

**VORLÄUFIGE ARBEITSKOPIE!**

**ÜBERSETZEN VON**

**SCHRITTMOTORPROTOKOLLEN**

**Entwurf eines Hardwareübersetzers**

**Praxisbericht**

im Fachgebiet Mess- und Sensortechnik



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

© 2012

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

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>III</b>
<b>1. Einleitung</b>	<b>1</b>
1.1. Motivation . . . . .	1
1.2. Ziel der Arbeit . . . . .	1
1.3. Aufbau der Arbeit . . . . .	2
<b>2. Hardware</b>	<b>3</b>
2.1. Lasererfassungssystem VI-900 . . . . .	3
2.2. Ansteuerung für den Drehtisch . . . . .	4
2.2.1. Drehtisch . . . . .	4
2.2.2. Spannungsversorgung . . . . .	4
2.2.3. Schrittmotoren . . . . .	4
2.2.4. Schrittmotorkarten . . . . .	4
2.2.5. Motorverkabelung . . . . .	5
2.2.6. Endschalter . . . . .	5
2.3. Mikrocontroller . . . . .	6
2.3.1. Entwicklerboard STK500 . . . . .	6
2.3.2. AVRISP mkII . . . . .	7
2.3.3. MAX232 . . . . .	8
2.4. Platinenlayout . . . . .	8
<b>3. Software</b>	<b>9</b>
3.1. RapidForm2004 . . . . .	9
3.2. Entwicklungsumgebung . . . . .	9
3.2.1. AVR Studio 5 . . . . .	9
3.2.2. Eclipse . . . . .	9
3.3. Mikrocontroller . . . . .	9
3.3.1. Fuses . . . . .	10
3.3.2. LEDs . . . . .	10

3.3.3. Taster . . . . .	11
3.3.4. LCD Bibliothek . . . . .	11
3.3.5. RS-232 . . . . .	12
3.3.6. Menü Bibliothek . . . . .	13
3.3.7. Interrupts . . . . .	13
3.3.7.1. Endschalter . . . . .	13
3.3.7.2. Watchdog . . . . .	14
3.3.8. Protokoll der Schrittmotorkarte . . . . .	15
3.3.9. Manueller Betrieb . . . . .	16
3.3.10. Protokolle aus RapidForm . . . . .	16
3.3.11. Übersetungs Logik . . . . .	16
3.3.11.1. Zeta . . . . .	16
3.3.11.2. Isel . . . . .	16
3.3.11.3. Weitere . . . . .	16
3.3.12. Automatische Protokollwahl . . . . .	16
<b>4. Fazit und Zukunft</b>	<b>17</b>
4.1. Fazit . . . . .	17
<b>Eidesstattliche Erklärung</b>	<b>19</b>
<b>A. Anhang</b>	<b>i</b>
A.1. Schritt für Schritt Anleitung . . . . .	ii
A.2. Vom Autor verwendete Software . . . . .	iii
A.3. Codelistings . . . . .	iv
A.3.1. main.c . . . . .	iv

## Abkürzungsverzeichnis

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

# Abbildungsverzeichnis

2.1. Überblick des Arbeitsplatz . . . . .	3
2.2. Block Diagram eines Mikrocontroller . . . . .	7

# Tabellenverzeichnis

3.1. ASCII Befehlssatz R+S Schrittmotorsteuerung . . . . .	15
--	----

## Codeverzeichnis

3.1. Funktion - Laufflicht . . . . .	10
3.2. Taster . . . . .	11
3.3. Definitionen - LCD(Auszug) . . . . .	11
3.4. Funktionen - RS-232 . . . . .	12
3.5. ISR - Endschalte . . . . .	13
3.6. Watchdog . . . . .	14
A.1. main.c . . . . .	iv

# 1. Einleitung

**(TODO: DIE EINLEITUNG IST SEHR WICHTIG!!!) (TODO: AUSBAUEN? NEU SCHREIBEN!)** Die 3D-Lasererfassung bietet zahlreiche Anwendungsgebiete. Von der Erfassung kleiner Objekte über die Erkennung von

**(TODO: KLARSTELLEN DER BEGRIFFLICHKEITEN)** In der CAD-Entwicklung kommt es vor das für ein real existierendes Objekt eine Erweiterung konstruiert werden muss. Um die Erweiterung sinnvoll konstruieren zu können müssen dazu die Abmessungen des Objektes möglichst genau bekannt sein. Das übertragen der Abmessungen, kann insbesondere für komplexe Objekte, sehr aufwendig sein. Abhilfe soll ein Laserscanner schaffen der das Objekt aus mehreren Richtungen vermisst und aus diesen Informationen ein genaues 3D-Modell davon generiert.

## 1.1. Motivation

Im Projekt soll nun mit einer Kombination aus einem Lasererfassungssystem, einem Drehtisch und der dazugehörigen Software auf einfachem Wege ein 3D-Modell erfasst werden. Dieses soll dann in einer CAD-Software wie *Solidworks* nutzbar sein.

## 1.2. Ziel der Arbeit

Die Kommunikation zwischen Software und Drehtisch soll ermöglicht werden. Dazu werden die Befehle der Software mit einem Mikrocontroller ausgewertet und in für den Drehtisch verständliche Befehle übersetzt.

Um den Drehtisch manuell bedienen zu können und den aktuellen Status des Drehtisch anzuzeigen sind noch ein LC-Display und mehrere Taster vorgesehen.

Die Ansteuerung des Drehtisches ist als Einschub für ein 19Rack realisiert. Daher wird die Platine für den Mikrocontroller auch als 19Einschub realisiert.



### 1.3. Aufbau der Arbeit

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

Zur Hardware gehören die Auswahl des Mikrocontroller, die Endschalter, die Schrittmotorkarten, die Schrittmotoren, sowie die verwendeten PCs.

Zur Software gehören die Entwicklungsumgebungen und die 3D-Erfassungssoftware.

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

## 2. Hardware

(TODO: BESSERES BILD! SCHEMA? BESSERE BESCHRIFTUNG!)



Abbildung 2.1.: Überblick des Arbeitsplatz

(TODO: FUSSNOTEN FÜR KOMPONENTEN, HERST. U. WEBSITES)  
(TODO: KOMPONENTEN AUF ABILDUNG ERWÄHNEN!)

### 2.1. Lasererfassungssystem VI-900

(TODO: BILD) Das Lasererfassungssystem *VI-900* der Firma *Minolta* besteht aus einem *Lasertriangulator* und einer Kamera. 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 Ansteuerung via SCSI genutzt.

## 2.2. Ansteuerung für den Drehtisch

### 2.2.1. Drehtisch

Der Drehtisch ist eine Eigenkonstruktion der Werkstatt des RheinAhrCampus. Er besteht aus einer massiven Edelstahl Arbeitsplatte, welche auf 4 Füßen ruht. Aus dieser ist ein Rechteck mit aufgesetztem Halbkreis ausgeschnitten. In diesem Ausschnitt befindet sich der Drehtisch. Er ist auf einem Schienensystem gelagert. Mit dem Schienensystem lässt der Drehtisch sich in der Vertikalen positionieren. Mit einem Schrittmotor lässt sich der Drehtisch sich zusätzlich in der Höhe verstellen. Die Höhenverstellung wird mit einem Schneckengetriebe realisiert. Ein weiterer Schrittmotor ist für die Drehung des Tisches zuständig. Der Tisch ist über ein *Harmonic-Drive-Getriebe* mit dem Schrittmotor verbunden. Das Übersetzungsverhältnis beträgt 1:50.

### 2.2.2. Spannungsversorgung

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

### 2.2.3. Schrittmotoren

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

### 2.2.4. Schrittmotorkarten

Die Ansteuerung für die Schrittmotoren sind als 19Einschübe realisiert. Für jeden Schrittmotor wird ein Einschub benötigt. Die Einschübe sind Produkte der Firma R+S. Mittels RS-232 Schnittstelle lassen sich die Karten konfigurieren und ansteuern.

Die Konfiguration und Ansteuerung erfolgt über einen vorgegeben *ASCII* Befehlssatz. Der Befehlssatz befindet sich im Kapitel 3.3.8. Es können 2 oder mehr Karten als *Daisy-Chain*<sup>1</sup> in Reihe geschaltet werden.

### 2.2.5. Motorverkabelung

Die Schrittmotoren benötigen ein mindestens 4-adriges Kabel. Das Kabel für den Schrittmotor der für die Rotation zuständig ist war bereits gefertigt. Das Kabel für den Schrittmotor der für die Höhenverstellung zuständig ist habe ich selbst gefertigt. Hier wurden 3 weitere Adern für die beiden Endschalter benötigt. **(TODO: SCHEMAZEICHNUNG KABEL!)**

### 2.2.6. Endschalter

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

Im Aufbau waren bereits induktive Endschalter der Firma Pepperl+Fuchs verbaut. Normalerweise unterstützt die Schrittmotorkarte nur mechanische Endschalter. Durch geschickte Verdrahtung ließen sich die induktiven Endschalter verwenden. Hierzu musste über einen Spannungsteiler die Spannung herabgesetzt werden und konnte somit direkt an den Optokoppler der Schrittmotorkarte angeschlossen werden. **(TODO: SCHEMAZEICHNUNG DER VERDRAHTUNG)**

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

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

Auf der Mikrocontroller Platine sind diese Pins mit 2 Pins des Mikrocontroller verbunden. Die beiden Pins werden im Mikrocontroller als Interrupts definiert. Die

---

<sup>1</sup>Als Daisy Chain (englisch, wörtlich „Gänseblümchenkette“) bezeichnet man eine Anzahl von Hardware-Komponenten, welche in Serie miteinander verbunden sind (meist in sogenannten Bussystemen in der Automatisierungstechnik). [Wikipedia \[2012a\]](#)

Interrupt-Service-Routine ist im Kapitel (**TODO: SOFTWARE KAPITEL REFERENZIEREN**) beschrieben.

## 2.3. Mikrocontroller

Ein Mikrocontroller vereint, in einem IC, die wichtigsten Komponenten um komplexe technische Probleme leicht lösen zu können. Dazu gehören z.B. CPU, Flash-Speicher, Arbeitsspeicher, Register, Ports, ADC, DAC und mehr. Einen schematischen Überblick über die Komponenten eines Mikrocontroller bietet das Blockdiagramm in Abbildung 2.2.

In einer Programmierungsumgebung lässt sich dann für den Mikrocontroller ein Programm schreiben. Diese Programme können Signale an Pins des Mikrocontroller auswerten und Signale über andere Pins ausgeben. Eingehende Signale können binär ausgewertet werden oder mit einem ADC die Spannungshöhe bestimmt werden. Ausgehende Signale können auch binär oder mit einem DAC analog ausgegeben werden. Binäre Signale können zur Steuerung von LEDs oder Peripherie Geräten genutzt werden. Auch LC-Displays und Serielle Schnittstellen können so angesteuert werden. Für unterschiedliche Aufgaben sind verschiedene Mikrocontroller geeignet. Zu Beginn stand ein ATmega 8515 [Atmel \[2012b\]](#) im DIL-Gehäuse zur Verfügung. Dieser hatte 8 Kbyte Flash, 3 externe Interrupts, 1 Serielle Schnittstelle und konnte mit bis zu 16 MHz betrieben werden. Dieser war geeignet sich in die Programmierung mit C ein zu finden und eine Serielle Schnittstelle an zu steuern.

Für dieses Projekt sind jedoch 2 externe Schnittstellen nötig. Der ATmega 324A erfüllt diese Voraussetzung. [Atmel \[2012a\]](#) Er ist dem ATmega 8515 recht ähnlich, bietet jedoch die benötigten 2 seriellen Schnittstellen. Des weiteren hat er 32 Kbyte Flash. (**TODO: MEHR SCHREIBEN??**)

### 2.3.1. Entwicklerboard STK500

Um Mikrocontroller zu programmieren und die Programmierung zu überprüfen kann das *Entwicklerboard* STK500 der Firma ATMEL verwendet werden. Das Board enthält mehrere Mikrocontroller Steckplätze, 2 Serielle Schnittstellen, 8 Taster, 8 LEDs, 2 Erweiterungsports, ein *ISP* (**TODO: BESSERER NAME!**) und mehrere Jumper zum konfigurieren des Boards.

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

## 2. Hardware

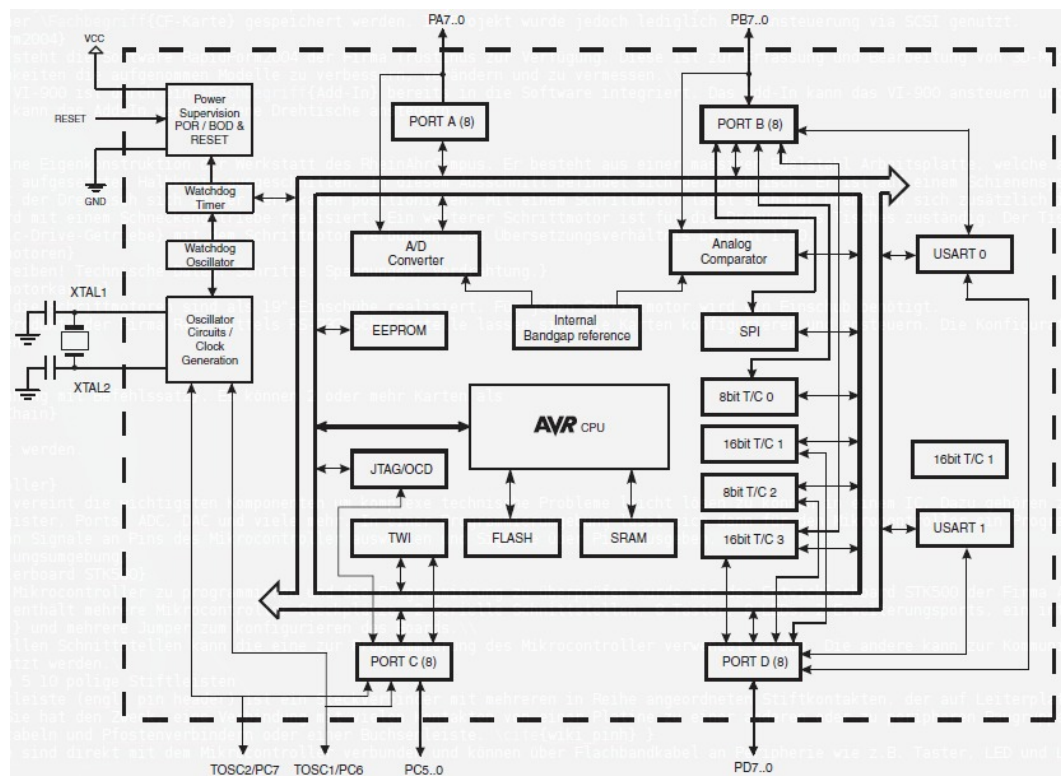


Abbildung 2.2.: Block Diagram eines Mikrocontroller  
[Atm 2011]

Auf dem Board stehen 5 10 polige Stiftleisten zur Verfügung. Diese sind direkt mit dem Mikrocontroller verbunden und können über Flachbandkabel an Peripherie wie z.B. Taster, LED und LC-Displays angeschlossen werden.

### 2.3.2. 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 über die Serielle Verbindung. 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.

### 2.3.3. 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 *Pege-lumsetzer* MAX232 genutzt. Dieser wandelt mit internen Operationsverstärkern die Spannungspegel auf den richtigen Wert. **(TODO: BESCHALTUNG?)**

## 2.4. Platinenlayout

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

## 3. Software

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

### 3.1. RapidForm2004

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

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

### 3.2. Entwicklungsumgebung

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

#### 3.2.1. AVR Studio 5

(TODO: AVR STUDIO ECLIPSE BUG DEFECTS BIBLIO?)

#### 3.2.2. Eclipse

### 3.3. Mikrocontroller

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



### 3.3.1. Fuses

Als Fuses werden Register bezeichnet mit denen sich, auf Hardwareebene, das Verhalten des Mikrocontroller verändern lässt. **(TODO: FUSES TABELLEN AUS DATENBLATT!)**

CKSEL	
SUT	•
CKDIV8	•
CKOUT	•
CKOPT	•
RSTDISBL	•
SPIEN	•
JTAGEN	•
DWEN	•
OCDEN	•
EESAVE	•
BODEN	•
BODLEVEL	•
WDTON	•
BOOTRST	•
BOOTSZ	•
Compatibility Bits	•
SELFPRGEN	•
HWBEN	•

### 3.3.2. LEDs

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

Listing 3.1: Funktion - Laufflicht

```

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

```

### 3. Software

```
6   i |= 0x08;
7   LED_PORT = i;
8 }
```

#### 3.3.3. Taster

Listing 3.2: Taster

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

#### 3.3.4. LCD Bibliothek

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

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

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

Listing 3.3: Definitionen - LCD(Auszug)

```
1 #define LCD_CONTROLLER_KS0073 0 /**< Use 0 for HD44780 controller,
2     1 for KS0073 controller */
3
4 #define LCD_LINES 4 /**< number of visible lines of the display */
5 #define LCD_DISP_LENGTH 19 /**< visibles characters per line of the display */
6 #define LCD_LINE_LENGTH 0x40 /**< internal line length of the display */
```

### 3. Software

```
7
8 #define LCD_START_LINE1 0x00 /**< DDRAM address of first char of line 1 */
9 #define LCD_START_LINE2 0x40 /**< DDRAM address of first char of line 2 */
10 #define LCD_START_LINE3 0x14 /**< DDRAM address of first char of line 3 */
11 #define LCD_START_LINE4 0x54 /**< DDRAM address of first char of line 4 */
12
13 #define LCD_WRAP_LINES 1 /**< 0: no wrap, 1: wrap at end of visibile line */
14
15 extern void lcd_init(uint8_t dispAttr);
16 extern void lcd_clrscr(void);
17 extern void lcd_home(void);
18 extern void lcd_gotoxy(uint8_t x, uint8_t y);
19 extern void lcd_putc(char c);
20 extern void lcd_puts(const char *s);
21 extern void lcd_puts_p(const char *progmem_s);
22 extern void lcd_command(uint8_t cmd);
23 extern void lcd_data(uint8_t data);
24 #define lcd_puts_P(__s) lcd_puts_p(PSTR(__s))
```

#### 3.3.5. RS-232

Listing 3.4: Funktionen - RS-232

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

```
22 else { // To Stepper
23     while (!(UCSR1A & (1 << UDRE1))) {} //warten bis Senden moeglich
24     UDR1 = c; // sende Zeichen
25 }
26 }
27 void uart_put_string (char *s, int dir) {
28     while (*s) // so lange *s != '\0' also ungleich dem "String-Endezeichen(Terminator)" {
29         uart_put_charater(*s, dir);
30         s++;
31     }
32 }
```

### 3.3.6. Menü Bibliothek

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

### 3.3.7. Interrupts

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

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

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

#### 3.3.7.1. Endschalter

Die Endschalter sind über die Schrittmotorkarten und eine Brücke in der Steuerung mit der Mikrocontroller Platine Verbunden. Dort sind sie an 2 Interrupt Pins angeschlossen. Bei einem Flanken Wechsel an den Pins wird ein Interrupt ausgelöst.

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

Listing 3.5: ISR - Endschalter

```

1 PCMSK3 |= ( 1 << PCINT28 ); // Interrupts definieren PD4 als Interrupt zulassen
2 PCICR |= ( 1 << PCIE3 ); // Pin Change Interrupt Control Register – PCIE3 setzen fuer
  PCINT30
3 ISR(PCINT3_vect){ // Endschalter Position erreicht
4   lcd_puts("Positive Endschalter Position Erreicht!");
5   LED_PORT ^= (1 << LED3);
6 }
7 ISR(PCINT2_vect){ // Endschalter Position erreicht
8   lcd_puts("Negative Endschalter Position Erreicht!");
9   LED_PORT ^= (1 << LED3);
10 }
    
```

### 3.3.7.2. Watchdog

Der *Watchdog* ist eine Sicherungseinrichtung des Mikrocontroller. In regelmäßigen Abständen wird überprüft ob das Watchdog Bit gesetzt ist und anschließend zurück gesetzt. Das Bit muss innerhalb der voreingestellten Zeit immer wieder neu gesetzt werden. Ist das Bit nicht gesetzt, wird der Mikrocontroller zurückgesetzt. Dies geschieht z.B. bei nicht geplanten Endlosschleifen.

Wahlweise kann kurz vor dem Reset noch die Watchdog-ISR durchlaufen werden.

Im Projekt wird hier die Fehler LED eingeschaltet und eine Meldung auf dem LC-Display ausgegeben. Siehe hierzu auch das Code-Listing 3.6.

Listing 3.6: Watchdog

```

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

### 3.3.8. Protokoll der Schrittmotorkarte

Tabelle 3.1 zeigt den ASCII Befehlssatz der Schrittmotorkarte.

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

Tabelle 3.1.: ASCII Befehlssatz R+S Schrittmotorsteuerung  
V9141 [2001]

### **3.3.9. Manueller Betrieb**

### **3.3.10. Protokolle aus RapidForm**

### **3.3.11. Übersetungs Logik**

#### **3.3.11.1. Zeta**

#### **3.3.11.2. Isel**

#### **3.3.11.3. Weitere**

### **3.3.12. Automatische Protokollwahl**

## 4. Fazit und Zukunft

### 4.1. Fazit

**(TODO: FAZIT SCHREIBEN!)**



## Literaturverzeichnis

### Atm 2011

ATMEL (Hrsg.): *ATmega164A/PA/324A/PA/644A/PA/1284/P Complete*. San Jose, CA 95131, USA: Atmel, 06 2011 **2.2**

### Atmel 2012a

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

### Atmel 2012b

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

### V9141 2001

RS (Hrsg.): *Schrittmotor-Platine mit integriertem Treiber*. Mörfelden-Walldorf: RS, 03 2001 **3.1**

### Wikipedia 2012a

WIKIPEDIA: *Daisy chain* — *Wikipedia, Die freie Enzyklopädie*. [http://de.wikipedia.org/w/index.php?title=Daisy\\_chain&oldid=98475104](http://de.wikipedia.org/w/index.php?title=Daisy_chain&oldid=98475104). Version: 2012. – [Online; Stand 11. Februar 2012] **1**

### 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 13. 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. Vom Autor verwendete Software

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

- **RapidForm2004**

asdf

- **AVRStudio 5**

Atmel.

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

- **Eclipse**

Eclipse mit CDT und AVRPlugin

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

- **AVRDude**

Prorammer

**(TODO: WEITERE?!)**

## A.3. Codelistings

### A.3.1. main.c

Listing A.1: main.c

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

A. Anhang

```
39 #define B_OK      "0\r\n"
40 #define D_RapidForm 0
41 #define D_Stepper 1
42 int      move = 0;
43 int      init_T = 0;
44 char     str_rx[100];
45
46 //// TinyMenu
47 // MCU_CLK = F_CPU fr TinyMenu
48 #define MCU_CLK F_CPU
49 #include "tinymenu/spin_delay.h"
50 #define CONFIG_TINYMENU_USE_CLEAR
51 #include "tinymenu/tinymenu.h"
52 #include "tinymenu/tinymenu_hw.h"
53
54 //// Funktionsdefinitionen
55 // UART Stuff
56 void     uart_init      ();
57 void     uart_put_charater (unsigned char c, int dir);
58 void     uart_put_string (char *s, int dir);
59 int      uart_get_character (int dir);
60 void     uart_get_string  (char * string_in, int dir);
61 void     uart_rx         (int dir);
62 // String Stuff
63 int      FindStringInArray (const char* pInput, const char* pOptions[], int cmp_length);
64 void     String_zerlegen_Isel(char * str_rx, char * Position, char * Winkel);
65 void     String_zerlegen_csg (char * str_rx);
66 // Hilfs Funktionen
67 void     csg_Status_melden ();
68 // Auswerte Logik
69 int      switch_Motor     (char * str_rx);
70 void     switch_Stepper   (char * str_rx);
71 void     switch_Isel      (char * str_rx);
72 void     switch_csg       (char * str_rx);
73 // LCD und LED Stuff
74 void     lcd_my_type      (char *s);
75 void     lcd_spielereien  (void);
76 void     led_spielerein   (void);
77 void     debounce_init   (void);
78 void     led_laufflicht   (void);
79 // Menu Stuff
80 void     mod_manual       (void *arg, void *name);
81 void     my_select        (void *arg, char *name);
82 void     menu_puts        (void *arg, char *name);
83 #include "mymenu.h"
```

```
84 // Init Stuff
85 void init_WDT (void);
86 void init (void);
87
88 ///////////////////////////////////////////////////
89 //
90 // Hauptschleife
91 //
92 ///////////////////////////////////////////////////
93 int main(void) {
94     init ();
95     while (1) {
96         wdt_reset();
97         if (get_key_press(1 << KEY0) || get_key_rpt(1 << KEY0)){
98             lcd_puts("Betrete Menue!\n");
99             menu_enter(&menu_context, &menu_main);
100         }
101         if (get_key_press(1 << KEY1))
102             menu_exit(&menu_context); // 1 - Back
103         if (get_key_press(1 << KEY2))
104             menu_prev_entry(&menu_context);
105         if (get_key_press(1 << KEY3) || get_key_rpt(1 << KEY3))
106             menu_next_entry(&menu_context);
107         if (get_key_press(1 << KEY4) || get_key_rpt(1 << KEY4))
108             menu_select(&menu_context); // 4 - Select
109         if ((UCSR0A & (1 << RXC0))){
110             LED_PORT &= (1 << LED2);
111             uart_rx(D_RapidForm);
112         }
113         if ((UCSR1A & (1 << RXC1))){
114             LED_PORT &= (1 << LED3);
115             uart_rx(D_Stepper);
116         }
117     }
118 }
119 ///////////////////////////////////////////////////
120 //
121 // Hauptschleife Ende
122 //
123 ///////////////////////////////////////////////////
124
125 // Interrupt Stuff
126 ISR(WDT_vect){ // Watchdog ISR
127     LED_PORT &= ~(1 << LED4); // LED5 einschalten
128     lcd_puts("Something went \nterribly wrong!\nRebooting!");
```



```

129 }
130 ISR(PCINT3_vect){ // Endschalter Position erreicht
131     lcd_puts("Positive Endschalter Position Erreicht!");
132     //uart_put_string("1H\n", D_Stepper);
133     LED_PORT ^= (1 << LED3);
134 }
135 ISR(PCINT2_vect){ // Endschalter Position erreicht
136     lcd_puts("Negative Endschalter Position Erreicht!");
137     //uart_put_string("1H\n", D_Stepper);
138     LED_PORT ^= (1 << LED3);
139 }
140 // UART Stuff
141 void uart_init () {
142     // UART 0 – IN (Rapidform Software/Terminal)
143     UBR0H = UBR0H_VALUE;
144     UBR0L = UBR0L_VALUE;
145     UCS0C = (3 << UCS0C);
146     UCS0B |= (1 << TXEN0); //Transmitter Enabled
147     UCS0B |= (1 << RXEN0); // UART RX einschalten
148
149     // UART 1 – OUT (Stepper Karte/Drehtisch)
150     UBR1H = UBR1H_VALUE;
151     UBR1L = UBR1L_VALUE;
152     UCS1C = (3 << UCS1C);
153     UCS1B |= (1 << TXEN1); //Transmitter Enabled
154     UCS1B |= (1 << RXEN1); // UART RX einschalten
155
156 }
157 void uart_put_charater (unsigned char c, int dir) {
158     // To Rapidform
159     if (dir == D_RapidForm) {
160         while (!(UCS0A & (1 << UDRE0))) //warten bis Senden moeglich
161         {
162         }
163         UDR0 = c; // sende Zeichen
164     }
165     // To Stepper
166     else {
167         while (!(UCS1A & (1 << UDRE1))) //warten bis Senden moeglich
168         {
169         }
170         UDR1 = c; // sende Zeichen
171     }
172     //return 0;
173 }

```

```

174 void    uart_put_string    (char *s, int dir) {
175     while (*s) // so lange *s != '\0' also ungleich dem "String-Endezeichen(Terminator)"
176     {
177         uart_put_charater(*s, dir);
178         s++;
179     }
180 }
181 int     uart_get_character (int dir) {
182     if (dir == D_RapidForm) {
183         while (!(UCSR0A & (1 << RXC0)))
184             // warten bis Zeichen verfuegbar
185             ;
186         return UDR0; // Zeichen aus UDR an Aufrufer zurueckgeben
187     }
188     if (dir == D_Stepper) {
189         while (!(UCSR1A & (1 << RXC1)))
190             // warten bis Zeichen verfuegbar
191             ;
192         return UDR1; // Zeichen aus UDR an Aufrufer zurueckgeben
193     }
194     return -1;
195 }
196 void    uart_get_string    (char * string_in, int dir) {
197     char c;
198     int i = 0;
199     do {
200         c = uart_get_character(dir);
201         if (c != '\r') {
202             *string_in = c;
203             string_in += 1;
204             i++;
205         }
206     } while (i < 100 && c != '\r' && c != '\n');
207     *string_in = '\0';
208     if (dir == D_Stepper)
209         LED_PORT |= ( 1 << LED3 );
210     else
211         LED_PORT |= ( 1 << LED2 );
212 }
213
214 // String Stuff
215 #define M_UNK    -2
216 #define M_NOTI   -1
217 #define M_ISEL    0
218 #define M_CSG     1
    
```

A. Anhang

```
219 #define M_ZETA 2
220 #define M_TERMINAL 3
221
222 #define P_INIT 0
223 #define P_FINISH 1
224 #define P_AROT 2
225 #define P_STOP 3
226 #define P_HOME 4
227 #define P_STEP 5
228 #define P_TIMEOUT 6
229
230 #define E_CLS 10
231 #define E_TEST 11
232
233 #define B_Zeta_Return "\r\n>\040\r\n>\040\r\n>\040\r\n>\040\r\n>\040"
234
235 #define MENU_ENTRY_NAMELEN 19
236 #define RETURN_LEN 40
237
238 // Struct Versuche
239 /*
240 typedef struct Entry_s {
241     char Name[19];                // Name zum Anzeigen
242     char Input[40];              // Vergleichswert
243     char Output[40];            // Ausgabebefehl
244 } PROGMEM Entry_t;              // Ergeben Struct P_Entry_t
245
246 typedef struct Motor_s {
247     uint8_t    num_Befehle;
248     Entry_t    *Befehl;          // 4 Motoren vom Typ Befehle_t
249 } Motor_t;                      // Ergeben Struct Motor_t
250
251 typedef struct Protokoll {
252     uint8_t    num_Motor;
253     Motor_t    Motor[4];
254 } Protokoll_t;
255
256 Entry_t PROGMEM Befehl[] = { // <===
257     { // Befehl[0] Init
258         .Name = "Init\n",
259         .Input = "@01",
260         .Output = "0\r\n",
261     },
262     { // Befehl[1] Home
263         .Name = "Home\n",
```

A. Anhang

```
264     .Input = "@01",
265     .Output = "0\r\n",
266 },
267 };
268
269 Protokoll_t Protokoll = {
270     .num_Motor = 4,
271     .Motor[M_ISEL] = { // Motor[0] Isel
272         .num_Befehle = 7,
273         .Befehl = progmem_Befehl,
274     },
275     .Motor[M_ZETA] = {
276         .num_Befehle = 7,
277         .Befehl = progmem_Befehl,
278     }
279 };
280 */
281
282 int FindStringInArray (const char* pInput, const char* pOptions[], int cmp_length) {
283     int n = -1;
284     while (pOptions[++n]) {
285         //lcd_puts(pOptions[n]);
286         //lcd_puts("\n");
287         if (!strcmp(pInput, pOptions[n], cmp_length)){
288             return n;
289         }
290     }
291     return 99;
292 }
293 void String_zerlegen_Isel(char * str_rx, char * Position, char * Winkel) {
294     //0M5200, +600
295     //Achse M Position, +Geschwindigkeit
296     char * Achse="0";
297     Achse[0] = str_rx[1];
298     Achse[1] = '\0';
299     if (atoi(Achse)==0){
300         lcd_puts("Achse: ");
301         lcd_puts(Achse);
302         lcd_puts(" (Rotation)\n");
303     }
304     if (atoi(Achse)==1){
305         lcd_puts("Achse: ");
306         lcd_puts(Achse);
307         lcd_puts(" (Hoehe) \n");
308     }
}
```

A. Anhang

```
309 lcd_puts("Test: ");
310 lcd_puts(Position);
311 char c;
312 int i = 0;
313 do {
314     c = str_rx[i + 3];
315     if (c != ',') {
316         Position[i] = c;
317         i++;
318     }
319 } while (i < 20 && c != '\0' && c != ',');
320 Position[i] = '\0';
321 int32_t z;
322 int32_t y;
323 z = atol(Position);
324 y = z / 7200;
325 z = (z * 71111) / 1024;
326 ltoa(y, Winkel, 10);
327 ltoa(z, Position, 10);
328 }
329 void String_zerlegen_csg(char * str_rx) {
330     //012 3456 78901 2345 6789 01234 5678
331     //D:2 S500 F5000 R200 S500 F5000 R200.
332     //D:2S500F5000R200S500F5000R200
333
334     // Format:
335     // D:[Speed range]S[Minimum speed]F[Maximum Speed]R[Acceleration/Deceleration time]
336     // |-----Axis1 parameters
337     // |-----|
338     // S[Minimum speed for Axis 2] S[Minimum speed for Axis 2]F[Maximum speed]R[
339     // Acceleration/Deceleration time]
340     // |-----Axis 2 parameters
341     // |-----|
342
343     int i = 4; // Index Input String | Bei 4. Zeichen Beginnen. Die ersten 3 Zeichen sind Fix.
344     int j = 0; // Index Variable
345     char c; // Zu kopierendes Zeichen
346     // Variablen Deklaration und Initialisierung mit Defaultwerten
347     char Speed_Range[2] = "2";
348     char ONE_Min_Speed[6] = "200";
349     char ONE_Max_Speed[6] = "2000";
350     char ONE_Acc_Speed[5] = "200";
351
352     ///////////////////////////////////
```

A. Anhang

```
351 //
352 // Speed Range (1 || 2)
353 //
354 ///////////////////////////////////////////////////
355 Speed_Range[0] = str_rx[2];
356 Speed_Range[1] = '\0';
357
358 ///////////////////////////////////////////////////
359 //
360 // Min Speed (50 – 20000)
361 //
362 ///////////////////////////////////////////////////
363 do {
364     c = str_rx[i];
365     if (c != 'F') {
366         ONE_Min_Speed[j] = c;
367         j++;
368         i++;
369     }
370 } while (j < 6 && c != '\0' && c != 'F');
371 ONE_Min_Speed[j] = '\0';
372
373 lcd_puts("1_MIN_SPEED:");
374 lcd_puts(ONE_Min_Speed);
375 lcd_puts("\n");
376 // TODO: Range      berprfen      ! 50–20000
377 //uart_puts();
378
379 ///////////////////////////////////////////////////
380 //
381 // Max Speed (50 – 20000)
382 //
383 ///////////////////////////////////////////////////
384 i++; // Steuerzeichen      berspringen
385 j = 0; // Variablenzhler      zurcksetzen
386 do {
387     c = str_rx[i];
388     if (c != 'R') {
389         ONE_Max_Speed[j] = c;
390         i++;
391         j++;
392     }
393 } while (j < 6 && c != '\0' && c != 'R');
394 ONE_Max_Speed[j] = '\0';
395
```

A. Anhang

```
396     lcd_puts("1_MAX_SPEED:");
397     lcd_puts(ONE_Max_Speed);
398     lcd_puts("\n");
399
400     ///////////////////////////////////
401     //
402     //  Acceleration (0 - 1000)
403     //
404     ///////////////////////////////////
405     i++; // Stuerzeichen   berspringen
406     j = 0; //  Variablenzähler   zurcksetzen
407     do {
408         c = str_rx[i];
409         if (c != 'S') {
410             ONE_Acc_Speed[j] = c;
411             i++;
412             j++;
413         }
414     } while (j < 4 && c != '\0' && c != 'S');
415     ONE_Acc_Speed[j] = '\0';
416
417     lcd_puts("1_ACC_SPEED:");
418     lcd_puts(ONE_Acc_Speed);
419     lcd_puts("\n");
420
421     //uart_put_string("0\n", D_Stepper);
422     uart_put_string(B_OK, D_RapidForm);
423 }
424 //  Hilfs Funktionen
425 void csg_Status_melden (void) {
426     uart_put_string("      0,      0,K,K,R\r\n", D_RapidForm); // Status an
427                                     RapidForm zurckmelden
428 }
429 void Position_Zeta      (char * Position) {
430     char c;
431     int i = 0;
432     do{
433         c = str_rx[i + 1];
434         if(c != ','){
435             Position[i] = c;
436             i++;
437         }
438     }
439     while(i < 20 && c != '\0' && c != ',');
440     Position[i] = '\0';
```

A. Anhang

```
440     int32_t z;
441     z = atol(Position);
442     z = z/9;
443     ltoa(z,Position,10);
444 }
445 //      Vearbeitungs Logik
446 int      Initialized = M_NOTI;
447 void     switch_Stepper      (char * str_rx) {
448     const char* pOptions[] = {
449         "#",    // 0 – Stepper Karte Befehl erkannt
450         "E",    // 1 – Error
451         "!CLS", // 2 – Clear Screen
452         "Test", // 3 – Test
453         0 };
454     switch (FindStringInArray(str_rx, pOptions, 1)) {
455     case 0:
456         lcd_puts("Erfolgreich\n");
457         //uart_put_string("0\n\r", D_RapidForm);
458         break;
459     case 1:
460         lcd_puts("Error\n");
461         uart_put_string("1\n\r", D_RapidForm);
462         break;
463     case 2:
464         lcd_clrscr();
465         break;
466     case 3:
467         lcd_puts("Test bestanden\n");
468         //uart_put_string("Test bestanden\n\r", D_RapidForm);
469         //uart_put_string("Test bestanden\n\r", D_Stepper);
470         break;
471     default :
472         ms_spin(10);
473         //lcd_puts("Stepper: ");
474         //lcd_puts(str_rx);
475         //lcd_puts("!\n");
476     }
477 }
478 void     switch_Isel      (char * str_rx) {
479     const char* pOptions[] = {
480         "XXXXXXX", // 0 – Reserve
481         "!CLS",    // 1 – LC–Display  lschen
482         "Test",    // 2 – Test
483         "@01",     // 3 – Achse   auswahlen
484         "@0R",     // 4 – Status abfrage
```



A. Anhang

```
485         "@0M",      // 5 - Gehe zu Position MX , +600
486         0 };
487
488     int Ret_Val = FindStringInArray(str_rx, pOptions, 3);
489     switch (Ret_Val) {
490     case 0:          // 0 - Reserve
491         lcd_puts("Reserve\r\n");
492         break;
493     case 1:          // 1 - LC-Display  lschen
494         lcd_clrscr();
495         break;
496     case 2:          // 2 - Test
497         lcd_puts("Test bestanden\r\n");
498         uart_put_string("Test bestanden\r\n", D_RapidForm);
499         //lcd_puts(Protokoll.Motoren.M_Motor[M_ISEL].P_Init);
500         break;
501     case 3:          // 3 - Achse  auswhlen
502         ms_spin(10);
503         /*
504         char buf[32];
505         PGM_P p;
506         int i;
507
508         memcpy_P(&p, &Protokoll.Motor[M_ISEL].Befehl[0].Name[0], sizeof(PGM_P));
509         strcpy_P(buf, p);
510         */
511         /*
512         char string_in[40];
513         char c;
514
515         char * str_in_p = &string_in;
516
517         do{
518             c = pgm_read_byte( s_ptr );
519             *str_in_p = c;
520             str_in_p += 1;
521             s_ptr++; // Increase string pointer
522         } while( pgm_read_byte( s_ptr ) != 0x00 ); // End of string
523         */
524
525         //lcd_puts( buf );
526         lcd_puts("Init");
527         //String_zerlegen_Isel(str_rx, Position);
528         uart_put_string("0\r\n", D_RapidForm);
529         //uart_put_string(Protokoll.Motor[M_ISEL].Befehl[0].Output, D_RapidForm);
```

```
530     break;
531     case 4:          // 4 – Status abfrage
532         lcd_puts("Statusabfrage:  \n");
533         uart_put_string("A\n", D_Stepper);
534         ms_spin(50);
535         if ((UCSR1A & (1 << RXC1)))
536             uart_rx(D_Stepper);
537         if (!strcmp(str_rx, "0#"))
538             uart_put_string("0\r\n", D_RapidForm);
539         else {
540             lcd_puts("Fehlgeschlagen  \n");
541             uart_put_string("1\r\n", D_RapidForm);
542         }
543         break;
544     case 5:          // 5 – Gehe zu Position MX , +600
545         ms_spin(10);
546         char Position [33], Winkel[6];
547         memset(Position, '\0', 33);
548         memset(Winkel, '\0', 6);
549         String_zerlegen_Isel(str_rx, Position, Winkel);
550         char Move_To[40];
551         memset(Move_To, '\0', 40);
552         Move_To[0] = 'M';
553         Move_To[1] = 'A';
554         Move_To[2] = ' ';
555         Move_To[3] = '\0';
556         strcat(Move_To, Position);
557         strcat(Move_To, "\n");
558         lcd_puts("Pos:");
559         lcd_puts(Move_To);
560
561         uart_put_string(Move_To, D_Stepper);
562         ms_spin(50);
563         if ((UCSR1A & (1 << RXC1)))
564             uart_rx(D_Stepper);
565         else {
566             //lcd_puts("Befehl n. bestaetig\n");
567             break;
568         }
569
570         uart_put_string("A\n", D_Stepper);
571         ms_spin(50);
572         if ((UCSR1A & (1 << RXC1)))
573             uart_rx(D_Stepper);
574         else {
```

```
575     lcd_puts("Keine Bewegung!\n");
576 }
577
578 while (!strcmp(str_rx,"1#")){
579     uart_put_string("A\n", D_Stepper);
580     ms_spin(50);
581     if ((UCSR1A & (1 << RXC1))){
582         uart_rx(D_Stepper);
583         lcd_clrscr();
584         lcd_puts("Gehe zu Winkel: ");
585         lcd_puts(Winkel);
586         lcd_puts("\n");
587     }
588     else {
589         lcd_puts("Keine Antwort\n");
590     }
591     wdt_reset();
592 }
593 lcd_puts("Winkel: ");
594 lcd_puts(Winkel);
595 lcd_puts(" Erreicht\n");
596 uart_put_string("0\r\n", D_RapidForm);
597 break;
598 default:
599     //lcd_puts("ISEL:  \n");
600     lcd_puts(str_rx);
601 }
602 }
603 void switch_csg (char * str_rx) {
604     const char* pOptions[] = {
605         "Test2", // 0 - Stepper Karte Befehl erkannt
606         "!CLS", // 1 - LC-Display lschen
607         "Test", // 2 - Test
608         "Q:", // 3 - Status abfrage
609         "D:2", // 4 - D:2S500F5000R200S500F5000R200.
610         "H:", // 5 - H:
611         "G", // 6 - Motor starten
612         "M:", // 7 - Move by Pulses
613         "!", // 8 - Busy Ready ?
614         "H1",
615         0 };
616     switch (FindStringInArray(str_rx, pOptions, 2)) {
617     case 0: // Motorkarte Erfolgreich angesprochen
618         lcd_puts("!");
619         break;
```

A. Anhang

```
620 case 1: // Display löschen
621     lcd_clrscr();
622     break;
623 case 2: // Interner Test
624     lcd_puts("!T");
625     //uart_puts("Test bestanden\n\r");
626     break;
627 case 3: // Status abfrage von Software
628     lcd_puts("Statusabfrage \n");
629     csg_Status_melden();
630     break;
631 case 4:
632     String_zerlegen_csg(str_rx);
633
634     break;
635 case 5:
636     lcd_puts("H: \n");
637     uart_put_string(B_OK, D_RapidForm);
638     break;
639 case 6:
640     lcd_puts("Motor starten\n");
641     //uart_put_string(B_OK, D_RapidForm);
642     break;
643 case 7:
644     move++;
645     char it [10];
646     itoa(move, it, 10);
647     lcd_puts(it);
648     lcd_puts("_Move!\n");
649     uart_put_string("M 160000\r\n", D_Stepper);
650
651     break;
652 case 8:
653     lcd_puts("R/B?");
654     uart_put_string("R\r\n", D_RapidForm);
655     break;
656 case 9:
657     lcd_puts("H1 empfangen \n");
658     break;
659 default:
660     lcd_puts("U_B: ");
661     lcd_puts(str_rx);
662     lcd_puts("!END \n");
663 }
664 }
```

```
665 void switch_Zeta (char * str_rx) {
666     const char* pOptions[] = {
667         "!CLS", // 0 - LC-Display lschen
668         "Test", // 1 - Test
669         "GO", // 2 - Motor Starten
670         "WAIT", // 3 - Wait till motor stops
671         "!XXXX", // 4 - Reserve
672         "COMEX", // 5 - *COMEXC0
673         "MA1", // 6 - Absolute Positioning
674         "D1125", // 7 - Position
675         "A8", // 8 - Accelartion 8
676         "V8", // 9 - Velocity 8
677         "ECHO0", // 10 - Echo abschalten
678         "PSET0", // 11 - Ursprung setzen
679         0 };
680     char Position [33];
681     char Move_To[40];
682     memset(Move_To, '\0', 40);
683     Move_To[0] = 'M';
684     Move_To[1] = 'A';
685     Move_To[2] = ' ';
686     Move_To[3] = '\0';
687     switch (FindStringInArray(str_rx, pOptions, 1)) {
688     case 0: // Display lschen
689         lcd_clrscr();
690         break;
691     case 1: // Interner Test
692         lcd_puts("Test bestanden \n");
693         break;
694     case 2: // Go
695         ms_spin(100);
696         strcat (Move_To, Position);
697         strcat (Move_To, "\n");
698         //lcd_puts("Pos:");
699         //lcd_puts(Move_To);
700
701         uart_put_string(Move_To, D_Stepper);
702         ms_spin(50);
703         if ((UCSR1A & (1 << RXC1)))
704             uart_rx(D_Stepper);
705         else {
706             lcd_puts("Befehl n. bestaetig\n");
707             break;
708         }
709 }
```

A. Anhang

```
710     uart_put_string("A\n", D_Stepper);
711     ms_spin(50);
712     if ((UCSR1A & (1 << RXC1)))
713         uart_rx(D_Stepper);
714     else {
715         lcd_puts("Keine Bewegung!\n");
716     }
717
718     while (!strcmp(str_rx, "1#")){
719         uart_put_string("W\n", D_Stepper);
720         ms_spin(100);
721         if ((UCSR1A & (1 << RXC1))){
722             uart_rx(D_Stepper);
723             lcd_clrscr();
724             lcd_puts("Position(Akt/Ges): \n");
725             lcd_puts(str_rx);
726             lcd_puts(" / ");
727         }
728         else {
729             lcd_puts("Keine Antwort\n");
730         }
731         wdt_reset();
732
733         uart_put_string("A\n", D_Stepper);
734         ms_spin(50);
735         if ((UCSR1A & (1 << RXC1))){
736             uart_rx(D_Stepper);
737             //lcd_clrscr();
738             //lcd_puts("running to\n");
739             //lcd_puts("Position: ");
740             lcd_puts(Position);
741             lcd_puts("\n");
742         }
743         else {
744             lcd_puts("Keine Antwort\n");
745         }
746         wdt_reset();
747     }
748     lcd_puts("Position: \n");
749     lcd_puts(Position);
750     lcd_puts(" Erreicht\n");
751     uart_put_string(B_Zeta_Return, D_RapidForm);
752     break;
753 case 3: // WAIT
754     break;
```

A. Anhang

```
755 case 4: // Reserve
756     break;
757 case 5: // COMEXC0
758     break;
759 case 6:
760     //lcd_puts("MA1 empfangen \n");
761     break;
762 case 7: // Position Setzen
763     memset(Position, '\0', 33); // Array mit Nullen befüllen
764     Position_Zeta(Position);
765     break;
766 case 8:
767     break;
768 case 9: //V8
769     lcd_puts("Speed set \n");
770     //uart_put_string(B_Zeta_Return, D_RapidForm);
771     break;
772 case 10:
773     lcd_puts("Echo off \n");
774     //uart_put_string(str_rx, D_RapidForm);
775     //uart_put_string("ECHO0\r", D_RapidForm);
776     break;
777 case 11:
778     break;
779 default:
780     lcd_puts("Z:");
781     lcd_puts(str_rx);
782     lcd_puts(" \n");
783     // Initialized = switch_Inputs(str_rx);
784 }
785 }
786 void switch_Terminal (char * str_rx) {
787     const char* pOptions[] = {
788         "!CLS", // 0 - LC-Display löschen
789         "Test", // 1 - Test
790         "!Manual", // 2 - Ignorieren
791         "!YYYY", // 3 - Wait till motor stops
792         0 };
793
794     if (init_T == 0){
795         init_T = 1;
796         uart_put_string("Willkommen im Terminal Modus\r\n",D_RapidForm);
797         uart_put_string(" mögliche Befehle sind: \r\n",D_RapidForm);
798         uart_put_string(" A - Motorstatus\r\n M - Move Steps\r\n", D_RapidForm);
799     }
```

A. Anhang

```
800     switch (FindStringInArray(str_rx, pOptions, 2)) {
801     case 0: // Display Ischen
802         lcd_clrscr();
803         break;
804     case 1: // Interner Test
805         lcd_puts("Test bestanden \n");
806         uart_put_string("Test bestanden", D_RapidForm);
807         break;
808     case 2: // Reserve 1
809
810     case 3: // Reserve 2
811
812         break;
813     default:
814         //lcd_puts("Z:");
815         lcd_puts(str_rx);
816         lcd_puts(" \n");
817         uart_put_string(str_rx,D_Stepper);
818         uart_put_string("\n",D_Stepper);
819     }
820 }
821 int switch_Motor (char * str_rx) {
822     const char* pOptions[] = {
823         "@01", // 0 - Isel
824         "Q:", // 1 - CSG
825         "ECHO0", // 2 - Zeta
826         "!Terminal", // 3 - Terminal ansteuerung!
827         0 };
828     switch (FindStringInArray(str_rx, pOptions, 3)) {
829     case 0: // 0 - ISEL
830         return M_ISEL;
831         break;
832     case 1: // 1 - CSG
833         return M_CSG;
834         break;
835     case 2: // 2 - Zeta
836         return M_ZETA;
837         break;
838     case 3: // 3 - Terminal ansteuerung
839         return M_TERMINAL;
840         break;
841     default:
842         return M_UNK;
843     }
844 }
```



A. Anhang

```
845 void    uart_rx          (int dir) {
846     uart_get_string(str_rx, dir);
847     if (dir == D_Stepper)
848         switch_Stepper(str_rx);
849     else{
850         if( Initialized == M_UNK){
851             lcd_puts("Unbekannter Motor!\n");
852             //lcd_puts(str_rx);
853             Initialized = M_NOTI;
854         }
855         if( Initialized == M_NOTI){
856             Initialized = switch_Motor(str_rx);
857         }
858         if( Initialized == M_ISEL)
859             switch_Isel(str_rx);
860         if( Initialized == M_CSG)
861             switch_csg(str_rx);
862         if( Initialized == M_ZETA)
863             switch_Zeta(str_rx);
864         if( Initialized == M_TERMINAL)
865             switch_Terminal(str_rx);
866     }
867 }
868 //      LCD und LED Stuff
869 void    lcd_my_type       (char *s) {
870     srand(TCNT0);
871     int min = 10;
872     int max = 250;
873     int erg = 0;
874     while (*s) // so lange *s != '\0' also ungleich dem "String-Endezeichen(Terminator)"
875     {
876         erg = (rand() % (max - min + 1) + min);
877         lcd_putc(*s);
878         s++;
879         for (int i = 0; i < erg; i++)
880             _delay_ms(1);
881     }
882 }
883 void    lcd_spielereien   (void) {
884     _delay_ms(100);
885     lcd_my_type("Hello Joe!\n");
886     _delay_ms(600);
887     lcd_clrscr();
888     lcd_my_type("Ready!\n");
889 }
```

A. Anhang

```
890 void led_spielerein (void) { // LEDs durchlaufen
891     for (int i = 1; i < 9; i++) {
892         _delay_ms(80); // warte 80ms
893         LED_PORT &= ~(1 << i); // loescht Bit an PortB – LED an
894         LED_PORT |= ((1 << (i - 1))); // setzt Bit an PortB – LED aus
895         //wdt_reset();
896     }
897 }
898 void debounce_init (void) {
899     KEY_DDR &= ~ALL_KEYS; // configure key port for input
900     KEY_PORT |= ALL_KEYS; // and turn on pull up resistors
901     TCCR0B = (1 << CS02) | (1 << CS00); // divide by 1024
902     TCNT0 = (uint8_t) (int16_t) - (F_CPU / 1024 * 10 * 10e-3 + 0.5); // preload for 10ms
903     TIMSK0 |= 1 << TOIE0; // enable timer interrupt
904     sei();
905 }
906 /* Wie funktioniert das?
907 * i 11110111 11111110 11111111
908 * is 11101111 11111101 11111110
909 * F0 11110000 FE 11111110 11111110
910 * r 11100000 11111100 11111110
911 *
912 * i 11110111 11111110 11111111
913 * 07 00000111 00 00000000 00000000
914 * l 00000111 00000000 00000000
915 *
916 * l|r 11100111 11111100 11111110
917 *
918 * if< 11110000 FE 11111110 11111110
919 *
920 * 08 00001000 00000001
921 * i| 11101111 11111101
922 */
923 void led_laufflicht (void) {
924     uint8_t i = LED_PORT;
925     i = (i & 0x00) | ((i << 1) & 0xFE);
926     if (i < 0xFE) i |= 0x01;
927     LED_PORT = i;
928 }
929 // Menu Stuff
930 void mod_manual (void *arg, void *name) {
931     lcd_puts("Manueller Modus\n");
932     lcd_puts("Aufnahme starten!\n");
933     lcd_puts("Danach Select\n");
934     lcd_puts("> Drehung um 45\n");
```

## A. Anhang

```

935     if (get_key_press(1 << KEY4))
936         uart_put_string("M 55750\r", D_Stepper);
937 }
938 void my_select      (void *arg, char *name) {
939     lcd_clrscr();
940     lcd_puts("Selected: ");
941     lcd_puts(name);
942
943     ms_spin(750);
944 }
945 void menu_puts      (void *arg, char *name) {
946     //my_select(arg, name);
947     uart_put_string(arg, D_Stepper);
948     lcd_clrscr();
949     lcd_puts("Send: ");
950     lcd_puts(arg);
951     lcd_puts("\n");
952     ms_spin(100);
953     //if ((UCSR1A & (1 << RXC1)))
954     uart_rx(D_Stepper);
955     ms_spin(1000);
956 }
957 // Init Stuff
958 void init_WDT(void) {
959     cli();
960     wdt_reset();
961     WDTCR |= (1 << WDCE) | (1 << WDE);
962     WDTCR = (1 << WDE) | (1 << WDIE) | (1 << WDP3) | (1 << WDP0); //Watchdog 8s
963     //WDTCR = 0x0F; //Watchdog Off
964     sei();
965 }
966 void init() {
967     init_WDT(); // Watchdog Initialisieren oder Abschalten
968     LED_DDR = 0xFF; // LED Port Richtung definieren (Ausgang)
969     LED_PORT = 0xFF; // LEDs ausschalten
970     PCMSK3 |= (1 << PCINT28); // Interrupts definieren PD4 als Interrupt zulassen
971     PCICR |= (1 << PCIE3); // Pin Change Interrupt Control Register – PCIE3 setzen
972     //fuer PCINT30
973     DDRC |= (1 << PB7); // Pin7 (Kontrast) als Ausgang definieren (Nur LCD an
974     //STK500)
975     LCD_PORT &= (1 << PB7); // Pin7 auf 0V legen (Nur LCD an
976     //STK500)
977     lcd_init(LCD_DISP_ON_CURSOR); // LC Display initialisieren
978     lcd_spiereien(); // Kurze Startup Meldung zeigen
979     led_spiereien(); // Starten des Mikrocontroller kennzeichnen
    
```

*A. Anhang*

---

```
977     debounce_init();           // Taster entprellen
978     uart_init();               // RS-232 Verbindung initialisieren
979     //menu_enter(&menu_context, &menu_main); // Kommentar entfernen um Menue zu
    aktivieren
980 }
```