Auflösung von 16 Schritten pro Umdrehung. Bei jedem Schritt verändern sich die Zustände von den Ausgangssignalen A und B wie in Bild 3 zu sehen. Dabei sind die beiden Signale zueinander Phasenverschoben. Dies ermöglicht es, die Drehrichtung des gerade geschehenen Schritts zu bestimmen.

Die Pinbelegung des Drehgebers zeigt Bild 4.

### 4.3 Drehrichtungserkennung in Quadratursignalen

Bild 3 zeigt den Zeitverlauf der Signale A und B des Drehgebers, wenn er vorwärts gedreht wird. Zum Zeitpunkt der steigenden Flanke von A ist B immer low. Um eine Rückwärtsdrehung nachzuvollziehen können Sie den Zeitverlauf von rechts nach links (also in zeitlich negative Richtung) durchlaufen. In diesem Fall ist bei einer steigenden Flanke von A das Signal B auf high.

Sie können die Drehrichtung im Quadratursignal also erkennen, indem Sie untersuchen, ob während einer steigenden Flanke von A Signal B low oder high ist.

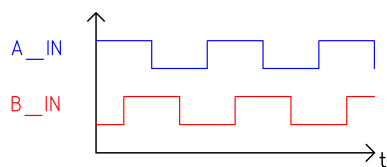


Abbildung 3: Quadratursignal

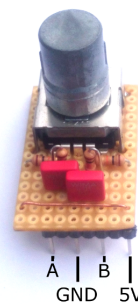


Abbildung 4: Anschlussbelegung  
Drehgeber

## 4.4 Aufgabenstellung

1. Schließen Sie den Drehgeber folgendermaßen an:
  - 5V an 5V
  - GND an GND
  - A an Arduino-Pin D2=PD2=INT0
  - B an Arduino-Pin D3=PD3=INT1
2. Schließen Sie eine LED an den Arduino-Pin D6 Boards. Dieser ist mit dem Pin PD6 des Mikrocontrollers verbunden, auf dem das PWM-Signal zum dimmen ausgegeben wird.
3. Konfigurieren Sie den Pin PD6, an dem die LED angeschlossen ist als Ausgang.
4. Konfigurieren Sie den externen Interrupt INT0 so, dass er bei einer steigenden Flanke (rising edge) des Tastersignals ausgelöst wird.
5. Entwickeln Sie nun eine Logik in der ISR, die überprüft, in welche Richtung der Drehgeber gedreht wird und entsprechend eine Variable hoch oder runter zählt. Jedes mal, wenn eine steigende Flanke von Signal A erfolgt wurde der Drehgeber einen Schritt gedreht. Die Richtung des Schritts ist wie in Abschnitt 4.3 beschrieben zu bestimmen. Je nachdem, ob ein Schritt vorwärts oder rückwärts erkannt wurde, kann nun eine Zählvariable hoch bzw. runter gezählt werden. Zählen Sie bei jedem Schritt die Zählvariable um 16 hoch bzw. runter.
6. Erweitern Sie nun ihr Programm so, dass die LED durch den Drehgeber gedimmt wird und ein Benutzer die Helligkeit steuern kann. Dies erreichen Sie, indem die Zählvariable des Drehgebers als PWM-Tastverhältnis in das OCR0A Register schreiben. Führen Sie dem Praktikumsleiter die Dimmfunktion vor!