

# MSE-Projekt P7: Aktiver Datenlogger

Matthias Krebs

18. Januar 2010

---

Institut für Mobile und Verteilte Systeme, Fachhochschule Nordwestschweiz, Windisch

Projektstufe: P7 (1. Semester MSE)

Student: Matthias Krebs

Advisor: Christoph Stamm, Hans Buchmann

Auftraggeber: Scintilla AG

# Inhaltsverzeichnis

<b>1 Einführung</b>	<b>4</b>
<b>2 Theorie</b>	<b>6</b>
2.1 Messtechnik	7
2.2 Bussysteme	10
2.3 Mikrocontroller	12
<b>3 Hardware</b>	<b>13</b>
3.1 Anforderungen	14
3.2 Konzept	15
3.3 Werkzeuge	18
3.4 Komponenten	22
3.4.1 Sensoren	22
3.4.2 Mikrocontroller	24
3.5 Das STK600-Experimentierboard	26
3.6 Das Sensor-Interface	27
3.6.1 Schema des Sensor-Interface	27
3.6.2 Herstellungsprozess	27
3.6.3 Testaufbau	28
<b>4 Software</b>	<b>32</b>
4.1 Mikrocontroller-Entwicklungsumgebung	33
4.2 Mikrocontroller-Software	34
4.2.1 Funktionsbibliothek	34
4.2.2 Hauptprogramm	38

---

4.2.3	Testprogramme	39
4.3	Client-Software	41
<b>5</b>	<b>Projektverlauf</b>	<b>43</b>
5.1	Zeitplan	44
5.2	Projektphasen	45
5.3	Ausblick	48
<b>6</b>	<b>Anhang</b>	<b>50</b>
6.1	Glossar	51
6.2	Dank	52
	Quellen	53

# **Kapitel 1**

## **Einführung**

## Vision

Professionelle Elektrowerkzeuge werden von Handwerkern heutzutage in unterschiedlichsten Situationen eingesetzt. Dies kann Grossbaustellen im Freien beinhalten, auf denen Beton bearbeitet wird, Umbauten im Innern von Gebäuden, aber unter anderem auch die Herstellung von Gegenständen aus Metall oder Holz. Diese diversifizierte Nutzung bedeutet, dass die Werkzeuge völlig unterschiedliche Nutzungsprofile besitzen und je nach Profil auf andere Weise belastet werden.

Die Firma Scintilla AG, Tochterfirma von Bosch und Herstellerin verschiedenster Werkzeuge für den professionellen Einsatz, möchte die Nutzungsprofile ihrer Fabrikate genauer analysieren, um zukünftige Entwicklungen und Produktverbesserungen gezielt auf die Kundenbedürfnisse hin optimieren zu können. Um dieses Ziel zu erreichen, möchte Scintilla einen Teil ihrer Werkzeugpalette mit Sensoren und einem Datenlogger ausstatten und diese präparierten Werkzeuge einer repräsentativen Menge von Handwerkern überlassen, welche die Werkzeuge normal im Rahmen ihrer Tätigkeiten einsetzen. Die daraus gewonnenen Informationen möchte Scintilla auswerten und in ihren Entwicklungsprozess einfließen lassen.

## Ziele

Das Projekt „Aktiver Datenlogger“ umfasst drei Hauptziele:

1. Entwicklung eines Sensor-Interface zum Einbau in die Werkzeuge
2. Entwicklung eines Datenloggers zum Aufzeichnen und Übertragen der Messdaten
3. Entwicklung einer Software zur Auswertung der Messdaten

Das Teilprojekt P7 im ersten Semester des MSE-Studiums umfasst den ersten Punkt, die Entwicklung des Sensor-Interface.

Zur Entwicklung eines guten Konzepts müssen Testexemplare einiger Bosch-Werkzeuge, in die später das Sensor-Interface eingebaut wird, analysiert werden, um einerseits die Funktionsweise zu verstehen, aber auch, um herauszufinden, ob und wo genug Platz im Gehäuse ist, um das ganze Sensor-Interface oder Teilkomponenten davon einbauen zu können.

Passende Sensoren, welche den Anforderungen entsprechen, müssen gefunden und beschafft werden. Sind die Sensoren bestimmt, so ist eine Platine zu entwickeln, auf die die Sensoren aufgelötet und mit einem Mikrocontroller angesteuert werden können. Dabei muss für den Mikrocontroller eine passende Software entwickelt werden, damit die erfassten Sensor-Messwerte seriell an den PC übertragen werden können. Ist der Testaufbau fertig, so sind alle Sensoren und Funktionen ausgiebig zu testen.

Wenn das Grundkonzept mit den Sensoren funktioniert, so muss ermittelt werden, wie man das Konzept so umsetzen kann, dass alle Komponenten des Sensor-Interface in das Werkzeug eingebaut werden können.

## **Kapitel 2**

# **Theorie**

## 2.1 Messtechnik

### Strom, Spannung und Leistung

Für Spannung und Strom bei ohmschen Lasten gilt das ohmsche Gesetz  $U = R \cdot I$ .

Die Messung der Spannung erfolgt normalerweise mit Hilfe eines sehr hochohmigen Messwiderstands, dessen Spannungsabfall gemessen wird. Er muss sehr hochohmig sein, weil er parallel zur eigentlichen Last liegt (Lastwiderstand) und sich durch parallele Widerstände der Gesamtwiderstand stark reduziert.

Zur Messung des Stroms existieren unterschiedliche Konzepte. Das einfachste Konzept ist ein sehr niederohmiger Messwiderstand (Shunt) in Serie zur Last. Aus dem Spannungsabfall über dem Messwiderstand kann die Stromstärke abgeleitet werden. Bei hohen Strömen kann dies problematisch werden, da viel Strom durch den Messwiderstand fließt, und die Existenz des Messwiderstands die Spannung über der Last beeinflusst. Besser ist die direkte Messung des Stroms mit Hilfe eines Magnetfelds. Im Falle von Wechselstrom erzeugt die ständig wechselnde Stromrichtung ein Magnetfeld im Kabel, welches mit einem passiven Hallsensor gemessen werden kann. Im Sensor wird dadurch ein Strom induziert, welcher durch einen Messwiderstand geleitet wird, dessen Spannungsabfall gemessen und daraus die Stromstärke abgeleitet wird. Bei Gleichstrom funktioniert dies leider nicht, da in diesem Fall kein Magnetfeld erzeugt wird.

Die Leistung errechnet sich im ohmschen Fall einfach durch  $P = U \cdot I$ . Gerade bei Lasten am Stromnetz kommen jedoch noch andere Komponenten hinzu, weshalb u.U. Schein- und Blindleistung berücksichtigt werden müssen. Integrierte Schaltungen zur Leistungsmessung können dies allerdings unterscheiden.

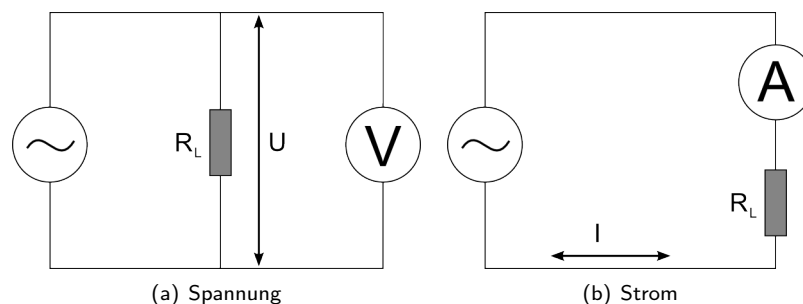


Abbildung 2.1: Messung von Spannung und Strom einer ohmschen Last bei Wechselstrom

### Temperatur

Die elektronische Temperaturmessung basiert für gewöhnlich auf einem einfachen Prinzip: Das Bauteil, welches als Temperaturfühler dient (z.B. Widerstand oder Diode) hat eine bestimmte Kennlinie. Im Falle eines Widerstands steigt der Widerstand mit steigender Temperatur, im Falle einer Diode ändert sich die Durchlassspannung. Um daraus die Temperatur errechnen zu können, kann der Spannungsabfall über dem Widerstand gemessen werden. Die Temperaturabhängigkeit ist allerdings normalerweise nicht linear, weshalb eine solche Messung kompliziert werden kann, und der Sensor erst kalibriert werden muss. Temperatursensoren mit integrierter Schaltung sind normalerweise so konstruiert, dass sie in ihrem definierten Messbereich eine quasi lineare Kennlinie besitzen, was die Messung vereinfacht.

## Räumliche Lage

Um die räumliche Lage eines Gegenstands elektronisch bestimmen zu können, bedarf es eines dreiachsigen Beschleunigungssensors. Anhand der Anteile der Erdbeschleunigung auf den drei Achsen kann die Lage relativ zum Erdboden ermittelt werden. Natürlich funktioniert dies nur, so lange der Sensor nicht durch andere Beschleunigungskräfte beeinflusst wird. Er muss also ruhig gehalten werden, andernfalls wird anstelle der räumlichen Lage hauptsächlich die Bewegung gemessen.

Für die Beschleunigungsmessung gibt es verschiedene Technologien, z.B. mittels Kreisel (Gyro) oder kapazitiv. Integrierte Sensoren verwenden in der Regel die kapazitive Methode, bei der winzige bewegliche Elemente verbaut werden, deren Bewegung kleinste Änderungen der Kapazität eines Kondensators hervorruft.

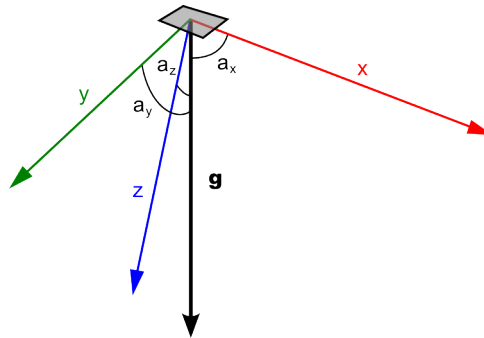


Abbildung 2.2: Messung der Lage mittels Beschleunigungssensor

Der Winkel einer Achse zum Beschleunigungsvektor errechnet sich folgendermassen:  $\cos\alpha_x = \vec{x} \cdot \vec{g}$

## Drehzahl

Die Drehzahl eines Motors wird bei den meisten Techniken auf dieselbe Weise gemessen: Pro Umdrehung werden ein oder mehrere elektrische Impulse generiert, welche mit Hilfe eines Taktgebers und eines Zählers in eine Drehzahl umgerechnet werden. Zur Generierung der Impulse stehen unterschiedliche Technologien zur Verfügung.

### Encoder

Ein Encoder besteht im Wesentlichen aus einem Rad, welches radiale Schlitze enthält und auf der Motorachse montiert wird, und einer Lichtschranke. Durch die Drehung des Motors wird die Lichtschranke immer dann unterbrochen, wenn das Licht nicht durch einen Schlitz hindurchkommen kann. Diese periodischen Unterbrüche werden nun in einen elektrischen Impuls umgewandelt, welcher gemessen werden kann.

Mit einem Encoder kann die Drehzahl sehr präzise gemessen werden. Nachteilig ist, dass der Encoder physikalisch mit der Motorachse verbunden werden muss und relativ viel Platz in Anspruch nimmt.

### Reflexlichtschranke

Eine weitere optische Variante ist die Reflexlichtschranke. Das Prinzip ist ähnlich wie beim Encoder. Allerdings wird hierbei auf einem kleinen Sektor der Motorachse ein Reflektor angebracht, während die Lichtschranke

so konzipiert ist, dass sie das am Reflektor reflektierte Licht empfängt. Da der Reflektor nur einen schmalen Sektor des Achsenumfangs ausmacht, schaltet die Lichtschranke nur ein Mal pro Umdrehung und nur während einer kurzen Zeit. Der resultierende elektrische Impuls kann nun gemessen werden.

Mit der Reflexlichtschranke kann die Drehzahl sehr platzsparend gemessen werden. Problematisch sind lediglich Verunreinigungen der Achse oder der Lichtschranke, welche zu Fehlfunktionen führen können.

### **Hall-Sensor**

Ein Hall-Sensor ist ein stromdurchflossener Leiter, der sich in einem senkrecht dazu ausgerichteten Magnetfeld befindet. Da im Sensor immer ein Strom fließt, kann darüber eine Spannung gemessen werden. Ändert sich das Magnetfeld, so fließt ein zusätzlicher Strom im Sensor, wodurch sich die gemessene Spannung ändert.

Um nun eine Drehzahl messen zu können, wird der Hall-Sensor so am Motor befestigt, dass er von dessen Magnetfeld beeinflusst werden kann. Durch die Drehung des Motor entsteht ein sich änderndes Magnetfeld, das nun mit Hilfe des Hall-Sensors in elektrische Impulse umgewandelt werden kann, welche wiederum gemessen werden können.

## 2.2 Bussysteme

Zur Vernetzung digitaler integrierter Schaltungen auf einer Platine bedarf es eines passenden Datenübertragungssystems. Zu diesem Zweck existieren verschiedene Bussysteme, über die ICs miteinander kommunizieren können.

### SPI

SPI (Serial Peripheral Interface) wurde von Motorola speziell zur Kommunikation von einzelnen ICs entworfen. Es arbeitet nach dem Master-Slave-Prinzip, wobei es einen einzelnen Master-IC (gewöhnlich ein Mikrocontroller) gibt, welcher mehrere Slave-ICs ansteuern kann.

SPI arbeitet mit mind. 5 Leitungen: Masse (Common), zwei Datenleitungen (Master-Out/Slave-In, Master-In/Slave-Out), Master-Clock, und Chip-Select. Datenleitungen und Clock sind auf dem Bus gemeinsam, aber pro Slave wird eine separate Chip-Select-Leitung benötigt.

Zur Kommunikation mit einem Slave aktiviert der Master die entsprechende Chip-Select-Leitung, um den Slave zu aktivieren. Danach wird mit jedem Clock-Impuls ein Bit synchron übertragen, wobei immer gleichzeitig ein Bit gesendet und eins empfangen wird.

Eine Fehlererkennung oder eine Sende- und Empfangsbestätigung gibt es nicht, Master und Slave müssen einfach von der Korrektheit der Daten ausgehen.

SPI wird in der Regel nur zur lokalen Vernetzung von ICs verwendet, da das Protokoll für eine Übertragung über grosse Distanzen zu störanfällig ist. Der Vorteil ist jedoch die sehr einfache Implementation.

### I<sup>2</sup>C

I<sup>2</sup>C wurde von Philips entwickelt, um eine Datenübertragung mit möglichst wenig Leitungen zu ermöglichen. Deshalb sind bei I<sup>2</sup>C nur eine Masseleitung, eine Clock-Leitung und eine bidirektionale Datenleitung nötig. I<sup>2</sup>C erlaubt den Betrieb von mehreren Master- und Slave-ICs, die sich alle den Bus teilen.

Master-ICs müssen den Bus zuerst für sich reservieren, um Daten übertragen zu können. Das geschieht, indem sie die Datenleitung auf Low ziehen, andere ICs erkennen dies und warten, bis der Bus wieder frei ist.

Bei der Datenübertragung darf die Datenleitung ihren Pegel nur ändern, während die Clock-Leitung auf Low ist. Ist die Clock-Leitung auf High, so wird der Pegel der Datenleitung als gültiges Bit definiert.

Das erste Datenbyte enthält die Adresse des Slave-ICs, der angesteuert werden soll. Der Slave-IC antwortet mit einem ACK-Bit, dass er bereit ist. Danach kann entweder der Master oder der Slave senden, wobei jedes Byte wiederum von der Gegenseite bestätigt werden muss. Ist die Übertragung beendet, setzt der Master den Bus auf idle (Clock- und Datenleitung auf High).

I<sup>2</sup>C erfordert eine komplexere Implementierung als SPI, hat allerdings den Vorteil, dass weniger Leitungen benötigt werden. Zudem kann eine grosse Anzahl ICs auf demselben Bus kommunizieren. Deshalb wird I<sup>2</sup>C insbesondere dort eingesetzt, wo viele ICs Daten austauschen müssen und die Anwendung nicht so zeitkritisch ist.

### RS232

RS232, auch als UART bezeichnet, ist ein asynchroner serieller Datenbus, der vor allem zur Verbindung von verschiedenen externen Geräten verwendet wird. Im einfachsten Fall genügen drei Leitungen: Eine gemeinsame Masse (Common) und zwei Datenleitungen zum Senden und Empfangen (TXD, RXD). Bei RS232 sind die Signalpegel folgendermassen definiert: 5..15V für High und -15..-5V für Low. Es existiert auch eine Variante

mit TTL-Pegel (5V für High, 0V für Low), welche SCI genannt wird.

Die Bitrate beträgt normalerweise zwischen 300 bps und 115 kbps.

Die Datenübertragung erfolgt asynchron, d.h. es gibt keinen gemeinsamen Taktgeber. Beide Seiten müssen sich selbst um die Synchronisation kümmern. Das Sampling erfolgt typischerweise 16x schneller als die Bitrate, weshalb kleine Unterschiede im Takt keine Auswirkungen haben.

Ein Datenframe besteht immer aus einer 0 als Startbit, 5 bis 9 Datenbits und einer 1 als Stopbit. Zusätzlich kann vor dem Stopbit ein Parity-Bit eingefügt werden, das die Anzahl 1en gerade oder ungerade macht. Auf diese Weise können einzelne Bitfehler detektiert werden. Nach jedem Frame erfolgt eine Synchronisation, weshalb bei kleinen Taktunterschieden dennoch kein Fehler auftritt.

RS232 kann in Form von SCI zur Vernetzung von ICs auf einer Platine verwendet werden, allerdings wird es hauptsächlich zur Vernetzung von Geräten über die Distanz von einigen Metern verwendet, da die Stömpfindlichkeit wegen der relativ grossen Pegeldifferenz nicht so gross wie bei 5V-Logik ist.

## 2.3 Mikrocontroller

Als Mikrocontroller bezeichnet man für gewöhnlich Embedded-Prozessoren, welche ein komplettes System-On-Chip enthalten, also einen CPU-Kern, RAM, Flash-Speicher und I/O-Funktionalität. Diese Controller enthalten kein eigentliches Betriebssystem, sondern einen kompilierten Programmcode, welcher beim Einschalten ausgeführt wird. Dadurch sind Mikrocontroller sehr flexibel und können für unterschiedlichste Steueraufgaben eingesetzt werden.

### Atmel AVR

Die AVR-Serie von Atmel ist eine weit verbreitete Mikrocontroller-Familie, welche sowohl von Hobbyelektronikern als auch Profis eingesetzt wird. Die Produktpalette ist sehr variantenreich, es gibt eine Vielzahl von AVR's mit unterschiedlichen Speicher-Größen, Anzahl I/O-Pins und verschiedenen Zusatzfunktionen wie externem RAM, USB, RS232 usw.

Allen gemeinsam ist ein spezieller 8Bit-RISC-Kern, ein paar KB RAM sowie Flash-Speicher für den Programmcode und etwas EEPROM-Speicher für persistente Datenspeicherung. Was I/O betrifft, so gibt es interne und externe Interrupts, Timer, verschiedene Datenbusse wie SPI, I<sup>2</sup>C und UART.

Die AVR's können mit Assembler programmiert werden, sie sind jedoch auch auf Hochsprachen optimiert, weshalb es Compiler für diverse Hochsprachen wie C, C++ oder Basic gibt. Zur Übertragung des Programmcodes vom PC wird ein Programmiergerät benötigt, welches den AVR via SPI flasht.

Neben einer grossen Anzahl von AVR-Varianten bietet Atmel verschiedene Experimentier-Boards an, mit denen sich ein Entwickler sehr schnell in die AVR-Entwicklung einarbeiten kann. Alternativ findet man im Internet zahlreiche Bauanleitungen für Experimentier-Boards, Programmiergeräte, weiteres Zubehör sowie Programmierbeispiele, was die Verwendung von AVR's attraktiv macht. Hinzu kommt der günstige Preis von ein paar Franken für einen der kleineren AVR's.

## **Kapitel 3**

# **Hardware**

## 3.1 Anforderungen

### Betriebsbedingungen

Umgebungstemperatur: -20 - 70°C  
Luftfeuchtigkeit: bis 92%

### Messparameter

Die folgenden Messparameter werden während eines Arbeitszyklus erfasst:

- |   |  |
|---|--|
| - Beginn des Arbeitszyklus bzw. Einschaltzeitpunkt:                     | Datum, Uhrzeit                           |
| - Dauer des Arbeitszyklus (Betriebszeit):                               | 1 - 60 s, typisch 10 s                   |
| - Spannung alle 0.5 s:  | 30 - 230 V, typisch 200 V                |
| - Stromfluss alle 0.5 s:  | 0.5 - 12.0 A, Auflösung 0.05 A           |
| - Leistung alle 0.5 s:  | 50 - 2000 W, typisch 800 W               |
| - Drehzahl alle 0.5 s (Motoren mit Drehzahlgeber 12 Impulse/Umdrehung): | 1000 - 35000 rpm                         |
| - Temperatur zu Beginn und Ende des Arbeitszyklus:                      | -20 - 1500°C, typisch 45°C               |
| - Lage im Raum:   | zeigt an, wie das Werkzeug gehalten wird |

### Umwelteinflüsse

In einem industriellen Umfeld sind diverse Störquellen vorhanden, welche den Betrieb eines Geräts negativ beeinflussen können. Folgende Einflüsse müssen berücksichtigt werden, damit später ein störungsfreier Betrieb möglich ist:

- Mechanische Belastung
- Verunreinigungen
- Elektrostatische Entladungen
- Innere und äussere elektrische Störungen
- Störungen durch Funksignale

## 3.2 Konzept

Dieser Abschnitt beinhaltet das Grundkonzepts des Projekts, welches für den weiteren Verlauf gewählt wird. Weitere mögliche Konzepte sind in [WY08] und [KR09] beschrieben.

### Einbauvariante

Die für den Anwendungsfall von Scintilla sinnvollste Variante ist die Variante, bei der die Sensoren und das Sensor-Interface direkt im Werkzeug eingebaut werden, der Datenlogger jedoch extern, d.h. beispielsweise am Stromkabel. Der Grund hierfür ist, dass innerhalb der Werkzeuge schlicht nicht genug Platz ist, um mehr als das Sensor-Interface unterzubringen.

Der Wunsch von Scintilla ist, dass das gesamte System erstens möglichst klein ist und zweitens die Arbeit des Handwerkers, der das präparierte Werkzeug verwendet, in keinster Weise behindert.

Je nach Platzangebot in den unterschiedlichen Werkzeugen müssen die einzelnen Sensoren auf mehrere Platinen verteilt werden, da eine einzige Platine in der Regel keinen Platz hat.

Da nicht alle Werkzeuge denselben Funktionsumfang haben, können nicht benötigte oder nicht sinnvolle Sensoren je nach Werkzeug auch weggelassen werden.

### Testplatine

Die Testplatine, welche ein erstes experimentelles Sensor-Interface darstellt, besteht aus einer einzigen Platine, welche sämtliche Sensoren und optional einen Mikrocontroller enthält. Sie besitzt externe Anschlüsse für eine 5V-Speisung, RS232-Konnektivität und einen Anschluss zur Messung der Gerätespannung (max. 230VAC). Die Messung des Stroms erfolgt durch Durchschlaufen der stromführenden Leitung durch den Stromwandler-Baustein.

Die Testplatine kann alternativ auch an ein Mikrocontroller-Experimentierboard angeschlossen werden. In diesem Fall erfolgt die Datenübertragung über herausgeführte Pins.

Zum Datenaustausch zwischen den ICs auf der Testplatine kommt das SPI-Protokoll zum Einsatz. Alle Sensor-ICs müssen dieses Protokoll beherrschen, damit ihre Messwerte mit einem einheitlichen Protokoll digital ausgelesen werden können. Auf die Verwendung von ICs mit analogen Ausgängen wird bewusst verzichtet, da die analogen Signale viel anfälliger für Störungen von aussen sind.

### Messung der Spannung und des Stroms

Dieser Abschnitt beschreibt die Aufbereitung von Spannung und Strom, so dass die Parameter von einem integrierten Sensor erfasst werden können.

Wird ein digitaler Mess-IC verwendet, so besitzt dieser in der Regel analoge Eingänge zur Messung der Spannung. Die Spannung wird dann anhand des Eingangspiegels via AD-Wandler digital abgespeichert. Die Messung der Stromstärke erfolgt in der Regel ebenfalls durch Messung einer Spannung, da der IC gerade bei höheren Strömen von mehreren Ampere nicht einfach als Durchflussleiter dienen kann. Die Lösung ist, den Spannungsabfall über einem kleinen Messwiderstand zu messen.

Die Verwendung eines Messwiderstands in Serie zur Last ist keine gute Lösung, da der Messwiderstand aktiv in den Stromkreis eingreift. Zudem ist der u.U. grosse Stromfluss ein Risiko, und es gibt keine galvanische

Trennung.

Gerade bei Verwendung einer Wechselspannung bietet sich eine recht einfache Lösung in Form eines Stromwandler-Transformators an. Dieser ist ein vergossener Block mit einem Loch in der Mitte, durch das der stromführende Leiter verlegt werden kann. Das Wicklungsverhältnis ist beispielsweise 1000:1. Fließt nun ein Wechselstrom durch den Leiter, so entsteht ein Magnetfeld, welches im Stromwandler einen kleinen Strom induziert. Bei einem Verhältnis von 1000:1 würde also bei 1 A im Leiter 1 mA im Stromwandler fließen. Dieser Strom fließt durch einen Messwiderstand, dessen Spannungsabfall vom Mess-IC gemessen werden kann.

Bei Gleichstrom (im Hinblick auf akkubetriebene Werkzeuge) funktioniert dieses Prinzip nicht, da kein wechselndes Magnetfeld anliegt. In diesem Fall müsste man einen gleichstromtauglichen Wandler verwenden. Der Nachteil ist, dass dieser eine positive und eine negative Versorgungsspannung braucht.

Zur Messung der Spannung genügt es, die zu messende Spannungsquelle an den Eingang des IC zu hängen. Bei Verwendung von 230 V muss diese Spannung natürlich heruntertransformiert werden, da der IC in der Regel nicht direkt 230 V messen kann. Durch Verwendung eines kleinen Transformator erhält man erstens eine kleinere Spannung und zweitens eine galvanische Trennung. Alternativ kann auch ein Trenntrafo mit einem Wicklungsverhältnis von 1:1 nehmen, der dann lediglich als galvanische Trennung dient. Die Spannung wird dann über einen Spannungsteiler aus Widerständen reduziert.

Bei Akkubetrieb ist die galvanische Trennung nicht von Bedeutung, da sowieso alle Komponenten von derselben Niedervolt-Gleichspannungsquelle gespeist werden und denselben Massebezug besitzen.

Das Konzept der Messung mit dem ADE7753 [ADE7] entstammt der Dokumentation des dazugehörigen Evaluationsboards [ADEE].

Messungen (Abbildung 3.2) ergeben, dass das Konzept aus Trafo und Spannungsteiler wie gewünscht funktioniert. Die Abweichung von der (linearen) Ideallinie ist dabei im Bereich von 0.05 V, was schon im Bereich der Messgenauigkeit des Multimeters liegt.

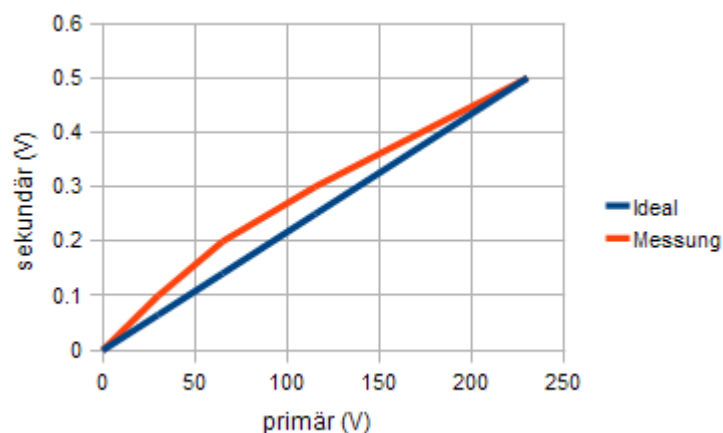


Abbildung 3.1: Messung mit Trafo und Spannungsteiler

## Drehzahlmessung

Die Drehzahl kann beim Werkzeug GST 135 BCE analog zur Abbildung 3.6 gemessen werden. Das Ausgangssignal des Drehzahlgebers kann mit einem externen Interrupteingang des Mikrocontrollers verbunden werden. Dank der Rechteckform können die Impulse im Mikrocontroller innerhalb einer bestimmten Zeit gezählt und daraus die Drehzahl berechnet werden. Da das Signal des Drehzahlgebers zwar einen differentiellen Pegel von

ca. 5 V hat, aber nicht galvanisch vom 230V-Netz getrennt ist, sollte ein Optokoppler dazwischengeschaltet werden. So wird verhindert, dass zwischen dem Ausgangssignal des Drehzahlgebers und der Logikschaltung mit dem Mikrocontroller gefährliche Potentialdifferenzen auftreten können. Sollte die Schaltung der Drehzahlregelung nicht genug Strom liefern können, um den Optokoppler anzutreiben, kann noch ein Transistor dazwischengeschaltet werden. Grundsätzlich sollte der Ausgang des Drehzahlgebers möglichst wenig belastet werden, da dies gemäss Scintilla die Drehzahlregelung stören könnte.

Messungen der Frequenz des Ausgangssignals des Drehzahlgebers (Abbildung 3.2) ergeben, dass das Drehzahlstellrad eine fast lineare Regelung der Drehzahl erlaubt.

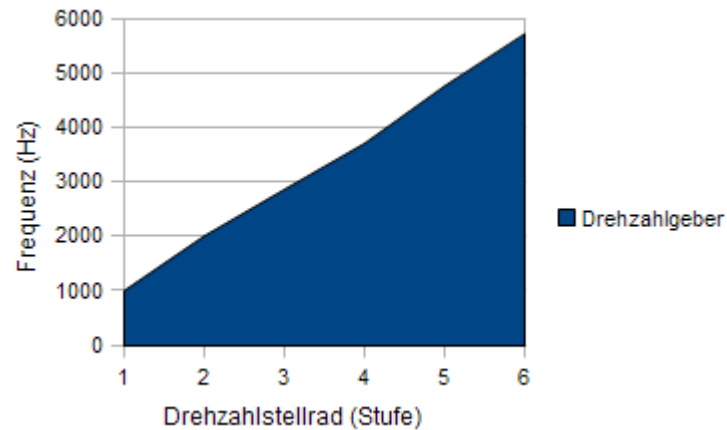


Abbildung 3.2: Messung des Drehzahlgebersignals

### 3.3 Werkzeuge

Grundsätzlich möchte Scintilla das Datenerfassungssystem in einer Vielzahl von Geräten einsetzen, sowohl in netz- als auch in akkubetriebenen Werkzeugen. Für den Anfang beschränken wir uns aber auf einige wenige Werkzeuge, welche mit 230V Netzspannung betrieben werden.

Die hier aufgeführten Bosch-Werkzeuge wurden von Scintilla zu Testzwecken zur Verfügung gestellt.

#### GEX 150 AC



Abbildung 3.3: GEX 150 AC

Die GEX 150 AC ist ein Exzentrerschleifer mit 340 W Leistung. Der 230V-Wechselstrommotor hat im Leerlauf eine Drehzahl von 4500-12000 rpm, wobei die Schwingzahl des Schleiftellers mit 9000-24000 rpm doppelt so hoch ist. Der Schleifteller hat einen Durchmesser von 150 mm und kann über eine Art Klettverschluss mit unterschiedlichen Schleifblättern bestückt werden.

Die Motordrehzahl kann mit einem Drehzahlstellrad voreingestellt werden, der Hauptschalter dient lediglich als Ein-/Aus-Schalter. Die Regelung erfolgt über eine Phasenanschnittsteuerung auf einer kleinen Steuerplatine, eine aktive Leistungsregelung gibt es nicht.

Der Platz im Innern des Gehäuses und damit Möglichkeiten für den Einbau von Sensoren und Elektronik ist stark begrenzt, lediglich im vorderen Teil oberhalb des Motors ist etwas Platz, sowie innerhalb des Handgriffs vor und hinter dem Hauptschalter. Die restlichen Gehäuseteile lassen sich nur schwer demontieren, deshalb wurde von einer weiteren Demontage abgesehen.

#### GST 135 BCE

Die GST 135 BCE ist eine Handstichsäge. Die maximale Leistungsaufnahme beträgt 720 W, wobei der Motor eine Abgabeleistung von 310 W besitzt. Der maximale Hub beträgt 26 mm und der maximale Schnittwinkel 45°. Beide Parameter können mechanisch eingestellt werden.

Im Leerlauf beträgt die Hubzahl zwischen 500 und 2800 rpm. Als einziges der zur Verfügung gestellten Werkzeuge besitzt die GST 135 BCE eine aktive Drehzahlregelung, das bedeutet, dass der Motor unter Last die



Abbildung 3.4: GST 135 BCE

Leistung bei Bedarf erhöht, um die Drehzahl konstant zu halten. Die eigentliche Motorsteuerung erfolgt über eine Phasenanschnittssteuerung.

Die maximale Drehzahl kann mit einem Stellrad konfiguriert werden, zusätzlich hat der Hauptschalter Marke Kautt & Bux ein eingebautes Potentiometer, das über den Schaltweg geregelt wird und mit der Drehzahlregelung verbunden ist.

Die Drehzahlmessung erfolgt über einen Hallsensor, welcher mit Hilfe eines Transistors ein Rechtecksignal erzeugt, welches, gemessen an der Leerlaufdrehzahl, ungefähr 120 Impulsen pro Umdrehung entspricht (siehe Abbildung 3.3). Das Übersetzungsverhältnis des Getriebes ist nicht bekannt, weshalb hier direkt die Enddrehzahl referenziert wird. Die Messung wurde mittels Oszilloskop direkt an der Steuerplatine durchgeführt (siehe Abbildung 3.6).

Die GST 135 BCE bietet mehrere Möglichkeiten zum Einbau von zusätzlichen Komponenten. Innerhalb des Handgriffs ist sowohl vorne als auch hinten etwas Raum vorhanden, ebenso unter der hinteren Abdeckung, unter der sich die Steuerplatine befindet.

### GSB 21-2 RE

Die GSB 21-2 RE ist eine Handbohrmaschine. Die Aufnahmeleistung beträgt 1100 W, die Abgabeleistung des Motors 630 W. Ein Zweigangetriebe, welches mechanisch umschaltbar ist, sorgt für ausreichend Drehmoment bei unterschiedlichen Materialien. Die Drehzahl des Bohrfutters ist dann zwischen 0 und 580 resp. 1900 rpm regelbar. Eine Drehmomentregelung gibt es allerdings nicht, der Motor wird lediglich über eine Phasenanschnittssteuerung geregelt, die bereits im Hauptschalter (Marquardt Serie 2500) integriert ist. Das Stellrad für die maximale Drehzahl befindet sich ebenfalls am Hauptschalter.

Desweiteren gibt es noch mechanische Schalter für die Wahl zwischen Bohren und Schlagbohren sowie Einstellen der Drehrichtung.

Der Platz für zusätzliche Komponenten ist begrenzt, lediglich im Handgriff direkt unterhalb des Motors ist

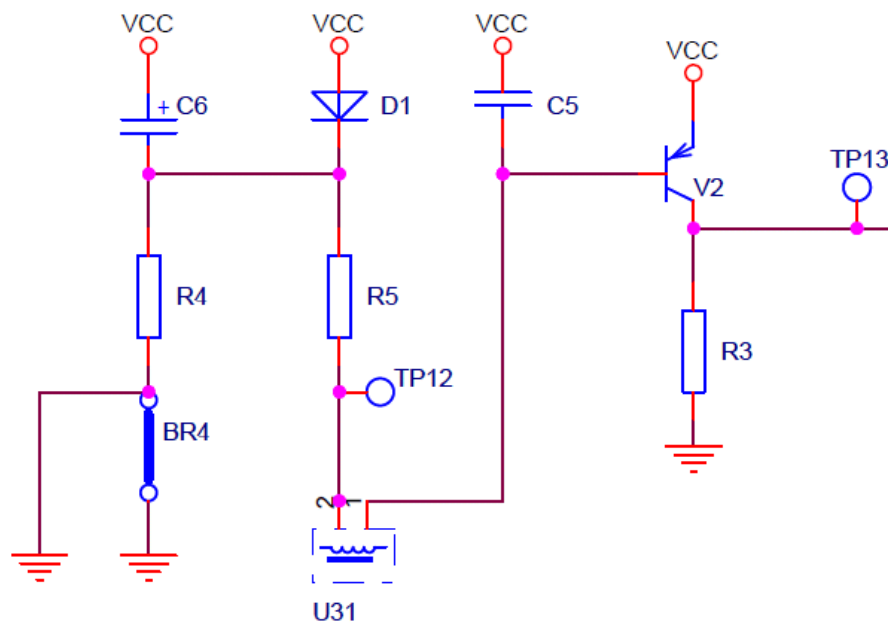


Abbildung 3.5: Schema der Drehzahlmessung bei der GST 135 BCE. Das Signal liegt zwischen TP13 und GND an. Quelle [HU09]

etwas Raum. Einen etwas grösseren Hohlraum gibt es im oberen Teil des Gehäuses oberhalb des Motors. Der gesamte vordere Teil kann nicht genutzt werden, da sich dort das mit Fett gefüllte Getriebe befindet. Parameter wie die Getriebeeinstellung oder die Drehrichtung können nicht so einfach ermittelt werden, da hierzu spezielle mechanische Schalter notwendig wären. Ebenso ist eine Drehzahlmessung nicht trivial, da der Motor keinen Drehzahlgeber hat, man muss also zuerst ein Messsystem konstruieren.

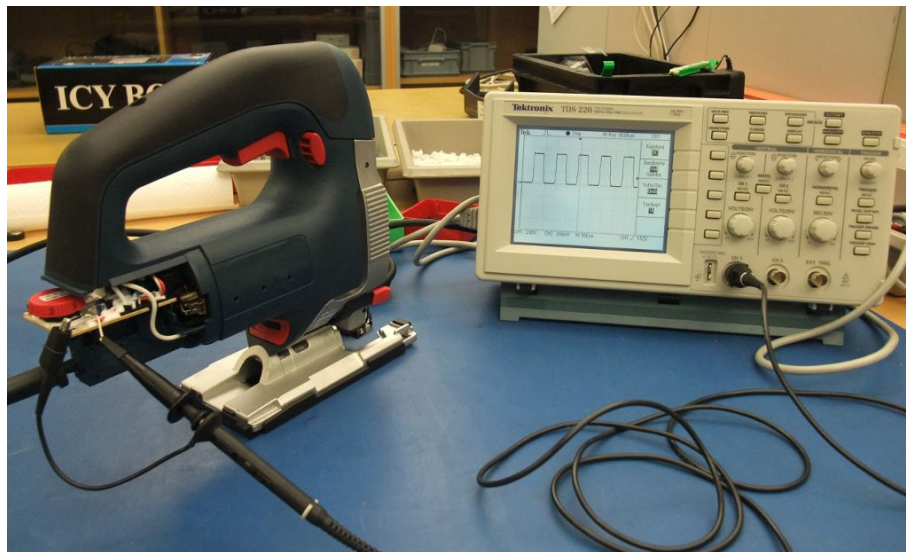


Abbildung 3.6: Messung des Drehzahlgebers der GST 135 BCE



Abbildung 3.7: GSB 21-2 RE

## 3.4 Komponenten

### 3.4.1 Sensoren

#### Temperatur

Der verwendete Temperatursensor ist ein Microchip TC77. Er ist ein kleiner IC im SOT23-Package, welcher seine Temperatur in einem Definitionsbereich von -55 bis +150°C messen kann. Dabei wird direkt die Temperatur des IC erfasst, da es keine Anschlüsse für externe Temperaturfühler gibt.

Die Temperatur kann digital via SPI in Form eines 13-Bit Signed Integer ausgelesen werden.

#### SPI-Konfiguration:

- Chip Select: Active-Low
- Clock Polarity: positiv (steigende Flanke)
- Clock Phase: Setup-Sample
- Data Order: MSB-first

#### Auswertung:

Der zurückgelieferte Registerwert ist ein Signed-Integer-Wert. Um daraus die Temperatur in Grad Celsius zu erhalten, muss dieser Wert noch mit 0.0625 multipliziert werden:

$$\text{Temperatur} = \text{Input} \cdot 0.0625^{\circ}\text{C}$$

Eine Alternative wäre der Maxim MAX6627 mit externer Temperaturfühler-Diode. Der Vorteil dieser Variante ist, dass die Temperatur dort gemessen werden kann, wo es am meisten Sinn macht, der IC aber an einem anderen Ort verbaut werden kann. Weil dieser IC schwer lieferbar ist, wird er für die Erprobung des Konzepts aber nicht verwendet.

Datenblatt: [\[TC77\]](#)

#### Strom, Spannung und Leistung

Bei der Leistungsmessung kommt ein Analog Devices ADE7753 zum Einsatz. Dieser hochintegrierte Sensor-IC ist in der Lage, alle wichtigen elektrischen Parameter zu ermitteln. Er besitzt zwei analoge Kanäle, wobei einer für den Strom und einer für die Spannung verwendet wird. Darüber hinaus kann er anhand der Messwerte Blind-, Schein- und Wirkleistung berechnen.

Per SPI kann der IC konfiguriert werden und die Register mit den Messdaten ausgelesen werden.

#### SPI-Konfiguration:

- Chip Select: Active-Low
- Clock Polarity: positiv (steigende Flanke)
- Clock Phase: Setup-Sample
- Data Order: MSB-first

#### Auswertung:

Der RMS-Wert der Stromstärke (resp. die Spannung über dem Messwiderstand) wird in einem 24-Bit-Register gespeichert und ist vom Typ unsigned Integer. Bei einem vollen Ausschlag von 0.5 V am Eingang entspricht

der Wert 2642412. Die effektiv gemessene Spannung berechnet sich also folgendermassen:

$$U_I = \frac{Input}{2642412} \cdot 0.5V$$

Der RMS-Wert der Spannung wird ebenfalls in einem 24-Bit-Register vom Typ unsigned Integer gespeichert. Bei einem vollen Ausschlag von 0.5 V am Eingang entspricht der Wert 1561400. Die effektiv gemessene Spannung berechnet sich also folgendermassen:

$$U_V = \frac{Input}{1561400} \cdot 0.5V$$

Datenblatt: [\[ADE7\]](#)

Die Umrechnung in die reale Stromstärke und Spannung muss extern in Software erfolgen und ist abhängig von der vorgeschalteten Analogschaltung, welche die originalen Messgrössen in für den ADE7753 messbare Grössen umwandeln.

Als Stromwandler-Baustein kommt ein CST-1015 von Triad zum Einsatz. Dies ist ein Transformator zur Messung von Wechselstrom bis 15 A. Das Wicklungsverhältnis ist 1000:1, also fliesst bei einem Durchgangsstrom von 15 A ein Messstrom von 15 mA. Bei einem Messwiderstand von 33  $\Omega$  ergibt das eine Messspannung von knapp 0.5 V, wodurch der gesamte Messbereich des ADE7753 genutzt werden kann. Bei einem anderen Messbereich kann der Widerstand angepasst werden, so dass wiederum der gesamte Messbereich des ADE7753 verwendet und damit eine optimale Auflösung erreicht wird.

Als Spannungstransformer wird ein einfacher Printtrafo verwendet, der bei primär 230 V sekundär 6 V ausgibt. Zu beachten ist, dass 6 V die effektive Spannung ist, die Peak-Spannung ist aber ca. 8.5 V. Beim Spannungsteiler aus Widerständen ergibt das ungefähr ein Verhältnis von 1:18, so dass bei 230 V Eingangsspannung schliesslich ca. 0.5 V am ADE7753 ankommen.

Die Verwendung eines Speisungstrafo ist nicht ideal, weshalb er, sobald ein besserer Trafo gefunden wird (z.B. ein 50Hz-tauglicher SMD-Trenntrafo), ersetzt wird.

### Lage

Zur Ermittlung der räumlichen Lage wird ein kapazitiver Beschleunigungssensor verwendet. Der MMA7455L von Freescale ist ein integrierter dreiachsiger Sensor, welcher einen Messbereich von 2-8g besitzt. Er besitzt sowohl 8- als auch 10-Bit-Register für die einzelnen Achsen. Angesteuert wird er entweder per SPI oder per I<sup>2</sup>C.

### SPI-Konfiguration:

- Chip Select: Active-Low
- Clock Polarity: negativ (fallende Flanke)
- Clock Phase: Setup-Sample
- Data Order: MSB-first

### Auswertung:

Die Achsenbeschleunigung wird je in einem 8- und einem 10-Bit-Register vom Typ signed Integer gespeichert. Bei einem maximalen Wert entspricht dies je nach Konfiguration 2g, 4g oder 8g. Für die Lagemessung genügen 2g, die reale Beschleunigung wird also im Falle von 8-Bit folgendermassen berechnet:

$$a = Input \cdot 0.015625g$$

Die sehr kompakte Bauweise im LGA- (Land Grid Array) Package wird zum Nachteil. Es ist selbst unter Verwendung von Lötpaste und eines Heissluftlötkolbens kaum möglich, den IC von Hand aufzulöten. Beide Testexemplare werden beim Versuch, sie aufzulöten, unbrauchbar.

Datenblatt: [MMA7]

### Drehzahl

Zur Ermittlung der Drehzahl kann nicht einfach irgendein IC verwendet werden.

Die GST 135 BCE besitzt als einziges Werkzeug einen eingebauten Impulsgeber für die Drehzahl. Dieser wird angezapft und die Impulse direkt mit dem Mikrocontroller gezählt und ausgewertet, indem ein externer Interrupt-Eingang und einer der Counter verwendet wird.

Die anderen Werkzeuge besitzen keinen eingebauten Drehzahlmesser, man muss also selbst ein Konzept zur Drehzahlmessung entwickeln. Denkbar ist einerseits eine ähnliche Methode wie bei der GST 135 BCE, welche einen Hallensensor verwendet. Der rotierende Anker des Wechselstrommotors erzeugt ein wechselndes Magnetfeld, das im Hallensensor einen Stromimpuls auslöst. Bei den anderen Werkzeugen muss allerdings erst der richtige Ort für die Montage des Hallensensors gefunden und evtl. die Motorachse verlängert werden. Danach ist zu messen, wie viele Impulse pro Umdrehung generiert werden.

Eine Alternative ist die optische Messung. Hierzu könnte z.B. die Motorachse auf einer Seite markiert und eine Reflexlichtschranke montiert werden. Dadurch wird bei jeder Umdrehung ein Impuls erzeugt, sobald die Markierung die Lichtschranke passiert. Diese Variante erfordert ebenso eine Modifikation der Motorachse und ist insofern anfällig, dass sich Schmutz auf der Lichtschranke ablagern könnte. Besser ist ein Encoder, der direkt an der Motorachse befestigt wird. Allerdings ist fraglich, ob ein Encoder gefunden werden kann, welcher in der Maschine platz hat.

### 3.4.2 Mikrocontroller

Zur Ansteuerung der Sensoren und Kommunikation mit dem Datenlogger kommt ein Atmel ATmega16-Mikrocontroller zum Einsatz. Dieser Mikrocontroller kommt als SMD-Variante in einem 44pin-TQFP-Package daher und misst damit nur etwa 1 cm<sup>3</sup>. Diese AVR-Variante eignet sich sehr gut zum Experimentieren, da sie zu einem relativ günstigen Preis einen grossen Funktionsumfang bietet.

#### Technische Daten:

- max. Taktfrequenz 16 MHz
- 16 KB Flash
- 1 KB RAM
- 512 Bytes EEPROM
- zwei externe Interrupts
- ein 8-Bit- und ein 16-Bit-Timer/Counter
- ein UART
- SPI Interface
- JTAG Interface

Getaktet wird der AVR mit einem 7.3728 MHz Quarz. Der Grund, dass keine Frequenz wie 4 oder 8 MHz verwendet wird, ist, dass der UART abhängig vom CPU-Takt ist. Bei einer geraden Frequenz würde die Bitrate zu stark von den gängigen Bitraten bei RS232 abweichen, was zu Übertragungsfehlern führen würde.

Details können dem Datenblatt [\[AT16\]](#) entnommen werden.

## 3.5 Das STK600-Experimentierboard

Das STK600 ist Atmels aktuelles Experimentierboard für die AVR-Entwicklung. Die Grundidee dahinter ist, dass es einerseits als Schnittstelle zum PC dient, um Programmcode auf den AVR aufspielen zu können, aber auch als I/O-Plattform, an die weitere Schaltungen angeschlossen werden können. Zu diesem Zweck sind sämtliche I/O-Pins der AVR auf Pin-Header herausgeführt.

Neu gegenüber älteren Experimentierboards ist, dass es keinen fixen AVR-Sockel gibt, sondern lediglich Kontaktflächen, die mit den Pin-Headern verbunden sind. Darauf muss eine Adapterplatine mit dem entsprechenden Sockel gesteckt werden. Das erlaubt die Verwendung eines AVR jeglicher Gehäuseform, von denen mittlerweile Dutzende existieren. Da es aber AVR gibt, die zwar dieselbe Gehäuseform, aber nicht immer dieselbe Pin-Belegung besitzen, wird zwischen Board und Sockel-Adapter noch eine Routing-Platine eingefügt, wovon eine für jede Pin-Belegung existiert. Diese Platine stellt sicher, dass die Pin-Header mit den AVR-Pins korrekt übereinstimmen.

Neben eines USB-Anschlusses, welcher sowohl als Speisung als auch als Programmierschnittstelle dient, besitzt das STK600 einen RS232-Anschluss und einen CAN-Bus. Zu Test- und Debugging-Zwecken gibt es noch jeweils acht LEDs und Taster. Damit diese Peripherie genutzt werden kann, müssen die Anschlüsse via Kabel mit den entsprechenden Pin-Headern verbunden werden.

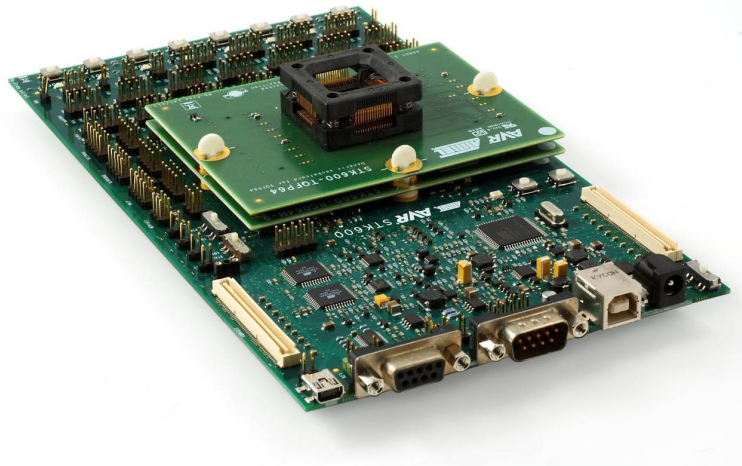


Abbildung 3.8: Atmel STK600

Für das P7-Projekt ist das STK600 mit einem TQFP44-Sockel ausgestattet, damit ein ATmega16 in SMD-Form verwendet werden kann.

## 3.6 Das Sensor-Interface

Das Sensor-Interface ist eine Platine, welche einen Mikrocontroller und alle benötigten Sensoren aufnehmen kann. Für die Speisung wird 5 V Gleichspannung verwendet, die Kommunikation mit dem Datenlogger (resp. im Rahmen des P7-Projekts mit einem PC) erfolgt via RS232 (5V-Pegel). Um Platz zu sparen, werden fast ausschliesslich SMD-Komponenten verwendet.

Das Layout ist ca. 10 x 8 cm gross und doppelseitig.

Für den Beschleunigungssensor wird eine Adapterplatine von LGA14 auf DIL14 (Abbildung 3.12) verwendet, da diese einfacher auszuwechseln ist, falls sich herausstellt, dass der Sensor nicht funktioniert.

Alle Platinen werden mit dem Eagle Layout Editor 5.6 designt. Dieses Produkt des Herstellers CADSoft ist als Studentenversion frei verfügbar und im Gegensatz zu mächtigen professionellen Tools relativ leicht zu bedienen. Trotzdem bietet es einen ausreichenden Funktionsumfang.

### 3.6.1 Schema des Sensor-Interface

Das Schema besteht grundsätzlich aus Bauteilen und Verbindungen. Für jedes elektronische Bauteil gibt es ein Objekt in der Eagle-Bibliothek. Ist ein Bauteil nicht vorhanden, so kann es mit einem Editor selbst erstellt werden. Im Fall des Sensor-Interface ist das auch nötig, da zu den verwendeten Sensoren keine passenden Bauteile in der Standardbibliothek existieren. Jedes Objekt besitzt Anschlüsse, welche die realen Anschlüsse des Bauteils repräsentieren. Diese können nun miteinander verbunden werden, wodurch ein Schema entsteht. Das Schema ist lediglich ein Schaltplan und stimmt mit den Dimensionen des späteren Layouts nicht überein.

Siehe Abbildung 3.9.

Erst auf dem Layout werden basierend auf dem Schema die Bauteile plaziert und die Leiterbahnen gezogen. Dies ist ein schwieriger, zeitraubender Prozess, da jede Versetzung eines Bauteils unter Umständen grosse Änderungen an den Leiterbahnen erfordert. Es ist also sinnvoll, zuerst alle Bauteile so sinnvoll wie möglich zu plazieren, bevor man die Leiterbahnen zeichnet.

Siehe Abbildung 3.10.

### 3.6.2 Herstellungsprozess

Zur Herstellung von Platinen werden unterschiedliche Fertigungsprozesse verwendet. Lässt man eine Platine extern anfertigen, so werden in der Regel die Zwischenräume zwischen den Leiterbahnen ausgefräst und Durchkontaktierungen eingepresst. Zusätzlich werden alle Löt pads verzinnt und eine Lötstopmmaske aufgetragen, so dass der Löt zinn nur auf die benötigten Stellen aufgetragen werden kann.

Bei der Prototypenherstellung wird oft eine Ätzmethode verwendet. Das bedeutet, dass auf der Platine alles weggeätzt wird, was kein Löt pad oder eine Leiterbahn ist.

Diese Methode kommt auch bei der Herstellung des Sensor-Interface zum Einsatz. Beide Seiten des in Eagle designte Layouts werden auf Folie ausgedruckt und mit der bedruckten Seite auf eine fotobeschichtete Platine gelegt. Nun kommt das Ganze in einen Belichtungsapparat, welcher alle Teile, welche später kein Kupfer enthalten sollen, belichtet. Anschliessend wird die Platine mit Fotoentwickler behandelt und fixiert. Sie ist nun bereit, geätzt zu werden. Danach kommt die Platine in einen Sprühzylinder, ein Behälter, in dem die Platine kontinuierlich mit einer ätzenden Chemikalie besprüht wird, bis nur noch die Leiterbahnen übrig sind. Zum Schluss wird die Platine mit einer Lauge neutralisiert.

Bevor gelötet werden kann, müssen noch Löcher für nicht-SMD-Bauteile und Durchkontaktierungen gebohrt

werden. Dazu wird eine spezielle Bohrmaschine mit besonders feinen Bohrern und einer Platinenhalterung verwendet. Da die Bohrlöcher besonders präzise sein müssen, besitzt die Bohrmaschine eine Vergrößerungsoptik mit einem Fadenkreuz.

Leider können mangels Ausrüstung keine direkten Durchkontaktierungen eingepresst werden. Notgedrungen müssen diese mit Drahtstücken, welche auf beiden Platinenseiten angelötet werden, realisiert werden.

### 3.6.3 Testaufbau

Zum Testen wird das Sensor-Interface mit dem STK600 verbunden. 5V-Speisung und SPI-Bus kommen dabei direkt vom STK600, da auf dem Sensor-Interface kein AVR bestückt ist, und auch RS232 nicht angeschlossen ist.

Die Tests des ADE7753 sind etwas speziell, so dass dazu noch ein selbst konstruiertes Netzkabel nötig ist. Das Netzkabel hat am Ende eine herkömmliche Typ13-Buchse, wobei die Phasenleitung durch den CST-1015-Stromwandler geführt ist. Nulleiter und Erdung dürfen nicht durch den Stromwandler hindurchgeführt werden, da sich sonst das Magnetfeld gegenseitig aufhebt. Parallel dazu werden Phase und Nulleiter über Schraubklemmen mit dem Printtrafo verbunden. Auf diese Weise liegen immer 230 V an, was mit dem ADE7753 gemessen werden kann. Wird ein beliebiges Gerät in die Typ13-Buchse gesteckt, so lässt sich dessen Stromverbrauch mit dem ADE7753 messen.

Wie es sich gehört ist die Erdleitung etwas länger als die anderen, so dass diese im worst case zuerst abreißen.

Siehe Abbildung [3.13](#).

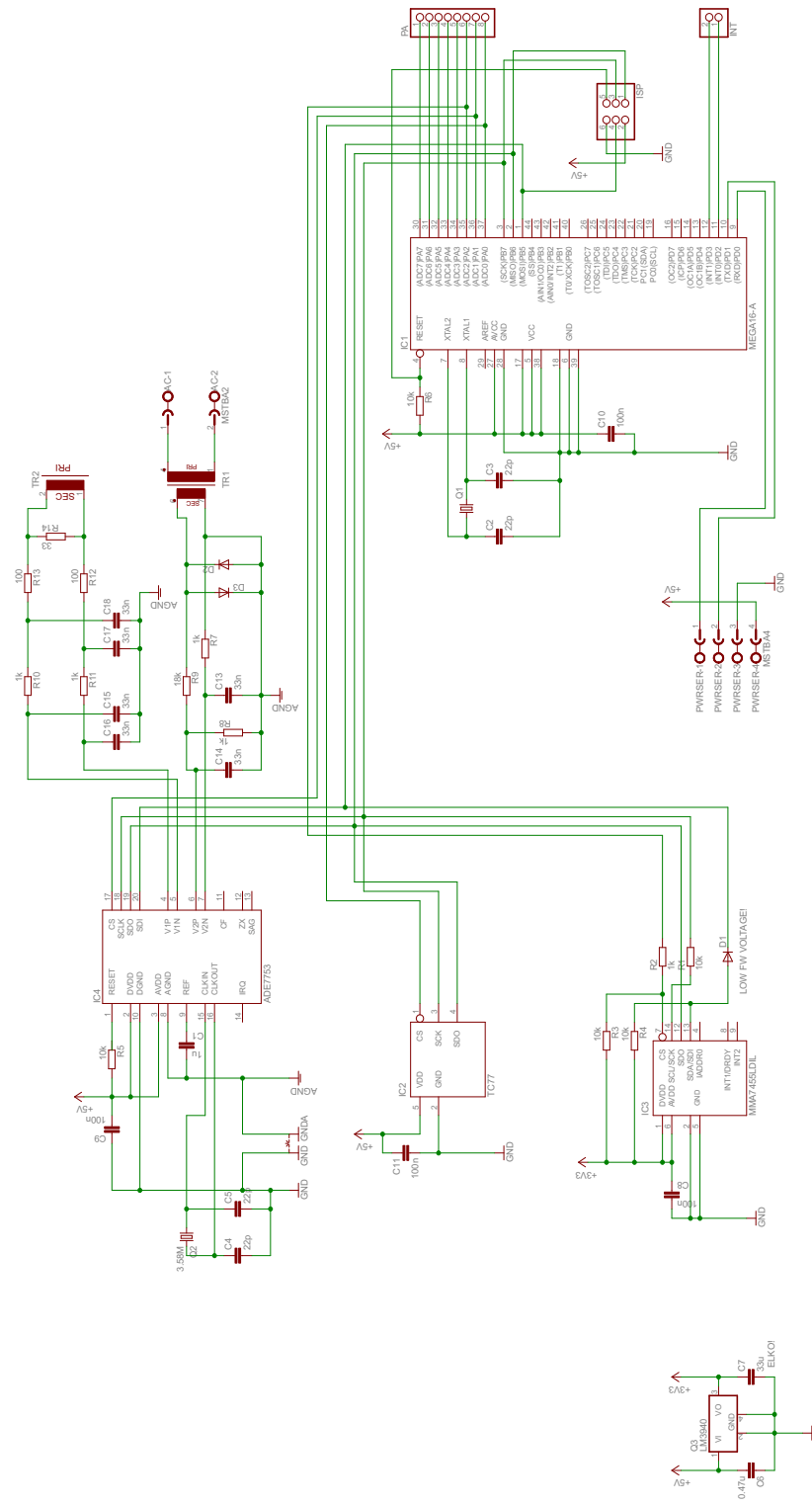


Abbildung 3.9: Schema des Sensor-Interface (Eagle 5.6)

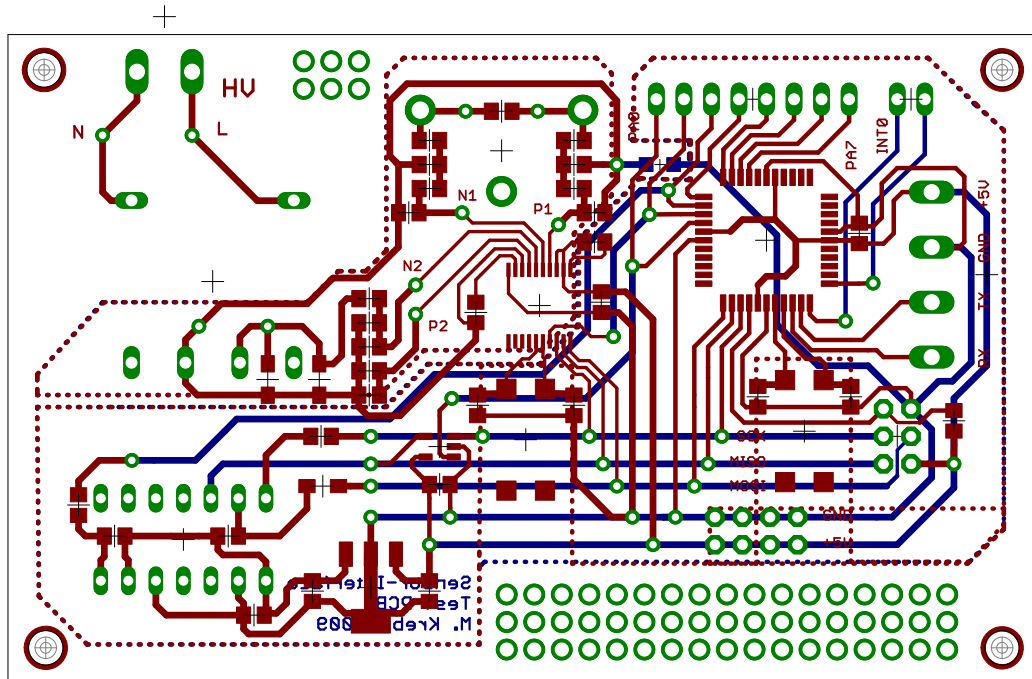
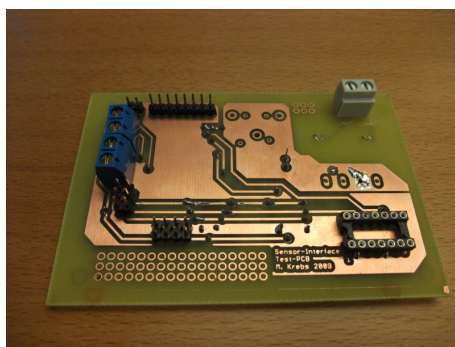
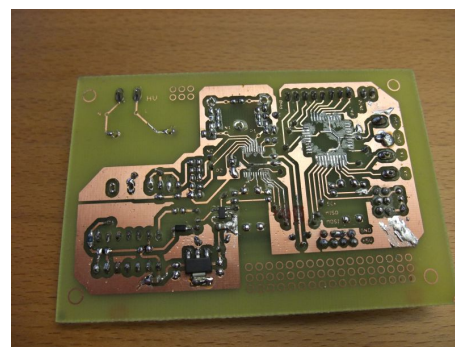


Abbildung 3.10: Layout des Sensor-Interface (Eagle 5.6)

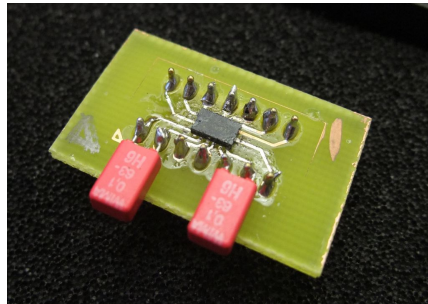


(a) Oberseite

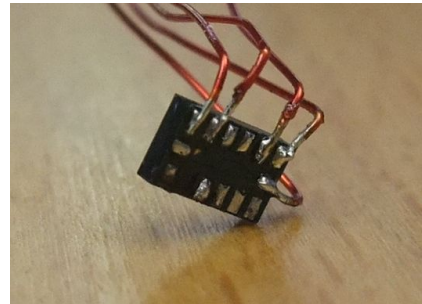


(b) Unterseite

Abbildung 3.11: Ober- und Unterseite des Sensorinterface



(a) LGA-Adapter mit MMA7455L



(b) MMA7455L mit Bondingdrähten

Abbildung 3.12: LGA14-DIL14-Adapter

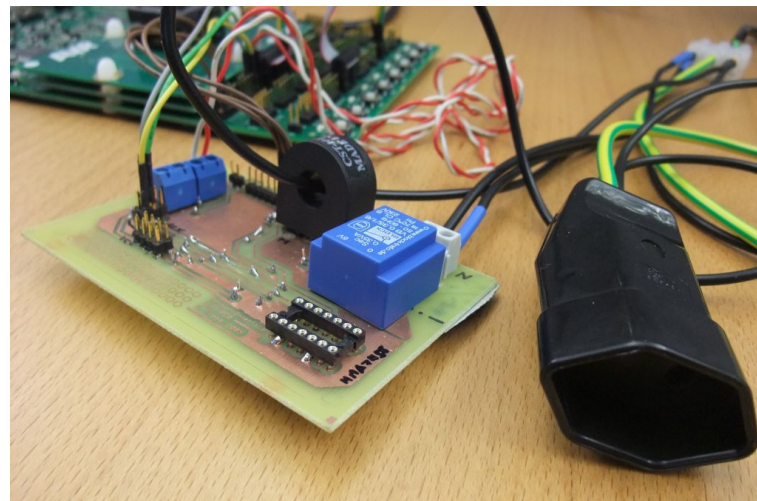


Abbildung 3.13: Testaufbau mit STK600, Sensor-Interface und 230V-Adapter

## **Kapitel 4**

# **Software**

## 4.1 Mikrocontroller-Entwicklungsumgebung

### Compiler

Es existieren zahlreiche kommerzielle und freie Compiler für unterschiedliche Programmiersprachen, die für AVR verwendet werden können.

Weit verbreitet ist der AVR-GCC, welcher auf dem GNU C Compiler basiert, aber AVR-spezifische Erweiterungen beinhaltet. Er wird in diesem Projekt verwendet, weil die Programmiersprache C ein guter Kompromiss zwischen Komfort und Effizienz ist. Gerade auf Systemen mit knappen Ressourcen, zu denen auch Mikrocontroller gehören, ist eine effiziente Programmierung unerlässlich.

Auch C++ wird von AVR-GCC unterstützt, allerdings nur mit reduziertem Funktionsumfang (kein new-Operator, keine virtuellen Methoden).

AVR-GCC ist auf allen Linux-Systemen erhältlich und lässt sich auch unter Windows in Form des WinAVR-Pakets komfortabel installieren.

### Programmer

Ist der Programmcode für den AVR kompiliert, so muss er auch noch auf den Controller übertragen werden. Dazu wird eine Software benötigt, welche auf die Programmier-Hardware (z.B. STK600) zugreifen kann.

Bei Opensource-Entwicklern ist avrdude sehr beliebt, da es alle gängigen Programmiergeräte und AVR unterstützt. Neben der Übertragung des kompilierten Programmcodes kann avrdude auch diverse Konfigurationsparameter des AVR auslesen und neu beschreiben. Diese Parameter heißen Fusebits und erlauben die u.a. die Konfiguration des verwendeten Taktgebers oder die Aktivierung und Deaktivierung bestimmter Schnittstellen des AVR. Neben den Fusebits gibt es noch Lockbits, die beispielsweise genutzt werden können, um den AVR gegen versehentliches Überschreiben zu schützen. Während der Entwicklung ist es jedoch ratsam, die Lockbits in Ruhe zu lassen, da im Falle eines Fehlers der AVR für immer unbrauchbar werden kann.

### IDE

Zur Softwareentwicklung für AVR-Mikrocontroller bietet Atmel die Software AVRStudio an, die kostenlos heruntergeladen werden kann. Sie enthält Dokumentation zu den verschiedenen Mikrocontrollern und Experimentierboards, als Programmiersprache wird ein AVR-spezifischer Assembler verwendet. Ein Nachteil ist, dass Hochsprachen wie C kaum unterstützt werden, und die Software nur unter Windows verfügbar ist.

Es existieren jedoch auch Opensource-Tools, welche ebenfalls eine recht komfortable Mikrocontroller-Entwicklung erlauben.

Für Eclipse existiert ein AVR-Plugin [[ECAV](#)], welches im Zusammenspiel mit dem C/C++-Plugin genutzt werden kann. Es bindet externe Compiler und Programmier-Tools wie AVR-GCC und avrdude ein und bietet damit dem Entwickler ein praktisches grafisches Frontend für die wesentlichen Funktionen. Der Funktionsumfang umfasst die Konfiguration des Programmiergeräts unter Verwendung von avrdude, die Konfiguration der Fusebits und Lockbits mittels eines grafischen Editors und die Übertragung des kompilierten Codes auf den AVR. Das Kompilieren übernimmt dann das C/C++-Plugin von Eclipse, das von vorneherein auf die Verwendung externer Compiler ausgelegt ist.

## 4.2 Mikrocontroller-Software

Die Software, welche auf dem AVR läuft, ist in C geschrieben. Sie besteht im wesentlichen aus zwei Teilen: einem Hauptprogramm, welches beim Einschalten des AVR geladen und kontinuierlich ausgeführt wird, sowie einer selbst erstellten Funktionsbibliothek, die es erlaubt, Schnittstellen des AVR einfacher anzusteuern und sensorspezifische Funktionen bereitstellt.

### 4.2.1 Funktionsbibliothek

Die Funktionsbibliothek kapselt Schnittstellen des AVR in Funktionen und benutzt diese, um wiederum Funktionen zur Ansteuerung der einzelnen Sensoren bereitzustellen.

Der Effizienz wegen werden gewisse Funktionen gar nicht als echte C-Funktionen implementiert, sondern lediglich als Makros definiert. Der Vorteil ist, dass die Lesbarkeit des Codes verbessert wird und gleichzeitig weniger Indirektionen durch Funktionsaufrufe entstehen.

Die Implementierung von Ringpuffer, UART und SPI-Bus basiert auf der Procyon AVRlib [\[PROC\]](#). Jedoch wurden die meisten Funktionen neu implementiert, da der Funktionsumfang der Bibliothek nicht ausreichte.

#### Definitionen

```
#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
```

Da C keinen echten Boolean-Typ kennt, werden für *true* und *false* einfach passende Konstanten definiert. In C entspricht *false* 0 und *true* einem Wert ungleich 0.

```
typedef uint8_t boolean;
typedef uint8_t byte;
typedef uint8_t uint8;
typedef uint16_t uint16;
typedef uint32_t uint32;
typedef int8_t int8;
typedef int16_t int16;
typedef int32_t int32;
```

Zur besseren Lesbarkeit werden die wichtigsten Datentypen neu definiert.

```
#define bit(b) (1 << b)
```

Bei der AVR-Programmierung muss man oft auf einzelne Bits zugreifen, welche als Makro definiert sind (Shift nach links um die Anzahl Bits, die mit der Position des Bits innerhalb eines Bytes übereinstimmt). Dieses Makro vereinfacht die Darstellung eines Bits. Die Register-Bits in der AVR-Bibliothek sind gemäss ihrer Position als numerische Konstanten definiert.

```
#define setBits(reg, bits) reg |= (bits)
#define clearBits(reg, bits) reg &= ~(bits)
```

Das Setzen eines einzelnen Bits in einem AVR-Register erfolgt durch ein bitweises OR, während das löschen eines Bits mit einem bitweisen AND des invertierten Werts erfolgt. Der Lesbarkeit halber sind diese als Makros definiert.

### Konkurrierende Zugriffe

Damit Funktionen, welche exklusiven Zugriff benötigen, nicht durch Interrupt-Handler unterbrochen werden können, werden passende Makros definiert, um kritische Bereiche abzugrenzen.

```
#define ENTER_CRITICAL_SECTION() byte sreg_tmp = SREG; cli()
#define EXIT_CRITICAL_SECTION() SREG = sreg_tmp
```

Dabei wird das Interrupt-Register *SREG* zwischengespeichert und anschliessend die Interrupts global deaktiviert. Am Ende des kritischen Bereichs wird dann das Interrupt-Register wiederhergestellt und damit die Interrupts wieder aktiviert.

### Ringpuffer

Bei bestimmten Schnittstellen, z.B. UART, macht eine Pufferung der Daten Sinn, weil der Datentransfer asynchron ist. Dazu eignet sich ein Ringpuffer, da dieser ähnlich einer Warteschlange (FIFO) verwendet wird. Der Vorteil gegenüber einem einfachen Array ist, dass nicht erst der ganze Puffer geleert werden muss, damit Daten wieder eingefügt werden können.

```
typedef struct ringbuffer_s {
    byte* data;
    uint8 size;
    uint8 start;
    uint8 dataLen;
} ringbuffer;
```

Definition des Ringpuffer-Objekts mit Daten-Array (*data*), Maximalgrösse (*size*), Position des ersten Datenbytes (*start*) und Anzahl Datenbytes im Puffer (*dataLen*).

```
void initRingBuffer(ringbuffer* buf, uint8 size)
```

Initialisiert ein Ringpuffer-Objekt mit einer bestimmten Grösse.

**void freeRingBuffer(ringbuffer\* buf)**

Gibt den Speicher, den ein Ringpuffer-Objekt benötigt, wieder frei. Danach darf auf das Objekt nicht mehr zugegriffen werden.

**boolean addByteToRingBuffer(ringbuffer\* buf, byte b)**

Fügt ein Byte in einen Ringpuffer ein. Ist der Puffer nicht voll, so wird das Byte eingefügt und TRUE zurückgegeben. Ist der Puffer voll, wird FALSE zurückgegeben. Diese Funktion verwendet exklusiven Zugriff.

**boolean getByteFromRingBuffer(ringbuffer\* buf, byte\* b)**

Entfernt das erste Byte im Ringpuffer und speichert es an der Adresse der Variable b. Danach wird TRUE zurückgegeben. Befinden sich keine Daten im Puffer, so wird einfach FALSE zurückgegeben und b wird ignoriert. Diese Funktion verwendet exklusiven Zugriff.

**Buffered UART**

Diese UART-Implementierung verwendet Interrupts für die asynchrone Datenübertragung, und die Daten werden vor dem Senden in einem Ringpuffer gespeichert. Der Empfang von Daten ist nicht implementiert, da er zur Zeit nicht benötigt wird.

**void uartSetup(uint32 baudrate, uint8 databits, uint8 stopbits, uint8 parity, uint8 txBufSize, uint8 rxBufSize)**

Konfiguriert und aktiviert den UART des AVR.

baudrate:	eine der gängigen Baudraten, z.B. 9600
databits:	eine Konstante DATABITS_x, wobei x eine Zahl zwischen 5 und 9 sein muss
stopbits:	STOPBITS_1 oder STOPBITS_2
parity:	PARITY_NONE, PARITY_EVEN oder PARITY_ODD
txBufSize:	Grösse des Sendepuffers
rxBufSize:	Grösse des Empfangspuffers (irrelevant)

**void uartShutdown()**

Deaktiviert den UART.

**boolean uartEnqueueByte(byte b)**

Fügt dem Sendepuffer ein Byte hinzu, sofern er nicht voll ist.

**void uartTransmitBuffer()**

Überträgt den gesamten Inhalt des Puffers asynchron. Da Interrupts verwendet werden, blockiert die Funktion nicht.

**SPI**

Die Implementierung des SPI-Bus erlaubt das Senden und Empfangen von Daten sowie eine benutzerdefinierte Konfiguration des SPI-Interface. Da die Chip-Selects nicht Teil eines Standards sind, werden diese extern behandelt (durch Setzen des Pins auf low oder high).

**void spiSetup(byte divider, byte phase, byte polarity, byte hosttype, byte order)**

Konfiguriert und aktiviert das SPI-Interface des AVR.

divider: der Divisor, durch den der CPU-Takt dividiert wird, um den SPI-Takt zu erhalten. Als Konstante CLOCKDIV\_x verfügbar, wobei x 4, 16, 64 oder 128 ist

phase: CLOCKPHASE\_SETUPSAMPLE oder CLOCKPHASE\_SAMPLESETUP

polarity: RISING\_EDGE oder FALLING\_EDGE

hosttype: SPI-Typ des AVR (MASTER oder SLAVE)

order: Bitreihenfolge (MSB\_FIRST oder LSB\_FIRST)

**void spiShutdown()**

Deaktiviert das SPI-Interface.

**void spiWrite(byte b)**

Sendet ein Byte an den gerade aktiven SPI-Slave. Der Transfer erfolgt asynchron, weshalb die Funktion nicht blockiert.

**byte spiWriteAndRead(byte b)**

Sendet ein Byte an den gerade aktiven SPI-Slave und empfängt gleichzeitig ein Byte, welches anschliessend zurückgegeben wird. Die Funktion blockiert, bis das zu empfangende Byte vollständig transferiert wurde.

**void spiWait()**

Blockiert, bis ein Transfer abgeschlossen ist. Nur bei spiWrite() nötig, wenn anschliessend das Chip-Select deaktiviert werden soll, da ansonsten Timing-Probleme auftreten.

**TC77**

Diese Funktionen ermöglichen das Auslesen der Temperatur eines TC77-Sensors via SPI. Das Chip-Select ist Pin 0 auf Port A.

**void tc77Init(byte spiClockDiv)**

Initialisiert das SPI-Interface mit Parametern für den TC77. Ein benutzerdefinierter Takt-Divider kann angegeben werden.

**int16 tc77ReadTemperature(int16 offset)**

Liest die Temperatur als 16-Bit Integer aus. Ist der Sensor ungenau, kann ein Offset angegeben werden. Eine Integer-Einheit entspricht 0.0625°C.

**ADE7753**

Diese Funktionen ermöglichen die Konfiguration des ADE7753-Sensors und das Auslesen von Spannung, Strom und Leistung. Das Chip-Select ist Pin 1 auf Port A.

**void ade7753Init(byte spiClockDiv)**

Initialisiert das SPI-Interface mit Parametern für den ADE7753. Ein benutzerdefinierter Takt-Divider kann angegeben werden.

**void ade7753Configure()**

Konfiguriert den ADE7753. Muss direkt nach der ersten Initialisierung aufgerufen werden.

#### **uint32 ade7753ReadCurrent()**

Liest den RMS-Wert des Stroms als 32-Bit unsigned Integer aus. Dies ist lediglich der Registerwert, die Umrechnung in einen realen Stromwert muss extern erfolgen.

#### **uint32 ade7753ReadVoltage()**

Liest den RMS-Wert der Spannung als 32-Bit unsigned Integer aus. Dies ist lediglich der Registerwert, die Umrechnung in einen realen Spannungswert muss extern erfolgen.

#### **int32 ade7753ReadPower()**

Liest den Wert des Leistungsregisters als 32-Bit Integer aus. Dies ist lediglich der Registerwert, die Umrechnung in einen realen Leistungswert muss extern erfolgen.

### **MMA7455L**

Diese Funktionen ermöglichen die Konfiguration des MMA7455L-Beschleunigungssensors und erlauben das Auslesen der Messwerte der drei Achsen im 8-Bit-Format. Das Chip-Select ist per Default auf Pin 2 des Ports A.

#### **void mma7455Init(byte spiClockDiv)**

Initialisiert das SPI-Interface mit Parametern für den MMA7455L. Ein benutzerdefinierter Takt-Divider kann angegeben werden.

#### **void mma7455IConfigure()**

Konfiguriert den MMA7455L so, dass das I<sup>2</sup>C-Interface deaktiviert und die kontinuierliche Beschleunigungsmessung im Bereich von max. 2 g aktiviert wird.

#### **void mma7455IReadXYZ(int8\* x, int8\* y, int8\* z)**

Liest die Register der drei Achsen aus und speichert die Werte jeweils in einer 8-Bit-Variablen. Eine Integer-Einheit entspricht ca. 0.015625 g.

## **4.2.2 Hauptprogramm**

Das Hauptprogramm enthält die main()-Funktion, welche auf dem AVR nach dem Einschalten ausgeführt wird. Das Spezielle an einem AVR-Programm ist, dass es immer mit einer Endlosschleife im Stil von `while(TRUE);` enden muss, da das Programm auf dem AVR nie beendet werden darf. Ansonsten befände sich der Mikrocontroller in einem undefinierten Zustand.

Das Hauptprogramm läuft folgendermassen ab:

Als allererstes müssen alle I/O-Pins, die verwendet werden, konfiguriert werden. Ihre Datenrichtung (Input oder Output) muss entsprechend der elektronischen Schaltung definiert werden, da ansonsten Fehler oder gar Schäden auftreten könnten. Danach werden die Output-Pins mit sinnvollen Werten initialisiert (z.B. alle Chip-Selects inaktiv).

Danach werden Konfigurationsdaten der Sensoren aus dem EEPROM geladen. Das EEPROM ist ideal als persistenter Speicher für variable Daten geeignet, da es unabhängig vom Flash-Speicher für den Programmcode

nutzbar ist. Die Anzahl Schreibzyklen ist auf ca. 100'000 begrenzt, also sollte das EEPROM nicht für sich häufig verändernde Daten benutzt werden.

Als erste Kommunikationsschnittstelle wird der UART konfiguriert. Danach werden alle Sensoren der Reihe nach ein Mal initialisiert und anschliessend konfiguriert. Dieser Schritt ist nach dem Einschalten zwingend, da die Sensoren keinen persistenten Speicher besitzen.

Da jetzt das Sensor-Interface initialisiert ist, beginnt die Endlosschleife. Innerhalb dieser Schleife werden alle gewünschten Messwerte der Reihe nach aus den Sensoren ausgelesen. Danach wird im UART-Puffer ein Datenpaket zusammengesetzt, welches mit einem Startbyte mit dem Hex-Wert *A5* beginnt. Alle Zahlenwerte werden im Little-Endian-Format gespeichert, um die Decodierung am Logger (während des Tests ein PC) zu vereinfachen.

Während der Testphase wird jeweils nur ein Sensor ausgelesen, um sicherzustellen, dass er korrekt funktioniert. Das Datenpaket entspricht dann dem Startbyte plus einer Aneinanderreihung aller Messwerte des Sensors. Wenn alle Sensoren ausreichend getestet sind, kann ein endgültiges Datenformat entwickelt werden.

Ist das Datenpaket im UART-Puffer zusammengestellt, so wird es an den Logger übermittelt.

Am Ende der Schleife wird eine Verzögerung von 0.5 Sekunden gemacht, bevor die Schleife von vorn beginnt. Man könnte auch einen Timer verwenden, Versuche haben aber gezeigt, dass dies zu Problemen aufgrund von verschachtelten Interrupts führt. Innerhalb des Interrupthandlers des Timers werden weitere Interrupts erzeugt, was ein Deadlock erzeugt.

Glücklicherweise fällt das Auslesen der Sensordaten nicht so ins Gewicht, denn es benötigt pro Sensor nur wenige Millisekunden. Bei einem SPI-Takt von knapp 60 kHz (CPU-Takt / 128) können, alle zusätzlichen Verzögerungen eingerechnet, mindestens 40-50 kbit pro Sekunde ausgelesen werden. Pro Sensor müssen aber nur 2 bis 10 Bytes an Daten ausgelesen werden.

Später kann der Timer noch zur Berechnung der Drehzahl verwendet werden, indem die Drehzahlimpulse gezählt und durch die verstrichene Zeit zwischen zwei Zählungen dividiert werden.

Für ein Flussdiagramm siehe Abbildung [4.2.2](#).

### 4.2.3 Testprogramme

Während der Arbeiten an der Funktionsbibliothek und dem Hauptprogramm wurden immer wieder kleine Testprogramme für den AVR geschrieben, um die einzelnen Funktionen separat testen und erforschen zu können. Die Erkenntnisse daraus sind danach in die Entwicklung eingeflossen.

Das erste Testprogramm *uarttest* besteht lediglich aus einem 8-Bit-Lauflicht, dessen aktuelle Position über den UART übermittelt wird. Dies erlaubt es, herauszufinden, wie der UART funktioniert, und worauf zu achten ist, damit die Daten korrekt ankommen.

Das Testprogramm *freqcount* ist ein Testprogramm für einen externen Interrupthandler und einen Timer. Eingehende Impulse am Interrupt-Pin werden gezählt und ihre mittlere Frequenz ausgegeben. Das Programm stellt einen ersten Ansatz zur Frequenzmessung dar, welcher später in verbesserter Form zur Drehzahlmessung verwendet werden kann.

Das Testprogramm *spitest* ist das erste Programm, das den SPI-Bus verwendet. Zusammen mit einer kleinen Adapterplatine, die mit einem TC77-Temperatursensor bestückt ist, kann die Funktionsweise des SPI-Bus, insbesondere das Timing, erforscht werden.

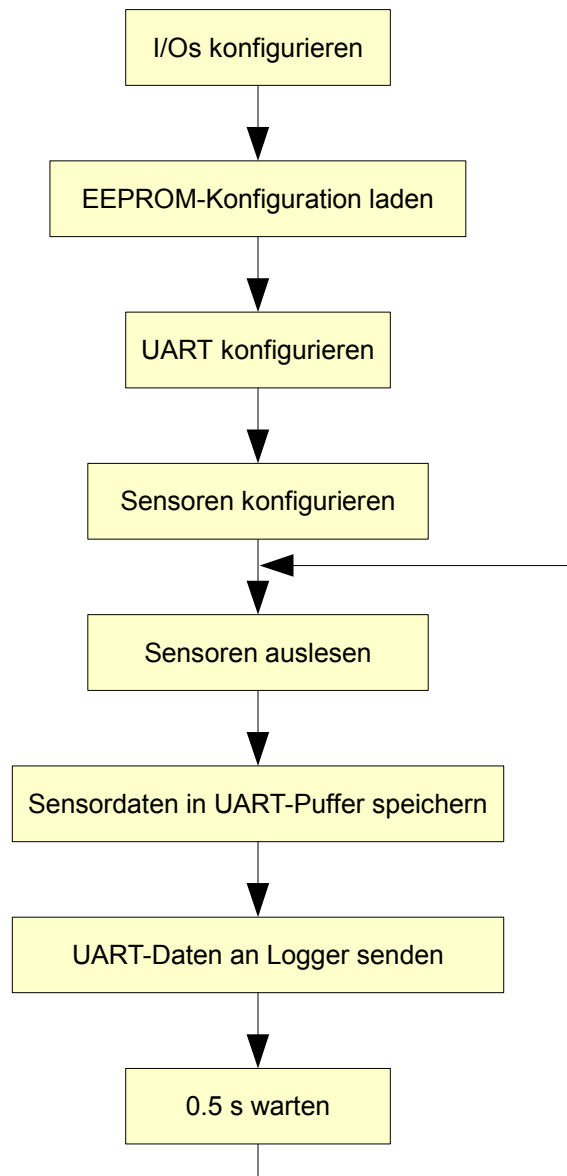


Abbildung 4.1: Flussdiagramm des Hauptprogramms auf dem AVR

## 4.3 Client-Software

Die Client-Software dient ausschliesslich Test- und Debuggingzwecken. Sie liest den Output des Sensor-Interface und stellt die Messdaten in lesbarer Form dar. Während der Hardwareentwicklungsphase stellen solche Client-Programme den ersten Schritt zu einem Datenlogger dar. Dabei geht es darum, die Messwerte in Form von Rohdaten richtig zu interpretieren, zu decodieren und in der gewünschten Form der Messdaten auszugeben.

Für die Entwicklung der clientseitigen Testprogramme wird Visual Studio 2008 mit C# verwendet. Damit lässt sich beispielsweise ein RS232-Anschluss mit Hilfe der SerialPort-Klasse recht einfach und event-basiert ansteuern.

Das C#-Projekt *RS232Test* ist folgendermassen aufgebaut:

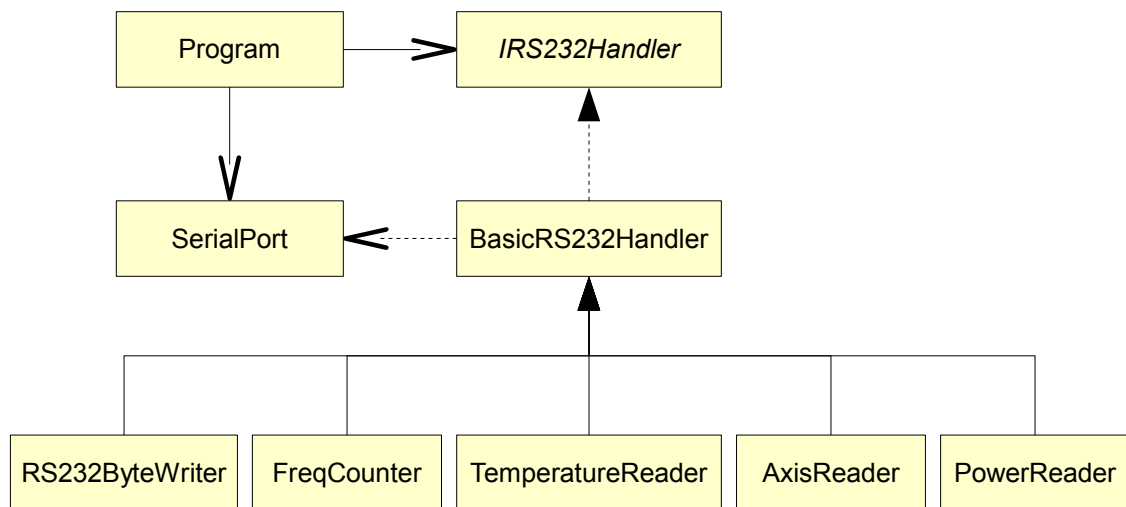


Abbildung 4.2: Klassendiagramm der Client-Software

Die Klasse *Program* enthält die Main-Methode. Dort wird das *SerialPort*-Objekt initialisiert, sowie ein Handler, der das *IRS232Handler*-Interface implementiert. Die Methode wartet dann auf Daten, die via RS232 empfangen werden. Das Programm wird erst beendet, wenn eine Taste gedrückt wird.

Das Interface *IRS232Handler* stellt zwei Methoden, *Start()* und *Stop()* bereit, um einen Handler, der dieses Interface implementiert, starten und stoppen zu können.

Die Klasse *BasicRS232Handler* implementiert *IRS232Handler* und ist die Basisimplementierung für alle spezifischen Handler. Zur Kommunikation wird ein *SerialPort*-Objekt referenziert, welches extern instanziiert werden muss.

Die Methoden *Start()* und *Stop()* öffnen resp. schliessen den seriellen Anschluss für den Datentransfer.

Die beiden virtuellen Methoden *OnDataReceived(...)* und *OnErrorReceived(...)* behandeln die Events *DataReceived* und *ErrorReceived* des *SerialPort*-Objekts. In *OnDataReceived(...)* wird ein Empfangspuffer mit empfangenen Daten gefüllt, in *OnErrorReceived(...)* wird ein Fehlercode gespeichert.

Spezifische Handler überladen diese Methoden und interpretieren zusätzlich noch die empfangenen Daten.

Die Klasse *RS232ByteWriter* ist ein einfacher Handler, welcher einfach die empfangenen Bytes hexadezimal auf die Konsole ausgibt.

Die Klasse *TemperatureReader* ist für Tests des TC77-Sensors gedacht. Der Handler enthält eine kleine State-Machine, welche im Zustand 0 auf ein Startbyte wartet und dann in Zustand 1 übergeht. In Zustand 1 liest der Handler Datenbytes ein, bis zwei Bytes eingelesen wurden. Danach wertet er die Daten aus und geht wieder in Zustand 0 über.

Da die Daten im Little-Endian-Format vorliegen, können sie ganz einfach mit `BitConverter.ToInt16()` konvertiert werden. Um die Temperatur zu berechnen, wird danach mit 0.0625 multipliziert.

Neben der Konsolenausgabe werden die Daten noch in eine ASCII-Datei „temp.csv“ geschrieben.

Die Klasse *FreqCounter* dient der Frequenzmessung. Der Handler liest vier Datenbytes pro Messung ein und konvertiert sie nach `UInt32`. Diese Zahl wird durch die konstante Zeitspanne von  $\frac{65536 \cdot 256}{7372800}$  s dividiert, welche dem Timer-Intervall der Messungen des AVR entspricht. Die gemessene Frequenz wird dann auf die Konsole ausgegeben.

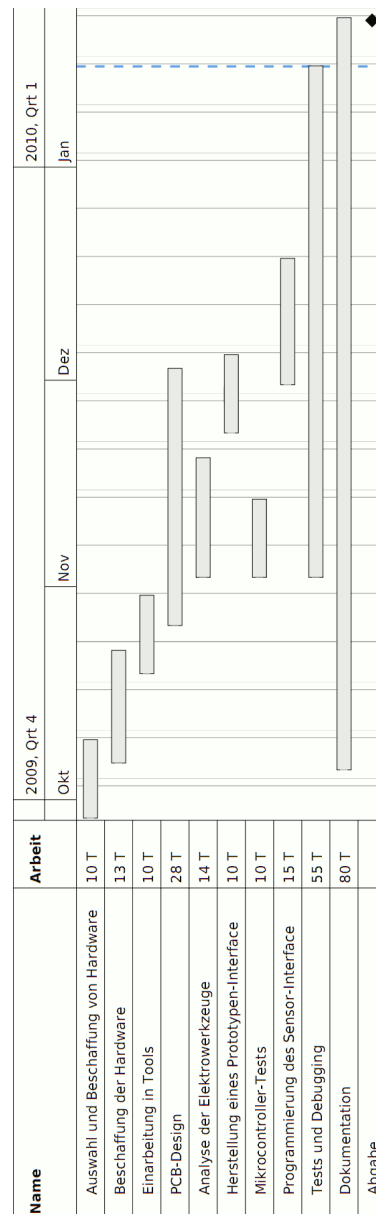
Die Klasse *AxisReader* verarbeitet Daten des Beschleunigungssensors. Der Handler liest drei Datenbytes ein (eins pro Achse) und gibt diese in Form eines 8-Bit Integer auf die Konsole aus. Weil die Tests mit dem Beschleunigungssensor misslungen sind, ist die Klasse nicht wirklich gut getestet worden.

Die Klasse *PowerReader* verarbeitet Daten des ADE7753-Sensors. Der Handler liest 13 Datenbytes ein, wobei das erste den Inhalt des Temperaturregisters darstellt und danach je vier Bytes für Strom, Spannung und Leistung. Die Daten werden dann gemäss Registertypen in `SByte`, `UInt32` und `Int32` konvertiert und umgerechnet. Bei Spannung und Strom wird der Eingangspegel am Sensor ausgegeben (0.5V), beim Temperaturregister gilt 0.667°C pro Einheit mit einem Offset von -25. Ob das stimmt, ist unklar, da die Temperaturmessung laut Datenblatt eine Toleranz von 25°C hat.

## **Kapitel 5**

# **Projektverlauf**

## 5.1 Zeitplan



Der Zeitplan umfasst alle Arbeiten innerhalb des P7-Projekts. Insbesondere bei der Beschaffung von elektronischen Bauteilen ist die Zeitplanung wegen Verfügbarkeit und Lieferzeiten schwierig, die Planung ist deshalb relativ grob. Damit bei Wartezeiten möglichst wenig Leerläufe entstehen, sind viele Tätigkeiten parallel geplant.

## 5.2 Projektphasen

### Konzept

Die erste Phase umfasst das Studium der bereits im Rahmen des Scintilla-Projekts geleisteten Arbeit und einer Ableitung eines Gesamtkonzepts daraus. Die bereits existierende Dokumentation zeigt, dass schon einige sinnvolle Ansätze erarbeitet wurden. Diese werden nun nochmals verfeinert und weitere Ansätze gesucht.

Nach einem Dialog mit Scintilla kristallisiert sich heraus, dass ein Konzept mit einem selbst entwickelten Sensor-Interface, welches in das Werkzeug eingebaut wird, und einem möglichst kleinen, extern angebrachten Embedded-Datenlogger am ehesten den Vorstellungen von Scintilla entspricht. Deshalb wird dieses Konzept auch definitiv gewählt.

Es wird nun vereinbart, dass im Rahmen des P7-Projekts der Fokus auf die Entwicklung eines Sensor-Interface gelegt wird. Die Entwicklung des Datenloggers und die Integration der Sensoren in die Werkzeuge wird in den Folgeprojekten stattfinden.

### Auswahl und Beschaffung der Hardware

In der Beschaffungsphase geht es darum, passende integrierte Schaltungen für das Sensor-Interface zu evaluieren und zu bestellen.

Die Wahl eines Mikrocontrollers zur Steuerung der Sensoren und Verarbeitung der Rohdaten fällt relativ leicht. Aufgrund bestehender Kenntnisse und früherer Erfahrungen fällt die Wahl auf einen Atmel AVR.

Die Auswahl passender Sensor-ICs gestaltet sich hingegen als schwieriger als erwartet. Die Auswahl an Herstellern und ICs ist gigantisch, deshalb werden zunächst Datenblätter potentieller Kandidaten gesammelt und studiert. Es ist aufgrund der grossen Auswahl kein Problem, ICs zu finden, die genau dem Anforderungsprofil der Sensoren entsprechen. Die Probleme entstehen erst bei der eigentlichen Beschaffung: Einige der gewünschten Sensoren sind auch bei grossen Distributoren wie Digikey, Distrelec oder Farnell nicht aufzutreiben, oder aber nur in grossen Stückzahlen. Der ADE7753 kann zumindest als Sample bei Analog Devices bestellt werden, eine Rückmeldung bleibt jedoch aus. Erst als seitens des Instituts für Mikroelektronik ein zweiter Versuch gestartet wird, kommt eine Lieferung an. Dies zeigt, dass die Beschaffung solcher Bauteile als Einzelkunde schwierig ist, wenn man keine guten Kenntnisse des Halbleitermarkts hat.

Bei der Wahl eines Beschleunigungssensors zeigt sich ein anderes Problem. Es können keine ICs gefunden werden, welche ein SMD-Gehäuse besitzen, das gut von Hand lötbar ist. Die Wahl fällt schliesslich auf den MMA7455L von Freescale, da dieser gut verfügbar ist.

Ein zweites Problem neben der Beschaffung der ICs ist die Wahl der passiven Bauteile. Standardbauteile wie Widerstände und Kondensatoren sind auch in kleiner Bauform grösstenteils im Lager des Instituts für Mikroelektronik vorhanden und können von dort bezogen werden. Bei einigen Bauteilen wie dem Stromwandler zur Messung der Stromstärke und dem Printtrafo zur galvanischen Trennung bei der Spannungsmessung gelingt es nicht, bei den bekannten Distributoren Produkte zu finden, die eine ideale Grösse für den Einbau in ein Werkzeug besitzen. Selbst die kleinsten Modelle haben ein Volumen von einigen Kubikzentimetern. Da aber im P7-Projekt noch keine Integration in die Werkzeuge nötig ist, können diese Bauteile dennoch verwendet werden.

Die meisten Komponenten können schlussendlich bei Farnell bestellt werden, da dieser Distributor eine sehr grosse Auswahl an Bauteilen bietet und dank einer Schweizer Vertretung eine Bestellung schnell und problemlos möglich ist.

## Hardware-Design

Der erste Schritt beim Hardware-Design ist die Herstellung einer Testplatine, auf der alle passiven aktiven Sensoren montiert werden können, um deren Funktion zu testen. Als Entwicklungstool kommt der Eagle Layout Editor zum Einsatz, da bereits Vorkenntnisse in der Nutzung vorhanden sind. Weil die meisten verwendeten ICs in der Eagle-Bibliothek nicht vorhanden sind, wird eine eigene Bibliothek mit den passenden Schemata und Packages erstellt. Diese Bauteile werden nun ins Schema der Testplatine eingefügt und darauf basierend ein Platinenlayout erstellt.

Die Fortschritte werden regelmässig mit dem Institut für Mikroelektronik besprochen, bis das Design reif für die Herstellung einer Platine ist. Erste Überlegungen gehen in die Richtung, dass die Platine extern gefertigt werden soll, es wird jedoch entschieden, dass eine interne Herstellung durch Ätzen einen zusätzlichen Lerneffekt bietet.

Neben der Testplatine wird noch eine Adapterplatine angefertigt, um den Beschleunigungssensor aufzulöten. Damit wird der Austausch im Falle des Scheiterns erleichtert.

Nach der Herstellung werden die Platinenkontakte getestet, da nur kleine Mängel vorhanden sind, kann mit der Bestückung begonnen werden. Die Platine wird jedoch nicht gleich komplett bestückt, sondern nur jeweils die Teile für einen einzelnen Sensor, der dann auf seine Funktion getestet wird. Alle Bauteile werden von Hand gelötet, wobei beim Beschleunigungssensor ein spezielles Verfahren unter Verwendung von Lötpaste und einem HeissluftlötKolben angewandt wird.

Der Mikrocontroller kann ebenfalls auf der Testplatine montiert werden, während den Tests verbleibt er allerdings auf dem AVR-Experimentierboard, da ein Austausch im Falle einer Zerstörung des AVR zu mühsam wäre. Deshalb wird das Experimentierboard stattdessen via SPI mit der Testplatine verbunden.

Die Werkzeuge, die von Scintilla zur Verfügung gestellt wurden, werden genau analysiert, um ihre Funktionsweise kennenzulernen und herauszufinden, an welchen Stellen sich die Sensoren einbauen lassen würden. Es wird entschieden, dass die Bohrmaschine als Referenz dienen soll. Einzig die Drehzahlmessung muss, bis ein passender Drehzahlmesser in die Bohrmaschine eingebaut werden kann, mit der Stichsäge durchgeführt werden, da dies das einzige Werkzeug mit vorhandenen Sensoren ist. Ausführliche Messungen der Steuerelektronik bringen zu Tage, dass das Drehzahlsignal durch eine kleine Modifikation der Steuerplatine abgegriffen werden kann. Die dazu nötigen Informationen werden von Scintilla bereitgestellt.

## Hardware-Tests

Bevor die Testplatine überhaupt bestückt wird, wird der analoge Teil der Strom- und Spannungsmessung, d.h. der Printtrafo und der Stromwandler, auf einer Lochrasterplatine montiert, um ihre Funktion zu testen. Mit Hilfe eines Regeltrafos und eines grossen Lastwiderstands werden die Teile getestet. Erfreulicherweise stellt sich heraus, dass das Konzept auf Anhieb funktioniert, so dass mit dem nächsten Schritt begonnen werden kann.

Der einfachste Sensor ist der Temperatursensor, weshalb er als erstes bestückt wird. Schon nach kurzer Zeit gelingt eine Ansteuerung, weshalb mit dem Beschleunigungssensor fortgefahren wird. Dabei stellt sich heraus, dass der Beschleunigungssensor offenbar nicht funktioniert, eine Ansteuerung via Mikrocontroller gelingt nicht. Der Versuch, ein zweites Exemplar des Sensors mit Bondingdrähten anzulöten misslingt ebenfalls, der Sensor wird zerstört. Damit ist offensichtlich, dass nach einer Alternative mit einem besser lötbaren Gehäuse gesucht werden muss. Für den neuen Sensor muss allerdings eine neue Testplatine entwickelt werden, welche erst noch getestet werden muss. Es ist auch nach längerer Suche unwahrscheinlich, dass eine SMD-Variante gefunden wird, welche sich leicht löten lässt. Ein erneuter Versuch mit dem ursprünglichen Sensor ist also durchaus denkbar.

Auch der Sensor zur Strom- und Spannungsmessung verursacht Probleme. Eine Ansteuerung gelingt zwar, die Register, welche die Messwerte zu Strom und Spannung enthalten, bleiben jedoch leer. Erst nach gründlicher Fehleranalyse gelingt die Ansteuerung.

## Software-Entwicklung

Die Software-Entwicklung beginnt schon während des Hardware-Designs. Der erste Schritt ist die Einarbeitung in die AVR-Entwicklungsumgebung, erste Mikrocontroller-Programme umfassen Tests mit dem UART, Interruptverarbeitung und Timern. Schnell ist eine Art Lauflicht mit LEDs realisiert, wobei der Output auch über den UART ausgegeben wird. Auf dem PC wird ein RS232-Testprogramm, welches Daten via UART empfängt, der Einfachheit halber in C# implementiert.

Nach der Montage des ersten Sensors wird mit den Tests des SPI-Bus begonnen. Die einfache Funktionsweise des SPI-Protokolls macht die Entwicklung einfach, lediglich das Timing führt teilweise zu Problemen. Als diese überwunden sind, können schliesslich die Messwerte via UART am PC empfangen werden.

## Fehleranalyse und -behebung

Die Fehleranalyse ist ein kontinuierlicher Prozess, welcher mit den ersten Hardwaretests startet. Als Messwerkzeuge stehen ein Multimeter und ein Oszilloskop bereit.

Ein erster Fehler, der erkannt wird, ist eine defekte Leiterbahn, welche dazu führt, dass die Spannungsversorgung instabil wird. Die Leiterbahn wird problemlos mit Draht repariert, was das Problem löst. Bei Signalmessungen mit dem Oszilloskop werden Fehler bei der SPI-Ansteuerung erkannt, insbesondere beim Timing. Diese können danach korrigiert werden. Es zeigt sich ebenfalls, dass bei einigen Bauteilen noch zusätzliche Glättungskondensatoren nötig sind, da die Signalform teilweise starke Störungen aufweist. Nach entsprechenden Modifikationen verbessert sich die Signalform massiv, die meisten Spannungsspitzen verschwinden.

Bei Tests mit dem Beschleunigungssensor zeigt sich, dass die Eingangssignale an den Pins offenbar korrekt sind, jedoch kein Signal zurückgeliefert wird. Da ein interner Test des IC nicht möglich ist, muss man davon ausgehen, dass er beim Auflöten durch zu grosse Hitze beschädigt wurde.

Der Grund für die Probleme mit dem Strom- und Spannungssensor ADE7753 ist die fehlende Spannungsversorgung der AD-Wandler, denn diese ist von der Spannungsversorgung des Logikteils getrennt. Nach einer entsprechenden Modifikation funktioniert der Sensor dann korrekt. So kann schliesslich doch noch gemessen werden, welche Werte an den Eingängen anliegen.

## 5.3 Ausblick

Die Entwicklung einer elektronischen Schaltung von Grund auf ist eine herausfordernde Aufgabe. Viele Kenntnisse sind nötig, damit am Schluss etwas Brauchbares herauskommen kann. Die ersten Schritte, insbesondere die Auswahl und Beschaffung der zu verwendeten Bauteile, sind wegweisend für den weiteren Verlauf des Projekts. Schliesslich bestimmt das Design der elektronischen Schaltung über Erfolg oder Misserfolg. Anders als bei der Software-Entwicklung ist ein iterativer Prozess wesentlich schwieriger, da jede Überarbeitung eines Schaltungsdesigns die erneute Herstellung eines Prototypen erfordert, was mit grossem Arbeitsaufwand und Kosten verbunden ist. Das Verhalten einer elektronischen Schaltung ist weniger vorhersagbar als bei Software, da physikalische Faktoren mitwirken. Das Studium der Datenblätter von ICs ist hilfreich, dennoch kommt es vor, dass sich diese Bauteile in einer realen Schaltung nicht ganz so verhalten, wie es in der Theorie beschrieben ist. Dadurch ist oft eine Nachprüfung in Form elektrischer Messungen notwendig, was den Zeitaufwand zusätzlich vergrössert.

Die Ergebnisse dieses Projekts muss man differenziert betrachten. Bei einigen Teilaufgaben konnten gute Resultate erzielt werden, andere müssen nochmals überarbeitet werden oder benötigen einfach mehr Zeit.

Das Design der Testplatine des Sensor-Interface funktioniert recht gut, es gibt nur wenige grundlegende Mängel. An der fertig geätzten Platine musste nachträglich nur wenig korrigiert werden. Verbessert werden kann aber beispielsweise die Platzierung von Entkopplungskondensatoren, da an gewissen Stellen die Spannungsversorgung leichte Störungen aufweist.

Die Wahl der Bauteile kann ebenfalls noch besser gemacht werden. Gerade bei Standardbauteilen wie Widerständen würde es Sinn machen, zuerst herauszufinden, welche Baugrössen man gerade an Lager hat. Man kann dann das Schaltungsdesign besser darauf abstimmen und muss nicht zuerst alles neu bestellen. Bei der Auswahl der ICs wird in Zukunft mehr darauf geachtet werden, wie die Verfügbarkeit bei den Distributoren aussieht. So können ICs, welche zwar theoretisch ideal sind, aber nur schwer lieferbar sind, von vorneherein ausgeschlossen werden.

Die Entscheidung, ausschliesslich SMD-Komponenten zu verwenden, ist aufgrund des Lerneffekts und der Platzersparnis sinnvoll, deshalb wird auch in Zukunft daran festgehalten.

Durchkontaktierungen werden in Zukunft vermieden, da sie bei selbst hergestellten Platinen zu Problemen führen, beispielsweise dann, wenn ein Bauteil von oben und unten gelötet werden muss, aber von oben nicht zugänglich ist. Sie werden also in künftig nur noch für Drahtbrücken bei einseitigen Platinenlayouts verwendet.

Die Idee, eine einzelne Testplatine herzustellen, welche sämtliche Sensoren beherbergt, ist grundsätzlich gut. Nachteilig ist aber, dass im Falle von Designfehlern die gesamte Platine neu konzipiert werden muss. Deshalb wird in zukünftigen Tests eher auf einzelne Platinen gesetzt, welche jeweils nur einen Sensor beinhalten. So muss im schlimmsten Fall nur ein Teil des Gesamtsystems neu designt werden. Für die spätere Integration in die Werkzeuge ist dieses Konzept sogar unerlässlich, weil es aufgrund der Platzverhältnisse unmöglich ist, eine einzelne Platine einzubauen. Besonderes Augenmerk ist dabei auf eine gute Verlegung der Versorgungs- und Datenbusleitungen zu legen. Das Design muss spezifisch an das Werkzeug angepasst werden, nur der grundlegende Schaltplan bleibt ungefähr gleich.

Die Mikrocontroller-Programmierung hat gut funktioniert, die Fehleranalyse gestaltet sich allerdings schwierig, da bei jeder noch so kleinen Korrektur der Programmcode erneut übertragen werden muss. Da echte Hardware angesteuert werden muss, kommt eine Simulation oder Emulation nicht in Frage. Es ist überlegenswert, sich einen JTAG-Adapter anzuschaffen, mit dem das On-Chip-Debugging möglich wird. Allerdings ist fraglich, ob dies beim vorliegenden Anwendungsfall sinnvoll ist, da insbesondere kritische Funktionen wie die Datenübertragung timingabhängig sind, und Interrupts asynchron auftreten, wobei der normale Programmfluss unterbrochen wird. Debugging würde eher bei softwarebasierten datenverarbeitenden Algorithmen Sinn machen.

In der nächsten Projektstufe (P8) wird der Fokus auf die Fertigstellung des Hardware-Konzepts und insbesondere auf die Entwicklung eines Datenloggers basierend auf einem Embedded-System gesetzt. Die Software-

Entwicklung wird dann von der Stufe Mikrocontroller zur Stufe Embedded-OS (z.B. Linux auf einem ARM-Core) verlagert. Das Ziel ist, dass, nachdem viele Erfahrungen mit dem ersten Sensor-Interface gemacht wurden, die relevanten Fehler im Hardware-Design in einer zweiten Iteration eliminiert werden, so dass man sich auf den Logger konzentrieren kann.

**Kapitel 6**

**Anhang**

## 6.1 Glossar

AVR	Mikrocontroller von Atmel
Bus	Datenübertragungsleitung, die von mehreren Komponenten gemeinsam genutzt wird
Chip Select	Datenleitung, die einen bestimmten IC aktiviert
Clock Phase	Bestimmt, ob Datenbits bei der ersten Taktflanke auf den Bus gelegt werden und bei der zweiten Flanke gesampelt werden (Setup-Sample), oder umgekehrt (Sample-Setup)
Clock Polarity	Bestimmt, ob ein Taktimpuls mit einer positiven oder negativen Flanke beginnt
Data Order	Bestimmt, ob zuerst das most significant Bit oder das least significant Bit gesendet wird
Drehzahl	Anzahl Umdrehungen einer Achse während einer bestimmten Zeit
I <sup>2</sup> C	Inter-Integrated Circuit (serieller Bus)
Lage	Ausrichtung eines Objekts relativ zur Richtung der Erdgravitation
Leistung	Energie, die während einer bestimmten Zeit umgesetzt wird
PCB	Printed Circuit Board (Platine)
RS232	Standard für serielle Schnittstelle
SMD	Surface Mounted Device, beschreibt eine Technologie elektronischer Bauteile, die direkt auf die Leiterbahnen aufgelötet werden
Spannung	elektrische Potentialdifferenz zwischen zwei Punkten
SPI	Serial Peripheral Interface (serieller Bus)
Strom	Anzahl Elektronen, die während einer bestimmten Zeit einen Leiter durchfließen
Typ13	Schweizer Standard-Netzstecker für 230 V und max. 10 A
UART	Universal Asynchronous Receiver Transmitter (serielles Interface)

## 6.2 Dank

Ich möchte mich herzlich bei all denen bedanken, die mich während dieses Projekts bereut und unterstützt haben, und die mir bei Fragen und Problemen auch dann Hilfestellung geboten haben, wenn es nicht zwingend ihre Aufgabe war.

Besonderer Dank geht an:

Christoph Stam, Bernhard Wyss, Martin Schindler, Cyrill Grüter (Institut für Mobile und Verteilte Systeme)

Hans Buchmann, Stefan Muhr (Institut für Mikroelektronik)

Christoph Biel (Institut für Automation)

Mario Frank, Roland Hund (Scintilla)

Jan Kleinert

# Literaturverzeichnis

- [WY08] Scintilla Konzept V2.0, Bernhard Wyss, FHNW, 2008
- [KR09] Aktiver Datenlogger - Hardwarekonzept, Matthias Krebs, FHNW, 2009
- [HU09] Roland Hund, Scintilla, 2009
- [AT16] Datenblatt Atmel ATmega16  
[http://www.atmel.com/dyn/resources/prod\\_documents/doc2466.pdf](http://www.atmel.com/dyn/resources/prod_documents/doc2466.pdf)
- [TC77] Datenblatt Microchip TC77  
<http://ww1.microchip.com/downloads/en/DeviceDoc/20092a.pdf>
- [ADE7] Datenblatt Analog Devices ADE7753  
[http://www.analog.com/static/imported-files/data\\_sheets/ADE7753.pdf](http://www.analog.com/static/imported-files/data_sheets/ADE7753.pdf)
- [ADEE] Dokumentation des ADE7753-Evaluationsboards  
[http://hallaweb.jlab.org/tech/Detectors/public\\_html/manuals/chip\\_specs/A-L/analog\\_devices\\_pmi/ADE7753Eval\\_prB.pdf](http://hallaweb.jlab.org/tech/Detectors/public_html/manuals/chip_specs/A-L/analog_devices_pmi/ADE7753Eval_prB.pdf)
- [MMA7] Datenblatt Freescale MMA7455L  
[http://www.freescale.com/files/sensors/doc/data\\_sheet/MMA7455L.pdf](http://www.freescale.com/files/sensors/doc/data_sheet/MMA7455L.pdf)
- [PROC] Procyon AVRlib  
<http://www.mil.ufl.edu/~chrisarnold/components/microcontrollerBoard/AVR/avr-lib/>
- [ECAV] Eclipse AVR Plugin  
<http://avr-eclipse.sourceforge.net>