

**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>II</b>
<b>1. Einleitung</b>	<b>1</b>
1.1. Ziel der Arbeit anhand eines Beispiels . . . . .	2
1.1.1. Vorhandene Komponenten . . . . .	2
1.1.2. Vorgaben . . . . .	2
1.1.3. Zielvorgabe . . . . .	3
1.2. Motivation . . . . .	3
<b>2. Vorstellung der vorhandenen Hardware</b>	<b>5</b>
2.1. Computer . . . . .	5
2.2. 3D-Laserscanner VI-900 . . . . .	5
2.2.1. Lasertriangulator Prinzip . . . . .	6
2.3. Ansteuerung für den Drehtisch . . . . .	6
2.3.1. Drehtisch . . . . .	6
2.3.2. Spannungsversorgung . . . . .	7
2.3.3. Schrittmotoren . . . . .	7
2.3.4. Schrittmotorkarten . . . . .	8
2.3.5. Motorverkabelung . . . . .	8
2.3.6. Endschalter . . . . .	8
2.4. Mikrocontroller . . . . .	8
2.4.1. Entwicklerboard STK500 . . . . .	9
2.4.2. AVRISP mkII . . . . .	10
2.4.3. MAX232 . . . . .	11
<b>3. Vorstellung der vorhandenen Software</b>	<b>12</b>
3.1. RapidForm2004 . . . . .	12
3.2. Entwicklungsumgebung . . . . .	12
3.3. Terminalprogramme . . . . .	12

<b>4. Arbeitsablauf</b>	<b>13</b>
4.1. Erste Schritte	13
4.1.1. Taster nutzbar machen	14
4.1.2. LEDs ansteuern	15
4.1.3. LCD ansteuern	15
4.1.4. Serielle Schnittstelle ansteuern	16
4.2. Protokolle	19
4.3. Kommunikation mit der Schrittmotorsteuerung	20
4.3.1. Befehle senden	20
4.3.2. Befehle Empfangen und auswerten	22
4.4. Verbesserungen an der vorhandenen Hardware	22
4.4.1. Spannungsversorgung	22
4.4.2. Inbetriebnahme der zweiten Schrittmotorkarte	23
4.4.3. Motorverkabelung	23
4.4.4. Endschalter	24
4.4.5. Zweite serielle Schnittstelle	25
4.5. Kommunikation mit RapidForm2004	25
4.5.1. Befehle empfangen	25
4.5.2. Befehle verarbeiten	27
<b>5. Parkplatz</b>	<b>28</b>
5.0.3. Übersetzungs Logik	28
5.0.3.1. FindStringInArray Funktion	28
5.0.3.2. Automatische Protokollwahl	28
5.0.3.3. Zeta	29
5.0.3.4. Isel	29
5.0.3.5. Weitere	32
5.0.4. Interrupts	32
5.0.4.1. Endschalter	32
5.0.4.2. Watchdog	33
<b>6. Probleme und Lösungen</b>	<b>34</b>
6.1. Entwicklungsumgebungen	34
6.1.1. AVR Studio 5	34
6.1.2. Eclipse	34
6.2. Fuses	35
6.3. Platinenlayout	36

<b>7. Fazit und Zukunft</b>	<b>37</b>
7.1. Fazit . . . . .	37
<b>Eidesstattliche Erklärung</b>	<b>40</b>
<b>A. Anhang</b>	<b>i</b>
A.1. Schritt für Schritt Anleitung . . . . .	ii
A.2. Protokoll der Schrittmotorkarte . . . . .	iii
A.3. Protokolle aus RapidForm2004 . . . . .	iv
A.4. Technische Daten VI-910 . . . . .	vii
A.5. Verwendete Hardware . . . . .	ix
A.6. Verwendete Software . . . . .	ix

## Abkürzungsverzeichnis

ADC .....	Analog Digital Convertor
ASCII .....	American Standard Code for Information Interchange
AVRISP .....	AVR in System Programmer
CAD .....	Computer Aided Design
CF .....	Compact Flash
CPU .....	Central Processing Unit
DAC .....	Digital Analog Convertor
DIL .....	Dual in Line Package
IC .....	Integrated Cuircuit
ISR .....	Interrupt Service Routine
LCD .....	Liquid Crystal Display
MHz .....	Megahertz
RS .....	Recommended Standard
SCSI .....	Small Computer System Interface
USART .....	Universal Asynchronous Receiver Transmitter
USB .....	Universal Serial Bus
V .....	Volt

## Abbildungsverzeichnis

1.1. Blick auf den Arbeitsaufbau . . . . .	2
2.1. VI-900 - 3D-Scanner . . . . .	5
2.2. Prinzip: Laser-Triangulation . . . . .	6
2.3. Ansteuerung im 19"-Rack . . . . .	7
2.4. Drehtisch . . . . .	7
2.5. Block Diagram: Mikrocontroller . . . . .	9
2.6. Schema: STK500 . . . . .	10
4.1. Stromverbinder - Y-Kabel . . . . .	22
4.2. Motor- und Endschalterverkabelung . . . . .	23
4.3. Motor- und Endschalterverkabelung . . . . .	24
4.4. Schema: MAX232 . . . . .	25

# Tabellenverzeichnis

4.1. Motor- und Endschalterverkabelung . . . . .	23
6.1. Fuses . . . . .	35
A.1. ASCII Befehlssatz R+S Schrittmotorsteuerung . . . . .	iii
A.2. Technische Daten - VI-910 . . . . .	vii

## Codeverzeichnis

4.1. Taster . . . . .	14
4.2. LEDs . . . . .	15
4.3. lcd.h (Auszug) . . . . .	16
4.4. RS-232 . . . . .	17
4.5. Protokoll aus Rapidform: Zeta . . . . .	19
4.6. Menü . . . . .	20
4.7. Menü Baum . . . . .	21
4.8. RS-232 Empfang . . . . .	26
5.1. Funktion: FindStringInArray . . . . .	28
5.2. Funktion: switchMotor . . . . .	28
5.3. Übersetzungs Logik: Isel . . . . .	30
5.4. ISR: Endschalter . . . . .	32
5.5. Watchdog . . . . .	33
A.1. RapidForm2004 Protokolle Empfang . . . . .	iv



# 1. Einleitung

(TODO: HIER KOMMT EIN EINLEITENDER TEXT HIN!)

## 1.1. Ziel der Arbeit anhand eines Beispiels

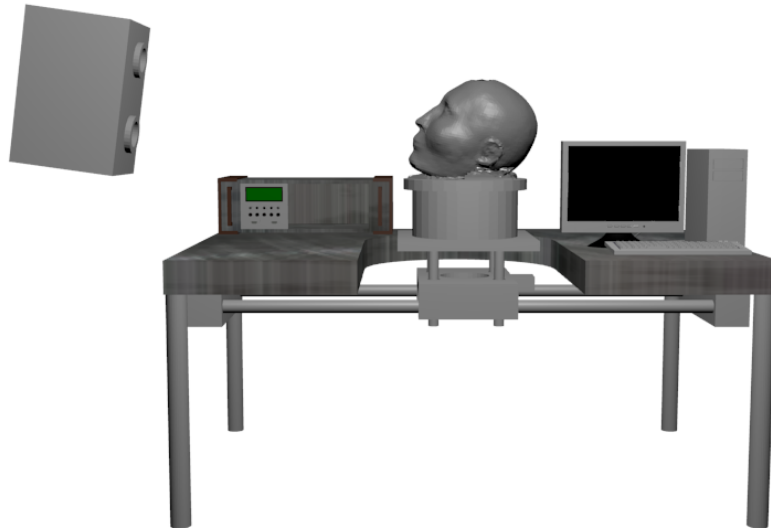


Abbildung 1.1.: Blick auf den Arbeitsaufbau

### 1.1.1. Vorhandene Komponenten

1. Computer mit Erfassungssoftware RapidForm2004
2. 3D-Laserscanner VI-900
3. Arbeitstisch mit integriertem Drehtisch
4. 19"-Rack mit 2 Schrittmotorkarten
5. Mikrocontroller

### 1.1.2. Vorgaben

Im konkreten Beispiel ging es um die automatische 3D-Erfassung eines Schädelmodells. In der Ausgangssituation war es zwar möglich, mit der Erfassungssoftware RapidForm2004 und dem 3D-Laserscanner VI-900 einen Schädel aus einer Richtung zu erfassen. Die Kommunikation zwischen dem VI-900 und der Erfassungssoftware funktionierte. Die Drehtischsteuerung, welche den Drehtisch nach den Vorgaben der Erfassungssoftware des Computers drehen sollte, war nicht in das System eingebunden. Dies war ein Problem der verschiedenen Befehlssätze.

## 1. Einleitung

---

Aufbau eines Übersetzers, basierend auf einem Mikrocontroller. Der Übersetzer sollte ein LC-Display, mehrere Taster, mehrere LEDs und zwei serielle Schnittstellen enthalten. Die Höhenverstellung des Drehtisches sollte genutzt werden und die zu Beginn noch nicht funktionierenden Endschalter sollten die vorgesehene Funktion erfüllen.

### 1.1.3. Zielvorgabe

1. Der VI-900 erstellt eine Aufnahme des Schädels.
2. Der VI-900 sendet die Aufnahme an die Erfassungssoftware im Computer.
3. Die Erfassungssoftware im Computer sendet nach der Speicherung der Aufnahme den Befehl zum Drehen des Drehtisches an die Drehtischsteuerung.
4. Die Drehtischsteuerung dreht den Tisch um die gewünschte Gradzahl.
5. Die Drehtischsteuerung meldet die erfolgreiche Rotation mit Hilfe des zu entwickelnden Übersetzers an die Erfassungssoftware im Computer.
6. Die Erfassungssoftware im Computer sendet erneut einen Aufnahmebefehl an den VI-900.

Nachdem ein kompletter Aufnahmesatz im Computer eingespeichert ist, kann das 3D-Modell als CAD-Datei exportiert werden. Das Erstellen eines kompletten Aufnahmesatzes soll auch für Laien leicht möglich sein.

## 1.2. Motivation

Die 3D-Lasererfassung bietet zahlreiche Anwendungsgebiete.

- Qualitätskontrolle und Bauteilprüfung für Guss- und Spritzgusstechnik
- Erstellung von Finite-Elemente-Daten für Blechteile im Karosseriebau usw. in Verbindung mit Bauteilanalyse
- Erstellung von 3D-Daten zur Kontrolle von Zubehörteilen und anderen Zukaufteilen, für die keine 3D-CAD-Daten verfügbar sind (z.B. Reverse Engineering)
- Umwandlung von Daten aus der Zahnmedizin und der plastischen Chirurgie in ein Datenbankformat

## 1. Einleitung

---

- Erstellen von Konstruktionsdaten aus Mustern und Verzugs-Prüfung an mechanischen Bauteilen
- Integration in Rapid-Prototyping-Systemen für die Erstellung von Mustern aus Kunststoff
- Vergleich von Eigenprodukten mit Produkten des Mitbewerbs und Umwandlung der erfassten Daten in ein Datenbankformat
- Archivierung in Museen und Forschungseinrichtungen
- Erstellung von Daten für die CAE-Analyse
- 3D-Daten von beliebigen Objekten für unterschiedlichste Forschungszwecke
- Informations- und Kommunikationstechnik, Mimikanalyse, Muskelbewegungsanalyse, Robot-Vision und Wachstumskontrolle landwirtschaftlicher Erzeugnisse
- Unterschiedliche Anwendungen in der Filmindustrie

[Minolta \[2012\]](#) Im konkreten Fall soll nun die Erstellung von 3D-Daten eines vorhandenen Objektes (*Reverse Engineering*) genutzt werden.

Es wird nun mit einer Kombination aus dem 3D-Laserscanner, dem Drehtisch und der dazugehörigen Erfassungsssoftware ein 3D-Modell erfasst. Dieses kann dann 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 3D-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.

## 2. Vorstellung der vorhandenen Hardware

### 2.1. Computer

Zur Verfügung steht ein IBM kompatibler x86 Standard PC mit einer SCSI-Schnittstelle und zwei seriellen RS-232-Schnittstellen. Von denen jedoch nur eine genutzt wird.

### 2.2. 3D-Laserscanner VI-900



Abbildung 2.1.: VI-900 - 3D-Scanner

Der 3D-Laserscanner *VI-900* der Firma Minolta besteht, wie auf Abbildung 2.1 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.

### 2.2.1. Lasertriangulator Prinzip

Ein Lasertriangulator, wie in Abbildung 2.2 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 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. [Minolta \[2012\]](#) Die Technischen Daten befinden sich im Anhang in Tabelle [A.2](#)

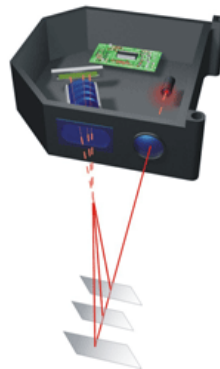


Abbildung 2.2.: Prinzip: Laser-Triangulation

## 2.3. Ansteuerung für den Drehtisch

Die Ansteuerung für den Drehtisch ist in einem 19"-Rack verbaut (siehe Abbildung [2.3](#)).

### 2.3.1. Drehtisch

Der Drehtisch (siehe Abbildung [1.1](#)) 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 Schie-

## 2. Vorstellung der vorhanden Hardware

---

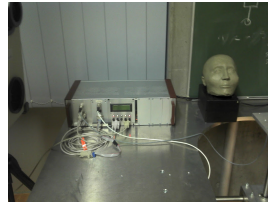


Abbildung 2.3.: Ansteuerung im 19"-Rack

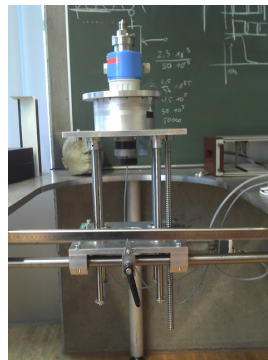


Abbildung 2.4.: Drehtisch

nensystem gelagert. Mit dem Schienensystem kann der Drehtisch in der Vertikalen positioniert werden. Mit einem Schrittmotor lässt sich der Drehtisch 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.

### 2.3.2. Spannungsversorgung

Die Schrittmotorkarten werden von einem PC-Netzteil gespeist. Die *Logikbausteine* werden mit 5V gespeist, zusätzlich werden die Schrittmotorkarten mit 12V für die Schrittmotoren gespeist. Die Kabel sind direkt an die **(TODO: VERBINDUNGS-LEISTEN)** gelötet.

Dies verhindert das einfache Ausbauen der Spannungsversorgung und die einfache Erweiterung um neue Einschubkarten.

### 2.3.3. Schrittmotoren

**(TODO: MOTOREN BESCHREIBEN! TECHNISCHE DATEN!)**

**(TODO: SCHRITTE, SPANNUNGEN. VERDRAHTUNG.)**

#### 2.3.4. Schrittmotorkarten

Die Ansteuerung für die Schrittmotoren sind als 19"-Einschübe 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 A.2. Es können zwei oder mehr Karten als *Daisy-Chain*<sup>2</sup> in Reihe geschaltet werden.

#### 2.3.5. Motorverkabelung

Die Schrittmotoren benötigen ein mindestens 4-adriges Kabel. Das Kabel für den Schrittmotor, der für die Rotation zuständig ist, war bereits gefertigt. Ein Kabel zwischen Schrittmotor und Schrittmotorkarte zur Höhenverstellung und für die Endschalter ist nicht vorhanden.

#### 2.3.6. 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 sind bereits induktive Endschalter der Firma *Pepperl+Fuchs* verbaut. Am Drehtisch ist ein Metallstutzen von ungenügender Länge angebracht. Durch die ungenügende Länge des Metallstutzen würde der Endschalter nicht rechtzeitig ausgelöst werden und der Aufbau des Drehtisches würde beschädigt.

### 2.4. 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 Mikrocontrollers bietet das Blockdiagramm in Abbildung 2.5.

---

<sup>1</sup>Der American Standard Code for Information Interchange (ASCII, alternativ US-ASCII, oft [æski] ausgesprochen) ist eine 7-Bit-Zeichenkodierung [Wikipedia \[2012a\]](#)

<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 \[2012b\]](#)



## 2. Vorstellung der vorhandenen Hardware

Für unterschiedliche Aufgaben sind unterschiedliche Mikrocontroller geeignet.

Es steht ein ATmega8515 [Corporation \[2012b\]](#) im DIL-Gehäuse zur Verfügung. Dieser hat 8 Kbyte Flash, drei externe Interrupts, eine serielle Schnittstelle und kann mit bis zu 16 MHz betrieben werden. Dieser ist geeignet um sich in die Programmierung mit *C* einzufinden und eine serielle Schnittstelle anzusteuern.

Für dieses Projekt sind jedoch zwei serielle Schnittstellen nötig. Der ATmega 324A [Corporation \[2012a\]](#) würde diese Voraussetzungen erfüllen, ist jedoch nicht vorhanden. Er ist dem ATmega 8515 recht ähnlich, bietet jedoch die benötigten zwei seriellen Schnittstellen. Des Weiteren hat er 32 Kbyte Flash.

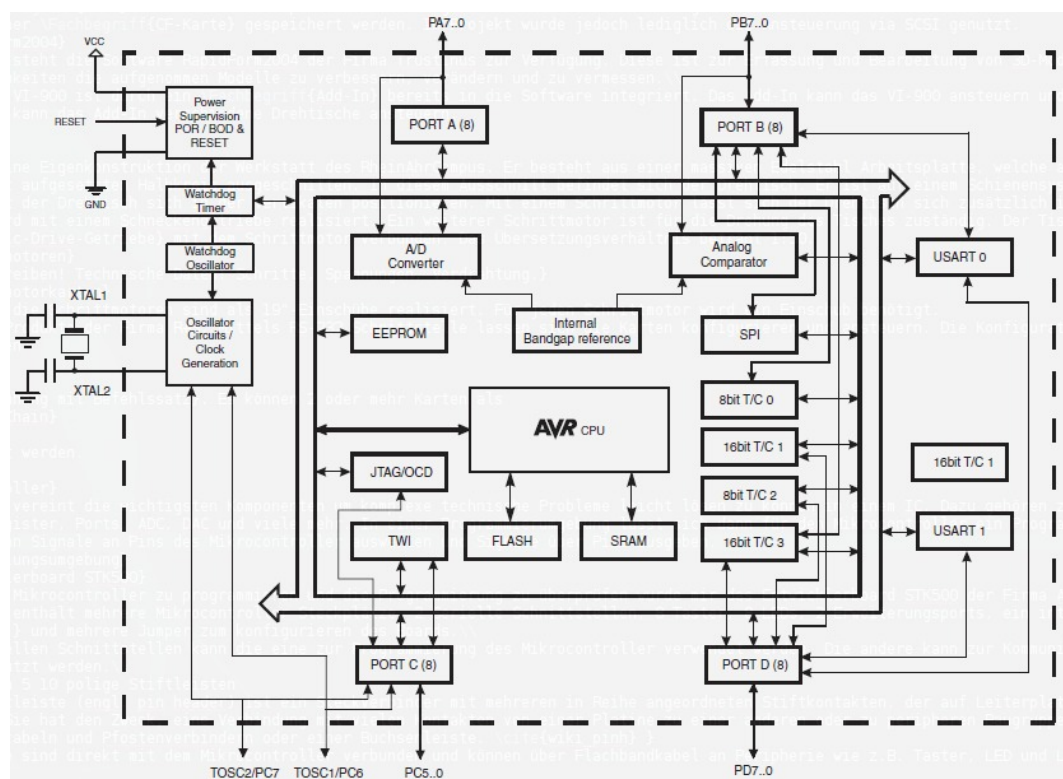


Abbildung 2.5.: Block Diagram: Mikrocontroller  
[Atm 2011]

### 2.4.1. Entwicklerboard STK500

Um den Mikrocontroller zu programmieren und die Programmierung zu überprüfen, soll das *Entwicklerboard STK500* (siehe Abbildung 2.6) der Firma Atmel verwendet werden. Das Board enthält mehrere Mikrocontroller-Steckplätze, 2 serielle Schnitt-

## 2. Vorstellung der vorhandenen Hardware

stellen, 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 Mikrocontrollers 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 **(TODO: PERIPHERIE)** wie z.B. Taster, LED, LC-Displays oder seriellen Schnittstellen verbunden werden.

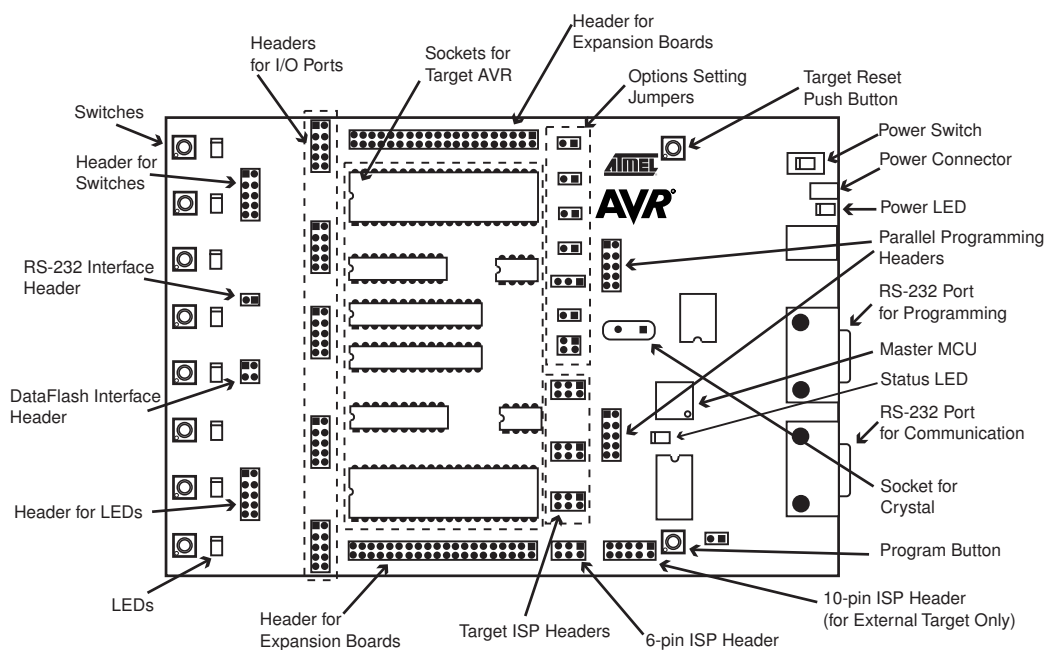


Abbildung 2.6.: Schema: STK500  
[Atm 2003]

### 2.4.2. AVRISP mkII

Das AVRISP mkII ist ein USB-basiertes *In-System-Programmierersystem*. Dieses kann anstelle des RS-232 basierten Programmersystem des STK500 verwendet werden. Die Übertragungsgeschwindigkeit des AVRISP mkII ist wesentlich höher als die der Seriellen Schnittstelle. Der AVRISP mkII lässt sich einfach an den Programmierport, eine 6-Polige Stiftleiste, des STK500 anschließen.

<sup>3</sup>In System Programmer

### 2.4.3. MAX232

Um die Serielle Schnittstelle am Mikrocontroller nutzen zu können müssen die Spannungspegel auf die des RS-232 Standard gewandelt werden. Dazu befindet sich auf dem STK500 der *Pegelwandler* MAX232. Dieser wandelt die Spannungspegel des Mikrocontroller (typ. 0 V – 5 V *TTL*<sup>4</sup>) auf die Spannungspegel des RS-232 Standards (typ. -12 V – +12 V).

---

<sup>4</sup>Transistor-Transistor-Logik

## 3. Vorstellung der vorhandenen Software

### 3.1. RapidForm2004

Zur Erfassung von 3D-Modellen am PC steht die Software RapidForm2004 der Firma INUS Technology Inc. zur Verfügung. Diese ist zur Erfassung und Bearbeitung von 3D-Modellen gedacht. Sie bietet umfangreiche Möglichkeiten die aufgenommenen Modelle zu verbessern, zu verändern, zu vermessen und in verschiedene Formate zu exportieren.

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

### 3.2. Entwicklungsumgebung

Die von Atmel bereitgestellte Entwicklungsumgebung besteht aus einem Editor, dem Compiler und einer Programmiersoftware. Der Editor bietet Komfortfunktionen wie *Syntaxhighlighting*, Autovervollständigung und Projektmanagement.

### 3.3. Terminalprogramme

Als Terminalprogramm zur Kommunikation zwischen Datengeräten über die serielle Schnittstelle steht das Programm "Hypterminal" der Firma Microsoft zur Verfügung.

## 4. Arbeitsablauf

(TODO: REGISTER BESCHREIBEN!)

### Hinweis

Die Schritte in diesem Kapitel folgen dem chronologischen Aufbau des Projektes.

Die ersten Arbeitsschritte 4.1–4.3.1 dienen dazu die Programmierung des Mikrocontrollers zu beschreiben.

Um den Mikrocontroller nutzen zu können ist es dafür notwendig den Mikrocontroller mit einem Programm zu beschreiben(*Flashen*). Diese Programmierung lässt sich am einfachsten in einer Entwicklungsumgebung(siehe Kapitel 3.2) schreiben.

Das Ziel des Programms besteht darin Spannungen an den Pins des Mikrocontrollers auszugeben oder auszuwerten und somit Komponenten wie LEDs, LC-Display, Taster und serielle Schnittstellen anzusteuern. Zuerst werden die ersten Schritte mit dem Mikrocontroller beschrieben. Diese dienen dazu, benötigte Grundfunktionalitäten bereitzustellen.

In den Arbeitsschritten ??–?? geht es um die Entwicklung und Verbesserung von Hardware.

Diese wird so erweitert, dass sie den Vorgaben entspricht.

In den Arbeitsschritten ?? wird beschrieben wie die Software so erweitert wird, dass auch sie allen Vorgaben genügt.

Im letzten Arbeitsschritt ?? wird dann noch die Entwicklung des Platinenlayouts und des Einschubes erklärt. (TODO: UMSTELLUNGSPROBLEME)

### 4.1. Erste Schritte

Im ersten Schritt ging es darum, den Drehtisch mithilfe eines Mikrocontrollers um 90° zu drehen.

Der Mikrocontroller befindet sich auf dem STK 500(siehe Kapitel 2.4.1). Dieses bietet grundlegenden Funktionalitäten wie Taster, LEDs, eine Programmierschnittstelle

#### 4. Arbeitsablauf

und eine serielle Schnittstelle. Um die Komponenten sinnvoll im Mikrocontroller nutzen zu können müssen dafür Funktionalitäten wie z.B. Bibliotheken bereit gestellt werden oder Register initialisiert werden.

Die folgenden Kapitel beschreiben dieses Bereitstellen der Funktionalitäten.

##### Hinweis

Die Codelistings in diesem Kapitel sind thematisch zusammen gefasst und gekürzt um die Lesbarkeit und das Verständnis zu gewährleisten. Ein komplettes Codelisting des Hauptprogramms befindet sich im Anhang ?? . Der komplette Code, mit allen Bibliotheken, liegt dem Praxisbericht als CD oder Archiv bei.

#### 4.1.1. Taster nutzbar machen

Um die Taster des STK500 im Mikrocontroller nutzen zu können müssen diese entprellt werden.

Im ersten Schritt verband ich die Stiftleiste des PortA mit der Stiftleiste für die Taster.

Das Entprellen der Taster realisierte ich softwareseitig in dem ich die Bibliothek [Dannegger \[2012\]](#) von Peter Dannegger einband.

Diese habe ich heruntergeladen und in das Projektverzeichnis entpackt.

Mit Zeile 1 des Codelisting 4.1 wird die Bibliothek in das Programm eingebunden.

Die Zeilen 3-10 spiegeln die Funktion zum Initialisieren der Bibliothek wieder.

Nach dem Einbinden der Bibliothek war es möglich Funktionen wie z.B. `get_key_press()` zu nutzen um den Status der Taster prellfrei auszulesen und diese Information für Entscheidungen im Programm zu verwenden.

Listing 4.1: 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 10
8         ms
9     TIMSK0 |= 1 << TOIE0; // enable timer interrupt
10     sei();
11 }
```

#### 4. Arbeitsablauf

```
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 }
```

##### 4.1.2. LEDs ansteuern

Die LEDs sollen im Programmablauf nutzbar sein.

Dazu verband ich zuerst die Stiftleiste von *PortB* mit der LED Stiftleiste.

Um LEDs an *PortB* betreiben zu können musste ich die Pins im *Register DDRB* als Ausgänge definieren. Dies geschieht in Zeile 3 des Codelisting 4.2. Die Bibliothek zum Entprellen der Taster nutzte die Variablen *LED\_DDR* und *LED\_PORT*. Auch ich nutzte diese Variablen um auf die Register zuzugreifen, da dies eine bessere Übersicht gewährleistet.

Die Werte im 8-Bit Register *LED\_PORT* spiegeln die Spannungen an den Pins des *PortB* am Mikrocontroller wieder.

Da die LEDs auf dem STK500 mit *active-low-Logik* betrieben werden, muss das jeweilige Bit gelöscht, also auf "0", gesetzt werden damit die LED leuchtet. Um alle Bits in einem Register zu verändern kann das Register mit einem 2-stelligen Hex-Wert(8-Bit) beschrieben werden. In Zeile 4 werden so alle Bits auf "1" gesetzt.

Um ein einzelnes Bit zu verändern, können die Zeilen 5 und 6 verwendet werden. Dabei steht das x in *PBX* für die x-te Stelle im Register die gesetzt oder gelöscht werden soll.

Es ist damit möglich im Programmablauf einzelne LEDs anzusteuern.

Listing 4.2: LEDs

```
1 #define LED_DDR DDRB  
2 #define LED_PORT PORTB  
3 LED_DDR = 0xFF;           // LED Port Richtung definieren (Ausgang)  
4 LED_PORT = 0xFF;         // LEDs ausschalten  
5 LED_PORT &= ~(1 << PBX); // loescht Bit an PortB – LED an  
6 LED_PORT |= (1 << PBX);  // setzt Bit an PortB – LED aus
```

##### 4.1.3. LCD ansteuern

Um den aktuellen Status des Motor komfortabel anzeigen zu können und den Mikrocontroller Menü basierte steuern zu können verwendete ich ein LC-Display.

Die meisten LC-Displays werden auf die selbe Art angesteuert. Hier gibt es fertige Bibliotheken die frei genutzt werden können. Im Projekt entschied ich mich für die

#### 4. Arbeitsablauf

von Peter Fleury [Fleury \[2012\]](#).

Dazu lud ich die Bibliothek herunter und entpackte die Dateien *lcd.c* und *lcd.h* in das Projektverzeichnis.

Die Bibliothek wird mit `#include "lcd.h"` eingebunden.

In der *lcd.h* mussten noch die Daten des Displays eingegeben werden (siehe Codelisting 4.3). Danach kann das Display mit den Befehlen aus Zeile 11–20 angesteuert werden.

Listing 4.3: lcd.h (Auszug)

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

#### 4.1.4. Serielle Schnittstelle ansteuern

*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). [Mikrocontroller.net \[2012\]](#)

Auf dem STK500 ist bereits eine serielle Schnittstelle vorbereitet. Um diese nutzen zu können musste ich den ersten UART des Mikrocontrollers (PortC 3:4) mit der Stiftleiste Rx/Tx auf dem STK500 verbinden.

Eine weitere Schnittstelle baute ich auf einem Steckbrett auf. Diese verband ich mit



#### 4. Arbeitsablauf

dem zweiten UART des Mikrocontrollers (PortC 1:2).

Um die Schnittstellen im Mikrocontroller nutzen zu können musste ich diese auch noch in der Software vorbereiten.

Das Codelisting 4.4 teilt sich in 4 wesentliche Bereiche:

- Zeilen 1 – 2: Setzen der Baudrate und einbinden der benötigten Bibliotheken.
- Zeilen 4 – 16: Initialisieren der Schnittstellen durch setzen der richtigen Bits in den entsprechenden Registern.
- Zeilen 17 – 32: Funktionen zum Senden von Daten
- Zeilen 34 – 65: Funktionen zum Empfangen von Daten

Listing 4.4: RS-232

```
1 #define BAUD 9600
2 #include <util/setbaud.h>
3
4 void    uart_init                () {    // Initialisierung der Schnittstellen
5     UBRR0H = UBRRH_VALUE; // UART 0 – IN (Rapidform Software/Terminal)
6     UBRR0L = UBRL_VALUE;
7     UCSR0C = (3 << UCSZ00);
8     UCSR0B |= (1 << TXEN0); //Transmitter Enabled
9     UCSR0B |= (1 << RXEN0); // UART RX einschalten
10
11     UBRR1H = UBRRH_VALUE; // UART 1 – OUT (Stepper Karte/Drehtisch)
12     UBRR1L = UBRL_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) {    // Versenden von einzelnen
    Zeichen
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) {    //Versenden von ganzen Strings
```

#### 4. Arbeitsablauf

```
28     while (*s) // so lange *s != '\0' also ungleich dem "String-Endezeichen(Terminator)"
29     {
30         uart_put_charater(*s, dir);
31         s++;
32     }
33 }
34 int uart_get_character (int dir) { // Empfang einzelner Zeichen
35     if (dir == D_RapidForm) {
36         while (!(UCSR0A & (1 << RXC0)))
37             // warten bis Zeichen verfuegbar
38             ;
39         return UDR0; // Zeichen aus UDR an Aufrufer zurueckgeben
40     }
41     if (dir == D_Stepper) {
42         while (!(UCSR1A & (1 << RXC1)))
43             // warten bis Zeichen verfuegbar
44             ;
45         return UDR1; // Zeichen aus UDR an Aufrufer zurueckgeben
46     }
47     return -1;
48 }
49 void uart_get_string (char * string_in, int dir) { // Empfang von ganzen Strings
50     char c;
51     int i = 0;
52     do { // Schleife zum zusammenbauen einzelner Zeichen zu einem String
53         c = uart_get_character(dir); // Einzelnes Zeichen holen
54         if (c != '\r') { // Wenn kein Zeilenende Zeichen
55             *string_in = c; // Aktuelles Zeichen im String = Zeichen
56             string_in += 1; // Position im String hochzaehlen
57             i++; // Schleifenzaehler hoch zaehlen.
58         }
59     } while (i < 100 && c != '\r' && c != '\n'); // Maximal bis 100 Zeichen oder
        // Zeilenende oder Zeilenvorschub
60     *string_in = '\0'; // Stringende Null-Terminieren
61     if (dir == D_Stepper) // Dateneingangs LEDs ausschalten
62         LED_PORT |= ( 1 << LED3 );
63     else
64         LED_PORT |= ( 1 << LED2 );
65 }
```

Mir stehen dadurch die essentiellen Funktionen `uart_put_string()` und `uart_get_string()` zur Verfügung. Mit diesen kann der Mikrocontroller über die serielle Schnittstelle Strings senden und empfangen.

## 4.2. Protokolle

Nun ging es um die Kommunikation zwischen PC und Peripheriegeräten bzw. der Peripheriegeräte untereinander. Das Ziel das erreicht werden sollte, bestand darin, dass der PC mit den Peripheriegeräten und die Peripheriegeräte untereinander miteinander kommunizieren sollten.

Die Befehlssätze die im Projekt verwendet werden um mit den Schrittmotorsteuerungen zu kommunizieren, werden als Protokolle bezeichnet. In diesen Protokollen ist geregelt, welche Befehle ausgesendet werden müssen und welche Antworten auf diese Befehle erwartet werden.

Jede Schrittmotorkarte verwendete ein eigenes Protokoll. Die Software RapidForm2004 kannte mehrere Protokolle um Schrittmotorkarten anzusteuern. Die vorhandenen Schrittmotorkarten hatten jedoch ein eigenes Protokoll das nicht in RapidForm2004 vorhanden war.

Nun soll auch der Mikrocontroller mit RapidForm2004 und einer der vorhandenen Schrittmotorkarten kommunizieren. Dazu musste der Mikrocontroller zumindest eines der Protokolle aus RapidForm2004 kennen und das Protokoll der vorhandenen Schrittmotorkarten.

In der ersten Phase verwendete ich die Software *Free Serial Port Monitor* um die Kommunikation zwischen dem PC und den Peripheriegeräten abzuhören. Dies hatte jedoch den Nachteil, das RapidForm2004 erst den nächsten Befehl sendete, wenn der erste mit der erwarteten Antwort quittiert wurde. Die Befehle die RapidForm erwartete, konnten zwar teilweise aus den Betriebsanleitungen der Schrittmotoren entnommen werden, dies war jedoch sehr mühsam.

Nach einiger Zeit kam ich auf die Idee *Reverse-Engineering* zu verwenden und war dadurch in der Lage alle Protokolle aus der ausführbaren Datei von RapidForm2004 auszulesen.

Das Codelisting 4.5 zeigt einen Auszug für das Protokoll eines Zeta Schrittmotors. Im Anhang befinden sich alle Protokolle.

Listing 4.5: Protokoll aus Rapidform: Zeta

```
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"
```

#### 4. Arbeitsablauf

```
9 timeout "60"  
10 firsttimeout "10"
```

Somit standen mir die Protokolle aller Schrittmotorsteuerungen zur Verfügung. Diese speicherte ich in einer Textdatei ab um sie bei Bedarf im Mikrocontroller Programm zu verwenden. Das bedeutete, dass nun alle Befehlssätze und deren Antworten, die ich im Projekt verwenden würde, vorhanden waren.

Im nächsten Schritt geht es darum, Befehle an die Schrittmotorkarte zu versenden. Da es nicht möglich ist, für jeden Befehl eine eigene Taste zu verwenden, entschied ich mich für eine Menübibliothek basierend auf verketteten Pointer Structs von AVR-Freaks [Twillman \[2011\]](#). Ich passte die Menüstruktur meinen Bedürfnissen an. Dazu gehörte dem Menü die Befehle und die Funktion zum versenden von Befehlen bekannt zu machen.

### 4.3. Kommunikation mit der Schrittmotorsteuerung

#### 4.3.1. Befehle senden

Der Drehtisch soll leicht und komfortabel vom Mikrocontroller aus angesteuert werden.

Dazu entschied ich mich für eine Menü basierte Steuerung auf dem LC-Display. Im Menü lässt sich mit den Tasten *Hoch* *Runter* *Ok* und *Zurück* navigieren.

Ich entschied mich für eine Menü Bibliothek Diese lud ich herunter und entpackte sie ins Arbeitsverzeichnis.

Danach passte ich die Menüstruktur auf meine Bedürfnisse an und speicherte sie in die Datei *mymenu.h*.

Außerdem musste der Menü-Bibliothek die Befehle der LCD-Bibliothek in der Datei *tinymenu.h* bekannt gemacht werden.

Die Zeilen 1–6 des Codelisting [4.6](#) dienen zum Einbinden der benötigten Bibliotheken.

Die Zeilen 8-16 zeigen eine vereinfachte Struktur meines Hauptprogramms. Wird ein Taster gedrückt wird dies durch die `get_key_press()` Funktion, bekannt aus Kapitel [4.1.1](#), erkannt und die entsprechende Menü Funktion aufgerufen.

Listing 4.6: Menü

```
1 #define MCU_CLK F_CPU  
2 #include "tinymenu/spin_delay.h"  
3 #define CONFIG_TINYMENU_USE_CLEAR  
4 #include "tinymenu/tinymenu.h"
```

#### 4. Arbeitsablauf

```
5 #include "tinymenu/tinymenu_hw.h"
6 #include "mymenu.h"
7
8 int main(void) {
9     while (1) {
10         if (get_key_press(1 << KEY0)) menu_enter(&menu_context, &menu_main);
11         if (get_key_press(1 << KEY1)) menu_prev_entry(&menu_context);
12         if (get_key_press(1 << KEY2)) menu_next_entry(&menu_context);
13         if (get_key_press(1 << KEY4)) menu_select(&menu_context);
14         if (get_key_press(1 << KEY4)) menu_exit(&menu_context);
15     }
16 }
17
18 void menu_puts          (void *arg, char *name) {
19     uart_put_string(arg, D_Stepper);
20     lcd_clrscr();
21     lcd_puts("Send: ");
22     lcd_puts(arg);
23     lcd_puts("\n");
24     ms_spin(100);
25     //if ((UCSR1A & (1 << RXC1)))
26     uart_rx(D_Stepper);
27     ms_spin(1000);
28 }
```

Wird einer der Menüpunkte aufgerufen, wird die im Menüpunkt hinterlegte Funktion mit dem hinterlegten Parameter aufgerufen. Wird beispielsweise der Befehl *+90* ausgewählt, wird die hinterlegte Funktion *menu\_puts()* (Codelisting 4.6 Zeile 18-28) mit dem hinterlegten Wert aufgerufen. Diese sendet dann mit der aus Kapitel 4.1.4 bekannten Funktion *uart\_puts(arg, dir)* einen Befehl an die Schrittmotorsteuerung. Nun kann mit den Tasten Hoch, Runter, Ok und Zurück im Menü Navigiert werden. Ist ein Befehl ausgewählt kann dieser durch Drücken des Ok Knopfes ausgewählt werden. Wird z.B. der Menüpunkt *+90* ausgewählt wird die Zeichenkette "M 125000" an die Drehtischsteuerung gesendet. Der Drehtisch dreht sich um 90° gegen den Uhrzeigersinn. **(TODO: ZUKUNFT: EINSTELLBARER WINKEL)**

Listing 4.7: Menü Baum

```
1 Main Menu
2   Bewegen – Rotation
3     +90
4     -90
5     +10.000 Schritte
6     -10.000 Schritte
```

#### 4. Arbeitsablauf

---

```
7   Gehe zum Ursprung
8   Bewegen – Hoehe
9   +500000
10  –500000
11  +1000000
12  –1000000
13  Gehe zum Ursprung
14  Konfigurieren
15  Motorstatus
16  Setze Ursprung
17  Write to EEPROM
18  Newline 1
19  Parameter Auslesen
```

#### 4.3.2. Befehle Empfangen und auswerten

asdf

### 4.4. Verbesserungen an der vorhandenen Hardware

#### 4.4.1. Spannungsversorgung

Um den Aufbau modular und erweiterbar zu machen, ersetzte ich die feste Lötverbindung durch eine Standard PC-Netzteil Verbindung(siehe Abbildung 4.1). Dadurch kann das Netzteil nun einfach ausgebaut werden, bzw. das System leicht mit neuen Einschubkarten erweitert werden.

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

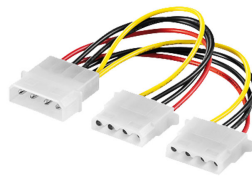


Abbildung 4.1.: Stromverbinder - Y-Kabel

100539.jpg)

#### 4. Arbeitsablauf

##### 4.4.2. Inbetriebnahme der zweiten Schrittmotorkarte

(TODO: SCHREIBEN!)

##### 4.4.3. Motorverkabelung

Zwischen der zweiten Schrittmotorkarte und dem zugehörigen Schrittmotor, der für die Höhenverstellung zuständig ist, war noch kein Kabel vorhanden. Dieses musste noch gefertigt und um 3 Leitungen für die Endschalter erweitert werden.

Dafür besorgte ich in der Werkstatt ein 7 adriges Kabel (Abbildung 4.3) und lies die passenden Endstecker bestellen.

Die Belegung wählte ich gleich zum Kabel für den ersten Schrittmotor, erweiterte sie jedoch um die 3 Adern für die beiden Endschalter.

Tabelle 4.1 gibt die Belegung des Kabels wieder.

Somit stand ein Kabel zur Verfügung mit dem sowohl der Schrittmotor gesteuert, als auch der Status der Endschalter ausgelesen werden konnte. (TODO: BELEGUNG ÜBERPRÜFEN!)



Abbildung 4.2.: Motor- und Endschalterverkabelung

Tabelle 4.1.: Motor- und Endschalterverkabelung

1	Phase A
2	Phase B
3	Phase C
4	Phase D
5	Endschalter oben
6	Endschalter unten
7	Endschalter Masse

#### 4.4.4. Endschalter

Normalerweise unterstützt die Schrittmotorkarte nur mechanische Endschalter. Durch geschickte Verdrahtung ließen sich die vorgegebenen induktiven Endschalter verwenden. Hierzu musste über einen Spannungsteiler die Spannung herabgesetzt werden. Dadurch konnten die Endschalter direkt an die Optokoppler der Schrittmotorkarte angeschlossen werden.

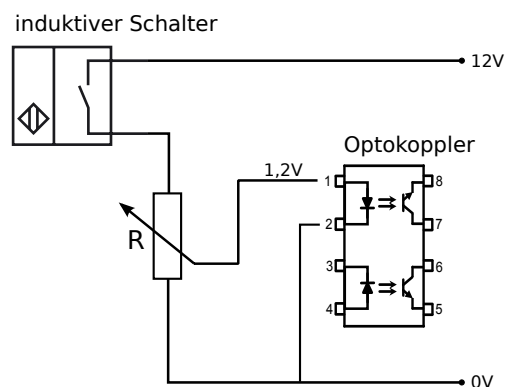


Abbildung 4.3.: Motor- und Endschalterverkabelung

Zur Abhilfe lies ich einen längeren Metallstutzen von der Werkstatt anfertigen. Wenn der Tisch sich in der Endposition befindet, soll dies auch auf dem Mikrocontroller angezeigt werden. Die Signale der Endschalter liegen auf der Rückseite der Schrittmotorkarte (**TODO: ZEICHNUNG DER ANSCHLÜSSE REFERENZIEREN.**) am Verbindungsstecker an. Ich lötete eine Brücke zwischen den Verbindungssteckern der Schrittmotorkarte und des Mikrocontrollers.

Auf der Mikrocontrollerplatine sind diese Pins mit dem Mikrocontroller verbunden. Die beiden Pins werden im Mikrocontroller als *Interrupts* definiert. Die Interrupt-Service-Routine wird in Kapitel 5.0.4 beschrieben.

Da die Signale der Endschalter nun an der Schrittmotorkarte anlagen, konnte diese den Motor stoppen. Zusätzlich lagen die Signale am Mikrocontroller an. Dieser konnte dadurch auf dem Display die Meldung *Endschalterposition erreicht!* ausgeben. Siehe dazu auch Kapitel 2.3.6





#### 4.5. Kommunikation mit RapidForm2004

#### 4.5.1. Befehle empfangen

© Johannes Dielmann

#### 4. Arbeitsablauf

Dieser Parameter gibt an das der Befehl von der ersten seriellen Schnittstelle empfangen wurde. Dies bestimmt wie der empfangene *String* weiterverarbeitet wird.

Die Funktion `uart_rx()` liest dann das Empfangsregister mit der aus Kapitel 4.1.4 bekannten Funktion `uart_get_string()` aus und schreibt den empfangenen String in die Variable `str_rx` (Codelisting 4.8-Zeile 7).

In einer ersten IF-Abfrage wird entschieden, von welcher Schnittstelle der empfangene Befehl kam. Da `D_RapidForm` übergeben wurde, wird der else-Teil der Abfrage ausgeführt.

Dieser entscheidet mit mehreren IF-Abfragen ob bereits ein Protokoll für einen bestimmten Motor ausgewählt wurde. Ist dies nicht der Fall wird der String `str_rx` an die Funktion `switch_Motor()` (Codelisting 5.2) übergeben (Codelisting 4.8 Zeile 17). Diese prüft den String gegen die Initialisierungsbefehle der einzelnen Protokolle. Die Initialisierungsbefehle sind die ersten Befehle die RapidForm2004 an eine Schrittmotorsteuerung sendet um zu prüfen ob diese vorhanden ist.

`switch_Motor()` gibt einen numerischen Wert zurück. Jede Zahl entspricht dabei einem Motorprotokoll. Die Zahlenwerte werden dabei mittels Makro-Definitionen (Codelisting 4.8 Zeile 1-6) mit lesbare Text-Variablen ersetzt. Dies erhöht die Lesbarkeit und das Verständnis ungemein.

War dieser Schritt erfolgreich, wird in den folgenden IF-Abfragen die richtige Auswertefunktion aufgerufen. Konnte die Funktion `switch_Motor()` den empfangenen Befehl nicht zuordnen, gibt sie `M_UNK` zurück und es wird auf dem Display *Unbekannter Motor!* ausgegeben.

Somit ist es mir möglich Befehle der Schnittstelle zu empfangen und an die richtigen Auswertefunktionen zu übergeben. Zusätzlich wird bereits automatisch das richtige Protokoll ausgewählt das RapidForm2004 verwendet hat.

Eine Limitierung ist, das für eine erneute Protokollwahl der Mikrocontroller neu gestartet werden müsste.

Listing 4.8: RS-232 Empfang

```
1 #define M_UNK      -2
2 #define M_NOTI     -1
3 #define M_ISEL      0
4 #define M_CSG       1
5 #define M_ZETA      2
6 #define M_TERMINAL 3
7
8 int      Initialized = M_NOTI;
9
10 if ((UCSR0A & (1 << RXC0))){
```

#### 4. Arbeitsablauf

---

```
11     LED_PORT &= ( 1 << LED2 );
12     uart_rx(D_RapidForm);
13 }
14
15 void  uart_rx                (int dir) {
16     uart_get_string(str_rx, dir);
17     if (dir == D_Stepper)
18         switch_Stepper(str_rx);
19     else {
20         if ( Initialized == M_UNK){
21             lcd_puts("Unbekannter Motor!\n");
22             //lcd_puts(str_rx);
23             Initialized = M_NOTI;
24         }
25         if ( Initialized == M_NOTI){
26             Initialized = switch_Motor(str_rx);
27         }
28         if ( Initialized == M_ISEL)
29             switch_Isel(str_rx);
30         if ( Initialized == M_CSG)
31             switch_csg(str_rx);
32         if ( Initialized == M_ZETA)
33             switch_Zeta(str_rx);
34         if ( Initialized == M_TERMINAL)
35             switch_Terminal(str_rx);
36     }
37 }
```

#### 4.5.2. Befehle verarbeiten

## 5. Parkplatz

### 5.0.3. Übersetzungs Logik

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

#### 5.0.3.1. FindStringInArray Funktion

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.

Listing 5.1: Funktion: FindStringInArray

```
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 }
```

#### 5.0.3.2. Automatische Protokollwahl

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 5.2: Funktion: switchMotor

```
1 #define M_UNK      -2  
2 #define M_NOTI     -1  
3 #define M_ISEL      0  
4 #define M_CSG       1  
5 #define M_ZETA      2
```

```
6 #define M_TERMINAL 3
7
8 int    Initialized = M_NOTI;
9
10 int    switch_Motor      (char * str_rx) {
11     const char* pOptions[] = {
12         "@01",           // 0 - Isel
13         "Q:",            // 1 - CSG
14         "ECHO0",         // 2 - Zeta
15         "!Terminal",     // 3 - Terminal ansteuerung!
16         0 };
17     switch (FindStringInArray(str_rx, pOptions, 3)) {
18     case 0:              // 0 - ISEL
19         return M_ISEL;
20         break;
21     case 1:              // 1 - CSG
22         return M_CSG;
23         break;
24     case 2:              // 2 - Zeta
25         return M_ZETA;
26         break;
27     case 3:              // 3 - Terminal ansteuerung
28         return M_TERMINAL;
29         break;
30     default:
31         return M_UNK;
32     }
33 }
```

#### 5.0.3.3. Zeta

Zeta

#### 5.0.3.4. 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 5.3: Übersetzungs Logik: Isel

```
1 void switch_Isel (char * str_rx) {
2     const char* pOptions[] = {
3         "XXXXXXX", // 0 - Reserve
4         "!CLS",    // 1 - LC-Display loeschen
5         "Test",    // 2 - Test
6         "@01",     // 3 - Achse auswaehlen
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 loeschen
15     case 2: // 2 - Test
16     case 3: // 3 - Achse auswaehlen
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] = ' ';
```

5. Parkplatz

```
45     Move_To[3] = '\0';
46     strcat(Move_To, Position);
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 }

#### 5.0.3.5. 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.

#### 5.0.4. 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-Cycles 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.

##### 5.0.4.1. 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 5.4 zeigt die ISR für die Endschalter.

Listing 5.4: ISR: Endschalter

```
1 PCMSK3 |= ( 1 << PCINT28 ); // Interrupts definieren PD4 als Interrupt zulassen
2 PCICR  |= ( 1 << PCIE3 ); // Pin Change Interrupt Control Register – PCIE3 setzen fuer
   PCINT30
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 }

#### 5.0.4.2. 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 5.5 Zeilen 12-15.

Listing 5.5: 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 }
```

## 6. Probleme und Lösungen

### 6.1. Entwicklungsumgebungen

#### 6.1.1. AVR Studio 5

Die von Atmel AVR Studio 5 ist eine von Atmel bereitgestellte Entwicklungsumgebung. Diese scheint jedoch eine fehlerhafte Bibliothek zu enthalten. Die Kombination aus Mikrocontroller ATmega324A und AVR Studio 5 erzeugte nicht nachvollziehbare Probleme. Bei dem selbem Programm und einem anderem Mikrocontroller oder einer anderen Entwicklungsumgebung tauchten keine Fehler auf. In der Entwicklungsumgebung Eclipse lies sich der Fehler reproduzieren wenn der Pfad der Atmel Bibliotheken eingestellt wurde. Die WinAVR Bibliotheken und eine selbst kompilierte *Toolchain* unter Linux zeigten keine Probleme.

Daher wechselte ich zur *Open Source* Entwicklungsumgebung Eclipse. Erst dadurch wurde es möglich erfolgreich zu arbeiten. Außerdem wurde das Projekt dadurch plattformunabhängig und ich nutzte bis auf RapidForm2004 nur noch freie Open Source Software.

#### 6.1.2. Eclipse

Eclipse ist eine in Java programmierte freie Open Source Entwicklungsumgebung für Java. Sie lässt sich durch *Plugins* leicht für viele Sprachen erweitern.

Mit dem CDT-Plugin, dem AVR-Plugin und einer Bibliothek wie z.B. WinAVR für Windows ist Eclipse eine vollwertige Entwicklungsumgebung für Atmel Mikrocontroller. Ergänzt wird diese durch die Programmiersoftware AVR-Dude.

## 6.2. Fuses

Als Fuses werden Register bezeichnet mit denen sich, auf Hardwareebene, das Verhalten des Mikrocontrollers 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 6.1.: 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

### 6.3. Platinenlayout

## 7. Fazit und Zukunft

### 7.1. Fazit

**(TODO: FAZIT SCHREIBEN!)**

## Literaturverzeichnis

### **Atm 2003**

ATMEL CORPORATION (Hrsg.): *AVR STK500 User Guide*. San Jose, CA 95131, USA: Atmel Corporation, 06 2003

### **Atm 2011**

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

### **Corporation 2012a**

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

### **Corporation 2012b**

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

### **Dannegger 2012**

DANNEGGER, Peter: *Entprellung - Mikrocontroller.net*. <http://www.mikrocontroller.net/articles/Entprellung>. Version: 2012. – [Online; Stand 11. Februar 2012]

### **Fleury 2012**

FLEURY, Peter: *Peter Fleury's Home Page*. <http://jump.to/fleury>. Version: 2012. – [Online; Stand 11. Februar 2012]

### **Mikrocontroller.net 2012**

MIKROCONTROLLER.NET: *RS-232 - Mikrocontroller.net*. <http://www.mikrocontroller.net/articles/RS-232>. Version: 2012. – [Online; Stand 11. Februar 2012]

### **Minolta 2012**

MINOLTA: *Funktionen - VI-910 / KONICA MINOLTA*. <http://>

[//www.konicaminolta.eu/de/messinstrumente/produkte/3d-messtechnik/beruehrungsloser-3d-scanner/vi-910/funktionen.html](http://www.konicaminolta.eu/de/messinstrumente/produkte/3d-messtechnik/beruehrungsloser-3d-scanner/vi-910/funktionen.html). Version: 2012. – [Online; Stand 11. Februar 2012]

**V9141 2001**

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

**Twillman 2011**

TWILLMAN, Tymm: *AVR Freaks*. <http://www.avrfreaks.net/>. Version: 2011. – [Online; Stand 01. November 2011]

**Wikipedia 2012a**

WIKIPEDIA: *American Standard Code for Information Interchange* — *Wikipedia, Die freie Enzyklopädie*. [http://de.wikipedia.org/w/index.php?title=American\\_Standard\\_Code\\_for\\_Information\\_Interchange&oldid=99892678](http://de.wikipedia.org/w/index.php?title=American_Standard_Code_for_Information_Interchange&oldid=99892678). Version: 2012. – [Online; Stand 23. Februar 2012]

**Wikipedia 2012b**

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]

**Wikipedia 2012c**

WIKIPEDIA: *Kommunikationsprotokoll* — *Wikipedia, Die freie Enzyklopädie*. <http://de.wikipedia.org/w/index.php?title=Kommunikationsprotokoll&oldid=99325271>. Version: 2012. – [Online; Stand 25. 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 25. 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. Protokoll der Schrittmotorkarte

Tabelle A.1 zeigt den ASCII Befehlssatz der Schrittmotorkarte.

_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 A.1.: ASCII Befehlssatz R+S Schrittmotorsteuerung

V9141 [2001] Der ”\_” wird mit der anzusteuernenden Kartennummer ersetzt. Dabei wird von 1 aufwärts gezählt. Bei der ersten Karte kann die Nummer weggelassen werden.

### A.3. Protokolle aus RapidForm2004

Listing A.1: RapidForm2004 Protokolle Empfang

```
1 model "CSG-602R(Ver.2.0)"
2 port "9600" "n81h"
3 init "D:2S500F5000R200S500F5000R200\r\nH:1-\r\n" ""
4 finish "L:1\r\nH:1-\r\n" ""
5 arot "M:1+P%d\r\nG:\r\n" ""
6 stop "L:E\r\n" ""
7 home "L:1\r\nH:1-\r\n" ""
8 step "-0.0025" "-99999999" "99999999"
9 timeout "60"
10 firsttimeout "10"
11
12 model "CSG-602R(Ver.1.0)"
13 port "9600" "n81h"
14 init "D:2S500F5000R200S500F5000R200\r\nH:1-\r\n" ""
15 finish "L:1\r\nH:1-\r\n" ""
16 arot "M:1+P%d\r\nG:\r\n" ""
17 stop "L:E\r\n" "" home "L:1\r\nH:1-\r\n" ""
18 step "-0.005" "-99999999" "99999999"
19 timeout "60"
20 firsttimeout "10"
21
22 model "Mark-202"
23 port "9600" "n81h"
24 init "S:180\r\nD:1S20000F200000R200\r\n" "OK\r\nOK\r\n"
25 finish "L:1\r\nH:1\r\n" "OK\r\nOK\r\n"
26 arot "M:1+P%d\r\nG:\r\n" "OK\r\nOK\r\n"
27 stop "L:E\r\n" "OK\r\n"
28 home "L:1\r\nH:1\r\n" "OK\r\nOK\r\n"
29 step "-0.0000625" "-99999999" "99999999"
30 timeout "60" firsttimeout "10"
31
32 model "Mark-102" port "9600" "n81h"
33 init "D:WS500F5000R200S500F5000R200\r\n" "OK\r\nOK\r\n"
34 finish "H:1-\r\n" "OK\r\n"
35 arot "M:1+P%d\r\nG:\r\n" "OK\r\nOK\r\n"
36 stop "L:E\r\n" "OK\r\n"
37 home "H:1-\r\n" "OK\r\n"
38 step "-0.0025" "-99999999" "99999999"
39 timeout "60"
40 firsttimeout "10"
41
```

## A. Anhang

```

42 model "isel(RF-1)"
43 port "9600" "n81h"
44 init "@01\r" "0"
45 finish "@0M0\054+600\r" "0"
46 arot "@0M%d\054+600\r" "0"
47 stop "" "0"
48 home "@0M0\054+600\r" "0"
49 step "-0.0173076" "-8000000" "8000000"
50 timeout "60"
51 firsttimeout "10"
52
53 model "isel(R-1)"
54 port "9600" "n81h"
55 init "@01\r" "0"
56 finish "@0M0\054+300\r" "0"
57 arot "@0M%d\054+300\r" "0"
58 stop "" "0"
59 home "@0M0\054+300\r" "0"
60 step "-0.05" "-8000000" "8000000"
61 timeout "60"
62 firsttimeout "10"
63
64 model          "ZETA6104"
65 port           "9600"                                "n81n"
66 init           "ECHO0\rCOMEXC0\PSET0\rA8\rV8\r" ""
67 finish         " D0 \rGO1\rWAIT(1PE<>1)\rRESET\r" "\r\n>\040\r\n>\040\r\n>\040\r\n>\040\r\n>\040\r\n>\040\r\n>\040\r\n>\040\r\n>"
68 arot           "MA1 D%d\rGO1\rWAIT(1PE<>1)\r" "\r\n>\040\r\n>\040\r\n>\040\r\n>\040\r\n>\040\r\n>\040\r\n>\040\r\n>"
69 home           "MA1 D0 \rGO1\rWAIT(1PE<>1)\r" "\r\n>\040\r\n>\040\r\n>\040\r\n>\040\r\n>\040\r\n>\040\r\n>"
70 stop           "!S\r"                               "\r\n>\040"
71 step           "0.00008"                             "-4500000" "4500000"
72 timeout        "60"
73 firsttimeout   "10"
74
75 model "MMC-2(15kg)"
76 port "9600" "n81h"
77 init "F:XP3\r\nS:X1\r\n" "\r\n\r\n"
78 finish "L:X\r\nA:XP0\r\nW:\r\n" "\r\n\r\n"
79 arot "A:XP%d\r\nW:\r\n" "\r\n\r\n"
80 stop "E:\r\n" "\r\n"
81 home "A:XP0\r\nW:\r\n" "\r\n"
82 step "0.002" "-99999999" "99999999"
83 timeout "60"
```

*A. Anhang*

---

```
84 firsttimeout "10"
85
86 model "MMC-2"
87 port "9600" "n81h"
88 init "F:XP3\r\nS:X1\r\n" "\r\n\r\n"
89 finish "L:X\r\nA:XP0\r\nW:\r\n" "\r\n\r\n"
90 arot "A:XP%d\r\nW:\r\n" "\r\n\r\n"
91 stop "E:\r\n" "\r\n"
92 home "A:XP0\r\nW:\r\n" "\r\n"
93 step "0.005" "-99999999" "99999999"
94 timeout "60"
95 firsttimeout "10"
```

## A.4. 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.2.: 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	111 bis 463mm (TELE),
X-Richtung (anhängig vom Anstand)	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: $\pm 0,22\text{mm}$ , Y: $\pm 0,16\text{mm}$ , Z: $\pm 0,10\text{mm}$ 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 × 480 × 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 × 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 × 413 × 271mm
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.5. Verwendete Hardware

- **VI-900**

Konica Minolta Sensing Europe, B.V.

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

- **ATMega 324A**

Atmel Corporation

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

- **STK500**

Atmel Corporation

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

- **AVRISP mkII**

Atmel Corporation

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

- **Induktiver Endschalter**

Pepperl+Fuchs

Website: [http://www.pepperl-fuchs.de/germany/de/classid\\_143.htm?view=productdetails&productid=3012](http://www.pepperl-fuchs.de/germany/de/classid_143.htm?view=productdetails&productid=3012)

- **MAX232**

Texas Instruments Incorporated

Website: <http://www.ti.com/product/max232>

## A.6. Verwendete Software

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

- **RapidForm2004** (Closed Source)

INUS Technology, Inc.

Website: <http://www.rapidform.com>

- **AVRStudio 5** (Closed Source)

Atmel Corporation

Website: <http://www.atmel.com/>

*A. Anhang*

---

- **Eclipse** mit CDT und AVRPlugin  
The Eclipse Foundation Website: <http://www.eclipse.org>  
Website: <http://www.eclipse.org/>
- **AVRDude**  
Prorammer
- **Blender**
- **Texmaker**
- **LaTeX**
- **GIT**
- **Inkscape**
- **Hyperterminal** (Closed Source)

**(TODO: WEITERE?!)**