

**VORLÄUFIGE ARBEITSKOPIE!**

**ÜBERSETZEN VON**

**SCHRITTMOTORPROTOKOLLEN**

**Entwurf eines Hardwareübersetzers**

**Praxisbericht**

im Fachgebiet Mess- und Sensortechnik



vorgelegt von: Johannes Dielmann  
Studienbereich: Technik  
Matrikelnummer: 515956  
Erstgutachter: Prof. Dr. Carstens-Behrens

© 2012

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>III</b>
<b>1. Einleitung</b>	<b>1</b>
1.1. Motivation . . . . .	1
1.2. Ziel der Arbeit . . . . .	1
1.3. Aufbau der Arbeit . . . . .	2
<b>2. Hardware</b>	<b>3</b>
2.1. Lasererfassungssystem VI-900 . . . . .	3
2.1.1. Lasertriangulator Prinzip . . . . .	4
2.2. Ansteuerung für den Drehtisch . . . . .	4
2.2.1. Drehtisch . . . . .	4
2.2.2. Spannungsversorgung . . . . .	5
2.2.3. Schrittmotoren . . . . .	6
2.2.4. Schrittmotorkarten . . . . .	6
2.2.5. Motorverkabelung . . . . .	7
2.2.6. Endschalter . . . . .	7
2.3. Mikrocontroller . . . . .	8
2.3.1. Atmega 324A . . . . .	10
2.3.2. Entwicklerboard STK500 . . . . .	10
2.3.3. AVRISP mkII . . . . .	11
2.3.4. MAX232 . . . . .	11
2.4. Platinenlayout . . . . .	11
2.5. 19"-Einschub . . . . .	12
<b>3. Software</b>	<b>13</b>
3.1. RapidForm2004 . . . . .	13
3.2. Entwicklungsumgebung . . . . .	13
3.2.1. AVR Studio 5 . . . . .	13
3.2.2. Eclipse . . . . .	13

3.3. Mikrocontroller . . . . .	13
3.3.1. Fuses . . . . .	14
3.3.2. LEDs . . . . .	15
3.3.3. Taster . . . . .	15
3.3.4. LCD Bibliothek . . . . .	16
3.3.5. RS-232 . . . . .	17
3.3.6. Menü Bibliothek . . . . .	18
3.3.7. Interrupts . . . . .	18
3.3.7.1. Endschalter . . . . .	18
3.3.7.2. Watchdog . . . . .	19
3.3.8. Protokoll der Schrittmotorkarte . . . . .	19
3.3.9. Manueller Betrieb . . . . .	20
3.3.10. Protokolle aus RapidForm . . . . .	20
3.3.11. Übersetzungs Logik . . . . .	21
3.3.11.1. FindStringInArray Funktion . . . . .	21
3.3.11.2. Automatische Protokollwahl . . . . .	21
3.3.11.3. Zeta . . . . .	22
3.3.11.4. Isel . . . . .	22
3.3.11.5. Weitere . . . . .	25
<b>4. Fazit und Zukunft</b>	<b>26</b>
4.1. Fazit . . . . .	26
<b>Eidesstattliche Erklärung</b>	<b>28</b>
<b>A. Anhang</b>	<b>i</b>
A.1. Schritt für Schritt Anleitung . . . . .	ii
A.2. Technische Daten VI-910 . . . . .	iii
A.3. Verwendete Software . . . . .	v
A.4. Verwendete Hardware . . . . .	v

## Abkürzungsverzeichnis

ADC .....	analog digital convertor
ASCII .....	American Standard Code for Information Interchange
AVRISP .....	AVR in system programmer
CAD .....	(engl. computer-aided design) Computer gestützter Entwurf
CF .....	Compact Flash
CPU .....	central processing unit
DAC .....	digital analog convertor
DIL .....	dual in line package
IC .....	integrated cuircuit
LCD .....	(engl. liquid crystal display) Flüssigkristall Display
MHz .....	megahertz
RS .....	Radio Sector
SCSI .....	Small Computer System Interface
USB .....	universal serial bus
V .....	Volt

## Abbildungsverzeichnis

2.1. Überblick des Arbeitsplatz . . . . .	3
2.2. VI-900 . . . . .	4
2.3. Laser-Triangulation - Prinzip . . . . .	5
2.4. Ansteuerung - 19"-Rack . . . . .	5
2.5. Drehtisch . . . . .	6
2.6. Stromverbinder - Y-Kabel . . . . .	6
2.7. Motor- und Umschalerverkabelung . . . . .	7
2.8. Motor- und Umschalerverkabelung . . . . .	8
2.9. Block Diagram eines Mikrocontroller . . . . .	9
2.10. Schema - STK500 . . . . .	10
2.11. Schema - MAX232 . . . . .	11
2.12. Platinenlayout . . . . .	12

# Tabellenverzeichnis

2.1. Motor- und Endschalterverkabelung . . . . .	7
3.1. Fuses . . . . .	14
3.2. ASCII Befehlssatz R+S Schrittmotorsteuerung . . . . .	20
A.1. Technische Daten - VI-910 . . . . .	iii

## Codeverzeichnis

3.1. Funktion - Laufflicht . . . . .	15
3.2. Taster . . . . .	15
3.3. Definitionen - LCD(Auszug) . . . . .	16
3.4. Funktionen - RS-232 . . . . .	17
3.5. ISR - Endschalte . . . . .	18
3.6. Watchdog . . . . .	19
3.7. Protokoll aus Rapidform - Zeta . . . . .	21
3.8. Funktion - FindStringInArray . . . . .	21
3.9. Funktion - switchMotor . . . . .	22
3.10. Übersetzungs Logik - Isel . . . . .	22

# 1. Einleitung

**(TODO: DIE EINLEITUNG IST SEHR WICHTIG!!!)**

Die 3D-Lasererfassung bietet zahlreiche Anwendungsgebiete. Von der Erfassung kleinerer Objekte, über die Erfassung von Hindernissen bis hin zur Erkennung von Hauswänden.

Im medizinischen Bereich z.B. können Zahnprothesen erfasst und vermessen werden. Im Straßenverkehr können Hindernisse erfasst, kategorisierte und bewertet werden. Google nutzt bei Streetview? Lasererfassung um die Entfernung zu Häuserfronten zu bestimmen.

## 1.1. Motivation

Im Projekt soll nun mit einer Kombination aus einem Lasererfassungssystem, einem Drehtisch und der dazugehörigen Software so vorbereitet werden, dass er unterstützend in der *CAD-Entwicklung* genutzt werden kann. In der CAD-Entwicklung kann es vorkommen das für ein real existierendes Objekt eine *Erweiterung* konstruiert werden soll. Um die Erweiterung, einen Anschlag zum Beispiel, leichter konstruieren zu können ist es von Vorteil die Abmessungen des Objektes möglichst genau zu kennen. Das übertragen der Abmessungen in die CAD-Software, kann insbesondere für komplexe Objekte, sehr aufwendig sein. Abhilfe soll der Laserscanner schaffen, der das Objekt aus mehreren Richtungen vermisst und aus diesen Informationen ein 3D-Modell generiert. Dieses soll dann in einem neutralen CAD-Format **(TODO: WELCHES?)** exportiert werden.

Dabei sind mehrere Hürden zu nehmen. Die Ansteuerung des Drehtisches ist nicht kompatibel mit den Protokollen der Software. Die Höhenverstellung des Drehtisches soll genutzt werden können und die Endschalter sollen funktionieren.

## 1.2. Ziel der Arbeit

Die Kommunikation zwischen Software und Drehtisch soll ermöglicht werden. Dazu sollen die Befehle der Software mit einem Mikrocontroller ausgewertet werden und



## 1. Einleitung

---

in für den Drehtisch verständliche Befehle übersetzt.

Für die Höhenverstellung des Drehtisches wird eine Manuelle Steuerung im Mikrocontroller realisiert und die Endschalter so verdrahtet das sie funktionieren.

Der Mikrocontroller lässt sich mit mehreren Tastern bedienen und ein *LC-Display* zeigt den aktuellen Status an.

Der Mikrocontroller und seine Peripherie werden als *19Einschub* realisiert, da die Ansteuerung des Drehtisches auch als Einschub realisiert ist.

Als Abschluss soll ein komplettes 3D-Modell für die Nutzung in einer CAD-Software zur Verfügung stehen und die Aufnahme eines 3D-Modelles für andere Personen möglich sein.

### 1.3. Aufbau der Arbeit

Der Aufbau der Arbeit gliedert sich im Wesentlichen in die Nutzung vorhandener und die Entwicklung neuer Hardware, sowie in die Entwicklung der Software für den Mikrocontroller und einer Schritt-für-Schritt Anleitung.

Zur Hardware gehören der Laserscanner, die Ansteuerung für den Drehtisch sowie der Drehtisch selbst, seine Spannungsversorgung, die Schrittmotoren und die Schrittmotorkarten, die Motorverkabelung, die Endschalter, sowie der Mikrocontroller als auch das Platinenlayout und der 19"-Einschub.

Zur Software gehören die 3D-Erfassungsoftware, die Entwicklungsumgebungen und die Software für den Mikrocontroller.

Im Anhang befindet sich eine Schritt-für-Schritt Anleitung um 3D-Modelle aufzunehmen und zu exportieren.

**(TODO: ÜBERBLICK VERSCHAFFEN. KOMMUNIKATION MIT SCHRITTMOTOR-KARTE VON PC AUS. AUFBAUEN DES STK500. EINARBEITEN IN UC ENTWICKLUNG. STEuern DER SCHRITTMOTOR KARTE VOM UC AUS. ERARBEITEN DER PROTOKOLLE (REVERSE ENGINEERING). C-PROGRAMM ZUM VERSTEHEN EINGEHENDER BEFEHLE. ÜBERSETZEN DER BEFEHLE. NEUER MIKROCONTROLLER. UMGEBUNGSWECHSEL. LC-DISPLAY. ENDSCHALTER. MAX232. PLATINENLAYOUT.)**

## 2. Hardware

Die Hardware besteht im wesentlichen aus den Komponenten in Abbildung 2.1.  
(TODO: NUMMERN ODER FARBEN IN BILD!) (TODO: KOMPONENTEN

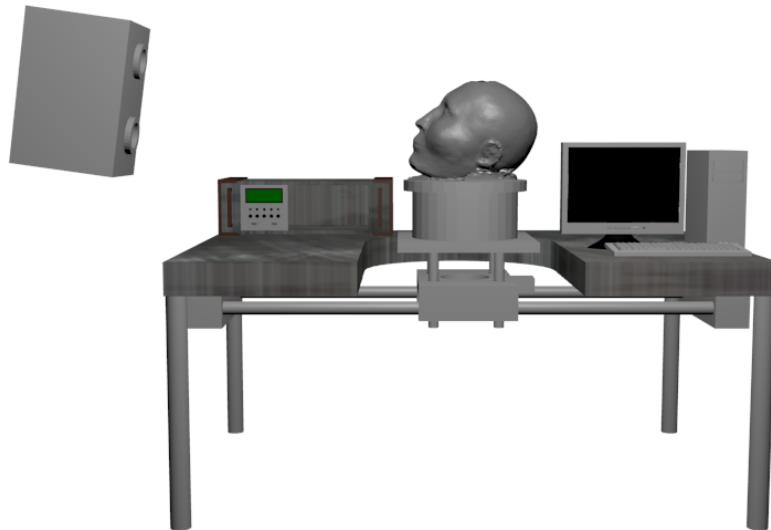


Abbildung 2.1.: Überblick des Arbeitsplatz

AUF ABBILDUNG ERWÄHNEN!)

### 2.1. Lasererfassungssystem VI-900

Das Lasererfassungssystem *VI-900* der Firma *Minolta* besteht, wie auf Abbildung 2.2 zu sehen, aus einem *Lasertriangulator*(unten) und einer Kamera(oben). Das System lässt sich über eine *SCSI*-Schnittstelle ansprechen und konfigurieren. Zur mobilen Nutzung kann das Gerät auch auf der Rückseite bedient werden. Aufgenommene Daten können auf einer *CF-Karte* gespeichert werden. Im Projekt wurde jedoch lediglich die direkte Ansteuerung via *SCSI* genutzt.



Abbildung 2.2.: VI-900

### 2.1.1. Lasertriangulator Prinzip

Ein Lasertriangulator, wie in Abbildung 2.3 zu sehen, besteht aus einem Laser, einem Linsensystem und im einfachsten Fall aus einer Pixeldetektorzeile. Der Laser strahlt auf ein Objekt und je nach Entfernung des Objektes wird das Streulicht unter einem anderen Winkel zurückgestrahlt. Das Streulicht wird durch die Linsen auf den Pixeldetektor abgebildet. Über die Position des Laserspots auf dem Pixeldetektor lässt sich auf die Entfernung des Objektes zurück schließen.

Der VI-900 digitalisiert Objekte durch ein Laser-Lichtschnittverfahren. Das vom Objekt reflektierte Licht wird von einer CCD-Flächenkamera erfasst, nach Ermittlung der Distanzwerte (Z-Achse) mittels Laser-Triangulation werden die 3D-Daten erstellt. Der Laserstrahl wird mit Hilfe eines hochpräzisen galvanischen Spiegels über das Objekt projiziert, pro Scan werden 640 x 480 Einzelpunkte erfasst.?

Die Technischen Daten befinden sich im Anhang in Tabelle A.1

## 2.2. Ansteuerung für den Drehtisch

Die Ansteuerung für den Drehtisch ist in einem 19"-Rack verbaut.

### 2.2.1. Drehtisch

Der Drehtisch(siehe Abbildung 2.1 ) ist eine Eigenkonstruktion der Werkstatt des RheinAhrCampus. Er besteht aus einer massiven Edelstahl Arbeitsplatte, welche auf 4 Füßen ruht. Aus dieser ist ein Rechteck mit aufgesetztem Halbkreis ausgeschnitten. In diesem Ausschnitt befindet sich der Drehtisch. Er ist auf einem Schienensystem

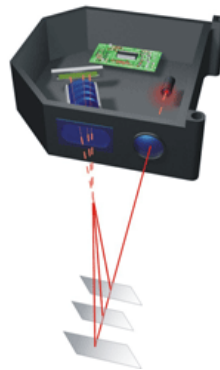


Abbildung 2.3.: Laser-Triangulation - Prinzip

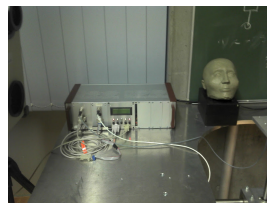


Abbildung 2.4.: Ansteuerung - 19"-Rack

gelagert. Mit dem Schienensystem kann der Drehtisch in der Vertikalen positioniert werden. Mit einem Schrittmotor lässt sich der Drehtisch sich zusätzlich in der Höhe verstellen. Die Höhenverstellung wird mit einem *Schneckengetriebe* realisiert. Ein weiterer Schrittmotor ist für die Drehung des Tisches zuständig. Der Tisch ist über ein *Harmonic-Drive-Getriebe* mit dem Schrittmotor verbunden. Das Übersetzungsverhältnis beträgt 1:50.

### 2.2.2. Spannungsversorgung

Die Schrittmotorkarten werden von einem PC-Netzteil gespeist. Die Kabel waren direkt an die Verbindungsleisten gelötet. Um den Aufbau modular und erweiterbar zu machen, ersetzte ich die feste Lötverbindung durch eine Standard PC-Netzteil Verbindung. Dadurch kann das Netzteil nun einfach ausgebaut werden, bzw. das System leicht mit neuen Einschubkarten erweitert werden.

Die Logik der Schrittmotorkarten und die Mikrocontroller-Platine werden mit 5V gespeist. Zusätzlich wurden die Schrittmotorkarten mit 12V, für die die Schrittmotoren gespeist. (TODO: [http://www.kosatec.de/prod\\_images/kc/640x480/100539.jpg](http://www.kosatec.de/prod_images/kc/640x480/100539.jpg))

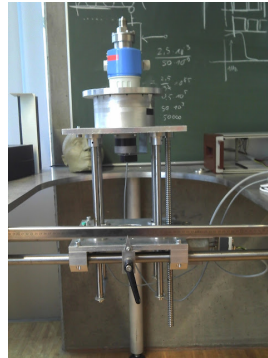


Abbildung 2.5.: Drehtisch

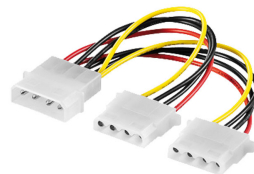


Abbildung 2.6.: Stromverbinder - Y-Kabel

### 2.2.3. Schrittmotoren

**(TODO: MOTOREN BESCHREIBEN! TECHNISCHE DATEN! SCHRITTE, SPANNUNGEN. VERDRAHTUNG.)**

### 2.2.4. Schrittmotorkarten

Die Ansteuerung für die Schrittmotoren sind als 19Einschübe realisiert. Für jeden Schrittmotor wird ein Einschub benötigt. Die Einschübe sind Produkte der Firma R+S. Mittels *RS-232 Schnittstelle* lassen sich die Karten konfigurieren und ansteuern. Die Konfiguration und Ansteuerung erfolgt über einen vorgegeben *ASCII*<sup>1</sup> Befehlssatz. Der Befehlssatz befindet sich im Kapitel 3.3.8. Es können zwei oder mehr Karten als *Daisy-Chain*<sup>2</sup> in Reihe geschaltet werden.

---

<sup>1</sup>Der American Standard Code for Information Interchange (ASCII, alternativ US-ASCII, oft [æski] ausgesprochen) ist eine 7-Bit-Zeichenkodierung?

<sup>2</sup>Als Daisy Chain (englisch, wörtlich „Gänseblümchenkette“) bezeichnet man eine Anzahl von Hardware-Komponenten, welche in Serie miteinander verbunden sind (meist in sogenannten Bussystemen in der Automatisierungstechnik). [Wikipedia \[2012a\]](#)

### 2.2.5. Motorverkabelung

Die Schrittmotoren benötigen ein mindestens 4-adriges Kabel. Das Kabel für den Schrittmotor der für die Rotation zuständig ist war bereits gefertigt. Das Kabel für den Schrittmotor der für die Höhenverstellung zuständig ist, wurde selbst gefertigt. Hier wurden 3 weitere Adern für die beiden Endschalter benötigt.

Abbildung 2.8 zeigt eine schematische Darstellung des Kabel. Tabelle ?? gibt die Belegung der Kabel wieder. **(TODO: BELEGUNG ÜBERPRÜFEN!)**

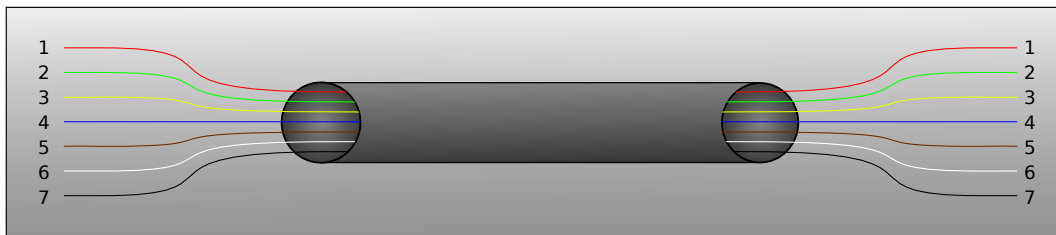


Abbildung 2.7.: Motor- und Endschalterverkabelung

Tabelle 2.1.: Motor- und Endschalterverkabelung

1	Phase A
2	Phase B
3	Phase C
4	Phase D
5	Endschalter Oben
6	Endschalter Unten
7	Endschalter Masse

### 2.2.6. Endschalter

Die Schrittmotorkarten unterstützen das Abschalten der Motoren wenn ein sogenannter Endschalter ausgelöst wird. Dies sind im allgemeinen mechanische Schalter die ausgelöst werden wenn der Tisch sich dem Ende des Arbeitsbereiches nähert. Dies verhindert eine Beschädigung des Aufbaus.

Im Aufbau waren bereits induktive Endschalter der Firma Pepperl+Fuchs verbaut. Normalerweise unterstützt die Schrittmotorkarte nur mechanische Endschalter. Durch geschickte Verdrahtung ließen sich die induktiven Endschalter verwenden. Hierzu musste über einen Spannungsteiler die Spannung herabgesetzt werden. Dadurch

konnten die Endschalter direkt an die Optokoppler der Schrittmotorkarte angeschlossen werden.

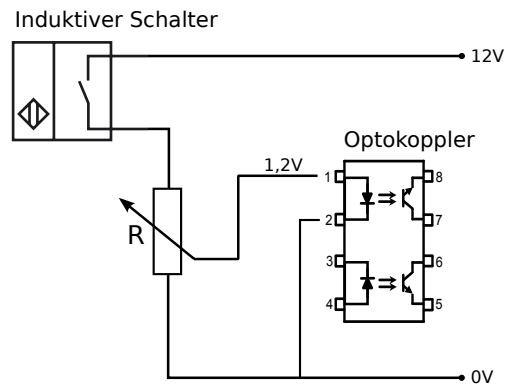


Abbildung 2.8.: Motor- und Enschalterverkabelung

Am Drehtisch war ein Metallstutzen angebracht der den Endschalter auslösen sollte. Dieser war jedoch ungeeignet da er nicht dicht genug an den Induktiven Schalter heran kam, obwohl der Tisch schon in der Endposition war. Abhilfe schaffte ein längerer Metallstutzen der von der Werkstatt gefertigt wurde.

Wenn der Tisch sich in der Endposition befindet soll dies auch auf dem Mikrocontroller angezeigt werden. Die Signale der Endschalter liegen auf der Rückseite (**TODO: ZEICHNUNG DER ANSCHLÜSSE REFERENZIEREN.**) am Verbindungsstecker an. Es muss also nur eine Brücke zu den entsprechenden Pins des Verbindungsstecker des Mikrocontroller gelötet werden.

Auf der Mikrocontroller Platine sind diese Pins mit zwei Pins des Mikrocontroller verbunden. Die beiden Pins werden im Mikrocontroller als Interrupts definiert. Die Interrupt-Service-Routine wird in Kapitel 3.3.7 beschrieben.

## 2.3. Mikrocontroller

Ein *Mikrocontroller* vereint, in einem IC, die wichtigsten Komponenten um komplexe technische Probleme leicht lösen zu können. Dazu gehören z.B. CPU, Flash-Speicher, Arbeitsspeicher, Register, Ports, ADC, DAC und mehr. Einen schematischen Überblick über die Komponenten eines Mikrocontroller bietet das Blockdiagramm in Abbildung 2.9.

Um die Aufgaben zu lösen muss der Mikrocontroller mit einem Programm beschrieben werden. Diese Programme lassen sich am einfachsten in einer Entwicklungsumgebung, siehe Kapitel 3.2 schreiben. Diese Programme können dann z.B. Signale an

## 2. Hardware

Pins des Mikrocontroller auswerten und Signale über andere Pins wieder ausgeben und dadurch externe Geräte steuern. Eingehende Signale können binär ausgewertet werden oder mit einem ADC die Spannungshöhe bestimmt werden. Ausgehende Signale können auch binär oder mit einem DAC analog ausgegeben werden. Binäre Signale können zur Steuerung von LEDs oder Peripherie Geräten genutzt werden. Auch LC-Displays und Serielle Schnittstellen können so angesteuert werden.

Für unterschiedliche Aufgaben sind unterschiedliche Mikrocontroller geeignet. Zu Beginn stand ein ATmega 8515 [Atmel \[2012b\]](#) im DIL-Gehäuse zur Verfügung. Dieser hatte 8 Kbyte Flash, 3 externe Interrupts, 1 Serielle Schnittstelle und konnte mit bis zu 16 MHz betrieben werden. Dieser war geeignet um sich in die Programmierung mit C ein zu finden und eine Serielle Schnittstelle an zu steuern.

Für dieses Projekt sind jedoch 2 externe Schnittstellen nötig. Der ATmega 324A [Atmel \[2012a\]](#) erfüllt diese Voraussetzung. Er ist dem ATmega 8515 recht ähnlich, bietet jedoch die benötigten 2 seriellen Schnittstellen. Des weiteren hat er 32 Kbyte Flash. Diese wurden aufgrund des recht umfangreichen Programms und den diversen Bibliotheken notwendig.

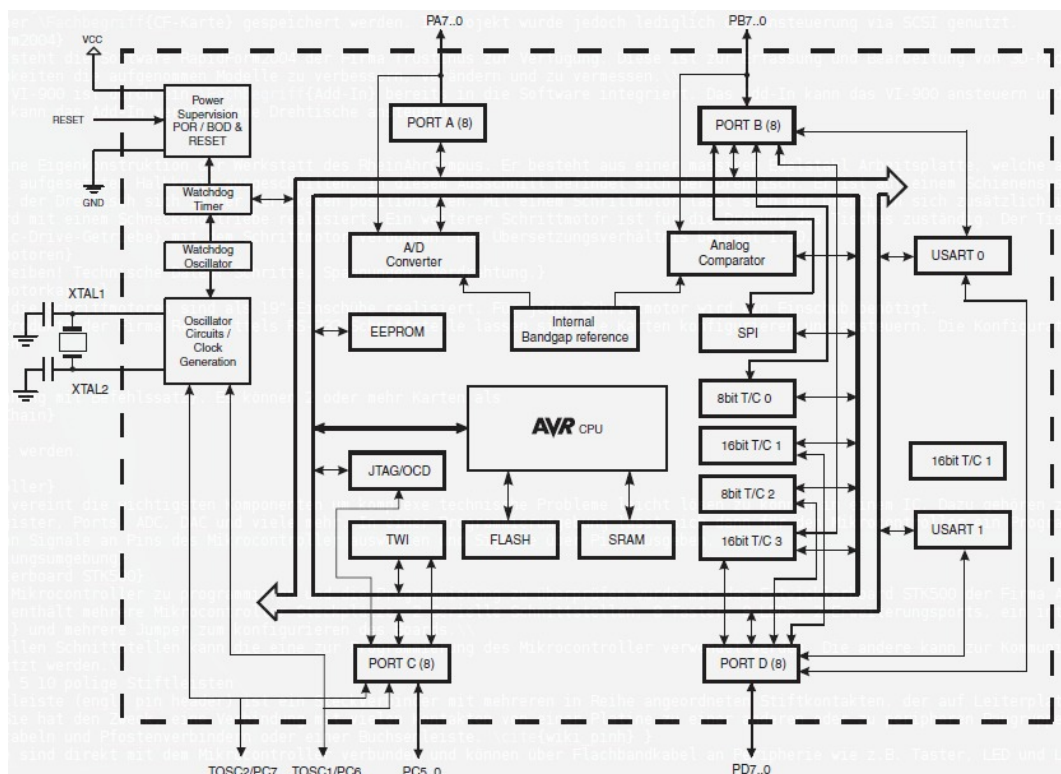


Abbildung 2.9.: Block Diagram eines Mikrocontroller  
[Atm 2011]



### 2.3.1. Atmega 324A

(TODO: BLUB)

### 2.3.2. Entwicklerboard STK500

Um Mikrocontroller zu programmieren und die Programmierung zu überprüfen kann das *Entwicklerboard* STK500 (siehe Abbildung 2.10) der Firma ATMEL verwendet werden. Das Board enthält mehrere Mikrocontroller Steckplätze, 2 serielle Schnittstellen, 8 Taster, 8 LEDs, 2 Erweiterungsports, eine Programmierschnittstelle *ISP*<sup>3</sup> und mehrere Jumper zum konfigurieren des Boards.

Von den beiden seriellen Schnittstellen kann die eine zur Programmierung des Mikrocontroller verwendet werden. Die andere kann zur Kommunikation mit dem Mikrocontroller genutzt werden.

Auf dem Board stehen fünf 10 polige Stiftleisten zur Verfügung. Diese sind direkt mit den Ports des Mikrocontroller verbunden und können über Flachbandkabel mit Peripherie wie z.B. Taster, LED, LC-Displays oder seriellen Schnittstellen verbunden werden.

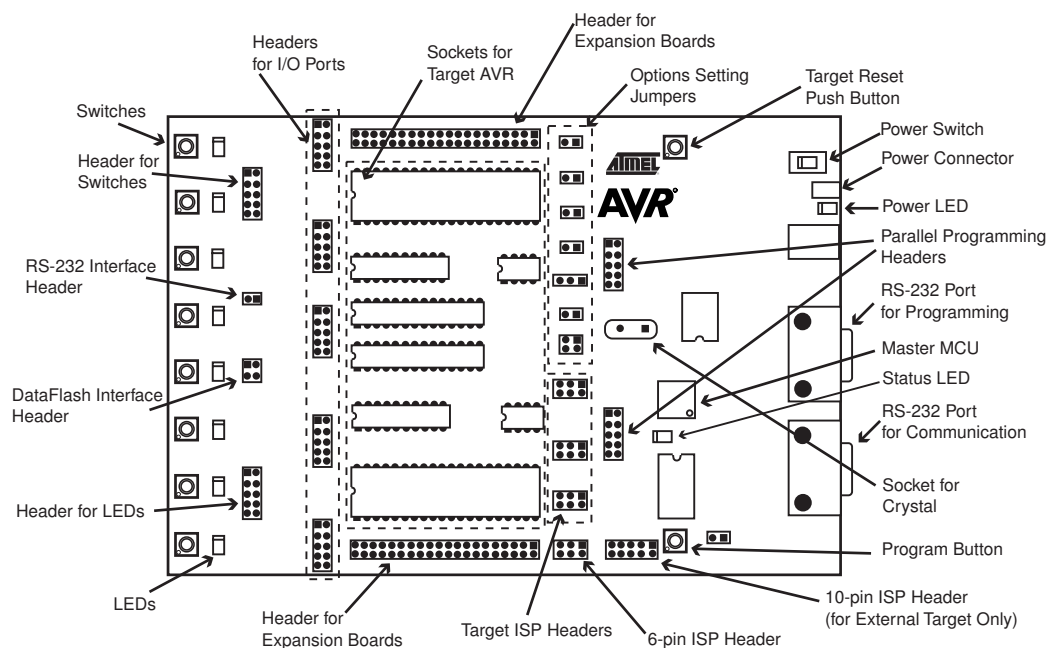


Abbildung 2.10.: Schema - STK500  
[?]

<sup>3</sup>In System Programmer

Der AVRISP mkII lässt sich einfach an den Programmierport, eine 6-Polige Stiftleiste, des STK500 anschließen.

### 2.3.4. MAX232

Abbildung 2.11.: Schema - MAX232  
[?]

## 2.4. Platinenlayout

Für den Mikrocontroller und seine Peripherie wurde ein Platinenlayout entwickelt. Dieses wurde in der Opensource Software KiCad entwickelt. Dazu wurden die Schaltungen wie auf dem STK500 in den Schaltplan übernommen und dort das Layout entwickelt. **(TODO: SCHALTPLAN UND EINBINDEN.)**

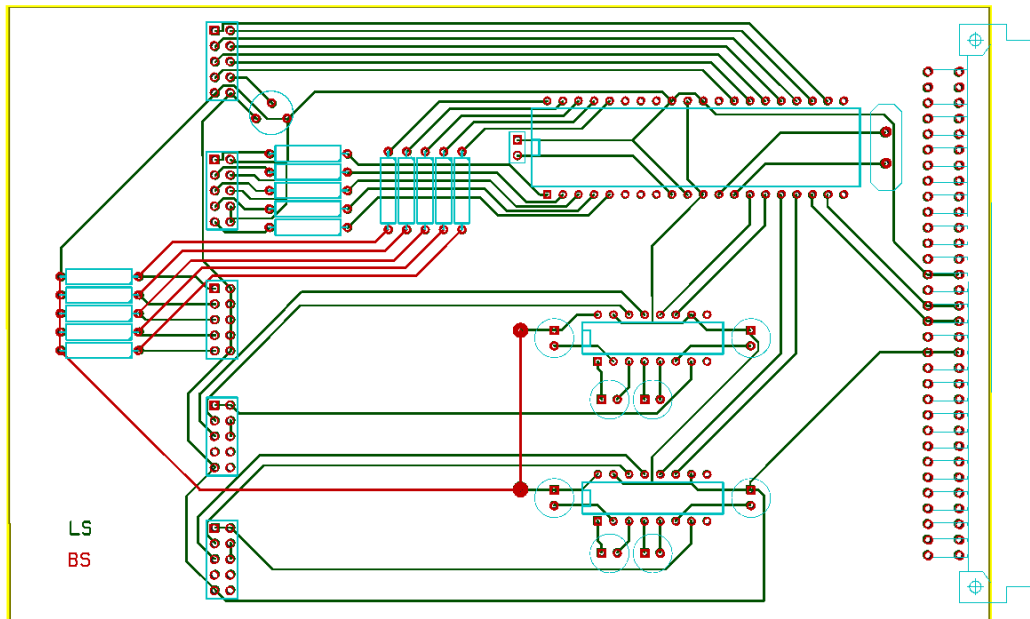


Abbildung 2.12.: Platinenlayout

## 2.5. 19"-Einschub

Die Platine für den Mikrocontroller wurde als 19"-Einschub konstruiert. Über den rückwärtigen Steckverbinden wird die Platine mit Spannung versorgt. Zusätzlich kommen hier auch die Signale der Endschalter an. An der Vorderseite wird die Blende befestigt. Auf der Blende befinden sich das LC-Display, fünf Taster, 5 LEDs und 2 serielle Schnittstellen.

Alle Bauteile sind steckbar mit der Platine verbunden.

## 3. Software

(TODO: EINFÜHRUNG SCHREIBEN) (TODO: WEITERE SOFTWARE IN BEGRIFFEN ERKLÄREN. MINOLTA)

### 3.1. RapidForm2004

Zur Erfassung am PC steht die Software RapidForm2004 der Firma TrustInus zur Verfügung. Diese ist zur Erfassung und Bearbeitung von 3D-Modellen gedacht. Sie bietet umfangreiche Möglichkeiten die aufgenommen Modelle zu verbessern, verändern und zu vermessen.

Die Ansteuerung des VI-900 ist durch ein *Add-In* bereits in die Software integriert. Das Add-In kann das VI-900 ansteuern und die Aufgenommenen Daten auslesen. Weiterhin kann das Add-In verschiedene Drehtische ansteuern.

### 3.2. Entwicklungsumgebung

Als Entwicklungsumgebung wird eine Software bezeichnet die es dem Anwender erleichtert Programme für den Mikrocontroller zu schreiben. Im allgemeinen bestehen Entwicklungsumgebungen aus einem Editor, dem Compiler und einer Programmiersoftware. Der Editor bietet dabei meist Komfortfunktionen wie Syntaxhighlighting, Autovervollständigung und Projektmanagement. (TODO: BESSER SCHREIBEN!)

#### 3.2.1. AVR Studio 5

(TODO: AVR STUDIO ECLIPSE BUG DEFECTS BIBLIO?)

#### 3.2.2. Eclipse

### 3.3. Mikrocontroller

(TODO: CODEBEISPIELE SIND ZUSAMMENGEFASST. VOLLSTÄNDIGER CODE IM ANHANG.) (TODO: BACKUP ANLEGEN UND CLEANEN!)

### 3.3.1. Fuses

Als Fuses werden Register bezeichnet mit denen sich, auf Hardwareebene, das Verhalten des Mikrocontroller verändern lässt.

Im Projekt wurden folgende Fuses problematisch.

- **JTAGEN** - Ist dieses Fusebit gesetzt werden 4 Pins des PortB genutzt um den Mikrocontroller zu Debuggen und können nicht anders genutzt werden. Hardware Debugging bietet viele Vorteile. Diese wurden im Projekt jedoch nicht genutzt da PortB für die LEDs genutzt wurde.
- **WDTON** - Ist dieses Fusebit gesetzt läuft der Watchdog Timer immer mit. Wird der Watchdog dann nicht regelmäßig zurückgesetzt startet der Mikrocontroller ständig neu.
- **CKDIV8** - Teilt den Systemtakt des Mikrocontroller durch 8. Dies ist Energiesparender. Der geringere Takt muss in F\_CPU angepasst werden da sonst Zeitkritische Prozesse mit der falschen Geschwindigkeit ablaufen.
- **CKOUT** - An PortB wird an einem Pin der Systemtakt ausgegeben. Dieser kann dann leicht mit einem Frequenz-Messgerät überprüft werden. Der Pin kann dann jedoch nicht anderweitig genutzt werden.
- **CKSELX** - Über diese 4 Bits kann der Systemtakt eingestellt werden.

Tabelle 3.1.: Fuses

OCDEN	On Chip Debugging
JTAGEN	Hardware Debugging
SPIEN	Serial Program and Data Downloading
WDTON	Watchdog Timer always on
EESAVE	EEPROM memory is preserved through the Chip Erase
BOOTSZ1	Select Boot Size
BOOTSZ0	Select Boot Size
BOOTRST	Select Reset Vector
CKDIV8	Divide clock by 8
CKOUT	Clock output
SUT1	Select start-up time
SUT0	Select start-up time

CKSEL3	Select Clock source
CKSEL2	Select Clock source
CKSEL1	Select Clock source
CKSEL0	Select Clock source

### 3.3.2. LEDs

Das Codebeispiel 3.1 zeigt ein einfaches Beispiel mit dem sich die Funktionalität der LEDs leicht überprüfen lässt. Bei jedem Aufruf der Funktion wird der aktuelle Status des LED Port abgefragt und der Hexwert um 1 Bit verschoben. Dadurch wird die daneben liegende LED eingeschaltet und die aktuelle aus geschaltet. Wird ein bestimmter Wert überschritten wird der Port wieder auf den Anfangszustand zurück gesetzt.

Listing 3.1: Funktion - Lauflicht

```

1
2 void led_laufflicht (void) {
3     uint8_t i = LED_PORT;
4     i = (i & 0x07) | ((i << 1) & 0xF0);
5     if (i < 0xF0)
6         i |= 0x08;
7     LED_PORT = i;
8 }

```

### 3.3.3. Taster

Listing 3.2: Taster

```

1 #include "Debounce.h"
2
3 void debounce_init (void) {
4     KEY_DDR &= ~ALL_KEYS; // configure key port for input
5     KEY_PORT |= ALL_KEYS; // and turn on pull up resistors
6     TCCR0B = (1 << CS02) | (1 << CS00); // divide by 1024
7     TCNT0 = (uint8_t) (int16_t) -(F_CPU / 1024 * 10 * 10e-3 + 0.5); // preload for 10ms
8     TIMSK0 |= 1 << TOIE0; // enable timer interrupt
9     sei();
10 }
11
12 if (get_key_press(1 << KEY0) || get_key_rpt(1 << KEY0)){
13     lcd_puts("Betrete Menue!\n");

```

```
14 menu_enter(&menu_context, &menu_main);
15 }
```

### 3.3.4. LCD Bibliothek

Die meisten LC-Displays werden auf die selbe Art angesteuert. Hier gibt es fertige Bibliotheken die frei genutzt werden können. Im Projekt wird die von Peter Fleury?

verwendet. Dazu müssen die Dateien lcd.c und lcd.h in das Arbeitsverzeichnis kopiert werden und die Bibliothek mit `#include(lcd.h)` eingebunden werden.

Anschließend müssen noch in der lcd.h die Daten des Display eingegeben werden. Danach kann das Display mit den Befehlen aus Zeile 15-24 aus dem Codebeispiel 3.3 angesteuert werden.

Listing 3.3: Definitionen - LCD(Auszug)

```
1 #define LCD_CONTROLLER_KS0073 0 /**< Use 0 for HD44780 controller,
2     1 for KS0073 controller */
3
4 #define LCD_LINES 4 /**< number of visible lines of the display */
5 #define LCD_DISP_LENGTH 19 /**< visibles characters per line of the display */
6 #define LCD_LINE_LENGTH 0x40 /**< internal line length of the display */
7
8 #define LCD_START_LINE1 0x00 /**< DDRAM address of first char of line 1 */
9 #define LCD_START_LINE2 0x40 /**< DDRAM address of first char of line 2 */
10 #define LCD_START_LINE3 0x14 /**< DDRAM address of first char of line 3 */
11 #define LCD_START_LINE4 0x54 /**< DDRAM address of first char of line 4 */
12
13 #define LCD_WRAP_LINES 1 /**< 0: no wrap, 1: wrap at end of visibile line */
14
15 extern void lcd_init(uint8_t dispAttr);
16 extern void lcd_clrscr(void);
17 extern void lcd_home(void);
18 extern void lcd_gotoxy(uint8_t x, uint8_t y);
19 extern void lcd_putc(char c);
20 extern void lcd_puts(const char *s);
21 extern void lcd_puts_p(const char *progmem_s);
22 extern void lcd_command(uint8_t cmd);
23 extern void lcd_data(uint8_t data);
24 #define lcd_puts_P(__s) lcd_puts_p(PSTR(__s))
```

### 3.3.5. RS-232

*RS-232* ist der Name der am meisten verwendeten seriellen asynchronen Schnittstelle, im Fachjargon auch Übertragungsstandard genannt, um Daten zwischen zwei elektronischen Geräten hin und her zu schicken (im Fachjargon: Datenkommunikation).?

Eine serielle Schnittstelle ist auf dem STK500 vorbereitet. Eine weitere wurde auf einem Steckbrett vorbereitet und mit dem STK500 verbunden.

Um die Schnittstellen im Mikrocontroller nutzen zu können wird in Listing 3.4 - Zeile 2 die `setbaud.h` eingebunden. Zuvor muss noch die Baudrate gesetzt werden. In der Funktion werden die entsprechenden Register im Mikrocontroller gesetzt um die Schnittstellen zu konfigurieren.

Anschließend kann mit der Funktionen `uart_put_string()` eine Zeichenkette versendet werden. Dabei kann mit der Variable *dir* die Schnittstelle ausgewählt werden über die gesendet werden soll.

Listing 3.4: Funktionen - RS-232

```
1 #define BAUD 9600
2 #include <util/setbaud.h>
3
4 void uart_init () {
5     UBRROH = UBRRH_VALUE; // UART 0 – IN (Rapidform Software/Terminal)
6     UBRROL = UBRRL_VALUE;
7     UCSR0C = (3 << UCSZ00);
8     UCSR0B |= (1 << TXEN0); //Transmitter Enabled
9     UCSR0B |= (1 << RXEN0); // UART RX einschalten
10
11     UBRRIH = UBRRH_VALUE; // UART 1 – OUT (Stepper Karte/Drehtisch)
12     UBRRIH = UBRRL_VALUE;
13     UCSR1C = (3 << UCSZ00);
14     UCSR1B |= (1 << TXEN1); //Transmitter Enabled
15     UCSR1B |= (1 << RXEN1); // UART RX einschalten
16 }
17 void uart_put_charater (unsigned char c, int dir) {
18     if (dir == D_RapidForm) { // To Rapidform
19         while (!(UCSR0A & (1 << UDRE0))) {} //warten bis Senden moeglich
20         UDR0 = c; // sende Zeichen
21     }
22     else { // To Stepper
23         while (!(UCSR1A & (1 << UDRE1))) {} //warten bis Senden moeglich
24         UDR1 = c; // sende Zeichen
25     }
```



```
26 }  
27 void uart_put_string (char *s, int dir) {  
28     while (*s) // so lange *s != '\0' also ungleich dem "String-Endezeichen(Terminator)" {  
29         uart_put_charater(*s, dir);  
30         s++;  
31     }  
32 }
```

### 3.3.6. Menü Bibliothek

Der Drehtisch kann Manuell über Taster am Einschub bedient werden. Die Menü Bibliothek gestaltet dies einfach und Komfortabel. Mit den Tasten Zurück, Select, Hoch und Runter lässt sich durch die Einzelnen Menü Punkte Navigieren. **(TODO: REF! EINBINDEN ERLÄUTERN!)** **(TODO: MENÜ BAUM ERSTELLEN!)**

### 3.3.7. Interrupts

Viele Mikrocontroller bieten die Möglichkeit zeitkritische Subroutinen auszuführen. Wenn einer der Interrupts ausgelöst wird, wird das Hauptprogramm unterbrochen und die Entsprechende Interrupt-Service-Routine ausgeführt. Nach Beendigung der ISR wird das Hauptprogramm an der vorherigen Stelle wieder aufgenommen. ISR dürfen nur sehr wenige Befehle enthalten und müssen innerhalb weniger Clock-Cicles abgeschlossen sein.

Interrupts können z.B. der Überlauf eines internen Timer sein, oder ein externens Signal an einem Pin.

Im Projekt werden externe Interrupts, Timer-Überlauf Interrupts und der Watchdog Interrupt genutzt.

#### 3.3.7.1. Endschalter

Die Endschalter sind über die Schrittmotorkarten und eine Brücke in der Steuerung mit der Mikrocontroller Platine Verbunden. Dort sind sie an 2 Interrupt Pins angeschlossen. **(TODO: PINS RAUS SUCHEN!)** Bei einem Flanken Wechsel an den Pins wird ein Interrupt ausgelöst.

Das Code-Listing 3.5 zeigt die ISR für die Endschalter.

Listing 3.5: ISR - Endschalter

```
1 PCMSK3 |= ( 1 << PCINT28 ); // Interrupts definierenPD4 als Interrupt zulassen  
2 PCICR  |= ( 1 << PCIE3 ); // Pin Change Interrupt Control Register – PCIE3 setzen fuer  
   PCINT30
```

### 3. Software

```
3 ISR(PCINT3_vect){ // Endschalter Position erreicht
4   lcd_puts("Positive Endschalter Position Erreicht!");
5   LED_PORT ^= (1 << LED3);
6 }
7 ISR(PCINT2_vect){ // Endschalter Position erreicht
8   lcd_puts("Negative Endschalter Position Erreicht!");
9   LED_PORT ^= (1 << LED3);
10 }
```

#### 3.3.7.2. Watchdog

Der *Watchdog* ist eine Sicherungseinrichtung des Mikrocontroller. In regelmäßigen Abständen wird überprüft ob das Watchdog Bit gesetzt ist und anschließend zurück gesetzt. Das Bit muss innerhalb der voreingestellten Zeit immer wieder neu gesetzt werden. Dies kann mit der `wdt_reset()` Funktion realisiert werden. Ist das Bit nicht gesetzt, wird der Mikrocontroller zurückgesetzt. (**TODO: INVERSE LOGIK?**) Dies geschieht z.B. bei nicht geplanten Endlosschleifen.

Wahlweise kann kurz vor dem Reset noch die Watchdog-ISR durchlaufen werden. Im Projekt wird in der ISR die Fehler LED eingeschaltet und eine Meldung auf dem LC-Display ausgegeben. Siehe hierzu auch Listing 3.6 Zeilen 12-15.

Listing 3.6: Watchdog

```
1 #include <avr/wdt.h>
2
3 void init_WDT(void) {
4   cli();
5   wdt_reset();
6   WDTCSR |= (1 << WDCE) | (1 << WDE);
7   WDTCSR = (1 << WDE) | (1 << WDIE) | (1 << WDP3) | (1 << WDP0); //Watchdog 8s
8   //WDTCSR = 0x0F; //Watchdog Off
9   sei();
10 }
11
12 ISR(WDT_vect){ // Watchdog ISR
13   LED_PORT &= ~(1 << LED4); // LED5 einschalten
14   lcd_puts("Something went \nterribly wrong!\nRebooting!");
15 }
```

#### 3.3.8. Protokoll der Schrittmotorkarte

Tabelle 3.2 zeigt den ASCII Befehlssatz der Schrittmotorkarte.

### 3. Software

_A	Motorstatus liefern
_C n	konstante Geschwindigkeit einstellen
_D n	Bezugswert definieren
_E n	Motorstrom einstellen
_F	Standardeinstellungen aktivieren
_H	Sanfter stop
_I	4-Bit-Eingang lesen
_J jdss	Joystickparameter einstellen
_L n	lokalen Modus aktivieren/beenden
_M n	n Schritte ausführen
_MA n	zu n bewegen
_MC n	mit konstanter Geschwindigkeit bewegen
_MCA n	MA mit konstanter Geschwindigkeit
_MCL n	MC zu Endschalterposition
_ML n	zur Endschalterposition bewegen
_N n	Zeilenvorschub (LF, hex. 0A) einfügen/löschen
_O n	n an 4-Bit-Ausgang senden
_P nnnn	Motorparameter einstellen
_Q	Parameter in EEROM speichern
_R n	Mikroschritteilung einstellen
_RL	Endschalterwerte lesen
_RS	verbleibende Schritte lesen
_S	Nothalt
_T n	Eingang n auslösen
_W	Position anfordern

Tabelle 3.2.: ASCII Befehlssatz R+S Schrittmotorsteuerung  
V9141 [2001]

#### 3.3.9. Manueller Betrieb

#### 3.3.10. Protokolle aus RapidForm

Die Protokolle die RapidForm nutzt um mit den Schrittmotoren zu kommunizieren können leicht mit der Software (**TODO: NAME**) abgehört werden. Dies hat jedoch den Nachteil das RapidForm erst den nächsten Befehl sendet wenn der erste richtig quittiert wurde. Die Befehle die RapidForm erwartet konnten aus den Betriebsanleitungen der Schrittmotoren entnommen werden. Dies war jedoch auch nicht immer leicht.

Durch *Reverse-Engineering* konnten alle Befehle und der Antwort die darauf erwar-

```

1 model "ZETA6104"
2 port "9600" "n81n"
3 init "ECHO0\rCOMEXC0\PSET0\rA8\rV8\r" ""
4 finish "D0 \rGO1\rWAIT(1PE<>1)\rRESET\r" "\r\n>\040\r\n>\040\r\n>\040\r\n>\040"
5 arot "MA1 D%d\rGO1\rWAIT(1PE<>1)\r" "\r\n>\040\r\n>\040\r\n>\040\r\n>\040"
6 home "MA1 D0 \rGO1\rWAIT(1PE<>1)\r" "\r\n>\040\r\n>\040\r\n>\040\r\n>\040"
7 stop "!S\r" "\r\n>\040"
8 step "0.00008" "-4500000" "4500000"
9 timeout "60"
10 firsttimeout "10"

```

Das Herzstück der Übersetzungs Software. Für jedes Protokoll muss eine eigene Auswerte Logik geschrieben werden.

Zuerst wird eine Auswerte Logik geschrieben die ankommende Strings mit einem übergebenen Array vergleicht und die Arrayposition übergibt. Diese numerische Rückgabe kann dann mittels einer switch/case Struktur ausgewertet werden.

```

1 int FindStringInArray (const char* pInput, const char* pOptions[], int cmp_length) {
2     int n = -1;
3     while (pOptions[++n]) { //Array durchlaufen bis 0 terminiert
4         //Wenn pInput == pOptions dann gib Array Position zurueck
5         if (!strcmp(pInput, pOptions[n], cmp_length)) return n;
6     }
7     return 99;
8 }

```

Nicht jede Software unterstützt alle Protokolle. Daher wurden alle Protokolle realisiert. Um automatisch fest zu stellen welches Protokoll verwendet wird, wird der erste ankommende Befehl gegen die Initialisierungssequenz der Protokolle geprüft und in einer globalen Variable gespeichert.

Listing 3.9: Funktion - switchMotor

```
1 int    switch_Motor      (char * str_rx) {
2     const char* pOptions[] = {
3         "@01",           // 0 - Isel
4         "Q:",            // 1 - CSG
5         "ECHO0",         // 2 - Zeta
6         "!!Terminal",    // 3 - Terminal ansteuerung!
7         0 };
8     switch (FindStringInArray(str_rx, pOptions, 3)) {
9     case 0:              // 0 - ISEL
10        return M_ISEL;
11        break;
12     case 1:              // 1 - CSG
13        return M_CSG;
14        break;
15     case 2:              // 2 - Zeta
16        return M_ZETA;
17        break;
18     case 3:              // 3 - Terminal ansteuerung
19        return M_TERMINAL;
20        break;
21     default:
22        return M_UNK;
23    }
24 }
```

### 3.3.11.3. Zeta

Zeta

### 3.3.11.4. Isel

**(TODO: SPLITTEN UND JEDEN ABSCHNITT EINZELN ERKLÄREN!)** Auch hier wird wieder ein Array mit möglichen ankommenden Befehlen definiert und der ankommende String gegen dieses Array geprüft. Kommt z.B. die Sequenz "@01än, wird case 3 ausgewählt. Das Display zeigt Init an und sendet die Antwort die RapidForm erwartet zurück.

Die Sequenz "@0R&steht für eine Statusabfrage. Es wird eine Statusabfrage an die Schrittmotorkarte gesendet und der Status an RapidForm zurück gemeldet.

Listing 3.10: Übersetzungs Logik - Isel

```
1 void    switch_Isel      (char * str_rx) {
```

### 3. Software

```
2     const char* pOptions[] = {
3         "XXXXXXX", // 0 - Reserve
4         "!CLS",    // 1 - LC-Display löschen
5         "Test",    // 2 - Test
6         "@01",     // 3 - Achse auswählen
7         "@0R",     // 4 - Status abfrage
8         "@0M",     // 5 - Gehe zu Position MX , +600
9         0 };
10
11     int Ret_Val = FindStringInArray(str_rx, pOptions, 3);
12     switch (Ret_Val) {
13     case 0:        // 0 - Reserve
14     case 1:        // 1 - LC-Display löschen
15     case 2:        // 2 - Test
16     case 3:        // 3 - Achse auswählen
17         ms_spin(10);
18         lcd_puts("Init");
19         uart_put_string("0\r\n", D_RapidForm);
20         break;
21     case 4:        // 4 - Status abfrage
22         lcd_puts("Statusabfrage: \n");
23         uart_put_string("A\n", D_Stepper);
24         ms_spin(50);
25         if ((UCSR1A & (1 << RXC1)))
26             uart_rx(D_Stepper);
27         if (!strcmp(str_rx, "0#"))
28             uart_put_string("0\r\n", D_RapidForm);
29         else {
30             lcd_puts("Fehlgeschlagen \n");
31             uart_put_string("1\r\n", D_RapidForm);
32         }
33         break;
34     case 5:        // 5 - Gehe zu Position MX , +600
35         ms_spin(10);
36         char Position[33], Winkel[6];
37         memset(Position, '\0', 33);
38         memset(Winkel, '\0', 6);
39         String_zerlegen_Isel(str_rx, Position, Winkel);
40         char Move_To[40];
41         memset(Move_To, '\0', 40);
42         Move_To[0] = 'M';
43         Move_To[1] = 'A';
44         Move_To[2] = ' ';
45         Move_To[3] = '\0';
46         strcat(Move_To, Position);
```

3. Software

```
47     strcat(Move_To, "\n");
48     lcd_puts("Pos:");
49     lcd_puts(Move_To);
50
51     uart_put_string(Move_To, D_Stepper);
52     ms_spin(50);
53     if ((UCSR1A & (1 << RXC1)))
54         uart_rx(D_Stepper);
55     else {
56         break;
57     }
58
59     uart_put_string("A\n", D_Stepper);
60     ms_spin(50);
61     if ((UCSR1A & (1 << RXC1)))
62         uart_rx(D_Stepper);
63     else {
64         lcd_puts("Keine Bewegung!\n");
65     }
66
67     while (!strcmp(str_rx, "1#")){
68         uart_put_string("A\n", D_Stepper);
69         ms_spin(50);
70         if ((UCSR1A & (1 << RXC1))) {
71             uart_rx(D_Stepper);
72             lcd_clrscr();
73             lcd_puts("Gehe zu Winkel: ");
74             lcd_puts(Winkel);
75             lcd_puts("\n");
76         }
77         else {
78             lcd_puts("Keine Antwort\n");
79         }
80         wdt_reset();
81     }
82     lcd_puts("Winkel: ");
83     lcd_puts(Winkel);
84     lcd_puts(" Erreicht\n");
85     uart_put_string("0\r\n", D_RapidForm);
86     break;
87 default:
88     lcd_puts(str_rx);
89 }
90 }
```

#### **3.3.11.5. Weitere**

Es sind weitere Protokolle ansatzweise implementiert. Diese werden im allgemeinen jedoch nicht benötigt da nur das Isel und das Zeta Protokoll ordentlich in der Clientsoftware umgesetzt sind.



## 4. Fazit und Zukunft

### 4.1. Fazit

**(TODO: FAZIT SCHREIBEN!)**

## Literaturverzeichnis

### Atm 2011

ATMEL (Hrsg.): *ATmega164A/PA/324A/PA/644A/PA/1284/P Complete*. San Jose, CA 95131, USA: Atmel, 06 2011 2.9

### Atmel 2012a

ATMEL: *ATmega324A- Atmel Corporation*. <http://www.atmel.com/devices/ATMEGA324A.aspx>. Version: 2012. – [Online; Stand 11. Februar 2012] 2.3

### Atmel 2012b

ATMEL: *ATmega8515- Atmel Corporation*. <http://www.atmel.com/devices/ATMEGA8515.aspx>. Version: 2012. – [Online; Stand 11. Februar 2012] 2.3

### V9141 2001

RS (Hrsg.): *Schrittmotor-Platine mit integriertem Treiber*. Mörfelden-Walldorf: RS, 03 2001 3.2

### Wikipedia 2012a

WIKIPEDIA: *Daisy chain* — *Wikipedia, Die freie Enzyklopädie*. [http://de.wikipedia.org/w/index.php?title=Daisy\\_chain&oldid=98475104](http://de.wikipedia.org/w/index.php?title=Daisy_chain&oldid=98475104). Version: 2012. – [Online; Stand 11. Februar 2012] 2

### Wikipedia 2012b

WIKIPEDIA: *Stiftleiste* — *Wikipedia, Die freie Enzyklopädie*. <http://de.wikipedia.org/w/index.php?title=Stiftleiste&oldid=99052435>. Version: 2012. – [Online; Stand 11. Februar 2012]

## Eidesstattliche Erklärung

Hiermit versichere ich, dass ich den vorliegenden Bericht:

*Übersetzen von Schrittmotorprotokollen*  
*Entwurf eines Hardwareübersetzers*

selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe.

Remagen, den 22. Februar 2012



JOHANNES DIELMANN

## A. Anhang

## **A.1. Schritt für Schritt Anleitung**

Eine Schritt für Schritt Anleitung zum vollständigen Scannen und exportieren eines 3D-Objektes.

## A.2. Technische Daten VI-910

Die Technischen Daten beziehen sich auf den VI-910. Dies ist das Nachfolgemodell. Die meisten Daten sollten jedoch ähnlich sein.

Tabelle A.1.: Technische Daten - VI-910

Modellbezeichnung	Optischer 3D-Scanner VI-910
Messverfahren	Triangulation durch Lichtschnittverfahren
Autofokus	Autofokus auf Objektoberfläche (Kontrastverfahren); aktiver AF
Objektive (wechselbar)	TELE Brennweite f=25mm MITTEL: Brennweite f=14 mm WEIT: Brennweite f=8mm
Messabstand	0,6 bis 2,5m (2m für WIDE-Objektiv)
Optimaler Messabstand	0,6 bis 1,2m
Laserklasse	Class 2 (IEC60825-1), Class 1 (FDA)
Laser-Scanverfahren	Galvanisch-angetriebener Drehspiegel
Messbereich in X-Richtung (anhängig vom Anstand)	111 bis 463mm (TELE), 198 bis 823mm (MITTEL), 359 bis 1.196mm (WEIT)
Messbereich in Y-Richtung (abhängig vom Abstand)	83 bis 347mm (TELE), 148 bis 618mm (MITTEL), 269 bis 897mm (WEIT)
Messbereich in Z-Richtung (abhängig vom Abstand)	40 bis 500mm (TELE), 70 bis 800mm (MITTEL), 110 bis 750mm (WEIT/Modus FINE)
Genauigkeit	X: $\pm 0,22$ mm, Y: $\pm 0,16$ mm, Z: $\pm 0,10$ mm zur Z-Referenzebene (Bedingungen: TELE/Modus FINE , Konica Minolta Standard)
Aufnahmezeit	0,3s (Modus FAST), 2,5s (Modus FINE), 0,5s (COLOR)
Übertragungszeit zum Host-Computer	ca. 1s (Modus FAST) oder 1,5s (Modus FINE)
Scanumgebung, Beleuchtungsbedingungen	500 lx oder geringer

A. Anhang

Aufnahmeeinheit	3D-Daten: 1/3CCD-Bildsensor (340.000 Pixel) Farbdaten: Zusammen mit 3D-Daten (Farbtrennung durch Drehfilter)
Anzahl aufgenommener Punkte	3D-Daten: 307.000 (Modus FINE), 76.800 (Modus FAST) Farbdaten: $640 \times 480 \times 24$ Bit Farbtiefe
Ausgabeformat	3D-Daten: Konica Minolta Format, (STL, DXF, OBJ, ASCII-Punkte, VRML; Konvertierung in 3D-Daten durch Polygon Editing-Software / Standardzubehör) Farbdaten: RGB 24-Bit Rasterscan-Daten
Speichermedium	Compact Flash Memory Card (128MB)
Dateigrößen	3D- und Farbdaten (kombiniert): 1,6MB (Modus FAST) pro Datensatz, 3,6MB (Modus FINE) pro Datensatz
Monitor	5,7LCD ( $320 \times 240$ Pixel)
Datenschnittstelle	SCSI II (DMA-Synchronübertragung)
Stromversorgung	Normale Wechselstromversorgung, 100V bis 240 V (50 oder 60 Hz), Nennstrom 0,6 A (bei 100 V)
Abmessungen (B x H x T)	$213 \times 413 \times 271$ mm
Gewicht	ca. 11kg
Zulässige Umgebungsbedingungen (Betrieb)	10 bis 40°C; relative Luftfeuchtigkeit 65% oder niedriger (keine Kondensation)
Zulässige Umgebungsbedingungen (Lagerung)	-10 bis 50°C, relative Luftfeuchtigkeit 85% oder niedriger (bei 35°C, keine Kondensation)

### A.3. Verwendete Software

Hier ist die verwendete Software aufgelistet. Soweit es möglich war, wurden Open-Source-Programme eingesetzt. **(TODO: ÜBERARBEITEN!!!)**

- **RapidForm2004**  
INUS Technology, Inc.  
Website: <http://www.rapidform.com>
- **AVRStudio 5**  
Atmel  
Website: <http://www.atmel.com/>
- **Eclipse**  
Eclipse mit CDT und AVRPlugin  
Website: <http://www.eclipse.org/>
- **AVRDude**  
Prorammer  
**(TODO: WEITERE?!)**
- **Blender**
- **Texmaker**
- **LaTeX**
- **Inkscape**

### A.4. Verwendete Hardware

- **VI-900**  
Minolta  
Website: <http://www.konicaminolta.eu/de/messinstrumente/produkte/3d-messtechnik/beruehrungsloser-3d-scanner/vi-910/einfuehrung.html>



*A. Anhang*

---

- **ATMega 324A**

Atmel

Website: <http://www.atmel.com/devices/ATMEGA324A.aspx>

- **STK500**

Atmel

Website: <http://www.atmel.com/tools/STK500.aspx>

- **AVRISP mkII** Atmel

Website: <http://www.atmel.com/tools/AVRISPMKII.aspx>