

VORLÄUFIGE ARBEITSKOPIE!

ÜBERSETZEN VON SCHRITTMOTORBEFEHLEN

Entwurf eines Hardwareübersetzers

Praxisbericht

im Fachgebiet Mess- und Sensortechnik



vorgelegt von: Johannes Dielmann

Geburtsdatum: 10. Januar 1984

Geburtsort: Kirchen

Matrikelnummer: 515956

Erstgutachter: Prof. Dr. Carstens-Behrens

© 2012

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

Inhaltsverzeichnis

Inhaltsverzeichnis

| | |
|--|-----------|
| 1. Einleitung | 1 |
| 2. Vorstellung der vorhandenen Hardware | 2 |
| 2.1. Computer | 3 |
| 2.2. 3D-Laserscanner VI-900 | 3 |
| 2.2.1. Lasertriangulator Prinzip | 3 |
| 2.3. Drehtisch und Ansteuerung | 4 |
| 2.3.1. Drehtisch | 4 |
| 2.3.2. Spannungsversorgung | 4 |
| 2.3.3. Schrittmotoren | 5 |
| 2.3.4. Schrittmotorkarten | 6 |
| 2.3.5. Motorverkabelung | 6 |
| 2.3.6. Endschalter | 7 |
| 2.4. Mikrocontroller | 7 |
| 2.4.1. Entwicklerboard STK500 | 7 |
| 2.4.2. AVRISP mkII | 8 |
| 2.4.3. MAX232 | 9 |
| 3. Vorstellung der vorhandenen Software | 10 |
| 3.1. RapidForm2004 | 10 |
| 3.2. Entwicklungsumgebung | 10 |
| 3.3. Terminalprogramme | 10 |
| 4. Zeitlicher Arbeitsablauf | 11 |
| 4.1. Bereitstellen grundlegender Funktionalitäten | 12 |
| 4.1.1. Taster | 12 |
| 4.1.2. LEDs | 13 |
| 4.1.3. Ansteuerung des LC-Display | 13 |
| 4.1.4. RS-232-Schnittstelle | 14 |
| 4.2. Befehlssätze | 16 |
| 4.3. Kommunikation mit der vorhandenen Schrittmotorsteuerung | 17 |
| 4.3.1. Befehle senden | 17 |
| 4.3.2. Antworten empfangen und speichern | 19 |

Inhaltsverzeichnis

| | |
|---|-----------|
| 4.3.3. Antworten auswerten | 20 |
| 4.4. Verbesserungen an der vorhandenen Hardware | 21 |
| 4.4.1. Netzteil | 21 |
| 4.4.2. Zweite Schrittmotorkarte | 22 |
| 4.4.3. Motor- und Endschalterverkabelung | 23 |
| 4.4.4. Endschalter | 23 |
| 4.4.5. Zweite serielle Schnittstelle | 24 |
| 4.5. Kommunikation mit RapidForm2004 | 25 |
| 4.5.1. Befehle empfangen | 25 |
| 4.5.1.1. Automatische Auswahl eines Befehlssatzes | 26 |
| 4.6. Auswerte-Funktionen | 28 |
| 4.6.1. Auswerte-Funktion für Isel-Motoren | 29 |
| 4.6.1.1. Initialisierung | 29 |
| 4.6.1.2. Statusabfrage | 30 |
| 4.6.1.3. Rotation | 30 |
| 4.7. Platinenlayout und 19"-Einschub | 33 |
| 5. Probleme und Lösungen | 36 |
| 5.1. Entwicklungsumgebungen | 36 |
| 5.1.1. AVR Studio 5 | 36 |
| 5.1.2. Eclipse | 36 |
| 5.2. Interrupts | 36 |
| 5.2.1. Endschalter | 37 |
| 5.2.2. Watchdog | 37 |
| 5.3. Fuses | 38 |
| 6. Fazit und weitere Möglichkeiten | 40 |
| 6.1. Fazit | 40 |
| 6.2. Bekannte Probleme | 40 |
| 6.3. Weitere Möglichkeiten | 40 |
| Eidesstattliche Erklärung | 42 |
| A. Anhang | 43 |
| A.1. Schritt für Schritt Anleitung | 44 |
| A.2. Befehlssatz der vorhandene Schrittmotorkarte | 51 |
| A.3. Befehlssätze aus RapidForm2004 | 52 |
| A.4. Technische Daten VI-910 | 55 |
| A.5. Verwendete Hardware | 57 |
| A.6. Verwendete Software | 57 |

Inhaltsverzeichnis

| | |
|------------------------|----|
| A.7. main.c | 59 |
| A.8. Credits | 78 |

Abbildungsverzeichnis

Abbildungsverzeichnis

| | | |
|------|--|----|
| 2.1. | Blick auf den Arbeitsaufbau | 2 |
| 2.2. | VI-900 - Kamera oben, Lasertriangulator unten | 3 |
| 2.3. | Prinzip: Laser-Triangulation | 4 |
| 2.4. | Drehtisch | 5 |
| 2.5. | Ansteuerung im 19"-Rack | 6 |
| 2.6. | Block Diagramm: Mikrocontroller ATmega324A[2] | 8 |
| 2.7. | Schemazeichnung eines STK500[1] | 9 |
| 4.1. | Stromverbinder - Y-Kabel[6] | 22 |
| 4.2. | Motor- und Endschalterverkabelung | 23 |
| 4.3. | Endschalterverkabelung | 24 |
| 4.4. | Schaltplan für die zweite serielle Schnittstelle [7] | 25 |
| 4.5. | Schema der Kommunikation | 26 |
| 4.6. | Schaltplan | 34 |
| 4.7. | Platinenlayout | 35 |
| 4.8. | 19"-Einschub | 35 |

Tabellenverzeichnis

| | |
|--|----|
| 2.1. Aufbau | 2 |
| 4.1. Motor- und Endschalterverkabelung | 23 |
| 5.1. Fuses | 39 |
| A.1. Schritt für Schritt Anleitung | 44 |
| A.2. ASCII Befehlssatz R+S Schrittmotorsteuerung | 51 |
| A.3. Technische Daten - VI-910 | 55 |

Codeverzeichnis

| | | |
|-------|---|----|
| 4.1. | Taster | 12 |
| 4.2. | LEDs | 13 |
| 4.3. | lcd.h (Auszug) | 14 |
| 4.4. | RS-232 | 15 |
| 4.5. | Befehlssatz aus Rapidform: Isel | 17 |
| 4.6. | Menü | 18 |
| 4.7. | Menü Baum | 18 |
| 4.8. | RS-232 Empfang | 20 |
| 4.9. | FindStringInArray() | 20 |
| 4.10. | switchStepper() | 21 |
| 4.11. | RS-232 Empfang - RapidForm2004 | 26 |
| 4.12. | Funktion: uart rx() | 27 |
| 4.13. | Funktion: switch Motor() | 27 |
| 4.14. | Übersetzungs Logik für einen Isel-Motor | 29 |
| 4.15. | case 3: Initialisierung | 29 |
| 4.16. | case 4: Statusabfrage | 30 |
| 4.17. | case 5: Rotation | 31 |
| 4.18. | Funktion: string zerlegen Isel() | 32 |
| 5.1. | ISR: Endschalter | 37 |
| 5.2. | Watchdog | 38 |
| A.1. | Befehlssätze aus RapidForm2004 | 52 |
| A.2. | main.c | 59 |

1. Einleitung

1. Einleitung

Ein 3D-Laserscanner bietet vielfältige Möglichkeiten und Einsatzgebiete. Die Haupt-einsatzgebiete finden sich in der Bauteileprüfung, der Erstellung von Finite-Elemente-Daten in Verbindung mit Bauteilanalyse, der Erstellung von 3D-Daten, der Kontrolle von Zubehörteilen und dem Reverse-Engineering.

Im Besitz der Fachhochschule Koblenz befindet sich ein komplettes 3D-Laserer-fassungssystem. Dazu gehören eine Erfassungssoftware, ein 3D-Laserscanner und ein Drehtisch. Bisher müssen für eine Aufnahme, alle Komponenten zueinander passen. Der Drehtisch in diesem System ist jedoch ein Eigenbau der Fachhochschule Koblenz und die darin verbaute Drehtischsteuerung nicht kompatibel zu denen, von der Er-fassungssoftware unterstützten, Drehtischsteuerungen.

Mittels eines Mikrocontrollers soll der vorhandene Aufbau so erweitert werden, dass der Drehtisch von der Software angesteuert werden kann und so der volle Umfang des Systems nutzbar gemacht werden. Dabei sind folgende Aufgaben zu realisieren. Die Höhenverstellung des Drehtisches soll genutzt werden können und die verbauten Endschalter ihre vorhergesehene Funktion erfüllen. Der Mikrocontroller soll sich mit mehreren Tastern bedienen lassen und über ein LC-Display verfügen, welches den aktuellen Status anzeigt. Mit einer Schritt-für-Schritt-Anleitung soll es auch für Stu-denten und Mitarbeiter der Fachhochschule möglich sein, schnell und einfach eine Aufnahme durchzuführen. Die Daten dieser Aufnahme sollen exportiert und in z.B. CAD-Anwendungen nutzbar sein.

Der Aufbau der Arbeit gliedert sich im Wesentlichen in die Vorstellung der vorhandenen Hard- und Software, dem chronologischen Arbeitsablauf während des Projektes, ein Kapitel das Probleme und deren Lösungen aufzeigt, in ein Fazit und mögliche zu-künftige Verbesserungen. Im Anhang befindet sich eine Schritt-für-Schritt-Anleitung die es Laien ermöglicht 3D-Modelle aufzunehmen und zu exportieren.

2. Vorstellung der vorhandenen Hardware

2. Vorstellung der vorhandenen Hardware

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

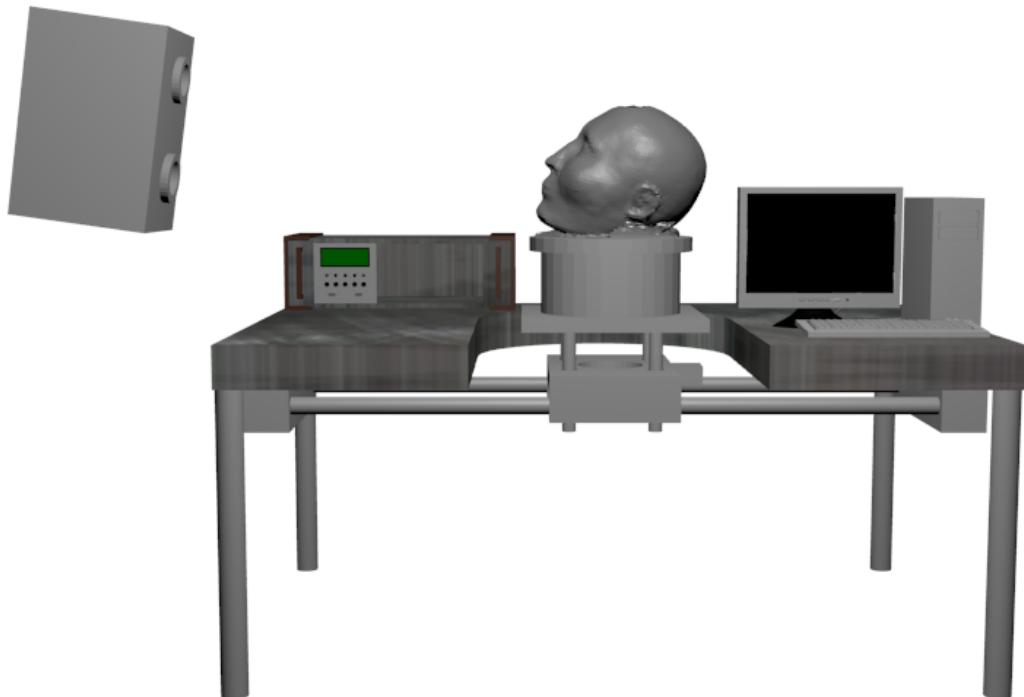


Abbildung 2.1.: Blick auf den Arbeitsaufbau

| | |
|---|---|
| 1 | 2.1 Computer |
| 2 | 2.2 3D-Laserscanner VI-900 |
| 3 | 2.3 Ansteuerung für den Drehtisch |
| 4 | 2.4 Mikrocontroller |

Tabelle 2.1.: Aufbau

 2. Vorstellung der vorhandenen Hardware

2.1. Computer

Zur Verfügung steht ein IBM kompatibler x86 Standard PC mit einer **SCSI-** und einer **RS-232-Schnittstelle**. Auf diesem ist die Erfassungssoftware *RapidForm2004* [A.6] installiert. Die SCSI Schnittstelle wird zur Kommunikation mit dem 3D-Laserscanner und die RS-232-Schnittstelle zur Kommunikation mit einer Schrittmotorsteuerung genutzt.

2.2. 3D-Laserscanner VI-900

Der 3D-Laserscanner *VI-900* der Firma *Konica Minolta*[A.5] besteht, wie auf Abbildung 2.2 zu sehen, aus einer Kamera und einem **Lasertriangulator**. 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.

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.[8]

Die Technischen Daten befinden sich im Anhang in Tabelle A.3



Abbildung 2.2.: VI-900 - Kamera oben, Lasertriangulator unten

2.2.1. Lasertriangulator Prinzip

Ein Lasertriangulator besteht, wie in Abbildung 2.3 zu sehen, aus einem Laser, einem Linsensystem und im einfachsten Fall, aus einer Pixeldetektorzeile. Der Laser

2. Vorstellung der vorhandenen Hardware

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.

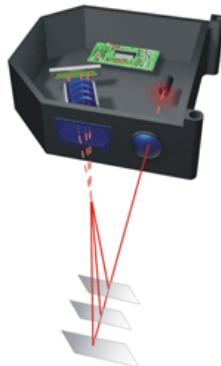


Abbildung 2.3.: Prinzip: Laser-Triangulation

2.3. Drehtisch und Ansteuerung

2.3.1. Drehtisch

Der Tisch in dem der Drehtisch verbaut ist, ist eine Eigenkonstruktion der Werkstatt des RheinAhrCampus Remagen. Er besteht aus einer massiven Edelstahlarbeitsplatte, welche auf 4 Füßen ruht. Aus dieser ist ein Rechteck mit aufgesetztem Halbkreis ausgeschnitten. In diesem Ausschnitt befindet sich der Drehtisch(siehe Abbildung 2.4). Er ist auf einem Schienensystem 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 des Getriebes 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 Verbindungsleisten gelötet. Dies verhindert das einfache Ausbauen der Spannungsversorgung und die einfache Erweiterung um neue Einschubkarten.

2. Vorstellung der vorhandenen Hardware



Abbildung 2.4.: Drehtisch

2.3.3. Schrittmotoren

Für die Rotation kommt der Schrittmotor 440-458 der Firma R+S zum Einsatz. Dieser hat einen Schrittewinkel von $1,8^\circ$, eine Haltedrehmoment von 500mNm, wird mit 8-Drahtleitung verschaltet und mit 12V Gleichspannung versorgt. Aus dem Schrittewinkel ergeben sich 200 Schritte pro Umdrehung. Diese werden mit einem **Harmonic-Drive-Getriebe**, mit einer Übersetzung von 500:1, auf 100.000 Schritte pro Umdrehung erhöht.

Für die Höhenverstellung wird der Schrittmotor 440-420, ebenfalls von der Firma R+S, verwendet. Dieser hat auch einen Schrittewinkel von $1,8^\circ$, hat jedoch ein Haltemoment von 70mNm, wird in 6-Drahtleitung verschaltet und mit 5V Gleichspannung gespeist. Dieser ist mit einer Übersetzung von 5:1 und einem Schneckengetriebe mit dem Drehtisch verbunden.

 2. Vorstellung der vorhandenen Hardware

2.3.4. Schrittmotorkarten

Die Ansteuerung für die Schrittmotoren sind als 19"-Einschübe realisiert, siehe Abbildung 2.5 links. 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 vorgegebenen ASCII¹ Befehlssatz. Der Befehlssatz befindet sich im Kapitel A.2. Es können zwei oder mehr Karten als Daisy-Chain² in Reihe geschaltet werden. Zu Beginn des Projekts war nur die erste Schrittmotorsteuerung vorbereitet.

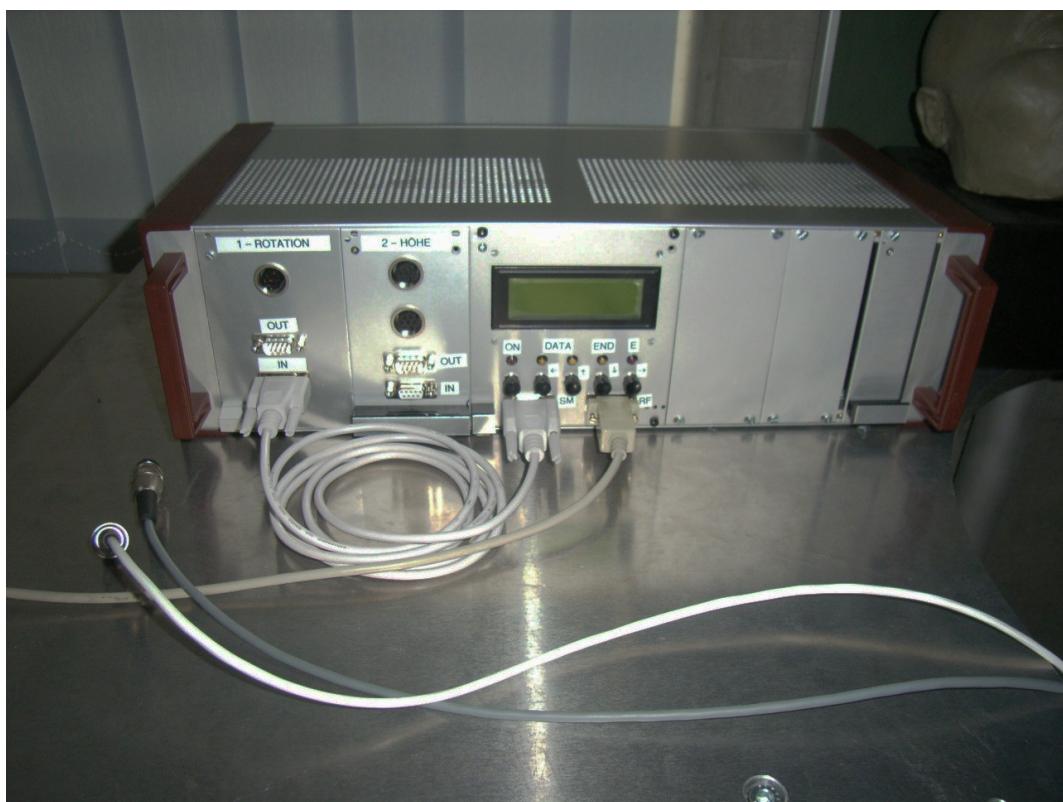


Abbildung 2.5.: Ansteuerung im 19"-Rack

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

¹Der American Standard Code for Information Interchange (ASCII, alternativ US-ASCII, oft [æski] ausgesprochen) ist eine 7-Bit-Zeichenkodierung[10]

²Als 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).[11]

2. Vorstellung der vorhandenen Hardware

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. Diese werden durch einen Metallstutzen ausgelöst. Dieser ist jedoch schlecht positioniert oder ungenügend lang. Würde der Drehtisch über seine Grenzen hinaus in der Höhe verstellt werden, würden die Endschalter nicht rechtzeitig ausgelöst werden und der Aufbau würde beschädigt werden.

2.4. Mikrocontroller

Ein Mikrocontroller besteht, wie in Abbildung 2.6 zu sehen, aus CPU, Flash-Speicher, EEPROM, Registern, Ports und mehreren Peripherie-Funktionen wie z.B. Timern, ADC, DAC und seriellen Schnittstellen. Für unterschiedliche Aufgaben können unterschiedliche Mikrocontroller verwendet werden, welche sich in ihrem Funktionsumfang unterscheiden.

Besonders wichtig im Mikrocontroller sind die sogenannten Register. Diese sind spezielle, meist 8-Bit breite, Abschnitte im Speicher. Sie repräsentieren Werte und Einstellungen im Mikrocontroller. Diese können beschrieben und ausgelesen oder nur ausgelesen werden. Durch das Auslesen oder Beschreiben der Register kann der Mikrocontroller mit internen und externen Komponenten interagieren. Die Register die zur externen Kommunikation dienen werden als Ports bezeichnet.

Es stand ein ATmega8515 [3] im DIL-Gehäuse zur Verfügung. Dieser hatte 8 Kbyte Flash, drei externe Interrupts, eine serielle Schnittstelle und konnte mit bis zu 16 MHz betrieben werden. Dieser war geeignet um sich mit den speziellen Eigenheiten der Mikrocontroller Programmierung vertraut zu machen.

2.4.1. Entwicklerboard STK500

Um den Mikrocontroller zu programmieren und die Programmierung zu überprüfen, wird das Entwicklerboard STK500[A.5], wie auf Abbildung 2.7 zu sehen, verwendet. Das Board enthält mehrere Mikrocontroller-Steckplätze, 2 serielle Schnittstellen, 8 Taster, 8 LEDs, 2 Erweiterungssports, eine ISP³ Programmierschnittstelle und mehrere Anschlussmöglichkeiten.

³In System Programmer

2. Vorstellung der vorhandenen Hardware

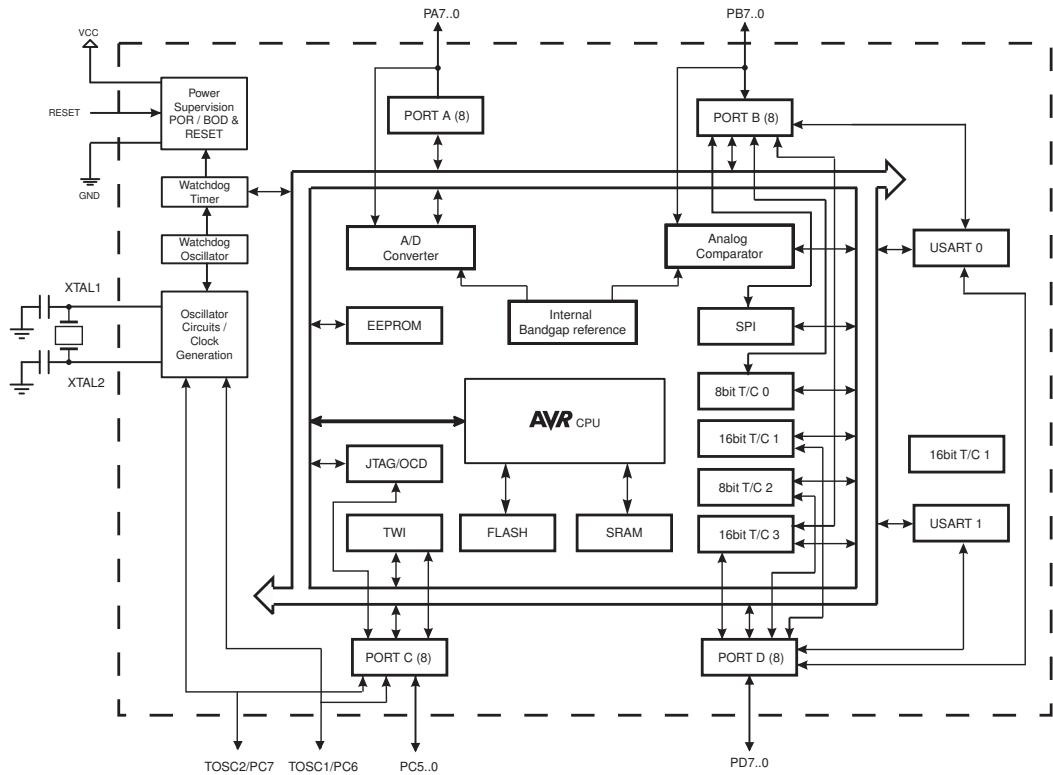


Abbildung 2.6.: Block Diagramm: Mikrocontroller ATmega324A[2]

reere 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 Stifteleisten zur Verfügung. Diese sind direkt mit den Ein- und Ausgabe Pins, den sogenannten *Ports*, des Mikrocontroller verbunden und können über Flachbandkabel mit Hardwarekomponenten wie z.B. Taster, LED, LC-Displays oder seriellen Schnittstellen verbunden werden.

2.4.2. AVRISP mkII

Der *AVRISP mkII*[A.5] ist ein USB-basierter In-System-Programmer. Dieser kann anstelle des RS-232 basierten Programmiersystem 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 Stifteleiste, des STK500 anschließen.

2. Vorstellung der vorhandenen Hardware

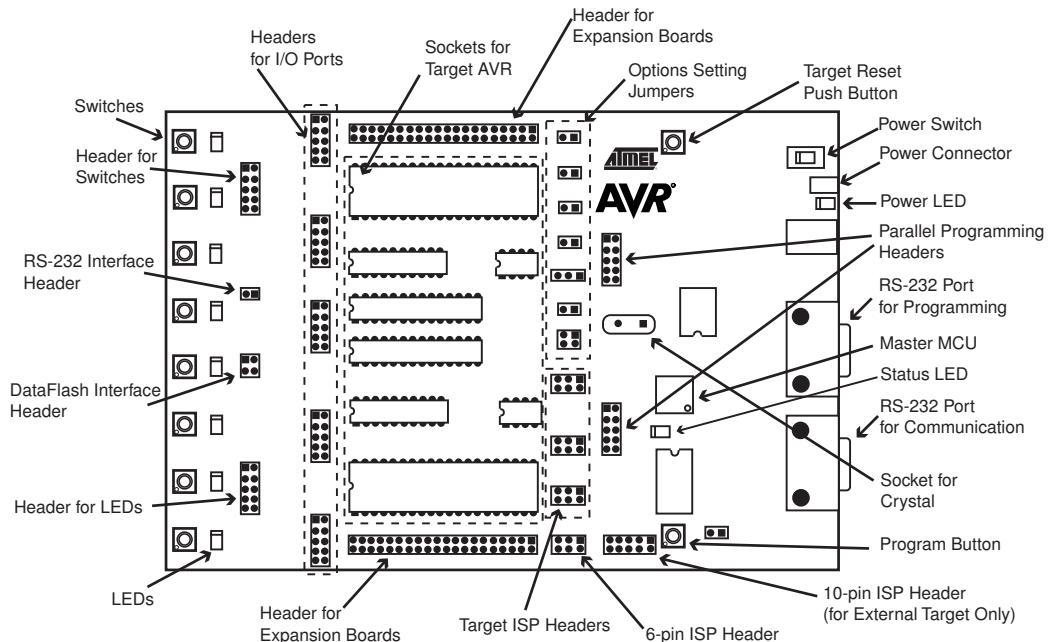


Abbildung 2.7.: Schemazeichnung eines STK500[1]

2.4.3. MAX232

Um die serielle Schnittstelle am Mikrocontroller nutzen zu können, müssen die Spannungspegel auf die des RS-232 Standards gewandelt werden. Dazu befindet sich auf dem STK500 der Pegelwandler MAX232. Dieser wandelt die Spannungspegel des Mikrocontrollers (typ. 0 V – 5 V TTL⁴) auf die Spannungspegel des RS-232 Standards (typ. -12 V – +12 V).

⁴Transistor-Transistor-Logik

3. Vorstellung der vorhandenen Software

3. Vorstellung der vorhandenen Software

3.1. RapidForm2004

Zur Erfassung von 3D-Modellen am PC steht die Software *RapidForm2004* [A.6] 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.

Mittels eines Add-In kann der VI-900 angesteuert und aufgenommenen Daten ausgelerzen werden. Weiterhin kann das Add-In über eine RS-232-Schnittstelle verschiedene Drehtische ansteuern.

3.2. Entwicklungsumgebung

Die von Atmel bereitgestellte Entwicklungsumgebung *AVR Studio 5* [A.6] besteht aus einem Editor, einem Compiler und einer Programmiersoftware. Der Editor bietet Komfortfunktionen wie Syntaxhighlighting, Auto vervollständigung und Projektmanagement. Der Compiler übersetzt den Quelltext in einen maschinenlesbaren Code und die Programmiersoftware kann diesen auf einen Mikrocontroller spielen.

3.3. Terminalprogramme

Zur Kommunikation über die RS-232-Schnittstelle steht das Programm *Hypterminal* [A.6] zur Verfügung. Dieses wurde im Verlauf des Projekts durch das wesentlich umfangreichere Open Source Programm *Putty* [A.6] abgelöst.

4. Zeitlicher Arbeitsablauf

Dieses Kapitel spiegelt den chronologischen Ablauf des Projektes wieder und zeigt die Schritte auf, die notwendig waren um den Mikrocontroller so zu programmieren, dass dieser die Kommunikation zwischen RapidForm2004 und der vorhandenen Schrittmotorkarte ermöglicht. So das er sozusagen als Übersetzer für die unterschiedlichen Befehlssätze von RapidForm2004 und dem der Schrittmotorkarte fungiert.

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 [A.7](#). Der komplette Code, mit allen Bibliotheken, liegt dem Praxisbericht als CD oder Archiv bei.

Kapitel [4.1](#) beschreibt die Programmierung des Mikrocontrollers welche die notwendigen Grundvoraussetzungen für dieses Projekt schafft. Das Ziel dieser Programmierung besteht darin die geforderten Komponenten, LEDs, LC-Display, Taster und serielle Schnittstellen im Mikrocontroller nutzbar zu machen.

Kapitel [4.2](#) beschreibt die Erarbeitung der Befehlssätze die die Software RapidForm2004 enthält um mit den von ihr unterstützten Schrittmotorkarten zu kommunizieren. Auch der Befehlssatz zur Kommunikation zwischen dem Mikrocontroller und der Schrittmotorkarte wird beschrieben.

Kapitel [4.3](#) beschreibt wie der Mikrocontroller diesen Befehlssatz für die Kommunikation mit der vorhandenen Schrittmotorkarte nutzt.

Kapitel [4.4](#) gibt die Schritte zur Entwicklung und Verbesserung der Hardware, um diese so zu erweitern, dass sie den Vorgaben entspricht, wieder.

Kapitel [4.5](#) erläutert die Kommunikation zwischen dem Mikrocontroller und RapidForm2004.

Kapitel [4.6](#) beschreibt, wie die vorherigen Kapitel zusammenspielen, sodass eine reibungslose Kommunikation zwischen RapidForm2004 und der vorhandenen Schrittmotorkarte möglich ist.

Kapitel [4.7](#) beschreibt dann das Erstellen des Platinenlayouts und das Fertigen des Einschubs.

4. Zeitlicher Arbeitsablauf

4.1. Bereitstellen grundlegender Funktionalitäten

Im ersten Schritt ging es darum, den Mikrocontroller so zu programmieren, dass dieser die für dieses Projekt grundlegenden Funktionalitäten bereitstellen kann.

Der Mikrocontroller befand sich vorerst auf dem STK500 (siehe Kapitel 2.4.1). Um dessen Komponenten im Mikrocontroller nutzen zu können, müssen dafür Register initialisiert werden oder Funktionalitäten wie z.B. Bibliotheken für das LC-Display bereit gestellt werden.

Die folgenden Kapitel beschreiben den Programmcode, der notwendig ist um diese Funktionalitäten bereitzustellen.

4.1.1. Taster

Um die Taster des STK500 im Mikrocontroller nutzen zu können müssen diese mit dem Mikrocontroller verbunden und entprellt⁵ werden.

Dazu muss die Stiftleiste von *PortA* mit der Stiftleiste für die Taster verbunden werden. Das Entprellen der Taster wird softwareseitig realisiert. Dies bietet sich bei einem Mikrocontroller an. Dazu gibt es vorgefertigte Bibliotheken die genutzt werden können. Im Projekt wurde die Bibliothek `Debounce.h` [4] von Peter Dannegger genutzt. Sie ist sehr komfortabel und funktionsreich und basiert auf Timer-Interrupts. Um sie zu nutzen wird die Datei `Debounce.h` in das Projektverzeichnis kopiert und mit Zeile 1 des Codelisting 4.1 in das Programm eingebunden. Die Zeilen 2-10 spiegeln die Funktion zum Initialisieren der Bibliothek wieder. Diese Zeilen müssen auf den jeweils verwendeten Mikrocontroller angepasst werden.

Durch die Verwendung der Bibliothek ist es möglich Funktionen wie z.B. `get_key_press()` zu nutzen um den Status der Taster prellfrei auszulesen und diese Information für Entscheidungen im Programmablauf zu verwenden.

Listing 4.1: Taster

```

1 #include "Debounce.h"                                // Taster entprellen
2 void    debounce_init      (void) {                  // Taster entprellen
3     KEY_DDR &= ~ALL_KEYS;                          // configure key port for input
4     KEY_PORT |= ALL_KEYS;                          // and turn on pull up resistors
5     TCCR0B = (1 << CS02) | (1 << CS00); // divide by 1024
6     // preload for 10ms
7     TCNT0 = (uint8_t) (int16_t) -(F_CPU / 1024 * 10 * 10e-3 + 0.5);
8     TIMSK0 |= 1 << TOIE0;                         // enable timer interrupt
9     sei();                                         // global enable Interrupts
10 }
11 if (get_key_press(1 << KEY4))
12     menu_select(&menu_context);                // Aktuellen Menuepunkt auswaehlen

```

⁵ Als Prellen bezeichnet man das ungewollte mehrfache Schalten eines mechanischen Schalters bei einem einzelnen Schaltvorgang.

4. Zeitlicher Arbeitsablauf

4.1.2. LEDs

LEDs sollen im Programmablauf genutzt werden können, um z.B. Fehler zu signalisieren.

Dazu muss zuerst die Stifteiste von *PortB* mit der LED Stifteiste des STK500 verbunden werden. Um LEDs an *PortB* betreiben zu können müssen die entsprechenden Pins im Register `DDR` als Ausgänge definiert werden. Dies geschieht in Zeile 1 des Codelisting 4.2. Die Bibliothek zum Entprellen der Taster nutzt die Variablen `LED_DDR` und `LED_PORT`. Diese Variablen werden auch hier genutzt um auf die Register zuzugreifen. Dies gewährleistet eine bessere Übersicht. 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) oder einem 8 stelligen binären Bitmuster beschrieben werden. In Zeile 2 werden alle Bits mit dem Hex-Wert `0xFF` auf "1" gesetzt und somit alle LEDs ausgeschaltet. Um ein einzelnes Bit zu verändern, können die Anweisungen in den Zeilen 3 und 4 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 LED_DDR = 0xFF;           // LED Port Richtung definieren (Ausgang)
2 LED_PORT = 0xFF;          // LEDs ausschalten
3 LED_PORT &= ~((1 << PBX)); // loescht Bit an PortB – LED an
4 LED_PORT |= ((1 << PBX)); // setzt Bit an PortB – LED aus

```

4.1.3. Ansteuerung des LC-Display

Um den aktuellen Status des Motor komfortabel in Textform anzeigen zu können und die Schrittmotorkarte menübasiert ansteuern zu können wird ein LC-Display verwendet. Das verwendete Display ist alpha numerisch und kann 4x20 Zeichen anzeigen.

Die meisten LC-Displays werden auf die gleiche Weise angesteuert. Hier gibt es fertige Bibliotheken die frei genutzt werden können. Im Projekt wurde die Bibliothek von Peter Fleury [5] verwendet. Die Bibliothek wird heruntergeladen und die Dateien `lcd.c` und `lcd.h` in das Projektverzeichnis entpackt. Die Bibliothek wird mit `#include 'lcd.h'` eingebunden. In der `lcd.h` müssen dann noch die Daten des Displays eingegeben werden (siehe Codelisting 4.3 Zeilen 2–10).

Danach kann das Display im Programm mit den Befehlen aus Zeile 12–21 angesteuert werden.

4. Zeitlicher Arbeitsablauf

Listing 4.3: lcd.h (Auszug)

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

4.1.4. RS-232-Schnittstelle

RS-232 ist der Name der am häufigsten verwendeten seriellen Schnittstelle um Daten zwischen zwei elektronischen Geräten hin und her zu senden. [7]

Auf dem STK500 ist bereits eine serielle Schnittstelle vorbereitet. Um diese nutzen zu können, müssen die Pins 3 und 4 des *PortC* (erster **UART**) des Mikrocontrollers mit der Stifteleiste *Rx/Tx* auf dem STK500 verbunden werden. Eine weitere Schnittstelle wurde auf einem Steckbrett aufgebaut. Diese wurde mit den Pins 1 und 2 des *PortC* (zweiter **UART**) des Mikrocontrollers verbunden. Um die Schnittstellen im Mikrocontroller nutzen zu können müssen diese noch durch setzen von Bits in den entsprechenden Registern des Mikrocontrollers aktiviert werden.

Das Codelisting 4.4 teilt sich in 4 wesentliche Bereiche:

- Zeilen 1 – 2: Setzen der Baudrate und einbinden der benötigten Bibliotheken.
- Zeilen 3 – 17: Initialisieren der Schnittstellen durch setzen der richtigen Bits in den entsprechenden Registern.
- Zeilen 18 – 35: Funktionen zum Senden von Daten
- Zeilen 36 – 65: Funktionen zum Empfangen von Daten

4. Zeitlicher Arbeitsablauf

Listing 4.4: RS-232

```

1 #define BAUD 9600           // BAUD Rate definieren
2 #include <util/setbaud.h>    // UART Funktionen
3 // UART Initialisieren
4 void  uart_init() {
5     // UART 0 – IN (Rapidform Software/Terminal)
6     UBRR0H = UBRRLH_VALUE;
7     UBRR0L = UBRRL_L_VALUE;
8     UCSZ0C = (3 << UCSZ00);
9     UCSZ0B |= (1 << TXEN0); //Transmitter Enabled
10    UCSZ0B |= (1 << RXEN0); // UART RX einschalten
11    // UART 1 – OUT (Stepper Karte/Drehtisch)
12    UBRR1H = UBRRLH_VALUE;
13    UBRR1L = UBRRL_L_VALUE;
14    UCSZ1C = (3 << UCSZ00);
15    UCSZ1B |= (1 << TXEN1); //Transmitter Enabled
16    UCSZ1B |= (1 << RXEN1); // UART RX einschalten
17 }
18 // UART Zeichen senden
19 void  uart_put_charater(unsigned char c, int dir) {
20     if (dir == D_RapidForm) { // To Rapidform
21         while (!(UCSR0A & (1 << UDRE0))) {} //warten bis Senden moeglich
22         UDR0 = c; // sende Zeichen
23     }
24     else { // To Stepper
25         while (!(UCSR1A & (1 << UDRE1))) {} //warten bis Senden moeglich
26         UDR1 = c; // sende Zeichen
27     }
28 }
29 // UART String senden
30 void  uart_put_string(char *s, int dir) {
31     while (*s){ // so lange *s != '\0' Terminierungszeichen
32         uart_put_charater(*s, dir); // Zeichenweise senden
33         s++;
34     }
35 }
36 // UART Zeichen empfangen
37 int   uart_get_character(int dir) {
38     if (dir == D_RapidForm) { // Aus RapidForm Register auslesen
39         while (!(UCSR0A & (1 << RXC0))) ; // warten bis Zeichen verfuegbar
40         return UDR0; // Zeichen aus UDR an Aufrufer zurueckgeben
41     }
42     if (dir == D_Stepper) { // Aus Schrittmotor Register auslesen
43         while (!(UCSR1A & (1 << RXC1))) ; // warten bis Zeichen verfuegbar
44         return UDR1; // Zeichen aus UDR an Aufrufer zurueckgeben
45     }
46     return -1; // Wenn nichts ausgelesen wurde -1 zurueckgeben
47 }
```

4. Zeitlicher Arbeitsablauf

```

48 // UART String empfangen
49 void uart_get_string (char * string_in, int dir) {
50     char c;           // Einzelnes Zeichen
51     int i = 0;        // Zaehlvariable
52     do {
53         c = uart_get_character(dir); // Einzelnes Zeichen holen
54         if (c != '\r') {           // Wenn keinn \r
55             *string_in = c;        // Zeichen in Empfangsstring schreiben
56             string_in += 1;       // Adresse des Empfangsstring um 1 ink
57             i++;                 // Zaehlvariable um 1 erhoehen
58         }
59     } while (i < 100 && c != '\r' && c != '\n'); // So lange bis \r \n o. >100 Zeichen
60     *string_in = '\0';            // 0 Terminieren
61     if (dir == D_Stepper)
62         LED_PORT |= ( 1 << LED3 ); // "Daten Vorhanden" LED ausschalten
63     else
64         LED_PORT |= ( 1 << LED2 ); // "Daten Vorhanden" LED ausschalten
65 }
```

Damit stehen die essentiellen Funktionen `uart_put_string(dir)` und `uart_get_string(dir)` zur Verfügung. Mit diesen kann der Mikrocontroller über die serielle Schnittstelle Strings senden und empfangen. Der Parameter `dir` gibt dabei die Schnittstelle an über die gesendet oder empfangen werden soll.

4.2. Befehlssätze

Das zu erreichende Ziel bestand darin, dass RapidForm2004 mit dem Mikrocontroller und dieser mit der Schrittmotorkarte kommunizieren können sollte. Die Kommunikation läuft dabei über Befehle ab, die über die serielle Schnittstelle gesendet werden. Jede Schrittmotorkarte verwendet eigenen Befehle. Alle Befehle für eine Schrittmotorkarte werden im Folgenden als Befehlssatz bezeichnet. Die Software RapidForm2004 kennt mehrere Befehlssätze um verschiedene Schrittmotorkarten anzusteuern. Der Befehlssatz der vorhandenen Schrittmotorkarten zum Ansteuern der Motoren des Drehtisches ist jedoch nicht in RapidForm2004 vorhanden.

Nun soll der Mikrocontroller sowohl mit RapidForm2004 als auch mit der ersten der vorhandenen Schrittmotorkarten kommunizieren. Befehle an die zweite Schrittmotorkarte werden über die Erste gesendet. Um mit beiden Seiten kommunizieren zu können muss der Mikrocontroller den Befehlssatz der vorhandenen Schrittmotorkarten und zumindest einen der Befehlssätze aus RapidForm2004 kennen. Außerdem muss er wissen welche Antwort RapidForm2004 auf einen gesendeten Befehl erwartet.

In der ersten Phase wurde die Software *Free Serial Port Monitor* verwendet um die Kommunikation zwischen RapidForm2004 und dem Mikrocontroller abzuhören. Dies hatte jedoch den Nachteil, das RapidForm2004 erst dann den nächsten Befehl

4. Zeitlicher Arbeitsablauf

sendete, wenn der Erste mit der erwarteten Antwort quittiert wurde. Die Befehle die RapidForm erwartete, konnten zwar teilweise aus den Betriebsanleitungen der Schrittmotorsteuerungen entnommen werden, dieses Vorgehen war jedoch sehr mühsam. Eine besseres Vorgehen, war das sogenannte **Reverse-Engineering**. Dadurch konnten alle Befehle und die darauf erwarteten Antworten aus einer ausführbaren Datei von RapidForm2004 ausgelesen werden.

Das Codelisting 4.5 zeigt einen Auszug für den Befehlssatz eines Isel Schrittmotors. Im Anhang A.3 befinden sich die Befehlssätze aller Schrittmotorkarten. Somit stehen die Befehlssätze aller Schrittmotorsteuerungen zur Verfügung. Diese wurden in einer Textdatei gespeichert und werden später im Programm verwendet. Dadurch sind alle Befehle und die Antworten die RapidForm2004 auf einen daraus ausgesendeten Befehl erwartet bekannt.

Hinweis

In Codelistings und im Quelltext wird teilweise noch die Bezeichnung *Protokolle* statt *Befehlssätze* verwendet. Diese sind gleichbedeutend.

Listing 4.5: Befehlssatz aus Rapidform: Isel

```

1 model "isel (RF-1)"
2 port "9600"           "n81h"
3 init "@01\r"          "0"
4 finish "@0M0\054+600\r" "0"
5 arot "@0M%d\054+600\r" "0"
6 stop ""
7 home "@0M0\054+600\r" "0"
8 step "-0.0173076" "-8000000" "8000000"
9 timeout "60"
10 firsttimeout "10"
```

4.3. Kommunikation mit der vorhandenen Schrittmotorsteuerung

4.3.1. Befehle senden

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, wird eine menübasierte Steuerung mittels des LC-Displays verwendet. Im Menü lässt sich mit den Tasten *Hoch*, *Runter*, *Ok*, und *Zurück*, navigieren.

Analog wie beim LC-Display und bei den Tastern wird hier eine vorhandene Bibliothek genutzt. Um die Bibliothek verwenden zu können musste die Menüstruktur

4. Zeitlicher Arbeitsablauf

den Bedürfnissen des Projekts angepasst werden und die Funktionen zum Ausgeben von Text auf dem LC-Display und zum Versenden von Befehlen über die RS-232-Schnittstelle, aus den vorangegangen Kapiteln, bekannt gemacht werden. Dies geschieht in der Datei `tinymenu/tinymenu.h`.

Die Zeilen 1–6 des Codelisting 4.6 dienen zum Einbinden der benötigten Bibliotheken. Die Zeilen 8–20 zeigen eine vereinfachte Struktur des 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"
5 #include "tinymenu/tinymenu_hw.h"
6 #include "mymenu.h"
7 // Gekuerzte Main-Funktion
8 int main(void) {
9     while (1) {      // In Endlosschleife wechseln
10         wdt_reset(); // Watchdog zuruecksetzen
11         if (get_key_press(1 << KEY1)) // 1 – Zurueck
12             menu_exit(&menu_context);
13         if (get_key_press(1 << KEY2)) // 2 – Hoch
14             menu_prev_entry(&menu_context);
15         if (get_key_press(1 << KEY3)) // 3 – Runter
16             menu_next_entry(&menu_context);
17         if (get_key_press(1 << KEY4)) // 4 – Ok
18             menu_select(&menu_context);
19     }
20 }
21 // Funktion zum senden der Menuepunkte ueber die serielle Schnittstelle
22 void menu_puts (void *arg, char *name) { // Menu/Sende Funktion
23     uart_put_string(arg, D_Stepper); // Uebergebenen String an Stepper senden
24     // Befehl auf Display ausgeben
25     lcd_clrscr();
26     lcd_puts("Sent: ");
27     lcd_puts(arg);
28     lcd_puts("\n");
29     ms_spin(100);
30     //if ((UCSR1A & (1 << RXC1)))
31     uart_rx(D_Stepper); // Antwort des Stepper empfangen
32     ms_spin(1000); // Antwort noch eine weile anzeigen
33 }
```

Folgende Menüpunkte wurden realisiert:

Listing 4.7: Menü Baum

4. Zeitlicher Arbeitsablauf

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

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(arg, name)` (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.

Es ist somit nun möglich mit Tasten vordefinierte Befehle aus dem Menü auszuwählen und an die Schrittmotorsteuerung zu senden.

4.3.2. Antworten empfangen und speichern

Die Schrittmotorsteuerung antwortet auf Befehle mit einem `String`. In diesem Arbeitsschritt wird die Funktionalität zum Empfangen von Antworten der Schrittmotorsteuerung auf Befehle des Mikrocontrollers hergestellt. Diese Antworten sollen in einem String gespeichert und im nächsten Schritt an eine *Auswerte-Funktion* weitergegeben werden.

Dazu wird in der Hauptschleife des Programms ständig das Eingangsregister der ersten seriellen Schnittstelle abgefragt (siehe Codelisting 4.8 Zeile 10–13). Dieses Vorgehen bezeichnet man als `Polling`. Sind Daten im Register vorhanden, wird `LED3` eingeschaltet und die Funktion `uart_rx(int dir)` mit dem Parameter `D_Stepper` aufgerufen. Der übergebene Parameter gibt an, dass der Befehl von der für die Schrittmotorkarte zuständigen Schnittstelle empfangen wurde. Dadurch wird sichergestellt, dass der empfangene `String` aus dem richtigen Datenempfangsregister ausgelesen wird und festgelegt wie er weiterverarbeitet wird. Die Funktion `uart_rx(dir)`

 4. Zeitlicher Arbeitsablauf

liest dann das Empfangsregister mit der aus Kapitel 4.1.4 bekannten Funktion `uart_get_string(str_rx, dir)` aus und schreibt den empfangenen String in die Variable `str_rx` (Codelisting 4.8, Zeile 7). In einer `if`-Abfrage wird entschieden von welcher Schnittstelle der empfangene Befehl kam. Da `D_Stepper` übergeben wurde, wird der if-Teil der Abfrage ausgeführt. In dieser wird der empfangene String an die *Auswerte-Funktion* für die Schrittmotorkarte (Codelisting 4.8, Zeile 15-45) übergeben. Durch diesen Teil des Programms ist es nun möglich Antworten der Schrittmotorkarte zu empfangen, in dem String `str_rx` zu speichern und an die Auswerte-Funktion `switch_Stepper(str_rx)` zu übergeben.

Listing 4.8: RS-232 Empfang

```

1 if ((UCSR1A & (1 << RXC1))){      // Stepper Polling
2     LED_PORT &= ( 1 << LED3 );        // LED einschalten
3     uart_rx(D_Stepper);              // Register auslesen
4 }
5 // UART Empfangsregister auslesen
6 void    uart_rx           (int dir) {
7     uart_get_string(str_rx, dir);   // String aus Empfangsregister auslesen
8     if (dir == D_Stepper)          // Empfangsregister Stepper
9         switch_Stepper(str_rx); // Uebersetzungsfunktion fuer Stepper aufrufen
10    else{                         // Empfangsregister RapidForm
11        // Wird spaeter erklaert
12    }
13 }
```

4.3.3. Antworten auswerten

Die Funktion zum Auswerten empfangener Strings spielt eine zentrale Rolle im Projekt. Diese Funktion ermöglicht es, ankommende Strings im Mikrocontroller gegen die bekannten Antworten zu prüfen und eine entsprechende Reaktion auszuführen.

In der Auswerte-Funktion wird der übergebene String mittels der Funktion

`FindStringInArray(str_rx, pOptions, length)` (Codelisting 4.9) gegen ein `Array` (Codelisting 4.10, Zeile 3) mit bekannten Befehlen geprüft. Ist der String in diesem Array vorhanden, wird die Position des Strings im Array zurückgegeben, ansonsten wird "99" zurückgegeben. In einer anschließenden `switch/case`-Struktur wird dann der Position im Array ein bestimmtes Verhalten des Mikrocontrollers zugeordnet. Wird beispielsweise der String `#` empfangen, wird Position `0` zurück gegeben und auf dem LC-Display wird *Erfolgreich* ausgegeben.

Durch diese Funktion kann nun auf Strings reagiert werden und eine entsprechende Reaktion seitens des Mikrocontrollers erfolgen.

Listing 4.9: FindStringInArray()

4. Zeitlicher Arbeitsablauf

```
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 (!strncmp(pInput, pOptions[n], cmp_length)) return n;  
6     }  
7     return 99; // Wenn keine uebereinstimmung, gib 99 zurueck  
8 }
```

Listing 4.10: switchStepper()

```
1 // Uebersetzung Schrittmotorkarte  
2 void    switch_Stepper      (char * str_rx) {  
3     const char* pOptions[] = {      // Array mit bekannten Befehlen  
4         "#",    // 0 – Stepper Karte hat Befehl erkannt  
5         "E",    // 1 – Stepper Karte meldet Error  
6         "!CLS", // 2 – Clear Screen (Debugging)  
7         "Test", // 3 – Test (Debugging)  
8         0 };  
9     switch (FindStringInArray(str_rx, pOptions, 1)) { // String gegen bekannte Antworten  
10        pruefen  
11        case 0:           // 0 – Stepper Karte hat Befehl erkannt  
12            lcd_puts("Erfolgreich\n");  
13            break;  
14        case 1:           // 1 – Stepper Karte meldet Error  
15            lcd_puts("Error\n");  
16            uart_put_string("1\r\n", D_RapidForm);  
17            break;  
18        case 2:           // 2 – Clear Screen (Debugging)  
19            lcd_clrscr();  
20            break;  
21        case 3:           // 3 – Test (Debugging)  
22            lcd_puts("Test bestanden\n");  
23            break;  
24        default:  
25            ms_spin(10);  
26    }
```

4.4. Verbesserungen an der vorhandenen Hardware

4.4.1. Netzteil

Ziel dieses Arbeitsschrittes war es, die festen Lötverbindungen zwischen dem PC-Netzteil und den einzelnen Karteneinschüben im 19"-Rack durch Steckverbindungen zu ersetzen und dadurch leicht erweiterbar zu machen.

4. Zeitlicher Arbeitsablauf

Die festen Lötverbindungen am Einschub für die Schrittmotorkarte wurden durch Standard PC-Netzteil Stecker ersetzt. Die Logikbausteine der Schrittmotorkarte werden mit $5V$ gespeist. Die Schrittmotorkarte wird zusätzlich mit $12V$ für den Schrittmotor gespeist. Der Stecker lässt sich nun einfach mit einer Buchse des Standard PC-Netzteils verbinden und es ist nicht mehr Notwendig zu löten wenn das Netzteil ausgebaut wird. Mittels eines Y-Kabels(siehe Abbildung 4.1) können nun leicht weitere Buchsen hinzugefügt werden.

Dadurch kann das Netzteil nun einfach ein- und ausgebaut werden, bzw. das System leicht um neue Einschubkarten erweitert werden.



Abbildung 4.1.: Stromverbinder - Y-Kabel[6]

4.4.2. Zweite Schrittmotorkarte

Zu Anfang war nur eine Schrittmotorkarte für die Rotation des Drehtisches vorbereitet. Mit einem zweiten Schrittmotor konnte der Tisch in der Höhe verstellt werden. Für diesen fehlte jedoch noch eine zweite Schrittmotorkarte. Diese musste noch vorbereitet und mit der Ersten verbunden werden.

Dazu wurde, wie in Kapitel 4.4.1 beschrieben, ein weiterer Einschubplatz für die Schrittmotorkarte vorbereitet. Die Karte wurde mit einer Frontblende versehen und auf dieser eine Buchse für die Motorverkabelung und je eine Buchse und einen Stecker für die seriellen Schnittstellen verbaut. Diese wurden mit den entsprechenden Anschlüssen auf der Schrittmotorkarte verlötet. Die Karte wird in den Einschubplatz geschoben und mit einem seriellen Kabel als Daisy-Chain mit der ersten Schrittmotorkarte verbunden. Dadurch kann die zweite Schrittmotorkarte über die Erste angesteuert werden.

Somit steht eine baugleiche Schrittmotorkarte zur Verfügung. Diese kann nun den Schrittmotor für die Höhenverstellung ansteuern. Befehle an diese Schrittmotorkarte werden an die erste Karte geschickt, jedoch mit dem Prefix 2. Dieser weist die erste Karte an, den Befehl an die zweite Karte weiter zu senden. So kann das System um weitere Karten erweitert werden.

*4. Zeitlicher Arbeitsablauf***4.4.3. Motor- und Endschalterverkabelung**

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 wurde in der Werkstatt des RheinAhrCampus Remagen ein 7adriges Kabel (siehe Abbildung 4.2) besorgt und die passenden Endstecker bestellt. Die Belegung wurde gleich zum Kabel für den ersten Schrittmotor gewählt, jedoch um die 3 Adern für die beiden Endschalter erweitert. 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 an die Schrittmotorkarte übermittelt werden konnte. (**(TODO: BELEGUNG ÜBERPRÜFEN!)**)

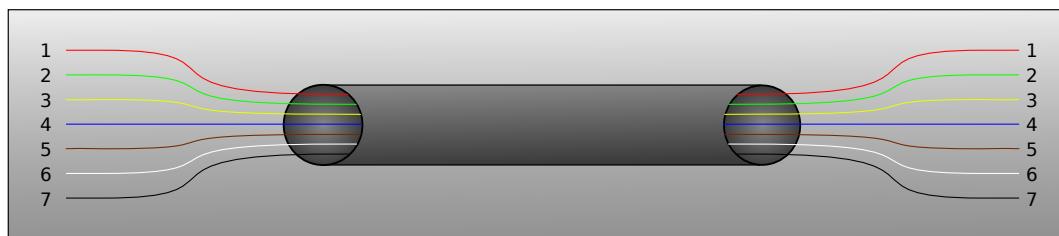


Abbildung 4.2.: Motor- und Endschalterverkabelung

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

Nun geht es darum, die vorgegebenen induktiven Endschalter mit der Schrittmotorkarte und dem Mikrocontroller zu verbinden. Dadurch soll gewährleistet werden, dass der Drehtisch nicht über den Arbeitsbereich hinaus bewegt werden kann. Zusätzlich soll das Erreichen der Endpositionen auf dem LC-Display angezeigt werden.

Da die Schrittmotorkarte nur mechanische Endschalter unterstützt, ließen sich die induktiven Endschalter nicht ohne weiteres nutzen. Um die induktiven Endschalter nutzen zu können, musste die Spannung über einen Spannungsteiler heruntergesetzt

4. Zeitlicher Arbeitsablauf

werden und die standardmäßigen Eingänge für die mechanischen Endschalter umgangen werden. Die induktiven Endschalter werden direkt an den Optokoppler angeschlossen, welcher für die mechanischen Endschalter zuständig ist. Dadurch wurden die Signale der Endschalter für die Schrittmotorkarte nutzbar. Ein weiteres Problem

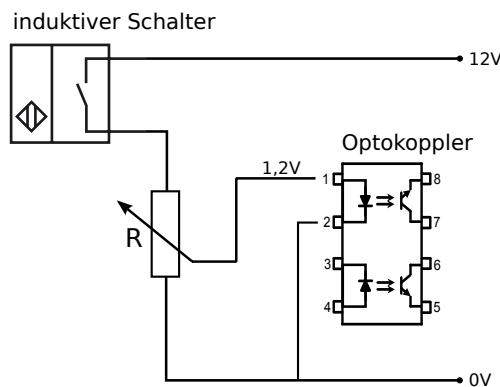


Abbildung 4.3.: Endschalterverkabelung

bestand darin, dass, wenn der Tisch sich bereits in der Endposition befand, die Endschalter noch nicht aktiviert wurden. Dies lag daran, dass der Metallstutzen, der die Endschalter auslösen sollte, sich nicht im Schaltbereich der induktiven Schalter befand. Zur Abhilfe wurde ein längerer Metallstutzen von der Werkstatt des RheinAhrCampus angefertigt.

Wenn der Tisch sich in der Endposition befindet, soll dies auch auf dem LC-Display angezeigt werden. Die Signale der Endschalter liegen auf der Rückseite der Schrittmotorkarte am Verbindungsstecker an. Um die Signale dem Mikrocontroller zugänglich zu machen wurde eine Brücke zwischen den Verbindungssteckern der Schrittmotorkarte und der Mikrocontroller-Platine gelötet. Auf der Mikrocontroller-Platine sind diese beiden Pins mit je einem Pin des Mikrocontrollers verbunden. Diese beiden Pins werden im Mikrocontroller als `Interrupts` definiert. Die `Interrupt-Service-Routine` zum Anzeigen der Nachricht auf dem LC-Display wird in Kapitel 5.2.1 beschrieben. Da die Signale der Endschalter nun an der Schrittmotorkarte anliegen, stoppt diese den Motor wenn eine der Endschalterpositionen erreicht wird. Zusätzlich liegen die Signale am Mikrocontroller an. Dieser gibt dadurch auf dem Display die Meldung *Endschalterposition erreicht!* aus.

4.4.5. Zweite serielle Schnittstelle

Das STK500 bietet nur eine serielle Schnittstelle. Um zusätzlich zur Schrittmotorkarte auch mit RapidForm2004 kommunizieren zu können, wird eine zweite RS-232-Schnittstelle benötigt.

Dafür wurde vorerst auf einem Steckbrett eine zweite serielle Schnittstelle nach dem

4. Zeitlicher Arbeitsablauf

Schaltplan in Abbildung 4.4 aufgebaut. Später wird diese Schnittstelle direkt auf der Mikrocontroller-Platine realisiert. Dadurch ist es möglich mit dem Mikrocontroller über zwei RS-232-Schnittstellen gleichzeitig zu kommunizieren.

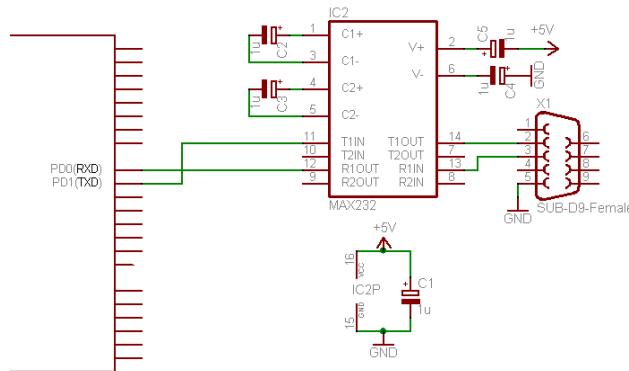


Abbildung 4.4.: Schaltplan für die zweite serielle Schnittstelle [7]

4.5. Kommunikation mit RapidForm2004

RapidForm2004 sendet Befehle die für die Drehtischsteuerung bestimmt sind an den Mikrocontroller. Diese sollen dort empfangen, ausgewertet und in verständlicher Form an die Drehtischsteuerung weiter gegeben werden. RapidForm2004 verwendet dabei verschiedene Befehlssätze für verschiedene Schrittmotorsteuerungen. Für jeden dieser Befehlssätze wird eine eigene Auswerte-Funktion geschrieben. Im folgenden Kapitel wird nun das Empfangen der Befehle beschrieben und eine erste Auswertung, die den empfangenen Befehl dem Befehlssatz einer Schrittmotorsteuerung zuordnet. Nachdem ein Befehl zugeordnet wurde und in der entsprechenden Auswerte-Funktion erkannt wurde, soll ein entsprechender Befehl an die Drehtischsteuerung gesendet und die Antwort der Drehtischsteuerung vom Mikrocontroller ausgewertet werden. Abschließend soll eine entsprechende Antwort an RapidForm2004 zurück gesendet werden. Abbildung 4.5 zeigt eine schematische Übersicht dieser Kommunikation. Die Kommunikation mit RapidForm2004 ist ähnlich zu der mit der Schrittmotorsteuerung. Diese wurde bereits in Kapitel 4.3 ausführlich beschrieben. Daher wird die Kommunikation hier etwas oberflächlicher behandelt.

4.5.1. Befehle empfangen

Zuerst sollen nun die Befehle von RapidForm2004 an den Mikrocontroller, gespeichert werden. Anschließend wird die automatische Auswahl des Befehlssatzes beschrieben. Um anstehende Befehle zu empfangen wird, ähnlich wie in Kapitel 4.3.2, eine Funktion die ständig das Eingangsregister der ersten seriellen Schnittstelle abfragt verwenden.

ÜBERSETZEN VON SCHRITTMOTORBEFEHLEN

Entwurf eines Hardwareübersetzers

4. Zeitlicher Arbeitsablauf

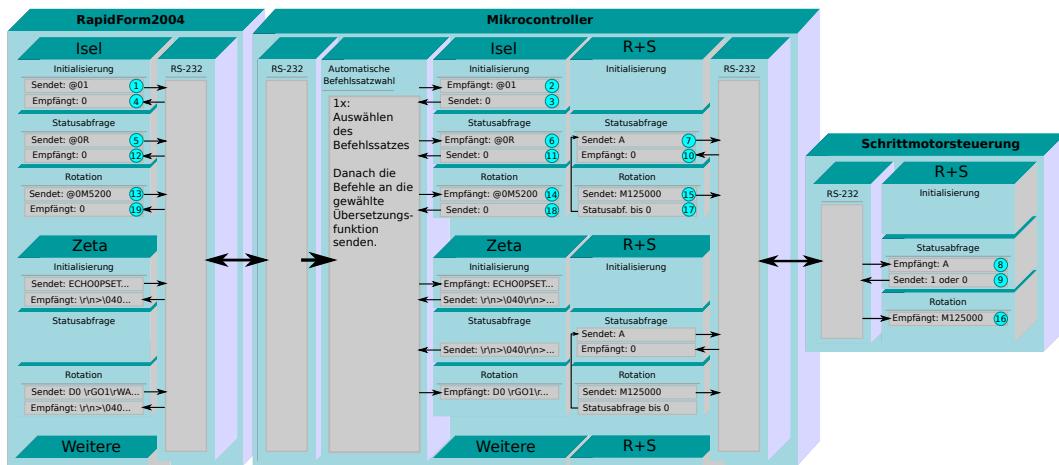


Abbildung 4.5.: Schema der Kommunikation

det (siehe Codelisting 4.11). Auch hier wird die Funktion `uart_rx(dir)` aufgerufen, jedoch mit dem Parameter `D_RapidForm`. Der empfangenen String wird auch hier in die Variable `str_rx` gespeichert. Somit können nun auch Strings von RapidForm2004 empfangen und in der Variablen `str_rx` gespeichert werden.

Listing 4.11: RS-232 Empfang - RapidForm2004

```

1 if ((UCSR0A & (1 << RXC0))){ // RapidForm Polling
2     LED_PORT &= ( 1 << LED2 ); // LED einschalten
3     uart_rx(D_RapidForm);      // Register auslesen
4 }
```

4.5.1.1. Automatische Auswahl eines Befehlssatzes

Nun geht es darum, dass der Mikrocontroller anhand eines ersten Befehls der empfangen wird, festlegt, mit welchem Befehlssatz fortan kommuniziert werden soll. Die Kennung für den Befehlssatz wird in einer globalen Variable gespeichert und alle weiteren Befehle werden an die entsprechende Auswerte-Funktion für diesen Befehlssatz übergeben.

In der Funktion `uart_rx(dir)` (Codelisting 4.12) wird nun in der ersten *if-Abfrage* entschieden, von welcher Schnittstelle der empfangene Befehl kam. Diese verzweigt nun, da `D_RapidForm` übergeben wurde, in den else-Teil. In diesem wird mit mehreren *if-Abfragen* überprüft, ob bereits der Befehlssatz für einen bestimmten Motor ausgewählt wurde. Ist dies nicht der Fall, wird der empfangende String an die Funktion `switch_Motor(str_rx)` (Codelisting 4.13) übergeben. Diese prüfte mit der aus Kapitel 4.3.3 bekannten Funktion `FindStringInArray(str_rx, pOptions, 3)`, den angekommenen String gegen die Initialisierungsbefehle der einzelnen Befehlssätze. Die Initialisierungsbefehle sind die ersten Befehle die RapidForm2004 an eine Schritt-

4. Zeitlicher Arbeitsablauf

motorsteuerung sendet um zu prüfen ob diese vorhanden ist. In diesem ersten Schritt wird der String nur zur Identifizierung des von RapidForm2004 verwendeten Befehlsatzes verwendet. Das Antworten auf einen String wird erst in den nachfolgenden Kapiteln beschrieben. Die Funktion `switch_Motor(str_rx)` gibt einen numerischen Wert zurück. Jede Zahl entspricht dabei dem Befehlssatz für eine Schrittmotorsteuerung. Die Zahlenwerte werden dabei mittels Makro-Definitionen (Codelisting 4.13 Zeile 1-6) durch lesbare Text-Variablen ersetzt. Dies erhöhte die Lesbarkeit und das Verständnis. War dieser Schritt erfolgreich, wird in den folgenden if-Abfragen die richtige Auswerte-Funktion aufgerufen. Konnte die Funktion `switch_Motor(str_rx)` den empfangen Befehl nicht zuordnen, gibt sie `M_UNK` zurück und es wird auf dem Display *Unbekannter Motor!* ausgegeben.

Somit ist es möglich Befehle von RapidForm2004 zu empfangen und an die richtige Auswerte-Funktionen zu übergeben. Zusätzlich wird die Programmierung dadurch wesentlich robuster, da unbekannte Befehle ignoriert werden.

Der Nachteil dieses Vorgehens besteht darin, dass für ein wechseln des Befehlssatzes der Mikrocontroller neu gestartet werden muss. Ein Beheben dieses Nachteils wäre nicht ohne weiteres möglich gewesen.

Listing 4.12: Funktion: uart rx()

```

1 // UART Empfangsregister auslesen
2 void    uart_rx(          int dir) {
3     uart_get_string(str_rx, dir); // String aus Empfangsregister auslesen
4     if (dir == D_Stepper)        // Empfangsregister Stepper
5         switch_Stepper(str_rx); // Uebersung Stepper
6     else{                      // Empfangsregister RapidForm
7         // Uebersetzungsfunktion auswaehlen
8         if( Initialized == M_UNK){ // Unbekannter Initialisierungsbefehl
9             lcd_puts("Unbekannter Motor!\n");
10            Initialized = M_NOTI; // Variable Initialized zuruecksetzen
11        }
12        if( Initialized == M_NOTI){ // Befehlssatz bestimmen
13            Initialized = switch_Motor(str_rx); //Automatische Befehlssatzwahl
14        }
15        if( Initialized == M_ISEL) // Uebersetzung ISEL
16            switch_Isel(str_rx);
17        if( Initialized == M_CSG) // Uebersetzung CSG
18            switch_csg(str_rx);
19        if( Initialized == M_ZETA) // Uebersetzung Zeta
20            switch_Zeta(str_rx);
21        if( Initialized == M_TERMINAL) // Uebersetzung Terminal
22            switch_Terminal(str_rx);
23    }
24 }
```

4. Zeitlicher Arbeitsablauf

Listing 4.13: Funktion: switch Motor()

```

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 int     Initialized = M_NOTI;
8 // Automatische Befehlssatzwahl
9 int     switch_Motor      (char * str_rx) {
10    const char* pOptions[] = {      // Array mit Initialisierungsbefehlen
11        "@01",           // 0 – Isel
12        "Q:",            // 1 – CSG
13        "ECHO0",          // 2 – Zeta
14        "!Terminal",     // 3 – Terminal ansteuerung!
15        0 };
16    // Ankommenden String gegen Array pruefen
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 }
```

4.6. Auswerte-Funktionen

Die Auswerte-Funktionen sind das Herzstück des Programms. Es geht darum für jedes Protokoll eine eigene Auswerte-Funktion zu schreiben. Diese sollen die von RapidForm2004 kommenden Strings verstehen können und in einen, für die vorhandene Schrittmotorkarte, verständlichen Befehl übersetzen können. Die Funktionen sollen weiterhin prüfen, ob der Befehl von der Schrittmotorkarte erkannt wurde und den Status der Schrittmotorkarte zurück an RapidForm2004 melden.

Alle bisherigen Arbeitsschritte hatten zum Ziel, die Kommunikation zwischen RapidForm2004 und der ersten Schrittmotorkarte zu ermöglichen. Nun fehlt nur noch der

4. Zeitlicher Arbeitsablauf

Teil des Programms der die ankommenden Befehle auswertet und in verständlicher Form an die Schrittmotorkarte weitergibt. Im folgenden Kapitel wird dieser Ablauf nun exemplarisch für den Befehlssatz eines Isel-Motors erklärt.

4.6.1. Auswerte-Funktion für Isel-Motoren

Wird der Befehl **001** empfangen, übergibt die in Kapitel 4.5.1.1 beschriebene Funktion, den String an die Auswerte-Funktion **switch_Isel(str_rx)**. Der Ablauf dieser Funktion ist ähnlich aufgebaut wie bei der Kommunikation mit der Schrittmotorkarte (Kapitel 4.3) und bei der automatischen Auswahl des Befehlssatzes (Kapitel 4.5.1.1). In der Funktion **switch_Isel(str_rx)** sind in dem Array **pOptions** alle benötigten Befehle des Isel-Befehlssatzes hinterlegt. Mit der aus Kapitel 4.3.3 bekannten Funktion **FindStringInArray(str_rx, pOptions)** wird **str_rx** gegen diese Befehle geprüft. Wird der Befehl im Array gefunden gibt die Funktion **FindStringInArray()** die Position des Strings im Array zurück. Mittels einer switch-case-Struktur lässt sich nun so für jeden Befehl ein entsprechender Ablauf ausführen. Die einzelnen Abläufe werden übersichtlich in den folgenden Kapiteln beschrieben.

Listing 4.14: Übersetzungs Logik für einen Isel-Motor

```

1 // Uebersetzung Isel
2 void switch_Isel (char * str_rx) {
3     const char* pOptions[] = {
4         "XXXXXXX", // 0 - Reserve
5         "!CLS", // 1 - LC-Display loeschen
6         "Test", // 2 - Test
7         "@01", // 3 - Achse auswaehlen
8         "@0R", // 4 - Status abfrage
9         "@0M", // 5 - Gehe zu Position
10        0 };
11     switch (FindStringInArray(str_rx, pOptions, 3))

```

4.6.1.1. Initialisierung

Für den String **001** wird **case 3** ausgeführt. Dieser Codeblock zeigt die Meldung *Init* auf dem Display an und sendet den erwarteten Befehl an RapidForm2004.

Listing 4.15: case 3: Initialisierung

```

1 case 3: // 3 - Achse auswaehlen (Initialisierung)
2     ms_spin(10);
3     lcd_puts("Init");
4     uart_put_string("0\r\n", D_RapidForm);
5     break;

```

4. Zeitlicher Arbeitsablauf**4.6.1.2. Statusabfrage**

Wird der String `@OR` empfangen, wird der Codeblock `case 4` ausgeführt. Auf dem LC-Display wird *Statusabfrage:* ausgegeben. Danach wird der entsprechende Befehl für eine Statusabfrage an die Schrittmotorkarte gesendet. Nach einer kurzen Pause von 50ms, um die Verarbeitung auf der Schrittmotorkarte zu gewährleisten, wird mit einer if-Anweisung geprüft ob sich Daten im Schrittmotorkarten-Empfangsregister befinden. Sprich, die Schrittmotorkarte reagiert hat. Ist dies der Fall, wird der Ablauf, bekannt aus Kapitel 4.3, durchlaufen. Während diesem Ablauf wird die entsprechende Antwort der Schrittmotorkarte auf dem LC-Display ausgegeben. In einer weiteren if-Anweisung wird überprüft ob der angekommene String erfolgreich war. Wenn ja, wird dies an RapidForm2004 gemeldet. Andernfalls zeigt das Display *Fehlgeschlagen* an und sendet eine `1` an RapidForm2004.

Listing 4.16: case 4: Statusabfrage

```

1 case 4:           // 4 – Status abfrage
2     lcd_puts("Statusabfrage:    \n");
3     uart_put_string("A\n", D_Stepper); // Statusabfrage an Stepper senden
4     ms_spin(50);                   // Verarbeitungszeit gewaehren
5     if ((UCSR1A & (1 << RXC1))) // Wenn ein Zeichen empfangen wurde
6         uart_rx(D_Stepper);      // Zeichen auslesen
7     if (!strcmp(str_rx,"0#"))
8         uart_put_string("0\r\n", D_RapidForm); // Antwort Ok an RF melden
9     else {
10         lcd_puts("Fehlgeschlagen    \n"); // Fehler auf Display anzeigen
11         uart_put_string("1\r\n", D_RapidForm); // Fehler an RapidForm melden
12     }
13     break;

```

4.6.1.3. Rotation

Der Codeblock von `case 5` ist für die Rotation verantwortlich. Es werden je ein String für die Endposition und für den Winkel mit Stringterminierungszeichen vorbelegt. Diese werden an die Funktion `String_zerlegen_Isel(str_rx, Position, Winkel)` (Codelisting 4.18) übergeben. Dort wird der String in die Bestandteile *Achse*, *Rotationsbefehl*, *Position/Anzahl der Schritte* und *Geschwindigkeit* zerlegt. Von diesen ist nur die Anzahl der Schritte relevant. Da die Anzahl der Schritte für den Schrittmotor angepasst werden muss, wird der String in eine Zahl umgewandelt und mit einem entsprechenden Faktor multipliziert. Zugunsten der Rechenzeit wird nicht exakt gerechnet und die Division im Faktor mit 1024 durchgeführt. Diese wird beim Komplizieren durch eine `Bitverschiebung` ersetzt. Dies spart mehrere Sekunden Rechenzeit und die Abweichung der Schritte beträgt nur maximal 3 Schritte. Die berechnete Anzahl der

4. Zeitlicher Arbeitsablauf

Schritte wird anschließend wieder als String gespeichert. Dieser wird dann an den String für den Rotationsbefehl der Schrittmotorkarte angehängt. Der neue String wird auf dem Display ausgegeben und an die Schrittmotorsteuerung gesendet. Die Antwort der Schrittmotorsteuerung wird ausgelesen und anschließend wird in einer **while-Schleife** so lange der Status des Motors abgefragt bis dieser keine Bewegung mehr meldet. Die Position ist damit erreicht und es wird der erwartete Befehl an RapidForm2004 gesendet.

Listing 4.17: case 5: Rotation

```

1 case 5:                      // 5 – Gehe zu Position MX , +600
2     ms_spin(10);
3     char Position [33], Winkel[6];
4     memset(Position, '\0', 33); // String 0 Terminiert vorbelegen
5     memset(Winkel, '\0', 6);   // String 0 Terminiert vorbelegen
6     String_zerlegen_Isel(str_rx, Position, Winkel); // String auswerten
7     // String fuer Stepper vorbereiten
8     char Move_To[40];
9     memset(Move_To, '\0', 40);
10    Move_To[0] = 'M';
11    Move_To[1] = 'A';
12    Move_To[2] = ' ';
13    Move_To[3] = '\0';
14    strcat(Move_To, Position);
15    strcat(Move_To, "\n");
16    lcd_puts("Pos:");
17    lcd_puts(Move_To);
18    // String an Stepper senden
19    uart_put_string(Move_To, D_Stepper);
20    ms_spin(50);
21    if ((UCSR1A & (1 << RXC1)))
22        uart_rx(D_Stepper); // Antwort des Stepper auslesen
23    else {
24        break; // Bei Fehler abbrechen
25    }
26    // Status des Stepper Abfragen
27    uart_put_string("A\n", D_Stepper);
28    ms_spin(50);
29    // Antwort des Stepper Abfragen
30    if ((UCSR1A & (1 << RXC1)))
31        uart_rx(D_Stepper);
32    else {
33        lcd_puts("Keine Bewegung!\n");
34    }
35    // So lange der Stepper Bewegung meldet erneut Statusabfrage
36    while (!strcmp(str_rx,"1#")){
37        // Statusabfrage an Stepper
38        uart_put_string("A\n", D_Stepper);

```

4. Zeitlicher Arbeitsablauf

```

39         ms_spin(50);
40         // Statusabfrage auslesen und auswerten
41         if ((UCSR1A & (1 << RXC1))){
42             uart_rx(D_Stepper);
43             lcd_clrscr();
44             lcd_puts("Gehe zu Winkel: ");
45             lcd_puts(Winkel);
46             lcd_puts("\n");
47         }
48         else {
49             lcd_puts("Keine Antwort\n");
50         }
51         wdt_reset();
52     }
53     lcd_puts("Winkel: ");
54     lcd_puts(Winkel);
55     lcd_puts(" Erreicht\n");
56     // Bewegung abgeschlossen an RapidForm melden
57     uart_put_string("0\r\n", D_RapidForm);
58     break;

```

Listing 4.18: Funktion: string zerlegen Isel()

```

1 void String_zerlegen_Isel(char * str_rx, char * Position, char * Winkel) {
2     //0M5200, +600
3     //Achse M Position, +Geschwindigkeit
4     char * Achse="0";
5     Achse[0] = str_rx[1];    // Achse setzen
6     Achse[1] = '\0';
7     // Ausgeben welche Achse gewaehlt wurde
8     if(atoi(Achse)==0){
9         lcd_puts("Achse: ");
10        lcd_puts(Achse);
11        lcd_puts(" (Rotation)\n");
12    }
13    if(atoi(Achse)==1){
14        lcd_puts("Achse: ");
15        lcd_puts(Achse);
16        lcd_puts(" (Hoehe) \n");
17    }
18    // Anzahl der Schritte aus dem String auslesen
19    char c;
20    int i = 0;
21    do {
22        c = str_rx[i + 3];
23        if (c != ',') {
24            Position[i] = c;
25            i++;
26        }

```

4. Zeitlicher Arbeitsablauf

```

27 } while (i < 20 && c != '\0' && c != ',');
28 Position[i] = '\0'; // String 0 Terminieren
29 int32_t z;
30 int32_t y;
31 z = atol(Position); // String in Zahl(long) umwandeln
32 y = z / 7200; // Berechnung des Winkel
33 z = (z * 71111) / 1024; // Berechnung der Schritte
34 ltoa(y, Winkel, 10); // Winkel in String umwandeln
35 ltoa(z, Position, 10); // Schritte in String umwandeln

```

4.7. Platinenlayout und 19"-Einschub

Der Mikrocontroller und seine Peripherie befanden sich noch auf dem STK500. Es soll ein Platinenlayout entwickelt werden, welches den Mikrocontroller und seine Peripherie enthält.

Dazu wird ein Platinenlayout in der Open Source Software KiCad entwickelt. Diese bietet fast alles, was benötigt wird um ein Platinenlayout zu entwickeln. Ein Schaltplaneditor, ein Bauteileditor und ein Layouteditor. Da die Schrittmotorkarten als 19"-Einschübe realisiert sind, wird auch das Platinenlayout für den Mikrocontroller als 19"-Einschub entwickelt. Dazu gehören vor allem der Steckverbinder an der Rückseite der Platine und genügend Platz für die Verschraubung der Blende an der Vorderseite, sowie die Größe der Platine. Die Schaltungen wie sie auf dem STK500 vorhanden sind, werden im Schaltplaneditor von KiCad in den eigenen Schaltplan (siehe Abbildung 4.6) übernommen. Anschließend wird das Layout im Layouteditor entwickelt. Dabei waren mehrere enge Vorgaben einzuhalten. Da in der Werkstatt des RheinAhrCampus keine Platten mit Durchkontaktierungen hergestellt werden können, sollen Vias vermieden, IC-Sockel, Kondensatoren und Potis nur auf der Unterseite verlötet werden. Abschließend werden die benötigten Verbindungen zwischen den Bauteilen berechnet. Die Aufgabe übernimmt im allgemeinen ein Autorouter. Dies kann nicht in der Software KiCad selbst durchgeführt werden. Diese Funktionalität wird jedoch durch die Java-Web-Anwendung Freeroute[A.6] bereitgestellt. Da der Autorouter die Aufgabe nicht zufriedenstellend lösen konnte, mussten viele Verbindungen nachträglich manuell angelegt werden. Das fertige Layout (siehe Abbildung 4.7) wurde von der Werkstatt am RheinAhrCampus gefertigt und anschließend bestückt. Über den rückwärtigen Steckverbinder wird die Platine mit der Spannungsversorgung verbunden. Zusätzlich kommen hier auch die Signale der Endschalter an. An der Vorderseite der Platine wird eine Blende verbaut. Auf dieser Blende befinden sich das LC-Display, fünf Taster, 5 LEDs und 2 serielle Schnittstellen. Alle Bauteile sind mittels Flachbandkabel, steckbar, mit der Platine verbunden.

ÜBERSETZEN VON SCHRITTMOTORBEFEHLEN

Entwurf eines Hardwareübersetzters

4. Zeitlicher Arbeitsablauf

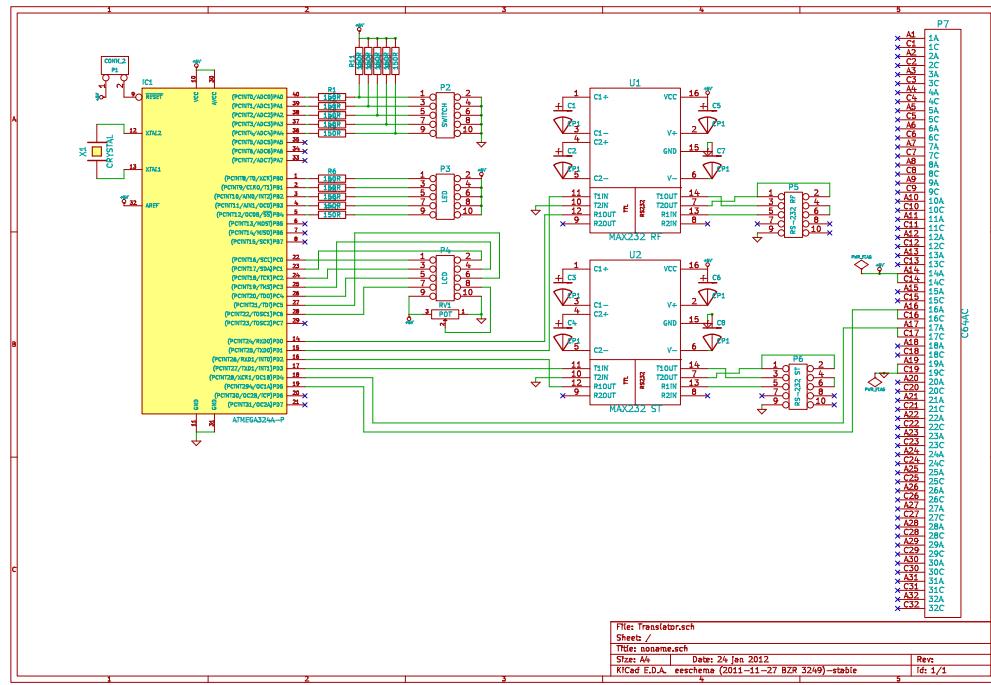


Abbildung 4.6.: Schaltplan

Dadurch sind alle im Projekt verwendeten Komponenten auf einem 19"-Einschub (siehe Abbildung 4.8) vereint.

4. Zeitlicher Arbeitsablauf

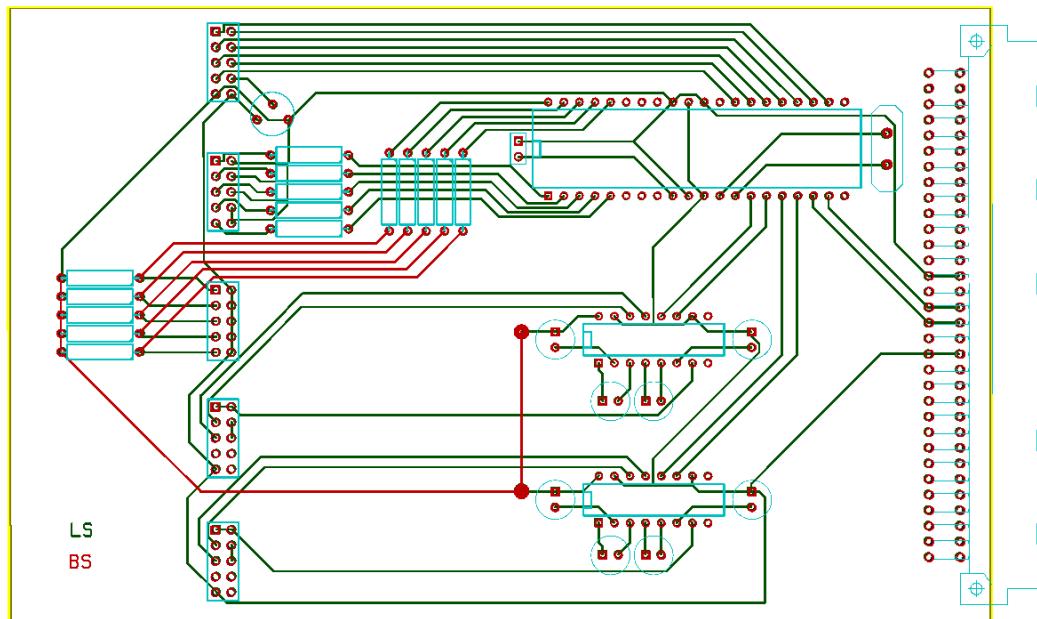


Abbildung 4.7.: Platinenlayout

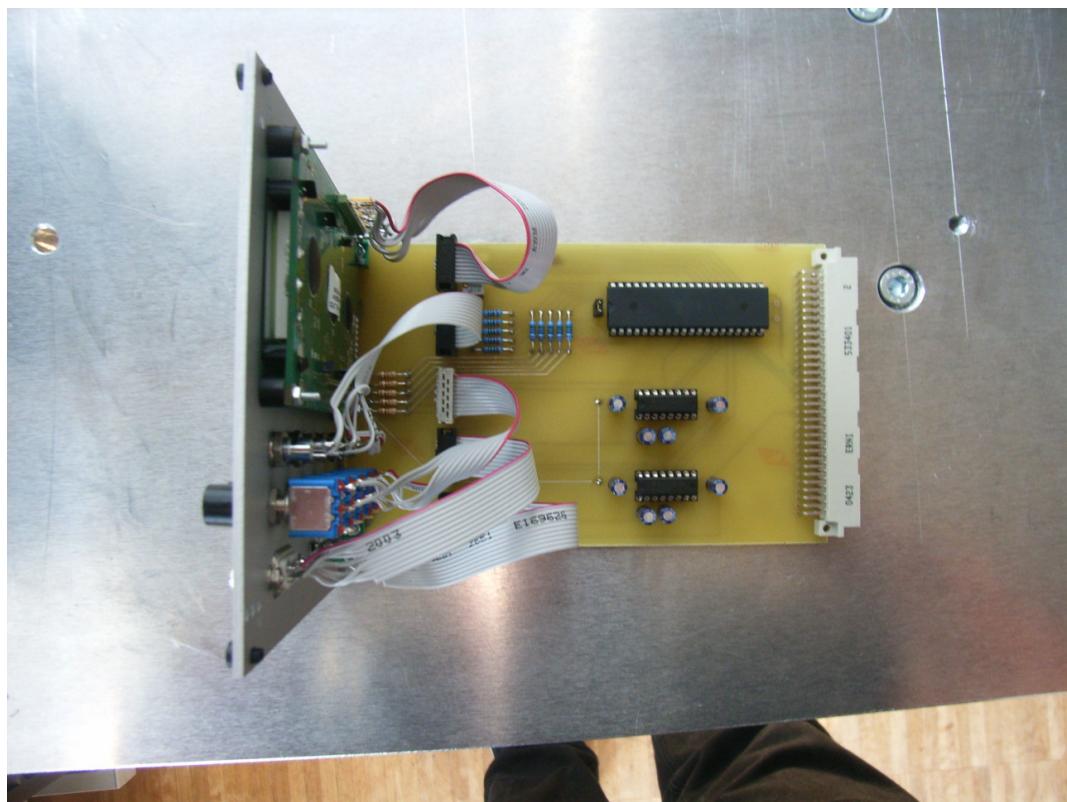


Abbildung 4.8.: 19"-Einschub

5. Probleme und Lösungen

5.1. Entwicklungsumgebungen

5.1.1. AVR Studio 5

AVR Studio 5[A.6] 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 selben Programm und einem anderem Mikrocontroller oder einer anderen Entwicklungsumgebung tauchten keine Fehler auf. In der Entwicklungsumgebung *Eclipse*[A.6] lies sich der Fehler reproduzieren wenn der Pfad der von Atmel bereitgestellten Bibliotheken eingestellt wurde. Eine von *WinAVR* bereitgestellte Bibliothek und eine selbst kompilierte Toolchain unter *Linux* zeigten diese Probleme nicht.

Daher wurde zur Open Source Entwicklungsumgebung Eclipse mit freien Bibliotheken von WinAVR gewechselt. Dadurch wurde das Problem umgangen und das Projekt plattformunabhängig. Bis auf RapidForm2004 wurde somit nur noch freie Open Source Software verwendet.

5.1.2. Eclipse

Eclipse ist eine in Java programmierte freie open source Entwicklungsumgebung für Java. Sie lässt sich durch Plugins leicht für viele Programmiersprachen erweitern. Mit dem *CDT-Plugin*, dem *AVR-Plugin*, der Bibliothek von *WinAVR* und der Programmiersoftware *AVRDude* ist Eclipse eine vollwertige Entwicklungsumgebung für Mikrocontroller von Atmel.

5.2. Interrupts

Viele Mikrocontroller bieten die Möglichkeit *eventbasierte Subroutinen* auszuführen. Wenn ein sogenannter Interrupt ausgelöst wird, wird das Hauptprogramm unterbrochen und eine entsprechende Interrupt-Service-Routine, kurz ISR, ausgeführt. Nach Beendigung der ISR wird das Hauptprogramm an der Stelle wieder aufgenommen, an der es unterbrochen wurde. ISR dürfen nur sehr wenige Befehle enthalten

5. Probleme und Lösungen

und sollten innerhalb weniger Taktzyklen abgeschlossen sein.

Interrupts können z.B. der Überlauf eines internen Timer, oder ein externes Signal an einem Pin sein. Im Projekt werden Externe-Interrupts für die Endschalter, Timer-Überlauf-Interrupts für das Entprellen der Taster und der Watchdog-Interrupt zum erkennen von Fehlern genutzt.

5.2.1. Endschalter

Die Endschalter sind über die Schrittmotorkarten und eine Brücke auf der Rückseite der Einschubsteckplätze mit der Mikrocontrollerplatine verbunden. Dort sind sie an 2 Interrupt Pins angeschlossen. (**(TODO: PINS RAUS SUCHEN!)**) Bei einem Flankenwechsel an den Pins wird ein Interrupt ausgelöst.

Mit den Zeilen 1–2 des Codelistings 5.1 werden Pin-Change-Interrupts für bestimmte Pins zugelassen. Die Zeilen 3–7 und 8–12 zeigen die Interrupt-Service-Routinen für die beiden Interrupts.

Listing 5.1: ISR: Endschalter

```
1 PCMSK3 |= ( 1 << PCINT28 ); // Interrupts definieren PD4 als Interrupt zulassen
2 PCICR |= ( 1 << PCIE3 ); // Pin Change Interrupt ControlR PCIE3 setzen fuer PCINT30
3 ISR(PCINT3_vect){ // + Endschalter Position erreicht
4     lcd_clrscr();
5     lcd_puts("Obere\nEndposition\nnErreicht!");
6     LED_PORT ^= (1 << LED3);
7 }
8 ISR(PCINT2_vect){ // - Endschalter Position erreicht
9     lcd_clrscr();
10    lcd_puts("Untere\nEndposition\nnErreicht!");
11    LED_PORT ^= (1 << LED3);
12 }
```

5.2.2. Watchdog

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

Mit den Zeilen 3–10 des Codelisting 5.2 wird der Watchdog initialisiert und festgelegt in welchen Zeitintervallen das Watchdog-Bit überprüft werden soll. Der Ablauf zum einstellen des Zeitinveralls muss genau wie im Datenblatt des Mikrocontrollers beschrieben eingehalten werden. Dies verhindert ein versehentliches Ändern der

5. Probleme und Lösungen

Einstellung. Ist das Fusebit *WDTON* gesetzt kann der Watchdog nicht abgeschaltet werden (siehe Kapitel 5.3).

Wahlweise kann kurz vor dem zurücksetzen des Mikrocontroller 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 Codelisting 5.2 Zeilen 12-16).

Listing 5.2: Watchdog

```
1 #include <avr/wdt.h>
2 // Initialisierung des Watchdog
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 // Watchdog ISR
12 ISR(WDT_vect){
13     LED_PORT &= ~(1 << LED4); // LED5 einschalten
14     lcd_clrscr();
15     lcd_puts("Something went terribly wrong!\nRebooting!");
16 }
```

5.3. 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 Bit gesetzt, kann an 4 Pins des *PortB* ein **JTAG-Debugger** angeschlossen werden. Debuggen auf Hardwareebene bietet viele Vorteile. Diese wurden im Projekt jedoch nicht genutzt, da kein JTAG-Debugger zur Verfügung stand und PortB für die LEDs genutzt wurde.
- **WDTON** - Ist dieses Bit gesetzt, läuft der Watchdog-Timer immer mit. Wird der Watchdog dann nicht regelmäßig mit der Funktion **wdt_reset()** zurückgesetzt, startet der Mikrocontroller ständig neu.
- **CKDIV8** - Teilt den Systemtakt des Mikrocontroller durch 8. Dies ist Energieparender. Der geringere Takt muss in *F_CPU* angepasst werden da sonst zeitkritische Prozesse mit der falschen Geschwindigkeit ablaufen. Das Bit wurde jedoch im Projekt nicht gesetzt.

5. Probleme und Lösungen

- **CKOUT** - An einem Pin an *PortB* wird 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 5.1.: Fuses

| | |
|---------|---|
| OCDEN | On Chip Debugging |
| JTAGEN | Hardware Debugging |
| SPIEN | Serial Program and Data Downloading |
| WDTON | Watchdot 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. Fazit und weitere Möglichkeiten

6. Fazit und weitere Möglichkeiten

6.1. Fazit

Das vorgegebene Ziel den Aufbau in Betrieb zu nehmen und mit einem Mikrocontroller so zu erweitern, dass die Erfassungssoftware RapidForm2004 kompatibel mit dem vorhandenen Drehtisch ist, wurde erreicht. Die Software kann vollständig genutzt werden und fast alle Funktionen der Schrittmotorsteuerungen werden unterstützt. Im Anhang A.1 befindet sich eine Schritt-für-Schritt Anleitung mit der auch Laien das System nutzen können.

6.2. Bekannte Probleme

Bei einem abschließenden Test funktionierte das Anzeigen einer Meldung beim erreichen der Endschalter, auf dem Display nicht. Alle Verbindungen sind vorhanden und die Programmierung des Mikrocontrollers vollständig. Das Problem ist nicht bekannt und das Auffinden würde weitere Zeit in Anspruch nehmen.

Das Display zeigt während der Rotation θ anstatt dem Winkel an, um den rotiert wird. Für die Anzahl der Schritte funktionierte diese Anzeige. Vermutlich liegt hier ein Fehler in der Berechnung des Winkels vor.

6.3. Weitere Möglichkeiten

Eine elegantere Lösung als die Befehle der Befehlssätze in einem Array zu speichern wäre es diese in verketteten Pointer Structs zu speichern. Diese Idee kam leider erst gegen Ende des Projektes und konnte daher aus Zeitmangel nicht mehr umgesetzt werden.

Im Menü lassen sich zur Zeit nur voreingestellte Winkel bzw. Schrittzahlen auswählen. Hier könnte noch eine Funktion geschrieben werden die es dem Benutzer erlaubt Winkel frei einzustellen.

Literaturverzeichnis

- [1] Atmel Corporation, San Jose, CA 95131, USA. *AVR STK500 User Guide*, 06 2003. ([document](#)), [2.7](#)
- [2] Atmel Corporation, San Jose, CA 95131, USA. *ATmega164A/PA/324A/PA/644A/PA/1284/P Complete*, 06 2011. ([document](#)), [2.6](#)
- [3] Atmel Corporation. Atmega8515- atmel corporation, 2012. [Online; Stand 11. Februar 2012]. [2.4](#)
- [4] Peter Dannegger. Entprellung - mikrocontroller.net, 2012. [Online; Stand 11. Februar 2012]. [4.1.1](#)
- [5] Peter Fleury. Peter fleury's home page, 2012. [Online; Stand 11. Februar 2012]. [4.1.3](#)
- [6] Kosatec. Kosatec, 2012. [Online; Stand 01. November 2011]. ([document](#)), [4.1](#)
- [7] Mikrocontroller.net. Rs-232 - mikrocontroller.net, 2012. [Online; Stand 11. Februar 2012]. ([document](#)), [4.1.4](#), [4.4](#)
- [8] Minolta. Funktionen - vi-910 | konica minolta, 2012. [Online; Stand 11. Februar 2012]. [2.2](#)
- [9] RS, Mörfelden-Walldorf. *Schrittmotor-Platine mit integriertem Treiber*, 03 2001. [A.2](#)
- [10] Wikipedia. American standard code for information interchange — wikipedia, die freie enzyklopädie, 2012. [Online; Stand 23. Februar 2012]. [1](#)
- [11] Wikipedia. Daisy chain — wikipedia, die freie enzyklopädie, 2012. [Online; Stand 11. Februar 2012]. [2](#)

Eidesstattliche Erklärung

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich den vorliegenden Bericht:

*Übersetzen von Schrittmotorbefehlen
Entwurf eines Hardwareübersetzers*

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

Remagen, den 18. März 2012



JOHANNES DIELMANN

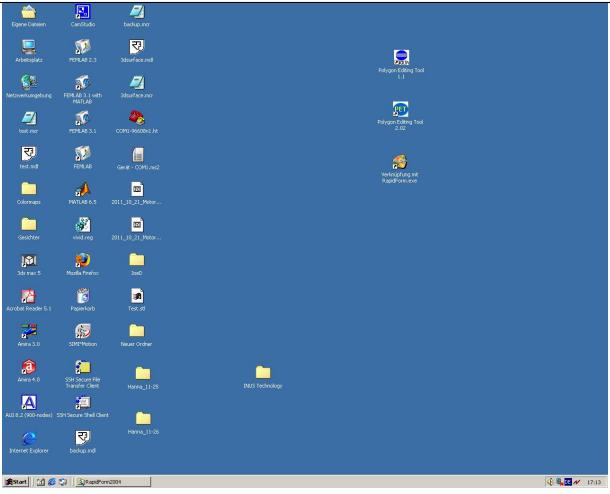
A. Anhang

A. Anhang

A. Anhang**A.1. Schritt für Schritt Anleitung**

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

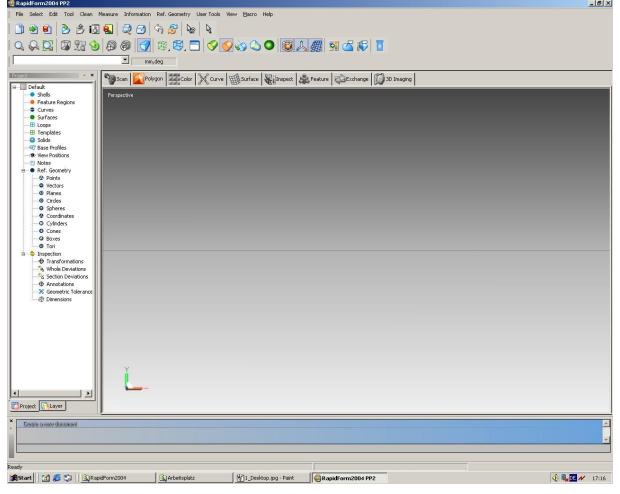
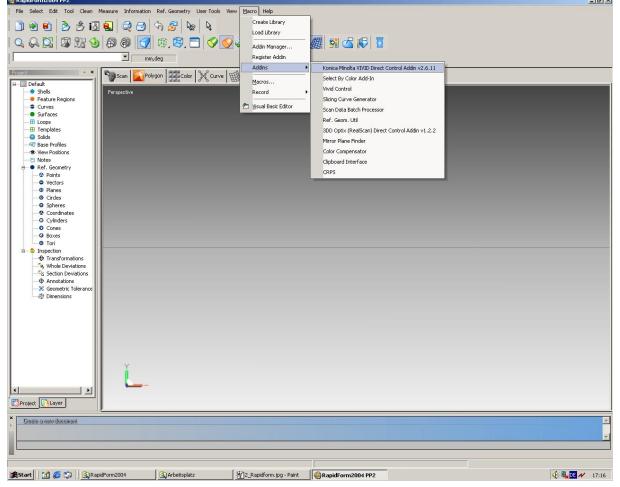
Tabelle A.1.: Schritt für Schritt Anleitung

| Schritt 1 - Einschalten der Schrittmotorsteuerung | |
|--|--|
| Auf der Rückseite des 19"-Rack befindet sich ein Schalter. Mit diesem kann die Steuerung eingeschaltet werden. |  |
| Schritt 2 - Einschalten des VI-900 | |
| Bevor der VI-900 eingeschaltet wird, muss der Erfassungs-PC eingeschaltet werden. Kurz nach dem BIOS wird nach SCSI Geräten gesucht. Zu diesem Zeitpunkt muss der VI-900 eingeschaltet werden. | |
| Schritt 3 - Einloggen | |
| Der Benutzername lautet Minolta , das Passwort kann leer bleiben. | |
| Schritt 4 - Starten von RapidForm2004 | |
| Auf dem Desktop doppelt auf das Icon RapidForm klicken. |  |
| Schritt 5 - Oberfläche von RapidForm2004 | |

ÜBERSETZEN VON SCHRITTMOTORBEFEHLEN

Entwurf eines Hardwareübersetzers

A. Anhang

| | |
|---|---|
| <p>Die Oberfläche unterteilt sich in Menü, Werkzeugeleisten, Projektbaum und Anzeigefläche. Je nach dem welche Ansicht in der Anzeigefläche gewählt ist, verändern sich auch das Menü und die Werkzeugeleisten.</p> |  |
| Schritt 6 - Starten des "ADD-IN" | |
| <p>In der Menüzeile auf Macro -> Addins -> Konica Minolta VIVID Direct Control Addin v2.6.11 klicken. (TODO: SCHRITTMOTOR VERBINDELN!)</p> |  |
| Schritt 7 - Kalibrieren vorbereiten | |

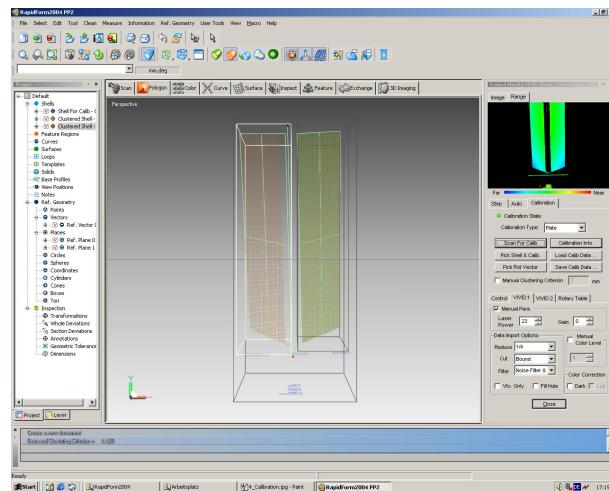
A. Anhang

| | |
|---|--|
| <p>Hinweis</p> <p>Für ein erfolgreiches Zusammenführen der einzelnen Aufnahmen ist die Kalibrierung unerlässlich!</p> <p>Auf dem Add-In Panel, unter dem Vorschau Fenster, auf Live-Preview klicken. Das Kalibrierungsblech auf dem Drehtisch positionieren. Dabei muss der Noppen an der Unterseite des Kalibrierungsblechs in das mittlere Loch des Drehtisches gesteckt werden. Die abgeklebte Seite muss zum VI-900 zeigen.</p> | |
| <p>Schritt 8 - Kalibrieren</p> <p>Den Reiter VI-VID: 1 auswählen. Bei Manual Para. ein Häkchen setzen. Im Feld Laser Power "23" eintragen. Auf Scan for Calib klicken.</p> | |
| <p>Schritt 9 - Kalibrierungsergebnis</p> | |

A. Anhang

Das Ergebnis sollte ähnlich zu dem in der rechten Abbildung sein.

Falls das Add-In einen Fehler ausgibt, muss das Kalibrierungsblech eventuell anders positioniert werden, der Wert im Feld **Laser Power** verändert werden oder der Fokus manuell eingestellt werden. War die Kalibrierung erfolgreich, können die *Kalibrationsebenen* im *Projektbaum* ausgeblendet werden.



Schritt 10 - StepScan

Bei **Manual Para.**

das Häkchen entfernen.

Zum Reiter **Step** wechseln.

Bei **Angle Tag** und **Rotate Table to next Scan position** Häkchen setzen.

Bei **Init. Align in RF**

Using Rotary In-

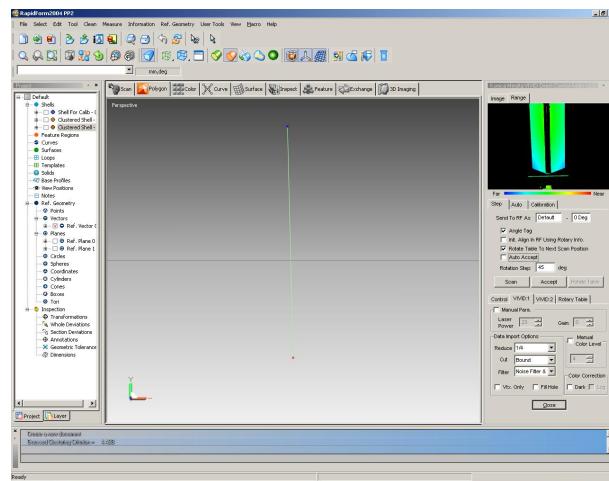
fo. und **Auto Accept**

die Häkchen entfernen.

Bei **Rotation Step**

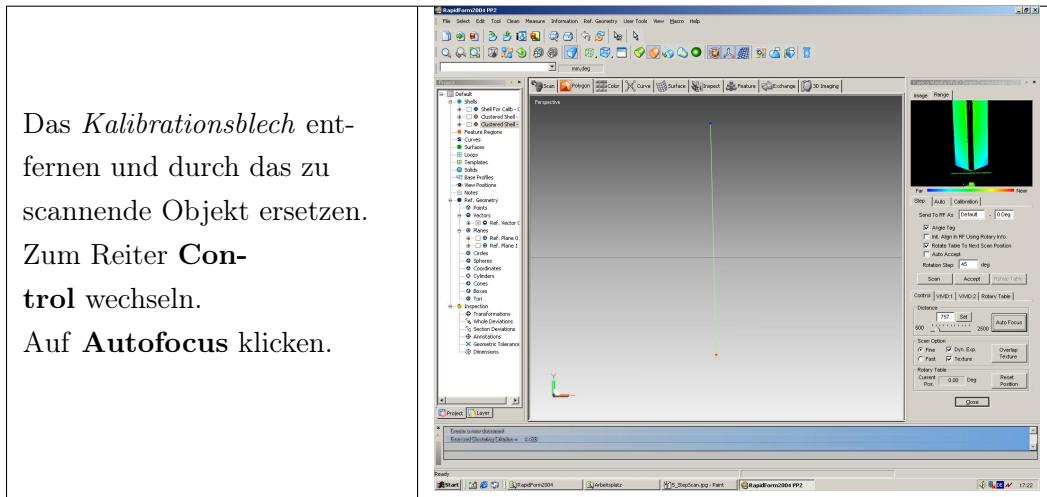
die gewünschte Dre-
hung in Grad eingeben.

”45”, ”60” und ”90” sind gute Werte.



Schritt 11 - AutoFocus

A. Anhang

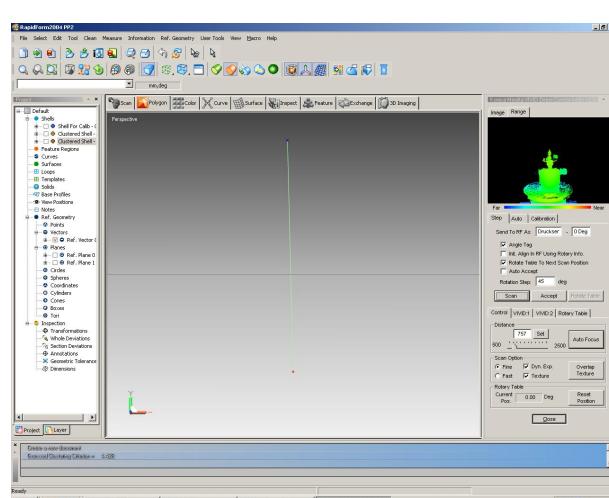


Schritt 12 - Scan

Auf **Scan** klicken.
Das Objekt sollte möglichst schon zu erkennen sein und die Farben sich im Mittleren Bereich bewegen.
Ansonsten muss mit den Parametern **Focus** und **Laser-Power** gespielt werden.

Hinweis

Die Position des VI-900 darf nach der Kalibrierung nicht mehr verändert werden!



Schritt 13 - Akzeptieren

ÜBERSETZEN VON SCHRITTMOTORBEFEHLEN

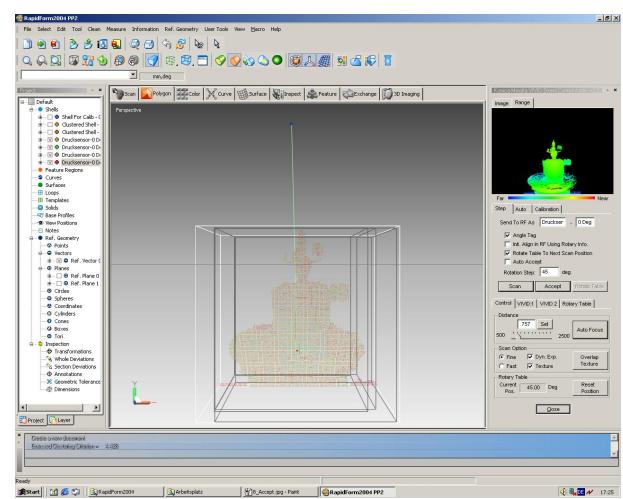
Entwurf eines Hardwareübersetzers

A. Anhang

Ist das Objekt gut zu erkennen, können mit **Accept** die Daten an RapidForm2004 gesendet werden. Der Drehtisch sollte sich nun automatisch um den eingestellten Winkel drehen.

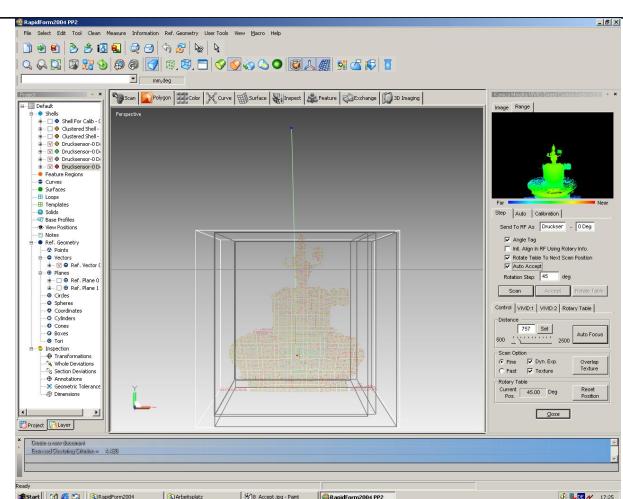
Hinweis

Bei **AutoAccept** kann nun ein Häkchen gesetzt werden.



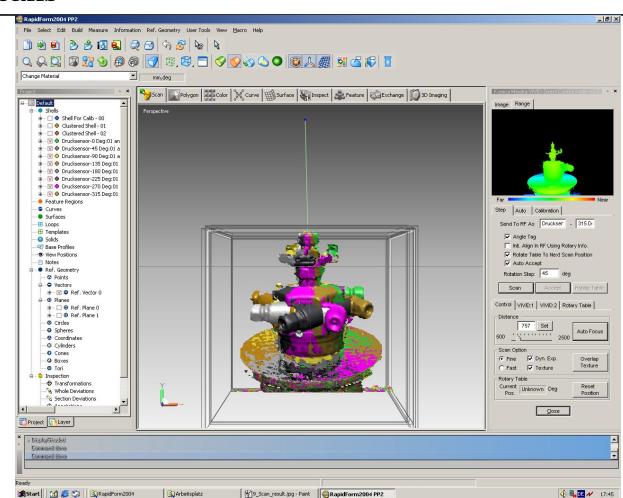
Schritt 14 - Scannen

Auf **Scan** klicken und warten bis der Scan abgeschlossen ist und der Tisch sich gedreht hat. Diesen Schritt wiederholen bis alle Aufnahmen abgeschlossen sind.



Schritt 15 - Ergebnis der Scans

Nach Abschluss aller Scans dreht der Tisch sich automatisch in die Ursprungsposition zurück. Im Arbeitsbereich sollten sich nun alle Scans befinden.



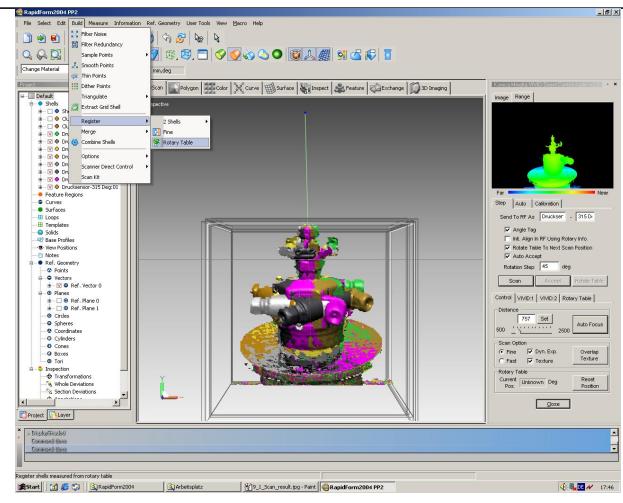
ÜBERSETZEN VON SCHRITTMOTORBEFEHLEN

Entwurf eines Hardwareübersetzers

A. Anhang

Schritt 16 - Drehen und Zusammenführen der Scans

In der Menüzeile auf
Build -> Register ->
Rotary Table klicken.

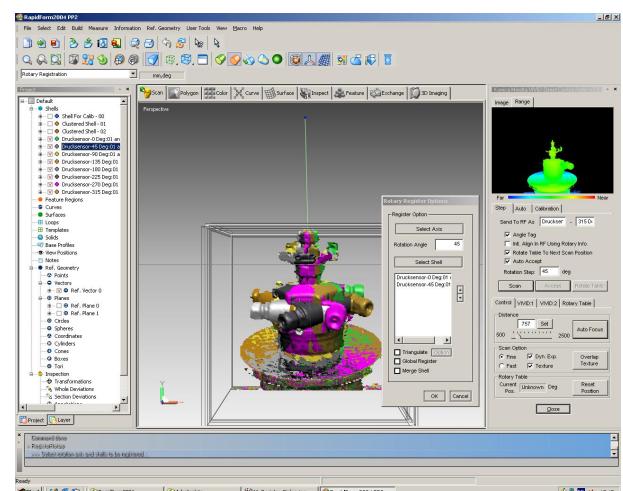


Schritt 17 - Registrieren

Im Dialog auf **Select Axis** klicken.

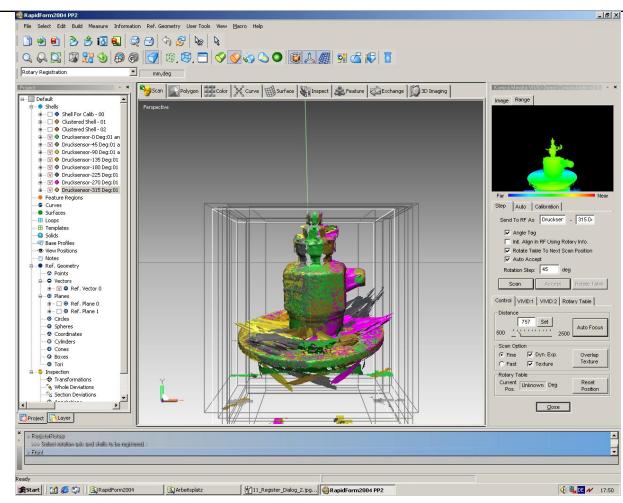
Im Darstellungsbereich auf die Achse aus dem Kalibrationsscan klicken. Bei **Rotation-Angle** den Winkel eines Scans eintragen.

Im *Projektbaum* den entsprechenden Scan auswählen. Die letzten beiden Schritte mit allen Scans wiederholen. Den Dialog mit **Ok** verlassen.



Schritt 18 - Ergebnis

Das Ergebnis ist ein komplettes 3D-Modell des Objektes.



A. Anhang

A.2. Befehlssatz der vorhandene Schrittmotorkarte

Tabelle A.2 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 EEPROM speichern |
| _R n | Mikroschritteilung einstellen |
| _RL | Endschalterwerte lesen |
| _RS | verbleibende Schritte lesen |
| _S | Nothalt |
| _T n | Eingang n auslösen |
| _W | Position anfordern |

Tabelle A.2.: ASCII Befehlssatz R+S Schrittmotorsteuerung

[9] Der „_“ wird mit der anzusteuernden Kartennummer ersetzt. Dabei wird von 1 aufwärts gezählt. Bei der ersten Karte kann die Nummer weggelassen werden.

A. Anhang

A.3. Befehlssätze aus RapidForm2004

Listing A.1: Befehlssätze aus RapidForm2004

```
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
42 model "isel(RF-1)"
43 port "9600" "n81h"
44 init "@01\r" "0"
```

A. Anhang

```

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"
68 " "
69 arot "MA1 D%d\rGO1\rWAIT(1PE<>1)\r" "\r\n>\040\r\n>\040\r\n>\040\r\n>\040"
70 home "MA1 D0 \rGO1\rWAIT(1PE<>1)\r" "\r\n>\040\r\n>\040\r\n>\040\r\n>\040"
71 stop "!$\"r" "\r\n>\040"
72 step "0.00008" "-4500000" "4500000"
73 timeout "60"
74 firsttimeout "10"
75
76 model "MMC-2(15kg)"
77 port "9600" "n81h"
78 init "F:XP3\r\nS:X1\r\n" "\r\n\r\n"
79 finish "L:X\r\nA:XP0\r\nW:\r\n" "\r\n\r\n"
80 arot "A:XP%d\r\nW:\r\n" "\r\n\r\n"
81 stop "E:\r\n" "\r\n"
82 home "A:XP0\r\nW:\r\n" "\r\n"
83 step "0.002" "-99999999" "99999999"
84 timeout "60"
85 firsttimeout "10"
86
87 model "MMC-2"
88 port "9600" "n81h"
89 init "F:XP3\r\nS:X1\r\n" "\r\n\r\n"
90 finish "L:X\r\nA:XP0\r\nW:\r\n" "\r\n\r\n"
91 arot "A:XP%d\r\nW:\r\n" "\r\n\r\n"
92 stop "E:\r\n" "\r\n"

```

A. Anhang

```
92 home "A:XP0\r\nW:\r\n" "\r\n"
93 step "0.005" "-99999999" "99999999"
94 timeout "60"
95 firsttimeout "10"
```

*A. Anhang***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.3.: 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 $f=25\text{mm}$ MITTEL: Brennweite $f=14\text{ mm}$ WEIT: Brennweite $f=8\text{mm}$ |
| 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: $\pm 0,22\text{mm}$, 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 |
| Aufnahmeeinheit | 3D-Daten: 1/3"CCD-Bildsensor (340.000 Pixel) Farbdaten: Zusammen mit 3D-Daten (Farbtrennung durch Drehfilter) |

A. Anhang

| | |
|---|--|
| 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. Anhang

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&prod_id=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/>

- **Eclipse** mit CDT und AVRPlugin

The Eclipse Foundation

Website: <http://www.eclipse.org/>

A. Anhang

• **AVRDude - AVR Downloader/UploaDER**

Brian S. Dean

Website: <http://www.nongnu.org/avrdude/>

• **Blender**

Blender Foundation Website: <http://www.blender.org/>

• **Texmaker**

Pascal Brachet

Website: <http://www.xm1math.net/texmaker/>

• **LaTeX**

LaTeX

Website: <http://www.latex-project.org/>

• **GIT**

Scott Chacon

Website: <http://git-scm.com/>

• **Inkscape**

Inkscape

Website: <http://inkscape.org/>

• **Hyperterminal** (Closed Source)

Microsoft

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

• **Putty**

Simon Tatham

Website: <http://www.chiark.greenend.org.uk/~sgtatham/putty/>

• **Freeroute**

Alfons Wirtz

Website: <http://www.freerouting.net/>

A. Anhang

A.7. main.c

Listing A.2: main.c

```
1 /*  
2 Stepper Translator – Recieve commands over RS–232, translate them and transmit them over  
3 RS–232.  
4 Copyright (C) 2011 Johannes Dielmann  
5  
6 This program is free software: you can redistribute it and/or modify  
7 it under the terms of the GNU General Public License as published by  
8 the Free Software Foundation, either version 3 of the License, or  
9 (at your option) any later version.  
10  
11 This program is distributed in the hope that it will be useful,  
12 but WITHOUT ANY WARRANTY; without even the implied warranty of  
13 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
14 GNU General Public License for more details.  
15  
16 You should have received a copy of the GNU General Public License  
17 along with this program. If not, see <http://www.gnu.org/licenses/>.  
17 */  
18  
19 #define BAUD 9600 // BAUD Rate definieren  
20 // Falls nicht bereits gesetzt, Taktfrequenz definieren  
21 #ifndef F_CPU  
22 #define F_CPU 8000000  
23 #endif  
24 // AVR Includes  
25 #include <avr/io.h> // Standardbibliothek  
26 #include <util/delay.h> // Verzögerungen  
27 #include <util/setbaud.h> // UART Funktionen  
28 #include <stdlib.h> // Standardbibliothek  
29 #include <string.h> // String Funktionen nutzen (strcmp)  
30 #include <avr/interrupt.h> // Interrupts  
31 #include <avr/wdt.h> // Watchdog  
32 #include <avr/pgmspace.h> // Stringspeicher vorbelegen  
33 // Meine Includes  
34 #include "mystuff.h" // Eigene Makrodefinitionen für erhöhte Lesbarkeit  
35 #include "Debounce.h" // Taster entprellen  
36 // #include "lcd.h" // LC-Display (wird automatisch gebaut)  
37 // Globale Variablen  
38 #define D_RapidForm 0  
39 #define D_Stepper 1  
40 // Motor Protokolle  
41 #define M_UNK -2  
42 #define M_NOTI -1  
43 #define M_ISEL 0
```

A. Anhang

```
44 #define M_CSG    1
45 #define M_ZETA   2
46 #define M_TERMINAL 3
47 // Protokoll Befehle
48 #define P_INIT    0
49 #define P_FINISH  1
50 #define P_AROT    2
51 #define P_STOP    3
52 #define P_HOME   4
53 #define P_STEP    5
54 #define P_TIMEOUT 6
55 // Erweiterte Befehle
56 #define E_CLS     10
57 #define E_TEST    11
58 // Antworten
59 #define B_Zeta    "\r\n>\040\r\n>\040\r\n>\040\r\n>\040"
60 #define B_OK      "0\r\n"
61 // Variablen Definitionen
62 #define MENU_ENTRY_NAMELEN 19
63 #define RETURN_LEN 40
64 int Initialized = M_NOTI;
65 int move = 0;
66 int init_T = 0;
67 char str_rx[100];
68
69 //// Tinymenu
70 #define MCU_CLK F_CPU // MCU_CLK = F_CPU fuer TinyMenu
71 #include "tinymenu/spin_delay.h"
72 #define CONFIG_TINYMENU_USE_CLEAR
73 #include "tinymenu/tinymenu.h"
74 #include "tinymenu/tinymenu_hw.h"
75
76 //// Funktionsdefinitionen
77 // UART Stuff
78 void uart_init();
79 void uart_put_charater(unsigned char c, int dir);
80 void uart_put_string(char *s, int dir);
81 int uart_get_character(int dir);
82 void uart_get_string(char *string_in, int dir);
83 void uart_rx(int dir);
84 // String Stuff
85 int FindStringInArray (const char* pInput, const char* pOptions[], int cmp_length);
86 void String_zerlegen_Isel(char *str_rx, char *Position, char *Winkel);
87 void String_zerlegen_csg (char *str_rx);
88 // Hilfs Funktionen
89 void csg_Status_melden();
90 // Auswerte Logik
91 int switch_Motor (char *str_rx);
```

A. Anhang

```

92 void switch_Stepper (char * str_rx);
93 void switch_Isel (char * str_rx);
94 void switch_csg (char * str_rx);
95 // LCD und LED Stuff
96 void lcd_my_type (char *s);
97 void lcd_boot (void);
98 void led_boot (void);
99 void debounce_init (void);
100 void led_lauflicht (void);
101 // Menu Stuff
102 void mod_manual (void *arg, void *name);
103 void my_select (void *arg, char *name);
104 void menu_puts (void *arg, char *name);
105 #include "mymenu.h"
106 // Init Stuff
107 void init_WDT (void);
108 void init (void);

109
110 /////////////////
111 //
112 // Hauptschleife
113 //
114 /////////////////
115 int main(void) {
116     init();      // Komponenten Initialisieren
117     while (1) { // In Endlosschleife wechseln
118         wdt_reset(); // Watchdog zuruecksetzen
119         if (get_key_press(1 << KEY1)) // 1 - Back
120             menu_exit(&menu_context);
121         if (get_key_press(1 << KEY2)) // 2 - Hoch
122             menu_prev_entry(&menu_context);
123         if (get_key_press(1 << KEY3)) // 3 - Runter
124             menu_next_entry(&menu_context);
125         if (get_key_press(1 << KEY4)) // 4 - Select
126             menu_select(&menu_context);
127         if ((UCSR0A & (1 << RXC0))){ // RapidForm Polling
128             LED_PORT &= ( 1 << LED2 ); // LED einschalten
129             uart_rx(D_RapidForm); // Register auslesen
130         }
131         if ((UCSR1A & (1 << RXC1))){ // Stepper Polling
132             LED_PORT &= ( 1 << LED3 ); // LED einschalten
133             uart_rx(D_Stepper); // Register auslesen
134         }
135     }
136 }
137 /////////////////
138 //
139 // Hauptschleife Ende

```

A. Anhang

```
140 //////////////////////////////////////////////////////////////////
141 //////////////////////////////////////////////////////////////////
142
143 // Interrupt Stuff
144 ISR(WDT_vect){           // Watchdog ISR
145     LED_PORT &= ~(1 << LED4); // LED5 einschalten
146     lcd_clrscr();
147     lcd_puts("Something went \n terribly wrong!\nRebooting!");
148 }
149 ISR(PCINT3_vect){        // + Endschalter Position erreicht
150     lcd_clrscr();
151     lcd_puts("Obere\nEndposition\nErreicht!");
152     LED_PORT ^= (1 << LED3);
153 }
154 ISR(PCINT2_vect){        // - Endschalter Position erreicht
155     lcd_clrscr();
156     lcd_puts("Untere\nEndposition\nErreicht!");
157     LED_PORT ^= (1 << LED3);
158 }
159 // UART Stuff
160 void    uart_init          () { // UART Initialisieren
161     // UART 0 – IN (Rapidform Software/Terminal)
162     UBRR0H = UBRRH_VALUE;
163     UBRR0L = UBRRL_VALUE;
164     UCSR0C = (3 << UCSZ00);
165     UCSR0B |= (1 << TXEN0); //Transmitter Enabled
166     UCSR0B |= (1 << RXEN0); // UART RX einschalten
167     // UART 1 – OUT (Stepper Karte/Drehtisch)
168     UBRR1H = UBRRH_VALUE;
169     UBRR1L = UBRRL_VALUE;
170     UCSR1C = (3 << UCSZ00);
171     UCSR1B |= (1 << TXEN1); //Transmitter Enabled
172     UCSR1B |= (1 << RXEN1); // UART RX einschalten
173 }
174 void    uart_put_charater (unsigned char c, int dir) { // UART Zeichen senden
175     if (dir == D_RapidForm) { // To Rapidform
176         while (!(UCSR0A & (1 << UDRE0))){} // warten bis Senden moeglich
177         UDR0 = c; // sende Zeichen
178     }
179     else { // To Stepper
180         while (!(UCSR1A & (1 << UDRE1))){} // warten bis Senden moeglich
181         UDR1 = c; // sende Zeichen
182     }
183 }
184 void    uart_put_string   (char *s, int dir) {           // UART String senden
185     while (*s){ // so lange *s != '\0' also ungleich dem "String-Endzeichen(Terminator)"
186         uart_put_charater(*s, dir); // Zeichenweise senden
187         s++;
188 }
```

A. Anhang

```

188    }
189}
190 int     uart_get_character (int dir) {           // UART Zeichen empfangen
191   if (dir == D_RapidForm) { // Aus RapidForm Register auslesen
192     while (!(UCSR0A & (1 << RXC0))) ; // warten bis Zeichen verfuegbar
193     return UDR0; // Zeichen aus UDR an Aufrufer zurueckgeben
194   }
195   if (dir == D_Stepper) { // Aus Schrittmotor Register auslesen
196     while (!(UCSR1A & (1 << RXC1))) ; // warten bis Zeichen verfuegbar
197     return UDR1; // Zeichen aus UDR an Aufrufer zurueckgeben
198   }
199   return -1; // Wenn nichts ausgelesen wurde -1 zurueckgeben
200 }
201 void    uart_get_string   (char * string_in, int dir) { // UART String empfangen
202   char c; // Einzelnes Zeichen
203   int i = 0; // Zaehlvariable
204   do {
205     c = uart_get_character(dir); // Einzelnes Zeichen holen
206     if (c != '\r') {           // Wenn kein \r
207       *string_in = c;         // Zeichen in Empfangsstring schreiben
208       string_in += 1;         // Adresse des Empfangsstring um 1 inkrementieren
209       i++;                  // Zaehlvariable um 1 erhoehen
210     }
211   } while (i < 100 && c != '\r' && c != '\n'); // So lange bis \r \n oder ueber 100 Zeichen
212   *string_in = '\0';           // 0 Terminieren
213   if (dir == D_Stepper)
214     LED_PORT |= (1 << LED3); // "Daten Vorhanden" LED ausschalten
215   else
216     LED_PORT |= (1 << LED2); // "Daten Vorhanden" LED ausschalten
217 }
218
219 // String Stuff
220 // Auffinden eines Strings aus einem vorgegebenen Array
221 int     FindStringInArray (const char* pInput, const char* pOptions[], int cmp_length) {
222   int n = -1;
223   while (pOptions[++n]) { //Array durchlaufen bis 0 terminiert
224     //Wenn pInput == pOptions dann gib Array Position zurueck
225     if (!strcmp(pInput, pOptions[n], cmp_length)) return n;
226   }
227   return 99; // Wenn keine uebereinstimmung, gib 99 zurueck
228 }
229 // Strings zerlegen
230 void    String_zerlegen_Isel(char * str_rx, char * Position, char * Winkel) {
231   //0M5200, +600
232   //Achse M Position, +Geschwindigkeit
233   char * Achse="0";
234   Achse[0] = str_rx[1]; // Achse setzen
235   Achse[1] = '\0';

```

A. Anhang

```
236 // Ausgeben welche Achse gewaehlt wurde
237 if(atoi(Achse)==0){
238     lcd_puts("Achse: ");
239     lcd_puts(Achse);
240     lcd_puts(" (Rotation)\n");
241 }
242 if(atoi(Achse)==1){
243     lcd_puts("Achse: ");
244     lcd_puts(Achse);
245     lcd_puts(" (Hoehe) \n");
246 }
247 // Anzahl der Schritte aus dem String auslesen
248 char c;
249 int i = 0;
250 do {
251     c = str_rx[i + 3];
252     if (c != ',') {
253         Position[i] = c;
254         i++;
255     }
256 } while (i < 20 && c != '\0' && c != ',');
257 Position[i] = '\0'; // String 0 Terminieren
258 int32_t z;
259 int32_t y;
260 z = atol(Position); // String in Zahl(long) umwandeln
261 y = z / 7200; // Berechnung des Winkel
262 z = (z * 71111) / 1024; // Berechnung der Schritte
263 ltoa(y, Winkel, 10); // Winekl in String umwandeln
264 ltoa(z, Position, 10); // Schritte in String umwandeln
265 }
266 void String_zerlegen_csg (char * str_rx) { // Unvollstaendig?
267 //012 3456 78901 2345 6789 01234 5678
268 //D:2 S500 F5000 R200 S500 F5000 R200.
269 //D:2S500F5000R200S500F5000R200
270
271 // Format:
272 // D:[Speed range]S[Minimum speed]F[Maximum Speed]R[Acceleration/Deceleration time]
273 // |-----Axis1 parameters
274 // |-----Axis2 parameters
275 // |-----Axis3 parameters
276
277
278 int i = 4; // Index Input String | Bei 4. Zeichen Beginnen. Die ersten 3 Zeichen sind Fix.
279 int j = 0; // Index Variable
280 char c; // Zu kopierendes Zeichen
```

A. Anhang

```
281 // Variablen Deklaration und Initialisierung mit Defaultwerten
282 char Speed_Range[2] = "2";
283 char ONE_Min_Speed[6] = "200";
284 char ONE_Max_Speed[6] = "2000";
285 char ONE_Acc_Speed[5] = "200";
286
287 /////////////////
288 //
289 // Speed Range (1 || 2)
290 //
291 /////////////////
292 Speed_Range[0] = str_rx[2];
293 Speed_Range[1] = '\0';
294
295 /////////////////
296 //
297 // Min Speed (50 – 20000)
298 //
299 ///////////////
300 do {
301     c = str_rx[i];
302     if (c != 'F') {
303         ONE_Min_Speed[j] = c;
304         j++;
305         i++;
306     }
307 } while (j < 6 && c != '\0' && c != 'F');
308 ONE_Min_Speed[j] = '\0';
309
310 lcd_puts("1_MIN_SPEED:");
311 lcd_puts(ONE_Min_Speed);
312 lcd_puts("\n");
313 // TODO: Range ueberpruefen! 50–20000
314 //uart_puts();
315
316 /////////////////
317 //
318 // Max Speed (50 – 20000)
319 //
320 ///////////////
321 i++; // Stuerzeichen ueberspringen
322 j = 0; // Variablenzaehler zuruecksetzen
323 do {
324     c = str_rx[i];
325     if (c != 'R') {
326         ONE_Max_Speed[j] = c;
327         i++;
328         j++;
329     }
330 }
```

A. Anhang

```
329     }
330 } while (j < 6 && c != '\0' && c != 'R');
331 ONE_Max_Speed[j] = '\0';
332
333 lcd_puts("1_MAX_SPEED:");
334 lcd_puts(ONE_Max_Speed);
335 lcd_puts("\n");
336
337 /////////////////
338 //
339 // Acceleration (0 – 1000)
340 //
341 /////////////////
342 i++; // Stuerzeichen ueberspringen
343 j = 0; // Variablenzaehler zuruecksetzen
344 do {
345     c = str_rx[i];
346     if (c != 'S') {
347         ONE_Acc_Speed[j] = c;
348         i++;
349         j++;
350     }
351 } while (j < 4 && c != '\0' && c != 'S');
352 ONE_Acc_Speed[j] = '\0';
353
354 lcd_puts("1_ACC_SPEED:");
355 lcd_puts(ONE_Acc_Speed);
356 lcd_puts("\n");
357
358 //uart_put_string("0\n", D_Stepper);
359 uart_put_string(B_OK, D_RapidForm);
360 }
361 // Hilfs Funktionen
362 void csg_Status_melden (void) {           // Unvollstaendig?
363     uart_put_string("      0,          0,K,K,R\r\n", D_RapidForm); // Status an
364     RapidForm zurueckmelden
365 }
366 void Position_Zeta (char * Position) { // Schritte Auslesen – Zeta
367     char c;
368     int i = 0;
369     do{
370         c = str_rx[i + 1];
371         if(c != ','){
372             Position[i] = c;
373             i++;
374         }
375     while(i < 20 && c != '\0' && c != ',');
```

A. Anhang

```
376     Position[i] = '\0';
377     int32_t z;
378     z = atol(Position);
379     z = z/9;
380     ltoa(z,Position,10);
381 }
382 // Uebersetzungs Logik
383 void switch_Stepper (char * str_rx) { // Uebersetzung Schrittmotorkarte
384     const char* pOptions[] = { // Array mit bekannten Befehlen
385         "#", // 0 - Stepper Karte hat Befehl erkannt
386         "E", // 1 - Stepper Karte meldet Error
387         "!CLS", // 2 - Clear Screen (Debugging)
388         "Test", // 3 - Test (Debugging)
389         0 };
390     switch (FindStringInArray(str_rx, pOptions, 1)) { // String gegen bekannte Antworten
391         pruefen
392         case 0: // 0 - Stepper Karte hat Befehl erkannt
393             lcd_puts("Erfolgreich\n");
394             break;
395         case 1: // 1 - Stepper Karte meldet Error
396             lcd_puts("Error\n");
397             uart_put_string("1\r\n", D_RapidForm);
398             break;
399         case 2: // 2 - Clear Screen (Debugging)
400             lcd_clrscr();
401             break;
402         case 3: // 3 - Test (Debugging)
403             lcd_puts("Test bestanden\n");
404             break;
405         default:
406             ms_spin(10);
407     }
408 }
409 void switch_Isel (char * str_rx) { // Uebersetzung Isel
410     const char* pOptions[] = {
411         "XXXXXXX", // 0 - Reserve
412         "!CLS", // 1 - LC-Display loeschen
413         "Test", // 2 - Test
414         "@01", // 3 - Achse auswaehlen
415         "@0R", // 4 - Status abfrage
416         "@0M", // 5 - Gehe zu Position
417         0 };
418     switch (FindStringInArray(str_rx, pOptions, 3)) {
419         case 0: // 0 - Reserve
420             lcd_puts("Reserve\r\n");
421             break;
422         case 1: // 1 - LC-Display loeschen
423             lcd_clrscr();
```

A. Anhang

```

423     break;
424 case 2:           // 2 - Test
425     lcd_puts("Test bestanden\n");
426     uart_put_string("Test bestanden\r\n", D_RapidForm);
427     break;
428 case 3:           // 3 - Achse auswaehlen
429     ms_spin(10);
430     lcd_puts("Init");
431     uart_put_string("0\r\n", D_RapidForm);
432     break;
433 case 4:           // 4 - Status abfrage
434     lcd_puts("Statusabfrage:    \n");
435     uart_put_string("A\n", D_Stepper); // Statusabfrage an Stepper senden
436     ms_spin(50);                  // Verarbeitungszeit gewaehren
437     if ((UCSR1A & (1 << RXC1))) // Wenn ein Zeichen empfangen wurde
438         uart_rx(D_Stepper);      // Zeichen auslesen
439     if (!strcmp(str_rx,"0#"))    // Empfangenes Zeichen ueberpruefen
440         uart_put_string("0\r\n", D_RapidForm); // Antwort Ok an RapidForm melden
441     else {
442         lcd_puts("Fehlgeschlagen    \n"); // Fehler auf Display anzeigen
443         uart_put_string("1\r\n", D_RapidForm); // Fehler an RapidForm melden
444     }
445     break;
446 case 5:           // 5 - Gehe zu Position MX , +600
447     ms_spin(10);
448     char Position [33], Winkel[6];
449     memset(Position, '\0', 33); // String 0 Terminierte vorbelegen
450     memset(Winkel, '\0', 6);   // String 0 Terminierte vorbelegen
451     String_zerlegen_Isel(str_rx, Position, Winkel); // String auswerten
452     // String fuer Stepper vorbereiten
453     char Move_To[40];
454     memset(Move_To, '\0', 40);
455     Move_To[0] = 'M';
456     Move_To[1] = 'A';
457     Move_To[2] = ' ';
458     Move_To[3] = '\0';
459     strcat(Move_To, Position);
460     strcat(Move_To, "\n");
461     lcd_puts("Pos:");
462     lcd_puts(Move_To);
463     // String an Stepper senden
464     uart_put_string(Move_To, D_Stepper);
465     ms_spin(50);
466     if ((UCSR1A & (1 << RXC1)))
467         uart_rx(D_Stepper); // Antwort des Stepper auslesen
468     else {
469         break; // Bei Fehler abbrechen
470     }

```

A. Anhang

```
471 // Status des Stepper Abfragen
472 uart_put_string("A\n", D_Stepper);
473 ms_spin(50);
474 // Antwort des Stepper Abfragen
475 if ((UCSR1A & (1 << RXC1)))
476     uart_rx(D_Stepper);
477 else {
478     lcd_puts("Keine Bewegung!\n");
479 }
480 // So lange der Stepper Bewegung meldet erneut Statusabfrage
481 while (!strcmp(str_rx,"1#")){
482     // Statusabfrage an Stepper
483     uart_put_string("A\n", D_Stepper);
484     ms_spin(50);
485     // Statusabfrage auslesen und auswerten
486     if ((UCSR1A & (1 << RXC1))){
487         uart_rx(D_Stepper);
488         lcd_clrscr();
489         lcd_puts("Gehe zu Winkel: ");
490         lcd_puts(Winkel);
491         lcd_puts("\n");
492     }
493     else {
494         lcd_puts("Keine Antwort\n");
495     }
496     wdt_reset();
497 }
498 lcd_puts("Winkel: ");
499 lcd_puts(Winkel);
500 lcd_puts(" Erreicht\n");
501 // Bewegung abgeschlossen an RapidForm melden
502 uart_put_string("0\r\n", D_RapidForm);
503 break;
504 default: // Unbekannte Befehle auf dem Display anzeigen
505     lcd_puts(str_rx);
506 }
507 }
508 void switch_csg (char * str_rx) { // Uebersetzung CSG Unvollstaendig?
509 const char* pOptions[] = {
510     "Test2", // 0 - Stepper Karte Befehl erkannt
511     "!CLS", // 1 - LC-Display loeschen
512     "Test", // 2 - Test
513     "Q:", // 3 - Status abfrage
514     "D:2", // 4 - D:2S500F5000R200S500F5000R200.
515     "H:", // 5 - H:
516     "G", // 6 - Motor starten
517     "M:", // 7 - Move by Pulses
518     "!", // 8 - Busy Ready ?
```

A. Anhang

```
519         "H1",
520         0 };
521     switch (FindStringInArray(str_rx, pOptions, 2)) {
522     case 0: // Motorkarte Erfolgreich angesprochen
523         lcd_puts("!");
524         break;
525     case 1: // Display loeschen
526         lcd_clrscr();
527         break;
528     case 2: // Interner Test
529         lcd_puts("!T");
530         //uart_puts("Test bestanden\n\r");
531         break;
532     case 3: // Status abfrage von Software
533         lcd_puts("Statusabfrage      \n");
534         csg_Status_melden();
535         break;
536     case 4:
537         String_zerlegen_csg(str_rx);
538
539         break;
540     case 5:
541         lcd_puts("H:                  \n");
542         uart_put_string(B_OK, D_RapidForm);
543         break;
544     case 6:
545         lcd_puts("Motor starten\n");
546         //uart_put_string(B_OK, D_RapidForm);
547         break;
548     case 7:
549         move++;
550         char it [10];
551         itoa(move, it, 10);
552         lcd_puts(it);
553         lcd_puts("_Move!\n");
554         uart_put_string("M 160000\r\n",D_Stepper);
555
556         break;
557     case 8:
558         lcd_puts("R/B?");
559         uart_put_string("R\r\n", D_RapidForm);
560         break;
561     case 9:
562         lcd_puts("H1 empfangen      \n");
563         break;
564     default:
565         lcd_puts("U_B: ");
566         lcd_puts(str_rx);
```

A. Anhang

```
567     lcd_puts("!END      \n");
568 }
569 }
570 void switch_Zeta      (char * str_rx) { // Uebersetzung Zeta
571     const char* pOptions[] = {
572         "!CLS", // 0 - LC-Display loeschen
573         "Test", // 1 - Test
574         "GO", // 2 - Motor Starten
575         "WAIT", // 3 - Wait till motor stops
576         "!XXXX", // 4 - Reserve
577         "COMEX", // 5 - *COMEXC0
578         "MA1", // 6 - Absolute Positioning
579         "D1125", // 7 - Position
580         "A8", // 8 - Accelartion 8
581         "V8", // 9 - Velocity 8
582         "ECHO0", // 10 - Echo abschalten
583         "PSET0", // 11 - Ursprung setzen
584         0 };
585     char Position[33];
586     char Move_To[40];
587     memset(Move_To, '\0', 40);
588     Move_To[0] = 'M';
589     Move_To[1] = 'A';
590     Move_To[2] = ' ';
591     Move_To[3] = '\0';
592     switch (FindStringInArray(str_rx, pOptions, 1)) {
593     case 0: // Display loeschen
594         lcd_clrscr();
595         break;
596     case 1: // Interner Test
597         lcd_puts("Test bestanden      \n");
598         break;
599     case 2: // Go
600         ms_spin(100);
601         strcat(Move_To, Position);
602         strcat(Move_To, "\n");
603
604         uart_put_string(Move_To, D_Stepper);
605         ms_spin(50);
606         if ((UCSR1A & (1 << RXC1)))
607             uart_rx(D_Stepper);
608         else {
609             lcd_puts("Befehl n. bestaetig\n");
610             break;
611         }
612
613         uart_put_string("A\n", D_Stepper);
614         ms_spin(50);
```

A. Anhang

```
615     if ((UCSR1A & (1 << RXC1)))
616         uart_rx(D_Stepper);
617     else {
618         lcd_puts("Keine Bewegung!\n");
619     }
620
621     while (!strcmp(str_rx,"1#")){
622         uart_put_string("W\n", D_Stepper);
623         ms_spin(100);
624         if ((UCSR1A & (1 << RXC1))){
625             uart_rx(D_Stepper);
626             lcd_clrscr();
627             lcd_puts("Position(Akt/Ges): \n");
628             lcd_puts(str_rx);
629             lcd_puts(" / ");
630         }
631         else {
632             lcd_puts("Keine Antwort\n");
633         }
634         wdt_reset();
635
636         uart_put_string("A\n", D_Stepper);
637         ms_spin(50);
638         if ((UCSR1A & (1 << RXC1))){
639             uart_rx(D_Stepper);
640             //lcd_clrscr();
641             //lcd_puts("running to\n");
642             //lcd_puts("Position: ");
643             lcd_puts(Position);
644             lcd_puts("\n");
645         }
646         else {
647             lcd_puts("Keine Antwort\n");
648         }
649         wdt_reset();
650     }
651     lcd_puts("Position: \n");
652     lcd_puts(Position);
653     lcd_puts(" Erreicht\n");
654     uart_put_string(B_Zeta, D_RapidForm);
655     break;
656 case 3: // WAIT
657     break;
658 case 4: // Reserve
659     break;
660 case 5: // COMEXC0
661     break;
662 case 6:
```

A. Anhang

```
663     //lcd_puts("MA1 empfangen \n");
664     break;
665 case 7: // Position Setzen
666     memset(Position, '\0', 33);           // Array mit Nullen befuellen
667     Position_Zeta(Position);
668     break;
669 case 8:
670     break;
671 case 9:    //V8
672     lcd_puts("Speed set      \n");
673     //uart_put_string(B_Zeta_Return, D_RapidForm);
674     break;
675 case 10:
676     lcd_puts("Echo off      \n");
677     //uart_put_string(str_rx, D_RapidForm);
678     //uart_put_string("ECHO0\r", D_RapidForm);
679     break;
680 case 11:
681     break;
682 default:
683     lcd_puts("Z:");
684     lcd_puts(str_rx);
685     lcd_puts(" \n");
686     // Initialized = switch_Inputs(str_rx);
687 }
688 }
689 void switch_Terminal (char * str_rx) { // Uebersetzung Terminal Unvollstaendig
690 const char* pOptions[] = {
691     "!CLS", // 0 - LC-Display loeschen
692     "Test", // 1 - Test
693     "!Manual",// 2 - Ignorieren
694     "!YYYY",// 3 - Wait till motor stops
695     0 };
696
697 if (init_T == 0){
698     init_T = 1;
699     uart_put_string("Willkommen im Terminal Modus\r\n",D_RapidForm);
700     uart_put_string("moegliche Befehle sind: \r\n",D_RapidForm);
701     uart_put_string(" A - Motorstatus\r\n M - Move Steps\r\n", D_RapidForm);
702 }
703 switch (FindStringInArray(str_rx, pOptions, 2)) {
704 case 0: // Display loeschen
705     lcd_clrscr();
706     break;
707 case 1: // Interner Test
708     lcd_puts("Test bestanden \n");
709     uart_put_string("Test bestanden", D_RapidForm);
710     break;
```

A. Anhang

```
711     case 2: // Reserve 1
712
713     case 3: // Reserve 2
714
715         break;
716     default:
717         //lcd_puts("Z:");
718         lcd_puts(str_rx);
719         lcd_puts("\n");
720         uart_put_string(str_rx,D_Stepper);
721         uart_put_string("\n",D_Stepper);
722     }
723 }
724 int      switch_Motor      (char * str_rx) { // Automatische Befehlssatzwahl
725 const char* pOptions[] = { // Array mit Initialisierungsbefehlen
726     "@01",      // 0 - Isel
727     "Q:",       // 1 - CSG
728     "ECHO0",    // 2 - Zeta
729     "!Terminal", // 3 - Terminal ansteuerung!
730     0 };
731 // Ankommenden String gegen Array pruefen
732 switch (FindStringInArray(str_rx, pOptions, 3)) {
733     case 0:        // 0 - ISEL
734         return M_ISEL;
735         break;
736     case 1:        // 1 - CSG
737         return M_CSG;
738         break;
739     case 2:        // 2 - Zeta
740         return M_ZETA;
741         break;
742     case 3:        // 3 - Terminal ansteuerung
743         return M_TERMINAL;
744         break;
745     default:
746         return M_UNK;
747     }
748 }
749 void      uart_rx          (int dir) { // UART Empfangsregister auslesen
750     uart_get_string(str_rx,dir); // String aus Empfangsregister auslesen
751     if (dir == D_Stepper) // Empfangsregister Stepper
752         switch_Stepper(str_rx); // Uebersungsfunktion fuer Stepper aufrufen
753     else{ // Empfangsregister RapidForm
754         // Uebersetzungsfunktion auswaehlen
755         if( Initialized == M_UNK){ // Unbekannter Initialisierungsbefehl
756             lcd_puts("Unbekannter Motor!\n");
757             Initialized = M_NOTI; // Variable Initialized zuruecksetzen
758         }
```

A. Anhang

```
759     if( Initialized == M_NOTI){ // Befehlssatz bestimmen
760         Initialized = switch_Motor(str_rx); //Automatische Befehlssatzwahl
761     }
762     if( Initialized == M_ISEL) // Uebersetzung ISEL
763         switch_Isel(str_rx);
764     if( Initialized == M_CSG) // Uebersetzung CSG
765         switch_csg(str_rx);
766     if( Initialized == M_ZETA) // Uebersetzung Zeta
767         switch_Zeta(str_rx);
768     if( Initialized == M_TERMINAL)// Uebersetzung Terminal
769         switch_Terminal(str_rx);
770     }
771 }
772 // LCD und LED Stuff
773 void lcd_my_type (char *s) {          // Zeichen auf Display ausgeben
774     // Spielerei !
775     // Zeichen mit unterschiedlicher Geschwindigkeit ausgeben
776     // Dies Simuliert einen Menschlichen Benutzer...
777     srand(TCNT0);
778     int min = 10;
779     int max = 250;
780     int erg = 0;
781     while (*s) // so lange *s != '\0' also ungleich dem "String-Endezeichen(Terminator)"
782     {
783         erg = (rand() % (max - min + 1) + min);
784         lcd_putc(*s);
785         s++;
786         for (int i = 0; i < erg; i++)
787             _delay_ms(1);
788     }
789 }
790 void lcd_boot (void) {                // Boot: Nachricht ausgeben
791     _delay_ms(100);
792     lcd_my_type("Guten Tag!\n");
793     _delay_ms(400);
794     lcd_my_type("Bereit!\n");
795     _delay_ms(400);
796     lcd_clrscr();
797 }
798 void led_boot (void) {                // Boot: LEDs durchlaufen
799     for (int i = 1; i < 9; i++) {
800         _delay_ms(80);                  // warte 80ms
801         LED_PORT &= ~(1 << i); // loescht Bit an PortB - LED an
802         LED_PORT |= ((1 << (i - 1))); // setzt Bit an PortB - LED aus
803     }
804 }
805 void debounce_init (void) {           // Taster entprellen
806     KEY_DDR &= ~ALL_KEYS; // configure key port for input
```

A. Anhang

```

807 KEY_PORT |= ALL_KEYS; // and turn on pull up resistors
808 TCCR0B = (1 << CS02) | (1 << CS00); // divide by 1024
809 TCNT0 = (uint8_t) (int16_t) -(F_CPU / 1024 * 10 * 10e-3 + 0.5); // preload for 10ms
810 TIMSK0 |= 1 << TOIE0; // enable timer interrupt
811 sei();
812 }
813 void led_lauflicht (void) // LED Lauflicht
814 {
815     uint8_t i = LED_PORT;
816     i = (i & 0x00) | ((i << 1) & 0xFE);
817     if (i < 0xFE) i |= 0x01;
818     LED_PORT = i;
819 }
820 // Menu Stuff
821 void mod_manual (void *arg, void *name) { // Manuelle Aufnahme
822     lcd_puts("Manueller Modus\n");
823     lcd_puts("Aufnahme starten!\n");
824     lcd_puts("Danach Select\n");
825     lcd_puts("-> Drehung um 45\n");
826     if (get_key_press(1 << KEY4))
827         uart_put_string("M 55750\r", D_Stepper);
828 }
829 void my_select (void *arg, char *name) { // Deprecated?
830     lcd_clrscr();
831     lcd_puts("Selected: ");
832     lcd_puts(name);
833     ms_spin(750);
834 }
835 void menu_puts (void *arg, char *name) { // Menu/Sende Funktion
836     uart_put_string(arg, D_Stepper); // Uebergebenen String an Stepper senden
837     // Befehl auf Display ausgeben
838     lcd_clrscr();
839     lcd_puts("Sent: ");
840     lcd_puts(arg);
841     lcd_puts("\n");
842     ms_spin(100);
843     //if ((UCSR1A & (1 << RXC1)))
844     uart_rx(D_Stepper); // Antwort des Stepper empfangen
845     ms_spin(1000); // Antwort noch eine weile anzeigen
846 }
847 // Init Stuff
848 void init_WDT (void) // Watchdog Initialisieren
849 {
850     // Vordefinierte Sequenz aus Anleitung
851     cli();
852     wdt_reset();
853     WDTCR |= (1 << WDCE) | (1 << WDE);
854     WDTCR = (1 << WDE) | (1 << WDIE) | (1 << WDP3) | (1 << WDP0); //Watchdog 8s
855     //WDTCR = 0x0F; //Watchdog Off
856     sei();

```

A. Anhang

```

855 }
856 void init(void) { // Initialisierung durchlaufen
857     init_WDT(); // Watchdog Initialisieren oder Abschalten
858     LED_DDR = 0xFF; // LED Port Richtung definieren (Ausgang)
859     LED_PORT = 0xFF; // LEDs ausschalten
860     PCMSK3 |= ( 1 << PCINT28 ); // Interrupts definieren PD4 als Interrupt zulassen
861     PCICR |= ( 1 << PCIE3 ); // Pin Change Interrupt Control Register – PCIE3 setzen
862     fuer PCINT30
863     DDRC |= ( 1 << PB7 ); // Pin7 (Kontrast) als Ausgang definieren (Nur LCD an
864     STK500)
865     LCD_PORT &= ( 1 << PB7 ); // Pin7 auf 0V legen (Nur LCD an
866     STK500)
867     lcd_init(LCD_DISP_ON_CURSOR); // LC Display initialisieren
868     lcd_boot(); // Kurze Startup Meldung zeigen
869     led_boot(); // Starten des Mikrocontroller kennzeichnen
870     debounce_init(); // Taster entprellen
871     uart_init(); // RS-232 Verbindung initialisieren
872     menu_enter(&menu_context, &menu_main); // Kommentar entfernen um Menue zu
873     aktivieren
874 }

```

A. Anhang

A.8. Credits

Den folgenden Personen gebührt mein Dank:

| | |
|---|---|
| Betreuer Hilfestellung bei Problemen aller Art | Prof. Dr. Carstens-Behrens Prof. Dr. Carstens-Behrens Herr Bildhauer André Steimers sowie allen weiteren Personen aus dem Labor bei Herrn Steimers |
| Lektoren | André Steimers Daniel Frank Sarina Steinke |
| 3D Grafiken | Benjamin Dielmann |

Congratulations! You've reached the end.