Vorläufige Arbeitskopie!

Übersetzen von Schrittmotorprotokollen

Entwurf eines Hardwareübersetzers

Praxisbericht

im Fachgebiet Mess- und Sensortechnik

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

Inhaltsverzeichnis

Abkürzungsverzeichnis

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

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis

1. Einleitung

1.1. Ziel der Arbeit anhand eines Beispiels

Abbildung 1.1.: Blick auf den Arbeitsaufbau

1.1.1. Vorhanden waren:

1. Arbeitstisch mit integriertem Drehtisch
2. Computer
3. 3D-Laserscanner VI-900
4. 19”-Rack mit 2 Schrittmotorkarten

1.1.2. Vorgaben des betreuenden Professors:

Aufbau eines Übersetzers, basierend auf einem Mikrocontroller.
Der Übersetzer sollte ein LC-Display, mehrere Taster, mehrere LEDs und zwei serielle Schnittstellen enthalten. Die Höhenverstellung des Drehtisches sollte genutzt werden und die zu Beginn noch nicht funktionierenden Endschalter sollten die vorgesehene Funktion erfüllen.
Im konkreten Beispiel ging es um die automatische 3D-Erfassung eines Schädelmodells. In der Ausgangssituation war es zwar möglich, mit der Erfassungssoftware RapidForm2004 und dem 3D-Laserscanner VI-900 einen Schädel aus einer Richtung zu erfassen. Die Kommunikation zwischen dem VI-900 und der Erfassungssoftware funktionierte. Die Drehtischsteuerung, welche den Drehtisch nach den Vorgaben der Erfassungssoftware des Computers drehen sollte, war nicht in das System eingebunden. Dies war ein Problem der verschiedenen Befehlssätze.

1.1.3. Die Zielvorgabe:

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

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

1.2. Motivation

Die 3D-Lasererfassung bietet zahlreiche Anwendungsgebiete.

- Qualitätskontrolle und Bauteilprüfung für Guss- und Spritzgusstechnik
- Erstellung von Finite-Elemente-Daten für Blechteile im Karosseriebau usw. in Verbindung mit Bauteilanalyse
- Erstellung von 3D-Daten zur Kontrolle von Zubehörteilen und anderen Zukaufteilen, für die keine 3D-CAD-Daten verfügbar sind (z.B. Reverse Engineering)
- Umwandlung von Daten aus der Zahnmedizin und der plastischen Chirurgie in ein Datenbankformat
- Erstellen von Konstruktionsdaten aus Mustern und Verzugs-Prüfung an mechanischen Bauteilen
- Integration in Rapid-Prototyping-Systemen für die Erstellung von Mustern aus Kunststoff
- Vergleich von Eigenprodukten mit Produkten des Mitbewerbs und Umwandlung der erfassten Daten in ein Datenbankformat
- Archivierung in Museen und Forschungseinrichtungen
- Erstellung von Daten für die CAE-Analyse
- 3D-Daten von beliebigen Objekten für unterschiedlichste Forschungszwecke
- Informations- und Kommunikationstechnik, Mimikanalyse, Muskelbewegungsanalyse, Robot-Vision und Wachstumskontrolle landwirtschaftlicher Erzeugnisse
- Unterschiedliche Anwendungen in der Filmindustrie

?

Im konkreten Fall soll nun die Erstellung von 3D-Daten eines vorhanden Objektes (*Reverse Engineering*) genutzt werden.
Es wird nun mit einer Kombination aus dem 3D-Laserscanner, dem Drehtisch und der dazugehörigen Erfassungssoftware ein 3D-Modell erfasst. Dieses kann dann unterstützend in der *CAD-Entwicklung* genutzt werden kann.
In der CAD-Entwicklung kann es vorkommen das für ein real existierendes Objekt eine *Erweiterung* konstruiert werden soll. Um die Erweiterung, einen Anschlag zum Beispiel, leichter konstruieren zu können, ist es von Vorteil, die Abmessungen des Objektes möglichst genau zu kennen. Das Übertragen der Abmessungen in die CAD-Software kann, insbesondere für komplexe Objekte, sehr aufwendig sein. Abhilfe soll

der 3D-Laserscanner schaffen, der das Objekt aus mehreren Richtungen vermisst und aus diesen Informationen ein 3D-Modell generiert. Dieses soll dann in einem neutralen CAD-Format **(TODO: welches?)** exportiert werden.

2. Aufbau der Arbeit

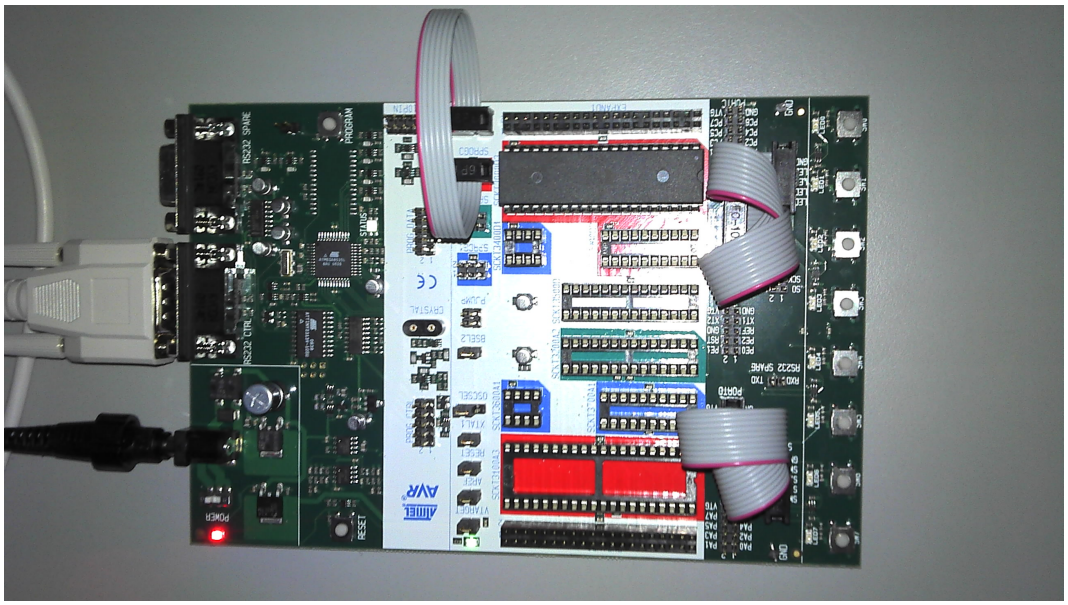

Abbildung 2.1.: STK500

2.1. Erste Schritte

Im ersten Schritt ging es darum, den Drehtisch mithilfe des Mikrocontrollers um 90° zu drehen. Der Mikrocontroller befindet sich auf dem STK 500.

2.1.1. Taster entprellen

Um Taster vernünftig nutzen zu können müssen diese Entprellt werden. Dazu kann die Bibliothek von Peter Fleury genutzt werden. Mit Zeile 1 des Code Listing 3.2 wird diese Bibliothek eingebunden. Zeilen 3-10 inintialisieren die Bibliothek. Danach kann mit den Funktionen get_key_press der Status der Taster prellfrei ausgelesen werden.

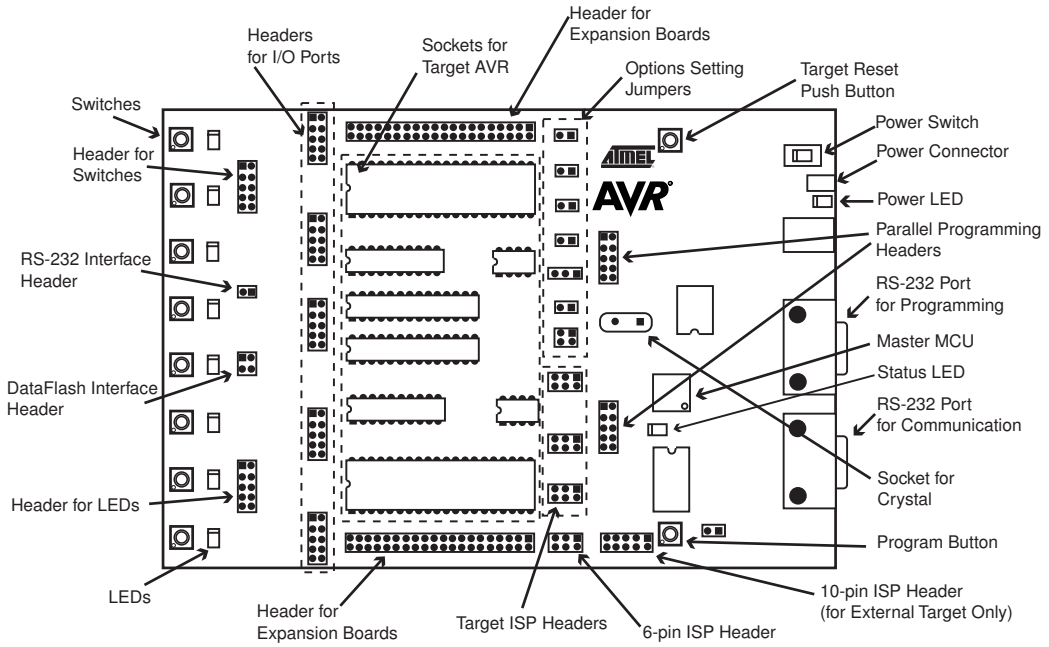

Abbildung 2.2.: STK500 - Schema

Listing 2.1: Taster

```
1  #include "Debounce.h"
2
3  void  debounce_init  (void) {
4    KEY_DDR &= ~ALL_KEYS; // configure key port for input
5    KEY_PORT |= ALL_KEYS; // and turn on pull up resistors
6    TCCR0B = (1 << CS02) | (1 << CS00); // divide by 1024
7    TCNT0 = (uint8_t) (int16_t) -(F_CPU / 1024 * 10 * 10e-3 + 0.5); // preload for 10ms
8    TIMSK0 |= 1 << TOIE0; // enable timer interrupt
9    sei();
10 }
11
12 if   (get_key_press(1 << KEY0) || get_key_rpt(1 << KEY0)){
13   lcd_puts("Betrete Menue!\n");
14   menu_enter(&menu_context, &menu_main);
15 }
```

2.1.2. LEDs ansteuern

2.1.3. LCD ansteuern

2.1.4. Serielle Schnittstelle ansteuern

Der Aufbau der Arbeit gliedert sich im Wesentlichen in die Nutzung vorhandener und die Entwicklung neuer Hardware, sowie in die Entwicklung der Software für den Mikrocontroller und eine Schritt-für-Schritt Anleitung.
Zur Hardware gehören der 3D-Laserscanner, die Ansteuerung für den Drehtisch sowie der Drehtisch selbst, seine Spannungsversorgung, die Schrittmotoren und die Schrittmotorkarten, die Motorverkabelung, die Endschalter, sowie der Mikrocontroller, der Pegelwandler MAX232, ein LC-Display, als auch das Platinenlayout und der 19”-Einschub.
Zur Software gehören die 3D-Erfassungsoftware, die Entwicklungsumgebungen und die Software für den Mikrocontroller. Die Software für den Mikrocontroller deckt das Reverse-Engineering der Protokolle, deren Auswertung und Übersetzung ab; außerdem eine manuelle Ansteuerung des Drehtisches.
Im Anhang befindet sich eine Schritt-für-Schritt Anleitung, 3D-Modelle aufzunehmen und zu exportieren.

3. Hardware

Die Hardware besteht im wesentlichen aus den Komponenten in Abbildung 3.1. (TODO: Nummern oder Farben in Bild!) (TODO: Komponenten

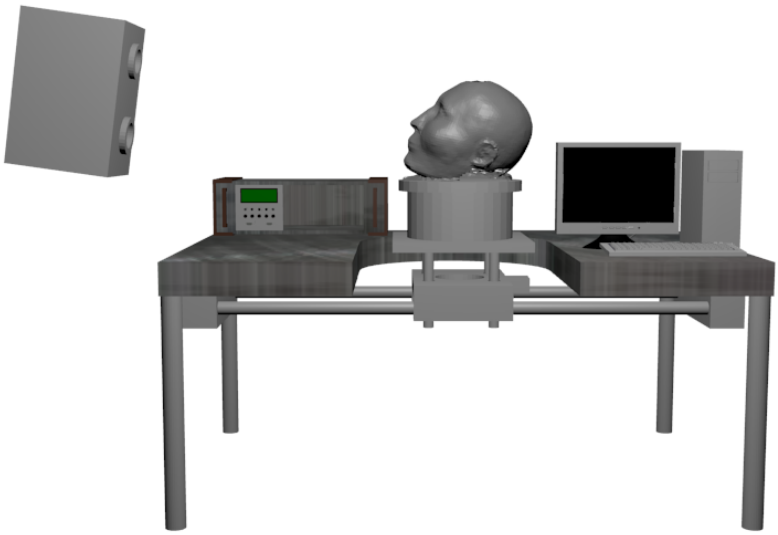

Abbildung 3.1.: Blick auf den Arbeitsaufbau

auf Abbildung erwähnen!)

3.1. 3D-Laserscanner VI-900

Der 3D-Laserscanner *VI-900* der Firma Minolta besteht, wie auf Abbildung 3.2 zu sehen, aus einem *Lasertriangulator*(unten) und einer Kamera(oben). Das System lässt sich über eine *SCSI*-Schnittstelle ansprechen und konfigurieren. Zur mobilen Nutzung kann das Gerät auch auf der Rückseite bedient werden. Aufgenommene Daten können auf einer *CF-Karte* gespeichert werden. Im Projekt wurde jedoch lediglich die direkte Ansteuerung via SCSI genutzt.

Abbildung 3.2.: VI-900 - 3D-Scanner

3.1.1. Lasertriangulator Prinzip

Ein Lasertriangulator, wie in Abbildung 3.3 zu sehen, besteht aus einem Laser, einem Linsensystem und im einfachsten Fall aus einer Pixeldetektorzeile. Der Laser strahlt auf ein Objekt und je nach Entfernung des Objektes wird das Streulicht unter einem anderen Winkel zurückgestrahlt. Das Streulicht wird durch die Linsen auf den Pixeldetektor abgebildet. Über die Position des Laserspots auf dem Pixeldetektor lässt sich auf die Entfernung des Objektes schließen.
Der VI-900 digitalisiert Objekte durch ein Laser-Lichtschnittverfahren. Das vom Objekt reflektierte Licht wird von einer CCD-Flächenkamera erfasst, nach Ermittlung der Distanzwerte (Z-Achse) mittels Laser-Triangulation werden die 3D-Daten erstellt. Der Laserstrahl wird mit Hilfe eines hochpräzisen galvanischen Spiegels über das Objekt projiziert, pro Scan werden 640 x 480 Einzelpunkte erfasst.**?**
Die Technischen Daten befinden sich im Anhang in Tabelle A.1

3.2. Ansteuerung für den Drehtisch

Die Ansteuerung für den Drehtisch ist in einem 19”-Rack verbaut(siehe Abbildung 3.4).

3.2.1. Drehtisch

Der Drehtisch(siehe Abbildung 3.1) ist eine Eigenkonstruktion der Werkstatt des RheinAhrCampus. Er besteht aus einer massiven Edelstahl Arbeitsplatte, welche auf 4 Füßen ruht. Aus dieser ist ein Rechteck mit aufgesetztem Halbkreis ausge-

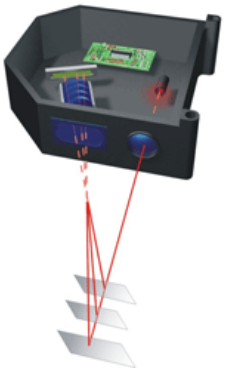

Abbildung 3.3.: Prinzip: Laser-Triangulation

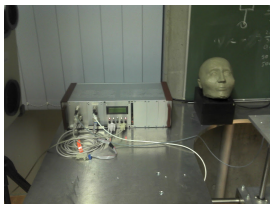

Abbildung 3.4.: Ansteuerung im 19"-Rack

schnitten. In diesem Ausschnitt befindet sich der Drehtisch. 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 beträgt 1:50.

3.2.2. Spannungsversorgung

Die Schrittmotorkarten werden von einem PC-Netzteil gespeist. Die Kabel waren direkt an die Verbindungsleisten gelötet. Um den Aufbau modular und erweiterbar zu machen, ersetzte ich die feste Lötverbindung durch eine Standard PC-Netzteil Verbindung(siehe Abbildung 3.6). Dadurch kann das Netzteil nun einfach ausgebaut werden, bzw. das System leicht mit neuen Einschubkarten erweitert werden.
Die *Logikbausteine* der Schrittmotorkarten und die Mikrocontroller-Platine werden mit 5V gespeist. Zusätzlich werden die Schrittmotorkarten mit 12V für die Schrittmotoren gespeist. (**TODO:** http://www.kosatec.de/prod_images/kc/640x480/100539.jpg)

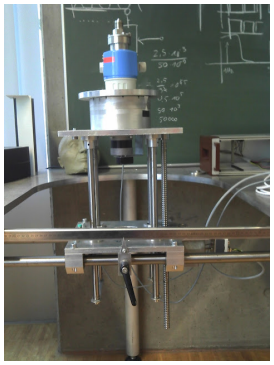

Abbildung 3.5.: Drehtisch

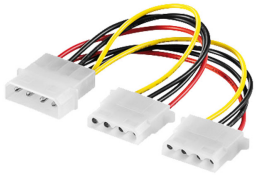

Abbildung 3.6.: Stromverbinder - Y-Kabel

3.2.3. Schrittmotoren

(TODO: Motoren beschreiben! Technische Daten! Schritte, Spannungen. Verdrahtung.)

3.2.4. Schrittmotorkarten

Die Ansteuerung für die Schrittmotoren sind als 19"-Einschübe realisiert. Für jeden Schrittmotor wird ein Einschub benötigt. Die Einschübe sind Produkte der Firma R+S. Mittels *RS-232 Schnittstelle* lassen sich die Karten konfigurieren und ansteuern. Die Konfiguration und Ansteuerung erfolgt über einen vorgegeben *ASCII*[1] Befehlssatz. Der Befehlssatz befindet sich im Kapitel 4.3.8. Es können zwei oder mehr Karten als *Daisy-Chain*[2] in Reihe geschaltet werden.

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

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

3.2.5. Motorverkabelung

Die Schrittmotoren benötigen ein mindestens 4-adriges Kabel. Das Kabel für den Schrittmotor, der für die Rotation zuständig ist, war bereits gefertigt. Das Kabel für den Schrittmotor der für die Höhenverstellung zuständig ist, fertigte ich selbst. Hier wurden 3 weitere Adern für die beiden Endschalter benötigt.
Abbildung 3.8 zeigt eine schematische Darstellung des Kabel. Tabelle 3.1 gibt die Belegung der Kabel wieder. **(TODO: Belegung überprüfen!)**

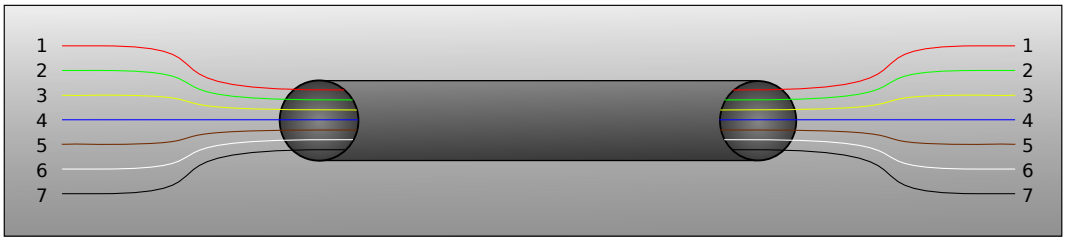

Abbildung 3.7.: Motor- und Endschalterverkabelung

Tabelle 3.1.: Motor- und Endschalterverkabelung

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

3.2.6. Endschalter

Die Schrittmotorkarten unterstützen das Abschalten der Motoren wenn ein sogenannter Endschalter ausgelöst wird. Dies sind im allgemeinen mechanische Schalter die ausgelöst werden wenn der Tisch sich dem Ende des Arbeitsbereiches nähert. Dies verhindert eine Beschädigung des Aufbaus.
Im Aufbau waren bereits induktive Endschalter der Firma Pepperl+Fuchs verbaut. Normalerweise unterstützt die Schrittmotorkarte nur mechanische Endschalter. Durch geschickte Verdrahtung ließen sich die induktiven Endschalter verwenden. Hierzu musste über einen Spannungsteiler die Spannung herabgesetzt werden. Dadurch

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

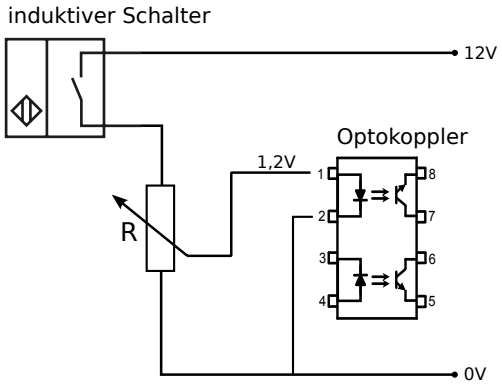

Abbildung 3.8.: Motor- und Endschalterverkabelung

Am Drehtisch war ein Metallstutzen angebracht der den Endschalter auslösen sollte. Dieser war jedoch ungeeignet da er nicht dicht genug an den induktiven Schalter heran kam, obwohl der Tisch sich bereits in der Endposition befand. Zur Abhilfe lies ich einen längeren Metallstutzen von der Werkstatt anfertigen.
Wenn der Tisch sich in der Endposition befindet, soll dies auch auf dem Mikrocontroller angezeigt werden. Die Signale der Endschalter liegen auf der Rückseite der Schrittmotorkarte (TODO: Zeichnung der Anschlüsse referenzieren.) am Verbindungsstecker an. Ich lötete eine Brücke zwischen den Verbindungssteckern der Schrittmotorkarte und des Mikrocontrollers.
Auf der Mikrocontrollerplatine sind diese Pins mit dem Mikrocontroller verbunden. Die beiden Pins werden im Mikrocontroller als *Interrupts* definiert. Die Interrupt-Service-Routine wird in Kapitel 4.3.7 beschrieben.

3.3. Mikrocontroller

Ein *Mikrocontroller* vereint, in einem IC, die wichtigsten Komponenten um komplexe technische Probleme leicht lösen zu können. Dazu gehören z.B. CPU, Flash-Speicher, Arbeitsspeicher, Register, Ports, ADC, DAC und mehr. Einen schematischen Überblick über die Komponenten eines Mikrocontrollers bietet das Blockdiagramm in Abbildung 3.9.
Um Aufgaben lösen zu können ist es notwendig ein Programm für den Mikrocontroller zu schreiben. Dies lässt sich am einfachsten in einer Entwicklungsumgebung(siehe

Kapitel 4.2). Diese Programme können dann z.B. Signale an Pins des Mikrocontrollers auswerten und Signale über andere Pins wieder ausgeben und dadurch externe Geräte steuern. Eingehende Signale können binär ausgewertet, oder mit einem ADC die Spannungshöhe bestimmt werden. Ausgehende Signale können auch binär oder mit einem DAC analog ausgegeben werden. Binäre Signale können zur Steuerung von LEDs oder Peripherie-Geräten genutzt werden. Auch LC-Displays und serielle Schnittstellen können so angesteuert werden. Für unterschiedliche Aufgaben sind unterschiedliche Mikrocontroller geeignet.
Zu Beginn stand mir ein ATmega 8515Atmel [2012b] 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 in die Programmierung mit C einzufinden und eine serielle Schnittstelle anzusteuern.
Für dieses Projekt sind jedoch zwei serielle Schnittstellen nötig. Der ATmega 324AAtmel [2012a] erfüllt diese Voraussetzung. Er ist dem ATmega 8515 recht ähnlich, bietet jedoch die benötigten zwei seriellen Schnittstellen. Des Weiteren hat er 32 Kbyte Flash. Diese wurden aufgrund des recht umfangreichen Programms und der diversen Bibliotheken notwendig.

3.3.1. ATmega 324A

(TODO: Technische Daten)

3.3.2. Entwicklerboard STK500

Um Mikrocontroller zu programmieren und die Programmierung zu überprüfen, kann das *Entwicklerboard* STK500(siehe Abbildung 3.10) der Firma ATMEL verwendet werden. Das Board enthält mehrere Mikrocontroller-Steckplätze, 2 serielle Schnittstellen, 8 Taster, 8 LEDs, 2 Erweiterungsports, eine Programmierschnittstelle *ISP*[3] und mehrere Jumper zum Konfigurieren des Boards.
Von den beiden seriellen Schnittstellen kann die eine zur Programmierung des Mikrocontrollers verwendet werden. Die andere kann zur Kommunikation mit dem Mikrocontroller genutzt werden.
Auf dem Board stehen fünf 10 polige Stiftleisten zur Verfügung. Diese sind direkt mit den Ports des Mikrocontroller verbunden und können über Flachbandkabel mit (TODO: Peripherie) wie z.B. Taster, LED, LC-Displays oder seriellen Schnittstellen verbunden werden.

[3] In System Programmer

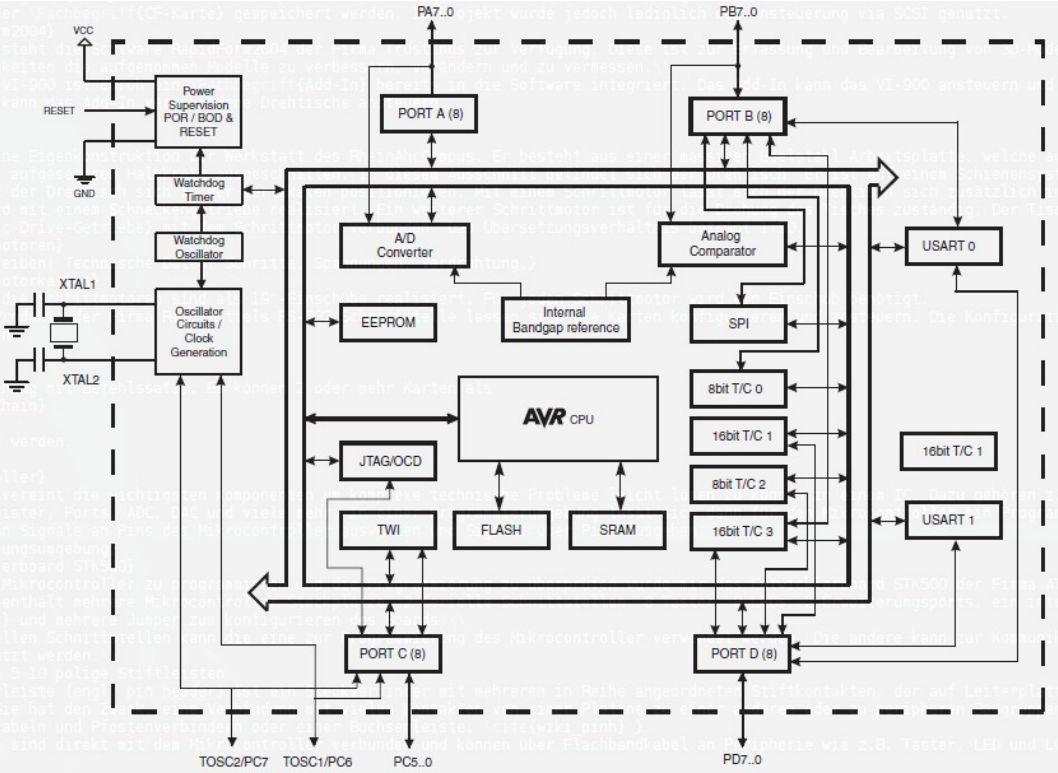

Abbildung 3.9.: Block Diagram: Mikrocontroller
[Atm 2011]

3.3.3. AVRISP mkII

Das AVRISP mkII ist ein USB-basiertes *In-System-Programmiersystem*. Dieses kann anstelle des RS-232 basierten Programmiersystem des STK500 verwendet werden. Die Übertragungsgeschwindigkeit des AVRISP mkII ist wesentlich höher als die der Seriellen Schnittstelle. Desweiteren wurde der ATmega324A nicht mehr vom STK500 internen ISP unterstützt.
Der AVRISP mkII lässt sich einfach an den Programmierport, eine 6-Polige Stiftleiste, des STK500 anschließen.

3.3.4. MAX232

Die Spannungspegel des Mikrocontroller(typ. 0-5 V) sind nicht kompatibel zu den Spannungspegeln des RS-232 Standards (typ. -12-+12 V). Daher wird der *Pegelumsetzer* MAX232 genutzt. Dieser wandelt, mit internen Operationsverstärkern, die Spannungspegel auf für den RS-232 Standard passenden Wert um.

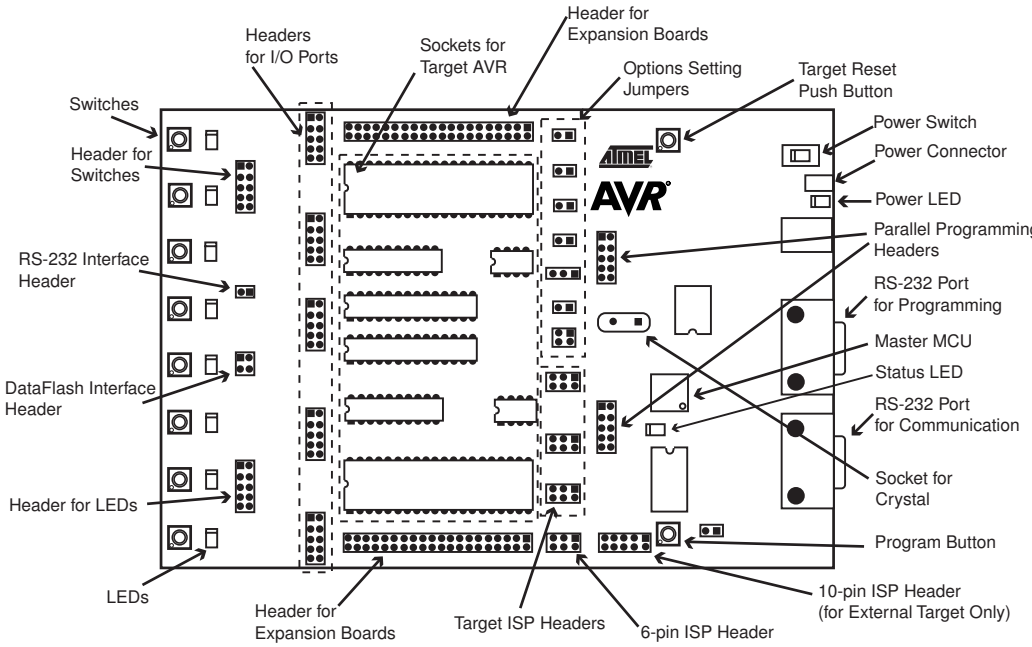

Abbildung 3.10.: Schema: STK500
[?]

3.4. Platinenlayout

Für den Mikrocontroller und seine Peripherie entwickelte ich ein Platinenlayout in der Open Source Software KiCad.
Dazu wurden die Schaltungen wie auf dem STK500 in den Schaltplan übernommen und dort das Layout entwickelt. (TODO: Schaltplan und einbinden.)

3.5. 19”-Einschub

Die Platine für den Mikrocontroller konstruierte ich als 19”-Einschub. Über den rückwärtigen Steckverbinder verband ich die Platine mit der Spannungsversorgung. Zusätzlich kommen hier auch die Signale der Endschalter an. An der Vorderseite befestigte ich eine Blende. Auf der Blende befinden sich das LC-Display, fünf Taster, 5 LEDs und 2 serielle Schnittstellen. Alle Bauteile sind steckbar mit der Platine verbunden. (TODO: Bild des Einschub)

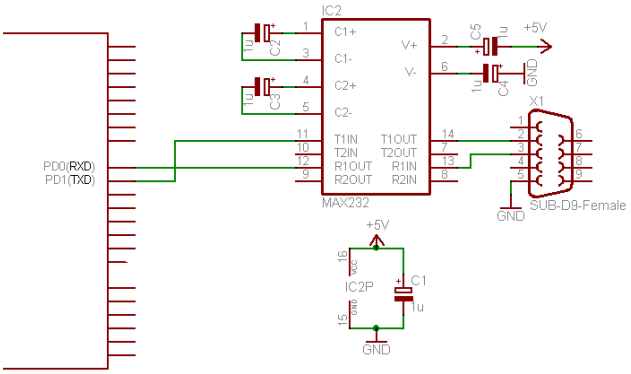

Abbildung 3.11.: Schema: MAX232
[?]

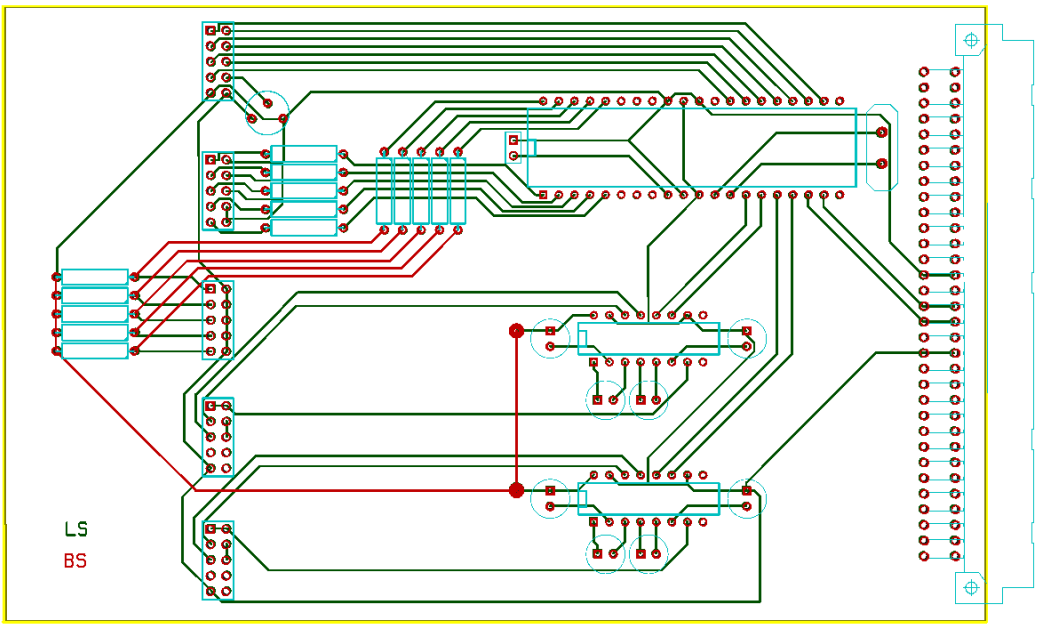

Abbildung 3.12.: Platinenlayout

4. Software

(TODO: Einführung schreiben) (TODO: Weitere Software in Begriffen erklären. Minolta)

4.1. RapidForm2004

Zur Erfassung von 3D-Modellen am PC steht die Software RapidForm2004 der Firma INUS Technology Inc. zur Verfügung. Diese ist zur Erfassung und Bearbeitung von 3D-Modellen gedacht. Sie bietet umfangreiche Möglichkeiten die aufgenommenen Modelle zu verbessern, zu verändern, zu vermessen und in verschiedene Formate zu exportieren.
Die Ansteuerung des VI-900 ist durch ein *Add-In* bereits in die Software integriert. Das Add-In kann den VI-900 ansteuern und die aufgenommenen Daten auslesen. Weiterhin kann das Add-In verschiedene Drehtische ansteuern.

4.2. Entwicklungsumgebung

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

4.2.1. AVR Studio 5

Zu Beginn nutzte ich die von Atmel bereitgestellte Entwicklungsumgebung. Diese scheint jedoch eine fehlerhafte Bibliothek zu enthalten. Die Kombination aus Mikrocontroller ATmega324A und AVR Studio 5 erzeugte nicht nachvollziehbare Probleme. Bei dem selbem Programm und einem anderem Mikrocontroller oder einer anderen Entwicklungsumgebung tauchten keine Fehler auf. In der Entwicklungsumgebung Eclipse lies sich der Fehler reproduzieren wenn der Pfad der Atmel Bibliotheken

eingestellt wurde. Die WinAVR Bibliotheken und eine selbst kompilierte *Toolchain* unter Linux zeigten keine Probleme.
Daher wechselte ich zur *Open Source* Entwicklungsumgebung Eclipse. Erst dadurch wurde es möglich erfolgreich zu arbeiten. Außerdem wurde das Projekt dadurch plattformunabhänig und ich nutzte bis auf RapidForm2004 nur noch freie Open Source Software.

4.2.2. Eclipse

Eclipse ist eine in Java programmierte freie Open Source Entwicklungsumgebung für Java. Sie lässt sich durch *Plugins* leicht für viele Sprachen erweitern.
Mit dem CDT-Plugin, dem AVR-Plugin und einer Bibliothek wie z.B. WinAVR für Windows ist Eclipse eine vollwertige Entwicklungsumgebung für Atmel Mikrocontroller. Ergänzt wird diese durch die Programmiersoftware AVR-Dude.

4.3. Mikrocontroller

In diesem Kapitel werde ich einzelne Abschnitte des von mir geschriebenen Programms beschreiben und erklären. Dabei gliedert sich das Kapitel in folgende Unterkapitel:

- 4.3.1 Fuses
- **??** LEDs
- **??** Taster
- **??** LCD-Bibliothek
- **??** RS-232 Schnittstelle
- **??** Menü-Bibliothek

Die Codebeispiele sind dabei thematisch zusammen gefasst und gekürzt um die Lesbarkeit zu gewährleisten. Ein komplettes Codelisting der *main.c* befindet sich im Anhang. Der komplette Code, mit allen Bibliotheken, liegt dem Praxisbericht als CD oder Archiv bei. **(TODO: Backup anlegen und cleanen!)**

4.3.1. Fuses

Als Fuses werden Register bezeichnet mit denen sich, auf Hardwareebene, das Verhalten des Mikrocontrollers verändern lässt.
Im Projekt wurden folgende Fuses problematisch.

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

Tabelle 4.1.: 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

4.3.2. LEDs

Das Codebeispiel 4.1 zeigt ein einfaches Beispiel mit dem sich die Funktionalität der LEDs leicht überprüfen lässt. Bei jedem Aufruf der Funktion wird der aktuelle Status des LED Port abgefragt und der Hexwert um 1 Bit verschoben. Dadurch wird die daneben liegende LED eingeschaltet und die aktuelle aus geschaltet. Wird ein bestimmter Wert überschritten wird der Port wieder auf den Anfangszustand zurück gesetzt.

Listing 4.1: Funktion: Lauflicht

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

4.3.3. Taster

Listing 4.2: Taster

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

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

4.3.4. LCD Bibliothek

Die meisten LC-Displays werden auf die selbe Art angesteuert. Hier gibt es fertige Bibliotheken die frei genutzt werden können. Im Projekt wird die von Peter Fleury? verwendet.
Dazu müssen die Dateien lcd.c und lcd.h in das Arbeitsverzeichnis kopiert werden und die Bibliothek mit *#include(lcd.h)* eingebunden werden.
Anschließend müssen noch in der lcd.h die Daten des Display eingegeben werden. Danach kann das Display mit den Befehlen aus Zeile 15-24 aus dem Codebeispiel 4.3 angesteuert werden.

Listing 4.3: Definitionen: LCD

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

4.3.5. RS-232

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

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

Um die Schnittstellen im Mikrocontroller nutzen zu können wird in Listing 4.4 - Zeile 2 die setbaud.h eingebunden. Zuvor muss noch die Baudrate gesetzt werden. In der Funktion werden die entsprechenden Register im Mikrocontroller gesetzt um die Schnittstellen zu konfigurieren.

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

Listing 4.4: RS-232

```
1  #define BAUD 9600
2  #include <util/setbaud.h>
3
4  void  uart_init   () {
5    UBRR0H = UBRRH_VALUE;  // UART 0 - IN (Rapidform Software/Terminal)
6    UBRR0L = UBRRL_VALUE;
7    UCSR0C = (3 << UCSZ00);
8    UCSR0B |= (1 << TXEN0);  //Transmitter Enabled
9    UCSR0B |= (1 << RXEN0);  // UART RX einschalten
10
11   UBRR1H = UBRRH_VALUE;  // UART 1 - OUT (Stepper Karte/Drehtisch)
12   UBRR1L = UBRRL_VALUE;
13   UCSR1C = (3 << UCSZ00);
14   UCSR1B |= (1 << TXEN1);  //Transmitter Enabled
15   UCSR1B |= (1 << RXEN1);  // UART RX einschalten
16 }
17 void  uart_put_charater (unsigned char c, int dir) {
18   if (dir == D_RapidForm) { // To Rapidform
19     while (!(UCSR0A & (1 << UDRE0))) {}//warten bis Senden moeglich
20     UDR0 = c;    // sende Zeichen
21   }
22   else {          // To Stepper
23     while (!(UCSR1A & (1 << UDRE1))) {}//warten bis Senden moeglich
24     UDR1 = c; // sende Zeichen
25   }
```

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

4.3.6. Menü Bibliothek

Der Drehtisch kann Manuell über Taster am Einschub bedient werden. Die Menü Bibliothek gestaltet dies einfach und Komfortabel. Mit den Tasten Zurück. Select, Hoch und Runter lässt sich durch die Einzelnen Menü Punkte Navigieren. **(TODO: Ref! Einbinden erläutern!) (TODO: Menü Baum erstellen!)**

4.3.7. Interrupts

Viele Mikrocontroller bieten die Möglichkeit zeitkritische Subroutinen auszuführen. Wenn einer der Interrupts ausgelöst wird, wird das Hauptprogramm unterbrochen und die Entsprechende Interrupt-Service-Routine ausgeführt. Nach Beendigung der ISR wird das Hauptprogramm an der vorherigen Stelle wieder aufgenommen.
ISR dürfen nur sehr wenige Befehle enthalten und müssen innerhalb weniger Clock-Cicles abgeschlossen sein.
Interrupts können z.B. der Überlauf eines internen Timer sein, oder ein externens Signal an einem Pin.
Im Projekt werden externe Interrupts, Timer-Überlauf Interrupts und der Watchdog Interrupt genutzt.

4.3.7.1. Endschalter

Die Endschalter sind über die Schrittmotorkarten und eine Brücke in der Steuerung mit der Mikrocontroller Platine Verbunden. Dort sind sie an 2 Interrupt Pins angeschlossen. **(TODO: Pins raus suchen!)** Bei einem Flanken Wechsel an den Pins wird ein Interrupt ausgelöst.
Das Code-Listing 4.5 zeigt die ISR für die Endschalter.

Listing 4.5: ISR: Endschalter

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

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

4.3.7.2. Watchdog

Der *Watchdog* ist eine Sicherungseinrichtung des Mikrocontroller. In regelmäßigen Abständen wird überprüft ob das Watchdog Bit gesetzt ist und anschließend zurück gesetzt. Das Bit muss innerhalb der voreingestellten Zeit immer wieder neu gesetzt werden. Dies kann mit der wdt_reset() Funktion realisiert werden. Ist das Bit nicht gesetzt, wird der Mikrocontroller zurückgesetzt. **(TODO: Inverse Logik?)** Dies geschieht z.B. bei nicht geplanten Endlosschleifen.
Wahlweise kann kurz vor dem Reset noch die Watchdog-ISR durchlaufen werden.
Im Projekt wird in der ISR die Fehler LED eingeschaltet und eine Meldung auf dem LC-Display ausgegeben. Siehe hierzu auch Listing 4.6 Zeilen 12-15.

Listing 4.6: Watchdog

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

4.3.8. Protokoll der Schrittmotorkarte

Tabelle 4.2 zeigt den ASCII Befehlssatz der Schrittmotorkarte.

4.3.9. Manueller Betrieb

4.3.10. Protokolle aus RapidForm

Die Protokolle die RapidForm nutzt um mit den Schrittmotoren zu kommunizieren können leicht mit der Software **(TODO: Name)** abgehört werden. Dies hat jedoch den Nachteil das RapidForm erst den nächsten Befehl sendet wenn der erste richtig quittiert wurde. Die Befehle die RapidForm erwartet konnten aus den Betriebsanleitungen der Schrittmotoren entnommen werden. Dies war jedoch auch nicht immer leicht.
Durch *Reverse-Engineering* konnten alle Befehle und der Antwort die darauf erwartet wird, aus der Executebale ausgelesen werden. Listing 4.7 zeigt einen Auszug für das Protokoll eines Zeta Schrittmotors. Im Anhang befinden sich alle Protokolle.

Listing 4.7: Protokoll aus Rapidform: Zeta

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

4.3.11. Übersetzungs Logik

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

4.3.11.1. FindStringInArray Funktion

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

Listing 4.8: Funktion: FindStringInArray

```
int      FindStringInArray      (const char* pInput, const char* pOptions[], int cmp_length) {
         int n = -1;
         while (pOptions[++n]) { //Array durchlaufen bis 0 terminiert
                  //Wenn pInput == pOptions dann gib Array Position zurueck
```

```
5                   if (!strncmp(pInput, pOptions[n], cmp_length)) return n;
6           }
7           return 99;
8   }
```

4.3.11.2. Automatische Protokollwahl

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

Listing 4.9: Funktion: switchMotor

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

4.3.11.3. Zeta

Zeta

4.3.11.4. Isel

(TODO: Splitten und jeden Abschnitt einzeln erklären!) Auch hier wird wieder ein Array mit möglichen ankommenden Befehlen definiert und der ankommende String gegen dieses Array geprüft. Kommt z.B. die Sequenz "@01än, wird case 3 ausgewählt. Das Display zeigt Init an und sendet die Antwort die RapidForm erwartet zurück.
Die Sequenz "@0Rßteht für eine Statusabfrage. Es wird eine Statusabfrage an die Schrittmotorkarte gesendet und der Status an RapidForm zurück gemeldet.

Listing 4.10: Übersetzungs Logik: Isel

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

```
34          case 5:                     // 5 - Gehe zu Position MX , +600
35                  ms_spin(10);
36                  char Position[33], Winkel[6];
37                  memset(Position, '\0', 33);
38                  memset(Winkel, '\0', 6);
39                  String_zerlegen_Isel(str_rx, Position, Winkel);
40                  char Move_To[40];
41                  memset(Move_To, '\0', 40);
42                  Move_To[0] = 'M';
43                  Move_To[1] = 'A';
44                  Move_To[2] = ' ';
45                  Move_To[3] = '\0';
46                  strcat(Move_To, Position);
47                  strcat(Move_To, "\n");
48                  lcd_puts("Pos:");
49                  lcd_puts(Move_To);
50
51                  uart_put_string(Move_To, D_Stepper);
52                  ms_spin(50);
53                  if ((UCSR1A & (1 << RXC1)))
54                          uart_rx(D_Stepper);
55                  else {
56                          break;
57                  }
58
59                  uart_put_string("A\n", D_Stepper);
60                  ms_spin(50);
61                  if ((UCSR1A & (1 << RXC1)))
62                          uart_rx(D_Stepper);
63                  else {
64                          lcd_puts("Keine Bewegung!\n");
65                  }
66
67                  while (!strcmp(str_rx,"1#")){
68                          uart_put_string("A\n", D_Stepper);
69                          ms_spin(50);
70                          if ((UCSR1A & (1 << RXC1))){
71                                  uart_rx(D_Stepper);
72                                  lcd_clrscr();
73                                  lcd_puts("Gehe zu Winkel: ");
74                                  lcd_puts(Winkel);
75                                  lcd_puts("\n");
76                          }
77                          else {
78                                  lcd_puts("Keine Antwort\n");
```

```
79                          }
80                          wdt_reset();
81                  }
82                  lcd_puts("Winkel: ");
83                  lcd_puts(Winkel);
84                  lcd_puts(" Erreicht\n");
85                  uart_put_string("0\r\n", D_RapidForm);
86                  break;
87          default:
88                  lcd_puts(str_rx);
89          }
90  }
```

4.3.11.5. Weitere

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

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

Tabelle 4.2.: ASCII Befehlssatz R+S Schrittmotorsteuerung

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

5. Fazit und Zukunft

5.1. Fazit

(TODO: Fazit schreiben!)

Literaturverzeichnis

Atm 2011
Atmel (Hrsg.): *ATmega164A/PA/324A/PA/644A/PA/1284/P Complete*. San Jose, CA 95131, USA: Atmel, 06 2011 3.9

Atmel 2012a
Atmel: *ATmega324A- Atmel Corporation*. http://www.atmel.com/devices/ATMEGA324A.aspx. Version: 2012. – [Online; Stand 11. Februar 2012] 3.3

Atmel 2012b
Atmel: *ATmega8515- Atmel Corporation*. http://www.atmel.com/devices/ATMEGA8515.aspx. Version: 2012. – [Online; Stand 11. Februar 2012] 3.3

V9141 2001
RS (Hrsg.): *Schrittmotor-Platine mit integriertem Treiber*. Mörfelden-Walldorf: RS, 03 2001 4.2

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

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

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich den vorliegenden Bericht:

> *Übersetzen von Schrittmotorprotokollen*
> *Entwurf eines Hardwareübersetzers*

selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe.
Remagen, den 22. Februar 2012

Johannes Dielmann

A. Anhang

A.1. Schritt für Schritt Anleitung

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

A.2. Technische Daten VI-910

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

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

Modellbezeichnung	Optischer 3D-Scanner VI-910
Messverfahren	Triangulation durch Lichtschnittverfahren
Autofokus	Autofokus auf Objektoberfläche (Kontrastverfahren); aktiver AF
Objektive (wechselbar)	TELE Brennweite f=25mm MITTEL: Brennweite f=14 mm WEIT: Brennweite f=8mm
Messabstand	0,6 bis 2,5m (2m für WIDE-Objektiv)
Optimaler Messabstand	0,6 bis 1,2m
Laserklasse	Class 2 (IEC60825-1), Class 1 (FDA)
Laser-Scanverfahren	Galvanisch-angetriebener Drehspiegel
Messbereich in X-Richtung (anhängig vom Anstand)	111 bis 463mm (TELE), 198 bis 823mm (MITTEL), 359 bis 1.196mm (WEIT)
Messbereich in Y-Richtung (abhängig vom Abstand)	83 bis 347mm (TELE), 148 bis 618mm (MITTEL), 269 bis 897mm (WEIT)
Messbereich in Z-Richtung (abhängig vom Abstand)	40 bis 500mm (TELE), 70 bis 800mm (MITTEL), 110 bis 750mm (WEIT/Modus FINE)
Genauigkeit	X: ±0,22mm, Y: ±0,16mm, Z: ±0,10mm 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: 1/3CCD-Bildsensor (340.000 Pixel) Farbdaten: Zusammen mit 3D-Daten (Farbtrennung durch Drehfilter)
Anzahl aufgenommener Punkte	3D-Daten: 307.000 (Modus FINE), 76.800 (Modus FAST) Farbdaten: 640 × 480 × 24 Bit Farbtiefe
Ausgabeformat	3D-Daten: Konica Minolta Format, (STL, DXF, OBJ, ASCII-Punkte, VRML; Konvertierung in 3D-Daten durch Polygon Editing-Software / Standardzubehör) Farbdaten: RGB 24-Bit Rasterscan-Daten
Speichermedium	Compact Flash Memory Card (128MB)
Dateigrößen	3D- und Farbdaten (kombiniert): 1,6MB (Modus FAST) pro Datensatz, 3,6MB (Modus FINE) pro Datensatz
Monitor	5,7LCD (320 × 240 Pixel)
Datenschnittstelle	SCSI II (DMA-Synchronübertragung)
Stromversorgung	Normale Wechselstromversorgung, 100V bis 240 V (50 oder 60 Hz), Nennstrom 0,6 A (bei 100 V)
Abmessungen (B x H x T)	213 × 413 × 271mm
Gewicht	ca. 11kg
Zulässige Umgebungsbedingungen (Betrieb)	10 bis 40°C; relative Luftfeuchtigkeit 65% oder niedriger (keine Kondensation)
Zulässige Umgebungsbedingungen (Lagerung)	-10 bis 50°C, relative Luftfeuchtigkeit 85% oder niedriger (bei 35°C, keine Kondensation)

A.3. Verwendete Software

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

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

A.4. Verwendete Hardware

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

- **ATMega 324A**
  Atmel
  Website: http://www.atmel.com/devices/ATMEGA324A.aspx
- **STK500**
  Atmel
  Website: http://www.atmel.com/tools/STK500.aspx
- **AVRISP mkII** Atmel
  Website: http://www.atmel.com/tools/AVRISPMKII.aspx