F

DIPLOMARBEIT

HPGL-Plotter

Verfasser: Florian Dopona

Betreuer: Fachlehrer Ing. Stefan Grömer

Abgabedatum: 23.Mai 2014

Bahnhofstraße 52, 4050 Traun

Tel: 07229 623 11, Fax: 07229 623 11-41

E-Mail: [htl.traun@eduhi.at](mailto:htl.traun@eduhi.at)

Web: <http://www.htltraun.at>

Schulkennzahl: 410457

# Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Traun, Tag. Monat. Jahr

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Unterschrift(en)

# Zusammenfassung

Die vorliegende Diplomarbeit befasst sich mit der elektronischen Steuerung für eine Fräse, die von Herrn Professor Dipl. Ing. Anton Mayr entwickelt und gebaut wurde. Hierfür wurde zuerst das Modell besichtigt, um die Elektronik individuell darauf auslegen zu können. Im Anschluss wurden einige Testprogramme erstellt, um die Funktionalitäten (UART für RS-232 Kommunikation, interruptgesteuerte Schrittmotorenansteuerung (TCA3727) und LCD Ansteuerung (HD44780)) auf einem Steckbrett austesten und anschließend ohne weitere Probleme in das Hauptprogramm integrieren zu können. Dafür wurden einerseits Testplatinen (für TCA3727) vom Betreuungslehrer Herrn Ing. Stefan Grömer und andererseits ein spezieller Seriell-zu-USB-Wandler mit TTL-Pegeln (für UART) verwendet. Nach Implementierung der Funktionen und anschließender Integration dieser in das Hauptprogramm, wurde ein PCB benötigt, um die Entwicklung des Hauptprogramms fortsetzen zu können. Zu diesem Zweck wurde Target 3001! in der Version 16 verwendet, welches jedoch in der freien Variante eine Begrenzung von 250 Pins aufweist. Diese Größe wurde im Rahmen des Projektes überschritten, wodurch es erforderlich war, eine weitere Aufsteckplatine zu designen.

Der Kern der Steuerung besteht aus einem ATMEGA8-16PU. Dieser erhält seine Befehle kodiert in einem Subset der Sprache HPGL über RS-232. Die Hauptaufgabe des ATMEGA8-16PU ist es somit, diese Befehle zu dekodieren und über die Schrittmotoren auf das Modell auszugeben.

Zur Bewegung der Achsen des Modells wurden Schrittmotoren verwendet, da diese präzise eine Position anfahren können, ohne einen Positionssensor oder Drehgeber verwenden zu müssen, um deren Position zu bestimmen.

# Abstract

The present thesis deals with the electronic controller for a milling machine designed and built by Professor Dipl. Ing. Anton Mayr. For this purpose, the model was first inspected to be able to design the electronics individually. Following some test programs were created to test the functionality (UART for RS -232 communication, interrupt-driven stepper motor control (TCA3727 ) and LCD driver (HD44780)) on a breadboard,. This made it possible to reduce errors and simplify the development by integrating those finished functions into the main program. This program parts were initially developed using a breadboard and test boards from project supervisor Ing. Stefan Grömer and further a special serial-to-USB converter with TTL levels (for UART). After implementation of the functions and subsequent integration of those into the main program, a PCB was needed in order to continue the development of the main program. For this purpose TARGET 3001! in version 16 was used, which has in its free version a limit of only 250 pins. This size has been exceeded during the project, which made it necessary to design an additional daughterboard.  
The heart of the control system consists of a ATMEGA8-16PU, which obtains its commands encoded in a subset of the language HPGL via RS-232. Its main task is therefore to decode these commands and output them via the stepper motors to the model.  
For the movement of the axes of the model, stepper motors were used, because they can accurately move to a position without needing a position sensor or encoder.

# Inhaltsverzeichnis

[Eidesstattliche Erklärung I](#_Toc388502910)

[Zusammenfassung II](#_Toc388502911)

[Abstract III](#_Toc388502912)

[Inhaltsverzeichnis IV](#_Toc388502913)

[1 Einleitung 1](#_Toc388502914)

[1.1 Motivation 1](#_Toc388502915)

[1.2 Zielsetzung und Aufgaben 1](#_Toc388502916)

[1.3 Projektumfeld 2](#_Toc388502917)

[1.4 Lösungskonzept 3](#_Toc388502918)

[2 Hauptteil - Grundlagen und Methoden 5](#_Toc388502919)

[2.1 Erstellen des Blockschaltbilds 5](#_Toc388502920)

[2.1.1 Wahl der Handsteuerung 5](#_Toc388502921)

[2.1.1.1 Selbst gebaute Steuerung 6](#_Toc388502922)

[2.1.1.2 Game-Port Joystick 6](#_Toc388502923)

[2.1.1.3 Entscheidung für Nintendo Wii™-Nunchuck 7](#_Toc388502924)

[2.1.1.4 Entscheidung für Nintendo Wii™ Nunchuck 7](#_Toc388502925)

[2.1.2 Wahl des Displays 8](#_Toc388502926)

[2.1.2.1 HD44780 kompatibles Display 8](#_Toc388502927)

[2.1.2.2 Andere Displays 8](#_Toc388502928)

[2.1.2.3 Entscheidung für HD44780-kompatible Displays 9](#_Toc388502929)

[2.1.3 Anzahl der zu verwendenden Mikrocontroller 9](#_Toc388502930)

[2.1.3.1 Zwei (IO und Haupt-)Kontroller 9](#_Toc388502931)

[2.1.3.2 Ein einziger Mikrocontroller 10](#_Toc388502932)

[2.1.3.3 Entscheidung für einen einzigen Mikrocontroller 10](#_Toc388502933)

[2.2 Wahl des Mikrocontrollers 10](#_Toc388502934)

[2.2.1 Microchip PIC 10](#_Toc388502935)

[2.2.2 Xilinx FPGA/CPLD 11](#_Toc388502936)

[2.2.3 Arduino 11](#_Toc388502937)

[2.2.4 ATMEGA328 mit Arduino-Firmware 12](#_Toc388502938)

[2.2.5 Atmel ATMEGA8-16PU 13](#_Toc388502939)

[2.2.6 Entscheidung für Atmel ATMEGA8-16PU 14](#_Toc388502940)

[2.3 Auswahl der Programmiersprache 14](#_Toc388502941)

[2.3.1 Basic 14](#_Toc388502942)

[2.3.2 Assembler 15](#_Toc388502943)

[2.3.3 C 15](#_Toc388502944)

[2.3.4 Entscheidung für C 15](#_Toc388502945)

[2.4 Auswahl der Entwicklungsumgebung 16](#_Toc388502946)

[2.4.1 AVR Studio 16](#_Toc388502947)

[2.4.2 Kontrollerlab 16](#_Toc388502948)

[2.4.3 Vim mit Makefile und externem Compiler 16](#_Toc388502949)

[2.4.4 Entscheidung für Vim mit Makefile und exterem Compiler 17](#_Toc388502950)

[2.5 Auswahl der Schrittmotorentreiber 17](#_Toc388502951)

[2.5.1 Selbst gebaute H-Brücke 18](#_Toc388502952)

[2.5.2 L297/L298 Kombination 18](#_Toc388502953)

[2.5.3 TCA3727 19](#_Toc388502954)

[2.5.4 Entscheidung für TCA3727 19](#_Toc388502955)

[2.6 Übertragung der Befehle 20](#_Toc388502956)

[2.6.1 USB 20](#_Toc388502957)

[2.6.2 Ethernet 20](#_Toc388502958)

[2.6.3 RS232 20](#_Toc388502959)

[2.6.4 Entscheidung für RS232 21](#_Toc388502960)

[2.7 Erstellung des PCBs 21](#_Toc388502961)

[2.7.1 Wahl des Schaltplaneditors 21](#_Toc388502962)

[2.7.1.1 gEDA gschem 21](#_Toc388502963)

[2.7.1.2 Eagle 22](#_Toc388502964)

[2.7.1.3 Target 3001! und Entscheidung für Target 3001! 22](#_Toc388502965)

[2.7.2 Schnittstelle zwischen Modell und Elektronik 22](#_Toc388502966)

[2.7.2.1 Mehrere Stecker 22](#_Toc388502967)

[2.7.2.2 Ein Stecker für alles 23](#_Toc388502968)

[2.7.2.3 Ein Stecker für das Modell 24](#_Toc388502969)

[2.7.2.4 Entscheidung für einen Stecker für das Modell 24](#_Toc388502970)

[2.8 Wahl des Werkzeuges zur Erstellung der Dokumentation 24](#_Toc388502971)

[2.8.1 LaTeX 24](#_Toc388502972)

[2.8.2 LibreOffice 25](#_Toc388502973)

[2.8.3 Microsoft Word 2007 25](#_Toc388502974)

[2.8.4 Entscheidung für Microsoft Office 2007 26](#_Toc388502975)

[3 Ergebnisse 27](#_Toc388502976)

[3.1 Schrittmotorensteuerung 27](#_Toc388502977)

[3.2 RS232-Schnittstelle 29](#_Toc388502978)

[3.2.1 Aufgetretene Probleme 32](#_Toc388502979)

[3.2.2 Protokoll (HPGL) 32](#_Toc388502980)

[3.3 Handsteuerungs-Schnittstelle (I²C-Nunchuck) 34](#_Toc388502981)

[3.4 Schaltplan und PCB 34](#_Toc388502982)

[3.4.1 Schnittstelle zum Modell 34](#_Toc388502983)

[3.4.2 Schnittstelle zum Endschalterkonverter 35](#_Toc388502984)

[4 Abbildungsverzeichnis 36](#_Toc388502985)

[5 Literaturverzeichnis 37](#_Toc388502986)

[6 Anhang 6-1](#_Toc388502987)

[6.1.1 Projektdokumentation 6-1](#_Toc388502988)

[6.1.1.1 Pflichtenheft 6-1](#_Toc388502989)

[6.1.1.2 Terminplan 6-2](#_Toc388502990)

[6.1.1.3 Projekttagebuch 6-3](#_Toc388502991)

[6.1.1.4 Schaltpläne und PCBs 6-14](#_Toc388502992)

# Einleitung

## Motivation

Als letztes Jahr Herr Professor Dipl. Ing. Anton Mayr mit seinem Gerüst für seine Fräse zu uns kam und uns fragte, ob jemand von uns Interesse hätte, diese als Diplomarbeit zu vervollständigen, wurde mir klar, dass hier alle für mich interessanten Themen zur Anwendung kommen. Als außerdem mein Klassenkamerad Lukas Ameisbichler ebenfalls zustimmte, diese Arbeit mit mir gemeinsam zu erledigen, entschied ich mich für dieses Projekt. Letztendlich entschied sich Lukas Ameisbichler jedoch nach den Sommerferien unmittelbar vor Abgabe des Projektantrages für ein anderes Projekt mit einem anderen Projektpartner. Da ich aber nach wie vor sehr interessiert daran war, eine Steuerung für die Fräse zu entwickeln, sich aber so kurzfristig kein anderer Projektpartner finden ließ, entschloss ich mich dazu, das Projekt eben alleine durchzuführen.

## Zielsetzung und Aufgaben

Das Ziel dieser Diplomarbeit ist eine vollständige Steuerung des vorhandenen Plotters via PC zu ermöglichen.

Hierzu sind folgende Aufgaben zu erledigen:

* Programmierung
  + interruptgesteuerte Schrittmotorenansteuerung (TCA3727)
  + Kommunikation mit serieller Schnittstelle (inklusive Hardware-Handshaking)
  + Kommunikation mit Handsteuerung, Display via I2C-Bus
  + Parser für HPGL-Code und die Berechnung/Ausführung für die geparsten Befehle
* Erstellen eines Schaltplans
* Erstellen eines PCBs
* Schreiben der Dokumentation

Nicht im Rahmen der Diplomarbeit zu erledigen ist die Verbauung der Elektronik in das bestehende Fräsengrundgerüst.

Es wird zudem kein Werkzeugwechsel implementiert, dieser muss bei Bedarf von Hand durchgeführt werden, da das Grundgerüst nicht dafür ausgelegt wurde. Um diesen dennoch zu implementieren, müssten Änderungen an der Werkzeugaufnahme durchgeführt, sowie ein geeigneter Platz für die Werkzeuge geschaffen werden. Da sich aber der Umfang der Diplomarbeit ausschließlich auf die Elektronik beschränkt, ist diese Funktion zu umfangreich.

Es ist außerdem kein wesentlicher Bestandteil der Arbeit die Druckgeschwindigkeit zu maximieren. Die Genauigkeit des fertigen Produktes soll aber 1/40mm (HPGL-Standardauflösung) betragen.

## Projektumfeld

Für die Steuerung wurde der Mikrocontroller Atmel ATMEGA8-16PU verwendet, da dieser in der Schule oft verwendet wird, dieser hat 8kB Programmspeicher, 1kB RAM für flüchtige Daten und 512 Byte EEPROM zum nicht-flüchtigen Speichern von Daten zB. für die Konfiguration. Außerdem hat er 23 Ein-/Ausgänge, 2 8-Bit Counter, 1 16-Bit Counter, Hardware UART sowie I²C (alias TWI) Support.

Um die Schrittmotoren ansteuern zu können, musste ein Schrittmotorentreiber verwendet werden, um die notwendige Leistung bereitstellen zu können. Dafür eignete sich der TCA3727 sehr gut, da dieser extra für Schrittmotoren konzipiert wurde und somit die notwendige Strombegrenzung, um die Schrittmotoren nicht zu überlasten, sowie eine bereits auf Schrittmotoren zugeschnittene Eingangslogik besitzt.

Für die Ansteuerung der Fräse wird ein Subset der Plottersprache HPGL verwendet, da hierfür schon Werkzeuge zum Erstellen und Bearbeiten der Daten vorhanden sind[[1]](#footnote-2), diese Sprache extra für Stiftplotter konzipiert wurde und sie somit sehr einfach zu verarbeiten ist.

Die Befehle werden mittels RS-232 vom PC übertragen, da RS-232 mit Pegeln von -15V bis +15V arbeitet (mindestens -3V(HIGH), +3V(LOW)) und der ATMEGA sowie sehr viele andere Logikbausteine mit TTL-Pegeln (+5V(HIGH),0V(LOW)) muss ein Pegelwandler verwendet werden. Hierfür kam der vielfach bewährte MAX232 zur Geltung, welcher mittels Switched-Capacitor-Technik die notwendige Spannung erzeugt. Außerdem wird Hardware-Handshaking verwendet um den Puffer von <1kB[[2]](#footnote-3) nicht zu überfüllen. Später soll durch Setzen eines Makros in einem Header vor der Kompilierung zwischen Software und Hardware-Handshaking umgeschaltet werden können. Hardware-Handshaking wird jedoch präferiert, da dieses durch Übertragungsschwierigkeiten trotzdem früher oder später ankommt, im Gegensatz zu Software-Handshake, der im schlimmsten Fall überhört wird und somit Daten einfach weiter gesendet werden könnten.

Für die Spannungsversorgung der gesamten Elektronik wird ein 12V Steckernetzteil verwendet, da dies die Arbeit erleichtert, die Kosten verringert und diese sehr oft bei kaputten Geräten überbleiben und somit sehr viele vorhanden sind. Der Nachteil ist die oft sehr starke Ungenauigkeit (zu hohe Spannung) im unbelasteten Zustand, doch da sich dies (dank des verwendeten Linearreglers L7805CV) nur auf die Schrittmotoren auswirkt, ist dies kein großes Problem.

## Lösungskonzept

Es wurde der ATMEGA8-16PU gewählt, da in der Schule sehr viele davon vorhanden sind und alle notwendigen Funktionen wie ein Timer für die Schrittmotorenansteuerung, Hardware-UART für die serielle Kommunikation und Hardware-I²C Support für die Ansteuerung des LCDs und die Handsteuerung zur Verfügung stehen. Das Datenblatt[[3]](#footnote-4) des ATMEGAs ist sehr klar strukturiert, übersichtlich geordnet und verständlich formuliert, außerdem sind zahlreiche Tutorials dafür verfügbar[[4]](#footnote-5). Da schon in vorhergehenden Klassen mit diesen Baustein gearbeitet wurde, kannte ich diesen schon sehr gut, wodurch mir die Einarbeitung leichter fiel. Außerdem wurde damals einen Programmierer für diesen Baustein, welcher ermöglicht die meisten (In-System-Programmable) Atmel Mikrocontroller zu programmieren, gebaut. Da dieser Programmierer jedoch auf meinem Windows 7 System nicht wie gewollt funktionierte, musste ich auf ein linuxartiges Betriebssystem namens Gentoo ausweichen. Dies ist zwar in der Handhabung sehr gewöhnungsbedürftig, ich hatte es aber schon fertig auf meinem Laptop installiert und musste nur noch die notwendigen Pakete zur Mikrocontrollerentwicklung installieren. Die Verwendung des Programmierers war nach kurzer Suche nach dem Gerätenamen im Syslog möglich, die einzige Einschränkung bestand darin, nur den USB 3.0 Port des Laptops (Acer Aspire 5950G) für die Entwicklung verwenden zu können, da das Gerät auf den anderen USB-Ports zwar erkannt wird, aber die Kommunikation mit dem Mikrocontroller und somit auch die Programmierung fehlschlägt. Da bei Gentoo keine graphische Oberfläche installiert wurde, um das System möglichst schnell und klein zu halten und den Einrichtungsaufwand gering, wurde der Editor Vim in der Version 7.4 und die Entwicklungswerkzeuge avr-gcc und binutils verwendet.

Zur Kommunikation zwischen Mikrocontroller und PC wurde die Serielle Schnittstelle (RS232 oder offiziell ANSI EIA/TIA-232-F) verwendet um die komplizierte Treiberprogrammierung für die moderneren Schnittstellen wie USB oder FireWire und massiven Leistungsbedarf von Ethernet (mind 10MBit --> volle Auslastung des Mikrocontrollers bei 10MHz Quarz) zu umgehen. Da diese Schnittstelle aber schon auf nahezu allen neuen Endgeräten fehlt, musste ein USB-zu-Seriell Konverter verwendet werden.

Als Befehlssprache wurde HPGL benützt, da diese extra für Stiftplotter (der in dieser Diplomarbeit angesteuert werden soll) konzipiert wurde und somit sehr viele einfache Befehle wie beispielsweise PU (für Pen Up) vorhanden sind, deren Implementierungsaufwand sehr gering ist. Da diese Sprache außerdem früher zur Übertragung und Speicherung von Vektorgrafiken diente, sind sehr viele Programme vorhanden, die dieses Format unterstützen.

Für die Software-Entwicklung wurde ein modularer Ansatz gewählt, bei dem jede Funktion, bevor sie in das Hauptprogramm einfließt, soweit wie möglich einzeln entwickelt und getestet wird. Dies bringt den Vorteil auftretende Fehler vorzeitig erkennen zu können, ohne diese in das Hauptprogramm einzubauen und dort bei groben Fehlern, die einen Ausfall des gesamten Kontrollers zur Folge haben, eine sehr umfangreiche Fehlersuche starten zu müssen.

# Hauptteil - Grundlagen und Methoden

## Erstellen des Blockschaltbilds

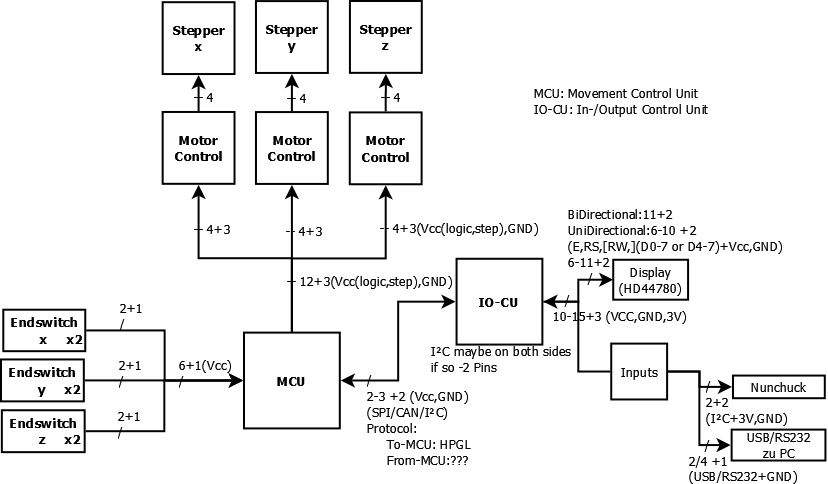
Vorerst war es wichtig sich einen Überblick über die Arbeit zu verschaffen. Daher erfolgte die Erstellung eines Blockschaltbilds um eine grundlegende Struktur festlegen zu können. Die Wahl fiel deshalb auf ein Blockschaltbild, da dieses schnell und ohne viel Wissen über die Hardware erstellt werden kann.

Abbildung Blockschaltbild 2 Kontroller

Eine andere Möglichkeit der Darstellung wäre zum Beispiel die Erstellung eines Schaltplanes gewesen. Dies hätte jedoch schon genaue Informationen über die verwendeten Komponenten und Bauteile notwendig gemacht. Da diese Kenntnisse aber noch nicht vorhanden waren, fiel die Entscheidung zugunsten des oben erwähnten Blockschaltbilds.

Bei der schrittweisen Ausarbeitung des Blockschaltbilds fielen verschiedene Entscheidungen an.

### Wahl der Handsteuerung

Da vorgesehen ist, die Fräse auch mit der Hand steuern und konfigurieren zu können, wurde diese im Blockschaltbild eingezeichnet. Allerdings war es nicht einfach, eine Entscheidung zwischen den verschiedenen Geräten zu treffen, da hier sehr viele verschiedene Verbindungsmöglichkeiten bestehen.

#### Selbst gebaute Steuerung

Die einfachste und am besten konfigurierbare Möglichkeit wäre es sich die Steuerung selbst zu bauen. Hiermit würde die Steuerung genauso aussehen wie gewollt und hätte außerdem die gewünschten Bedienelemente. Der Nachteil wäre, die Hardware als auch den Kontroller und somit auch eine Firmware selbst implementieren bzw. zusammenbauen zu müssen. Außerdem würde die selbstgebaute Steuerung mit den zur Verfügung stehenden Mitteln vom Gehäuse her nicht sehr stabil und komfortabel sein. Daher wurden andere Möglichkeiten vorgezogen.

#### Game-Port Joystick

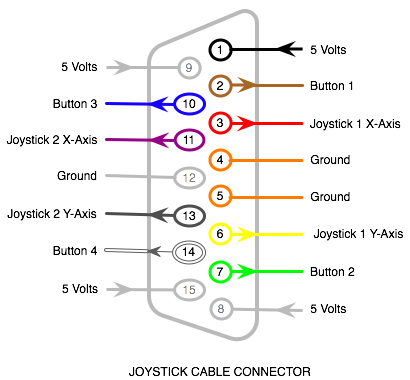


Abbildung Gameport Anschlüsse  
Quelle: http://www.alphalanding.com/rc-track/wp-content/uploads/2009/11/joystick\_port.png

Wie in Abbildung 1 ersichtlich stünden bei einer Gameport-Steuerung 4 Tasten (digital) und 4 Achsen (analog) zur Verfügung. Da hier aber das analoge Signal selbst digitalisiert werden müsste und der Mittelpunkt des Bereiches nicht zwangsläufig die Mittelstellung des Joysticks sein muss, würde die Verarbeitung der Signale erschwert werden. Zusätzlich würde dieser sehr viele Ports des Mikrocontrollers benötigen. Abgesehen davon werden die Gameport-Schnittstellen ab Windows Vista nicht mehr unterstützt[[5]](#footnote-6), da dieser durch den USB abgelöst wurde. Somit sind Geräte für diese veraltete Schnittstelle nicht mehr so leicht zu bekommen. Für einen selbst gebauten Controller wäre diese Schnittstelle sehr gut geeignet, dieser wurde aber wegen oben genannter Gründe nicht realisiert.

Um diese Nachteile zu beheben, müsste ein eigener Mikrocontroller für die Verarbeitung der Signale verwendet und programmiert werden. Da dies mit sehr großem Aufwand verbunden ist, wurde diese Möglickeit zwar als gut vorgemerkt, aber dennoch wurde die Suche nach weniger portintensiven Handsteuerungen fortgesetzt.

#### Entscheidung für Nintendo Wii™-Nunchuck

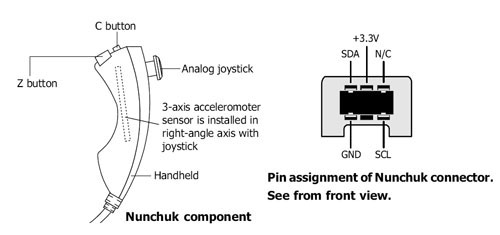


Abbildung Wii Nunchuck  
Quelle: http://www.robotshop.com/media/files/images/design/inex-zx-nunchuk.jpg

Beim Nintendo Wii™ Nunchuck sind wie in der oben angeführten Grafik zu sehen 2 Tasten, 1 Joystick (2 Achsen) und ein Accelerometer verbaut. Diese werden durch die integrierte Schaltung mit einem I²C Bus verbunden, durch den sich diese Daten einfach auslesen lassen. Dies bietet die Vorteile, nur insgesamt 2 Pins am Mikrocontroller zu benötigen und dort zusätzlich mehrere Geräte anschließen zu können. Außerdem müssen die analogen Werte nicht mehr digitalisiert werden und können sofort verwendet werden. Da hier aber auch das Problem besteht, dass der Mittelpunkt des Wertebereichs nicht gleichzeitig der Mittelpunkt des Joysticks ist, fällt hier die Ansteuerung ebenfalls nicht so einfach aus, wie gewünscht. Zudem würde eine Achse mehr benötigt werden (z-Achse für die Höhe).

Da außerdem beim I²C Bus keine Möglichkeit besteht, dass der Slave ohne Aufforderung eine Meldung abgibt, muss jede Änderung an der Steuerung durch eine Abfrage festgestellt werden, wohingegen aber zum Beispiel bei der parallelen Game-Port Schnittstelle die Tasten sehr einfach verodert auf einen externen Interrupt des Mikrocontrollers gelegt werden könnten und deren Abfrage somit wegfallen würde.

#### Entscheidung für Nintendo Wii™ Nunchuck

Die Entscheidung fiel auf diese Steuerung, da trotz der komplizierteren Ansteuerung und der fehlenden Achse bedeutend weniger Pins benötigt werden und somit kein weiterer Mikrocontroller notwendig wurde, welcher den Projektumfang bedeutend vergrößert hätte.

Da es zudem sehr viele hilfreiche Anleitungen zur Ansteuerung des Nunchucks gibt, schien dieses Problem auch schnell lösbar.

### Wahl des Displays

#### HD44780 kompatibles Display

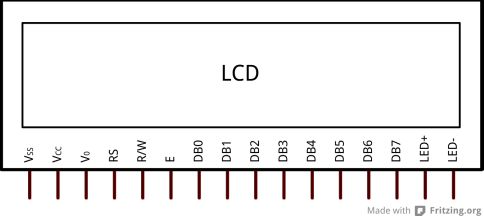


Abbildung Pinout HD44780 kompatibles Display  
Quelle: http://cdn.instructables.com/F2H/QPNE/HJKC4C4G/F2HQPNEHJKC4C4G.LARGE.jpg

HD44780 kompatible Displays sind wegen der hohen Verbreitung des günstigen HD44780 Displaycontrollers ein Quasi-Standard unter Textdisplays. Zur Ansteuerung dieser sind sehr viele Anleitungen und Libraries vorhanden. Zudem kann bei einem Defekt nahezu jedes andere Display, auch wenn es nicht gleich viele Zeichen abbilden kann und von einem anderen Hersteller stammt, verbaut werden. Ein wesentliche Nachteil von Besagtem ist jedoch, der hohe Pinbedarf am Mikrocontroller (mind. 4 Datenleitungen +Enable+RS(RegisterSelect) ergibt bei reinem Schreiben 6 Leitungen). Dieser kann mittels Schieberegister ausgeglichen werden. Da dieses aber wieder zusätzlichen Ansteuerungsaufwand mit sich zieht, könnte ein I²C-Portexpander (zB PCF8574N) verwendet werden um den bereits für die Handsteuerung genutzten I²C Bus auch für das Display zu nutzen, wodurch keine weiteren Pins für das Display aufgewendet werden müssen. Da wegen der hohen Verbreitung und großen Auflagen die Kosten dieser Displays sehr niedrig sind, sind diese für Bastlerprojekte als auch für industrielle Verwendungen sehr gut geeignet. Daher wurde diese Variante letztendlich bevorzugt.

#### Andere Displays

Es gibt Textdisplays die bereits einen I²C-Portexpander oder ein Schieberegister integriert haben, jedoch auch einen anderen Befehlssatz zur Ansteuerung benötigen. Diese sind oft für spezielle Anwendungen konzipiert und somit auch um einiges teurer als die obengenannten Displays mit HD44780-kompatiblem Kontroller. Daher sind diese für die meisten Standardanwendungen ungeeignet.

Grafikdisplays wären auch eine Möglichkeit gewesen, da aber nur Text ausgegeben werden soll, und alles weitere den Aufwand beträchtlich erhöhen würde, den Nutzen aber nicht vergrößern wurde diese Displayart schon sehr bald als ungeeignet abgestempelt. Der Vorteil dieser Displays wäre aber bei diesem Projekt, die auszugebenden Daten durch unterschiedliche Textgrößen und Symbole auf einem Bildschirm übersichtlich darstellen zu können. Da diese Grafikoperationen aber höchst aufwendig sind, sehr viele Daten (zum Beispiel: Display 128x64 Pixel Farbtiefe 8Bit --> 8192Pixel x8Bit --> 8kB Daten/Frame) anfallen und bearbeitet werden müssen um den Inhalt zu aktualisieren und das Bild anschließend wieder anzuzeigen, würde ein anderer Mikrocontroller/FPGA für diesen Zweck benötigt werden. Der für dieses Projekt verwendete Mikrocontroller (ATMEGA8-16PU) hat nicht genügend Speicher um ein komplettes Bild zu speichern oder im Ganzen zu bearbeiten[[6]](#footnote-7).

#### Entscheidung für HD44780-kompatible Displays

Da diese Displays so verbreitet sind und es für die Ansteuerung dieser sehr viele Anleitungen gibt, wurden diese ausgewählt. Um zusätzlich Ports zu sparen wurde die Verwendung eines I²C-Portexpanders präferiert. Dieser verlangsamt zwar die Kommunikation mit dem Display, da dieses aber ohnehin nicht sehr reagieren können muss schnell (max 5 Updates/sek um das Ablesen zu ermöglichen)[[7]](#footnote-8), ist dies akzeptabel.

### Anzahl der zu verwendenden Mikrocontroller

Es wäre möglich gewesen, die Kontroller nach Funktion aufzuteilen, diese somit zu entlasten und dadurch auf kleinere, günstigere Kontroller umsteigen zu können. Außerdem könnten dadurch die Tätigkeiten parallelisiert werden und die Ansteuerung beschleunigt werden.

#### Zwei (IO und Haupt-)Kontroller

Um die Kontroller zu entlasten und eventuell kompliziertere Schnittstellen zwischen Plotter und PC verwenden zu können, müssten zwei Kontroller verwendet werden. Da dies aber bedeuten würde auch zwei Kontroller programmieren zu müssen und ein Kommunikationsprotokoll zwischen diesen zu entwickeln schien dies ein sehr zeitaufwendiges Verfahren zu sein. Die Vorteile davon wären aber zum einen mehr Rechenzeit für das (Grafik-)Display, die Handsteuerung und die Schnittstelle zwischen PC und Plotter als auch Ports am Mikrocontroller zur Verfügung zu haben, als auch bei mehreren Entwicklern dadurch eine bessere Arbeitsaufteilung ermöglichen zu können. Da das Projekt aber nur von einer Person bearbeitet wird, fiel der zweite Vorteil weg und die Rechenzeit eines Mikrocontrollers ist für ein simpleres Display und eine einfache PC-Schnittstelle ausreichend. Wodurch nur die Nachteile des zusätzlichen Overheads überbleiben.

#### Ein einziger Mikrocontroller

Obwohl einem einzigen Mikrocontroller die Probleme mit der Kommunikation zwischen den einzelnen Mikrocontrollern und der Arbeitsaufteilung wegfallen, tauchen hier andere Probleme auf. Die Leistung ist begrenzt und muss sparsam auf die verschiedenen Tätigkeiten aufgeteilt werden und außerdem stehen weniger Pins zur Verfügung. Interrupts helfen bei der Aufteilung der Leistung, da hiermit für alle Nebenaufgaben, die dies unterstützen, nur dann Rechenzeit verwendet werden muss, wenn ein bestimmtes Ereignis eintritt (zum Beispiel bei der seriellen Schnittstelle, dann wenn der Empfangspuffer voll ist). Dadurch kann die ansonsten durch Warten verschwendete Zeit für andere Dinge verwendet werden. Da die Pinanzahl aber für manche Zwecke wie die Verwendung des Displays sehr gering ist, könnten I²C-Portexpander oder Schieberegister verwendet werden um dennoch die benötigten Ports verwenden zu können. Diese sind zwar beträchtlich langsamer, reichen aber für die meisten Anwendungen aus.

#### Entscheidung für einen einzigen Mikrocontroller

Die Entscheidung fiel auf einen einzigen Mikrocontroller, da somit der Overhead der Kommunikation unter den Mikrocontrollern wegfällt und die Programmierung vereinfacht wird. Der Nachteil dieser Entscheidung ist aber, dass keine umfangreichen PC-Schnittstellen, Displays oder Handsteuerungen verwendet werden können, da diese entweder zu viel Leistung, Pins oder beides beanspruchen würden. Daraus ergibt sich ein weiterer Nachteil bei späteren Erweiterungen bezüglich Display, Handsteuerung oder PC-Schnittstelle, bei denen die Leistung des aktuellen Mikrocontrollers nicht ausreicht, da der Aufwand durch Einführung eines weiteren Mikrocontrollers weit höher ist, als wenn dieser von Anfang an eingeplant gewesen wäre. Das durch die zu hohe Pinanzahl des Displays verursachte Problem wäre mittels I²C-Portexpander, der voraussichtlich bei der Ansteuerung des aktuellen Displays Verwendung finden wird, auszugleichen.

## Wahl des Mikrocontrollers

Zur Realisierung der Steuerung gab es viele Möglichkeiten. Zuerst musste daher der Kontroller ausgewählt werden, der die wichtigste Steueraufgabe in diesem System übernimmt.

### Microchip PIC

Da bereits Debugger, Entwicklungsumgebung und der notwendige Baustein 16F260(TODO), sowie einige Testprogramme vorhanden waren, bestand vorerst die Überlegung sich diese zunutze zu machen. Allerdings fiel schließlich die Entscheidung gegen die Verwendung dieses Bausteines, da im Falle von eventuell auftretenden Schwierigkeiten keine Unterstützung durch fachkundige Personen zu erwarten gewesen wäre. Aus diesem Grund wurde dieser Baustein für dieses Projekt als ungeeignet angesehen.

### Xilinx FPGA/CPLD

Als nächste Möglichkeit wurde die Verwendung von Logikelementen wie FPGAs oder CPLDs in Betracht gezogen. Einerseits wäre der Einarbeitungsaufwand dieser verschwindend gering gewesen, da bereits im Unterricht damit gearbeitet wurde. Andererseits hätte sich die Entwicklung von prozeduralen Sequenzen damit um einiges komplizierter gestaltet als mit einem Mikrocontroller, vor allem wegen dem eigentlichen Vorteil der Logikbausteine alles parallel "auszuführen". Dies hätte bedeutet, dass für jede Prozedur eine Statemachine verwendet hätte werden müssen, die wiederum sehr viele der internen Makrozellen/(TODO FPGA-Zellen) benötigt hätte, wodurch sich diese Bausteine für diese Aufgabe als nicht geeignet erwiesen. Andererseits hätte man aber die Schrittmotorentreiber und die restlichen nebenläufigen Aktivitäten wie Kommunikation mit dem Display und der Handsteuerung, sowie die Aufbereitung der Eingangssignale (Endanschläge) damit sehr gut lösen können. Da dies aber die Diplomarbeit unnötig verkompliziert hätte und die Nachteile (zusätzliche Entwicklungsumgebung, nicht vorhandener Programmierer, neue Fehlerquelle, ...) überwogen, wurde schließlich eine reine Mikrocontroller-Lösung bevorzugt.

### Arduino[[8]](#footnote-9)

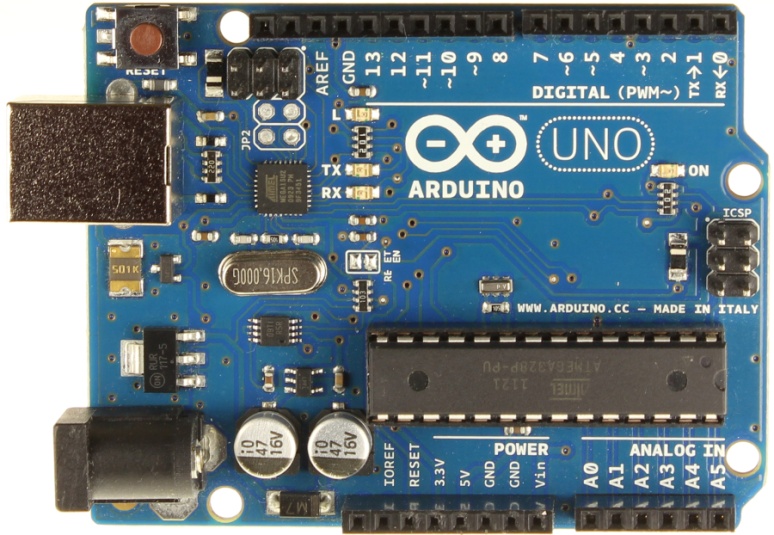


Abbildung Arduino Uno  
Quelle: http://arduino.cc/en/uploads/Main/ArduinoUno\_R3\_Front.jpg

Um die Entwicklung mittels Objektorientierung und vorgefertigten Libraries zu vereinfachen, bot es sich an einen Arduino Uno (siehe Abbildung 4) zu verwenden, da dieser an der Schule verfügbar gewesen wäre. Allerdings sind Arduinos für schnelle Bastlerprojekte gedacht und haben somit eine fixe Platine mit einer gewissen Größe auf die Zusatzmodule gesteckt werden können und in diesem Fall auch gesteckt werden hätten müssen. Somit wäre es möglich gewesen während des Betriebes Platinen hinzuzufügen und abzunehmen. Diese Vorgangsweise schien für eine Fräse nicht geeignet, da dadurch während des Betriebes nicht vorhersehbare Fehler auftreten hätten können. Bei ausschließlicher Nutzung der Fräse durch den Entwickler selbst, hätte dies kein großes Problem dargestellt, da dieser über mögliche Fehlerquellen Bescheid weiß und somit entsprechende Maßnahmen setzen könnte um eventuell auftretende Fehler zu beheben. Bei Verwendung durch andere Personen hätte jedoch ein Gehäuse für den Platinenstapel gebaut bzw. gekauft werden müssen, um unbeabsichtigte Vorgänge zu minimalisieren. Durch die Größe des Arduinos schien diese Handhabe nicht sehr praktikabel.

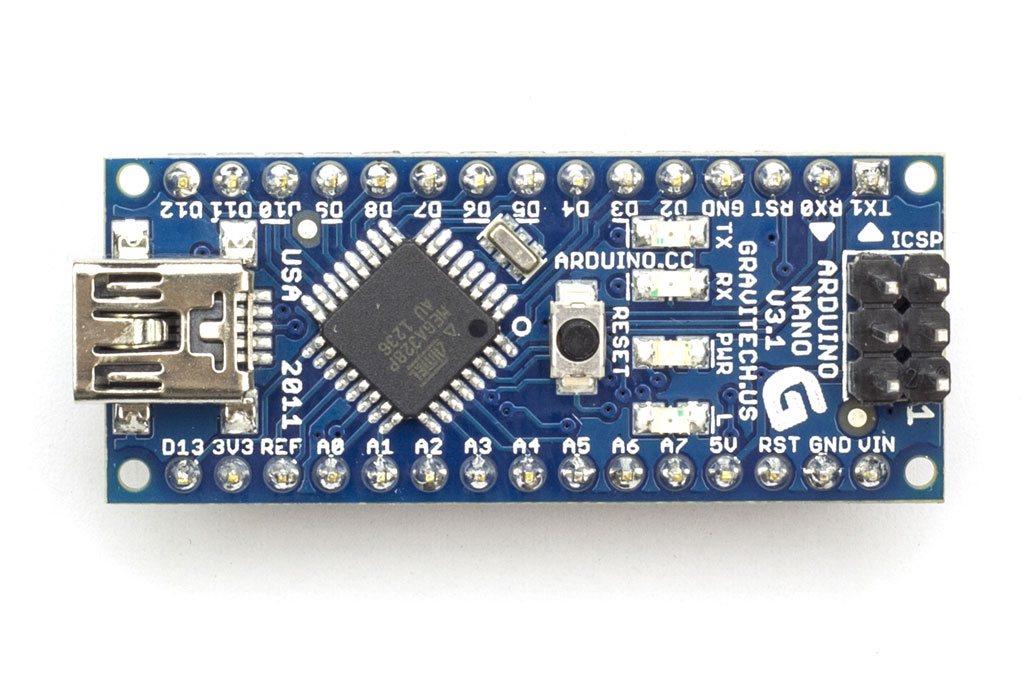


Abbildung Arduino Nano  
Quelle: http://arduino.cc/en/uploads/Main/ArduinoNanoFront\_3\_lg.jpg

Auch die Möglichkeit einen Arduino Nano (Abbildung 5) zu verwenden, wurde verworfen, da dieser an der Schule nicht verfügbar war. Aufgrund der Tatsache, dass ein Arduino Nano auf einem ATMEGA328/ATMEGA168 aufbaut, ergab sich die Überlegung diesen Baustein zu verwenden und Arduino-Firmware aufzuspielen. Dies hätte die Kosten des Arduinos einerseits beträchtlich verringert. Andererseits wäre dadurch eine eigens auf das Projekt zugeschnittene Platine erforderlich gewesen, welche zeitaufwändig hergestellt werden hätte müssen. Dennoch wäre der Baustein selbst aus programmtechnischer Sicht immer noch als Arduino zu verwenden gewesen, mit all seinen Vor- und Nachteilen.

### ATMEGA328 mit Arduino-Firmware

Abgesehen von der Tatsache, dass der Arduino Nano wegen oben genannter Gründe nicht praktikabel war, musste eine neue Lösung gefunden werden. Als nächstes wurde in Erwägung gezogen einen Atmel ATMEGA328 mit Arduino-Firmware zu verwenden. Dieser hätte somit aus programmtechnischer Sicht wie ein Arduino verwendet werden und trotzdem auf eine eigens dafür zugeschnittene Platine montiert werden können, wodurch die hardwaretechnischen Nachteile des Arduinos weggefallen wären. Da der Arduino aber speziell zu programmieren gewesen wäre (USB), hätten zusätzliche Bauteile eingefügt werden müssen, um diese Programmierung ermöglichen zu können.

Außerdem wäre die Library nach mehrmaligem Lesen der Dokumentation aufgrund einiger Probleme (zum Beispiel hätte die Verwendung der Schrittmotoren nicht interruptgesteuert erfolgen können und somit hätte während der Bewegung eines Motors nichts anderes erledigt werden können) auch nur teilweise zu gebrauchen gewesen, wodurch diese Funktionen durch Verwenden von Standard-AVR C-Code implementiert hätte werden müssen. Allerdings könnte nur bei eigener Implementierung sichergestellt werden, dass ausschließlich die gewollten Ports verwendet und keine anderen benötigt werden. Außerdem könnten eventuelle Optimierungen nur durchgeführt werden, wenn alle notwendigen Programmteile selbst entwickelt wurden.

Da dies bei der Arduino Library nicht möglich gewesen wäre, wurde letztendlich auch dieser Entwicklungsschritt abgelehnt.

### Atmel ATMEGA8-16PU

Der Atmel ATMEGA8-16PU bietet folgende für die Verwendung in dieser Arbeit notwendigen Funktionen:

1 Timer/Counter mit Interrupt und variablen Prescaler für die Schrittmotorenansteuerung,  
1 Hardware-UART für RS232 Kommunikation,  
1 Hardware-I²C (TWI) um die Handsteuerung und das Display anzusteuern.

Außerdem waren einige andere nützliche Funktionen vorhanden:

Watchdog mit variablem Prescaler um bei Programmfehlern den Kontroller rücksetzen zu können,  
Brown-Out Detector, welcher bei Spannungseinbrüchen, die die Peripherie möglicherweise nicht verkraftet, den Kontroller zurücksetzt um mögliche Fehler zu unterbinden.

Einzig das Fehlen eines Debuggers für den Atmel ATMEGA8-16PU[[9]](#footnote-10) sollte die Verwendung etwas erschweren. Doch dies war durch Ändern des Programmierstils lösbar. Daher fiel die Entscheidung letztendlich auf diesen Baustein.

### Entscheidung für Atmel ATMEGA8-16PU

Schließlich fiel die Entscheidung zugunsten des Atmel ATMEGA8-16PU[[10]](#footnote-11), da dieser in der Schule vorrätig war und bei Unklarheiten auch die Unterstützungsmöglichkeit durch den Betreuungslehrer gegeben war. Als weiterer Pluspunkt für die Verwendung des Atmel ATMEGA8-16PU sprach, dass sowohl der nötige Programmierer vorhanden als auch die Entwicklung mittels kostenloser Software möglich war. Da alle benötigten Funktionen vorhanden waren, sprach auch nur das Fehlen des Debuggers dagegen.

Dies lies sich aber mit einigem Mehraufwand wie folgt lösen:

* Durchführen kleiner Änderungen wie das Vorziehen der Seriellen Steuerung im Terminplan um die Fehlerausgaben so früh wie möglich implementieren zu können
* Einige zusätzliche Debug-Ausgaben auf LEDs
* Aber auch größere Änderungen in der Entwicklungsmethodik auf einen Ansatz alle Funktionalitäten, die implementiert werden sollen, soweit wie möglich getrennt zu entwickeln und zu testen und diese erst bei deren Fertigstellung in das Hauptprogramm zu integrieren und dort anzupassen.

Dies brachte den Vorteil die Ursache von schwerwiegenden Fehlern, die einen Absturz des gesamten Kontrollers nach sich ziehen können, schneller zu finden und nicht in das Hauptprogramm einzuschleppen, wo diese schwieriger zu finden wären. Außerdem können so auch keine Fehler durch andere Programmkomponenten entstehen, die für diese Funktionalität nicht benötigt werden.

## Auswahl der Programmiersprache

Da der zu verwendende Kontroller jetzt ausgewählt war, konnte die Programmiersprache in der dieser programmiert werden sollte ausgewählt werden.

### Basic

Der bascom-avr Compiler wird im Internet manchmal als besonders einsteigerfreundlich beworben. Nach kurzer Recherche stellte sich heraus, dass kompilierte Programme auch für alle AVR Mikrocontroller mit RAM verwendbar sind und sehr umfangreiche Libraries vorhanden sein sollten[[11]](#footnote-12). Die Nachteile wären gewesen, dass im Falle von Schwierigkeiten bei der Software nur im Internet nach Hilfe gesucht werden hätte können, da das Programmieren in Basic in Schulen kaum mehr unterrichtet wird. Zudem wäre es erforderlich gewesen einen Compiler für diese Sprache zu kaufen. Da diese Sprache aber sowieso eher zu Lernzwecken und nicht für den eigentlichen produktiven Gebrauch konzipiert wurde und der Entwickler bisher nur über geringfügige Kenntnisse in Basic verfügt, wurde eine andere Alternative gesucht.

### Assembler

Bereits vor ein paar Monaten entwickelte der Verfasser dieser Diplomarbeit einige Testprogramme und sogar eine PWM-Drehzahlsteuerung für einen Elektromotor für einen AVR-Mikrocontroller in Assembler. Beim Verwenden von Assembler besteht die größte Entwicklungsfreiheit und solange die Befehle richtig angewandt werden, kann man damit um einiges flexibler und hardwareoptimierter programmieren. Ein Minuspunkt vom Programmieren in Assembler ist allerdings, dass sich sehr leicht Fehler einschleichen können, die oft nur schwer auffindbar sind. Ein weiterer Nachteil ist, dass Assembler in der Schule selten Anwendung findet und daher nur wenige fachkundigen Ansprechpersonen vorhanden gewesen wären. Aus diesem Grund wurde schließlich auch die Verwendung dieser Programmiersprache verworfen.

### C

Abgesehen von Assembler, welcher hauptsächlich für die Programmierung von Optimierungen Verwendung findet, ist C die Standardsprache für Betriebssysteme und andere hardwarenahen Programme. Darüber hinaus sind sehr viele Anleitungen für die AVR-Programmierung in C vorhanden. Viele Libraries wurden bereits in C geschrieben, ebenso wie die früher in der Schule geschriebenen Testprogramme.

Da der Compiler auch kostenlos für alle derzeit in Verwendung befindlichen Betriebssysteme vorhanden ist, konnte sofort mit dem Programmieren begonnen werden.

### Entscheidung für C

Die Entscheidung fiel auf C, da diese Sprache für die Mikrocontrollerprogrammierung am stärksten verbreitet ist und dadurch auch sehr viele Libraries und Tutorials vorhanden sind.

Außerdem sind im Gegensatz zu reinem Assembler Standarad-Libraries vorhanden, auf die zurückgegriffen werden kann.

Da C rein prozedural ist, wird dadurch im Gegensatz zu objektorientierten Sprachen nicht sehr viel Arbeitsspeicher verbraucht, wodurch der meiste für die Nutzung durch tatsächliche Daten zur Verfügung steht. Zusätzlich kann beispielsweise für Optimierungen mittels asm-Direktive Assembler Code eingebunden werden. Zudem ist der Compiler frei verfügbar.

## Auswahl der Entwicklungsumgebung

Nach Festlegung der Programmiersprache konnte eine Entwicklungsumgebung ausgewählt werden. Dafür gab es mehrere Auswahlkriterien. Diese hängten in erster Linie vom Betriebssystem ab.

### AVR Studio[[12]](#footnote-13)

Die erste Wahl fiel auf ATMELs AVR Studio, da dies mit der Version 4 in der Schule verwendet wurde und dort gut funktionierte. Bedauerlicherweise stellte sich heraus, dass Version 4 von ATMEL nicht mehr unterstützt wird und aufgrund von Treiberproblemen nicht mit dem selbstgebauten Programmierer funktionierte. Auch bei der als nächstes gewählten Version 6 ergaben sich ähnliche Schwierigkeiten. Somit war es notwendig auf ein linuxartiges Betriebssystem zu wechseln, da hier keine zusätzlichen Treiber notwendig sind.

### Kontrollerlab[[13]](#footnote-14)

Wie bereits oben erwähnt, wurde um Treiberprobleme zu vermeiden, auf ein linuxartiges Betriebssystem ausgewichen. Hier bot sich die integrierte Entwicklungsumgebung Kontrollerlab an. Es tauchten bei der Installation jedoch einige Probleme auf: Die zur Verfügung gestellten Ubuntu Binaries funktionieren unter dem verwendeten Ubuntu 12.4 LTS 64-Bit nicht. Da es selbst nach Kompilierung des Quellcodes und wechseln des Systems auf Debian und Gentoo (an und für sich zwei sehr stark konfigurierbare und flexible Betriebssystemderivate), nicht funktionierte, musste diese IDE abgeschrieben werden.

### Vim mit Makefile und externem Compiler

Aufgrund nicht zur Verfügung stehender Zeit bei der Suche nach weiteren möglicherweise besser geeigneteren Alternativen wurde schließlich der Editor Vim ausgewählt. Dieser Kommandozeileneditor verfügt über Syntaxhighlighting und weitere nützliche Features, wie die Möglichkeit vom Editor aus Kommandozeilenbefehle oder Makefiles auszuführen und dabei entstandene Fehler sofort einer Zeile zuordnen zu können. Da Vim alleine aber noch nicht zur Entwicklung reicht wurde avr-gcc als Compiler und avrdude zur Übertragung auf den Mikrocontroller verwendet. Die gemeinsame Nutzung dieser Werkzeuge wurde durch ein Makefile gelöst, da dies wie oben erwähnt direkt im Editor ausgeführt werden kann.

### Entscheidung für Vim mit Makefile und exterem Compiler

Da die getesteten Entwicklungswerkzeuge nicht funktionierten, wurde eine unproblematischere nicht integrierte Entwicklungsumgebung bevorzugt. Da die Kenntnisse von Vim bereits vorhanden waren, konnte hiermit sehr schnell losgelegt werden. Da Vim außerdem auch die Möglichkeit besitzt, Makefiles direkt vom Editor aus ausführen zu können, ergibt sich dadurch ein ähnliches Verhalten zu den integrierten Entwicklungsumgebungen, bei denen durch einen Tastendruck kompiliert und das Ergebnis auf den Mikrocontroller geflasht werden kann. Nur, dass bei der Verwendung von nicht integrierten Entwicklungsumgebungen entweder alle Schritte per Hand durchgeführt werden müssen oder aber ein Makefile erstellt werden muss, welches die gewünschten Befehle beinhaltet.

## Auswahl der Schrittmotorentreiber

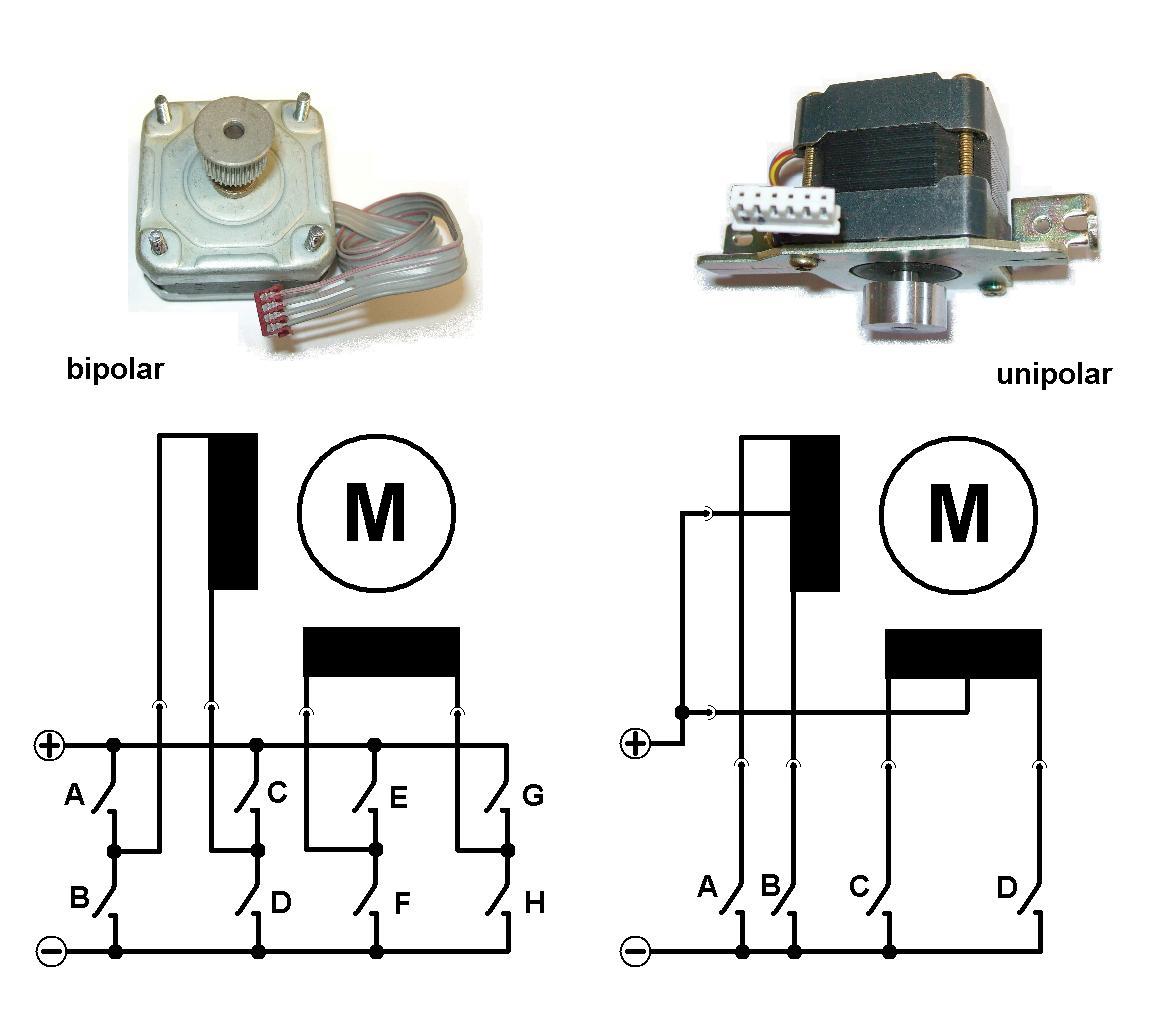


Abbildung Schrittmotorenarten  
Quelle: https://upload.wikimedia.org/wikipedia/commons/4/48/Stepmotscheme.jpg

Nach Klärung der Entwicklungsumgebung mussten nun die restlichen Teile bestimmt werden. Zuerst wurden die Schrittmotorentreiber ausgewählt. Hier war es notwendig zuerst zu analysieren, ob die Schrittmotoren uni- oder bipolar angesteuert werden sollten (siehe obenstehender Abbildung). Beides hat seine Vor- und Nachteile. Die unipolaren Schrittmotoren sind zwar einfacher anzusteuern, haben dafür aber nur die halbe Leistung, da immer nur die halbe Spule verwendet wird. Bipolare Schrittmotoren hingegen verwenden die gesamte Spule, benötigen dafür aber eine H-Brücke um den Stromfluss und somit die Drehrichtung umkehren zu können. Das führt dazu, das die Ansteuerung komplizierter, aber auch der Wirkungsgrad beträchtlich erhöht wird.

Da nahezu jeder Schrittmotor bipolar verwendet werden kann, aber nicht jeder unipolar und das Drehmoment bei bipolaren Motoren größer ist wurde die bipolare Ansteuerung bevorzugt.

### Selbst gebaute H-Brücke

Als Ansteuerungsmöglichkeit wäre es möglich gewesen, die H-Brücke selbst aus MOSFETs aufzubauen. Dies hätte den Vorteil mit sich gebracht, dass diese besser als eine integrierte Schaltung gekühlt worden wäre (da die Bauteile einzeln herumstehen, müssen diese auch nicht so stark gekühlt werden) und die Teile nach Bedarf ausgewählt werden können. Jedoch wären die Kosten dieser Lösung höher gewesen, der Arbeitsaufwand und auch die Komplexität des PCBs größer. Da sich herausstellte, dass integrierte Bausteine für diese Zwecke auch nicht stark gekühlt werden müssen und diese außerdem im Gegensatz zu den benötigten MOSFETs in der Schule lagernd waren, erschien das eigenständige Erstellen einer H-Brücke nicht mehr rentabel.

### L297/L298 Kombination

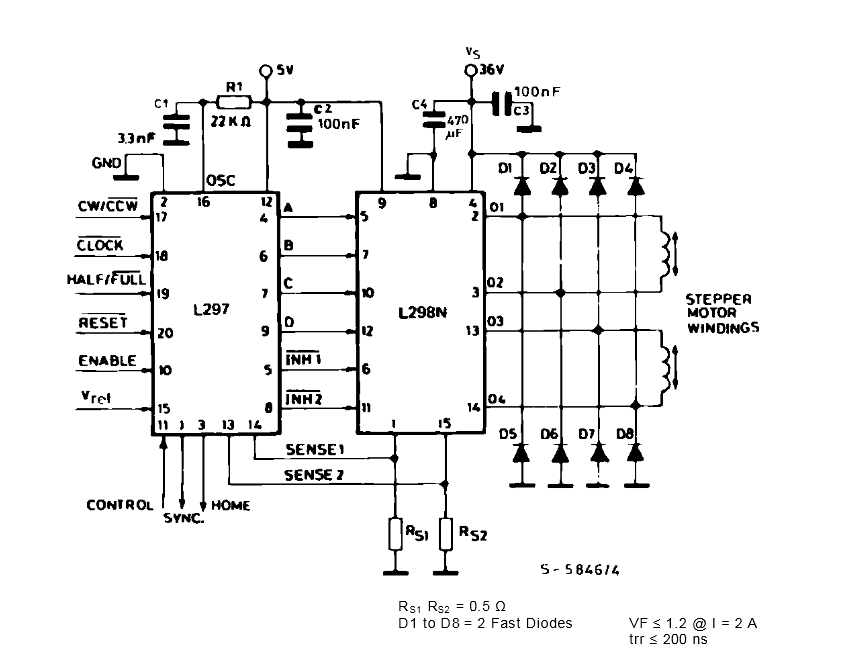


Abbildung Verwendung L297 mit L298  
Quelle: [ST\_AN470] Seite 14 Figure 21

Die beiden Bausteine L297 (Schrittmotoren-Kontroller) und L298 (H-Brücke) wurden extra für die Kombination dieser miteinander konzipiert und erlauben es Schrittmotoren ohne großen Aufwand mit zwei Ausgängen vom Mikrocontroller (Richtung und Takt) im Voll- oder Halbschrittbetrieb anzusteuern. Die Schaltung erledigt den Rest (Strombegrenzung, Umrechnung der Signale in Schrittmuster). Der Nachteil sind die hohen Kosten, die vielen Zusatzteile (Freilaufdioden, Abblock-/Speicherkondensatoren) und der hohe Stromverbrauch der Bauteile. Da diese Schaltung für jeden Schrittmotor benötigt würde, und der Betreuungslehrer davon abriet, musste eine Alternative gesucht werden.

### TCA3727

Der TCA3727 ist ein reiner Schrittmotorentreiber, der extra für diesen Zweck entwickelt wurde, wodurch die Ansteuerung der Schrittmotoren durch interne Logik mit nur zwei Leitungen ermöglicht wird. Außerdem haben die Ausgänge eine eigene Spannungsversorgung, wodurch die Schrittmotoren mit größerer Spannung angesteuert werden können und die Eingänge trotzdem TTL-kompatibel bleiben, dafür sind aber auch zwei Spannungsquellen erforderlich. Ein weiteres Feature ist die Strombegrenzung, welche über zwei Eingänge pro Phase geregelt werden kann,um die Schrittmotoren nicht zu überlasten.

### Entscheidung für TCA3727

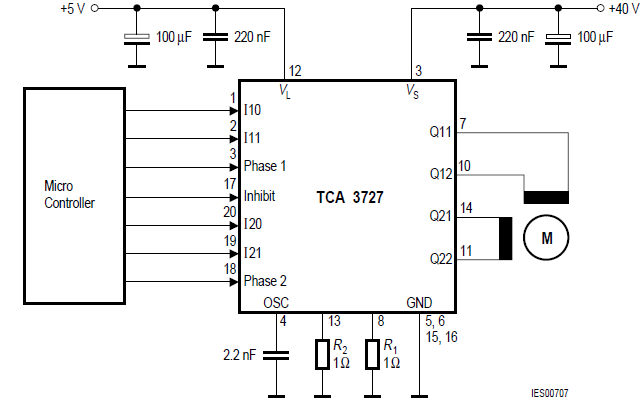


Abbildung Beispiel für Verwendung von TCA3727  
Quelle: [TCA3727]: Figure 5

Der Betreuungslehrer schlug den TCA3727 vor, da dieser in der Schule lagernd war und dessen Beschaltung sehr einfach ist. Der TCA3727 ist ein extra für Schrittmotoren konzipierter Treiberbaustein, bei dem die Ansteuerung mittels Schrittmuster vorgenommen wird. Somit ist die Aufgabe des Schrittmotoren-Kontrollers von dem Mikrocontroller zu erledigen. Da hier keine zusätzlichen Freilaufdioden benötigt werden und die Strombegrenzung vom Chip über Sense-Wiederstände selbst durchgeführt wird sowie in der Schule Testboards dafür vorhanden waren, war die Einarbeitung nicht sehr schwierig.

## Übertragung der Befehle

Als nächster Schritt waren Überlegungen über die Schnittstelle anzustellen, mit der die Befehle vom PC auf den Mikrocontroller übertragen werden sollten. Hier gab es die Möglichkeit moderne Schnittstellen zu verwenden und somit komplexe Algorithmen und Treiber schreiben zu müssen oder auf ältere Schnittstellen zurückzugreifen, bei denen die Gefahr bestand, dass diese bald auf keinen PCs mehr vorhanden sind.

### USB

Eine der meistverbreitetsten Schnittstellen für Drucker und andere Dinge ist derzeit USB. Aus diesem Grund wäre es denkbar USB zu verwenden. Zudem ist diese Schnittstelle auf allen modernen Geräten vorhanden. Das Problem ist jedoch die komplexe Verwendung, da PC-seitig ein eigener Treiber notwendig werden würde, das Übertragungsprotokoll auch wesentlich komplizierter ist als bei anderen Schnittstellen und die Übertragungsgeschwindigkeiten zu hoch sind (mind 1,5MBit/s bei USB1.x Low Speed) wodurch sehr viel Leistung für die Schnittstelle verwendet werden müsste. Da das Signal wegen der hohen Übertragungsgeschwindigkeiten zusätzlich noch differentiell übertragen wird, müsste die Elektronik auch darauf abgestimmt werden. Somit würde die Verwendung dieser Schnittstelle die Arbeit stark verkomplizieren.

### Ethernet

Ein anderer Ansatz wäre Ethernet, wodurch das Gerät als Netzwerkdrucker verwendet werden könnte und somit bequem von jedem netzwerkfähigem Gerät aus gedruckt werden könnte. Dies würde wegen der hohen Komplexität die Verwendung eines zweiten Mikroprozessors für die Bearbeitung des Signals notwendig machen. Da die minimale Übertragungsrate hier bei 10Mbit/s liegt und die Übertragung wegen vieler möglicher Geräte im Netzwerk und der Notwendigkeit die OSI-Schichten selbst zu implementieren wurde diese Möglichkeit für diesen Zweck als zu aufwendig bewertet.

### RS232

Selbst wenn fast alle aktuell erscheinenden PCs keine serielle Schnittstelle mehr beinhalten, ist RS232 eine der simpelsten Schnittstellen. Daher wird sie häufig von Bastlern eingesetzt. Sie ist aber auch aus selbigem Grund häufig auf Industriegeräten wie Fräsen, oder als Konfigurationsschnittstelle auf Routern und anderen Geräten anzutreffen.

### Entscheidung für RS232

Die simpelste Methode war die Verwendung von RS232. Dies ermöglicht es Daten, ohne der Notwendigkeit einen eigenen Gerätetreibers für den Plotter zu schreiben, in geeignet niedrigen Bitraten von standardmäßigen 9600 Baud (Bit/s) zu übertragen. Der große Nachteil bei der Verwendung dieser Schnittstelle ist das hohe Alter des Standards und die Verdrängung durch USB, wodurch die Serielle Schnittstelle auf neueren Geräten selten verbaut wird. Da aber USB-zu-Seriell Wandler verfügbar sind, kann dieses Problem zumindest vorläufig für diese Arbeit umgangen werden.

## Erstellung des PCBs

Da nun alle wesentlichen Entscheidungen getroffen wurden, konnte mit der Erstellung des PCBs begonnen werden. Hierzu musste zuerst ein Schaltplan entworfen werden. Da für diesen aber noch kein Programm ausgewählt wurde, musste dies nun getan werden.

### Wahl des Schaltplaneditors

#### gEDA gschem

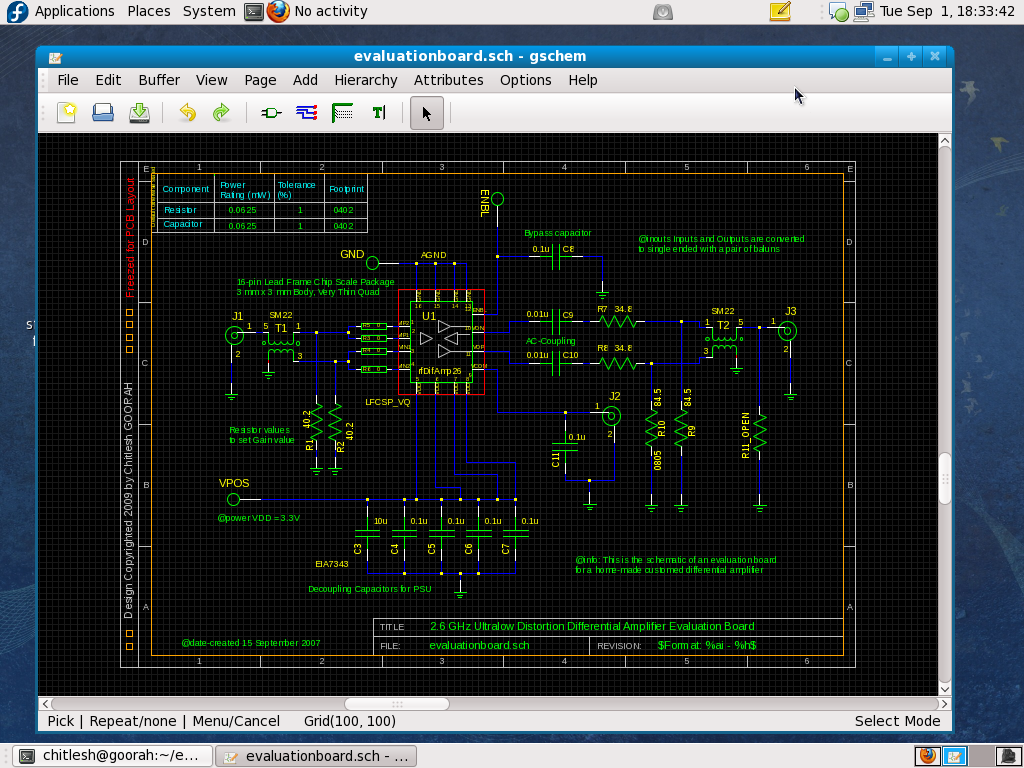


Abbildung gschem  
Quelle: http://vishnuwhitestone.files.wordpress.com/2011/08/gschem.png

Gschem ist ein Werkzeug zur Erstellung von Schaltplänen auf POSIX Systemen. Als Teil der gEDA Suite kann man mit gschem erstellte Schaltpläne in PCB, ein weiterer Teil von gEDA, verwenden um dort die Platine zu erstellen. Einige Nachteile davon wären die eingeschränkte Nutzbarkeit unter Windows und die Notwendigkeit für manche nicht so häufige Bauteile erst Symbole herunterladen zu müssen, da diese nicht mitgeliefert werden. Für manche selten verwendeten Bauteile kann es passieren, dass die Symbole selbst erstellt werden müssen. Oft sind diese aber schon von der Community erstellt worden. Da es als freie Software entwickelt wurde, gibt es auch keine Einschränkungen in der Größe der Platine oder der Anzahl an verfügbaren Kontakten. Zusätzlich können, sofern für die verwendeten Symbole Simulationsdaten vorhanden sind, mittels ngspice Simulationen am erstellten Schaltplan durchgeführt werden.

Da hierfür aber auf dem Linux Entwicklungssystem eine graphische Oberfläche installiert werden müsste, deren Installation aber nicht funktionierte und sehr zeitintensiv verlief, musste diese Möglichkeit für dieses Projekt abgelehnt werden.

#### Eagle

Eagle ist ein sehr verbreitetes Werkzeug zur Erstellung von Schaltplänen. Da aber in der freien Variante eine maximale Platinengröße von 100x80mm besteht, wurde dies abgelehnt.

#### Target 3001! und Entscheidung für Target 3001!

Da Target in der Version v15 in der Schule verwendet wird, fiel die Wahl schließlich darauf.

Ein Vorteil ist die große Bauteildatenbank, bei der annähernd jedes benötigte Bauteil vorhanden ist. Hier war auch nur eine eingeschränkte kostenlose Version vorhanden. Die Einschränkungen schienen allerdings akzeptabel, da lediglich die Anzahl der verwendbaren Pins auf 250 beschränkt wurde. Weitere Vorteile wären der Import von EAGLE Daten und die Möglichkeit Teile des Schaltplanes durch Copy&Paste in die Dokumentation einfügen zu können.

### Schnittstelle zwischen Modell und Elektronik

Da es viele verschiedene Möglichkeiten gibt die beiden Dinge zu kombinieren, wurde eine Gegenüberstellung der Möglichkeiten vorgenommen.

#### Mehrere Stecker

Dies wäre leitungstechnisch die einfachste Möglichkeit, da hier für jeden Schrittmotor und anderes ein eigener Stecker verwendet wird und die Leiterbahnen somit sehr kurz sind und sich nicht so leicht gegenseitig in die Quere kommen können. Da aber bei dieser Variante, die meisten Stecker nicht am Rand sein werden würde es schwierig werden hier ein Gehäuse zu verwenden, das beispielsweise für Änderungen oder Verbesserungen ohne viel Arbeit abgenommen und wieder aufgesteckt werden kann ohne Gefahr zu laufen die einzelnen Stecker miteinander zu verwechseln.

#### Ein Stecker für alles

Hier würde das Gehäuse sehr einfach ausfallen, da nur ein Loch benötigt werden würde. Es bestünde keine Gefahr den Stecker zu verwechseln und bei den meisten Steckern auch keine Gefahr den Stecker zu verdrehen. Ein großes Problem wäre allerdings die Belastbarkeit des Steckers, da die meisten gewöhnlichen Stecker nur für 2 Ampere konzipiert sind.

Da die Schrittmotoren für jede Spule 0,24A benötigen, würden durch die Schrittmotoren alleine insgesamt 1,44A benötigen. Da nur Linearregler verwendet werden, fällt durch das Display, die Kontroller und die restliche Elektronik genauso viel Strom ab und es wird auch genauso viel Energie verbraucht, wie wenn die gesamte Elektronik mit 12V betrieben werden würde. Da durch diese Linearregler die restlichen 7V verbraucht werden müssen, wird der Linearregler auch entsprechend heiß und muss unter Umständen gekühlt werden. In untenstehender Tabelle wird der Strom der einzelnen Bauteile zusammengerechnet um dadurch die maximale Belastung des 7805 und der Schnittstelle zu berechnen.

Tabelle Auflistung Strom  
Quellen: Datenblätter (siehe Literaturverzeichnis)



Hier wird ersichtlich, dass die Schnittstelle, wenn diese 2 Ampere aushält, bei maximaler Last sehr nahe am Schnittstellenlimit wäre. Dem könnte entgegengewirkt werden, indem mehrere Kontakte für die Versorgung verwendet werden. Da aber bei der Erstellung des Schaltplanes keine zusätzlichen Pins mehr vorhanden waren, die aber dafür erforderlich gewesen wären, und zudem die Verwendung von nur einer Schnittstelle zu komplizierten Leiterbahnrouting führen wird, wurde sicherheitshalber eine andere Möglichkeit vorgezogen.

#### Ein Stecker für das Modell

Hier wird nur für das Fräsenmodell ein einzelner Stecker verwendet. Dies bietet den Vorteil, dass die Elektronik schnell angebracht und wieder abmontiert werden kann, außerdem können auch keine Probleme mit falsch angehängten Steckern auftreten. Es werden aber neben dem Modellstecker noch zwei Stecker benötigt. Ein Stecker für die Stromversorgung und ein weiterer für die Kommunikation mit dem PC (RS232).

#### Entscheidung für einen Stecker für das Modell

Die Entscheidung fiel letztendlich auf den Modellstecker, da hier vergleichsweise der wenigste Aufwand betrieben werden muss, um dennoch so wenig Stecker wie möglich verwenden zu müssen.

## Wahl des Werkzeuges zur Erstellung der Dokumentation

Für die Dokumentation der Diplomarbeit (dieses Dokument) wurden verschiedene Programme in Erwägung gezogen, die alle dafür geeignet wären.

### LaTeX[[14]](#footnote-15)

Dieses Softwarepaket wird für viele technische Dokumentationen, Diplomarbeiten oder Dissertationen an Universitäten insbesondere in den Fachbereichen Mathematik und anderen Naturwissenschaften verwendet[[15]](#footnote-16). Da LaTex auf dem Textsatzsystem TeX basiert, welches nicht mit gewöhnlichen Editoren bearbeitet wird, sondern in einem normalen Texteditor, muss die Formatierung mittels speziellen Tags beschrieben werden. Hierbei können unterschiedliche Abschnitte definiert werden, deren Formatierung extern bestimmt werden kann. Somit müsste während des Schreibens nicht auf Formatierungen, Abbildungsbeschriftungen oder ähnliches geachtet werden, diese werden anschließend für das gesamte Dokument vorgenommen. Hierdurch wird gewährleistet, dass die Formatierungen überall einheitlich sind und Änderungen daran sich auf alle Vorkommnisse erstrecken.

Ein weiterer Vorteil wäre die Plattformunabhängigkeit, welche es sogar auf reinen Textkonsolen ermöglichen würde, die Arbeit fortzusetzen. Da dies aber am Anfang sehr viel Einarbeitung erforderlich gemacht hätte, die vorgelegenen Formatvorlagen nur in odt und doc vorhanden waren und viele Personen mit dem modifizierbaren Quellcode nichts anfangen können (und somit keine Änderungen daran vornehmen könnten) wurde diese Möglichkeit für diese Arbeit abgelehnt.

### LibreOffice[[16]](#footnote-17)

Für LibreOffice würden die Punkte Plattformunabhängigkeit und freie Software sowie die Verwendung des eigentlichen Standardformates odt sprechen. Die Nachteile sind allerdings nicht sehr intuitive Bedienung, sowie einige Stabilitätsprobleme.

Da bei der Erstellung des Pflichtenheftes in LibreOffice, bereits Probleme mit der erhaltenen Formatvorlage auftraten, wurde dieser Office-Suite bei der Erstellung der Diplomarbeit entsagt.

### Microsoft Word 2007

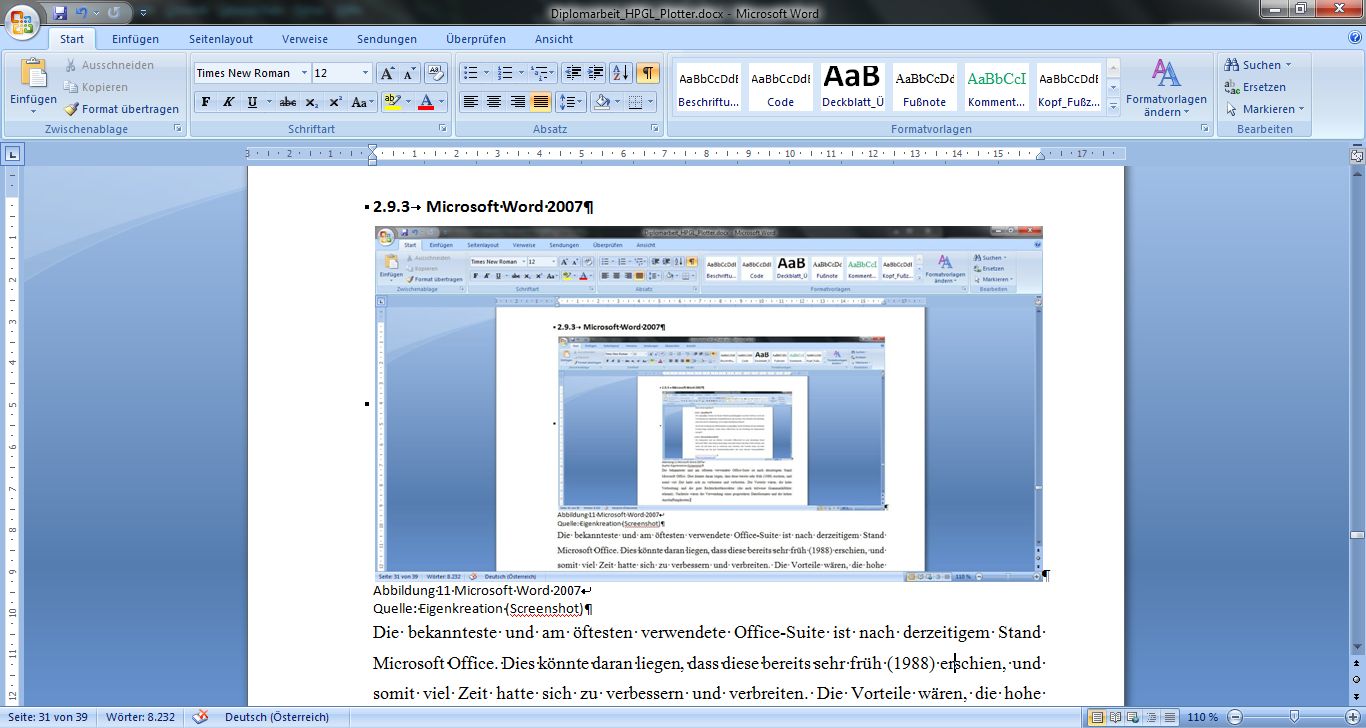


Abbildung Microsoft Word 2007  
Quelle: Eigenkreation (Screenshot)

Die bekannteste und am öftesten verwendete Office-Suite ist nach derzeitigem Stand Microsoft Office. Dies könnte daran liegen, dass diese bereits sehr früh (1988) erschien, und somit viel Zeit hatte sich zu verbessern und verbreiten. Die Vorteile wären, die hohe Verbreitung und die gute Rechtschreibkorrektur (die auch teilweise Grammatikfehler erkennt). Nachteile wären die Verwendung eines proprietären Dateiformates und die hohen Anschaffungskosten.

Da Microsoft Office bereits vorhanden war, und das proprietäre Dateiformat durch die hohe Verbreitung sowieso schon zum Quasi-Standard wurde, fiel der Entschluss dieses Produkt dafür zu verwenden.

### Entscheidung für Microsoft Office 2007

Wegen der hohen Verbreitung und der bei Verfassen des Pflichtenheftes aufgetretenen Probleme, wurde schließlich Microsoft Office 2007 verwendet. Dies war bereits auf dem PC des Entwicklers vorhanden und außerdem war eine Formatvorlage für dieses Dokument verfügbar.

# Ergebnisse

## Schrittmotorensteuerung

Die Schrittmotorenansteuerung wurde zuerst am Steckbrett entwickelt. Sie wurde zuerst ohne Verwendung von Interrupts erstellt, um die Funktionsweise austesten zu können. Anschließend wurde das Programm auf Interrupts (Timer Interrupt) umgestellt, um die Geschwindigkeit exakt regeln zu können. Außerdem wurde dadurch möglich andere Dinge "gleichzeitig" in der Hauptschleife ausführen zu können. Als das alles funktionierte, wurde die Ansteuerung erweitert, um mehrere Schrittmotoren gleichzeitig bewegen zu können. Außerdem wurden Methoden hinzugefügt um die Anzahl der Schritte, die der Schrittmotor in eine bestimmte Richtung ausführen soll, einfacher einstellen zu können. Als dann letztlich alle drei Schrittmotoren verwendet werden konnten, wurde das Testprogramm in eine Programmbibliothek umgeschrieben und in das Hauptprogramm integriert.

Die für den Programmierer verwendbaren Methoden sind:

**stepper\_init(): void**Diese Funktion initialisiert die Schrittmotorenansteuerung: Sie aktiviert Counter0 und den zugehörigen Interrupt TIMER0\_OVF. Zusätzlich werden die verwendeten Pins auf Ausgang gesetzt.

**stepx(int16\_t steps): void**Ändert die zu erledigenden Schritte in die x-Richtung. Negative Werte bewegen den Motor in die entgegengesetzte Richtung.

**stepy(int16\_t steps): void**Ändert die zu erledigenden Schritte in die y-Richtung. Negative Werte bewegen den Motor in die entgegengesetzte Richtung.

**stepz(int16\_t steps): void**Ändert die zu erledigenden Schritte in die z-Richtung(Höhe). Negative Werte bewegen den Motor in die entgegengesetzte Richtung.

Im Hintergrund werden zusätzlich noch folgende, außerhalb der Bibliothek nicht verwendbare, Methoden verwendet:

**ISR(TIMER0\_OVF\_vect)**In dieser ISR werden die Schrittmotoren bewegt. Die Aufrufhäufigkeit hängt von dem Makro STEPS\_PER\_SEC und F\_CPU ab.

**counter\_init(): void**Diese Methode wird von stepper\_init() aufgerufen und initialisiert den Counter durch Setzen des Prescalers (wodurch er gestartet wird) und Aktivieren des TIMER0\_OVF Interrupts.

Die verwendeten Variablen und Konstanten sind von außen nicht zugänglich.

**stepCode[4]: volatile const uint8\_t**Diese Konstante beinhaltet die für die Schrittmotoren notwendige Schrittreihenfolge. Diese lautet wie folgt:

|  |  |  |
| --- | --- | --- |
| Phase1 | Phase2 | Entspricht |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 4 |
| 1 | 0 | 3 |

Tabelle Schrittmotoren-Schrittfolge

Da die Schritte insgesamt nur 8Bit=1Byte benötigen würden, wäre es auch möglich die Schritte in einem einzigen Byte zu speichern. Da dies aber die Verwendung verkompliziert und nur 3 Bytes an Programmspeicher spart wurde diese Entscheidung auf einen späteren Zeitpunkt vertagt.

**curStep[3]: volatile uint8\_t**Beinhaltet die Nummer der aktuell ausgegebenen Schrittfolge für jeden Motor.  
Hier wäre es genauso wie bei stepCode möglich, die Daten (3x2Bit) in einem Byte zu speichern, wurde aber aus selbigen Gründen noch nicht erledigt.

**stepsToDo[3]: volatile int16\_t**Beinhaltet die noch zu erledigenden Schritte für jeden Motor. Negative Werte führen zu einer Bewegung in die entgegengesetzte Richtung (Stop bei 0).  
Diese Variable wird durch die Methoden stepx/stepy/stepz verändert.

Die folgenden Makros wurden definiert:

**STEPPORT,STEPPIN,STEPDDR**Diese drei Makros können zum Ändern des Schrittmotorenports verwendet werden. Es ist dabei aber zu beachten, dass überall der selbe Port verwendet wird, da ansonsten nicht leicht zu findende Fehler auftreten können.  
Zudem kann nur der Port geändert werden. Die Pins bleiben gleich:  
PIN0: Stepper1 Phase 1  
PIN1: Stepper1 Phase 2  
PIN2: Stepper2 Phase1  
PIN3: Stepper2 Phase2  
PIN4: Stepper3 Phase1  
PIN5: Stepper 3 Phase2  
Alle ungenutzten Pins können für andere Zwecke verwendet werden, ohne überschrieben zu werden.

**STEPS\_PER\_REV**Schritte die der Schrittmotor für eine Umdrehung benötigt (alle Schrittmotoren sollten den selben Schrittwinkel haben)

**MM\_PER\_REV**Distanz, die bei einer vollen Drehung zurückgelegt wird (für alle Schrittmotoren gleich)

**STEPS\_PER\_SEC**Schritte, die in einer Sekunde durchgeführt werden können

## RS232-Schnittstelle

Diese Schnittstelle wurde anfangs mittels USB zu Seriell Wandler (mit TTL-Pegel) im Rahmen eines Testprogrammes entwickelt. Zuerst wurde mittels Datenblatt des ATMEGA8-16PU eine Methode erstellt, mit der Daten (blockierend) empfangen werden konnten. Hiermit wurde zuerst eine LED angesteuert, die immer wenn ein Byte empfangen wurde, den Zustand wechselte. Anschließend wurde die dazu komplementäre, sendende Methode geschrieben, die ebenso blockierte. Mithilfe dieser wurde ein "aktiver Repeater" gebaut, der die Daten empfängt und weitersendet. Danach wurden diese Funktionen mithilfe von Interrupts (USART\_RXC,USART\_UDRE) neugeschrieben und nach dem Testen und Korrigieren als Programmbibliothek in das Hauptprogramm integriert.

Die vom Programmierer verwendbaren Funktionen sind:

**USART\_init(): void**Diese Funktion initialisiert den UART Sender und Empfänger durch Setzen der Übertragungsgeschwindigkeit, aktivieren des Senders und Empfängers sowie Setzen des Frame Formats ( 8 Daten- 1 Stoppbit). Zusätzlich wird der Empfangspuffer erstellt.

**USART\_transmit(char data): void**Überträgt ein einzelnes Byte blockierend. Wenn diese Funktion während des Übertragens eines Strings aufgerufen wird, hat das Byte Priorität und wird zuerst/währenddessen gesendet. Falls der Sender deaktiviert ist, wird er für der Sendevorgang aktiviert.

**USART\_transmit\_string(char \*data): void**Speichert einen String in txBuffer und aktiviert den UDRE Interrupt um die Daten dort zu senden. Außerdem wird der Sender aktiviert. Der String muss nullterminiert sein, da ansonsten der Speicher bis zum nächsten 0-Byte übertragen wird. Wenn bereits ein String gesendet wird, wird dieser gelöscht und durch den aktuellen ersetzt.

**USART\_receive():char**Empfängt ein einzelnes Byte blockierend. Bei Übertragungsfehlern wird -1 zurückgegeben.  
Falls der Empfänger deaktiviert war, wird zuerst aktiviert. Wenn der Receiver Interrupt aktiviert ist, kann der Aufruf dieser Methode zu Problemen führen (ewige Blockierung der Hauptschleife, Lieferung falscher Daten).

**USART\_enable\_rx\_interrupt(): void**Aktiviert den USART\_RXC-Interrupt (Receive complete). Dadurch werden Daten empfangen und können durch die Funktion USART\_get\_rx\_buffer() ausgelesen werden.

**USART\_disable\_rx\_interrupt():void**Deaktiviert den USART\_RXC-Interrupt wieder.

**USART\_get\_rx\_buffer(): char\***Gibt eine Kopie des aktuellen Empfangspuffers zurück und löscht ihn. Wenn der Empfangspuffer leer ist, wird 0 zurückgegeben.

Im Hintergrund laufen folgende Funktionen und Interrupt-Service-Routinen (vom restlichen Programm nicht zugreifbar):

**USART\_init\_with\_ubrr(unsigned int ubbr): void**Diese Funktion läuft im Hintergrund von USART\_init() und nimmt als Parameter den zu verwendenden Wert für UBRR (USART Baud Rate Register), der sich wie folgt zusammensetzt: Wobei F\_CPU die Systemfrequenz ist. Ansonsten entspricht die Funktion, der von USART\_init().

**ISR(USART\_UDRE\_vect)**Schreibt die zu sendenden Daten Byte für Byte in das Register UDR, wodurch diese gesendet werden. Wenn alle Daten gesendet wurden, wird der Interrupt deaktiviert und der Sendepuffer freigegeben und auf 0 gesetzt.

**ISR(USART\_RXC\_vect)**Speichert empfangene Daten in den Empfangspuffer. Wenn der Puffer voll wird, wird durch Handshaking signalisiert, dass der Empfänger nicht mehr bereit ist (CTS=0/senden von XOFF) und der Interrupt deaktiviert.

Verwendete Variablen:

**volatile char \*txBuffer**enthält zu übertragenden String, der durch USART\_transmit\_string(data) gesetzt wird. Dieser wird in der ISR(USART\_UDRE\_vect) übertragen und zum Schluss wieder gelöscht.

**volatile char \*txPtr**Pointer auf das nächste zu sendende Byte im txBuffer.

**volatile char \*rxBuffer**Empfangspuffer, der durch die Initialisierungsroutine USART\_init\_with\_ubrr allokiert wird (Länge im Makro RX\_BUFFER\_LENGTH definiert).  
Dieser wird zum Empfang von Strings im Hintergrund über die ISR(USART\_RXC\_vect) verwendet (wird durch USART\_enable\_rx\_interrupt() ausgeführt). Er beinhaltet einen nullterminierten String, der durch die Funktion USART\_get\_rx\_buffer() ausgelesen und wieder gelöscht werden kann.

**volatile char \* rxPtr**Pointer der auf das nächste freie Byte im rxBuffer zeigt. (Nächstes empfangenes Byte landet dort)

Makros:

**BAUD**Zu verwendende BAUD Rate (Standardmäßig 9600)

**UART\_BITS**Anzahl an zu übertragenden Bits pro Frame (muss zwischen 4 und 9 liegen)

**MYUBRR**Berechnete UBRR (USART BAUD Rate Register), wird in USART\_init() verwendet und USART\_init\_with\_ubbr(ubbr) übergeben. Es ist für die BAUD-Rate zuständig

**RX\_BUFFER\_LENGTH**Bestimmt die Größe des Empfangspuffers rxBuffer in Bytes  
Sollte nicht zu groß gewählt werden, um Probleme mit dem in der Größe variablen Stack zu vermeiden.

### Aufgetretene Probleme

Bei der Implementierung des blockenden Empfängers mit LED sind einige Fehler aufgetreten. Zuerst einige bei der Kompilierung, deren Namen sehr missverständlich waren:  
"Garbage at end of line" wegen falsch verwendeter \_\_asm\_\_ Blöcke () und anschließend "Unreferenced Method Call", da einer Funktion anstatt einer Variable eine Konstante übergeben wurde. Nach Ausbessern dieser Fehler musste festgestellt werden, dass der verwendete externe Quarzoszillator defekt war, wodurch auf den internen Oszillator ausgewichen werden musste. Zuerst wurde der interne Oszillator auf 1MHz gestellt, wobei, wie sich später herausstellte, dies der Grund war, dass anstatt der gesendeten Zeichen, ganz andere, nicht ASCII konforme, wieder zurückkamen. Somit musste der Oszillator auf 4MHz umgestellt werden, wodurch die Probleme verschwanden.

### Protokoll (HPGL)

Zur Übertragung der Daten wurde ein Subset der Sprache HPGL gewählt (siehe 2.7.4 Entscheidung für HPGL):

Die zur Implementierung ausgewählten Befehle sind:

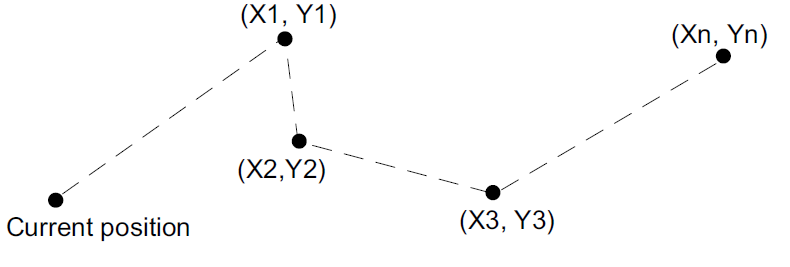
PU [X,Y [,...]][;] Pen Up  
Hebt den Stift an und bewegt sich anschließend zu den Punkten X,Y, ...   
Abhängig ob zuvor PR oder PA verwendet wurde, sind die Parameter relative oder absolute Koordinaten.

Abbildung HPGL Pen Up  
Quelle: http://cstep.luberth.com/HPGL.pdf Seite 10

PD [X,Y [,...]][;] Pen Down  
Setzt den Stift ab und bewegt sich anschließend zu den Punkten X,Y, ...  
Abhängig ob zuvor PR oder PA verwendet wurde, sind die Parameter relative oder absolute Koordinaten

PA [X,Y [,...]][;] Plot Absolute  
Bewegt Stift zu angegebener Position / angegebenen Positionen (Koordinaten absolut zu 0,0).  
Abhängig ob zuvor PU oder PD verwendet wurde, wird entweder gezeichnet oder nicht.

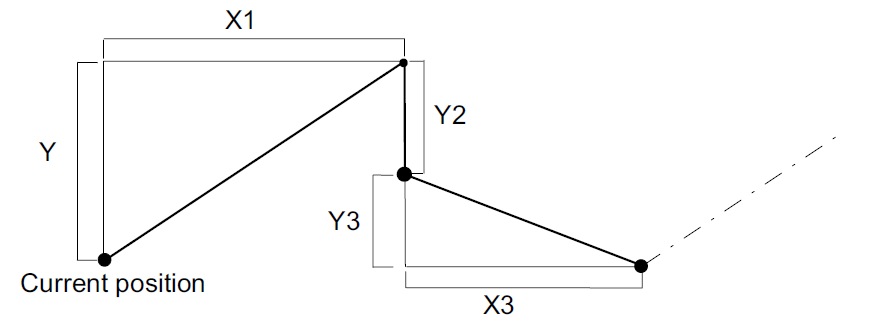
PR [X,Y [,...]][;] Plot Relative  
Bewegt Stift zu angegebenen Positionen (Koordinaten relativ zu letzter Position). Abhängig ob zuvor PU oder PD verwendet wurde, wird gezeichnet oder nicht.

Abbildung HPGL Plot Realtive  
Quelle: http://cstep.luberth.com/HPGL.pdf Seite 13

## Handsteuerungs-Schnittstelle (I²C-Nunchuck)

Die Handsteuerung wurde konzipiert um dem Nutzer eine Möglichkeit zu geben, den Plotter selbst zu steuern. Zusätzlich sollten noch einige Zusatzfunktionen, wie das manuelle Setzen des Nullpunktes oder das Einstellen der Hubhöhe und Position (unterschiedliche Längen der Stifte) implementiert werden. Dies sollte dem Nutzer ermöglichen, den Plotter für seine Bedürfnisse konfigurieren zu können. Diese Funktion konnte aber leider aufgrund Zeitmangels nicht implementiert werden.

## Schaltplan und PCB

Die Entwicklung erfolgte zuerst auf einem Steckbrett. Doch da das Steckbrett seine physikalischen Grenzen hat (Pinanzahl) und die Verbauung eines solchen sehr unprofessionell wirkt, fehleranfällig ist (Kabel können leicht herausgenommen und umgesteckt werden, ...) und auf Dauer teuer kommt, wurde eine gefräste Platine verwendet. Für diesen Zweck musste zuerst ein Schaltplan digital erstellt werden. Dies wurde mit Target 3001! (siehe 2.8.1.3) erledigt, da hiermit anschließend auch das Platinen-Layout erstellt werden konnte, welches im Anschluss an den Schaltplan designt wurde. Im Rahmen des Schaltplanes mussten einige Schnittstellen definiert werden, die weiter unten beschrieben werden.

### Schnittstelle zum Modell

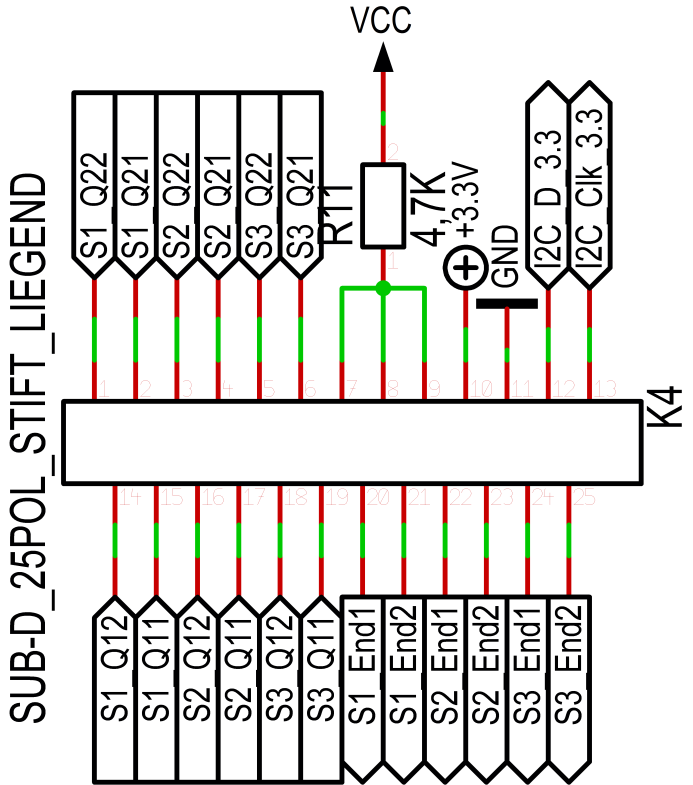


Abbildung Schnittstelle zum Modell

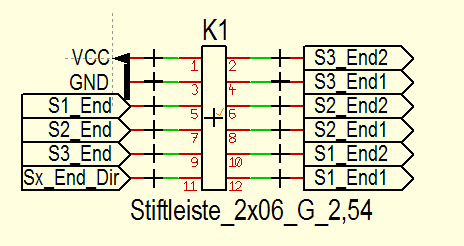
Quelle: Target 3001! Entwurf

Die Kontakte mit einer Bezeichnung in der Form Sx\_Qij werden an die Schrittmotoren angeschlossen. Wobei x die Nummer des Schrittmotors ist, i die Nummer der Spule und j die Anschlussnummer des Kontaktes der Spule. Wenn möglich sollten alle Schrittmotoren gleich angeschlossen werden, wobei bei einer Bewegung in die falsche Richtung die Spulen (und eventuell auch die Anschlüsse) vertauscht werden müssen.

Die Kontakte mit Bezeichnungen in der Form Sx\_Endi sind für die Endschalter (Mikrotaster mit 3 Kontakten (c(common),n.c (normally closed), n.o.(normally open)) gedacht, welche mit Common an diesen Pin angeschlossen werden sollten. N.o. sollte an einem der Kontakte 7,8 oder 9 liegen, die über einen Widerstand an Vcc liegen. N.c. wird an GND angeschlossen.   
Sx bezeichnet den betreffenden Schrittmotor und i die Seite, an der der Endanschlag montiert werden sollte.

Die Kontakte +3V, I²C\_D\_3.3 und I²C\_Clk\_3.3 wären für den Nunchuck gedacht, der nun nicht verbaut wird. Da diese Schnittstelle aber ein I²C- Bus ist, könnten hier auch viele andere Geräte, wie beispielsweise ein Display angeschlossen werden. Da der Nunchuck aber nur 3.3V unterstützt wurde diese Schnittstelle darauf ausgelegt, wodurch nur Geräte die damit zurecht kommen verwendet werden können.

### Schnittstelle zum Endschalterkonverter



Der Endschalterkonverter, der die Signale der Endschalter aufbereitet, erhält vom Modell die Eingänge Sx\_Endi, die weiter oben bereits beschrieben wurden. Diese werden wie folgt verodert:

# Abbildungsverzeichnis

[Abbildung 1 Blockschaltbild 2 Kontroller 5](#_Toc388502416)

[Abbildung 2 Gameport Anschlüsse Quelle: http://www.alphalanding.com/rc-track/wp-content/uploads/2009/11/joystick\_port.png 6](#_Toc388502417)

[Abbildung 3 Wii Nunchuck Quelle: http://www.robotshop.com/media/files/images/design/inex-zx-nunchuk.jpg 7](#_Toc388502418)

[Abbildung 4 Pinout HD44780 kompatibles Display Quelle: http://cdn.instructables.com/F2H/QPNE/HJKC4C4G/F2HQPNEHJKC4C4G.LARGE.jpg 8](#_Toc388502419)

[Abbildung 5 Arduino Uno Quelle: http://arduino.cc/en/uploads/Main/ArduinoUno\_R3\_Front.jpg 11](#_Toc388502420)

[Abbildung 6 Arduino Nano Quelle: http://arduino.cc/en/uploads/Main/ArduinoNanoFront\_3\_lg.jpg 12](#_Toc388502421)

[Abbildung 7 Schrittmotorenarten Quelle: https://upload.wikimedia.org/wikipedia/commons/4/48/Stepmotscheme.jpg 17](#_Toc388502422)

[Abbildung 8 Verwendung L297 mit L298 Quelle: [ST\_AN470] Seite 14 Figure 21 18](#_Toc388502423)

[Abbildung 9 Beispiel für Verwendung von TCA3727 Quelle: [TCA3727]: Figure 5 19](#_Toc388502424)

[Abbildung 10 gschem Quelle: http://vishnuwhitestone.files.wordpress.com/2011/08/gschem.png 21](#_Toc388502425)

[Abbildung 11 Microsoft Word 2007 Quelle: Eigenkreation (Screenshot) 25](#_Toc388502426)

[Abbildung 12 HPGL Pen Up Quelle: http://cstep.luberth.com/HPGL.pdf Seite 10 33](file:///C:\Users\Florian\Documents\Diplomarbeit\Diplomarbeit_HPGL_Plotter.docx#_Toc388502427)

[Abbildung 13 HPGL Plot Realtive Quelle: http://cstep.luberth.com/HPGL.pdf Seite 13 34](#_Toc388502428)

[Abbildung 14 Schnittstelle zum Modell 34](#_Toc388502429)

[Abbildung 15 Schaltplan HPGL-Plotter Hauptplatine Quelle: Target 3001! 6-14](file:///C:\Users\Florian\Documents\Diplomarbeit\Diplomarbeit_HPGL_Plotter.docx#_Toc388502430)

[Abbildung 16 Platinenlayout Hauptplatine HPGL-Plotter Quelle: Target 3001! 6-15](#_Toc388502431)

[Abbildung 17 Schaltplan HPGL-Plotter Endkonverter Aufsteckplatine Quelle: Target 3001 6-15](#_Toc388502432)

[Abbildung 18 Platinenlayout Aufsteckplatine HPGL-Plotter Quelle: Target 3001! 6-16](#_Toc388502433)

# Literaturverzeichnis

[AN470]SGS-Thomson Microelectronics. (1995). Application Note 470: The Stepper Motor Controller. *AN470/0392* . SGS-Thomson Microelectronics.

[TCA3727]Siemens -- Semiconductor Group. (01. 02 1998). 2-Phase Stepper-Motor Driver TCA 3727. *Datasheet TCA 3727* .

[MAX232]Texas Instruments. (02 1989). MAX232, MAX232I - Dual EIA-232 Drivers/Receivers. *Datasheet MAX232* .

ATMEGA8-16PU Datasheet: Atmel ATMEGA8-16PU

# Anhang

### Projektdokumentation

#### Pflichtenheft

#### Terminplan

#### Projekttagebuch

Di 20.08.13 5h

* In Mikrocontrollerprogrammierung eingelesen
* HelloWorld geschrieben
* Schaltplan für Programmieradapter für ATTiny25 gesucht (Internet)

Mi 21.08.13 5h

* Programmieradapter auf Basis des HTL-Traun Programmierers erstellt und zusammengebaut

Di 27.08.13 6h

* Für ATTiny in Assemblerprogrammierung eingelesen
  + HelloWorld in Assembler erstellt
  + HelloWorld auf interruptgesteuert (Timer) umgestellt

Mi 28.08.13 8h

* Zu Testzwecken 7 Segment Ansteuerung via Timer erstellt

Mo 01.09.13 10h

* Jn Schrittmotorenansteuerung eingelesen (L297)
  + Library dafür zu Testzwecken geschrieben  
    --> Testen wegen nicht vorhandenem Schrittmotor nicht möglich
  + Nach günstigen Schrittmotoren gesucht --> Alle gefundenen preislich >20€
  + Zuhause nach Schrittmotoren gesucht --> 4 schrittmotorenartige Anzeigen gefunden

--> Steuerung fehlt (Elektronik)

--> Testen bis Schulanfang aufgeschoben um festzustellen ob Betreuungslehrer andere Ansteuerung im Sinn hat

Di 02.09.13 2h

* Überlegungen über Projekt (Ablauf, Umfang, Realisierung)
  + Blockschaltbild erstellt

Mi 03.09.13 5h

* detailierte Informationen über RS-232, Schrittmotoren, Nintendo Wii Nunchuck im Internet gesucht
* Informationen über Arduino gesucht

Do 04.09.13 8h

* Projektauftrag erstellt

So 15.09.13 2h

* Pflichtenheft begonnen
* Informationen über Nunchuck gesammelt

So 22.09.13 3h

* Pflichtenheft fortgesetzt

So 29.09.13 5h

* Pflichtenheft erweitert

Sa 26.10.13 6h

* Pflichtenheft

So 27.10.13 6h

* Pflichtenheft fertiggestellt --> Lehrer zeigen

Sa 30.11.13 3h

* Pflichtenheft für Abgabe fertiggemacht und letzte Fehler ausgebessert

So 08.12.13 2h

* In schon vorhandenen Schrittmotoren-Testcode eingelesen
* Funktion zur Bestimmung der erfolgten Drehungen -> Zurückgelegter Weg hinzugefügt
* Festgestellt, dass Ausgabe zu Debugging- und Testzwecken notwendig wird

--> **Interaktion mit serieller Schnittstelle vorzuziehen**

--> Auf wichtige Grundsatzfrage gestoßen: Sourcecode= Assembler mit C oder C mit Assembler

* Offene Punkte Liste begonnen

Sa 16.12.13 4h

* Dokumentation zum Thema UART/USART (in Datasheet ATMEGA8-16PU) gelesen, Bitraten ausgesucht und Register detailiert aufgelistet
* Simple Library geschrieben (Polling)

So 17.12.13 2h

* Library auf Interrupts umgestellt
  + In Interrupts eingelesen
  + Schaltung überlegt (Informationen über verwendeten Quarzoszillator benötigt!)

--> Schaltung noch gemeinsam mit Testprogramm zu testen

So 20. 12. 13 2h

* Universal-Programmieradapter zusammengebaut

Do 26.12.13 2h

* Zuletzt erhaltenes Oszilloskop getestet
  + Keine Daten erhalten

--> Informationen über Abtasttheorem/Aliasing sammeln

Sa 28.12.13 2h

* Informationen über Abtasttheorem gesammelt
* Oszilloskop erweitert getestet
  + Probleme mit Stromversorgung von Breadboard

--> Programmieradapter-Kabelenden zu versteifen

Mo 13.01.14 2h

* Programmieradapter auf ATMEGA8 gelötet
* Informationen über notwendige Fuses gesammelt und niedergeschrieben

Di 14.01.14 2h

* Programmierer lauffähig gebracht (Treiberprobleme nach Neuaufsetzen)
* Fuses geflasht
* Auf Fehler in Testprogramm (UART) aufmerksam geworden

--> Error in xy.s:Garbage at end of line x2

Do 16.01.14 2h

* Fehler ausgebessert (\_asm\_ Block-Schreibweise falsch)
  + neuer Fehler (Unreferenced Method Call x3 in main)
    - Lösung im Internet gesucht --> Strings müssen explizit in Flash abgelegt werden und in den RAM geladen werden (Harvard-Architektur mit Tannenbaum-Programmiersprache)
    - abgeändert, Fehler tritt trotzdem auf
    - Änderungen rückgängig gemacht und const char\* durch char\* ersetzt
  + Fehler verschwunden --> Kompilierung möglich

--> Test notwendig

Sa. 18.01.14 6h

* Treiber für USB-Seriell-Adapter gesucht
  + in Kernel hineinkompiliert und anschließend an µC angeschlossen
* Testprogramm (UART) geflasht und auf Rückmeldung an PC gewartet
  + selbst nach mehrmaligem Reset und Reflash keine Rückmeldung erhalten
* Testprogramm um blinkende LED auf PB1 erweitert
  + LED blinkt nicht --> LED getestet (OK)
* Testprogramm nur auf blinkende LED reduziert
  + LED blinkt noch immer nicht
* Verstärkerschaltung für LED gebaut
  + noch immer keine Reaktion
* Fuses gecheckt und neu geflasht
  + lFuse auf default zurückgeflasht --> LED funktioniert
  + lFuse erneut geflasht um Oszillator zu verwenden
    - Nichts geht mehr (auch per Programmierer kann nicht mehr zugegriffen werden)
* Oszillator umgesteckt --> funktioniert solange Finger auf Oszillator bleibt
* Fuses wieder auf default zurückgeflasht
* Beine des Oszillators mit Lötzinn verstärkt um Wackelkontakte zu verhindern
* Fuses wieder für Oszillator umgeflasht --> selber Fehler wie vorhin

--> Vermutung, dass Oszillator fehlerhaft ist

Workaround: Interner Oszillator oder Quarz

* Quarz mit geeignerter Frequenz für UART (9600 Baud) gesucht
* Teilungsfaktoren für alle vorhandenen Quarze berechnet
* 4MHz Quarz hat am besten geeigneten Teilungsfaktor (ganzzahlig)

--> wird verwendet werden

So. 19.01.14 6h

* Geeignete Fuses gesucht
  + keine geeigneten Kondensatoren zuhause

--> internen Oszillator verwenden

* Testprogramm wieder erweitert und Funktionen korrigiert
  + getestet --> gibt komische Zeichen zurück
    - versucht auszubessern, hat jedoch nicht funktioniert
      * BAUD-Raten-Einstellung geändert
      * Bit/Frame auf 7 gestellt, ...
* Funktionen um Daten einzulesen auf Ausgang geleitet um internen Speicher (Strings im Speicher) auszuschließen
* Oszillator auf 4 MHz umgestellt
  + Funktioniert
* Einlesen der Daten versucht mittels Interrupts zu lösen
  + Puffer erstellt und ISR geschrieben
    - getestet --> hat nicht funktioniert

Mo 20.01.14 6h

* Änderungen rückgängig gemacht
  + hat nicht funktioniert
  + Code teilweise neu geschrieben
    - getestet --> funktioniert
* Programmcode in Versionsverwaltungssystem GitHub geladen und Konfigurationsdaten erstellt
  + Client heruntergeladen (CLI) und darin eingelesen
* Mit Zeitleistenumbesserungen begonnen

--> Problem: Gant-Project funktioniert nicht wie gewollt

--> Umstieg auf andere Software notwendig (zB MS Project, Excel)

Di 21.01.14 4h

* Versucht MS Project lauffähig zu bringen

--> Gescheitert, Umstieg auf Excel notwendig

Do 23.01.14 4h

* Zeitplan in Excel erstellt
  + Neue Meilensteine überlegt
  + Weiteren Ablauf neu geplant
  + Mann-Monats-Gebirge integriert
* Meilensteine auch auf GitHub geladen um bessere Übersicht zu haben
  + Meilensteine zusätzliche Unterpunkte zugewiesen um Fortschritt verfolgen zu können

--> TODO: Meilenstein-Trend-Analyse

Mi 29.01.14 4h

* UART-Testprogramm erweitert, dass es interruptgesteuert funktioniert

--> Problem: Nach gewisser Anzahl eingegebener Zeichen lässt sich nichts mehr übertragen (Programm läuft aber weiter)

* + Grund Gefunden: Buffer Overflow durch zu lange Wartezeit vor Ausgabe/Leerung des Puffers --> Lösung: Wartezeit in Main von 100ms auf 10ms verringert.

--> Nachhaltige Lösung: Hardware-Handshaking

Di 04.02.14 2h

* Hauptprojekt zusammengefügt
  + UART-Test in Library verwandelt und hinzugefügt
  + Struktur mit Headerfiles und C-Files erstellt
  + Zeilen/Befehlsenden von HPGL durchgesehen um FlowControl und Parser implementieren zu können

--> Flow Control und Parser noch zu implementieren

* Stepping Testprogramm durchgesehen
  + Informationen über Counter gesammelt

Do 06.02.14 5h

* Makefile für Hauptprogramm geschrieben
  + Fehler wegen falsch gesetzter Includes
  + Fehlerbehebung/Beschreibung für Kompilierung mehrerer Sourcedaten in eine ausführbare Datei gesucht
  + Makefile und Headerfiles abgeändert, auf dass die Includes stimmen und alle C-Files kompiliert werden können
* Implementierung von Flow Control (Software) begonnen
  + Abgebrochen, da bis morgen die Stepper-Library abgegeben werden soll
* Stepper Library/Stepper Testprogramm abgeändert
  + Counter hinzugefügt
  + Variablen um Schritte zu zählen hinzugefügt

--> Programm um 2 Schrittmotoren erweitern --> Problem: nur 1 Testboard

--> Platine zu entwerfen

* Unfertiges Testprogramm in StepperLibrary (Main/Stepper.[hc]) migriert

--> Funktionsnamen sind noch zu ändern

So 09.02.14 5h

* Main lauffähig gebracht
* Begonnen PCB zu entwerfen
  + Begonnen Schaltplan zu zeichnen
    - Informationen über Pegelwandler 5V-12V gesammelt (RS232)

--> Bei Schule nachfragen welche Wandler vorrätig sind

Mo 10.02.14 2h

* Schaltplan fortgesetzt
  + Informationen über I²C gesammelt
  + Informationen über 5V<->3.3V Bidirektionalwandler gesammelt
    - Verwendung 2er MOSFETs (SDA,SCL) möglich

Di 11.02.14 4h

* Schaltplan erweitert
  + Informationen über HW-Handshaking, RS-232-Pinbelegung gesammelt
  + "Kabelbaum" gezeichnet
  + Informationen über verwendbare Stecksysteme und deren Kosten eingeholt
  + Pinbelegung HTL-Traun Programmieradapter gesucht
* Mit PCB begonnen
  + Gehäuse-Standardgrößen + Kosten gesucht

Mi 12.02.14 2h

* Schaltplan und PCB erweitert
  + Informationen über I²C-Pullup-Widerstände gesammelt
  + Informationen über erlaubte I²C-Bus-Kapazitäten gesammelt
  + I²C Widerstände und Lötjumper (Verbinden von Schaltung mit Programmierschnittstelle) hinzugefügt

Di 18.02.14 6h

* PCB abgeändert

--> Beim Speichern führt Programmabsturz zum vollständigen Verlust des Projektes

* Neuzeichnen des Schaltplanes
* Neubeginn PCB

Mi 19.02.14 5h

* Nach SOT-23 MOSFET gesucht, der von der Eingangskapazität her passt
  + LTSpice verwendet um geeignete Eingangskapazitäten zu identifizieren
  + Auf Reichelt gesucht --> Nichts gefunden
  + Auf RS-Components gesucht --> gefunden

--> Problem: Nur Bestellungen von mind. 50 Stk möglich

So 23.02.14 3h

* Unterlagen für bevorstehende Präsentation zusammengestellt

So 02.03.14 5h

* Schaltplan erweitert
* PCB fortgesetzt

Mo 03.03.14 5h

* PCB umstrukturiert

Do 06.03.14 1h

* **Äußerung wegen Nichtfertigstellung der Diplomarbeit gegenüber Direktorin/Jahrgangsvorstand und Betreuungslehrer**

**--> Umfang kann nicht verringert werden**

Fr. 07.03.14 4h

* Fortsetzung PCB

--> Problem: Target 3001! erlaubt nur 250 Pins (Freeware Beschränkung)

* + Unbenötigte Teile einsparen

Sa 08.03.14 6h

* PCB um TCA3727 erweitert
* Schaltplan um 37-poligen Centronics-Stecker erweitert

So 09.03.14 4h

* PCB fortgesetzt
* MAX232 zu Schaltplan hinzugefügt

Mo 10.03.14 3h

* PCB fortgesetzt
* Bei Lehrer erkundigt ob MAX232 verwendet werden kann/ob er lagernd ist
* MAX232-Anwendung im Datenblatt angesehen und in Schaltplan integriert

Di 11.03.14 6h

* Elektronik zum Verringern der benötigten Leitungen für die Endanschläge designt

--> Problem 5 ORs benötigt --> Zuviele Pins auf PCB (Target Beschränkung)

* + Nach Einsparungsmaßnahmen (Pins) gesucht --> Pull-Ups in µC hineinrationalisiert

--> Reicht nicht aus

Do 13.03.14 6h

* 37 poligen Stecker in 25 poligen D-Sub (Modell) + 9 polige D-Sub (RS232) + DC-Jack abgeändert
* Stecker verbunden (Schaltplandesign)
* Fehler (Abstand) gesucht (PCB-Design) und Korrektur begonnen

Fr 14.03.14 3h

* Fehler (PCB-Design) korrigiert

Sa 15.03.14 8h

* Fehler korrigiert
* Stecker fertig verbunden

So 16.03.14 8h

* Fehlerkorrektur abgeschlossen
* Schnittstelle zum Endanschlagsverarbeiter (Pinsparmaßnahme) definiert und eingefügt (Schaltplan und PCB)
* Fehler gesucht --> keine vorhanden
* Schaltplan des Endanschlagsverarbeiters designt
* PCB angefertigt

--> Problem: nicht daran gedacht, dass PCB auf den Kopf gestellt montiert werden muss --> Schnittstelle müsste gespiegelt werden

--> Korrekturmöglichkeit durch SMD-Montage des Steckers => Montage nicht kopfüber

* + PCB um designt
* Abgemessen ob Gehäuse an vorgesehener Stelle Platz findet --> Nein

--> Korrekturmöglichkeit: Kondensator auf Hauptplatine einsparen (Mit Lehrer zu besprechen)

Mo 17.03.14 7h

* PCB nochmals durchgesehen
* Neues PCB für Aufsteckplatine designt (THT)
* Zu Lehrer gegangen

--> Lehrer auf Schikurs => Kommt ganze Woche nicht mehr

* Liste an benötigten (zu bestellenden) Teilen geschrieben
  + Mit der Teilesuche im Internet begonnen (Bestellung)

Di 18.03.14 4h

* Account auf RS-Components-Seite erstellt
* Bestellung fortgesetzt
* Mit Diplomarbeitsdokumentation begonnen

Mi 19.03.14 5h

* Vergleich zwischen RS-Components und Reichelt (Bestellung)

-->Problem, da nicht die gleichen Teile auf beiden Seiten vorhanden sind

Di 25.03.14 2h

* Abgabe des PCBs

--> Probleme mit Wahl der Widerstände (Zu klein)

--> Probleme mit Auren der Bauteile/Leiterbahnen (Falsche Auren gewählt)

* Mit Korrektur begonnen

Mi 26.03.14 2h

* Korrektur der Fehler durchgeführt

Do 27.03.14 1h

* Abgabe des PCBs

Sa 29.03.14 2h

* DA-Doku: Unterlagen zusammengesucht
* Code-Dokumentation verbessert

Di 01.04.14 2h

* Nachgesehen ob PCB fertig --> Nein
* Bestellung fortgesetzt

Do 03.04.14 5h

* Fertiges PCB erhalten
* Teile zusammengesucht

--> Nicht alle Teile vorhanden

* Beginn von Zusammenlöten

Fr 04.04.14 5h

* Nicht vorhandene Teile bei RS-Components zusammengesucht
* Löten fortgesetzt
  + Einlöten des Quarzes (DIL 8 in DIL 16-Bohrung)
    - Pinbelegungen vergleichen --> OK

Sa 05.04.14 8h

* Lötarbeiten fertiggestellt (soweit die Teile vorhanden waren)
* Bestellung bei RS-Components (Teilesuche) fertiggestellt

Mo 07.04.14 4h

* Testen der Platine
* Fehlerkorrektur (Patzer mit Lötzinn usw)

Di 08.04.14 5h

* Testen der Platine
  + Spannungsregler regelt nicht richtig und wird heiß: 16V in --> 12V out
  + Quarz wird heiß !!
* Fehlersuche: Lötstellen kontrolliert, Leiterbahnen auf Kontakt geprüft, ...
  + Gefundene Fehlerquellen eliminiert

Mi 09.04.14 5h

* Fehler gesucht und eliminiert --> Spannungsregler regelt wieder richtig (wird dennoch heiß)
* Quarz wird trotzdem heiß und funktioniert nicht --> wahrscheinlich kaputt
* Ausgebaute Bauteile (ICs) wieder eingesteckt

Do 10.04.14 2h

* Mit Lehrer über Inhalt der Dokumentation gesprochen
* Lehrer die Platine gezeigt --> neuen Quarz (DIL16) + Sockel erhalten

Sa 12.04.14 8h

* Fortsetzung/Neubeginn DA-Doku:
  + Einleitung: Zusammenfassung, Projektumfeld begonnen

So. 13.04.14 6h

* DA-Doku
  + Einleitung: Zusammenfassung fertiggestellt
  + Abstract geschrieben

Mo 14.04.14 6h

* DA-Doku
  + Einleitung: Projektumfeld fertiggestellt
  + Hauptteil begonnen
* Bestellung der notwendigen, fehlenden Teile bei RS-Components

Di 15.04.14 8h

* DA-Doku:
  + Hauptteil
    - Entwicklungsumgebung fertiggestellt
    - Mikrocontroller begonnen
* Anruf wegen Problemen mit Bestellung --> Privatpersonen können nicht bestellen

Mi 16.04.14 8h

* DA-Doku:
  + Hauptteil:
    - Mikrocontroller fertiggestellt
    - Programmiersprache fertiggestellt

Do 17.04.14 7h

* DA-Doku:
  + Hauptteil:
    - Entwicklungsumgebung fertiggestellt
    - Schrittmotorentreiber fertiggestellt
    - Einleitung erweitert: Motivation geschrieben

Fr 18.04.14 6h

* DA-Doku:
  + Hauptteil
    - Erstellen des Blockschaltbildes
      * Wahl der Handsteuerung
      * Wahl des Displays
    - Bilder eingefügt

Sa 19.04.14 9h

* DA-Doku:
  + Hauptteil
    - Erstellen des Blockschaltbildes
      * Anzahl der zu verwendenden Mikrocontroller
    - Übertragung der Befehle

Di 22.04.14 5h

* DA-Doku
  + Generelle Überarbeitung

Mi 23.04.14 6h

* DA-Doku
  + Hauptteil:
    - Wahl des Werkzeuges zur Erstellung der Dokumentation
  + Generelle Überarbeitungen
  + Erstellen eines Kurzschlussdiagramms
  + Beginn der Schnittstellendokumentation (Modell)

Do 24.04.14 2h

* Treffen mit Lehrer wegen Dokumentation (Abgabe von rev00)

Fr: 25.04.14 3h

* DA-Doku
  + Geringfügige Änderungen
  + Anhang: Projekttagebuch angefangen
* Abgabe DA-Doku (Zwischenabgabe per Mail) rev01

Di. 29.04.14 4h

* Treffen mit Rechnungsführerin wegen Bestellung bei RS-Components

--> Bestellung nächsten Dienstag möglich

* DA-Doku

Mi 30.04.14 3h

* DA-Doku

Do 1.05.14 5h

* DA-Doku

Mi 07.05.14 8h

* PCB lauffähig gebracht
* Ändern von Leiterbahnen notwendig um µC flashen zu können

Do 08.05.14 8h

* Mail wegen Zwischenabgabe gelesen
* DA-Doku

Fr 09.05.14 7h

* DA-Doku (nicht fertiggebracht)
* Gespräch mit Direktorin und Betreuungslehrer über Herbsttermin und Projektwoche

Di 13.05.14 1h

* Gespräch mit Direktorin und Betreuungslehrer über Diplomarbeit (mit Klassensprecher+Stellvertreter)

Mi 14.05.14 1h

* Gespräch mit Direktorin über Diplomarbeit (mit Klassensprecher+Schulsprecher)

Do 15.05.14 1h

* Gespräch mit früherem Jahrgangsvorstand/Direktor über Diplomarbeit

Fr 16.05.14 1h

* Gespräch mit Betreuungslehrer über Diplomarbeit

Di 20.05.14 3h

* DA-Doku

Mi 21.05.14 13h

* Lötarbeiten bei PCB
* DA-Doku:
  + Technische Dokumentation
  + Tagesberichte in elektronische Form gebracht
  + Hauptteil erweitert

Do 22.05.14 8h

* DA-Doku
  + Tagesberichte in elektronische Form gebracht
  + Arbeit fertiggestellt, 2x gedruckt und gebunden

#### Schaltpläne und PCBs

Abbildung Schaltplan HPGL-Plotter Hauptplatine  
Quelle: Target 3001!

##### Hauptplatine

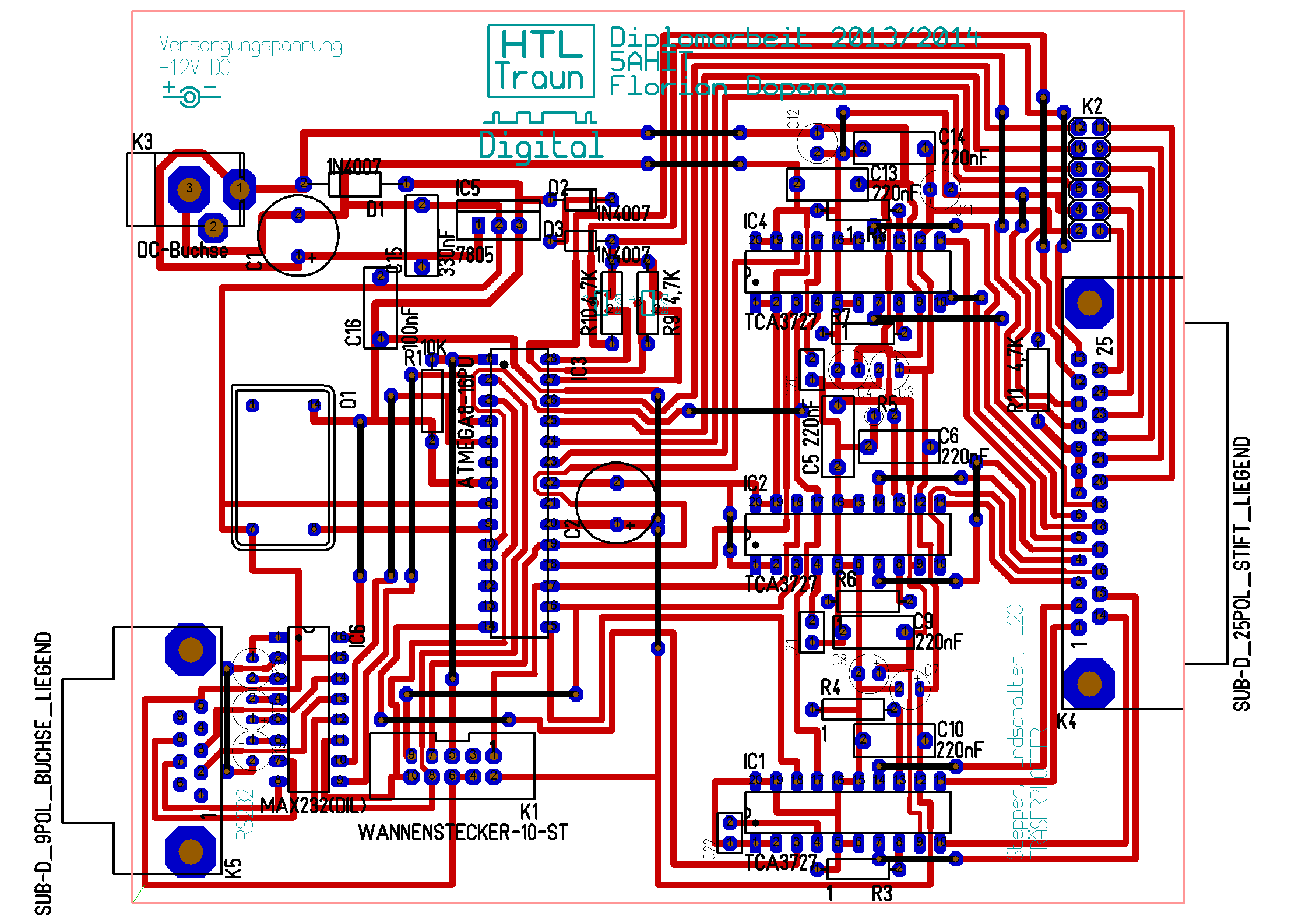


Abbildung Platinenlayout Hauptplatine HPGL-Plotter  
Quelle: Target 3001!

##### Aufsteckplatine (Endkonverter)

Abbildung Schaltplan HPGL-Plotter Endkonverter Aufsteckplatine  
Quelle: Target 3001

##### 

Abbildung Platinenlayout Aufsteckplatine HPGL-Plotter  
Quelle: Target 3001!

1. TODO [↑](#footnote-ref-2)
2. Vgl. Datasheet ATMEGA8-16PU (1kB SRAM) [↑](#footnote-ref-3)
3. Siehe Datasheet ATMEGA8-16PU [↑](#footnote-ref-4)
4. zB. <http://www.mikrocontroller.net> [↑](#footnote-ref-5)
5. <http://forums.creative.com/creativelabs/board/message?board.id=Vista&message.id=4732&view=by_date_ascending&page=2> [↑](#footnote-ref-6)
6. Vgl. Datasheet ATMEGA8-16PU: 8kB Flash, 512Byte EEPROM, 1 kB SRAM [↑](#footnote-ref-7)
7. TODO (vll) Verweis auf technische Doku [↑](#footnote-ref-8)
8. <http://arduino.cc/en/Main/Products> [↑](#footnote-ref-9)
9. Datenblatt Siehe Datasheet ATMEGA8-16PU [↑](#footnote-ref-10)
10. Datenblatt Siehe Datasheet ATMEGA8-16PU [↑](#footnote-ref-11)
11. <http://www.avrfreaks.net/index.php?module=Freaks%20Tools&func=viewItem&item_id=172> [↑](#footnote-ref-12)
12. <http://www.atmel.com/tools/atmelstudio.aspx> [↑](#footnote-ref-13)
13. <http://sourceforge.net/projects/kontrollerlab/> [↑](#footnote-ref-14)
14. <http://www.latex-project.org/> [↑](#footnote-ref-15)
15. [https://de.wikipedia.org/wiki/LaTeX#Kein\_WYSIWYG](https://de.wikipedia.org/wiki/LaTeX%23Kein_WYSIWYG) [↑](#footnote-ref-16)
16. <https://www.libreoffice.org/> [↑](#footnote-ref-17)