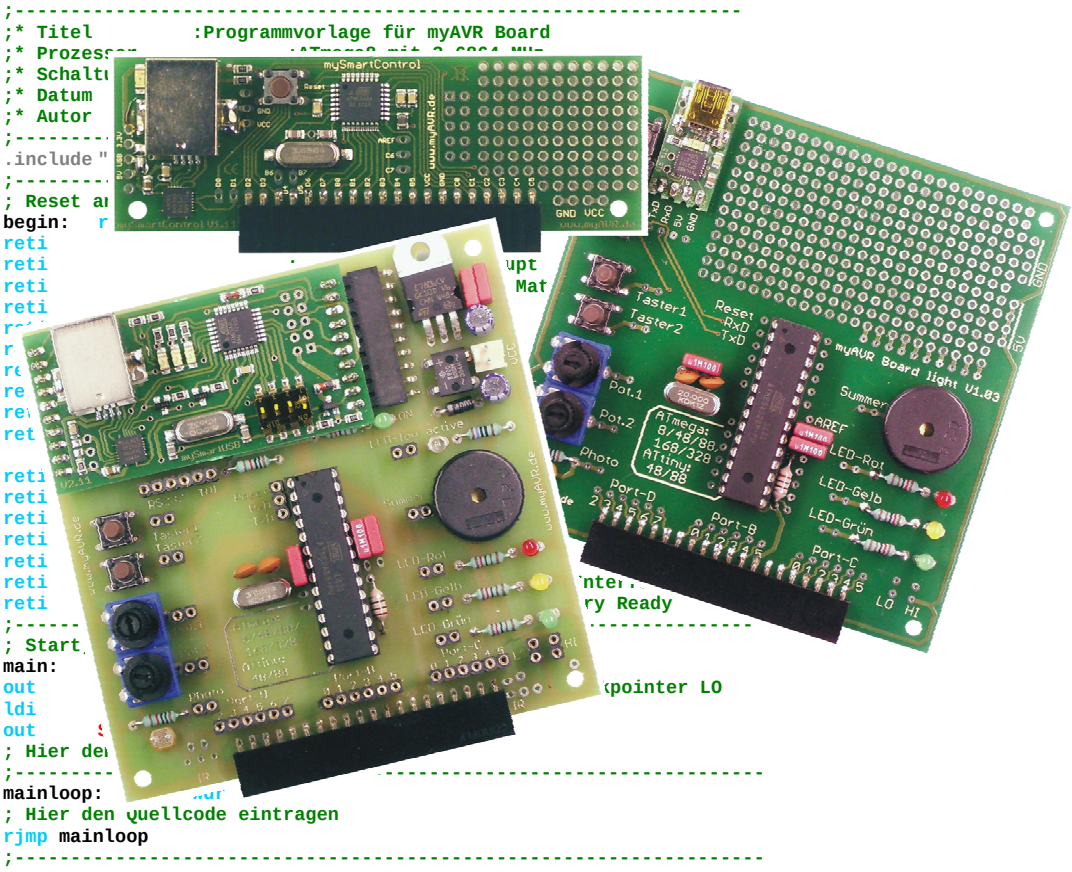
Dipl. Ing. Toralf Riedel
Dipl. Ing. Päd. Alexander Huwaldt

myAVR Lehrbuch Mikrocontroller-Programmierung

Hardwarenahe Programmierung
von AVR-Mikrocontrollern in
Assembler, C und BASCOM



14. Auflage: Oktober 2018

www.sisy-solutions.de
www.SiSy.de
www.myMCU.de
www.myAVR.de
Tel: ++49 (0) 3585 470 222
Fax: ++49 (0) 3585 470 233
Email: service@myMCU.de

Inhalt

Vorwort

Das vorliegende Lehrmaterial ist so konzipiert, dass Sie Schritt für Schritt in die hardwarenahe Programmierung eingeführt werden. Die aufgeführten Grundprinzipien, Problemstellungen, Fallstudien und Lösungsansätze gelten prinzipiell für eingebettete Prozessoren in kleinen und kleinsten Geräten und wenigen Ein- und Ausgabegeräten über Personalcomputern bis hin zu Großrechnern. Die Aufgaben, die sie zu lösen haben, werden in „größeren“ Systemen durch das entsprechende Betriebsystem oder aufwändige Hardware erledigt. Bei Mikrocontrollerlösungen in Embedded-Systems, wie in Waschmaschinen, Autos, Telefonen, Steuerung von Anlagen und Robotern usw. finden Sie oftmals kein standardisiertes Betriebssystem. Dort müssen diese Aufgaben vom Entwickler selbst gelöst werden. Dafür sind diese eingebetteten Systeme hochspezialisiert, sowie bauteil- und energieeffizient bei geringstem Platzbedarf. Bedenkt man die weite Verbreitung von Mikrocontrollern, erkennt man schnell, dass hardwarenahe Programmierung keine Aufgabe von wenigen Betriebssystementwicklern, sondern ein weitverbreitetes Aufgabenspektrum in unterschiedlichsten Branchen und Berufsbildern, ist. Automatisierungstechnik, Mechatronik, Kommunikationstechnik, Medizintechnik, Automobilindustrie, Elektrotechnik, Maschinenbau usw.; kaum eine Branche kann heute noch auf die Anwendung von Mikrocontrollern verzichten. Der PC auf unserem Arbeitsplatz ist nur die berühmte Spitze des Eisberges bei der Anwendung von Prozessoren und der dazu notwendigen Software.
In diesem Lehrbuch wird die hardwarenahe Programmierung am Beispiel einer Mikrocontrollerlösung (Referenzhardware) erarbeitet. Das hat nicht nur den Grund, dass Mikrocontrollern eine zunehmende Bedeutung zukommt, sondern auch dass eine solche Lösung im Vergleich zu einem PC eine überschaubare und im Detail leicht verständliche Hardware besitzt.

Weitere Informationen und Beispiele finden Sie unter
www.myMCU.de

Mit Ihren Fragen können Sie sich direkt an unseren Service wenden,
service@myMCU.de

Beim Lernen wünschen wir Ihnen viel Erfolg.

Ihr myMCU-Team

1 Einleitung

Mikrocontroller oder Mikrorechner?

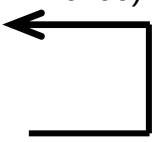
Ein Mikrocontroller ist ein Prozessor, bei dem im Unterschied zu PC-Prozessoren (Mikrorechnern) Speicher, wichtige Baugruppen wie Zeitgeber, digitale sowie analoge Ein- und Ausgabegeräte, auf einem einzigen Chip integriert sind, so dass eine Mikrocontroller-Lösung oft mit einigen wenigen externen Bauteilen auskommt. Ein PC-Prozessor verfügt nicht über eigene Eingabe- und Ausgabekomponenten, sondern über eine Verbindung zu einem externen Systembus, an dem die Ein- und Ausgabegeräte zum Beispiel als Steckkarten angeschlossen sind. In immer mehr Geräten des Alltags werden die Aufgaben von analogen Schaltungen durch Mikrocontroller realisiert. Damit lassen sich vor allem die Produktionskosten der Hardware drastisch senken. Prinzipiell kann die Struktur und Arbeitsweise von Mikrorechnern und Mikrocontrollern mit der von-Neumann-Architektur erklärt werden.

von-Neumann-Architektur (ausgewählte Aspekte):

- Ein Rechner besteht aus einer Zentraleinheit mit Rechenwerk und Steuerwerk, Eingabegeräten, Ausgabegeräten und einem Arbeitsspeicher;
- Die intern verwendete Signalmenge ist binär codiert (Digitalrechner);
- Der Rechner verarbeitet Worte fester Länge (Verarbeitungsbreite, Bussystem, Register, Speicher);
- Die Programmbefehle werden sequenziell in der Reihenfolge ihrer Speicherung abgearbeitet.

Befehlszyklus:

1. POWER-ON oder RESET
2. lade Befehlszeiger mit fester Startadresse (z. B.: 0x00)
3. lese Befehl (von Adresse im Befehlszeiger)
4. decodiere Befehl (Steuerwerk)
5. führe Befehl aus (Rechenwerk)
6. erhöhe den Befehlszeiger um 1, weiter mit 3.



- Der Rechner arbeitet taktgesteuert;
- Die sequenzielle Verarbeitung von Befehlen kann durch Sprunganweisung geändert werden;
- Befehlszeiger = Sprungadresse, Sprung mit oder ohne Bedingung, Unterprogrammaufruf mit Speicherung der Rücksprungadresse.

Prozessoren werden als erstes an der Verarbeitungsbreite der internen Architektur unterschieden: 8 Bit, 16 Bit, 32 Bit, 64 Bit usw. Diese Bit-Zahl kann man als die Länge der Datenworte interpretieren, die der Controller/Prozessor in einem Befehl verarbeiten kann. Die größte in 8 Bit (=1 Byte) darstellbare Zahl ist die 255. Somit kann ein 8-Bit-Mikrocontroller z. B. in einem einfachen Additionsbefehl nur Zahlen von 0 bis 255 verarbeiten und das Ergebnis kann ebenfalls nicht größer als 255 sein, sonst liegt ein Ergebnis-Überlauf vor. Zur Bearbeitung von größeren Zahlen werden dann mehrere Befehle hintereinander benötigt, was bei diesem Prozessor natürlich länger dauert.

Ein Prozessor benötigt eine meist extern eingespeiste Taktfrequenz. Dieses Taktsignal gibt dem Prozessor vor, wann der nächste Befehlsschritt ausgeführt werden soll. Da die Baugruppen innerhalb des Prozessors weder unendlich schnell noch gleich schnell arbeiten, ist dieses Synchronisationssignal für die Zusammenarbeit von Steuerwerk, Rechenwerk, Ein- und Ausgabegeräten notwendig. Die maximale Frequenz, mit der ein Prozessor betrieben werden kann, reicht heute von 1 MHz bis hin zu über 3 GHz.

Bei Mikrocontrollern sind im Moment Taktfrequenzen von 1 bis 300 MHz üblich. Diese Taktfrequenz sagt jedoch noch nichts über die tatsächliche Geschwindigkeit eines Prozessors aus. Es ist auch entscheidend, ob für den oben beschriebenen Befehlszyklus viele, wenig oder gar nur ein Taktzyklus benötigt wird.
Bei vielen Prozessoren wird die Frequenz intern geteilt, ein mit 24 MHz getakteter Mikrocontroller der Intel Reihe 8051 arbeitet eigentlich nur mit 2 MHz, da der externe Takt durch 12 geteilt wird. Benötigt dieser dann für einen Befehl durchschnittlich 2 Taktzyklen, so bleiben "nur" noch 1 Mio. Befehle pro Sekunde übrig (1 MIPS, Million Instructions per Second). Dieser Sachverhalt trifft besonders auf Prozessoren zu, die eine CISC-Architektur besitzen (Complex Instruction Set Computer). Betrachten wir dazu im Gegensatz eine RISC-Architektur (Reduced Instruction Set Computer), finden wir hier kaum interne Taktteilungen und für die Ausführung eines Befehls wird oft nur ein Taktzyklus benötigt. Ein moderner RISC-Mikrocontroller, der ungeteilt mit 8 MHz arbeitet und für viele Befehle nur einen Prozessorzyklus braucht, schafft gegenüber dem mit 24 MHz getakteten CISC Controller fast 8 Mio. Befehle pro Sekunde (8 MIPS). Ein RISC-Prozessor hat dafür einen geringeren Befehlssatz. Komplexe Befehle wie eine Division, müssen deshalb algorithmisch umgesetzt werden.

Wozu ist ein Mikrocontroller gut?

Welche Aufgaben Mikrocontroller haben und warum sie eine zunehmende Bedeutung erlangen, soll folgender Einsatzfall verdeutlichen. Uns allen sind Detektoren unterschiedlichster Art bekannt. Am auffälligsten sind die Portalmetalldetektoren in Flughäfen oder wichtigen öffentlichen Gebäuden, aber auch der stationäre Blitzer um die Ecke verfügt als Sensor über einen Metalldetektor in der Fahrbahn. Ein anderer Anwendungsfall ist zum Beispiel die Lebensmittelindustrie. Es ist sehr wichtig für den Hersteller von Backmischungen, dass keine Nägel, Schrauben, Metallsplitter oder ähnliches während des Produktionsprozesses in das Nahrungsmittel gelangt. Auf der Verpackungsstrecke werden alle Packungen, in denen der Metalldetektor Metallteile ortet, automatisch ausgesondert. Metalldetektoren werden in großen Stückzahlen für unterschiedlichste Anwendungsfälle benötigt.
Das folgende Schema zeigt die Funktionseinheiten eines Metalldetektors nach dem Pulsinduktionsverfahren (PI).

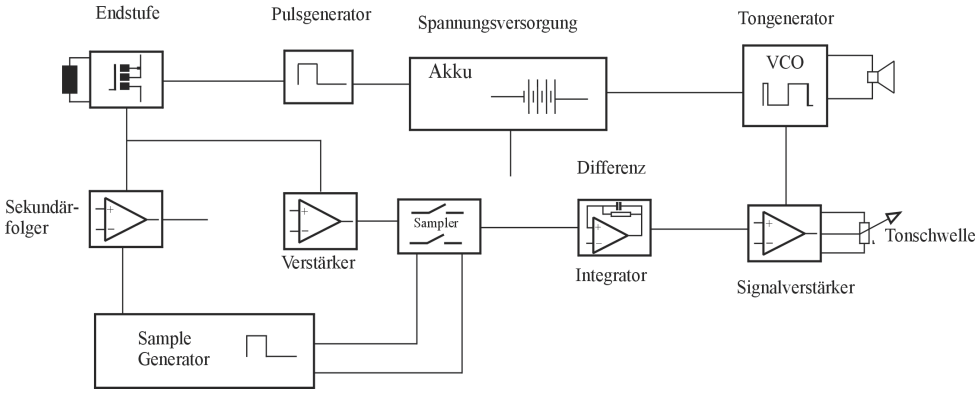


Abbildung: Funktionsschema eines PI-Metalldetektors

Der klassische Aufbau eines solchen Detektors als diskrete analoge Schaltung erfordert trotz Verwendung integrierter Schaltungen einen bestimmten, kaum weiter reduzierbaren Materialaufwand an aktiven und passiven Bauelementen.
Die Materialkosten für diesen konkreten Metalldetektor belaufen sich auf ca. 25 € (Stand: Februar 2004).

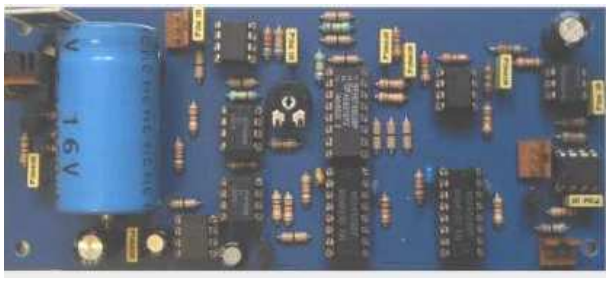


Abbildung: vollständig analoger PI-Metalldetektor

Einen nicht unerheblichen Teil dieser diskreten Schaltung macht die Realisierung des „Timings“, also der Steuerung der verschiedenen Komponenten für den Detektor aus. Der folgende Detektor hat das gleiche Arbeitsprinzip und die gleiche Leistungsfähigkeit wie der oben gezeigte Detektor. Der Materialaufwand ist jedoch um ein vielfaches geringer (Materialkosten ca. 8 €). Dies wurde erreicht, indem die Steuerung des Detektors von einem Mikrocontroller und der entsprechenden Software darauf übernommen wurde.

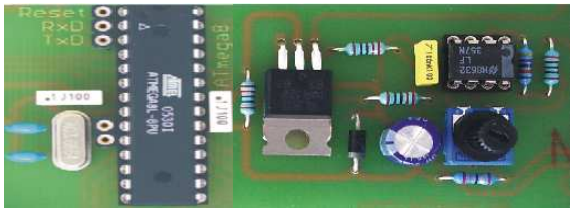


Abbildung: PI-Metalldetektor mit Mikrocontroller

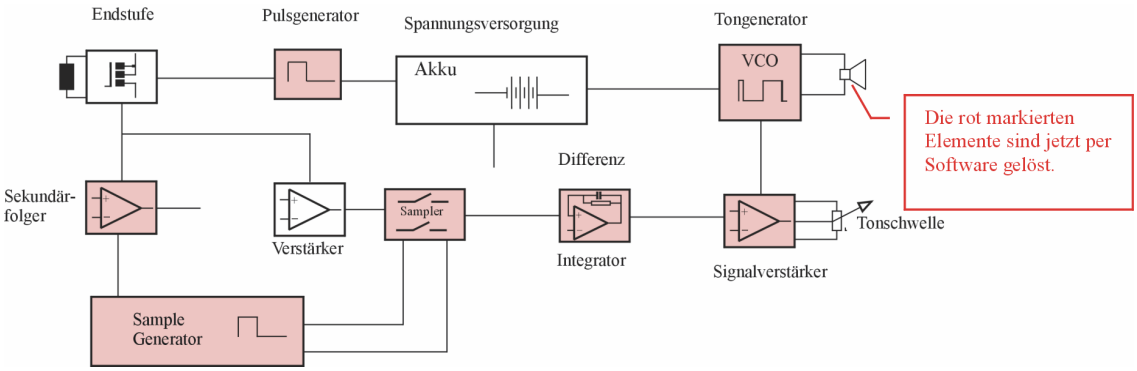


Abbildung: Funktionsverlagerung von der Hardware in die Software

Die zuvor diskret mit analogen Bauelementen realisierten Funktionen des Detektors werden in der mikrocontrollerbasierenden Lösung in die Software verlagert. Damit reduziert sich die Anzahl der aktiven und passiven analogen Bauelemente. Die Komplexität und Größe der Leiterplatte konnte reduziert werden. In der Gesamtheit konnten die Kosten für die Herstellung eines Detektors auf ein Drittel reduziert werden. Die wirtschaftliche Bedeutung der Reduzierung der Stückkosten, in diesem Fall um 60%, braucht man nicht weiter zu betonen. Aber selbst wenn die Kostenreduzierung durch den Mikrocontrollereinsatz pro System nur einige

Cent beträgt, liegt der Kostenreduzierungseffekt in der Massenfertigung schnell bei Millionen Euro.
Ganz klar, die Softwarelösung für den Tongenerator lässt sich auf jedes einzelne System kopieren und verursacht keine Beschaffungskosten mehr. Die diskrete Lösung benötigt eine integrierte Schaltung (IC), 3 Widerstände und einen Kondensator mehr; und diese müssen für jedes zu produzierende Gerät neu beschafft werden. Und das ist noch nicht alles. Die Mikrocontrollerlösung könnte sogar beliebige Melodien abspielen, die der Hersteller im Internet zum Herunterladen anbieten kann. Solche zusätzlichen Eigenschaften werden von den Anwendern der Lösungen gern angenommen, wie der aktuelle Trend bei Klingeltönen und Handylogos zeigt. Es werden nicht nur Kosten, Platz und Gewicht gespart, sondern zusätzliche Möglichkeiten eröffnet. Daher auch der ungebrochene Trend, selbst einfache Geräte wie eine Kaffeemaschine mit einem Mikrocontroller auszustatten.
Hier ein paar Beispiele, für welche Aufgaben Mikrocontroller verwendet werden können: Roboter, CD-, MP3- und DVD-Player, Temperaturregler, Füllstandsregler, Motorsteuerungen, Signaldecoder für Satellitenempfang, Fernbedienung, Alarmanlagen, Schaltuhren, Ladegeräte, Waschmaschinen, Geschirrspüler, Fernseher, Radio, Wecker/Uhr, Messwerterfassung (z. B. Drehzahlmessung im Auto), intelligente Geräte in der Automatisierungstechnik, intelligente Sensoren, intelligente Aktoren z. B. die Airbags im PKW, Handy, alle Formen von Heimelektronik, Geräte der Medizintechnik, Spielzeug …

Abbildung: Internetfähiger TOASTER ;-)

Anforderungen und Möglichkeiten von Mikrocontrollerlösungen

- programmierbar (Update, Optimierung, Wartung)
- flexible Schnittstellen (vielfältig, integriert, standardisiert)
- Selbstdiagnose, Fehlerkorrektur, Debuginterface
- Echtzeitfähigkeit (schnelle Reaktionszeiten)
- Timer, Interruptfähigkeit
- deterministisch (bestimmbares, berechenbares Verhalten)
- geringe Kosten, geringer Leistungsverbrauch

Die Anwendungsgebiete von Mikrocontrollern sind schier unendlich. In allen Bereichen unseres Lebens lassen sich heute „versteckte“ Mikrocontroller finden.

Welchen Mikrocontroller verwenden?

Mikrocontroller werden von unterschiedlichen Firmen angeboten. Weit verbreitete Mikrocontroller sind die der Intel-Reihe 8051, der Zilog-Reihe Z80, Z86, die PIC-Controller der Firma Microsystems und die AVR-Controller der Firma Atmel. Die AVR-Reihen von Atmel haben eine innovative RISC-Architektur, die schnell und einfach zu erlernen ist. Sie sind inzwischen sehr weit verbreitet. Sie sind elektrisch robust und bis zu 10.000-mal programmierbar. Da die AVR-Prozessoren zu den modernsten Controllern am Markt gehören und enorme Zuwachsraten aufweisen, sollen sich alle Ausführungen und die Experimentierhardware auf diese Controller beziehen. Prinzipiell lassen sich jedoch alle Aussagen auf alle anderen Mikrocontroller übertragen.
Ein pikanter Hintergrund ist, dass der AVR-Kern eine Entwicklung von zwei Studenten der Universität Trontheim in Norwegen ist. Atmel kaufte die Lizenz und

entwickelte dieses innovative Konzept weiter. Hartnäckig hält sich das Gerücht, dass die Abkürzung „AVR“ etwas mit den Vornamen der beiden inzwischen nicht mehr Studenten **A**lf Egil Bogen und **V**egard Wollan zu tun hat, die diesen **R**ISC-Prozessor entwickelt haben. Wer weiß ;-)

Welche Programmiersprache benutzen? Assembler, C oder BASIC?

Assembler ist für den Einstieg in die Programmierung von Mikrocontrollern zwar recht schwierig, aber am besten geeignet. Da Assemblerprogramme auf Maschinencodeebene angesiedelt sind, lernt man den Aufbau und Funktionsweise eines Prozessors richtig kennen und kann ihn dadurch optimal ausnutzen. Zudem stößt man bei jedem Compiler irgendwann einmal auf Problemstellungen, die sich nur durch das Verwenden von Assemblercode lösen lassen. Die zunehmende Leistungsfähigkeit und immer mehr Speicher auf den Mikrocontrollern erlauben inzwischen auch die Anwendung von höheren Programmiersprachen. Die Sprachen C/C++ bieten sich auf Grund ihrer Effizienz und Hardwarenähe für die Programmierung von komplexen Mikrocontrollerlösungen an.

Der AVR Assembler wird Ihnen Schritt für Schritt und verständlich vermittelt. Mit den so erworbenen Kenntnissen ist die Programmierung von AVR-Controllern mit jeder anderen Sprache noch effizienter. Die Auseinandersetzung mit den Maschinenbefehlen und der Programmierung in Assembler führt zum besseren Verständnis des Aufbaus und der Arbeitsweise des Prozessors/Mikrocontrollers.

```
;------------------------------------------------------------
;* Titel            :Programmvorlage für myAVR Board
;* Prozessor        :ATmega8 mit 3,6864 MHz
;* Datum            :15.03.2004
;* Autor            :Dipl. Ing. Päd. Alexander Huwaldt
;------------------------------------------------------------
.include            "avr.h"
;------------------------------------------------------------
; Reset and Interruptvectoren    ; VNr. Beschreibung
begin:      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 (I²C) Serial Interface
            reti                ; 19   Store Program Memory Ready
;------------------------------------------------------------
; Start, Power ON, Reset
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16             ; Init Stackpointer LO
            ldi     r16,hi8(RAMEND)
            out     SPH,r16             ; Init Stackpointer HI
            ; Hier den Init-Code eintragen
;------------------------------------------------------------
mainloop:   wdr
            ; Hier den Quellcode eintragen
            rjmp    mainloop
;------------------------------------------------------------
```

Abbildung: AVR Assemblerprogramm

2 Vorbereitung

In diesem Kapitel werden Sie über die notwendige Schritte zur Beschaffung, Installation, Konfiguration und Aufbau einer funktionsfähigen Entwicklungsumgebung informiert. Sollten Sie die Übungen und Beispiele nicht praktisch abarbeiten wollen, so können Sie dieses Kapitel überspringen.

2.1 Hard- und Softwarevoraussetzungen

Systemvoraussetzungen für das myAVR Board und für die Arbeit mit SiSy AVR bzw. myAVR Workpad:

- PC mit Windows XP / 2003 / Vista / 7 / 8 und USB-Anschluss
- mindestens 64 MB RAM, empfohlen 256 MB RAM
- Prozessor ab 800 MHz mit 300 MB freier Speicherplatz auf der Festplatte
- Microsoft Internet-Explorer Version 7 oder höher
- Maus oder ähnliches Zeigegerät
- Assembler Entwicklungsumgebung z. B. SiSy AVR oder myAVR Workpad
- myAVR Board MK2, mySmartControl Mk2 oder myAVR Board light
- USB-Kabel für myAVR Board MK2 und mySmartControl MK2;
  Mini USB-Kabel für myAVR Board light
- Bei Bedarf (z. B. autonomer Einsatz des myAVR-Boards oder Erweiterung mit Add-Ons) geeignete Spannungsversorgung z.B. 9 V Batterie oder stabilisiertes 9 V Netzteil

Hinweis: Alle Beispiele in diesem Lehrbuch werden mit der Referenzhardware myAVR Board MK2 beschrieben

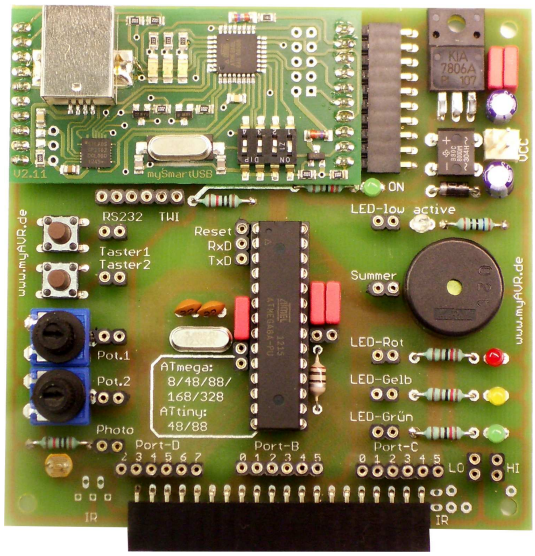
2.2 Beschaffen bzw. Herstellen der Hardware

Alle Ausführungen, Übungen und Aufgabenstellungen beziehen sich auf das myAVR Board MK2 als Referenzhardware, dieses ist im myAVR Einsteigerset enthalten. Grundsätzlich gibt es zwei Anschlussvarianten für das myAVR-System, über USB oder Mini USB.
Für das erfolgreiche Studium dieses Lehrbuches ist das physische Vorhandensein der beschriebenen Referenzhardware nicht notwendig aber empfohlen. Wenn Sie die Studieninhalte weiter vertiefen wollen und Spaß an Elektronik haben, können Sie die Hardware auch selbst fertigen. Die Komponenten erhalten Sie fertig bestückt oder als Bausatz inklusive Bauanleitung etc. unter

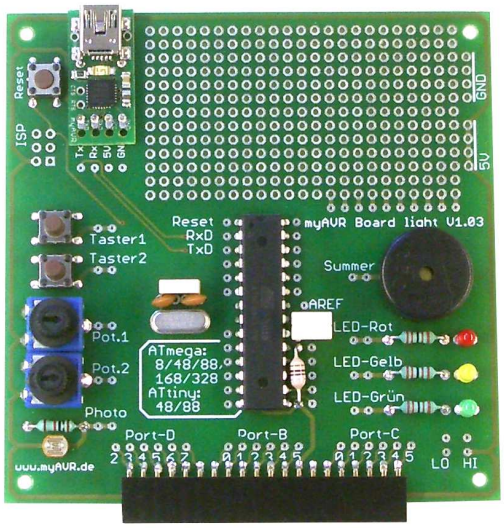
www.myMCU.de

Die bestückten Boards, komplette Einsteigersets oder Bausätze sowie Fertigungsunterlagen und weitere Informationen können angefordert werden unter

service@myMCU.de



Entwicklerboard: komplett bestücktes myAVR Board MK2, Version 2.2



Entwicklerboard: komplett bestücktes myAVR Board light, Version 1.03

kompaktes Prozessorboard: komplett bestücktes mySmartControl MK2, Version 1.11

Weitere Applikationen, Beispielprojekte, Zusatzplatinen:

- Metalldetektor
- Digitaler Speicheroszillograf
- GPS-Tracker
- Mini-Alarmanlage
- …

Besuchen Sie unsere Internetseiten **www.myMCU.de** und **www.sisy.de!**

2.3 Beschaffung und Installation der Software

Im myAVR Einsteigerset steht Ihnen für die Bearbeitung der Übungen und Aufgaben die AVR-Entwicklungsumgebung SiSy AVR bzw. myAVR Workpad zur Verfügung. Sollten Sie SiSy AVR bzw. myAVR Workpad bereits installiert haben, können Sie dieses Kapitel überspringen. Falls Sie noch nicht im Besitz der Software sind, können Sie diese in unserem Shop erwerben.

www.myMCU.de -> Shop

Für die Installation der Software benötigen Sie einen Freischaltcode (Lizenzdaten). Sie können diesen online anfordern unter **www.myMCU.de -> Service** oder oder direkt beim Hersteller:

Tel: 03585-470222
Fax: 03585-470233
Email: service@myMCU.de

Desweiteren sollten Sie Grundkenntnisse in einer beliebigen Programmiersprache besitzen.

2.3.1 Installation von SiSy AVR

Eine ausführliche Installationsanleitung finden Sie im Benutzerhandbuch SiSy. Für die Installation und den ersten Start von SiSy AVR benötigen Sie Administratorenrechte. Die Installation ist erst nach dem ersten Start von SiSy AVR abgeschlossen.

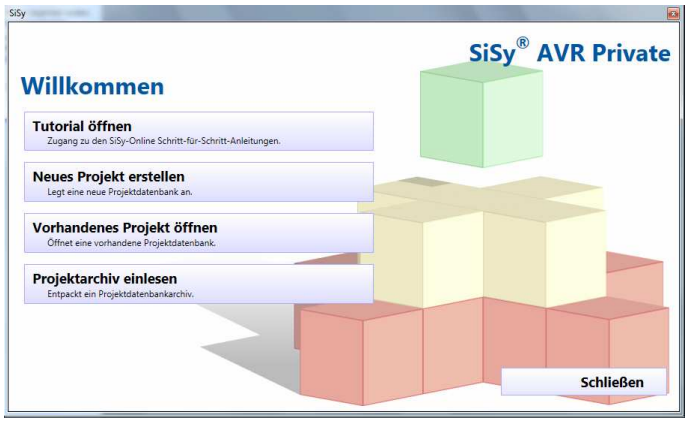


Abbildung: Willkommen in SiSy

Nach dem Start von SiSy AVR erscheint auf Ihren Bildschirm der Dialog „Willkommen in SiSy“. Wählen Sie für die weitere Arbeit eine der Schaltflächen aus.

2.3.2 Installation myAVR Workpad

Legen Sie die CD „myAVR Workpad“ in das CD-ROM Laufwerk ein. Betätigen Sie die Schaltfläche „Workpad installieren“ und folgen Sie dem Installationsassistent.

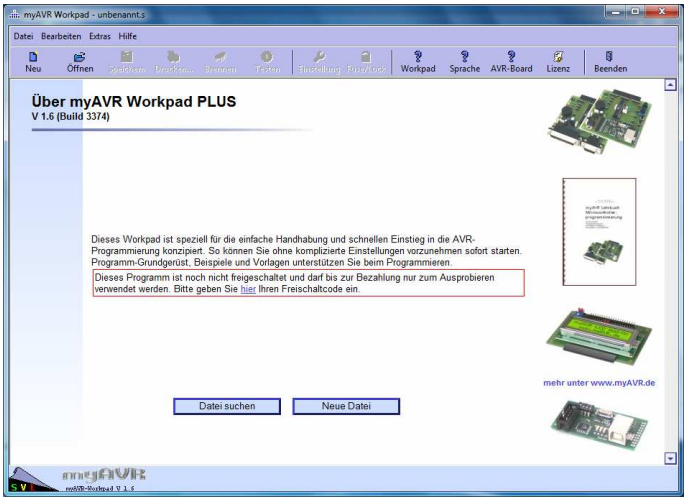


Abbildung: Willkommen im myAVR Workpad

Nach dem ersten Start haben Sie die Möglichkeit, Ihren Freischaltcode einzugeben. Solange dies nicht erfolgt ist, können Sie nur zeitlich begrenzt mit diesem Programm arbeiten

3 Grundlagen zu Mikrocontrollern

In diesem Kapitel soll nochmals kurz auf ausgewählte Aspekte der Hardware und Software eingegangen werden. Das myAVR Board MK2 wird als Referenzhardware vorgestellt.

3.1 Hardwaregrundlagen

Mikrocontroller gehören zu der Klasse der frei programmierbaren Universalprozessoren. Programmierbare Prozessoren gibt es in unterschiedlichen Ausprägungen und Einsatzfeldern. Im Folgenden eine Auswahl von verschiedenen Prozessorklassen:

- **Mikroprozessor:** Ein „herkömmlicher“ Prozessor, wie er auch in PCs zu finden ist. Die Verbindung mit der Außenwelt erfolgt ausschließlich über weitere Bausteine in einem Bussystem.
  Fokus: allgemeine Aufgaben, Leistung, Flexibilität, Standardhardware, Standardsoftware.

- **Mikrocontroller:** Ein Mikrocontroller beinhaltet in einem Chip bereits alle Komponenten, die ihn zu einem funktionsfähigen 1-Chip-µRechner machen. Er besitzt also neben einem Prozessor auch Speicher, diverse Schnittstellencontroller, Timer und einen Interruptcontroller. Er kann über digitale und analoge Ein- und Ausgabeleitungen Mess- und Steueraufgaben ausführen.
  Fokus: Spezialisierung auf konkrete Aufgaben, Platzbedarf, Energieverbrauch.

- **Signalprozessor**: Digitaler Signalprozessor (DSP), Mixed-Signal-Controller: Darunter versteht man Mikrocontroller, die sowohl digitale als auch analoge Signale sehr schnell verarbeiten können.
  Fokus: Spezialisierung auf Signalverarbeitung (Audio, Video, Datenübertragung), Geschwindigkeit, Platzbedarf, Energieverbrauch.

- **Embedded Prozessor:** Embedded System: Mikrocontroller oder DSP werden häufig als eingebettete Systeme verwendet. Das sind Systeme, in denen die Steuereinheit im Zielsystem integriert ist. Ein Beispiel wäre ein Mobiltelefon, hier ist der steuernde Controller im Gerät selbst integriert.

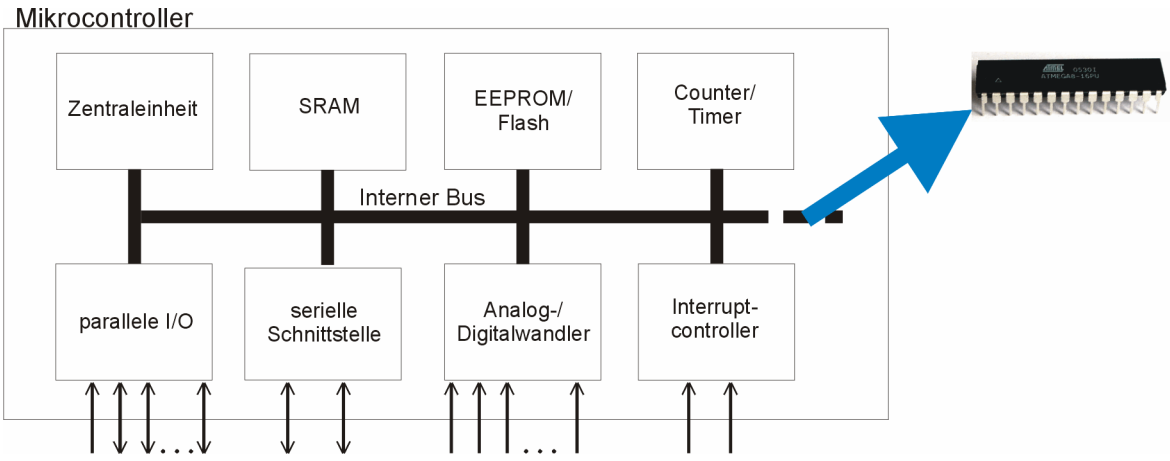


Abbildung: Prinzipaufbau eines Mikrocontrollers

4.2 Mikrocontroller-Entwicklungsumgebung SiSy AVR

Diesen und den folgenden Abschnitt können Sie überspringen, wenn Sie nicht SiSy AVR als Entwicklungsumgebung verwenden.

Schauen wir uns als nächstes kurz in der Entwicklungsumgebung SiSy AVR um. SiSy AVR ist ein allgemeines Entwicklungswerkzeug, mit dem man von der Konzeption eines Systems, bis zur Realisierung die verschiedensten Arbeitsschritte unterstützen kann. Für die Entwicklung von Mikrocontrollerlösungen bietet sich die einfache Programmierung an.

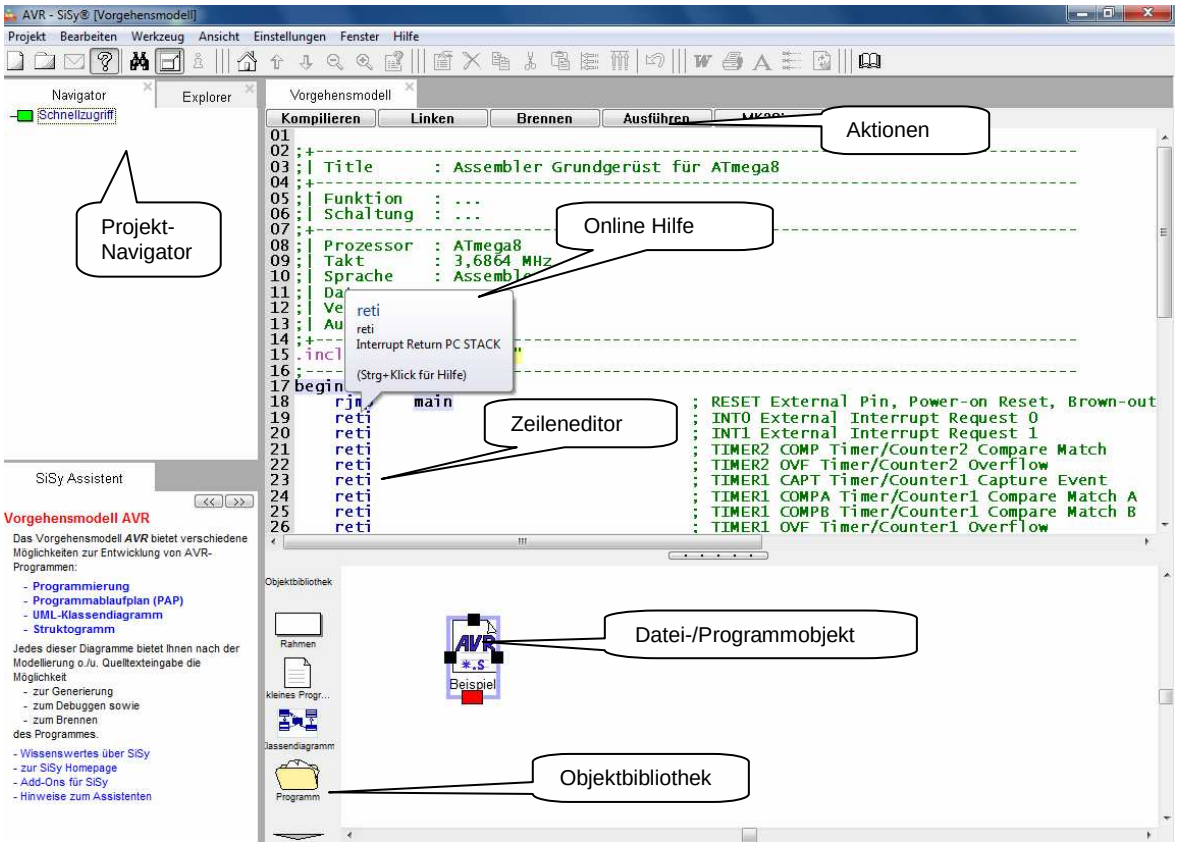


Abbildung: Bildschirmaufbau der Entwicklungsumgebung SiSy AVR

Beim Kompilieren, Linken und Brennen des Schnellstart-Beispiels öffnet sich ein Ausgabefenster und zeigt Protokollausgaben der Aktionen an. Wenn die Hardware ordnungsgemäß angeschlossen, von der Software erkannt und das Programm erfolgreich auf den Programmspeicher des Mikrocontrollers übertragen wurde, muss die letzte Ausschrift in Abhänigkeit der Konfiguration folgenden bzw. ähnlichen Inhalt haben:

```
vorbereiten ...
brennen ...
benutze: mySmartUSB an COM3 mit ATmega8
USB-Treiber installiert, aktiv (V 5.0.0.0), Port: COM3
Prozessor: ATmega8
schreibe 1 Bytes in Flash-Memory ...
... erfolgreich (0.39 s)
OK
```

Abbildung: Ausgabefenster mit „Brenn“ - Protokoll

Die Inbetriebnahme, Test und Datenkommunikation mit der Mikrocontrollerlösung erfolgen über das myAVR-Controlcenter. Über die Schaltfläche „verbinden“ wird das Testboard mit der nötigen Betriebsspannung versorgt und der Controller gestartet. Der Datenaustausch mit dem myAVR Board ist möglich, wenn das USB-Kabel an Rechner und Testboard angeschlossen, sowie die Mikrocontrollerlösung dafür vorgesehen ist. Es können Texte und Bytes (vorzeichenlose ganzzahlige Werte bis 255) an das Board gesendet und Text empfangen werden. Die empfangenen Daten werden im Protokollfenster angezeigt.

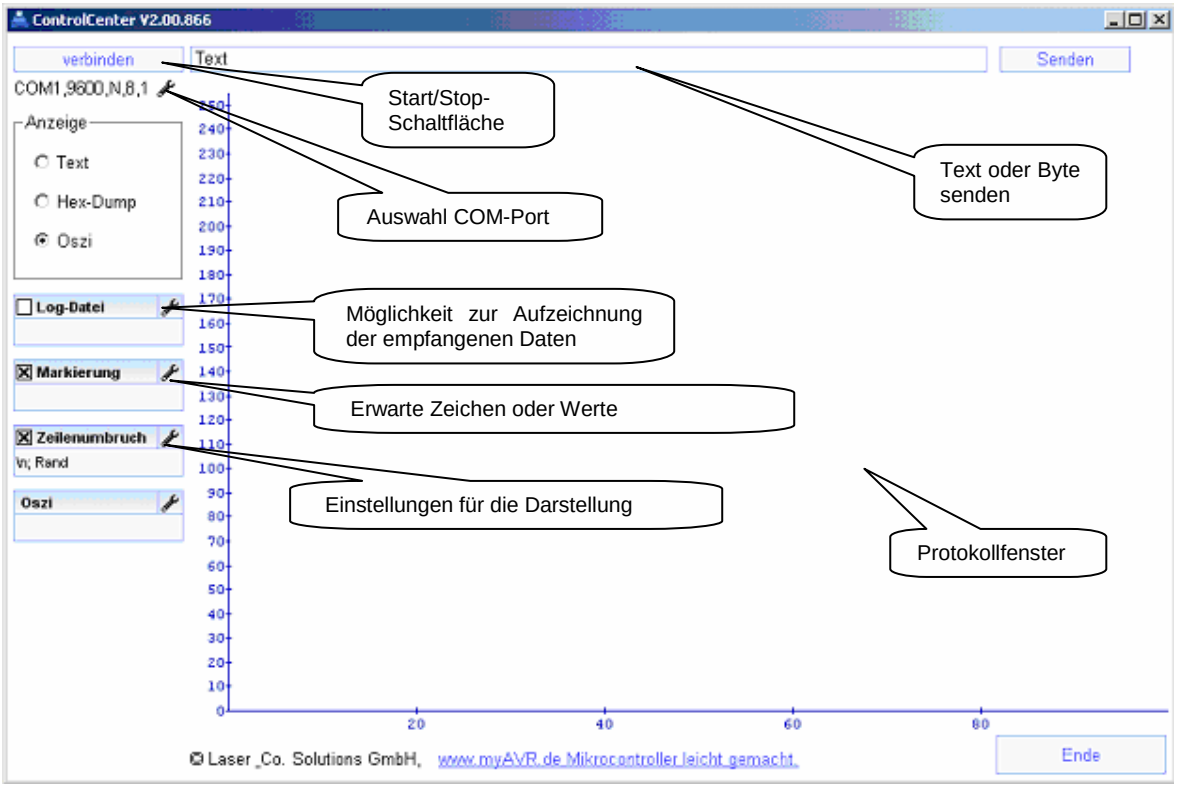


Abbildung: myAVR-Controlcenter

Nutzen Sie die zahlreichen Hilfen und Vorlagen, die SiSy AVR bietet!
Der SiSy LibStore bietet Ihnen Vorlagen und interessante Beispielprogramme.

Eine ausführliche Beschreibung zum SiSy LibStore und den Hilfefunktionen, z.B. Syntax zu Befehlen oder Druckmöglichkeiten, finden Sie im Benutzerhandbuch von SiSy.

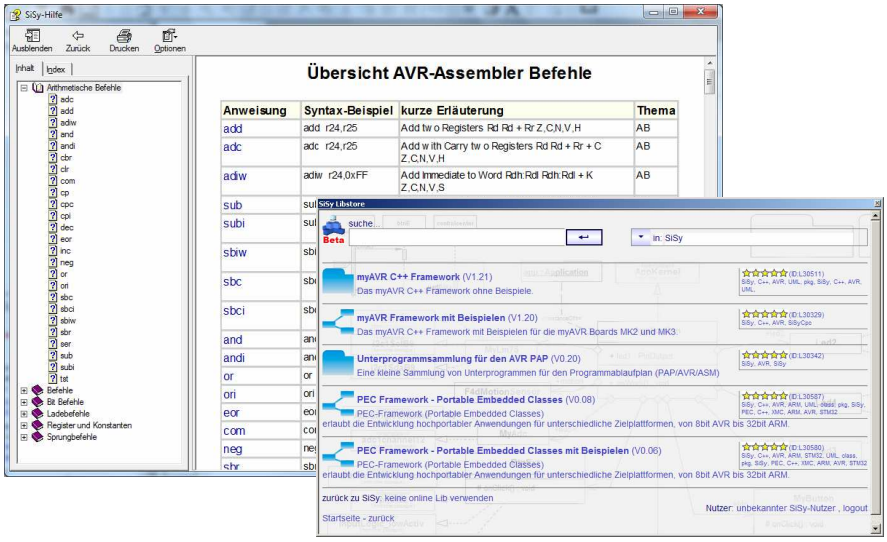
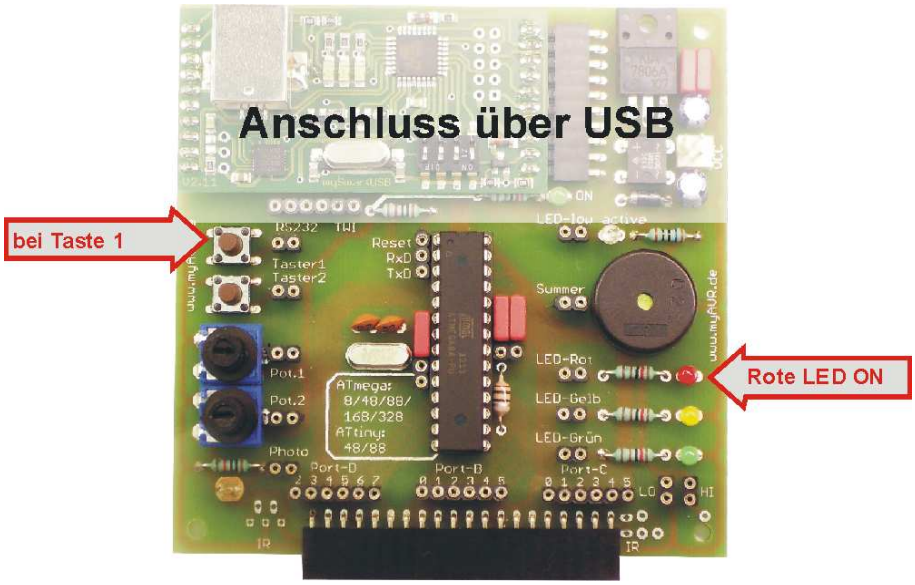


Abbildung: SiSy Hilfe : Dictionary und SiSy LibStore

5.9.2 Beispiel: „Lichtschalter mit Interrupt“

Die folgende Übung soll die Interruptprogrammierung verdeutlichen. Dabei ist eine Mikrocontrollerlösung zu entwickeln, bei der die rote LED per Interrupt eingeschaltet wird und ON bleibt. Der Interrupt soll durch einen Taster ausgelöst werden.



Technische Aspekte für den „Lichtschalter mit Interrupt“

Die Eingabe soll wiederum über einen der Taster (unsaubere Quelle) erfolgen. Damit ist für den gewählten Port der Pull-Up-Widerstand zu aktivieren. Die Ausgabe erfolgt an eine LED. Bei offenem Taster liegt an Port D.2 ein High-Pegel (Pull-Up, logisch 1) an. Wird der Taster geschlossen, zieht er den Pegel auf Low (logisch 0). Wird der Taster losgelassen, öffnet dieser sich und der Pegel geht wieder auf High (Pull-Up, logisch 1).

Lösungsansatz:

Der Port D.2 (INT0, PIN4) wird als Eingang konfiguriert und mit dem Taster verbunden. Der Pull-Up für Port D.2 wird aktiviert. Bei dem Ereignis „Tastendruck“ wird die rote LED eingeschaltet. Für den Interrupt ist die fallende Flanke (Taster zieht auf Low) interessant. Port B.0 wird als Ausgang konfiguriert und mit der roten LED verbunden.

```
ldi r16,0b01000000
out GICR,r16
```
Das Bit 6 im Register GICR erlaubt den externen Interrupt 0. Das Register GICR ist ein Steuerregister und wird wie ein I/O-Register angesprochen (`in/out` Befehl).

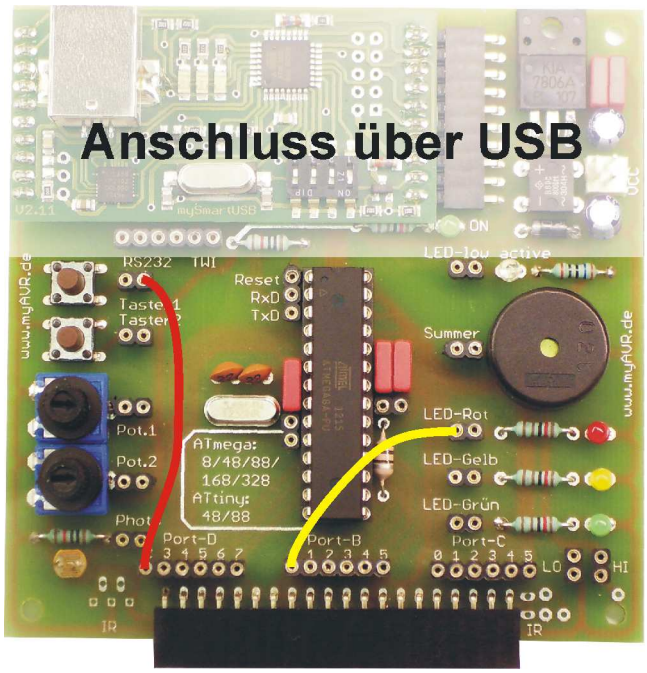
```
ldi r16,0b00000010
out MCUCR,r16
```
Die Bits 0 und 1 im Register MCUCR konfigurieren das Interruptereignis für den externen Interrupt 0.
Das Bit 1 = 1 und Bit 0 = 0 bewirken einen Interrupt bei fallender Flanke an Port D.2 (INT0).

```
rjmp EXT_0
```
Interruptvektor 1 INT0,
Sprung zur Interruptbehandlungsroutine EXT_0

```
reti
```
return from Interrupt

Schaltung:

Für die Eingabe per Taster wird dieser mit Port D.2 verbunden. Die Ausgabe wird über Port B Bit 0 realisiert.



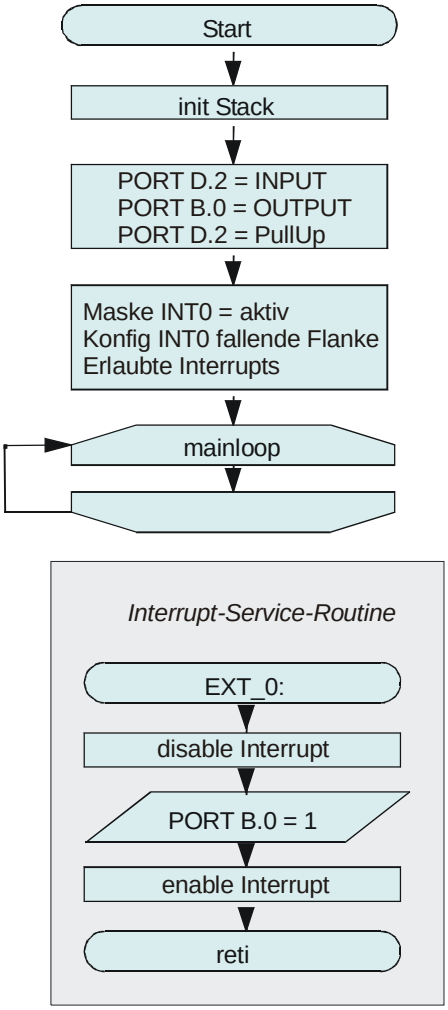


Abbildung: Schaltung und Programmablauf für „Lichtschalter mit Interrupt“

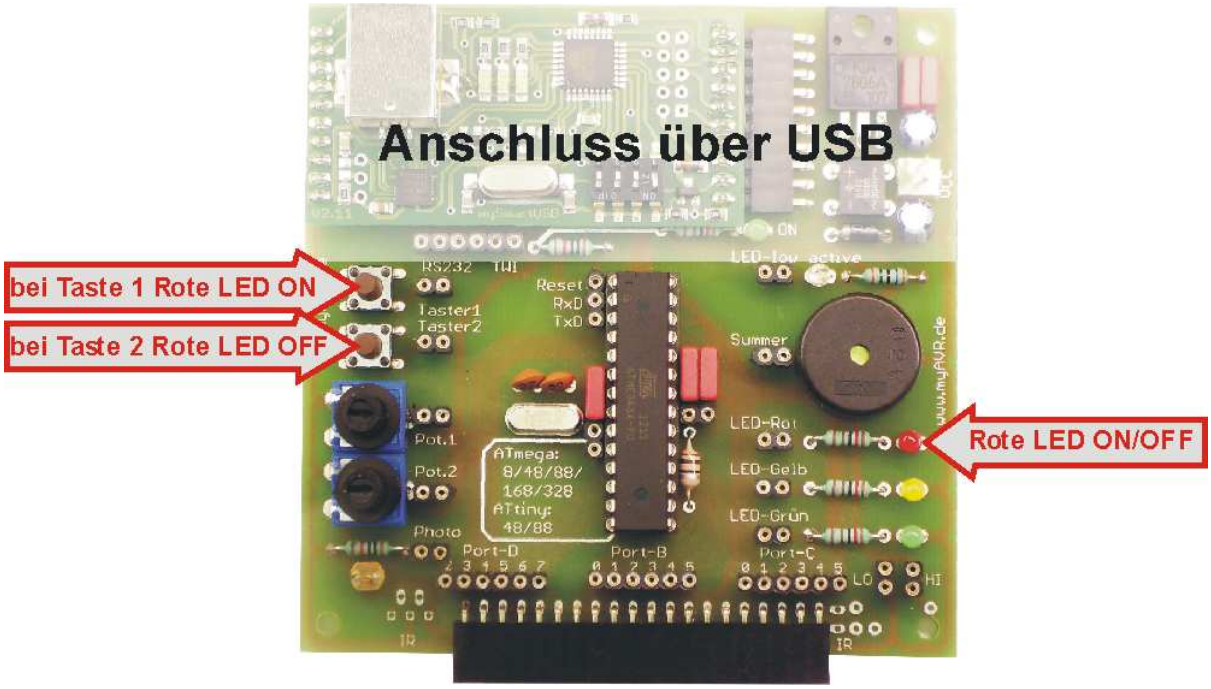
Realisierung:

Geben Sie den Quellcode ein. Benutzen Sie das Grundgerüst für ein AVR-Assemblerprogramm. Übersetzen, brennen und starten Sie das Programm.

```
;+----------------------------------------------------------------------------
;| Titel                  : myAVR Lichtschalter mit INT
;| Funktion               : rote LED per Taster ON, Nutzung ext. INT
;| Schaltung              : Port D.2 Taster, Port B.0 rote LED
;| Prozessor              : ATmega8 3,6864 MHz
;| Sprache                : Assembler
;| Datum                  : 12.03.2004
;| Version                : 1.0
;| Autor                  : Dipl. Ing. Päd. Alexander Huwaldt
;+----------------------------------------------------------------------------
.include "AVR.H"
;----------------------------------------------------------------------------
; Reset and Interruptvectoren    ; VNr. Beschreibung
begin:      rjmp    main        ; 1    POWER ON RESET
            rjmp    EXT_0       ; 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 (I²C) Serial Interface
            reti                ; 19   Store Program Memory Ready
;----------------------------------------------------------------------------
; Start, Power ON, Reset
main:       ldi     r16,lo8(RAMEND)
            out     SPL,r16                 ; Init Stackpointer LO
            ldi     r16,hi8(RAMEND)
            out     SPH,r16                 ; Init Stackpointer HI
            cbi     DDRD,2                  ; Port D.2 Taster = IN
            sbi     PORTD,2                 ; Pull-Up
            sbi     DDRB, 0                 ; Port B.0 = LED OUT
            ldi     r16,0b01000000          ; Maskiere INT0
            out     GICR,r16
            ldi     r16,0b00000010          ; Konfiguriere
            out     MCUCR,r16               ; fallende Flanke
            sei                             ; erlaube Interrupts
;----------------------------------------------------------------------------
mainloop:   rjmp    mainloop
;----------------------------------------------------------------------------
EXT_0:      sbi     PORTB,0                 ; Port B.0 = 1
            reti                            ; Rückkehr zum Hauptprogramm
;----------------------------------------------------------------------------
```

Übung: Wechselschaltung

Lösen Sie die folgende Aufgabe selbständig. Taster 1 soll die rote LED einschalten und Taster 2 soll die LED wieder ausschalten. Nutzen Sie die externen Interrupts INT0 (Port D.2, Pin 4) und INT1 (Port D.3, Pin 5). Konfigurieren Sie die Interrupts für die fallende Flanke des Signals. Legen Sie dazu ein neues „kleines Programm“ in Ihrem Projekt an. Gehen Sie wie oben beschrieben vor und protokollieren Sie die Übung.



Beachte:

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	

MCU Control Register – MCUCR

Bit	7	6	5	4	3	2	1	0	
$35 ($55)	-	-	SE	SM	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	