

# List of Figures

3.1	Blick auf den Arbeitsaufbau . . . . .	2
3.2	VI-900 - Kamera oben, Lasertriangulator unten . . . . .	3
3.3	Prinzip: Laser-Triangulation . . . . .	4
3.4	Drehtisch . . . . .	5
3.5	Ansteuerung im 19"-Rack . . . . .	6
3.6	Block Diagramm: Mikrocontroller ATmega324A[?] . . . . .	8
3.7	Schemazeichnung eines STK500[?] . . . . .	9
5.1	Stromverbinder - Y-Kabel[?] . . . . .	21
5.2	Motor- und Endschalerverkabelung . . . . .	22
5.3	Endschalerverkabelung . . . . .	22
5.4	Schaltplan fr die zweite serielle Schnittstelle [?] . . . . .	23
5.5	Schema der Kommunikation . . . . .	24
5.6	Schaltplan . . . . .	31
5.7	Platinenlayout . . . . .	32
5.8	19"-Einschub . . . . .	33

# List of Tables

3.1	Komponenten im Aufbau . . . . .	2
5.1	Motor- und Endschalterverkabelung . . . . .	22
6.1	Fuses . . . . .	36

# Codeverzeichnis

Code/Hinweis.c . . . . .	4
5.1 Taster . . . . .	12
5.2 LEDs . . . . .	13
5.3 lcd.h (Auszug) . . . . .	13
5.4 RS-232 . . . . .	14
5.5 Befehlssatz aus Rapidform: Isel . . . . .	16
5.6 Men . . . . .	17
5.7 Men Baum . . . . .	18
5.8 RS-232 Empfang . . . . .	19
5.9 FindStringInArray() . . . . .	19
5.10 switchStepper() . . . . .	20
5.11 RS-232 Empfang - RapidForm2004 . . . . .	24
5.12 Funktion: uart rx() . . . . .	25
5.13 Funktion: switch Motor() . . . . .	26
5.14 bersetzungs Logik fr einen Isel-Motor . . . . .	27
5.15 case 3: Initialisierung . . . . .	27
5.16 case 4: Statusabfrage . . . . .	28
5.17 case 5: Rotation . . . . .	29
5.18 Funktion: string zerlegen Isel() . . . . .	30
6.1 ISR: Endschalter . . . . .	35
6.2 Watchdog . . . . .	35

# Chapter 1

## Hinweise zum Dokument

1

**Fachbegriffe** werden blau hinterlegt.

*Eigennamen* werden kursiv dargestellt.

**Code-Fragmente** werden gelb hinterlegt.

1

```
//Quelltext wird in diesen Codelistings dargestellt .
```

Der komplette Quelltext für den Mikrocontroller, der Quelltext für diese Dokumentation, die Dokumentation selbst und weitere Bilder können unter <https://github.com/JoeD84/Praxisprojekt> abgerufen werden.

## Chapter 2

# 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-Lasererfassungssystem. 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 Erfassungssoftware 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 Studenten 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 zukünftige Verbesserungen. Im Anhang befindet sich eine Schritt-für-Schritt-Anleitung die es Laien ermöglicht 3D-Modelle aufzunehmen und zu exportieren.

## Chapter 3

# Vorstellung der vorhandenen Hardware

Die Hardware besteht im Wesentlichen aus den Komponenten in Abbildung 3.1.

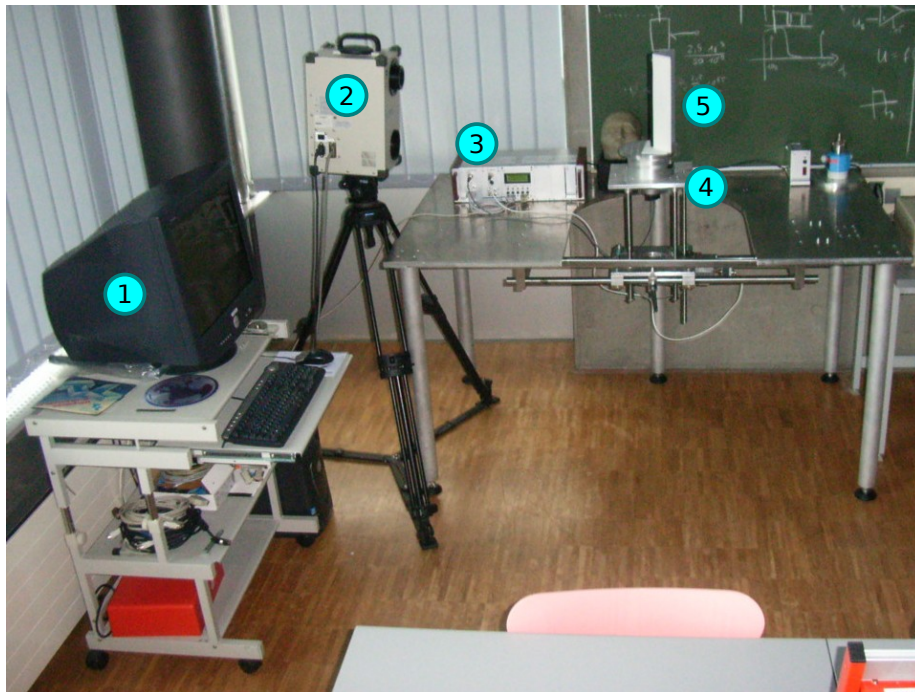


Figure 3.1: Blick auf den Arbeitsaufbau

Table 3.1: Komponenten im Aufbau

1	Computer
2	3D-Laserscanner VI-900
3	Ansteuerung fr den Drehtisch
4	Drehtisch
5	Zu scannendes Objekt (Kalibrierblech)

## 3.1 Computer

Zur Verfgung steht ein IBM kompatibler x86 Standard PC mit einer SCSI- und einer RS-232-Schnittstelle. Auf diesem ist die Erfassungssoftware *Rapid-Form2004* [??] installiert. Die SCSI Schnittstelle wird zur Kommunikation mit dem 3D-Laserscanner und die RS-232-Schnittstelle zur Kommunikation mit einer Schrittmotorsteuerung genutzt.

## 3.2 3D-Laserscanner VI-900

Der 3D-Laserscanner *VI-900* der Firma *Konica Minolta*[??] besteht, wie auf Abbildung 3.2 zu sehen, aus einer Kamera und einem Lasertriangulator. Das System lsst sich ber eine SCSI-Schnittstelle ansprechen und konfigurieren. Zur mobilen Nutzung kann das Gert auch auf der Rckseite bedient werden. Aufgenommene Daten knnen 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-Flchenkamera erfasst, nach Ermittlung der Distanzwerte (Z-Achse) mittels Laser-Triangulation werden die 3D-Daten erstellt. Der Laserstrahl wird mit Hilfe eines hochprzisen galvanischen Spiegels ber das Objekt projiziert, pro Scan werden 640 x 480 Einzelpunkte erfasst.[? ]

Die Technischen Daten befinden sich im Anhang in Tabelle ??

### 3.2.1 Lasertriangulator Prinzip

Ein Lasertriangulator besteht, wie in Abbildung 3.3 zu sehen, 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 zurckgestrahlt. Das Streulicht wird durch die Linsen auf den Pixeldetektor abgebildet. ber die Position des Laserspots auf dem Pixeldetektor lsst sich auf die Entfernung des Objektes schlieen.



Figure 3.2: VI-900 - Kamera oben, Lasertriangulator unten

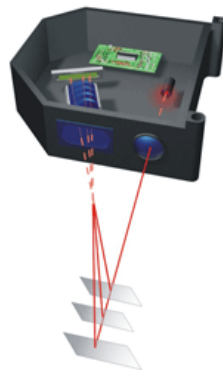


Figure 3.3: Prinzip: Laser-Triangulation

## 3.3 Drehtisch und Ansteuerung

### 3.3.1 Drehtisch

Der Tisch in dem der Drehtisch verbaut ist, ist eine Eigenkonstruktion der Werkstatt des RheinAhrCampus Remagen. Er besteht aus einer massiven Edelstahl-arbeitsplatte, welche auf 4 Füßen ruht. Aus dieser ist ein Rechteck mit aufgesetztem Halbkreis ausgeschnitten. In diesem Ausschnitt befindet sich der Drehtisch (siehe Abbildung 3.4). 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 übersetzt



zungsverhältnis des Getriebes beträgt 1:50.



Figure 3.4: Drehtisch

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

### 3.3.3 Schrittmotoren

Für die Rotation kommt der Schrittmotor 440-458 der Firma R+S zum Einsatz. Dieser hat einen Schrittwinkel von 1,8°, ein Haltedrehmoment von 500mNm, wird mit 8-Drahtleitung verschaltet und mit 12V Gleichspannung versorgt. Aus dem Schrittwinkel 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 Schrittwinkel von 1,8°, 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.

### 3.3.4 Schrittmotorkarten

Die Ansteuerung für die Schrittmotoren sind als 19"-Einschbe realisiert, siehe Abbildung 3.5 links. Für jeden Schrittmotor wird ein Einschub benötigt. Die Einschbe 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 ASCII<sup>1</sup> Befehlssatz. Der Befehlssatz befindet sich im Kapitel ???. Es können zwei oder mehr Karten als Daisy-Chain<sup>2</sup> in Reihe geschaltet werden.

Zu Beginn des Projekts war nur die erste Schrittmotorsteuerung vorbereitet.

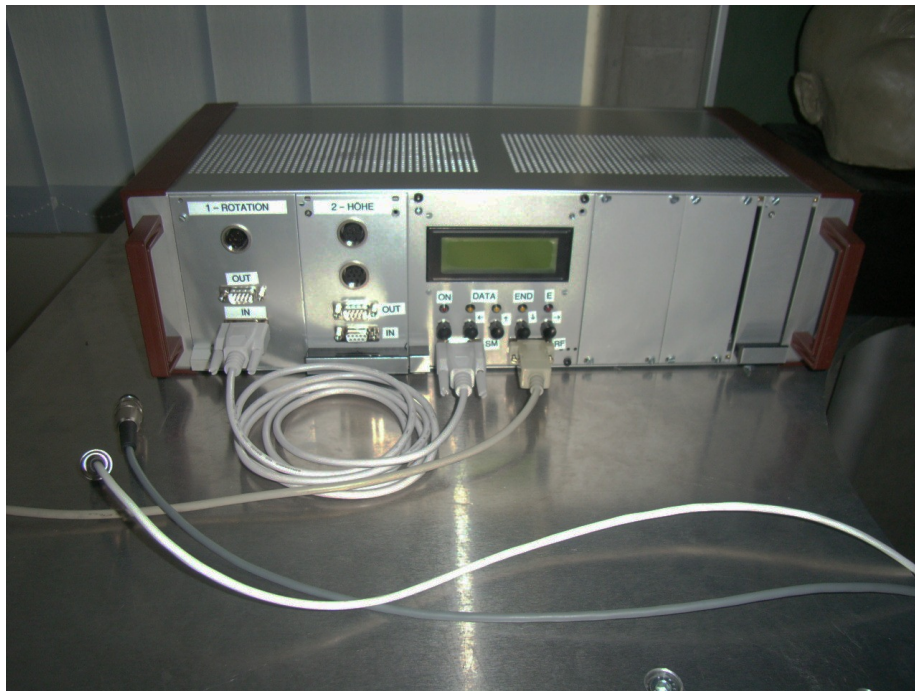


Figure 3.5: Ansteuerung im 19"-Rack

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

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

### 3.3.5 Motorverkabelung

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

### 3.3.6 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.

## 3.4 Mikrocontroller

Ein Mikrocontroller besteht, wie in Abbildung 3.6 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 [?] 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.

### 3.4.1 Entwicklerboard STK500

Um den Mikrocontroller zu programmieren und die Programmierung zu überprüfen, wird das Entwicklerboard *STK500* [??], wie auf Abbildung 3.7 zu sehen, verwendet. Das Board enthält mehrere Mikrocontroller-Steckplätze, 2 serielle Schnittstellen, 8 Taster, 8 LEDs, 2 Erweiterungsports, eine ISP<sup>3</sup> Programmierschnittstelle

---

<sup>3</sup>In System Programmer

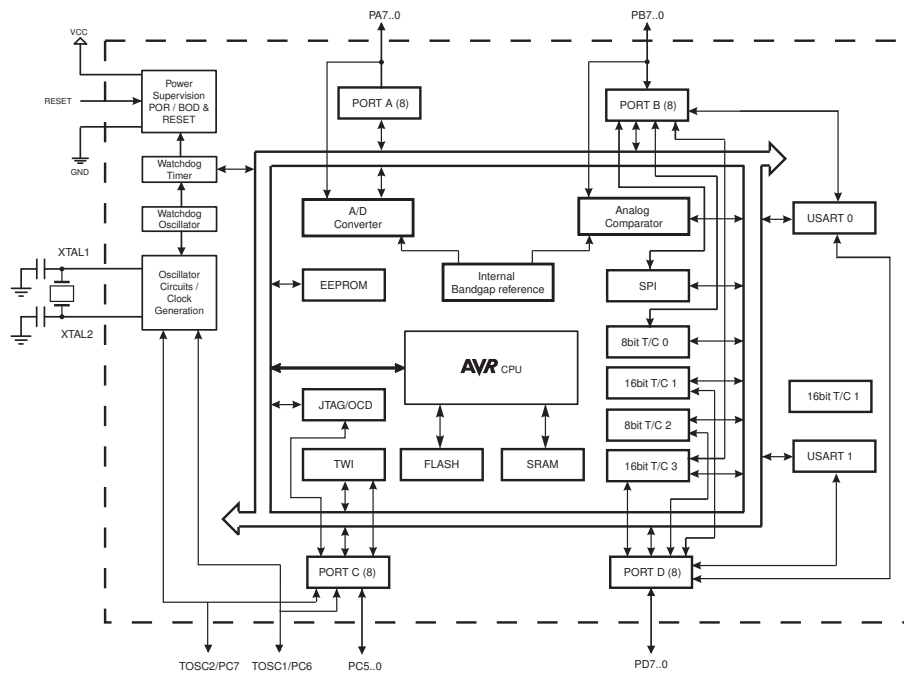


Figure 3.6: Block Diagramm: Mikrocontroller ATmega324A[? ]

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 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.

### 3.4.2 AVRISP mkII

Der *AVRISP mkII* ist ein USB-basierter In-System-Programmer. Dieser kann anstelle des RS-232 basierten Programmiersystems des STK500 verwendet werden.

Die Übertragungsgeschwindigkeit des AVRISP mkII ist wesentlich höher als die der seriellen Schnittstelle. Der AVRISP mkII lässt sich einfach an den Programmierport, eine 6-Polige Stiftleiste, des STK500 anschließen.

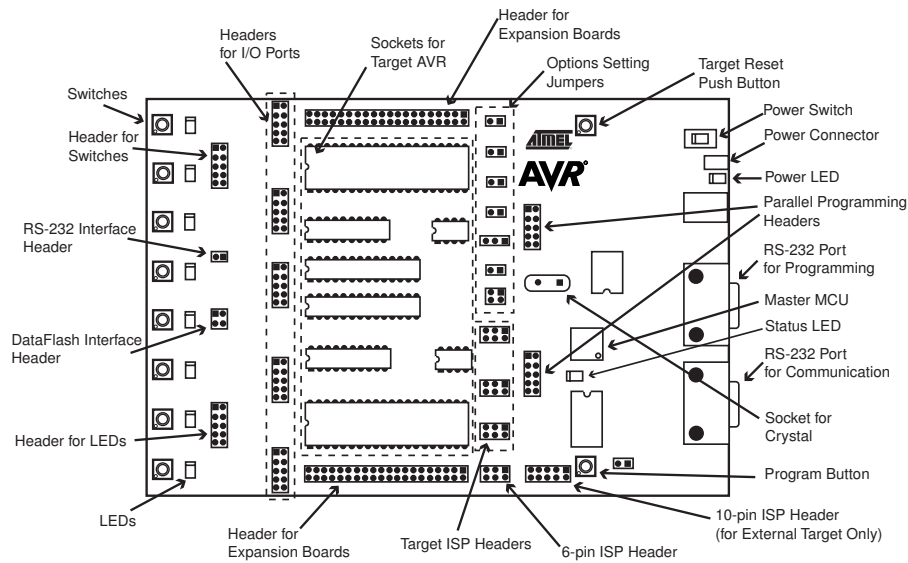


Figure 3.7: Schemazeichnung eines STK500[?] ]

### 3.4.3 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 TTL<sup>4</sup>) auf die Spannungspegel des RS-232 Standards (typ. -12 V – +12 V).

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

## Chapter 4

# Vorstellung der vorhandenen Software

### 4.1 RapidForm2004

Zur Erfassung von 3D-Modellen am PC steht die Software *RapidForm2004* [??] 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 ausgelesen werden. Weiterhin kann das Add-In über eine RS-232-Schnittstelle verschiedene Drehtische ansteuern.

### 4.2 Entwicklungsumgebung

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

### 4.3 Terminalprogramme

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

## Chapter 5

# 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 dass er sozusagen als Übersetzer für die unterschiedlichen Befehlsätze von RapidForm2004 und dem der Schrittmotorkarte fungiert. Kapitel 5.1 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 5.2 beschreibt die Erarbeitung der Befehlsätze 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 5.3 beschreibt wie der Mikrocontroller diesen Befehlssatz für die Kommunikation mit der vorhandenen Schrittmotorkarte nutzt.

Kapitel 5.4 gibt die Schritte zur Entwicklung und Verbesserung der Hardware, um diese so zu erweitern, dass sie den Vorgaben entspricht, wieder.

Kapitel 5.5 erläutert die Kommunikation zwischen dem Mikrocontroller und RapidForm2004.

Kapitel 5.6 beschreibt, wie die vorherigen Kapitel zusammenspielen, sodass eine reibungslose Kommunikation zwischen RapidForm2004 und der vorhandenen Schrittmotorkarte möglich ist.

Kapitel 5.7 beschreibt dann das Erstellen des Platinenlayouts und das Fertigen des Einschubs.

### 5.1 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 3.4.1). Um dessen Komponenten im Mikrocontroller nutzen zu können, müssen dafür Register initialisiert werden oder Funktionalitäten wie z.B. Bibliotheken für das LCD-Display bereit gestellt werden. Die folgenden Kapitel beschreiben den Programmcode, der notwendig ist um diese Funktionalitäten bereitzustellen.

### 5.1.1 Taster

Um die Taster des STK500 im Mikrocontroller nutzen zu können müssen diese mit dem Mikrocontroller verbunden und entprellt<sup>1</sup> 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** [?] 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 5.1 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 5.1: 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 Menüpunkt auswählen

```

### 5.1.2 LEDs

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

Dazu muss zuerst die Stiftleiste von *PortB* mit der LED Stiftleiste des STK500

<sup>1</sup>Als **Prellen** bezeichnet man das ungewollte mehrfache Schalten eines mechanischen Schalters bei einem einzelnen Schaltvorgang.



verbunden werden. Um LEDs an *PortB* betreiben zu können müssen die entsprechenden Pins im Register `DDRB` als Ausgang definiert werden. Dies geschieht in Zeile 1 des Codelisting 5.2. 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 5.2: LEDs

```

1 LED_DDR = 0xFF;           // LED Port Richtung definieren (Ausgang)
2 LED_PORT = 0xFF;         // LEDs ausschalten
3 LED_PORT &= ~(1 << PBX);  // löscht Bit an PortB – LED an
4 LED_PORT |= (1 << PBX);   // setzt Bit an PortB – LED aus

```

### 5.1.3 Ansteuerung des LC-Display

Um den aktuellen Status des Motor komfortabel in Textform anzeigen zu können und die Schrittmotorkarte *menbasiert* 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 [?] 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 5.3 Zeilen 2–10).

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

Listing 5.3: 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                               */
6 #define LCD_LINE_LENGTH 0x40  /**< internal line length of the display */
7 #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 visibile 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))

```

### 5.1.4 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. [? ]

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 Stiftleiste *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 5.4 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

Listing 5.4: 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      UBRRH0H = UBRRH_VALUE;
7      UBRR0L = UBRRL_VALUE;
8      UCSR0C = (3 << UCSZ00);

```

```

9      UCSR0B |= (1 << TXEN0); //Transmitter Enabled
10     UCSR0B |= (1 << RXEN0); // UART RX einschalten
11     // UART 1 – OUT (Stepper Karte/Drehtisch)
12     UBRR1H = UBRRH_VALUE;
13     UBRR1L = UBRL_VALUE;
14     UCSR1C = (3 << UCSZ00);
15     UCSR1B |= (1 << TXEN1); //Transmitter Enabled
16     UCSR1B |= (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 }
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 erhoeuen
58         }
59     } while (i < 100 && c != '\r' && c != '\n'); // So lange bis \r \n o. >100
60     Zeichen
61     *string_in = '\0'; // 0 Terminieren
62     if (dir == D_Stepper)
63         LED_PORT |= (1 << LED3); // "Daten Vorhanden" LED ausschalten
64     else
65         LED_PORT |= (1 << LED2); // "Daten Vorhanden" LED ausschalten

```

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.

## 5.2 Befehlsstze

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 Befehlsstze, 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 Befehlsstze 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, dass RapidForm2004 erst dann den nächsten Befehl sendete, wenn der Erste mit der erwarteten Antwort quittiert wurde. Die Befehle, die RapidForm erwartete, konnten zwar teilweise aus den Betriebsanleitungen der Schrittmotorsteuerungen entnommen werden, dieses Vorgehen war jedoch sehr mühsam. Ein 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 5.5 zeigt einen Auszug für den Befehlssatz eines Isel Schrittmotors. Im Anhang ?? befinden sich die Befehlsstze aller Schrittmotorkarten. Somit stehen die Befehlsstze 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.

Listing 5.5: 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    ""              "0"
7  home    "@0M0\054+600\r" "0"
8  step    "-0.0173076" "-8000000" "8000000"
9  timeout "60"
10 firsttimeout "10"

```

## 5.3 Kommunikation mit der vorhandenen Schrittmotorsteuerung

### 5.3.1 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 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 vorangegangenen Kapiteln, bekannt gemacht werden. Dies geschieht in der Datei `tinymenu/tinymenu.h`.

Die Zeilen 1–6 des Codelisting 5.6 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 5.1.1, erkannt und die entsprechende Menüfunktion aufgerufen.

Listing 5.6: 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  // Gekürzte Main-Funktion
8  int main(void) {
9      while (1) { // In Endlosschleife wechseln
10         wdt_reset(); // Watchdog zurücksetzen
11         if (get_key_press(1 << KEY1)) // 1 – Zurück
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 Menpunkte wurden realisiert:

Listing 5.7: Men Baum

```

1 Main Menu
2   Bewegen – Rotation
3     +90
4     -90
5     +10.000 Schritte
6     -10.000 Schritte
7     Gehe zum Ursprung
8   Bewegen – Hoehe
9     +500000
10    -500000
11    +1000000
12    -1000000
13    Gehe zum Ursprung
14  Konfigurieren
15    Motorstatus
16    Setze Ursprung
17    Write to EEPROM
18    Newline 1
19    Parameter Auslesen

```

Wird einer der Menpunkte aufgerufen, wird die im Menpunkt hinterlegte Funktion mit dem hinterlegten Parameter aufgerufen. Wird beispielsweise der Befehl `+90` ausgewählt, wird die hinterlegte Funktion `menu_puts(arg, name)` (Codelisting 5.6 Zeile 18-28) mit dem hinterlegten Wert aufgerufen. Diese sendet dann mit der aus Kapitel 5.1.4 bekannten Funktion `uart_puts(arg, dir)` einen Befehl an die Schrittmotorsteuerung.

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

### 5.3.2 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* weiter gegeben werden.

Dazu wird in der Hauptschleife des Programms ständig das Eingangsregister der ersten seriellen Schnittstelle abgefragt (siehe Codelisting 5.8 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)` liest dann das Empfangsregister mit der aus Kapitel 5.1.4 bekannten Funktion `uart_get_string(str_rx, dir)` aus und schreibt den empfangenen String in die Variable `str_rx` (Codelisting 5.8, 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 5.8, 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 5.8: 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); // Übersetzungsfunktion fuer Stepper aufrufen
10     else { // Empfangsregister RapidForm
11         // Wird spaeter erklart
12     }
13 }
```

### 5.3.3 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 5.9) gegen ein **Array** (Codelisting 5.10, 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 5.9: FindStringInArray()

```

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

```

Listing 5.10: 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
10        Antworten 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 }

```



## 5.4 Verbesserungen an der vorhandenen Hardware

### 5.4.1 Netzteil

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

Die festen Ltverbindungen am Einschub fr die Schrittmotorkarte wurden durch Standard PC-Netzteil Stecker ersetzt. Die Logikbausteine der Schrittmotorkarte werden mit 5V gespeist. Die Schrittmotorkarte wird zustzlich mit 12V fr den Schrittmotor gespeist. Der Stecker lsst sich nun einfach mit einer Buchse des Standard PC-Netzteils verbinden und es ist nicht mehr Notwendig zu lten wenn das Netzteil ausgebaut wird. Mittels eines Y-Kabels(siehe Abbildung 5.1) knnen nun leicht weitere Buchsen hinzugefgt werden.

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



Figure 5.1: Stromverbinder - Y-Kabel[? ]

### 5.4.2 Zweite Schrittmotorkarte

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

Dazu wurde, wie in Kapitel 5.4.1 beschrieben, ein weiterer Einschubplatz fr die Schrittmotorkarte vorbereitet. Die Karte wurde mit einer Frontblende versehen und auf dieser eine Buchse fr die Motorverkabelung und je eine Buchse und einen Stecker fr die seriellen Schnittstellen verbaut. Diese wurden mit den entsprechenden Anschlssen auf der Schrittmotorkarte verlftet. 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 ber die Erste angesteuert werden.

Somit steht eine baugleiche Schrittmotorkarte zur Verfugung. Diese kann nun den Schrittmotor fr die Hhenverstellung 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.

### 5.4.3 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 7 adriges Kabel (siehe Abbildung 5.2) 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 5.1 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 ermittelt werden konnte.



Figure 5.2: Motor- und Endschalterverkabelung

Table 5.1: Motor- und Endschalterverkabelung

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

### 5.4.4 Endschalter

Nun sollen die vorgegeben 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 Endscharter nicht ohne weiteres nutzen. Um die induktiven Endscharter nutzen zu können, musste die Spannung über einen Spannungsteiler herabgesetzt werden und die standardmäßigen Eingänge für die mechanischen Endscharter umgangen werden. Die induktiven Endscharter werden direkt an den Optokoppler angeschlossen, welcher für die mechanischen Endscharter zuständig ist. Dadurch wurden die Signale der Endscharter für die Schrittmotorkarte nutzbar. Ein weiteres Problem bestand darin, dass, wenn der Tisch sich bereits

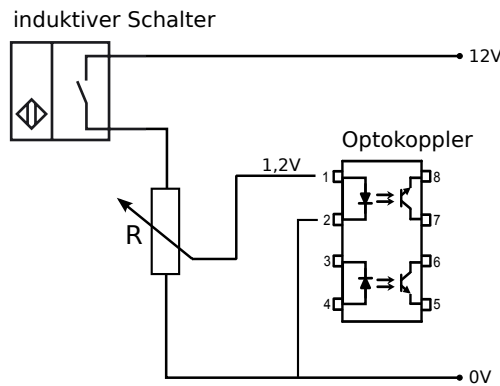


Figure 5.3: Endscharterverkabelung

in der Endposition befand, die Endscharter noch nicht aktiviert wurden. Dies lag daran, dass der Metallstutzen, der die Endscharter 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 Endscharter 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 getätigt. 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 6.2.1 beschrieben.

Da die Signale der Endscharter nun an der Schrittmotorkarte anliegen, stoppt diese den Motor wenn eine der Endscharterpositionen erreicht wird. Zusätzlich liegen die Signale am Mikrocontroller an. Dieser gibt dadurch auf dem Display die Meldung *Endscharterposition erreicht!* aus.

#### 5.4.5 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.

Dafr wurde vorerst auf einem Steckbrett eine zweite serielle Schnittstelle nach dem Schaltplan in Abbildung 5.4 aufgebaut. Spter wird diese Schnittstelle direkt auf der Mikrocontroller-Platine realisiert. Dadurch ist es mglich mit dem Mikrocontroller ber zwei RS-232-Schnittstellen gleichzeitig zu kommunizieren.

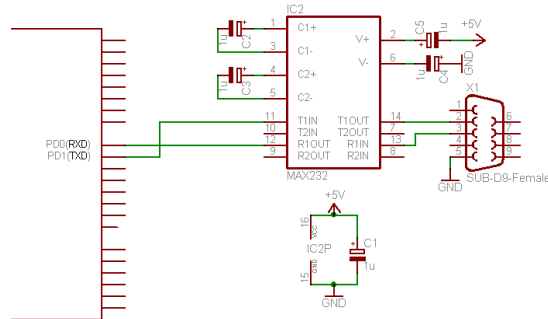


Figure 5.4: Schaltplan fr die zweite serielle Schnittstelle [? ]

## 5.5 Kommunikation mit RapidForm2004

RapidForm2004 sendet Befehle die fr die Drehtischsteuerung bestimmt sind an den Mikrocontroller. Diese sollen dort empfangen, ausgewertet und in verstndlicher Form an die Drehtischsteuerung weiter gegeben werden. RapidForm2004 verwendet dabei verschiedene Befehlsstze fr verschiedene Schrittmotorsteuerungen. Fr jeden dieser Befehlsstze 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. Abschlieend soll eine entsprechende Antwort an RapidForm2004 zurck gesendet werden. Abbildung 5.5 zeigt eine schematische bersicht dieser Kommunikation.

Die Kommunikation mit RapidForm2004 ist hnlich zu der mit der Schrittmotorsteuerung. Diese wurde bereits in Kapitel 5.3 ausfhrlich beschrieben. Daher wird die Kommunikation hier etwas oberflchlicher behandelt.

### 5.5.1 Befehle empfangen

Zuerst sollen nun die Befehle von RapidForm2004 an den Mikrocontroller, gespeichert werden. Anschlieend wird die automatische Auswahl des Befehlssatzes beschrieben.

Um anstehende Befehle zu empfangen wird, hnlich wie in Kapitel 5.3.2, eine

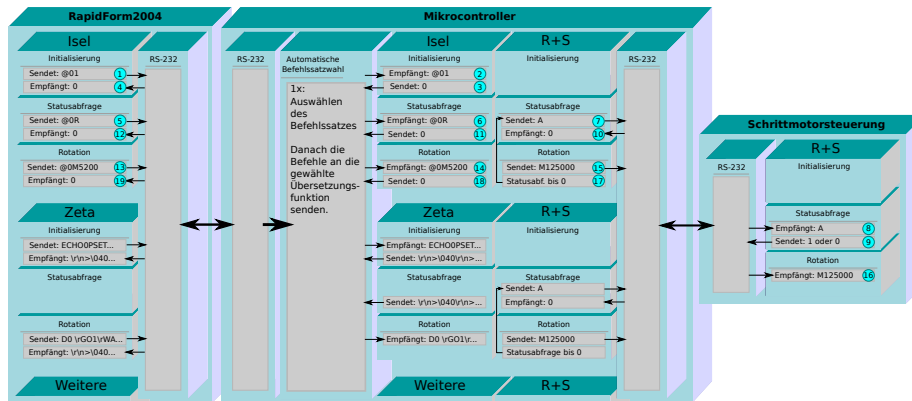


Figure 5.5: Schema der Kommunikation

Funktion die stndig das Eingangsregister der ersten seriellen Schnittstelle abfragt verwendet (siehe Codelistung 5.11). 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 knnen nun auch Strings von RapidForm2004 empfangen und in der Variablen `str_rx` gespeichert werden.

Listing 5.11: 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 }

```

### 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 fr den Befehlssatz wird in einer globalen Variable gespeichert und alle weiteren Befehle werden an die entsprechende Auswerte-Funktion fr diesen Befehlssatz bergeben.

In der Funktion `uart_rx(dir)` (Codelistung 5.12) wird nun in der ersten *if-Abfrage* entschieden, von welcher Schnittstelle der empfangene Befehl kam. Diese verzweigt nun, da `D_RapidForm` bergeben wurde, in den else-Teil. In diesem wird mit mehreren *if-Abfragen* berprft, ob bereits der Befehlssatz fr einen bestimmten Motor ausgewählt wurde. Ist dies nicht der Fall, wird der empfangende String an die Funktion `switch_Motor(str_rx)` (Codelistung 5.13)

bergeben. Diese prfte mit der aus Kapitel 5.3.3 bekannten Funktion `FindStringInArray(str_rx, pOptions,` den angekommenen String gegen die Initialisierungsbefehle der einzelnen Befehlssatze. Die Initialisierungsbefehle sind die ersten Befehle die RapidForm2004

an eine Schrittmotorsteuerung sendet um zu prüfen ob diese vorhanden ist. In diesem ersten Schritt wird der String nur zur Identifizierung des von RapidForm2004 verwendeten Befehlssatzes 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 5.13 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 empfangenen 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 5.12: 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);  // Uebersetzung Stepper
6     else {                      // Empfangsregister RapidForm
7         // Uebersetzungsfunktion auswählen
8         if (Initialized == M_UNK){ // Unbekannter Initialisierungsbefehl
9             lcd_puts("Unbekannter Motor!\n");
10            Initialized = M_NOTI; // Variable Initialized zurücksetzen
11        }
12        if (Initialized == M_NOTI){ // Befehlssatz bestimmen
13            Initialized = switch_Motor(str_rx); //Automatische
14                                   // Befehlssatzwahl
15        }
16        if (Initialized == M_ISEL) // Uebersetzung ISEL
17            switch_Isel(str_rx);
18        if (Initialized == M_CSG)  // Uebersetzung CSG
19            switch_csg(str_rx);
20        if (Initialized == M_ZETA) // Uebersetzung Zeta
21            switch_Zeta(str_rx);
22        if (Initialized == M_TERMINAL) // Uebersetzung Terminal
23            switch_Terminal(str_rx);
24    }

```

Listing 5.13: 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 }

```

## 5.6 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 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.

### 5.6.1 Auswerte-Funktion für Isel-Motoren

Wird der Befehl `@01` empfangen, Bergibt die in Kapitel 5.5.1 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 5.3) und bei der automatischen Auswahl des Be-

fehlssatzes (Kapitel 5.5.1. In der Funktion `switch_Isel(str_rx)` sind in dem Array `pOptions` alle benötigten Befehle des Isel-Befehlssatzes hinterlegt. Mit der aus Kapitel 5.3.3 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 5.14: Übersetzungs Logik für einen Isel-Motor

```

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

```

## Initialisierung

Für den String `@01` wird `case 3` ausgeführt. Dieser Codeblock zeigt die Meldung *Init* auf dem Display an und sendet den erwarteten Befehl an RapidForm2004.

Listing 5.15: case 3: Initialisierung

```

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

```

## Statusabfrage

Wird der String `@0R` 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 5.3, 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 5.16: 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
6         wurde
7         uart_rx(D_Stepper); // Zeichen auslesen
8         if (!strcmp(str_rx, "0#")) // Empfangenes Zeichen
9             ueberpruefen
10            uart_put_string("0\r\n", D_RapidForm); // Antwort Ok an RF melden
11        else {
12            lcd_puts("Fehlgeschlagen \n"); // Fehler auf Display anzeigen
13            uart_put_string("1\r\n", D_RapidForm); // Fehler an RapidForm melden
14        }
15        break;

```

## Rotation

Der Codeblock von **case 5** ist fr die Rotation verantwortlich. Es werden je ein String fr die Endposition und fr den Winkel mit Stringterminierungszeichen vorbelegt. Diese werden an die Funktion **String\_zerlegen\_Isel(str\_rx, Position, Winkel)** (Codelisting 5.18) bergeben. 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 fr 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 durchgefñhrt. Diese wird beim Kompilieren durch eine **Bitverschiebung** ersetzt. Dies spart mehrere Sekunden Rechenzeit und die Abweichung der Schritte betrgt nur maximal 3 Schritte. Die berechnete Anzahl der Schritte wird anschlieend wieder als String gespeichert. Dieser wird dann an den String fr den Rotationsbefehl der Schrittmotorkarte angehngt. Der neue String wird auf dem Display ausgegeben und an die Schrittmotorssteuerung gesendet. Die Antwort der Schrittmotorsteuerung wird ausgelesen und anschlieend 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 5.17: 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); // Strign 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);
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 5.18: 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         }
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 Winkels
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

```

## 5.7 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"-Einschbe 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 5.6) übernommen. Anschließend wird das Layout im Layouteditor entwickelt. Dabei waren mehrere enge Vorgaben einzuhalten. Da in der Werkstatt des RheinAhrCampus keine Platinen mit Durchkontaktierungen hergestellt werden können, sollen Vias vermieden, IC-Sockel, Kondensatoren und Potis nur auf der Unterseite verlegt werden.

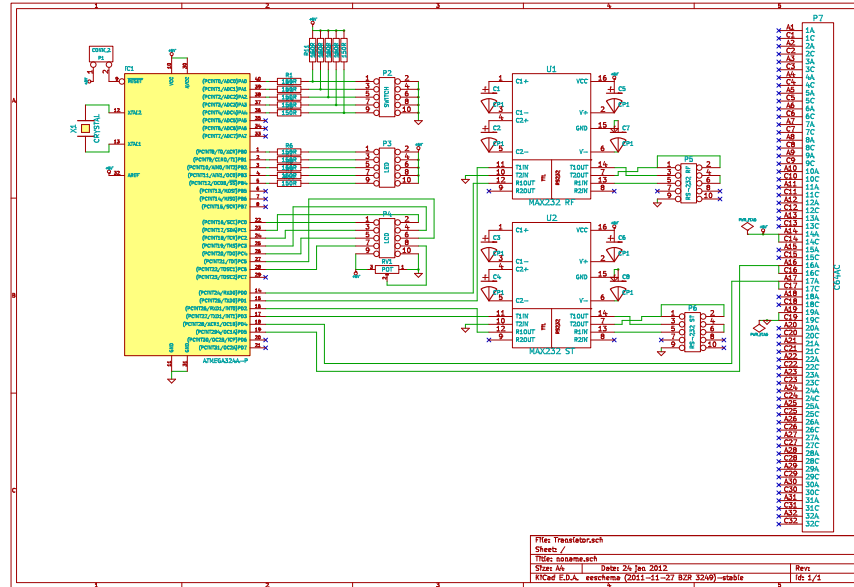


Figure 5.6: Schaltplan

den. 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* [??] bereitgestellt. Da der Autorouter die Aufgabe nicht zufriedenstellend lösen konnte, mussten viele Verbindungen nachträglich manuell angelegt werden. Das fertige Layout (siehe Abbildung 5.7) wurde von der Werkstatt am RheinAhrCampus gefertigt und anschließend bestückt. Der 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.

Dadurch sind alle im Projekt verwendeten Komponenten auf einem 19"-Einschub (siehe Abbildung 5.8) vereint.

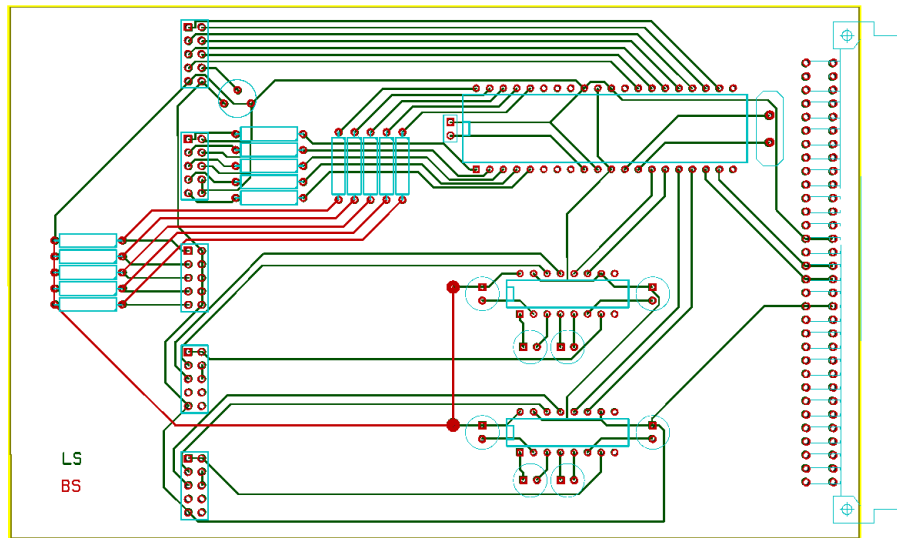


Figure 5.7: Platinenlayout

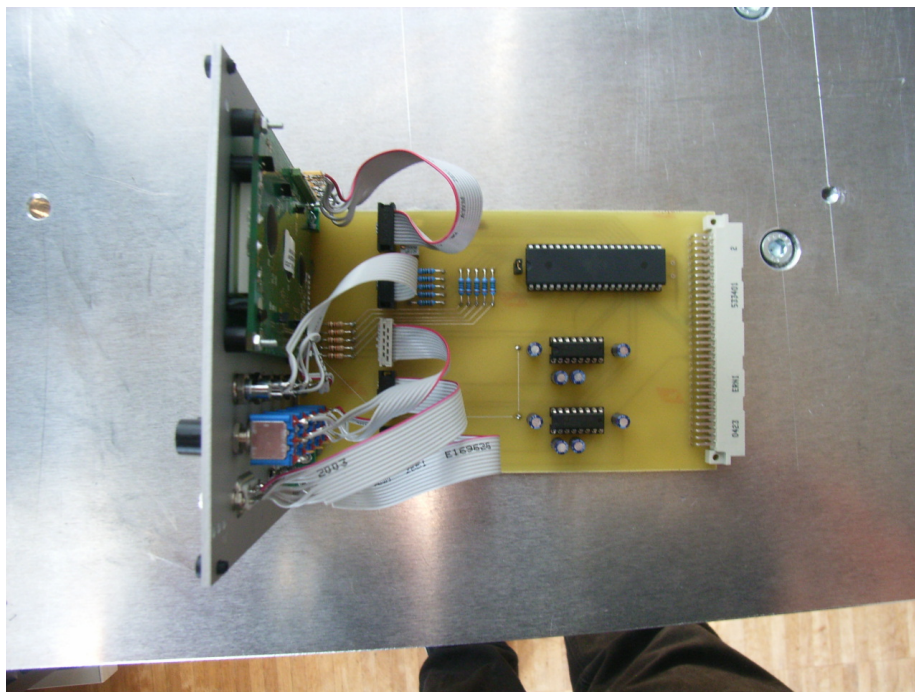


Figure 5.8: 19"-Einschub

## Chapter 6

# Probleme und Lsungen

### 6.1 Entwicklungsumgebungen

#### 6.1.1 AVR Studio 5

*AVR Studio 5* [??] ist eine von Atmel bereitgestellte Entwicklungsumgebung. Diese scheint jedoch eine fehlerhafte Bibliothek zu enthalten. Die Kombination aus Mikrocontroller ATmega324A und AVR Studio 5 erzeugte nicht nachvollziehbare Probleme. Bei dem selbem Programm und einem anderem Mikrocontroller oder einer anderen Entwicklungsumgebung tauchten keine Fehler auf. In der Entwicklungsumgebung *Eclipse* [??] lies sich der Fehler reproduzieren wenn der Pfad der 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 plattformunabhngig. Bis auf RapidForm2004 wurde somit nur noch freie Open Source Software verwendet.

#### 6.1.2 Eclipse

Eclipse ist eine in Java programmierte freie open source Entwicklungsumgebung fr Java. Sie lsst sich durch **Plugins** leicht fr viele Programmiersprachen erweitern.

Mit dem *CDT-Plugin*, dem *AVR-Plugin*, der Bibliothek von *WinAVR* und der Programmiersoftware *AVRDude* ist Eclipse eine vollwertige Entwicklungsumgebung fr Mikrocontroller von Atmel.

## 6.2 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 und sollten innerhalb weniger Taktzyklen abgeschlossen sein. Interrupts können z.B. der Verlauf eines internen Timer, oder ein externes Signal an einem Pin sein. Im Projekt werden Externe-Interrupts für die Endschalter, Timer-Verlauf-Interrupts für das Entprellen der Taster und der Watchdog-Interrupt zum Erkennen von Fehlern genutzt.

### 6.2.1 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. Bei einem Flankenwechsel an den Pins wird ein Interrupt ausgelöst.

Mit den Zeilen 1–2 des Codelistings 6.1 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 6.1: ISR: Endschalter

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

### 6.2.2 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 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 Codelistings 6.2 wird der Watchdog initialisiert und

festgelegt in welchen Zeitintervallen das Watchdog-Bit überprüft werden soll. Der Ablauf zum Einstellen des Zeitintervalls muss genau wie im Datenblatt des Mikrocontroller beschrieben eingehalten werden. Dies verhindert ein versehentliches Ändern der Einstellung. Ist das Fusebit *WDTON* gesetzt kann der Watchdog nicht abgeschaltet werden (siehe Kapitel 6.3).

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 6.2 Zeilen 12-16).

Listing 6.2: 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 \nterribly wrong!\nRebooting!");
16 }

```

## 6.3 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 Energiesparender. 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.



- **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.

Table 6.1: Fuses

OCDEN	On Chip Debugging
JTAGEN	Hardware Debugging
SPIEN	Serial Program and Data Downloading
WDTON	Watchdog Timer always on
EESAVE	EEPROM memory is preserved through the Chip Erase
BOOTSZ1	Select Boot Size
BOOTSZ0	Select Boot Size
BOOTRST	Select Reset Vector
CKDIV8	Divide clock by 8
CKOUT	Clock output
SUT1	Select start-up time
SUT0	Select start-up time
CKSEL3	Select Clock source
CKSEL2	Select Clock source
CKSEL1	Select Clock source
CKSEL0	Select Clock source

## Chapter 7

# Fazit und weitere Mglichkeiten

### 7.1 Fazit

Das vorgegeben 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 vollstndig genutzt werden und alle wesentlichen Funktionen der Schrittmotorssteuerungen werden von der Mikrocontroller Programmierung untersttzt. Im Anhang ?? befindet sich eine Schritt-fr-Schritt Anleitung mit der auch Laien das System nutzen knnen.

### 7.2 Bekannte Probleme

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

Das Display zeigt whrend der Rotation  $\theta$  anstatt dem Winkel an, um den rotiert wird. Fr die Anzahl der Schritte funktionierte diese Anzeige. Vermutlich liegt hier ein Fehler in der Berechnung des Winkels vor.

### 7.3 Weitere Mglichkeiten

Eine elegantere Lsung als die Befehle der Befehlsstze in einem Array zu speichern wre 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 Men 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.