AVR Mikrocontroller- Elektronik

R. Sitter www.sitter.de

Inhaltsverzeichnis

1. Einführung

Ziel dieser Zusammenstellung ist es, das Wissen um Mikrocontroller zu verbreiten. Die folgenden Zeilen sollen helfen, den Einstieg in Theorie und Praxis zu erleichtern und zum Experimentieren anregen. Mir kommt es dabei nicht auf Professionalität an. Es gibt sicherlich viele Wege, ein Ziel zu erreichen. Oft werde ich nur einen Weg aufzeigen oder vielleicht kenne ich manchmal auch gar keinen. Mehrmals werde ich mehrere Wege aufzeigen, weil es für unterschiedliche Aufgaben besser ist, Alternativen zu kennen. Dadurch ist es möglich, eine Lösung besser zu optimieren. Mir geht es im Wesentlichen um einfache Ansätze, da die tatsächlichen Probleme sehr differenziert sind und ein tieferes Einsteigen in die Problematik bedürfen. Da ich auch verschiedene Programmiersprachen verwenden möchte, würde es meinen Rahmen schnell sprengen, wenn ich spezielle Probleme tiefgründig analysiere. Ich denke, es gibt auch bereits genügend Fachliteratur und Internetseiten, wo man sich Anregungen und Lösungsansätze holen kann. Ein weiteres Ziel dieser Zeilen ist es, eine Übersicht und Zusammenfassung zu erhalten, was mit Mikrocontrollern möglich ist.

Mikrocontroller sind heute Bestandteil vieler intelligenter elektronischer Systeme und haben den Vorteil kompakter Flexibilität für fast alle Regelungs- und Steuerungssysteme. Die automatisch agierenden Rasenmäh- oder Staubsauger-Roboter verdanken ihre Existenz den Mikrocontrollern genauso wie intelligente Kaffeemaschinen, Einparkhilfen oder Waschmaschinen. Die rasante Entwicklung hat die Möglichkeiten in diesem Bereich bereits gravierend erweitert und verspricht Automatisierungen in Bereichen, die heute noch unvorstellbar sind. Die Controller werden in fast allen Bereichen unseres Lebens Einzug halten, weil sie immer kostengünstiger herstellbar und durch ihren Einsatz intelligente, verantwortliche Steuerungen unserer Verbrauchsmittel (z.B. Strom, Wärme, Wasser usw.) möglich sind. Sie werden uns stupide Aufgaben abnehmen, wie zählen, kontrollieren, rechnen, schalten im Haus, in Fahrzeugen, bei der Arbeit oder beim Einkauf.

Wer „per Hand“ z.B. Garagentore, Dachfenster, Alarmanlagen usw. steuern will oder wer „per Hand“ Temperaturen oder Zustände in Haus, Auto, Garten oder Geräten auslesen möchte, ist mit Mikrocontrollern gut bedient. Die Ansteuerung von Displays ist dabei genauso möglich wie die Übertragung von Messwerten ins Internet oder der Steuerung über das Smartphone, Internet, per SMS oder Funk.

Die Kombination von Mikrocontrollern mit den Computern macht aus den Zwergen riesige, mächtige Werkzeuge, da Unmengen von Daten, die die Controller liefern, mit leistungsfähigen Computerprogrammen ausgewertet und präsentiert werden können. Analysen, Berichte, Datenbanken mit Webzugriff auf den einen Seite und das Steuern oder festlegen von Regeln am Computer oder Tablett auf der anderen Seiten machen diese Hochzeit zu einer wertvollen Verbindung.

Aber auch autonom agierende Controller können unser Leben stark vereinfachen. Wer automatisch dafür sorgen möchte, dass sich z.B. bei Regen das Dachfenster schließt oder bei Trockenheit die Regenwasserpumpe in Gang gesetzt wird, hat mit Mikrocontrollern den idealen selbstständig agierenden Kandidaten gefunden.

Im Internet findet man mittlerweile ausreichend Beispiele für jedwede Anwendung. Wer einfache Schaltungen zusammenlöten kann, ist in der Lage, die Beispiele an eigene Bedürfnisse anzupassen und nachzubauen.

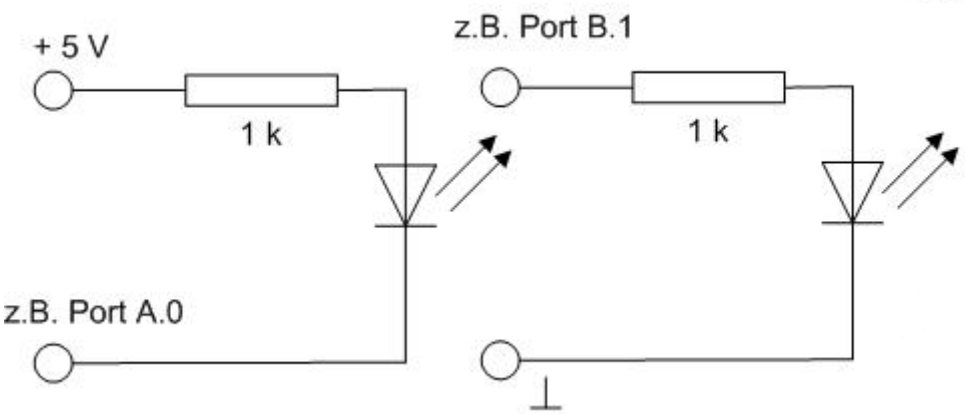
Im Gegensatz zu herkömmlichen elektronischen Schaltungen spart man mit dem Einsatz von Mikrocontrollern wesentlich Material und Aufwand. Im Gegenzug ist es nötig, ein wenig programmieren zu können und sich mit den Eigenheiten der einzelnen Typen von Mikrocontrollern zu beschäftigen.

2. AVR - Mikrocontroller

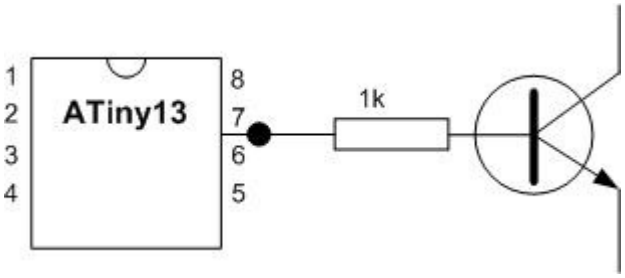
Für ca. einen Euro bekommt man bereits einen Mikrocontroller PIC oder AVR von Atmel mit:
- 8-Bit Prozessor, 8 MHz Taktfrequenz, 8 kByte Flash-Speicher für Programme,
- 512 Byte EEPROM (z.B. zum Ablegen von Daten, die nach dem Ausschalten wieder zur Verfügung stehen sollen)
- mehr als 10 Ein- / Ausgangssignalen zum Steuern
- internen Zählern und Analog / Digitalwandlern (z.B. für Messungen analoger Signale)
- integrierten Funktionen zur Ansteuerung von LCD-Anzeigen bzw. seriellen Datenübertragungen

Im weiteren Verlauf werden wir nur noch die Familie der AVR - Mikrocontroller betrachten. Es gibt die „Kleinen“, ATtiny, mit kleinerem Flash-Speicher für Programme (z.B. 1 kByte) und die „Großen“, ATMega, mit größerer Speicherkapazität. Ich werde nur eine Auswahl davon vorstellen, mit denen ich selbst Erfahrungen gesammelt habe. Es handelt sich nur um 8-Bit Mikrocontroller.

Die Controller haben Ports mit bis zu 8 Pins oder Anschlüssen, die jeweils mit PortA, PortB usw. bezeichnet werden. Auf die Pins greift man z.B. über die Ports zu. Wie in der Entwicklung der Mikroprozessoren kann man die Einstellungen in den Controllern über Register steuern. Das Data Direction Register (DDR) steuert die Ports und bestimmt, ob Ports bzw. Pins als Eingang (Bit = 0) oder Ausgang (Bit = 1) geschaltet werden soll. Dies passiert beim Programmieren der Controller in der Initialisierung. Das Schalten der als Ausgänge definierten Pins erfolgt ebenfalls über das Setzen der entsprechenden Bits auf 1 (Ausgang = 5V) oder 0 (Ausgang = 0V). abei begin die Zählung der Pins mit 0, also z.B. PortA.0. Leuchtdioden (LEDs) können direkt an die Pins angeschlossen werden, da die Ausgänge mit 20 mA belastet werden dürfen. Dabei kann sowohl mit 0, als auch mit 1 „ein“ geschaltet werden, je nachdem wie die Beschaltung erfolgt.

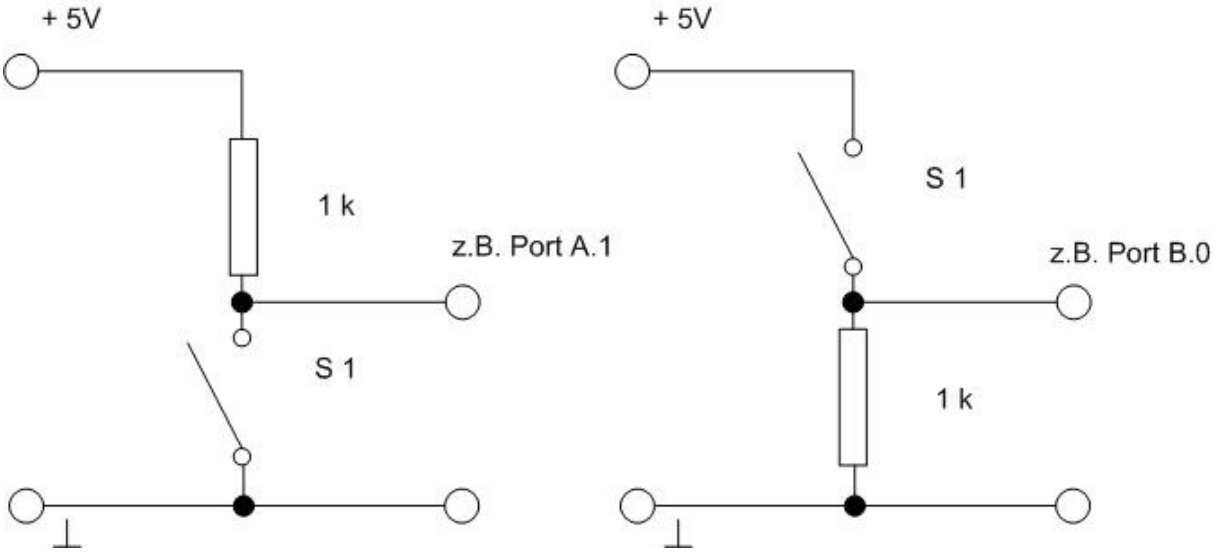


Größere Lasten kann mit Transistoren (z.B. BC 547 C) oder dem Treiberbaustein ULN 2803 schalten.



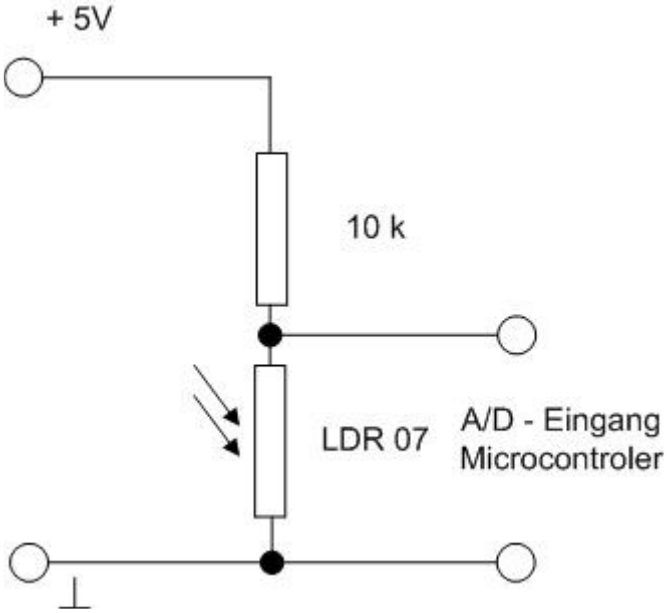
Eingänge können genauso abgefragt werden, indem Taster, Signale oder Schalter den als Eingänge definierten Pins zugeführt werden. Setzt man die Pins auf Eingang (z.B. DDRA = 0x00), dann kann man mit setzten der Ports (z.B. PORTA = 0xFF) die internen Pull Up -

Widerstände einschalten und spart sich die äußere Beschaltung. Natürlich kann man die Schaltung auch ohne interne Pull-Ups aufbauen:



Die Zustände der Pins/Ports müssen dann vom Programm abgefragt werden (z.B. If Pinb.3 = 0 …) und können evtl. den Programmablauf verändern.

Einige ATtinys haben auch Analog-Digital-Converter (ADC) und können direkt Spannungswerte zwischen 0 und 5 Volt in bis zu 10 Bit Tiefe digitalisieren und natürlich auch verwerten (z.B. Lichtmessung mit einem Fotowiderstand).



Wer sich vor 20 Jahren schon mit den damaligen Prozessoren und deren Peripherie beschäftigt hat, wird erstaunt sein, was die Controller heute alles können. Intern hat der Controller Timer und Counter, Interrupt-Steuerungen, ein komplettes Handling von serieller Datenübertragung (I²C, UART, usw.), LCD-Ansteuerung usw. Die umfangreiche Beispielsammlung und die vielen Bibliotheken für die Programme machen die Controller zu Universalgenies.

2.1. ATtiny13

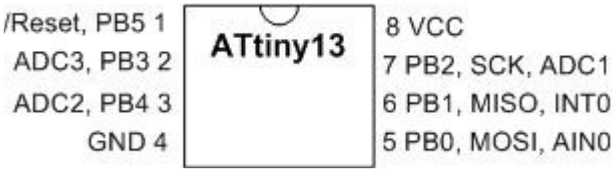
Der ATtiny13 hat 8 Beinchen (Pins) und da immer jeweils eins für Masse und die Betriebsspannung (+ 5 Volt) benötigt werden, bleiben 6 Beinchen für Funktionen übrig. Der Pin 1 ist der Reset-Pin und spielt auch für das Programmieren eine große Rolle, deshalb wird dieser oft nicht mit als Funktions-Pin benutzt. Trotzdem kann man mit dem ATtiny13

bereits sehr viel machen. Mit Peripherie, z.B. einem Schieberegister kann man die Funktions-Pins (I/O – Input/Output) natürlich theoretisch unendlich vergrößern. Der ATtiny hat bereits Analog-Digital-Converter (ADC) und seine interne Taktfrequenz, die an die Schnittstelle angepasst ist, macht ihn zu einem idealen kleinen Mess-Controller.

Anzahl Pins	8
UART (serielle Schnittstelle)	per Software
Taktfrequenz	intern 9.6 MHz / 8, extern max. 20 MHz
Programmspeicher	1 kByte
I/O - Pins	max. 6
externe Interrupts	max. 6
Betriebsspannung	1,8 bis 5,5 Volt
ADC-Pins	4 (10 Bit Genauigkeit)
SRAM*	0,06 kByte
EEPROM*	64 Byte

* siehe Abkürzungsverzeichnis auch für die Pin-Bezeichnungen Quelle: www.atmel.com

Unter www.atmel.com kann man sich das Datenblatt und auch Beispiele herunterladen (Application Notes).



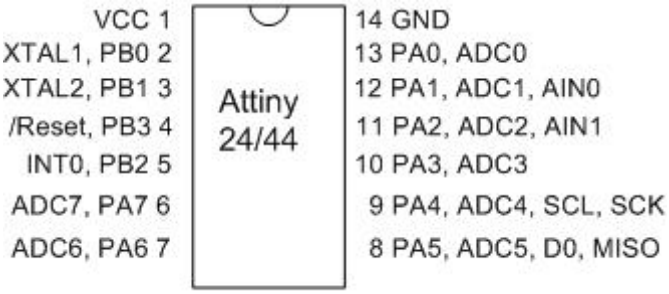
Vorteile sind natürlich die platzsparende Bauweise, die universelle Verwendung und der niedrige Preis. Nachteile könnten die geringe Baudrate und fehlende Pins sein.

2.2. ATtiny24

Der ATtiny24 ist baugleich zum ATtiny44 und ATtiny84 mit jeweils unterschiedlicher Speicherkapazität.

Anzahl Pins	14
UART (serielle Schnittstelle)	per Software
Taktfrequenz	intern 8 MHz / 8, extern max. 20 MHz
Programmspeicher	2 kByte
I/O - Pins	max. 12
externe Interrupts	max. 12
Betriebsspannung	1,8 bis 5,5 Volt
ADC-Pins	8 (10 Bit Genauigkeit)
SRAM	0,12 kByte
EEPROM	128 Byte

Quelle: www.atmel.com



Quelle: www.atmel.com

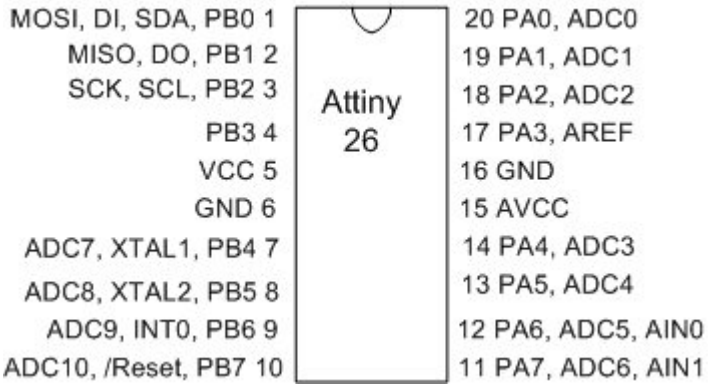
Vorteile sind in jedem Fall, dass es mehr Pins gibt, als beim ATtiny13 und trotzdem der Baustein sehr klein ist. Außerdem hat man mehr Programmspeicher. Bei externer Taktfrequenz hat man einen guten Analog- Digitalwandler-Baustein mit 8 bearbeitbaren Signalen. Die fehlende serielle Hardwareschnittstelle (UART) könnte unter Umständen ein Nachteil sein.

2.3. ATtiny26

Der ATtiny26 hat bereits 20 Pins und ist damit natürlich mit mehr Anschlussmöglichkeiten versehen.

Anzahl Pins	20
UART (serielle Schnittstelle)	per Software
Taktfrequenz	intern 8 MHz / 8, extern max. 16 MHz
Programmspeicher	2 kByte
I/O - Pins	max. 16
Betriebsspannung	2,7 (L-Version) bis 5,5 Volt
ADC-Pins	11 (10 Bit Genauigkeit)
SRAM	0,12 kByte
EEPROM	128 Byte

Quelle: www.atmel.com



Quelle: www.atmel.com

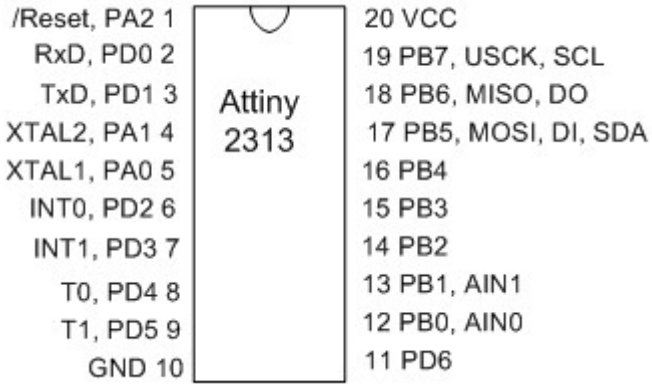
2.4. ATtiny2313

Der ATtiny2313 wird sehr oft verwendet. Im Internet findet man sehr viele Anwendungsbeispiele für diesen Controller. Was ihn so universell einsetzbar macht, ist seine Hardwareschnittstelle (UART), mit der man im Test oder auch in der Endanwendung alle gewünschten Informationen an den Computer übertragen kann. Er hat mehr I/O-Pins als der ATtiny26, weil er keine Referenzspannungsquelle benötigt und dies ist auch gleichzeitig der Nachteil: der ATtiny2313 hat keine Analog-Digitalwandler.

Was den Controller auch so beliebt gemacht hat, ist die Tatsache, dass es einfache Beispiele für Programme im Netz gibt, die ihn zu einem Controller mit USB-Anschluss machen. Die Controller mit Hardware-USB-Anschluss sind in der Regel wesentlich teurer. Zusätzlich hat der Controller Anschlüsse für den Bus von Phillips (I²C) mit der Bezeichnung SCL für den Takt und SDA für die Daten. Damit kann man diese Chips (z.B. den PCF 8574 – Porterweitungsbaustein) direkt ansteuern.

Anzahl Pins	20
UART (serielle Schnittstelle)	per Hardware (PD0 und PD1)
Taktfrequenz	intern 8 MHz / 8, extern max. 20 MHz
Programmspeicher	2 kByte
I/O - Pins	max. 18
Betriebsspannung	1,8 (V-Version) bis 5,5 Volt
ADC-Pins	keine
SRAM	0,12 kByte
EEPROM	128 Byte

Quelle: www.atmel.com



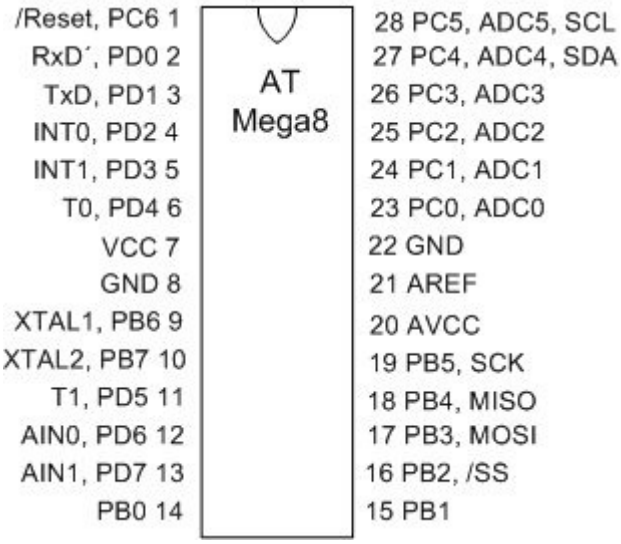
Quelle: www.atmel.com

2.5. ATMega8

Der wesentlichste Unterschied des ATMega zu den Tinys ist natürlich der Programmspeicher von 8 kByte und analog dazu der höhere SRAM und EEPROM-Speicher. Der dazu im Verhältnis geringe Preis macht ihn wieder zum Universal-Controller für alle Anwendungen mit diesem Speicherbedarf. Außerdem hat der Controller natürlich eine Hardwareschnittstelle (UART) und Analog-Digital-Wandler an Board. Auch für diesen Controller gibt es Softwarelösungen für eine USB-Schnittstelle und vielfältige Anwendungen im Internet.

Anzahl Pins	32
UART (serielle Schnittstelle)	1
Taktfrequenz	max. 16 MHz
Programmspeicher	8 kByte
I/O - Pins	max. 23
externe Interrupts	max. 2
Betriebsspannung	2,7 bis 5,5 Volt
ADC-Pins	6 (10 Bit Genauigkeit)
SRAM	1 kByte
EEPROM	512 Byte

Quelle: www.atmel.com



Quelle: www.atmel.com

3. Programmierung

Wer keine Erfahrungen in Programmiersprachen besitzt und auch nicht tiefer in diese eintauchen möchte, für den ist BASIC sicherlich die beste Wahl. BASCOM ist die Basic-Version für die AVR-Mikrocontroller. Die Sprache ist einfach zu lernen und zu verstehen. Da BASCOM eine Hochsprache ist, entscheidet diese notgedrungen über die Verwendung der Ressourcen und speichert z.B. Werte in Registern, dann von dort im SRAM, von dort wieder zurück in Register usw. In Assembler kann man dies direkt steuern und damit zeitkritische Prozesse so optimieren, dass unsinnige Zwischenspeicherungen entfallen. Hochsprachen müssen diesen Weg gehen, damit es einfach und standardisiert wird. Eingeschlossen in die Statements $ASM und $END ASM können in BASCOM auch Assemblerbefehle integriert werden.

Die Programmiersprache C ist weit verbreitet. Viele Betriebsysteme sind damit geschrieben. Es gibt einige Varianten wie z.B. C# oder C++. Die Syntax ist auch für die Mikrocontroller wie gewohnt, allerdings natürlich um die speziellen Funktionen der Mirkocontroller ergänzt. Das sind die Portfunktionen, Bitoperationen und Zugriffe auf Counter und Register. Im Gegensatz zu BASCOM, wo man direkt den Quellcode in den ausführbaren HEX-Code umwandeln kann, wird in C der Code kompiliert, dann mit den Bibliotheken gelinkt und erst dann in den HEX-Code verwandelt. Bespiele für C-Programme, die mit Mikrocontrollern benutzt werden können, findet man sehr oft im Internet und hat damit eine reichhaltige Auswahl zur Verfügung (z.B. www.mikrocontroller.net)

Assembler ist maschinennah. Das hat den Vorteil, dass die Befehle direkt mit dem Controller zu tun haben und nach der Übertragung auf den Controller fast direkt wirken. Das bedeutet dies geschieht sehr schnell. Nachteil ist, dass auch einfache Aufgaben vieler Befehle bedürfen. In Hochsprachen sind diese Standardaufgaben bereits hinterlegt. In Assembler kann man sich auch Standardcode anlegen und sammeln, aber der Sinn alles beeinflussen zu können besteht ja gerade darin, Details direkt ändern zu können und nicht auf fertige Funktionen angewiesen zu sein. Wer sich mit Assembler beschäftigt, kann die Datenblätter von Atmel zu den Mikrocontrollern direkt verwerten, weil die Funktionen der Controller dort oft mit Assembler-Befehlen beschrieben sind. Assembler zu verwenden bedeutet, zu verstehen, was der Controller wirklich macht.

Es gibt auch z.B. Pascal und Forth als Programmiersprachen für AVR-Controller. Es ist natürlich leichter, wenn man die Sprache benutzen kann, die man bereits beherrscht. Falls

man auf der Suche nach der *richtigen* Programmiersprache ist, kann ich nur empfehlen, die Suche abzubrechen. Es ist so ähnlich wie der Streit um die richtige Religion. Man sollte sich auf das zu lösende Problem konzentrieren und den individuell günstigsten Weg dahin finden. Der individuell günstigste Weg könnte auch sein, eine neue Sprache zu lernen, da man diese zukünftig sehr gut gebrauchen kann, auch wenn die bisherige Sprache aus dem bisher „bewohnten Land“, richtig gut zu gebrauchen war. Auch deshalb stelle ich hier drei verschiedene Varianten vor.

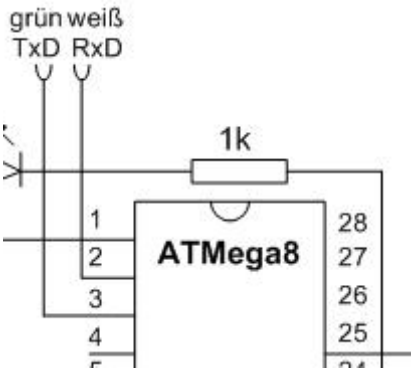
3.1. BASCOM

Für das Programmieren der AVR-Mikrocontroller benötigt man von der Firma MCS Electronics das Programm BASCOM-AVR. Die Entwicklungsumgebung ist auf der Homepage der Firma als Demoversion für das Programmieren bis zu 2 kByte kostenfrei.

Die Programmiersoftware verlangt die Angabe des verwendeten Typs, der Taktfrequenz und der Datenübertragungsrate bei Verwendung der seriellen Schnittstelle. In BASCOM-Basic bedeutet dies z.B.:

```
$regfile = "m8def.dat"
$crystal = 8000000
$baud = 9600
Config Adc = Single , Prescaler = Auto , Reference = Avcc
Print "Sitter Robotics"
Start Adc
Dim W As Word
Do
  Waitms 500
  'ADC messen
  W = Getadc(0)
  Print "Value: " ; W
Loop
End
```

Das Statement $regfile läd die controllerspezifischen Daten aus der zur Entwicklungsumgebung gehörenden Datei (für jeden Typ eine Datei). Das Statement $crystal definiert den verwendeten Prozessortakt unabhängig von den tatsächlich eingestellten Parametern (Look- und Fusebits). Mit $baud wird die Datenübertragungsrate der seriellen Schnittstelle definiert. Config ADC definiert den Analog Digital Konverter und legt z.B. fest, welche Referenzspannung für das zu messende Analogsignal verwendet werden soll. Mehr als ein “Print” ist als Befehl nicht nötig, um dem Controller zu sagen, was er seriell an den Computer übertragen soll. Zwei feste Pins des Controllers sind für die serielle Datenübertragung vorgesehen:



Getadc ist der Befehl zum Auslesen des Analogkonverters. Die Zahl in Klammern gibt den Pin bzw. das entsprechende Beinchen des Controllers an. Kommentare zur besseren Lesbarkeit und Dokumentation des Quellcodes können mit Hochkommata eingefügt werden, z.B. 'ADC messen. Variablen, die verwendet werden sollen, müssen vorher definiert werden. Die Variable W ist als Word-Variable definiert, weil der Digitalwert von 10-Bit eines Analog-Digital-Wandlers nicht in ein Byte passt. Alles, was in die Befehlsfolge Do und Loop eingeschlossen ist, wird endlos ausgeführt. Die Endlosschleife ist die Regel in Controllern, da sie ja selbstständig agieren sollen. Letztendlich ist dies von der Funktionsweise abhängig, die der Controller übernehmen soll.
Das BASCOM-Beispiel ermöglicht es, mit einem Terminal-Programm über die serielle Schnittstelle am Computer die Analog-Werte des Messsignals auszulesen.

Für Neuentwicklungen ist es sehr sinnvoll, die Werte oder auch Variablen-Inhalte über die serielle Schnittstelle an den Computer zu übertragen. Später, am Einsatzort des Controllers können z.B. Messwerte auch in den EEPROM gespeichert werden, damit sie auch bei fehlender Betriebsspannung nicht verloren gehen und bei Bedarf ausgelesen werden können. Das Auslesen von EEPROM-Speicherplätzen ist simbel:

```
For I = 0 To 3
  Readeeprom D , I
  Print D
Next
```

Der Bytevariablen D wird nacheinander der Byte-Speicherplatzwert aus dem EEPROM der Adresse 0 bis 3 (vier Werte, weil die Adresse mit 0 beginnt) zugewiesen und danach über die serielle Schnittstelle gesendet. Das Speichern von Werten in den EEPROM kann entweder über Programmieren in BASCOM erfolgen oder auch während der Laufzeit des Programms:

```
Dim D as Byte
D = 25
Writeeeprom D , 0
```

Dieser Befehl schreibt den Inhalt der Byte-Variablen D (25) auf den ersten EEPROM-Speicherplatz.

Alternativ kann auch ein LCD-Textdisplay an die Controller angeschlossen werden. Standarddisplays sind dabei sehr gut unterstützt und es entfällt die Programmierung der Ansteuerung durch verwenden entsprechender Bibliotheken oder Funktionen in BASCOM.

```
$regfile = "m8def.dat"
$crystal = 8000000
Config Lcdpin = Pin , Db4 = Porta.4 , Db5 = Porta.5 , Db6 = Porta.6 , Db7 = Porta.7 , E = Portc.7 , Rs = Portc.6
Config Lcd = 16 * 2                    'LCD-Typ festlegen
Cls                                    'löscht das LCD
Lcd "Hello world."                     ‘stellt den Text auf dem Display dar
End
```

Andere Displaytypen sind z.B. 16 * 4 oder 20 * 4, 20 * 2 , 16 * 1a (16 Zeichen über 2 Zeilen geteilt). Config Lcdpin ist die Zuordnung der einzelnen Pins des Displays zu den Pins des verwendeten Controllers.

Der I²C-Bus ist in BASCOM (z.B. hier beim ATMega8) auch sehr simpel zu implementieren:

```
Config Scl = PORTC.5
Config SDA = PORTC.4
```

```
Config I2cdelay = 10
I2CInit
Const Pcf_write = &H40

I2CStart
I2cwbyte Pcf_Write
I2cwbyte &HAA
I2CStop
```

Die Pin-Adressen (PC5 und PC4) entsprechen den Anschlüssen des ATMega8. Bei der Benutzung der Hardware TWI muss die entsprechende Bibliothek eingebunden werden, das Softwarebeispiel ist aber in BASCOM nicht langsamer. Je höher der I2Cdelay-Wert eingestellt wird, umso langsamer wird der Bus, was für die Peripherie entscheidend sein kann. Die Start und Stop-Sequenzen sind im I2C-Protokoll festgelegt und werden hier von den BASCOM-Befehlen übernommen. Die Pcf_Write-Adresse (Hexadezimal 40) ist die Adresse des Bausteins PCF8574 mit der Porterweiterung, wenn die Adresspins 1 bis 3 des Bausteins auf Masse gezogen sind. Bei mehreren Bausteinen müssen diese jeweils andere Adressen bekommen, damit man sie getrennt ansprechen kann.

3.2. Programmieren mit C

Für die Programmierung der Controller mit C reicht im Prinzip ein Notepad für das Schreiben des Codes. Zu Empfehlen ist z.B. die Entwicklungsumgebung ‚WinAVR' mit dem ‚Programmers Notepad' oder ‚AVR-Studio' von Atmel. Letzteres ist von der Homepage von Atmel (kostenlos) direkt herunterladbar und hat sehr viele Funktionen. Mit dem Studio werden gleich mehrere Programmiersprachen unterstützt.

In C sollte am Anfang der wichtigste Include-Befehl nicht fehlen (die Extension h steht für Header-Datei):

```
#include <avr/io.h>  // Bezeichnungen für Register und Bits
```

Um Pins eines Controllers als Ausgänge zu definieren, muss man das entsprechende DDR-Register des Ports setzen:

```
DDRD |= (1<<4);
```

Die <<- Zeichen bedeuten das Verschieben (shiften) nach links bis zur 4. Position und die wird dann auf 1 gesetzt. Dieser Befehl würde in BASCOM folgender Maßen aussehen: Config PortD.4 = Output. Beide Befehle definieren das Beinchen PORTD.4 als Ausgang. Ein High-Level an diesem Pin wird in der Folge dann so eingeschaltet:

```
PORTD |= (1<<4);
```

Ein Low-Level am PORTD4 lässt sich im Gegenzug so erzeugen:

```
PORTD &= ~(1<<4);
```

Der letzte Befehl löscht im Prinzip ein einzelnes Bit. Mit diesen beiden Befehlen bleiben alle anderen Definitionen zu Pins an diesem Port, wie sie vorher waren!

In C muss man wissen, dass Eingänge an Controllern im DDR-Register mit 0 definiert werden. Reichte in BASCOM z.B. Config PortD.5 = Input, so muss man in C schreiben:

```
DDRD &= ~(1<<5);
```

Soll der PullUp-Widerstand an diesem Pin gesetzt werden, wird dies so gemacht:

```
PORTD |= (1<<5);
```

Damit kann man jetzt über PortD.4 eine LED ansteuern und über PortD.5 eine Taste abfragen, die den Eingang des Pins bei Betätigung auf Masse zieht:

```
if ( ! (PIND & (1<<5) ))
{
//Code ausführen
}
```

Im Prinzip fragt die IF-Klausel, ob PIND an der 5. Position nicht mit 1 identisch ist.
Mit dem folgenden Befehl kann man ein Bit toggeln, das ist z.B. bei LEDs ganz praktisch:

```
PORTD ^= (1<<4);
```

Wichtige Statements zu Beginn des Quellcodes sind auch die Definition des Prozessortaktes (z.B. 1 MHz):

```
#define F_CPU 1000000
```

Bei Verwenden von Wartefunktionen „_delay_ms(50)“ sollte die Datei:

```
#include <util\delay.h>
```

geladen werden, um auf diese Funktion zugreifen zu können.

3.3. Assembler

Mit Assembler kann man sehr tief in die Hardware der Controller einsteigen und auf alles direkt zugreifen, was ein Controller zu bieten hat. Das bedeutet natürlich auch, dass man sich intensiver mit den Controllern und deren Funktionsweise beschäftigen muss. In den einzelnen Datenblättern ist sehr gut beschrieben, wie die Register funktionieren und welche Einstellungen man vornehmen muss oder kann.
Wenn man bereits C oder BASCOM kennt, sind einige Befehle in Assembler nicht ganz unverständlich. Kommentare beginnen im Assembler mit Semikola. Kommentare sind in Assembler besonders wichtig, weil viele Befehle sich nicht von selbst erklären. Mit dem INCLUDE – Befehl schließt man die Definitionen zu den einzelnen Controllern mit ein, die man nicht selbst definieren muss. Die Datei tn13def.inc ist z.B. beim Programm WAVRASM dabei. Das AVR-Studio von Atmel besitzt natürlich auch alle Funktionen für den Assembler. Die Dateien kommen direkt von Atmel, weil der Hersteller weiß, welche Registernamen welchen Adressen entsprechen. Man sollte also in dieser Datei nichts ändern.

Im Internet gibt es eine gut Seite, die verständlich erklärt, wie man Assembler für die Controller lernen kann:
http://www.avr-asm-tutorial.net

```
; AVR ASM Miniprogramm
.NOLIST
.INCLUDE "tn13def.inc"
.LIST

.DEF mp = R16

      rjmp main

main:
          ldi mp,0b00010000    ;PortB Pin4 auf Out-Direction
```

```
        out DDRB, mp
loop:

        ldi mp, 0b00010000   ;PortB Pin4 auf High
        out PORTB, mp

        ldi mp, 0b00000000   ;PortB Pin4 auf Low
        out PORTB, mp
rjmp loop
```

Statt „ldi mp, 0b00010000“ kann man auch „ldi mp, 1<<4“ schreiben. Dazu wird die 0b00000001 4x nach links geschoben (shift left). Das ist nur für den Assembler und zur möglicher Weise besseren Lesbarkeit, das Ergebnis ist im HEX-File das Gleiche.
Mit .NOLIST wird das Auflisten des Programms abgestellt, mit .LIST wieder eingeschaltet. In dem Beispiel wird mit .DEF mp = R16 der Variablen mp das Register R16 zugewiesen. Damit kann man in mp im Weiteren einen 8-Bit Wert abspeichern. Vorher muss man aber noch die Sprungadresse für den Beginn des Programms nach dem Reset oder Einschalten festlegen. Dies macht man mit dem Befehl rjmp main. ‚main’ ist dabei nur ein Label für eine Adresse. Der Befehl ldi (load immediately) läd eine Konstate sofort in R16 (mp). Dies geht ab Register 16! Die Binary-Zahl 0b00010000 wird von rechts nach links gelesen, d.h. Pin0, Pin1, Pin2, Pin3, Pin4 wird auf 1 gesetzt, falls man noch etwas tut, denn bisher ist die Zahl ja nu rim Register R16. Der Befehl out schreibt den Wert aus Register 16 in das DDR-Register mit ‘out DDRB, mp’. Damit wird dem Controller jetzt gesagt, dass der Pin4 des PortsB als Augang definiert ist. Damit weiß man jetzt, wie der BASCOM oder C-Befehl jetzt auch in Assembler geschrieben wird. Das Setzen der Bits ist dann nur analog.

Das Beispielprogramm ist nicht für den Anschluss einer LED geeignet. Sie würde unserem Auge als immer leuchtend erscheinen und nicht blinken, weil die Frequenz viel zu groß ist. Allerdings kann man sich den Pin4 mit einem Oszilloskop anschauen und stellt fest, dass der Pegel wechselt, genauso, wie man es dem Programm gesagt hat.

3.4. Codemix

Natürlich ist es möglich, sowohl in C als auch in BASCOM den Quellcode mit Assembler zu mixen. Damit kann man die Vorteile von strukturierten, portierbaren Code mit effektiven Routinen mit direkter Hardwarekontrolle verbinden. Das folgende Beispiel ist bei Atmel unter den Application Sheets zu finden:

C:

```
#include "io8515.h"
extern void get_port(void);    /* Function prototype for asm function */
void main(void)
{
DDRD = 0x00;                   /* Initialization of the I/O ports*/
DDRB = 0xFF;
while(1)                       /* Infinite loop*/
{
get_port();                    /* Call the assembler function */
}
}
```

Assembler:

```
NAME get_port
#include "io8515.h"            ; The #include file must be within the module
PUBLIC get_port                ; Declare symbols to be exported to C function
RSEG CODE                      ; This code is relocatable, RSEG
get_port;                      ; Label, start execution here
in R16,PIND                    ; Read in the pind value
```

```
swap R16                ; Swap the upper and lower nibble
out PORTB,R16           ; Output the data to the port register
ret                     ; Return to the main function
END
```

4. Übertragung in den Controller

Die Controller verstehen nur Maschinencode. Das ist auch bei allen anderen „Rechnern“ so. In der Regel ist dies eine HEX-Datei, die den assemblierten Code enthält:

```
:020000020000FC
:1000000000C000E107BB00E108BB00E008BBFBCF7C
:00000001FF
```

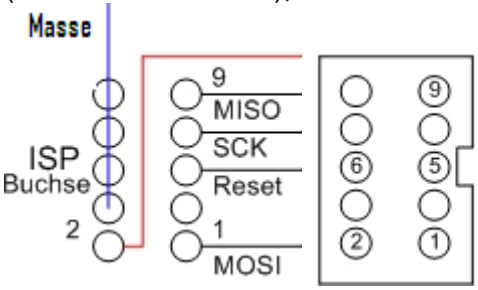
Dies ist das Ergebnis des weiter oben beschriebenen Miniprogramms in Assembler in kompilierter Form. Das HEX-Format ist schon sehr alt und enthält neben dem Maschinenprogramm Kontrollsummen und Adressen.
Wird es mit einem Brennprogramm in den Controller übertragen, sieht das Ganze dann etwa so aus:

```
       00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
0000   C0  00  E1  00  BB  07  E1  00  BB  08  E0  00  BB  08  CF  FB
0010   FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF
```

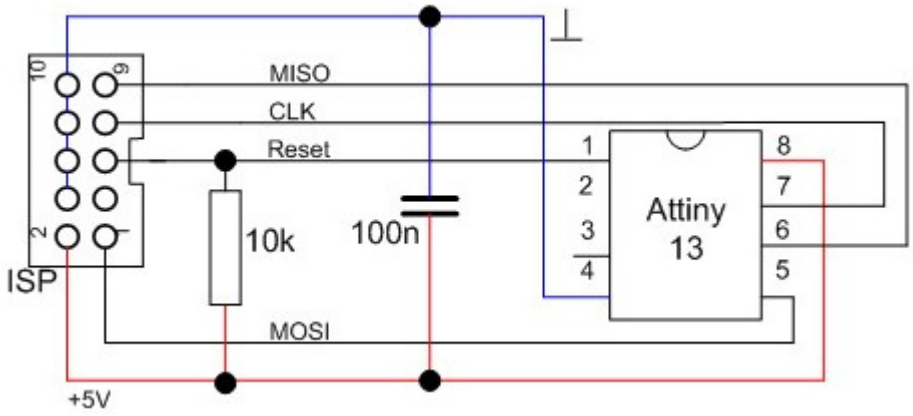
Adressen und Kontrollsummen aus der HEX-Datei entfallen und jeweils 2 Bytes sind vertauscht. Da das Programm sehr klein ist, verbraucht es genau 16 Byte (Adressen von 00 bis 0F). Ab Adresse 0010 steht nur noch FF, das bedeutet unbenutzten Flash-Speicher. Da das Program für den ATtiny13 ist, wurde von 1064 Byte Flash-Speicher dieses Controllers gerade einmal 1,5 % genutzt. Das zeigt auch gleich ein wenig, wie mächtig bereits die kleinen Controller sind und wie man evtl. mit Assembler auch Speicherplatz sparen kann, denn das Assemblerprogramm mit BASCOM geschrieben benötigt nach der Erzeugung des Codes bereits 100 Byte Speicherplatz im Controller, dabei ist der BASCOM-Quellcode auch nicht viel größer:

```
$regfile = "attiny13.dat"
$crystal = 1000000
Config Portb.4 = Output
Do
  Portb.4 = 1
  Portb.4 = 0
Loop
End
```

Die Befehle sind 2 Byte lang, so dass theoretisch ein Befehlssatz von ca. 65.000 Befehlen zur Verfügung steht. Im Normalfall ist die HEX-oder Binärdatei auf der Festplatte des Rechners, wo die Software BASCOM, der Assembler oder Atmel-Studio installiert ist. Um die HEX-Datei in den Controller zu übertragen, kann man verschiedene Schnittstellen des Computers benutzen (seriell, parallel oder USB). Am Controller ist das die ISP-Schnittstelle (ISP = In-System Programmable), die aus den Signalen MOSI (Master In Slave Out), MISO (Master Out Slave In), einem Clocksignal, dem Reset und der Betriebsspannung besteht:

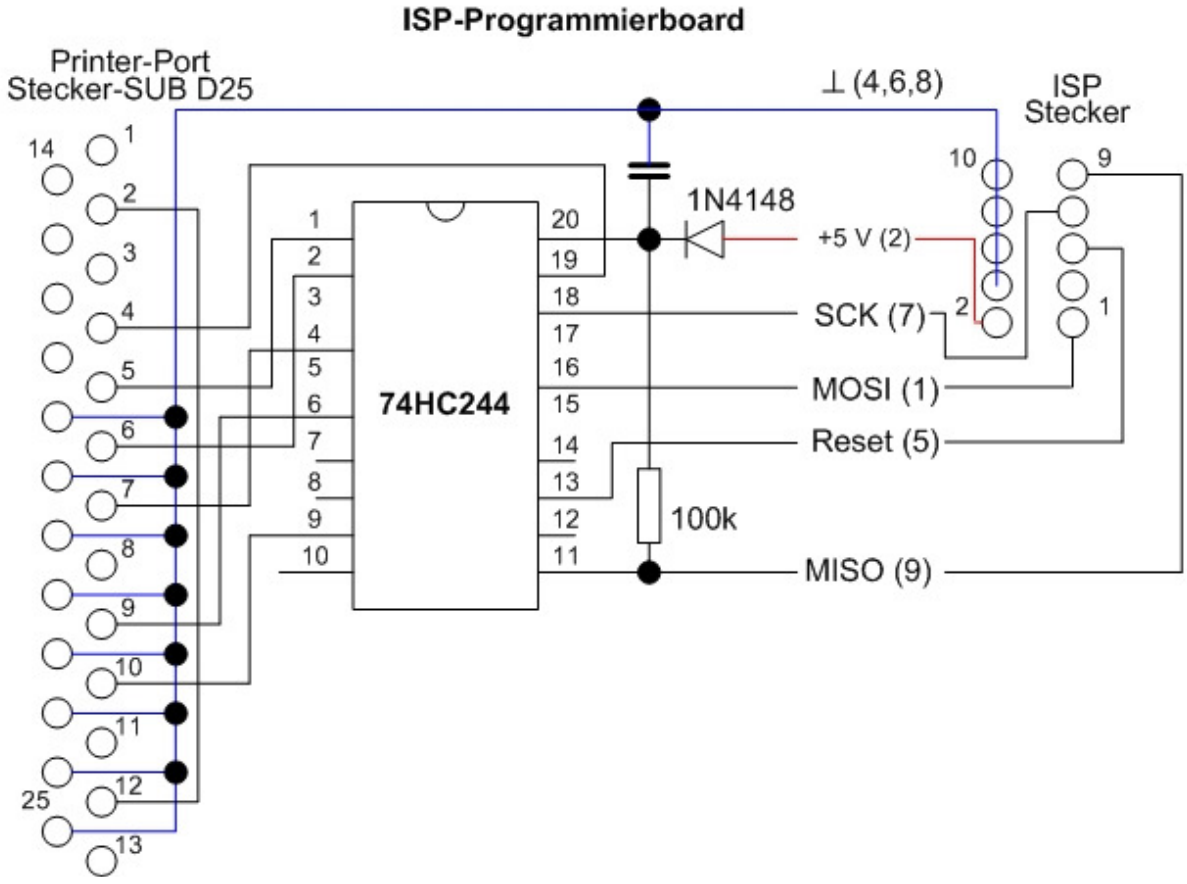


Es gibt 6-polige Steckverbinder oder die dargestellte 10-polige Schnittstelle. In Verbindung mit der absoluten Mindestbeschaltung des ATtiny13 sieht das dann so aus:



Man kann sich ein Evaluation-Board bauen, mit dem man immer wieder Controller programmiert und sie danach in eine endgültige Applikationsschaltung einbaut oder man baut die Schnittstelle mit in die endgültige Applikationsschaltung ein. Das Vorgehen sollte davon abhängig sein, wie viele Controller man programmieren möchte, welche Anwendungen man mit Controllern aufbauen muss und ob die Programmierung oft angepasst werden muss. Es ist sehr gut, dass ISP überhaupt möglich ist, d.h. das Umprogrammieren des Controllers am Einsatzort, ohne ihn ausbauen zu müssen.

Auf der Gegenseite, am Computer, ist je nach der verwendeten Schnittstelle mehr oder weniger Aufwand zu betreiben. Man kann auch mit dem ATtiny2313 einen Programmer für USB aufbauen oder auch einfach einen käuflich erwerben. Hier als Beispiel einen Programmer für eine parallele Druckerschnittstelle:



Bei der Verwendung der Druckerschnittstelle muss der Controller noch mit einer eigenen Spannung versorgt werden. Bei Verwendung der USB-Schnittstelle erfolgt die Versorgung in der Regel über die USB-Schnittstelle z.B. mit USBASP. Hierbei ist eine Treiberinstallation für Windowsbetriebssysteme nötig und die Einstellung des verwendeten Programmierboards in der Programmiersoftware.

Wenn Controller und Computer miteinander verbunden sind, kann die HEX-Datei z.B. aus BASCOM direkt in den Controller übertragen werden. Andere Programme für die Übertragung sind z.B. Ponyprog oder Twinavr.

5. Register und Funktionen

Die AVR-Mikrocontroller haben eine ganze Reihe von Funktionen integriert. Fast alle verwenden Register zur Eistellung, Steuerung oder für Berechnungen.
Die AVR-Controller haben 32 Arbeitsregister zu je 8 Bit (1 Byte). Die Register sind al R0 bis R31 bezeichnet. Zum schnellen Laden mit Konstanten (Befehl LDI) kann man nur die Register R16 bis R31 benutzen. Die letzten 6 Register (R26 bis R31) sind zusätzlich Pointer-Register (die Doppelregister X, Y und Z), die zur Adressierung von Speicheradressen benutzt werden. High- und Low-Teil der drei Doppelregister können mit ZH und ZL angesprochen werden.

Adressen bestehen aus 16 Bit und die dazugehörigen Speicherzellen sind 8 Bit groß (außer dem Programmspeicher, wo die 16-Bit-Befehle stehen). Die Befehle LD (Load) und ST (Store) können Speicherinhalte z.B. einer mit X adressierten Speicherzelle in ein Register geladen werden bzw. aus einem Register in eine adressierte Speicherzelle gespeichert werden.

Das wichtigste Port-Register ist das Statusregister (SREG) und beinhaltet Flags (Flaggen), die z.B. das Ergebnis einer Addition (z.B. Vorzeichen oder Überlauf) kennzeichnen.
Bit7 (I) muss gesetzt sein, wenn Interrupts gleich welcher Art ausführbar sein sollen
Bit6 (T) wird zum Laden und Speichern eines Bits benutzt
Bit5 (H) (Half Carry Flag)
Bit4 (S) Vorzeichen (Sign) Bit
Bit3 (V) enthält Two‘s Complement Overflow Flag
Bit2 (N) enthält das Negativ-Flag von arithmetischen oder logischen Operationen
Bit1 (Z) Null-Flag (Zero-Flag)
Bit0 (C) Carry Flag

Zwei weitere Register enthalten den Stack Pointer (SP), der die Rücksprungadresse beim Zurückkehren aus Unterprogrammen zwischenspeichert. Die Register GICR (General Interrupt Control Register) und GIFR (General Interrupt Flag Register) enthalten die Status und Einstellungen für die Interrupts. Für die Timer- und Counter-Funktionen gibt es die Register TCCR0 (Timer/Counter Control Register), TCNT0 (Timer/Counter Register), TIMSK (Timer/Counter Interrupt Mask Register) und das TIFR (Timer/Counter Interrupt Flag Register). Dazu gehört noch das SFIOR (Special Function Input Output Register). Das MCUCR (Microcontroller Unit Control Register) ist für das Power-Management zuständig. Das SPMCR (Store Program Memory Control Register) steuert einen möglichen Bootloader. Weitere Register sind für die Schnittstellen und für die Analog- Digitalwandler zuständig (UCSR, SPDR, UDR, ADMUX, ADCH, ADCL usw.).

Der Speicherbereich ist so organisiert, dass die Register R0 bis R31 von $0000 bis $001F adressiert sind. Danach folgen die I/O Register von $0020 bis $005F. Ab $0060 beginnt der SRAM-Speicher (beim ATMega 8: 1kByte). Wenn man die nötigen Operationen mit den Registern programmieren kann, ist man sehr schnell, denn diese dauern meist nur einen Prozessortakt. Trotzdem wird man bei aufwendigeren Programmen nicht auf den SRAM-Speicher verzichten können. Mit fester Speicherzelle kann man mit dem Befehl

STS 0x0060, R1 den Inhalt des Registers R1 in den SRAM-Speicher laden und mit
LDS R2, 0x0061 wird der Inhalt der SRAM-Speicherzelle in das Register R2 geladen.

5.1. Timer / Counter

Timer0 ist ein 8-Bit Timer, d.h. ohne weitere Einstellungen zählt er bis 256 und läuft dann über (Timer-Überlauf). Bei einer Taktfrequenz von 8 Mhz bedeutet dies, dass in 1 Sekunde 31.250 Timer-Überläufe stattfinden. Der Timer-Überlauf ist dabei ein Interrupt. Um ihn verwenden zu können, müssen der Timer-Interrupt und generell Interrupts zugelassen werden. Um 1 Sekunde als Messgröße zu bekommen, muss man also 31.250 Überläufe zählen. Wie genau das Ergebnis ist, hängt also von der Genauigkeit des Prozessortaktes ab. Ist ein externer Quarz vorhanden, so ist dieser auch nicht absolut genau. Genauer sind Uhrenquarze, die noch zusätzlich kalibriert werden. Aber auch per Software kann eine Quarzabweichung durch längere Messungen und Vergleich korrigiert werden. Der größte Fehler entsteht aber, wenn die Quarzfrequenz und die Vorteiler ungerade Werte ergeben. Die Nachkommastellen sind dann direkt bereits Abweichungen. Für eine Uhr ist ein Uhrenquarz von 32.768 Hz zu empfehlen, denn in der Digitaltechnik ist dies ja eine „glatte“ Zahl (8000 HEX).

Bei Verwendung als Counter übernimmt ein externes Signal die Funktion des Taktes, sonst ist fast alles ähnlich. Ein sauberes Signal ist dabei natürlich sehr nützlich.

5.2. Interrupts

6. Schaltungen

Vor der Entwicklung der Schaltung sollte die Aufgabe liegen. Durch die zu erledigende Aufgabe ergeben sich der zu verwendende Controller und das zu entwickelnde Programm. Was soll gemessen oder kontrolliert werden (Temperatur, Feuchte, Licht, Tür auf oder zu usw.), wie soll manuell eingegriffen werden (über den Computer, per Tasten, per Tastatur, Infrarot-Fernbedienung usw.) und was wird wie ausgegeben (LCD-Display, Motoren, Relais für die Heizung usw.). Daraus folgt, ob die UART-Schnittstelle des Controllers benötigt wird, ob externe Interrupts nötig sind oder der Anschluss des Analogkomparators AIN0 und AIN1. Für ein LCD Display sind mindestens 6 Anschlüsse nötig oder der I²C-BUS. Beim Zählen externer Impulse sollten die Anschlüsse T0, T1 usw. verfügbar sein. Reicht der interne kalibrierte Taktgeber oder ist der Anschluss eines Quarzes oder eines Oszillators nötig (bei höheren seriellen Datenübertragungsraten oder genauen Frequenzen unerlässlich)? Der benötigte Flash- (Programm), SRAM- und EEPROM-Speicher sollte natürlich abgeschätzt werden. Wie schnell muss das Programm laufen, wie klein oder groß darf die Einsatzschaltung sein und wie viel Strom darf sie verbrauchen?

Wenn man die meisten der Fragen beantworten kann, weiß man auch, welchen Controller man verwenden muss.

6.1. Grundschaltungen

Als Peripherie ist eigentlich je nach Verwendung ein Quarz, ein Programmierstecker (bei In-System-Programmierung), eine Spannungsversorgung und dann Sensoren oder Treiber, Transistoren, Thyristoren oder Relais für die Schaltung von Motoren, Licht, Signalen usw. je nach Verwendungszweck ausreichend. Die meisten konventionellen digitalen Schaltungen wie Trigger, Timer, Frequenzteiler lassen sich oft durch Software im Controller selbst implementieren.

Für die Schaltung des Einsatzortes (z.B. eine temperaturabhängige Lüftersteuerung mit Messwertübertragung an den PC) sind eigentlich nur der Controller, die Verkabelung, die

Spannungsversorgung und ein Transistor für den Lüfter nötig. Bei vielen Aufgaben ist selbst ein einfacher Controller völlig unterfordert, aber trotzdem empfehlenswert, weil der Aufwand sehr gering sein kann.

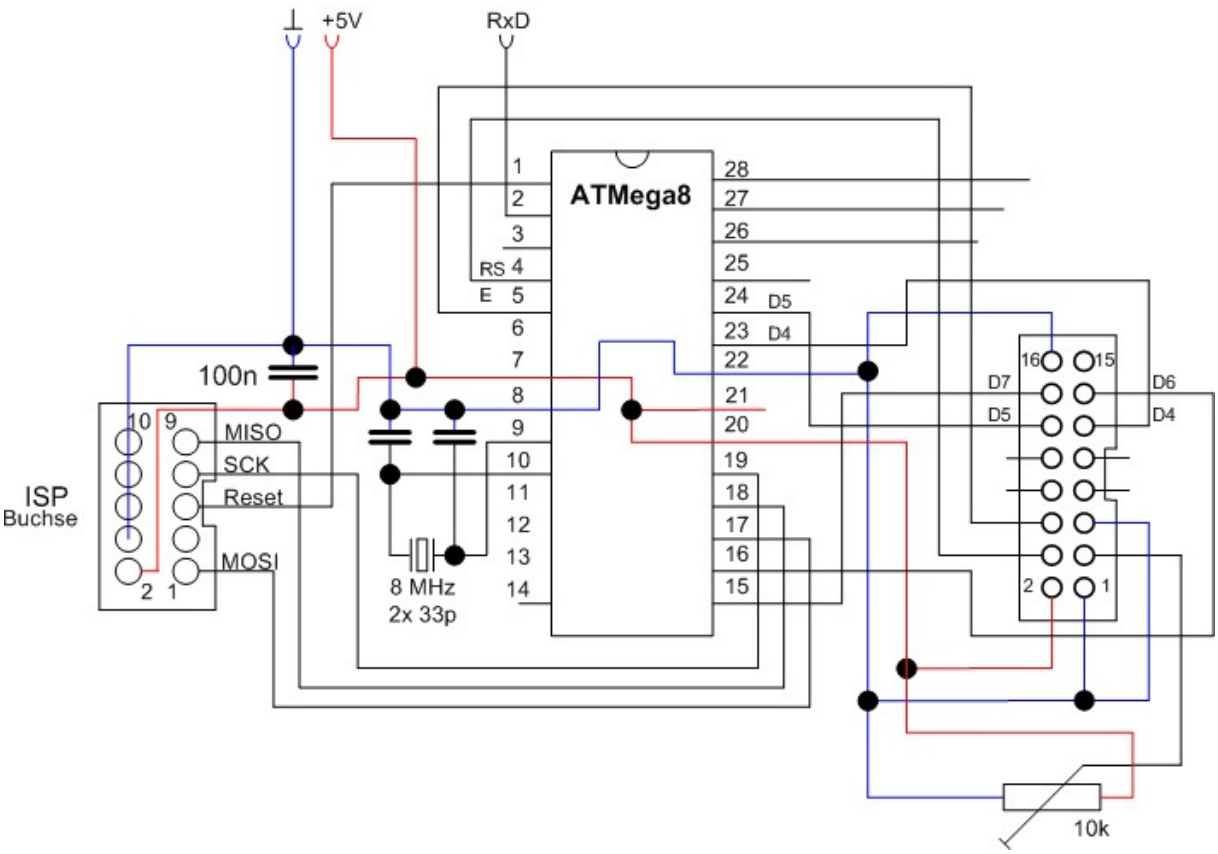
Es kann hier nur eine Auswahl an Peripherie dargestellt werden, da das Spektrum einfach riesig ist. Für die Interaktion sind Signale vom Mikrocontroller (z.B. LED`s, LCD-Anzeigen) und zum Controller (z.B. Taster, Sensoren) zu unterscheiden.

6.2. Text LCD-Ansteuerung

Fast alle textorientierten LCD`s haben die folgenden Anschlüsse (immer das genaue Datenblatt beachten):

Pin	Bezeichnung	Beschreibung
1	GND	Masse
2	VCC	Spannungsversorgung + 5V
3	VEE	Kontrast über Potentiometer (z.B. 10k)
4	RS	Register Select, 1=Daten schreiben / 0=Kommando senden.
5	R/W	1=Read / 0=Write (kann auf Masse gelegt werden, wenn man nur schreibt)
6	Enable	Fallende Flanke -> Übertragen des Kommandos oder der Daten, H-Pegel für Read
7	DB0	Datenbus Bit0 LSB
8	DB1	Datenbus Bit1
9	DB2	Datenbus Bit2
10	DB3	Datenbus Bit3
11	DB4	Datenbus Bit4
12	DB5	Datenbus Bit5
13	DB6	Datenbus Bit6
14	DB7	Datenbus Bit7 MSB

Bei der Ansteuerung im 4-Bit Modus benötigt man nur die Anschlüsse DB4 bis DB7. Ein Daten- oder Befehlsbyte wird dann jeweils als Halbbyte übertragen. Das spart Leitungen und Pins an den Controllern. Minimal benötigt man 6 Pins, 4 für das Halbbyte, eins für Enable und eins für RS. Hat das Display eine Hintergrundbeleuchtung, so gibt es noch zwei weitere Anschlüsse für die Anode und Kathode.



In dem Beispiel wird der 4-Bit Modus verwendet und nur Daten oder Befehle an das Display gesendet. Selbstverständlich ist es auch möglich, die Stromversorgung für das Display selbst und auch eine evtl. Hintergrundbeleuchtung über den Controller mit zu steuern.
Die meisten Bibliotheken bieten bereits benutzerfreundliche Funktionen zur Ansteuerung der Displays an, aber man kann die Befehle auch selbst generieren, da sie nicht sonderlich kompliziert sind:

Befehl	**RS**	**R/W**	**DB7**	**DB6**	**DB5**	**DB4**	**DB3**	**DB2**	**DB1**	**DB0**	**Beschreibung**
Clear Display	0	0	0	0	0	0	0	0	0	1	Löscht das Display und setzt den Cursor auf den Anfang der 1. Zeile (Addresse 0).
Cursor Home	0	0	0	0	0	0	0	0	1	*	setzt den Cursor auf den Anfang der 1. Zeile (Addresse 0)
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Setzt die Cursor Bewegungsrichtung (I/D), spezifiziert das Display zu schieben (S). Diese Operationen werden während des Daten lesen/schreiben durchgeführt.
Display on/off Control	0	0	0	0	0	0	1	D	C	B	Schaltet an/aus: das gesamte Display (D), Den Cursor (C) Cursor blinken (B).
Cursor/Display shift	0	0	0	0	0	1	S/C	R/L	*	*	Setzt Cursor Bewegung oder Display Bewegung (S/C), Bewegungsrichtung (R/L)

Function set	0	0	0	0	1	DL	N	F	*	*	Einstellen der Schnittstellen Datenlänge (DL), Anzahl Display Zeilen (N) und Zeichen Font (F).
Set CGRAM Address	0	0	0	1	CGRAM Adresse						Setzen der CGRAM Adresse. CGRAM Daten werden gesendet und empfangen nach dem setzen.
Set DDRAM Address	0	0	1	DDRAM Adresse							Setzen der DDRAM Adresse. DDRAM Daten werden gesendet und empfangen nach dem setzen.
Read busy-flag and address counter	0	1	BF	CGRAM / DDRAM Adresse							Liest das Busy-flag (BF), welches anzeigt das interne Operationen ausgeführt werden, und liest den CGRAM oder DDRAM Adress Zeiger Inhalt.
Write to RAM	1	0	Schreib Daten								Schreibt Daten zum CGRAM oder DDRAM.
read from RAM	1	1	Lese Daten								Liest Daten vom CGRAM oder DDRAM.

Quelle: http://www.roboternetz.de

Stehen noch weniger als sechs Pins zur Verfügung, empfiehlt sich der Weg über eine Porterweiterung z.B. mit dem I²C – Bus oder auch mit Schieberegistern (z.B. 4092).

6.3. Sensoren

Die meisten Sensoren können ihre Messwerte in Spannungs- oder Frequenzwerte umsetzen, die dann direkt von den Mikrocontrollern verarbeitet werden können. Die Abtastrate von analogen Signalen ist dabei immer eine Frage der benötigten Genauigkeit. Auch die Sensoren selbst haben allerdings Toleranzen und fast alle sind darüber hinaus temperaturabhängig.

Die NTC`s mit negativem Temperaturkoeffizienten sind günstige Temperatursensoren für einfache Messungen. Der Widerstand ändert sich aber leider nicht linear, so dass der Temperaturwert errechnet oder aus einer Wertetabelle übersetzt und ausgelesen werden muss. Der I²C Chip LM75 liefert dagegen seinen Wert direkt als Temperatur.

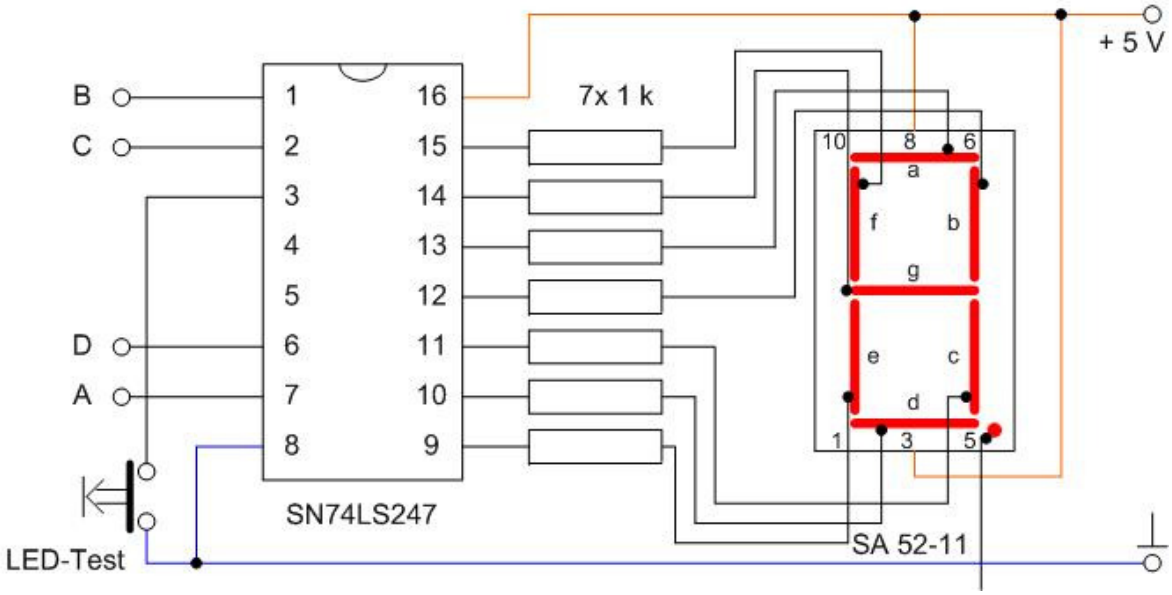
Spannungsmessungen über entsprechende Spannungsteiler sind mit den ADC-Pins der Controller kein Problem. Strommessungen können entweder über den Weg des Spannungsabfalls z.B. über einen Shunt-Widerstand durchgeführt werden oder besser potientalfrei durch Hall-Sensoren, die wiederum (als kompaktes Sensorsystem) lineare Spannungen in Abhängigkeit des fließenden Stromes liefern.

Fotowiderstände, Fototransistoren und Fotodioden werden wie in der herkömmlichen Elektronik verschaltet und die (evtl. angepassten) Spannungswerte den ADC-Pins der Controller zugeführt.

6.4. 7-Segment-Anzeigen, BCD-Decoder und Schieberegister

7-Segment-Anzeigen, die meist mit LED`s realisiert sind, können bei mehreren Stellen multiplex betrieben werden, um Leitungen zu sparen. Da unsere Augen Frequenzen ab ca. 50 Hz nicht mehr wirklich unterscheiden können, können die einzelnen Stellen eines

Displays zeitlich hintereinander angesteuert werden. Eine andere Variante ist die Ansteuerung über Schieberegister. Auch die Verwendung von BCD zu 7-Segment-Decodern kann einen Controller entlasten. Es kommt im Einzelfall immer auf die gewünschte Applikation an.

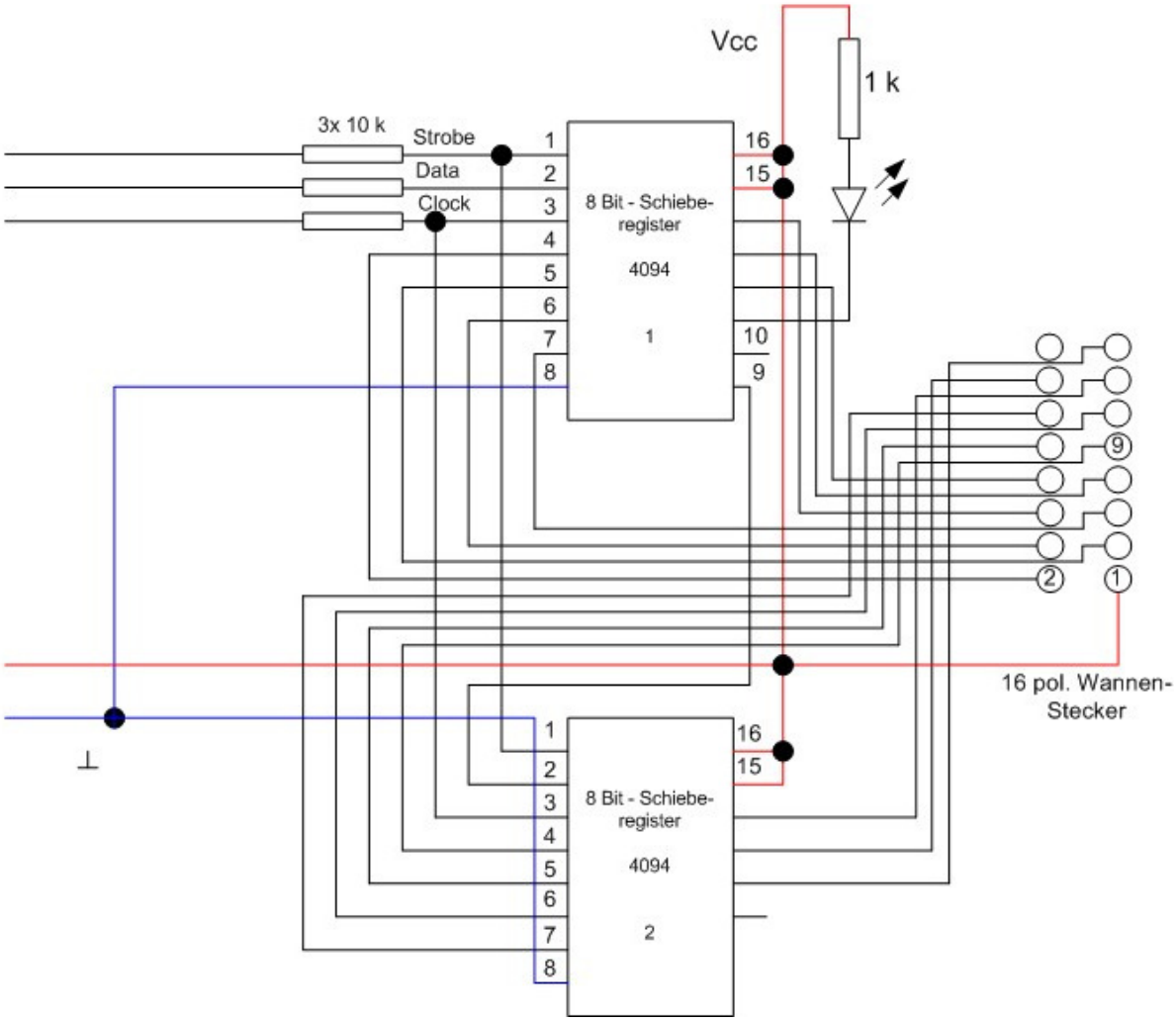


Dargestellt ist ein 7-Segment-Anzeige-Modul mit gemeinsamer Anode und dem BCD zu 7-Segment-Decoder 74LS247:

A	B	C	D	Anzeige
0	0	0	0	0
1	0	0	0	1
0	1	0	0	2
1	1	0	0	3
0	0	1	0	4
				usw.

Damit kann man ganz leicht einen halben Port z.B. B0 bis B3 damit verbinden und einfach die darzustellende Ziffer an den Port schicken.

Die Verwendung von Schieberegistern ermöglicht rein theoretisch unendlich viele Ports. Die Bedienung der Schieberegister ist denkbar einfach. Die Bits (0 oder 1) werden am Data-Signal produziert und das Signal mit dem Clock-Signal (0-1-0) freigeggeben. Sind alle Bits übertragen, erfolgt das Strobe-Signal (0-1-0) für die Freigabe der Ports. Wenn man Schieberegister kaskadiert, wird das Strobe-Signal einfach erst dann erzeugt, wenn 16, 24, 32 oder mehr Bits übertragen wurden. Man muss nur darauf achten, dass das letzte Bit unter Umständen am unteren Ende ankommt, nicht am letzten Schieberegister, weil die Kaskadierung vom Signal aus dem Controller bis zum letzten Schieberegister erfolgt und das letzte Byte im Schieberegister verbleibt, welches die Signale vom Controller zuerst empfängt. Für zwei 7-Segmetanzeigen benötigt man 14 Bit bzw. mit Komma, Punkt oder einer Kontroll-LED genau 2 Byte:



Ein Beispielprogamm in Assembler für 8-Bit könnte so aussehen:

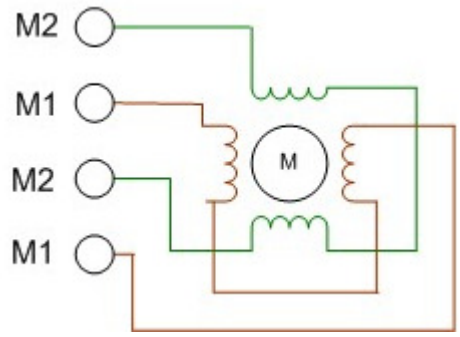
```
; AVR ASM Testprogramm Attiny12
; www.sitter.de
; Berlin, den 12.08.2008
; Schieberegisteransteurung über eine Schleife
; höchstes Bit (links) als erstes schieben
; wird zu A0 im 4094 letztes ist Bord-LED
; 8 Bit für 7-Segment-Anzeige
; Definitions-Werte im Flash
;
.NOLIST
.INCLUDE "tn12def.inc"
.LIST
;
.DEF mp = R16
.DEF temp = R20
.DEF temp1 = R21
.EQU Clock=2  ; PB2 Pin 7 (CLK)
.EQU Data=1   ; PB1 Pin 6 (MISO)
.EQU Strobe=0 ; PB0 Pin 5 (MOSI)
;
.org 0x0000
```

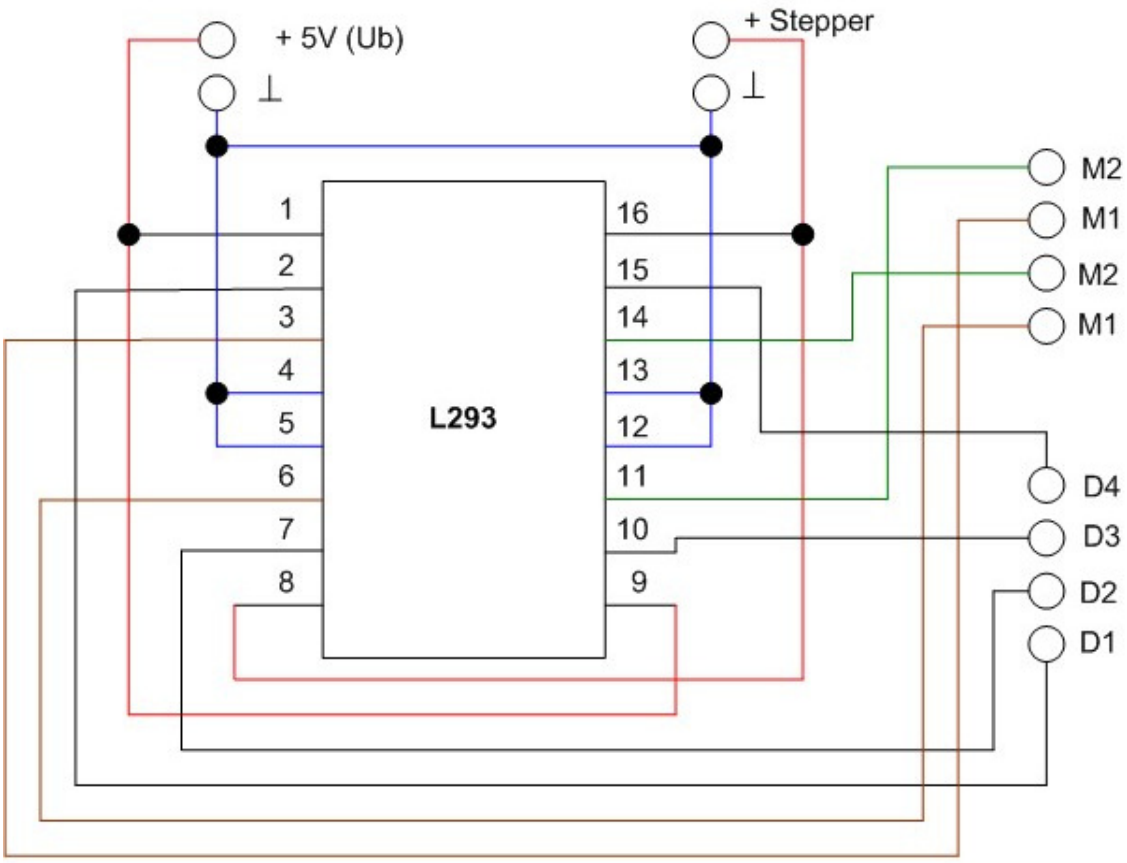
```
      rjmp main                         ;Reset-Handler
main:
        ldi temp, 0b00010111            ;PortB PB4 Pin 3 auf Out-Direction
        out DDRB, temp                  ;Datenrichtungsregister setzen
;
        ldi    temp, 0                  ;und den Zähler (r1) initialisieren
        mov    r1, temp
;
loop:
        ldi    ZL, LOW(Codes*2)         ;die Startadresse der Tabelle in den
        ldi    ZH, HIGH(Codes*2)        ;Z-Pointer laden
;
        mov    temp1, temp              ;die wortweise Adressierung der Tabelle
        add    temp1, temp              ;berücksichtigen
;
        add    ZL, temp1                ;und ausgehend vom Tabellenanfang
        adc    ZH, r1                   ;die Adresse des Code Bytes berechnen
;
        lpm                             ;dieses Code Byte in das Register r0 laden
;
        mov    mp, r0                   ;Wert in Arbeitsregister mp (r16) laden
;
;serielle Ausgabe
                        ldi  R17, $08   ;Schleife der seriellen Ausgabe
next:
                        CBI PORTB, Data ;Data auf 0 setzen
                        SBRC mp,0       ;überspringe, wenn Bit 0 = NULL
                        SBI PORTB, Data ;setze Data auf 1
                        LSR mp          ;Logical Schift right (Bit 0 in C-Flag)
                        NOP
                        NOP
                        SBI PortB, Clock ;Clock wird Eins
                        NOP
                        NOP
                        CBI PortB, Clock ;Clock wird Null
                        NOP
                        NOP
                        DEC R17
                        brne next       ;Branch if Not Equal
;nächstes Bit
                        NOP
                        NOP
                        SBI PortB, Strobe ;Strobe wird Eins
                        NOP
                        NOP
                        CBI PortB, Strobe ;Strobe wird Null
;
;ende serielle Ausgabe eines Byte
;
        inc    temp                     ;den Zähler erhöhen, wobei der Zähler
        cpi    temp, 10                 ;immer nur von 0 bis 9 zählen soll
        brne   wait
        ldi    temp, 0
;
wait:   ldi    r17, 10                  ;und etwas warten, damit die Ziffer auf
wait0:  ldi    r18, 0                   ;der Anzeige auch lesbar ist, bevor die
```

```
wait1:    ldi     r19, 0                ;nächste Ziffer gezeigt wird
wait2:    dec     r19
          brne    wait2
          dec     r18
          brne    wait1
          dec     r17
          brne    wait0
;
          rjmp    loop                  ;auf zur nächsten Ausgabe
;Ende Programm
;EEPROM - Daten
Codes:                                  ;Die Codetabelle für die Ziffern 0 bis 9
                                        ;sie regelt, welche Segmente für eine bestimmte
                                        ;Ziffer eingeschaltet werden müssen
;
          .db     0b00000011            ; 0: a, b, c, d, e, f
          .db     0b10011111            ; 1: b, c
          .db     0b00100101            ; 2: a, b, d, e, g
          .db     0b00001101            ; 3: a, b, c, d, g
          .db     0b10011001            ; 4: b, c, f, g
          .db     0b01001001            ; 5: a, c, d, f, g
          .db     0b01000001            ; 6: a, c, d, e, f, g
          .db     0b00011111            ; 7: a, b, c
          .db     0b00000001            ; 8: a, b, c, d, e, f, g
          .db     0b00001001            ; 9: a, b, c, d, f, g
```

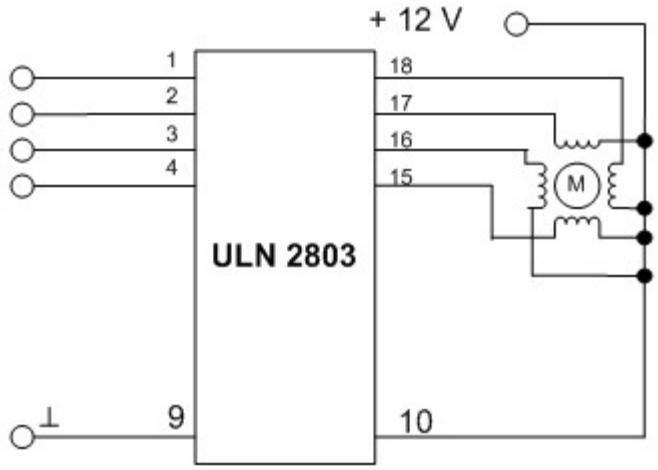
6.5. Schrittmotoren

Bipolare Schrittmotoren steuert man am besten mit dem Schaltkreis L293 an:





Schrittmotoren mit gemeinsamer Anode kann man auch mit einem Treiberschaltkreis wie ULN2803 ansteuern.



7. Quellen:

http://www.mikrocontroller.net

http://www.atmel.com/

http://www.mcselec.com

Programmieren der AVR RISC Mikrocontroller mit BASCOM AVR von Claus Kühnel, Skript Verlag 2004

Messen – Steuern – Regel mit dem C-Control/BASIC-System von Burkhard Kainka, Franzis-Verlag 1997

http://www.avr-asm-tutorial.net

http://www.roboternetz.de

8. Abkürzungsverzeichnis:

ADC = Analog-Digital-Converter
AIN = Analog Comparator Negative Input
ALU = Arithmetic Logic Unit
BCD = Binär Code Digit
CLK = Clock (Taksignal)
DAC = Digital-Analog-Converter
EEPROM = Electrically Erasable Programmable Read-Only Memory
I/O = Input/Output
IDE = Integrated Developing Environment
XTAL = Crystal-Anschluss (Taktgenerator, Quarz)
PCINT = Pin Change Interrupt
RTC = Real Time Clock
MCU = Microcontroller Unit
MOSI = Master In Slave Out
MISO = Master Out Slave In
ISR = Interrupt Service Routine
RxD = Receiver-Daten
PWM = Pulse Width Modulation (für DAC)
CTC = Clear Timer on Compare
OC = Output Compare
SS = Slave Select
CS = Chip Select
OE = Output Enable
DI = Digital Input
ICP = Input Capture Pin
REF = Referenzspannung (für ADC)
ISP = In-System Programmable
SRAM = Static random-access memory (Statisches RAM)
SREG = Statusregister
SPH = Stack-Pointer High
SPL = Stack-Pointer Low
TIFR = Timer Interrupt Flag-Register
TIMSK = Timer Interrupt Maskierungs-Register
TxD = Transmitter-Daten
SCL = Two Wire - Serial Bus Clock Line (I^2C Bus)
SDA = Two-wire Serial Bus Data Input/Output Line (I^2C Bus)
UART = Universal Asynchronous Receiver Transmitter (serielle Schnittstelle)