Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

D I P L O M A R B E I T

Linear-Positioniereinheit mit PCB-Wicklung und Niedervolt-Pulsumrichter

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs

unter Leitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Johann Ertl

am Institut für
Elektrische Antriebe und Maschinen
E372

eingereicht an der Technischen Universität Wien
Fakukltät für Elektrotechnik

von

Robert Schwab
Matrikelnummer: 9626523
Am Berg 5, A-6833 Klaus

Wien, im September 2008
Datum Unterschrift

Linear-Positioniereinheit mit PCB-Wicklung und Niedervolt-Pulsumrichter

Zusammenfassung

In der vorliegenden Arbeit wird der Entwurf und Bau einer linearen Positioniereinheit beschrieben. Ein besonderes Merkmal dabei ist, dass die Wicklungsspulen des Permanentmagnet-Linearmotors über entsprechende Ausgestaltung der Leiterplatte realisiert sind, auf welcher auch sämtliche leistungselektronischen Bauteile des speisenden Umrichters angeordnet sind. Durch das Wegfallen der konventionellen Kupferdrahtwicklungen und der einfacheren Montage ergeben sich deutliche Kostenvorteile für die Serienfertigung.

Die Aufgabenstellung umfasst den gesamten mechanischen und elektronischen Aufbau eines Labormusters inclusive der Implementierung der notwendigen Positionsmessung sowie einer volldigitalen Regelung unter Verwendung eines Signalprozessor-Evaluationsboards.

Linear Positioning System Based on PCB-Coil Winding and Low-Voltage PWM Converter

Abstract

The presented diploma thesis describes the design and realisation of a linear positioning unit based on a permanent magnet linear motor arrangement. A specific characteristic of the system is that the colis of the linear motor are formed by the printed circuit board which is used also for wiring the converter's power electronic components. By avoiding conventional coil-windings and due to the much simpler manufacturing process, substantial cost reductions are expected for mass production.

The project definition involves the complete desing of the mechanical part as well as of the electrical part of a laboratory prototype. This includes also the required optical position measurement system and further the design of a fully digital control unit based on the application of a commercial digital signal processor evaluation board.

Danksagung
Zunächst möchte ich mich bei meinem Betreuer Herrn Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Johann Ertl für die Unterstützung in technischer und organisatorischer Hinsicht bedanken. Nicht vergessen sei auch das Verständnis für meine familiäre Situation und die daraus folgenden Umständen.

Herzlichen Dank auch an die vielen anderen Lehrenden der Technischen Universität Wien, an deren Wissens- und Erfahrungsreichtum ich teilhaben durfte.

Ein besonderes Danke an meine Frau Christiane und meine Söhne Gregor und Marius für ihre Unterstützung, Verständnis und die vielen Verzichte.

Nicht zu vergessen seien auch meinen Eltern, die mir das Studium der Elektrotechnik erst ermöglichten und mich mit ihrem Rückhalt und ihrer Unterstützung beistanden.

Nicht zu vergessen auch mein Cousin Stefan für die technologische Unterstützung beim Fertigen der Mechanik. Herrn Norbert möchte für die Unterstützung bei der Korrektur der Diplomarbeit danken.

Robert Schwab

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Aufgabenstellung

1.1 Zieldefinition

Die Aufgabenstellung besteht darin, eine lineare Positioniereinheit für geringe Lasten, wie sie oft in Druckern, Plottern, Scannern, etc. vorkommen, zu erstellen. Die Positionierung soll, im Gegensatz zu den heute sehr verbreiteten Zahnriemenantrieben mit Schrittmotoren, direkt mit einem Linearmotor erfolgen. Der Kernpunkt der Arbeit soll ein „mitfahrender“ Stromrichter sein, der auf gewickelte Spulen verzichtet und statt dessen die Spulen mittels Leiterbahnen direkt auf der Elektronikplatine realisiert werden. Dies könnte durch weniger Komponenten und vereinfachte Montage Kostenvorteile bei der Produktion bringen.

Das Vermeiden gewickelter Spulen macht es erforderlich, mit weniger Windungen mehr Strom und geringen Spannungen zu arbeiten, um eine ausreichend große Kraftwirkung zu erreichen. Durch das breite Spektrum an leistungselektronischen Bauelementen, mit denen auch Computer-Prozessoren mit wenigen Volt und zig Ampere versorgt werden, sollte dies technisch realisierbar sein. Die zu entwickelnde Leistungsendstufe ist zusammen mit den Spulen auf einer Printplatte montiert und fährt deshalb mit dem Rotor mit. Der ansteuernde Prozessor wird wegen des großen Layout-Aufwandes nicht integriert. Hier wird ein fertiges Evaluationsboard verwendet.

Im Umfang der Aufgabe enthalten ist weiters die Entwicklung und Implementierung einer dazugehörigen Positionsmessung und eines einfachen Regelkreises zur Positionierung.
Ziel dieser Arbeit ist nicht ein fertiges Produkt, sondern ein Funktionsprototyp, um die Realisierbarkeit von Antrieben mit Wicklungen auf der Printplatte zu untersuchen und dabei vorhandene Probleme aufzuzeigen.

Die Kriterien der Arbeit in einem groben Lastenheft zusammengefasst:

Maximale Fahrgeschwindigkeit	1 m/s
Dimensionen	30cm Fahrweg
Transportgewicht	Eigenmasse (max ca. 100g)
Leistungsansteuerung	„mitfahrend“
Ansteuerung	DSP-Evaluationsboard
Ansteuerungssoftware	verwendung bestehender DSP-Bibliotheken
Kommunikation	PC mittels RS232-Schnittstelle
Mechanischer Aufau	Eigenkonstruktion
Positionsmessystem	Eigenkonstruktion
Positionsauflösung	0,1mm

Tabelle 1: Lastenheft für die Konstruktion

1.2 Analyse der Anforderungen

Die Aufgabe ist die lineare Bewegung und Positionierung eines massenbehafteten Schlittens.

Dies setzt voraus:

- lineare Führung in 5 Freiheitsgraden
- regelbare positive und negative Kraftwirkung auf die Masse in Bewegungsrichtung
- Positionsbestimmung des Schlittens
- Regelkreis zur Positionierung mittels Kraftregelung

Bedingt durch die newtonsche Trägheit lässt sich die benötigte Kraft zur Positionierung mit einer maximalen Geschwindigkeit von $1m/s$ über $30cm$ berechnen. Aus dem Lastenheft in Tabelle 1 und unter Annahme der maximalen Masse des Schlittens von 0.1kg bei konstanter Beschleunigung ergibt sich der in Abb. 1 dargestellte dreiecksförmige Geschwindigkeitsverlauf.

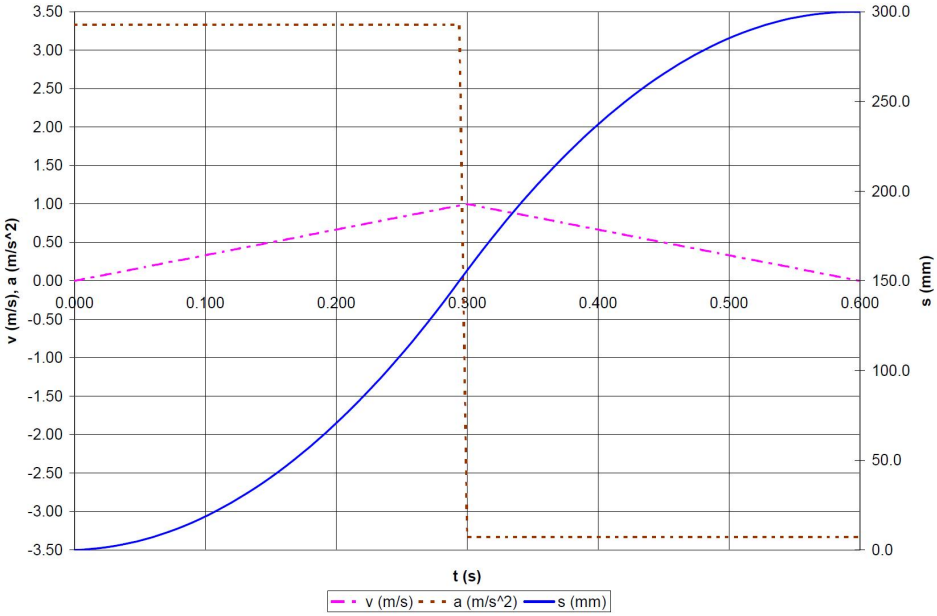


Abbildung 1: Geschwindigkeitsverlauf

Die dafür notwendige Kraft lässt sich bestimmen durch:

$$F = m \cdot a = m\frac{v_{max}}{t/2} = m \cdot v_{max}\frac{v_{max}}{l} = m\frac{v_{max}^2}{l} = 0.1kg\frac{1m^2}{0.3ms^2} = 0,33N \qquad (1)$$

Um dabei nicht berücksichtigten Verlusten durch Reibung und andere Einflüsse Rechnung zu tragen, erfolgt die weitere Auslegung auf das Dreifache dieser Kraft:

$$F = 3 \cdot 0,33N \approx 1N \tag{2}$$

Die für die Bewegung benötigte durchschnittliche Leistung beträgt:

$$P = F \cdot \overline{v} = 1N\frac{1m/s}{2} = 0.5W \tag{3}$$

Um diese Kraftwirkung zu erreichen, werden im Folgenden verschiedenste Antriebsmöglichkeiten auf Basis von Linearantrieben verglichen und die am besten geeignete realisiert.

1.3 Vorgehensweise und Struktur

Aus obiger Aufgabenstellung ist ersichtlich, dass die Aufgabe in verschiedene, nach dem Entwurf des Gesamtkonzepts in größtenteils unabhängige Teilaufgaben zerlegt werden kann. Dies ermöglicht es, jeden einzelnen Teilbereich als Modul in Betrieb zu nehmen, bevor das Gesamtsystem getestet wird. Das Zusammenwirken der einzelnen Teilbereiche wird in Abbildung 2 als Blockschaltbild dargestellt.

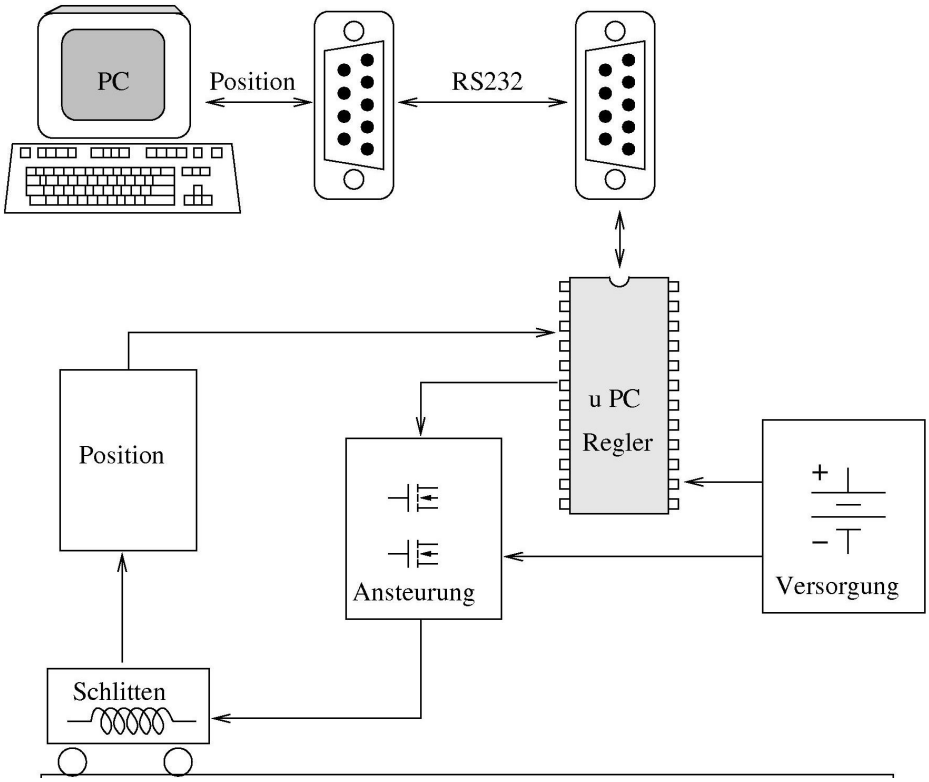


Abbildung 2: Struktur der Positionierungseinheit

Zu lösende Teilaufgaben:

- Bestimmung der Struktur und des Aufbaus des Linearantriebs
- Mechanischer Aufbaus des Linearantriebs
- Entwurf der Spulen und der zugehörigen Leistungsansteuerung
- Bau der Leistungsansteuerung mit den Spulen
- Auswahl des μ-Controllers und Implementierung von Ansteuerung und Regler
- Inbetriebnahme und Tuning der Stromregelung
- Entwurf und Bau einer geeigneten Positionsmessung
- Modellierung der sich ergebenden Regelstrecke und Reglerentwurf
- Inbetriebnahme und Tuning der Positionsregelung
- Evaluation des Gesamtkonzepts am gebauten Modell

Dieses Vorgehen spiegelt sich in der Struktur der Kapitel und deren Gliederung in dieser Arbeit wieder.

2 Mechanischer Aufbau

2.1 Typen von Linearmotoren

Linearmotoren können als eine abgewickelte Version eines Elektromotors dargestellt werden. Somit können sie auch in ähnliche Klassen wie die Rotationsmaschinen unterteilt werden. Jeder Typ bringt für die verschiedensten Anwendungsfälle gewisse Vor- und Nachteile. Sie sind in Abb. 3 zusammenfassend dargestellt.

- **Synchronmotor-Prinzip**
  Diese Motoren erzeugen in ihren Wicklungen ein wanderndes Wechselfeld, welches durch ein zweites stehendes Magnetfeld zu einer Kraftwirkung führt, sodass sich dieses synchron mitbewegt. Das stehende Feld kann fremd- oder permanentmagneterregt sein. Die Ausführung mit Permanentmagneten bietet den Vorteil, dass der rotierende Teil des Motors nicht mit Strom versorgt werden muss und somit keine Bürsten, Schleifkontakte oder Schleppkabel nötig sind. Um das Feld synchron zu halten und richtig zu kommutieren, muss der Ansteuerungslogik die Position relativ zu den Permanentmagneten bekannt sein. Hierfür werden oft Hall-Sensoren eingesetzt. Dieses Prinzip wird bei hochdynamischen Linearantrieben verwendet und auch um Pneumatikzylinder zu ersetzen.

- **Asynchronmotor-Prinzip**
  Diese Motoren erzeugen das nötige Gegenfeld durch Induktion der Primärspulen in die kurzgeschlossenen Sekundärspulen. Diese bewegen sich, um einen lastabhängigen „Schlupf“ versetzt, mit. Der Sekundärteil muss dabei nicht als konventionelle Wicklung ausgeführt sein. Es genügt auch eine leitfähige Metallplatte als Kurzschlusswicklung. Dies ermöglicht etwa den berührungslosen Transport von beliebigen leitfähigen Gegenständen auf einem solchen Motor. Ein solcher Motor kann sich aber auch berührungslos, z.B. auf einem langen Stahlträger, fortbewegen. Die Ansteuerung kann hierbei unabhängig von der Position erfolgen. Dieses Prinzip findet Verwendung bei Förderanlagen, Walzwerken, Stranggießanlagen, Hallenkränen, etc.

- **Gleichstrommotor-Prinzip**
  Diese Motoren ähneln vom Aufbau her den Syncromotoren. Das Magnetfeld wird hier aber über Schleifkontakte (Bürsten) kommutiert. Diese Bauart ist bei Linearmotoren sehr selten zu finden, da der Verschleiß der Bürsten und die offenen spannungsführenden Kontakte nachteilig sind.

- **Schrittmotor-Prinzip**
  Diese Motoren funktionieren nach dem Reluktanzprinzip. Über einem Eisenkamm befinden sich versetzte Spulen, die bei Stromfluss sich in die Position des kürzesten magnetischen Rückschlusses bewegen. Somit bewegt er sich durch abwechslungsweises Ansteuern der Spulen je um einen Schritt weiter. Durch Stromfluss in mehreren Spulen sind auch Zwischenpositionen möglich (Teilschrittbetrieb, Microschrittbetrieb). Wenn man Schrittverluste durch richtige Auslegung und Vermeiden von Überlast verhindert, lässt sich die Position durch Zählen der Ansteuerungsimpulse vorgeben. Somit kann eine Positionsmessung entfallen. Die

Positioniergenauigkeit wird durch die mechanische Schrittweite bestimmt. Durch Microschrittbetrieb kann diese noch verfeinert werden. Um hohe Geschwindigkeiten bei kleinen Schrittweiten zu erreichen, sind hohe Frequenzen des Ansteuerstromes nötig. Um diese schnellen Stromanstiege in den induktiven Wicklungen zu ermöglichen ist meist auch eine hohe Zwischenkreisspannung nötig. Vorteilhaft ist das hohe Start-Drehmoment und ein Rastmoment bei den Vollschritt-Stellungen. Lineare Schrittmotoren werden dann oft eingesetzt, wenn auf die teure Positionsmessung verzichtet werden soll.

- **Voice-Coil-Prinzip**
  Bei diesem bewegt sich eine Spule linear in einem magnetischen Gleichfeld. Die Linearität der Kraft zum Strom ermöglicht hierbei eine einfache Regelung. Schwierig ist jedoch der Rückschluss des Feldes bei größeren Längen durch die magnetische Sättigung im Rückschlussmaterial. Eingesetzt wird diese Art des Antriebes z.B. zum Antrieb von Lautsprechermembranen, Zeigerinstrumenten und diversen Kleinantrieben.

- **Sonderformen**
  Daneben gibt es noch eine Reihe recht exotischer Antriebskonzepte, wovon ich einige der Vollständigkeit halber erwähnen, aber nicht näher darauf eingehen möchte. Z.B.: Piezo-Antriebe (InchWorm), elektrostatische Antriebe, Unipolarmaschine, Transversalflussmaschinen, hydraulische (hydrostatische) Antriebe, Ionenstrahl-Prinzip, etc.

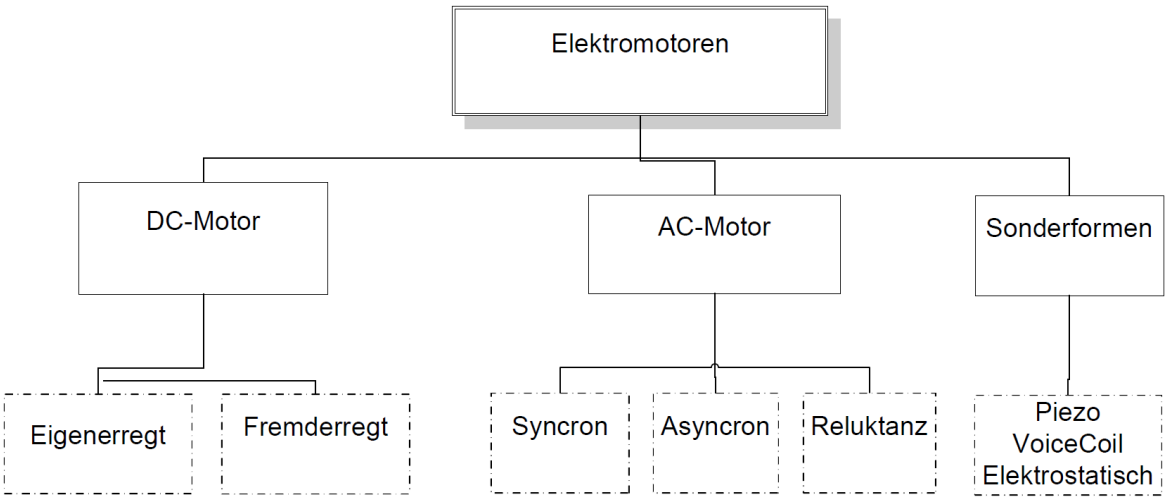


Abbildung 3: Zusammenstellung der verschieden Arten von Antrieben

Ein allgemeiner Nachteil aller Linearantriebe ist, dass sich immer nur ein Teil des Stators oder Aktors zur Krafterzeugung nutzen lässt. Bei Rotationsantrieben ist immer der ganze Umfang im Eingriff. Dies lässt sich beim Linearantrieb nur durch Unterteilung in verschiede Sektionen, die einzeln geschaltet werden, vermeiden. Es entstehen auch Probleme durch ungünstige Feldverhältnisse am Anfang und Ende des Feldes,

welche bei einer Rotationsmaschine mit geschlossenem Drehfeld nicht auftreten.

Eine besondere Klasse von Linearmotoren stellen jene dar, welche das Magnetfeld auch zur Führung nach dem Prinzip von Magnetlagern nützen, um die Reibungsverluste zu minimieren. Das bekannteste Beispiel eines solchen Antriebs, der meist als 'Maglev' bezeichnet wird, ist der Hochgeschwindigkeitszug Transrapid in Deutschland. Bei manchen Konzepten werden hier auch gekühlte Spulen aus Supraleitern eingesetzt um die Wirkungsgrade zu verbessern.

2.2 Auswahl des Antriebsprinzips

Folgende Erwägungen wurden getroffen:
Gegen eine Voice-Coil Lösung sprechen die geforderten Dimensionen. Wie eine Simulation des Magnetfeldes in Bild 4 zeigt, ist es, bedingt durch die Eisensättigung, nicht möglich ein konstantes unipolares Magnetfeld akzeptabler Stärke über eine Länge von 30cm aufrecht zu erhalten.

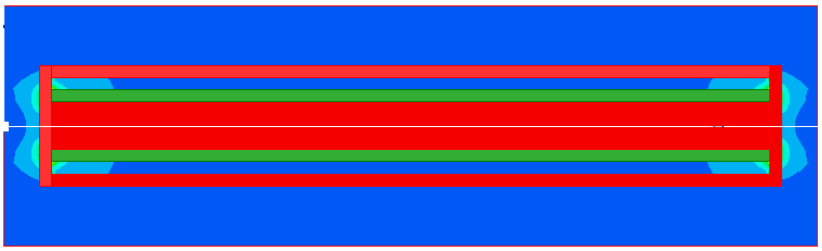
Die Ausführung als Schrittmotor lässt sich mit der geforderten Dynamik und Schrittgenauigkeit aus geometrischen Überlegungen nicht ohne diskrete Wicklungen realisieren. Die dafür nötigen Verzahnungen des sich bewegenden Teiles sind mit einer Printplattenwicklung konstruktiv nicht zu vereinbaren. Dies wird deshalb außer Acht gelassen.

Das asynchrone Prinzip wurde zugunsten des synchronen fallen gelassen, da sich durch den Einsatz von Permanentmagneten das Gegenfeld und somit die Kraftwirkung deutlich erhöhen lässt. Bedingt durch die „kleinen“ Spulen auf der Printplatte ist dies ein wichtiger Faktor.

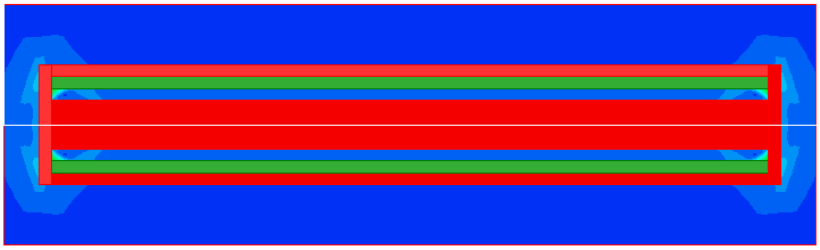
Von den Sonderformen wäre die Piezo-Technik sehr interessant. Dies geht jedoch in den Bereich Micro-Mechanik und ist deshalb schwer in einem Laboraufbau am Institut auszuführen.

Es wurde versucht parasitäre Effekte zu finden, welche sich zur gleichzeitigen magnetischen Führung ausnützen lassen. Es fand sich jedoch kein Ansatz hierzu, der sich einfach realisieren ließe. Wie in verschiedenster Literatur [14][13] gezeigt , benötigt ein solcher Aufbau einen zweiten Regelkreis und weitere Spulen zur Lagerung. Da dies nicht das vorrangige Ziel dieser Arbeit ist, wurde darauf verzichtet und eine mechanische Führung realisiert.
Um nun die magnetischen Verhältnisse besser abschätzen zu können, wurde eine Finite-Elemente-Simulation durchgeführt. Als Software hierfür wurde das am Institut vorhanden MAXWELL[2] verwendet, welches die dreidimensionale Simulation von Magnetfeldern erlaubt. Die dabei erhaltenen Daten des Feldstärkenverlaufs im Luftspalt können für die Abschätzung der erreichbaren Kräfte und das Streckenmodell zum Entwurf des Regelkreises verwendet werden.
Aus obigen Überlegungen und der Simulation der dabei entstehenden magnetischen Verhältnisse fiel die Wahl eindeutig auf einen permanenterregten Synchronmotor. Wie

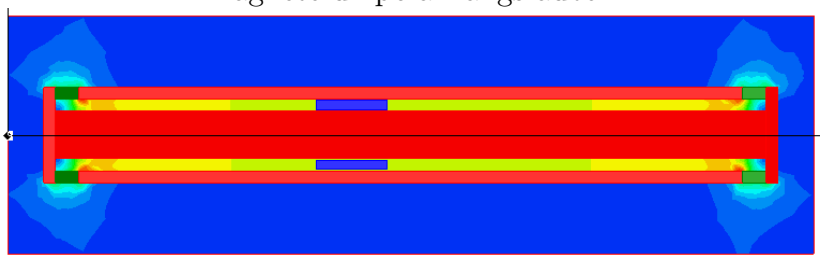
in Bild 4 ersichtlich, lässt sich damit das stärkste magnetische Feld erzeugen. Durch die kleinen Dimensionen der Spulen ist dies ein sehr wichtiger Faktor, um genügend Kraft für die geforderte Dynamik erzeugen zu können. Dieses Konzept wird nun genauer ausgearbeitet.



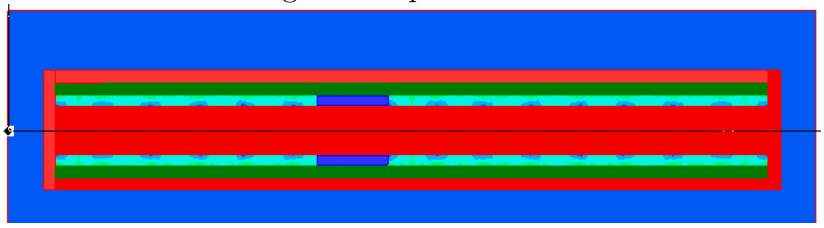
Magnete unipolar längs innen



Magnete unipolar längs außen



Magnete unipolar seitlich



Magnete abwechlungsweise außen

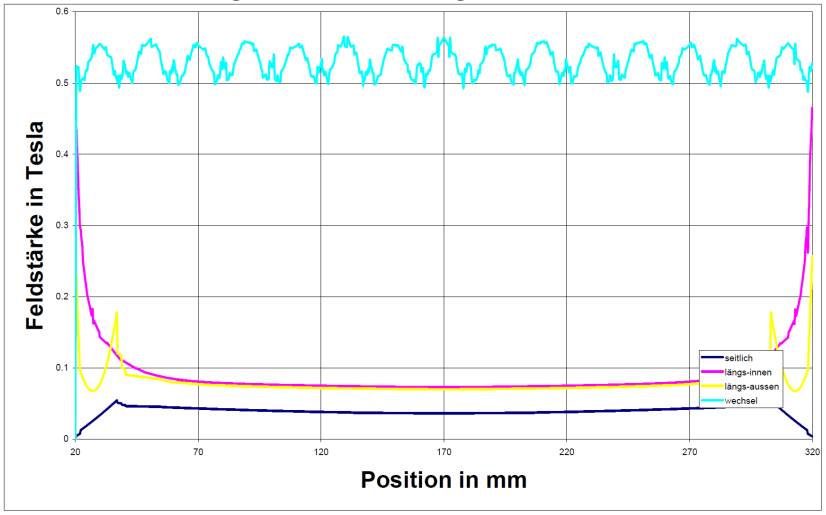


Abbildung 4: Feldvergleich bei verschiedenen Magnetanordnungen

2.3 Konstruktionsansatz

Dieser Linearmotor hat Wicklungen, die auf eine Platine geätzt wurden. Es gibt zwei prinzipielle Möglichkeiten der Anordnung:

- Die Spule wird als Wagen und der permanentmagnetbestückte Teil feststehend ausgeführt. Durch das geringere Gewicht der Platine gegenüber den Magneten mit dem Eisenrückschluss lässt sich hier eine höhere Dynamik erwarten. Nachteilig ist jedoch, dass der bewegliche Teil über Schleifkontakte oder Schleppkabel mit Energie versorgt werden muss.

- Es wird eine lange Platine mit Wicklungen und ein kurzer mit wenigen Magneten bestückter Wagen verwendet. Dies ergibt einen geringeren Magnetmaterialaufwand und der bewegliche Teil muss nicht mit Energie versorgt werden. Störend ist, dass immer die gesamte Länge der Spule stromdurchflossen ist, obwohl nur ein kleiner Teil davon im Einsatz ist und zur Kraftwirkung beiträgt. Dies senkt den Wirkungsgrad und führt zu unerwünschten Streufeldern und EMV-Problemen. Dies lässt sich vermeiden, indem viele Schaltelemente verwendet werden um immer nur den Teil der Spulen zu aktivieren, der gerade benötig wird. Das bedeutet aber einen sehr hohen Schaltungs- und Bauteilaufwand.

Aus obigen Überlegungen wird das Konzept eines mit Permanentmagneten bestückten feststehenden Eisenstators und einer sich bewegenden Platine mit darauf befindlichen Wicklungen, festgelegt. Quaderförmige Permanentmagnete mit den Maßen $6 \times 20 \times 1,6$mm waren am Institut vorhanden, und da diese von den Dimensionen her den Vorstellungen entsprachen, wurden diese verwendet.
Die Platine mit den Wicklungen trägt auch gleichzeitig die Leistungselektronik und es werden nur die Versorgung sowie die PWM- und die Strommess-Signale zugeführt. Die Ansteuerung erfolgt über ein Evaluationsboard des Prozessors, welches zu groß und schwer ist um mitbewegt zu werden.

Daraus ergibt sich folgender konkreter Konstruktionsansatz:

- massives Stahlgestell mit beidseitig aufgeklebten Permanentmagneten
- Permanentmagnete mit den Abmessungen $6 \times 20 \times 1.6$mm werden in 3mm Abstand aufgeklebt. Dies ergibt eine Periodenlänge des Feldes von 18mm.
- Platine mit Wicklungen, die sich dazwischen bewegt
- Die Platine beinhaltet auch die Leistungselektronik, nicht jedoch den Prozessor
- extern montierte Positionsmesseinheit

2.4 Konstruktive Details und Zeichnungen

Da es sich um einen Funktionsprototyp handelt, wird nicht eine kompakte, vollintegrierte Lösung angestrebt. Um den Versuchsaufbau möglichst flexibel für die im Laufe der Entwicklung nötigen Änderungen zu gestalten, wird die Konstruktion bewusst eher groß und robust gehalten. Dies erleichtert auch die Fertigung. Es sollen auch zusätzliche Probleme, die durch Verbiegung, zu geringe Stabilität o.ä. auftreten könnten von Anfang an vermieden werden. Funktionell gesehen sind die Eisenteile für den vorhandenen magentischen Fluss weit überdimensioniert.

Der Eisen-Aufbau besteht aus einer massiven Grundplatte, auf die der Länge nach zwei Winkeleisen aufgeschraubt sind. Durch zwei Stellschrauben an den Rändern lässt sich der Luftspalt ihnen feinjustieren. Auf den Winkeln sind auf den einander zugewandten Flächen die Magnete in abwechslungsweiser Polung aufgeklebt. Es wurden die oben erwähnten quaderförmigen Magnete mit $20 \times 6 \times 1,6$mm verwendet, da diese Dimension gut passt und sie einfach verfügbar waren. Siehe Bild 5 und Bild 6.
Die Spulen werden als Leiterbahnen direkt auf der Elektro-Platine gefertigt, welche sich zwischen den Winkeln mit den Magneten bewegt. Hierauf wird in Kapitel 3.2 noch genauer eingegangen werden.

Die lineare Führung des Schlittens wird durch eine V-förmige Einfräsung an den Seiten der Winkel realisiert. Darin gleiten vier Kunststoff-Stellschrauben am Schlitten, die so justiert werden, dass sich dieser ohne viel Spiel aber leichtgängig hin- und herbewegen lässt.
Seitlich am Schlitten wird auch die Positions-Messeinheit befestigt, auf die in Kapitel 5 genauer eingegangen wird.

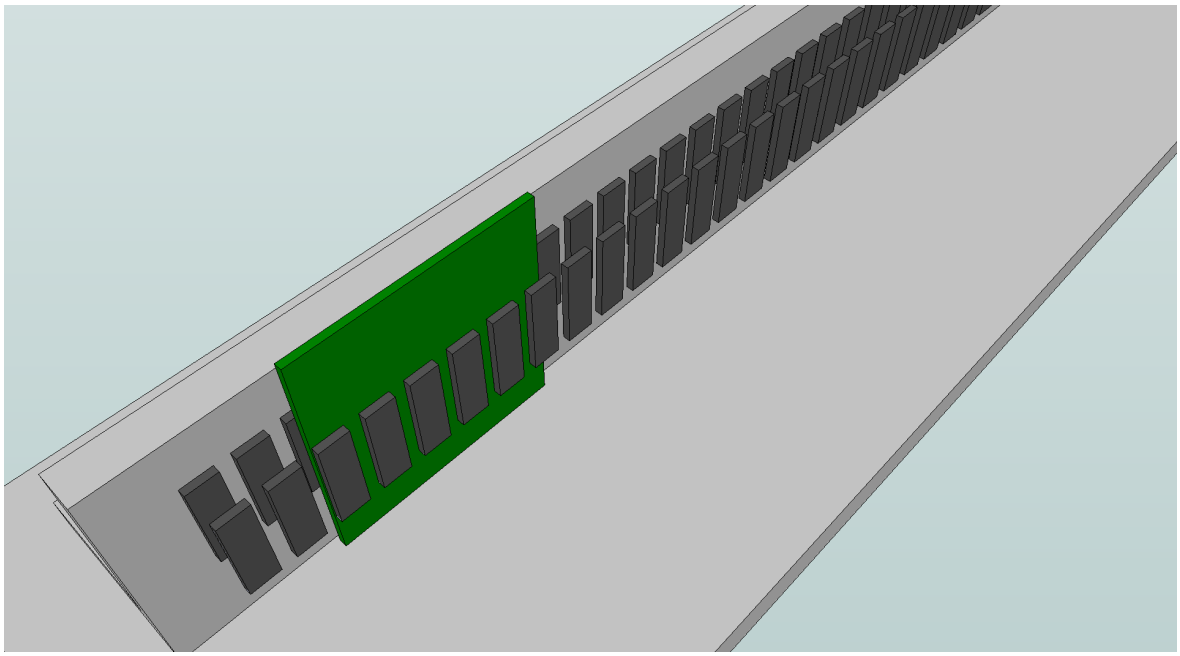


Abbildung 5: Positionierung der Magnete

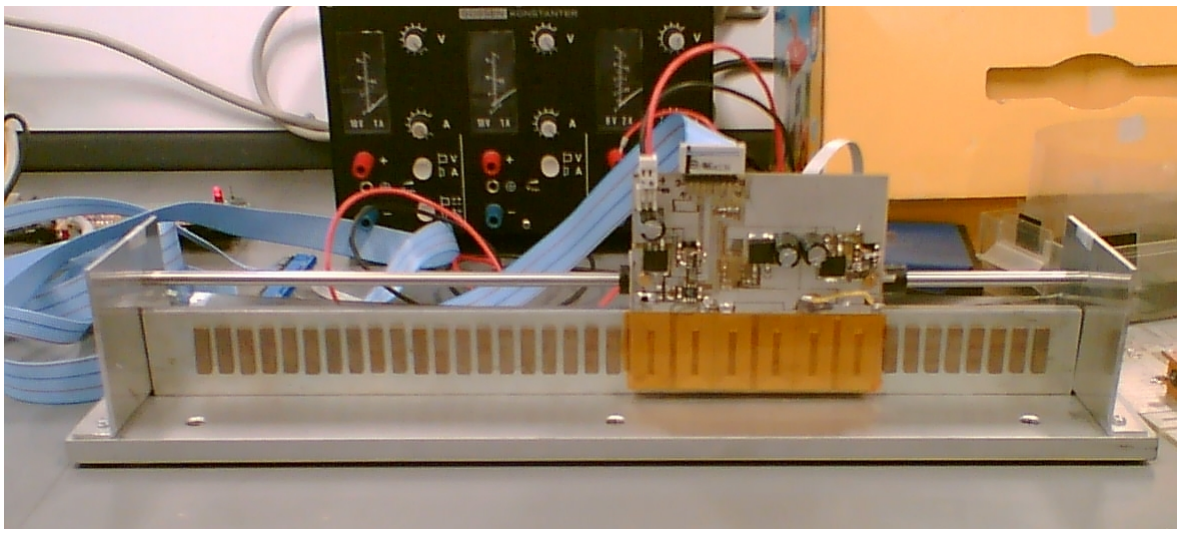


Abbildung 6: Foto des Versuchsaufbaus

Die Dimension der Magnete und die Entscheidung über die Konstruktion aus Winkel- und Flacheisen steht fest. Somit bleibt als wichtigster variabler Parameter der Mechanik noch der Abstand zwischen den Magneten. Ein sinusförmiger Feldverlauf kombiniert mit sinusförmigen Strömen in den Spulen ermöglicht einen sehr glatten Kraftverlauf. Diese Form der Sinus-Kommutierung ist derzeit Stand der Technik und daher sind auch viele Literaturquellen, Software- und Simulationsbeispiele verfügbar. Daher wird versucht ein Feld zu erhalten, das diesem Ideal möglichst nahe kommt.
Der in der Literatur [20],[16] beschriebene Ansatz zur Lösung wurde durch Simulation in Maxwell verifiziert, um auch die zu erwartende Stärke des resultierenden Magnetfeldes abschätzen zu können.

Die Ergebnisse der Simulation lassen sich wie folgt zusammenfassen:

- Bei einem Magnetabstand, welcher der Hälfte der Magnetbreite entspricht, erhält man in sehr guter Näherung ein Sinus-Feld.
- Bei Vergrößerung des Abstandes erhält man eine immer stärkere Abflachung im Bereich des Nulldurchganges.
- Bei Verkleinerung erhält man steilere Flanken und somit eine Abweichung von der Sinusform hin zur Trapezform.
- Es ist bei diesem Aufbau eine Feldstärke von ca 0.7T im Luftspalt zu erwarten.

Auf ein Nachvollziehen der Berechnung wird hier unter Hinweis auf das Literaturverzeichnis verzichtet. Die Ergebnisse lassen sich auf obige Kernaussagen zusammenfassen, welche durch die Simulationsergebnisse in Bild 7 veranschaulicht werden. Durch richtige Schrägung und Ausgestaltung der Magnete könnte dieser Verlauf nochmals verbessert werden. Für diese Arbeit ist dies jedoch nicht nötig und es wird darauf verzichtet.

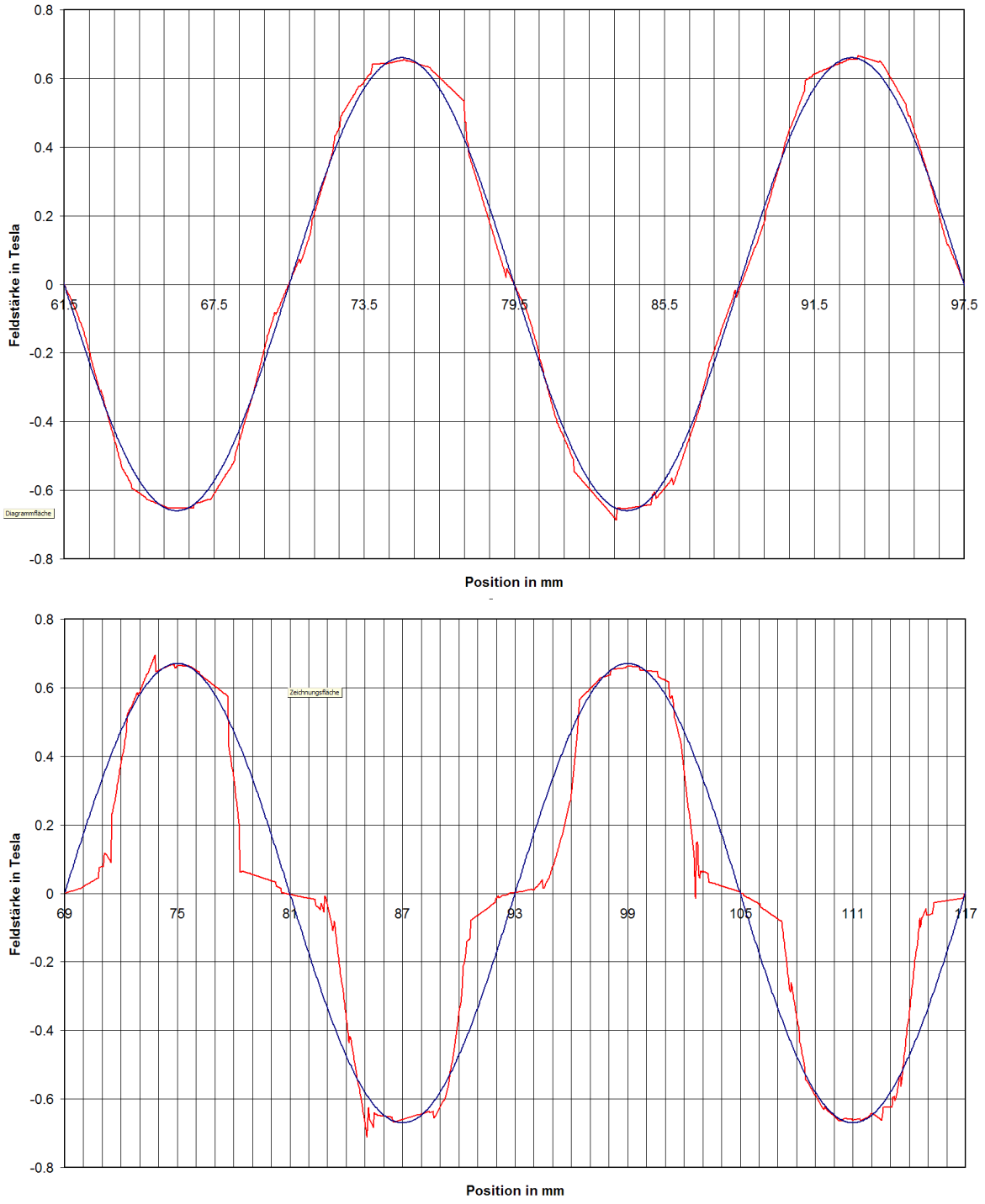


Abbildung 7: Feld in Abhängigkeit vom Magnetabstand bei 3mm und 6mm

Ein alternativer Ansatz wäre, den Abstand gleich der Magnetbreite zu wählen und die Kommutierung der Spulen im Block-Modus mittels Rechtecksignalen durchzuführen. Nachteilig hierbei wäre ein schlechter Gleichlauf und die dadurch schwierigere Regelbarkeit bei geringen Positionsauflösungen. Von Vorteil wäre die im flachen Nulldurchgang des Feldes, und somit im kraftfreien Bereich stattfindende Kommutie-

rung. Durch die Abweichung von der Sinus-Kommutierung könnte dann auch nicht auf Standard-Bibliotheken für die Koordinatentransformation zurückgegriffen werden.

Abschließend noch die 3D-Ansicht des Entwurfs der kompletten mechanischen Konstruktion in Bild 8. Konstruktive Detailzeichnungen finden sich im Anhang C.

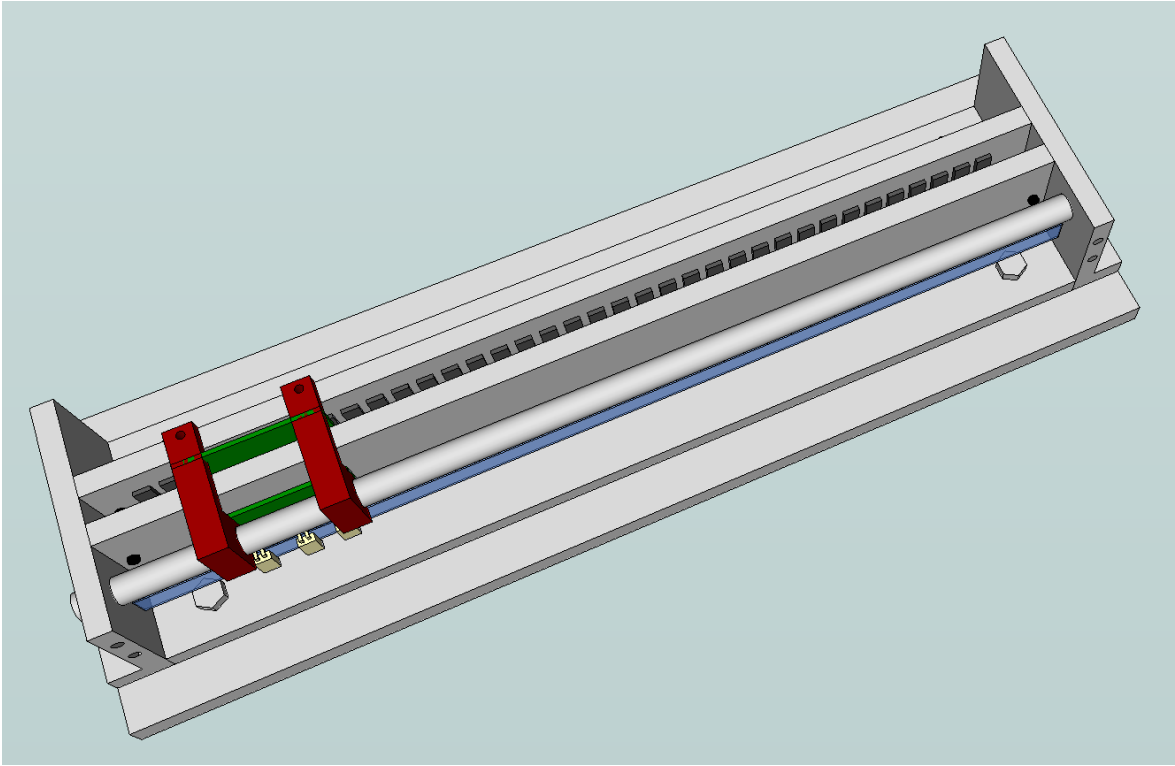


Abbildung 8: Mechanische Konstruktion (Endversion)

2.5 Parameterabschätzung

Die Simulation der magnetischen Verhältnisse führte zu folgendem Ergebnis:
Moderne Selten-Erden-Permanentmagneten haben ein Br von $1,3T$ und mehr. Bei gutem magnetischem Rückschluss und ohne Sättigung lässt sich bei einer Luftspaltdimension in der Größenordnung der Magnetdicke ein Feld von etwa $0,7T$ erreichen.

Um die benötigten Lorenzkraft von $1N$ aus 1.2 zu erreichen, ist bei einer aktiven Leiterlänge im Magnetfeld von $20mm$ folgender Strom nötig:

$$F = BIx- > I = \frac{F}{Bx} = \frac{1N}{0,7T0,02m} \approx 72A \tag{4}$$

Auf der Printplatte ist bei ca. $50\,^{\circ}C$ Erwärmung laut [7] eine maximale Stromdichte von $20A/mm$ zulässig. Eine maximalen Stromdichte von $20A/mm^2$ und eine Querschnittsfläche der Leiterbahn von $0,6mm^2$ (6mm Breite (=Magnetbreite) x 0.1mm Dicke (=max Kupferdicke) ergibt 12A Leiterstrom. Das bedeutet mindestens $72A/12A = 6Magnetfeldquerungen$ also drei Kreiswindungen mit je zwei Durchgängen. Bei einer

zweiseitigen Platine sind somit zwei Windungen pro Seite mehr als ausreichend. Die dabei entstehende vierte Reservewindung sollte ausreichen, um die allfälligen Ungenauigkeit dieser ersten Abschätzung zu kompensieren.
Es lässt sich auf jeden Fall die Aussage treffen, dass dieses Konzept grundsätzlich mit einer zweilagigen Printplatte realisierbar ist, bei mehrlagiger Printplatte entsprechend noch besser.

Nachdem alle obigen Abschätzungen in einem technisch realisierbaren Bereich sind, kann dieser Ansatz daher weiterverfolgt werden. In Kapitel 3.2 werden dann weitere Details über die Spulendimensionierung ausgearbeitet.

2.6 Modultest

Die Magnete wurden mittels einer einfachen Karton-Leere zur Einhaltung des Abstandes auf die Seitenteile gesetzt. Anschließend wurde die Gesamtlänge aller Magnete und Abstände mit einem Maßband kontrolliert. Es ist für die richtige Kommutierung sehr wichtig, den Magnetabstand möglichst genau einzuhalten.

Auf ein Festkleben bzw. Vergießen der Magnete wurde verzichtet, um während der Testphase eventuell andere Abstände ausprobieren zu können. Die Magnete sitzen durch ihre Anziehungskraft so fest auf der Stahlplatte, dass ein Verrutschen auszuschließen ist. Die korrekte Polarisation der Magnete wurde vor dem Zusammenbau der beiden Schenkel und der Grundplatte mittels eines Handkompasses kontrolliert.

Das Zusammenbauen gestaltete sich recht schwierig, da die beiden Winkel durch die vielen Magnete eine enorme Anziehungskraft aufeinander (und alle in der Nähe liegenden Metallteile) ausüben. Mit den Stellschrauben wurde beidseitig der Abstand der beiden Seitenschenkel auf die Printplattendicke plus je einem Millimeter Spiel auf beiden Seiten, also etwa $3,5mm$ justiert. Das Spiel sollte später nach Möglichkeit minimiert werden um ein möglichst starkes Feld zu erhalten.

Die lineare Führung wurde zuerst wie beschrieben durch V-förmige Einfräsungen an den Seiten der Winkel und darin gleitenden Justierschrauben realisiert. Es stellte sich nach dem Zusammenbau heraus, dass dieses System immer wieder verkantet und sehr ruppig läuft. Um eine möglichst reibungsarme Bewegung zu gewährleisten, wurde als Linearführung deshalb eine Rundstange verwendet. Auf dieser laufen zwei Kugellagerbüchsen, welche über Kunststoff-Distanzen mit der Spulenplatine verbunden sind. An diesen Distanzen kann auch die Positionsmessung befestigt werden. Die Stange wird an den Enden von zwei Aluminiumwinkeln gehalten. Auf der anderen Seite der Platine gleitet ein Kunststoffteil mit einer Justierschraube auf dem Winkeleisen, um auch den 5.Freiheitsgrad, das Kippen um die Stangenachse, zu fixieren. Dieser Aufbau läuft deutlich leichter und verkantet nicht mehr.
Eine weitere Herausforderung für die Leichtgängigkeit stellt sicher noch das nötige Schleppkabel zum DSP-Board und zur Versorgung dar. Der komplette Zusammenbau wird dies zeigen. Abschließend in Bild 9 noch ein Foto der fertigen Mechanik.

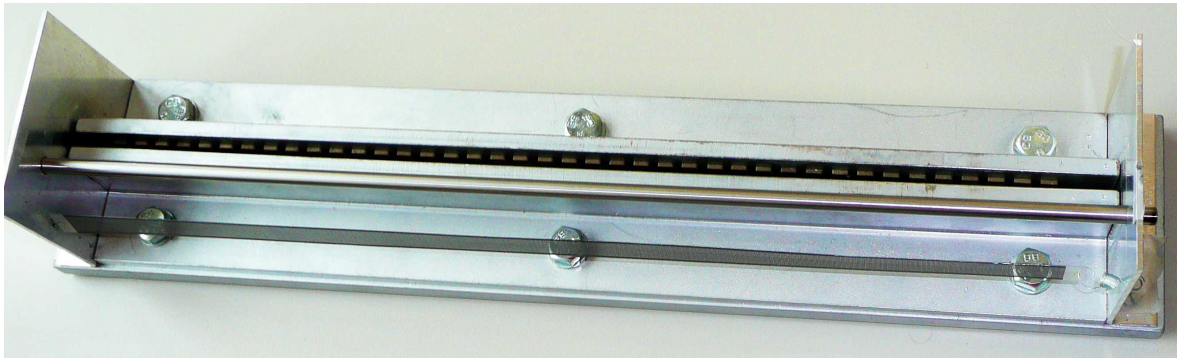


Abbildung 9: Foto der fertigen Mechanik ohne Printplatte und Positionssensor

3 Leistungselektronik

3.1 Grundüberlegungen

Die Aufgabe besteht darin, die antreibenden Spulen nicht in Form diskreter Bauelemente zu realisieren, sondern sie direkt auf der Leiterplatte zu integrieren. Dies schränkt die realisierbare Windungszahl und den mögliche Leiterquerschnitt stark ein. Insbesonders, da am Institut keine Multilayer-Platinen-Fertigung möglich ist, und somit eine zweiseitige Lösung verwendet werden soll.

In Anbetracht des sinusförmigen Feldverlaufes werden die Spulen als Drei-Phasen-System angeordnet. Die weite Verbreitung dieses Systems bringt den Vorteil, dass sehr viel Literatur und Standard-Software-Bibliotheken zur Ansteuerung der Spulen verfügbar sind. Der Stromrichter wird mit drei PWM-gesteuerten MOSFET-Halbbrücken aufgebaut. Details über deren Ansteuerung folgen später im Software-Kapitel 7.5.

Bedingt durch die kleinen Dimensionen und der sich daraus ergebenden kleinen Induktivität und des geringen Wicklungswiderstandes, werden die Spulen in Sternpunktschaltung betrieben. Durch die Ansteuerung mit hohen PWM-Frequenzen und geringen Versorgungsspannungen wird versucht, die negativen Effekte der kleinen Spulen-Parameter auszugleichen. Bei symmetrischen PWM-Signalen wird die Stromripple-Frequenz gegenüber der Ansteuerfrequenz zusätzlich noch verdoppelt, was einen weiteren kleinen Vorteil bringt. Als Schaltelemente eignen sich derzeit in einer solchen Niederspannung-Hochstrom-Anwendung Trench-MOSFETS am besten, die sehr geringe ON-Widerstände im Bereich von einigen $m\Omega$ (die deutlich unter dem Wicklungswiderstand liegen) und daher auf geringe Leitverluste führen.

3.2 Spulendimensionierung

Die Spulen sind so zu gestalten, dass immer ein möglichst großer Anteil der stromführenden Leiterbahn im Magnetfeld im Einsatz ist. Wie in Kapitel 2.5 berechnet, müssen mindestens zwei 6mm breite Windungen pro Seite vorhanden sein um genügend Kraftwirkung erzielen zu können. Bei den nachfolgenden Berechnungen wird davon ausgegangen, dass die gesamte Kraftwirkung von einer Spule erbracht werden muss. Dies ergibt eine zusätzliche Reserve da im 3-Phasen-System immer mehr als eine Spule zur Kraftwirkung beiträgt.

Aus diesen Berechnungen und den weiteren Erkenntnissen von Kapitel 2 entstand folgender Entwurf der PCB-Spulen:

Standard-Entwurf der Spulenanordnung:

- Die maximale Spulenschenkelbreite einer Wicklung ist durch die Magnetbreite auf 6 mm beschränkt.
- Zusammen mit dem Polabstand von $3mm$ hat eine Spule die Breite von $6mm + 3mm + 6mm = 15mm$.

- Die mechanische Periode des Feldes beträgt $p = 18mm$.
- Ein Versatz von 120 ° entspricht $\frac{18mm*120^\circ}{360^\circ} = 6mm$ oder um ein Vielfaches der Periodenbreite mehr z.B. 24mm, 42mm, 60mm, ...
- Die dritte, um 240 ° versetzte Spule ist entsprechend $\frac{18mm*240^\circ}{360^\circ} = 12mm$ oder 30mm, 48mm, 66mm, ... versetzt.

Um alle drei Phasen der Reihe nach nebeneinander anzuordnen, braucht man somit eine Gesamtlänge von 63mm. Siehe dazu auch Bild 10 a.

Es gelang jedoch durch einen Trick eine äquivalente Anordnung zu entwickeln, die dasselbe Ergebnis auf deutlich weniger Raum ermöglicht:

Kompakt-Entwurf der Spulenanordnung:

- Wie in Bild 10 b gezeigt, wird die erste Spule auf 0mm, und die dritte Spule auf 30mm angebracht.
- Die zweite Spule wird nun nicht auf die nächste freie Position bei 60mm gesetzt, sondern wird um 180 ° versetzt, auf 15mm positioniert.
- Diese mechanische Verschiebung um 180 ° wird durch eine elektrische Verschiebung um ebenfalls 180 ° wieder ausgeglichen. Dies entspricht lediglich einer Umpolung der Wicklung.

Somit lassen sich alle drei Windungen kompakt auf einer 45mm breiten Platine unterbringen. Dies bedeutet eine Ersparnis an Platz und Gewicht von ca. 30%.

Die geringere Breite von nur 45mm bringt auch einen weiteren Vorteil. Es lassen sich dadurch auf einer halben Europakarte mit $80mm \times 100mm$ zwei Wicklungen pro Phase unterbringen. Dadurch verdoppeln sich die Durchflutung, die Induktivität und die Kraftwirkung. Somit kann der Nachteil der zweilagigen gegenüber einer vierlagigen Platine verringert werden.

Bei zwei Spulen pro Phase und doppelseitiger Ausführung mit nur einer einzigen 6mm breiten Windung passiert der Strom pro Umlauf nun achtmal das Magnetfeld, was für die benötigte Kraft ausreicht. Durch spiralförmige Unterteilung der Leiterbahn lässt sich die Zahl der Windungen weiter erhöhen. Dies bringt Vorteile beim Stromripple durch die höhere Induktivität.

Die optimale Anzahl der Windungen ist ein Kompromiss zwischen der dadurch erzielten Erhöhung der Induktivität und der Verringerung der stromführenden Kupferfläche durch den Windungsabstand, was den zulässigen Gesamtstrom beschränkt. Herstellungsbedingt ergibt sich ein minimaler Windungsabstand von $0,1mm$ und die max. Dicke der Kupferschicht beträgt $0,1mm$. Die sich daraus ergebenden Werte sind in Tabelle 3.2 dargestellt. Um die maximale Erwärmung der Spule in einem vernünftigen Rahmen zu halten, wird die in Kapitel 2.5 beschriebene maximale Stromdichte von $20A/mm^2$ benutzt.

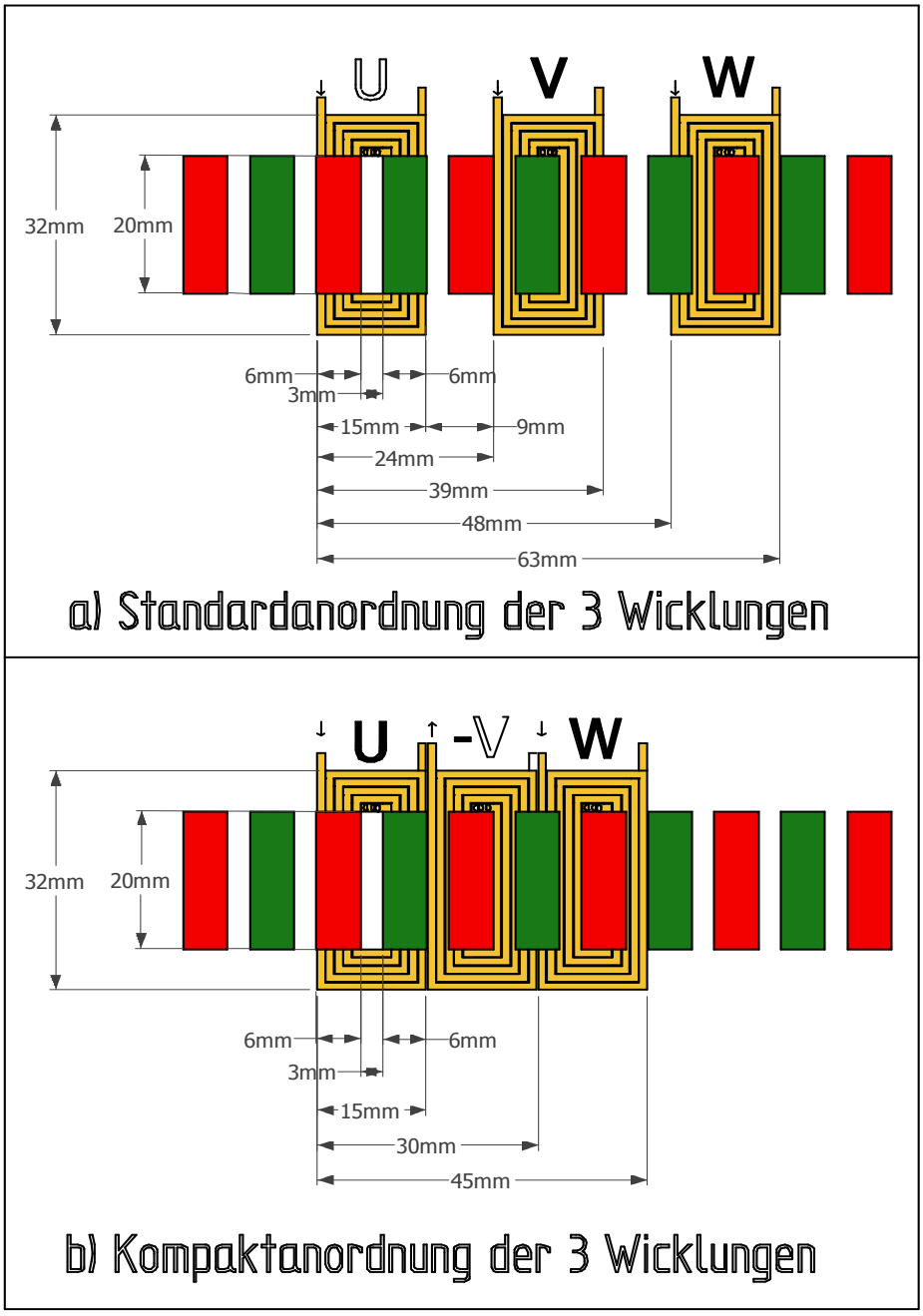


Abbildung 10: Spulenanordnung a)Standard-Entwurf, b) Kompakt-Entwurf

Es lässt sich der Spulenwiderstand aus den Abmessungen und dem spezifischen Widerstand des Kupfers von $\rho_{CU} = 17 \cdot 10^{-9}\Omega m$ berechnen,

$$R_{Spule} = \rho_{CU} \frac{l_{Spule}}{A_{Spule}}$$

wobei in diesem Fall die Länge und die Querschnittsfläche wie folgt von der Windungszahl N abhängt:

$$l_{Spule}(N) = l_{mittel} \cdot N \cdot Seiten \cdot Spulen = 2 \cdot (b_{mittel} + h_{mittel}) \cdot N \cdot 2 \cdot 2$$
$$l_{Spule}(N) = 2 \cdot (9mmm + 26mm) \cdot N \cdot 2 \cdot 2$$

$$A_{Spule}(N) = Kupferdicke \cdot \frac{Breite - N \cdot Abstand}{N}$$
$$A_{Spule}(N) = 0.1mm \cdot \frac{6mm - N \cdot 0.1mm}{N}$$

Somit ergibt sich der Spulenwiderstand abhängig von der Windungszahl zu:

$$R_{Spule} = \rho_{CU} \cdot \frac{l_{mittel} \cdot N^2}{0.1 \cdot (6mm - N \cdot 0.1mm)}$$

.

Die Induktivität lässt sich als Luftspule ohne Eisenkreis überschlagsmäßig abschätzen mit der empirisch ermittelten Wheeler-Formel [19] [18] für spiralförmige, mehrlagige Luftspulen:

$$L_{Wheeler} = 0.8 \cdot a^2 \cdot N^2/(6 \cdot a + 9 \cdot b + 10 \cdot c)$$

mit: a = mittlerer Windungsradius, b = höhe der Spule, c = äußerer minus innerer Spulenradius, N = Windungszahl, Alle Dimensionen in inches, L in μH

Der komplexe Widerstand der Spule beträgt also:

$$\overline{Z} = R + j\omega L = 1.5\Omega + j2\pi \cdot f_{PWM} \cdot 31.88 \cdot 10^{-6} H = 1.5\Omega + j4\Omega$$

In Betrag und Richtung dargestellt $|Z| = 4.27\Omega$ und $\alpha_Z = 70Grad$

Der maximale Dauerstrom bei einer Stromdichte von $20A/mm^2$ ergibt sich zu:

$$I_{max} = A_{Spule} \cdot N \cdot 20A/mm^2$$

Dieser muss, summiert über alle Windungen, mindestens die in Kapitel 2.5 berechneten 72A ergeben, um genügend Kraftwirkung erzielen zu können. In Tabelle 3.2 sind die Ergebnisse für verschiedene Windunszahlen dargestellt.

N	l_{Spule}/mm	A_{Spule}/mm^2	R/Ω	$L_{Luft}/\mu H$	I_{max}/A	I_{ges}/A
1	280	0.590	0.008	0.22	11.80	94.40
2	560	0.290	0.033	0.88	5.80	92.80
3	840	0.190	0.075	1.99	3.80	91.20
4	1120	0.140	0.136	3.54	2.80	89.60
5	1400	0.110	0.216	5.53	2.20	88.00
6	1680	0.090	0.317	7.97	1.80	86.40
7	1960	0.076	0.440	10.84	1.51	84.80
8	2240	0.065	0.586	14.16	1.30	83.20
9	2520	0.057	0.756	17.93	1.13	81.60
10	2800	0.050	0.952	22.13	1.00	80.00
11	3080	0.045	1.175	26.78	0.89	78.40
12	3360	0.040	1.428	31.88	0.80	76.80
13	3640	0.036	1.712	37.41	0.72	75.20
14	3920	0.033	2.028	43.39	0.65	73.60
15	4200	0.030	2.380	49.81	0.60	72.00

Tabelle 2: Spulenwerte bei verschiedenen Windungszahlen

Es wird eine Windungszahl von 12 gewählt, da hier der Widerstand deutlich den R_{ON} der Transistoren und der Wert des Shunts zur Strommessung übersteigt. Auch der Induktivitätswert hat dann eine annehmbare Größe, die Abmessungen lassen sich gut fertigen. Der Verlust an Kupfer durch den Leiterbahnabstand liegt in einem erträglichen Rahmen, sodass die Durchflutung deutlich über der Minimalanforderung liegt.

Der Widerstand und die maximalen Stromwerte wurden auch mit den Ergebnissen der PCB-Therm-Software [8] verglichen. Die Ergebnisse in Bild 11 deckten sich bezüglich des Widerstandes sehr gut mit den eigenen Berechnungen. Der Maximalstrom ist jedoch deutlich höher, da diese Software von $50A/mm^2$ Stromdichte ausgeht. Bedingt durch die schlechte Wäremabfuhr der eng gewickelten Leiterbahnen in dem schmalen Spalt zwischen den Magneten, erscheinen die $20A/mm^2$ aber eher sinnvoll.

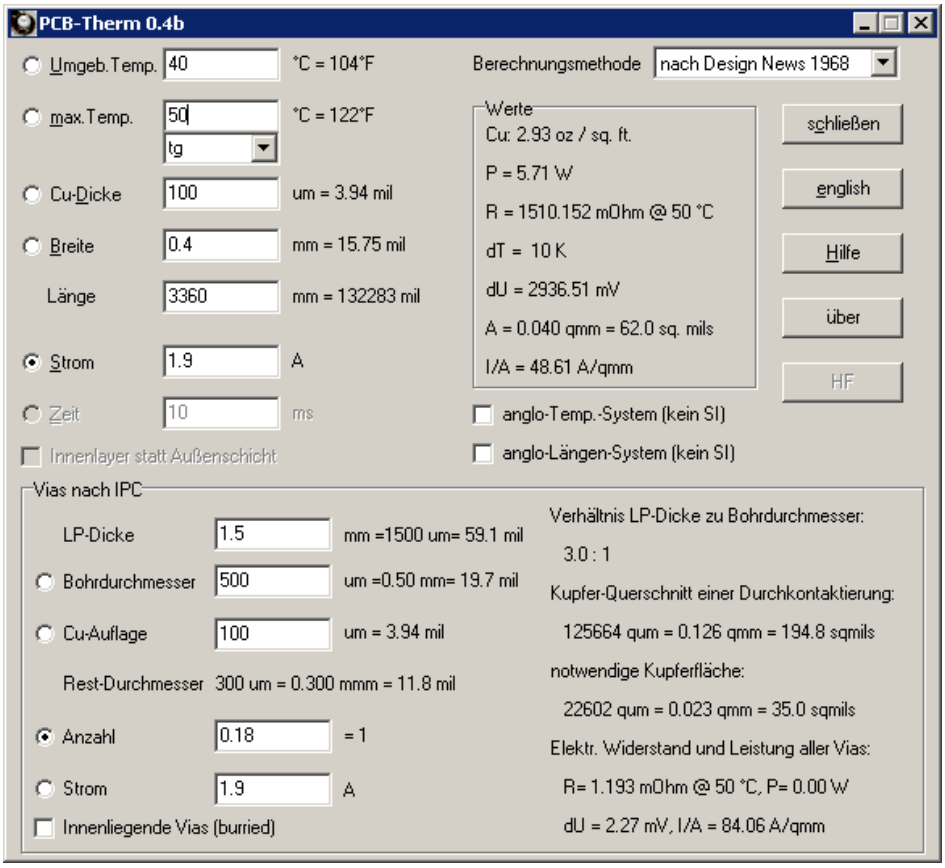


Abbildung 11: Mit PCB-Therm berechnete Spulenwerte

Es werden jedoch die von der Software berechneten 2A als Spitzenstrom und somit als Messbereich für die Strommessung herangezogen.
Um diesen Stom zu erreichen, ist eine Versorgungsspannung von mindestens

$$U = |Z| \cdot I = 4.27\Omega \cdot 2A = 8.5V$$

nötig. Bedingt durch die Gegen-EMK wird die benötigte Spannung bei bewegtem Motor nochmals höher sein.

Es lässt sich daraus auch der zu erwartende Strom-Ripple abschätzen aus dem Stromanstieg in der Induktivität von

$$dI/dt = U/L = 8.5V/32uH = 265A/ms$$

der bei einer PWM-Frequenz von 20kHz

$$dI/dpwm = 265A/ms \cdot 1s/20 \cdot 10^3 = 13.3A/PWM - Periode$$

beträgt.
Die Spulen können somit sehr wenig zur glättung des Stroms beitragen. Für den Fall, dass dies zu Schwirigkeiten führt, werden diskrete SMD-Glättungsdrosseln im Layout vorgesehen. Diese könnten bei Bedarf bestückt werden.

Da davon auszugehen ist, dass die mechanische Grenzfrequenz deutlich unter der PWM-Frequenz liegt, ist bei einhalten des gewünschten Mittelwertes des Stroms der Motor funktionstüchtig. Mechanische Schwingungen bei 20kHz sollten gut gedämpft sein.

Die Kommutierungsfrequenz bei einer mechanischen Periode von $18mm$ und einer Geschwindigkeit von $1m/s$ ergibt sich zu

$$f_{Kom} = \frac{1000mm/s}{18mm} = 56Hz$$

. Dies ergibt ein Verhältnis von

$$\frac{f_{PWM}}{fKom} = \frac{20 \cdot 10^3 Hz}{56Hz} = 357 \frac{PWM Perioden}{Kommutierung}$$

was für eine gute Stromregelung ausreichen sollte.

Betrachtet man die Spule als Regelstrecke für die Stromregelung, so hat diese bei der PWM-Frequenz einen starken ohmschen Charakter. Es würde sich daher anbieten, sie großteils über einen Feed-Forward Pfad zu steuern und lediglich die Nichtlinearitäten mit einem I-Anteil auszuregeln. Überlegungen dieser Art werden in Kapitel 6 bzw. bei der Regler-Inbetriebnahme folgen.

Das endgültige Layout der Spulen ist in Bild 12 zu sehen.

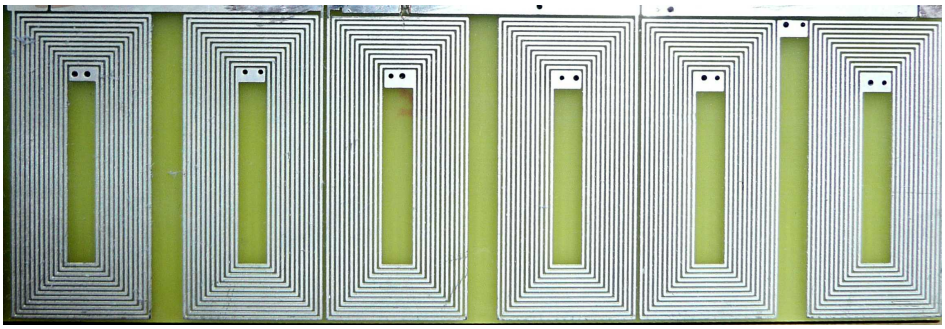


Abbildung 12: Foto der fertigen Printspulen

3.3 Ansteuerschaltung

Angesteuert werden die Spulen von je einer Mosfet-Halbbrücke, welche über vorgesetzte Treiberbausteine direkt von der PWM-Stufe des Prozessors angesteuert werden. Bei der Wahl der Transistoren wurde besonders auf einen geringen Leitwiderstand geachtet, da auch der Spulenwiderstand sehr gering ist. Nebenher wurde auch Wert auf eine geringe Ansteuerleistung und möglichst kleine Gate-Kapazität gelegt. Es wurde der FDB8441 N-Kanal Mosfet von Fairchild [5] gewählt, der einfach über Distributoren erhältlich ist und dessen Kenndaten den Anforderungen entsprechen.

Versorgt wird die Brücke mit einer Spannung im Bereich von unter $20V$ aus einem

Labornetzteil. Diese Spannung wird auf der Platine mit großen Kondensatoren, die einen niedrigen ESR-Wert haben, gestützt.

Um die oberen N-Kanal Transistoren zu schalten ist eine Spannung nötig, die höher als die Brückenversorgung ist. Daher werden Mosfet-Brückentreiber eingesetzt, die durch eine integrierte Ladungspumpe diese Spannung erzeugen können. Als gut zu den Transistoren passend wurde der IR2183 von International Rectifier [15] ausgewählt. Bei diesem wird beim Schalten der beiden Transistoren einer Halbbrücke durch eine integrierte Logikschaltung eine Todzeit eingefügt, die das gleichzeitige Durchschalten verhindert und die Transistoren vor Zerstörung schützt. Die beiden Transitoren werden immer komplementär geschaltet, sodass pro Halbbrücke nur ein PWM-Signal nötig ist. Um Einfluss auf die Slew-Rate des Transistors nehmen zu können, ist bei der Gate-Ansteuerung ein Serienwiderstand vorgesehen.

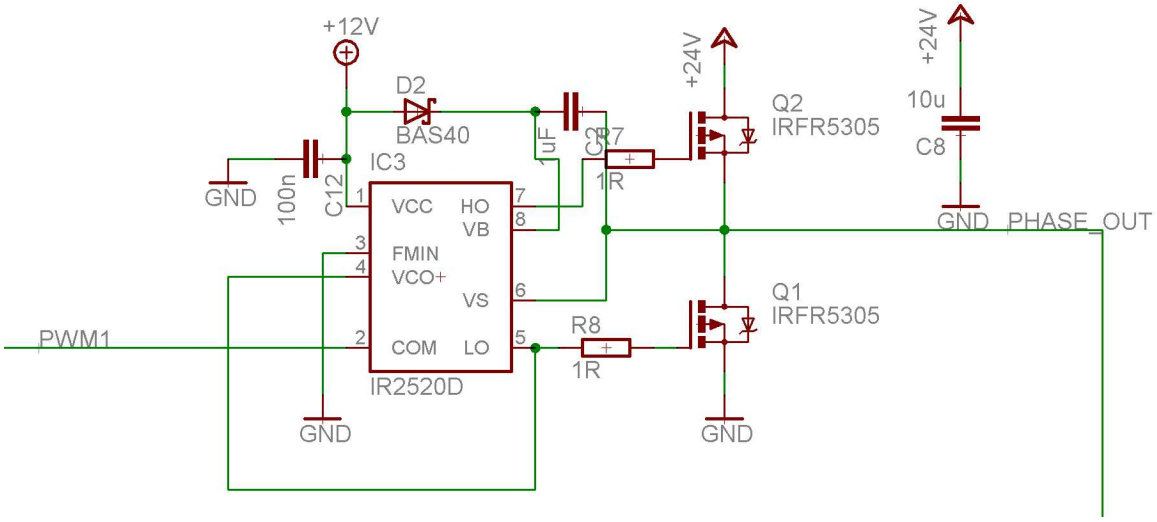


Abbildung 13: Schema Leistungsansteuerung

3.4 Strommessung

Um den in den Wicklungen fließenden Strom zu messen, müsste direkt an der Spule zwischen den Brücken gemesssen werden. Dies erfordert entweder einen Shunt mit einem angeschlossenen Verstärker, welcher eine hohe Common-Rate-Unterdrückung aufweist, oder die Messung muss mit einem komplett potentialgetrennten Sensor erfolgen.
Da die üblichen LEM-Wandler für diesen Aufbau zu groß und zu schwer sind, wird hier ein kleiner, auf dem Hall-Prinzip aufbauender Sensor verwendet. Es stellt sich laut Datenblatt der ACS712 Hall-Sensor von Allegro [1] als geeignet heraus.

Für die Messmethode via Shunt-Widerstand bietet sich ein speziell darauf getrimmter Bauteil, der AD8210 von Analog Devices [3] mit SMD-Shunt-Widerständen an.
Die Kirchhoffschen Gesetze ermöglichen es, den Strom nur in zwei Zweigen zu messen und daraus den dritten Strom rechnerisch zu bestimmen.

Berechnungen zum AD8210:
Bei einem Maximalstrom von $I_{max} = 2A$ aus Kapitel 3.2, einer Verstärkung des AD8210 Sromsensors von $g = 20\frac{mV}{A}$ und einer maximalen Eingangsspannung des ADCs von $U_{ADC} = 3.3V$ für den gesamten ±-Bereich, ergibt sich der Shunt-Widerstand zu:

ADC-Auflösung:

$$\frac{U_{ADC}}{2^{10}Bit} = \frac{3.3V}{1024} = 3.22\frac{mV}{Bit}$$

Sensorverstärkung:

$$U_{shunt} = \pm\frac{U_{ADC}}{2 \cdot g} = \pm\frac{3.3V}{2 * 20} = \pm 82.5mV$$

Shuntwiderstand:

$$R_{shunt} = \frac{U_{shunt}}{I_{max}} = \frac{82.5mV}{2A} = 41.25m\Omega$$

Es wird ein etwas größerer Shunt aus 2 parallelen $100m\Omega$ Widerständen gewählt. Dies reduziert zwar den Messbereich, was aber immer noch ausreichend ist.

Shunt gewählt $50m\Omega$:

$$I_{max} = 82.5mV/50m\Omega = 1.65A$$

Sensorauflösung:

$$\frac{3.3V}{2 \cdot 1.65A} = \frac{1000mV}{A}$$

Stromauflösung:

$$\frac{1A}{1000mV} * \frac{3.22mV}{Bit} = 3.22\frac{mA}{Bit}$$

Die Schaltung wird in der Form von „Figure 30“ im Datenblatt [3, Seite 12,13] als Split External Reference mit den 3.3V des ADCs aufgebaut.

Zusätzlich wird die in „Figure 32“ des Datenblattes beschriebene zusätzliche Filterung vorgesehen. Diese ermöglicht bei Einschränkung der Bandbreite eine hinreichende Glättung des Strommesswertes. Dies ist in diesem Aufbau besonders wichtig, da der Strom, bedingt durch die geringe Induktivität, einen sehr hohen Rippel enthält und in weiten Teilen des Arbeitsbereiches in den lückenden Betrieb fällt, der ansonsten zu falschen Messwerten führen würde.
Der Filter wurde dimensioniert auf $f_{3dB} = 16kHz$ durch $R = 1\Omega$ und $C = 10\mu F$.

Berechnungen zum ACS712-05:
Der ACS712-05 hat einen Strom-Messbereich von $I_{max} = \pm 5A$, eine Auflösung von $g = 185\frac{mV}{A}$. Das Rauschen von $37mV$ bei $fg = 16kHz$ entspricht somit etwa $= 200mA$ und der Offset von $\pm 40mV$ wärend entsprechend $= 220mA$ des beim Messstrom. Diese Werte erscheinen ziemlich hoch und sprechen gegen den Einsatz diese Bauteils

Der Ausgangsbereich von $2.5V \pm 2V$ bei einer Speisung von $U_{Vcc} = 5V$ muss auf den $0 - 3.3V$ Analogeingang angepasst werden. Dazu wird die Schaltung „Application 4.“ aus dem Datenblatt [1, Seite 10] verwendet.

ADC-Auflösung:

$$\frac{U_{ADC}}{2^{10}Bit} = \frac{3.3V}{1024} = 3.22\frac{mV}{Bit}$$

Sensorauflösung:

$$185\frac{mV}{A}$$

Stromauflösung:

$$\frac{1A}{185mV} * \frac{3.22mV}{Bit} * \frac{10k\Omega}{12k\Omega} * = 14.5\frac{mA}{Bit}$$

Die erreichbare Messgenauigkeit hängt nicht von oben errechnetem Wert ab, sondern ergibt sich bei diesem Bauteil klar durch das Rauschen von 200mA.

Es handelt sich hierbei um überschlagsmäßige Werte, ohne der detaillierten Berücksichtigung des Rauschens anderer Bauteile, Bauteiltoleranzen, Störungen durch Schaltvorgänge, etc.. Praktisch wird eine noch geringere Genauigkeit erwartet.

Da keine Erfahrungen mit dem ACS712 vorhanden sind und das Rauschen recht groß erscheint, werden im Layout beide Messmethoden vorgesehen. Es wird sich zeigen, ob der Hall-Sensor hinreichend genau ist, um den Strom regeln zu können und welche Schaltung die besseren Messergebnisse bringt. Die 200mA Auflösung erscheint hierfür zu gering. Falls nötig könnte auf dem DSP-Evaluationsboard bei den Analogeingängen eine zusätzliche Adapterplatine mit einer OPV-Schaltung zwischengeschaltet, um die Messsignale durch Verstärkung und weitere Tiefpassfilterung anpassen zu können.

Die Messmethode über einen Shuntwiderstand ist obigen Berechnungen zufolge weitaus besser geeignet. Besonders, da die Common-Mode Unterdrückung bei den hier verwendeten, kleinen Versorgungsspannungen unter $20V$ keine grosse Rolle spielt. Es wird deshalb diese Methode umgesetzt, wobei für eventuelle spätere Tests das Platinenlayout so erstellt wird, dass zum Vergleich auch der Hall-Sensor bestückt werden kann.

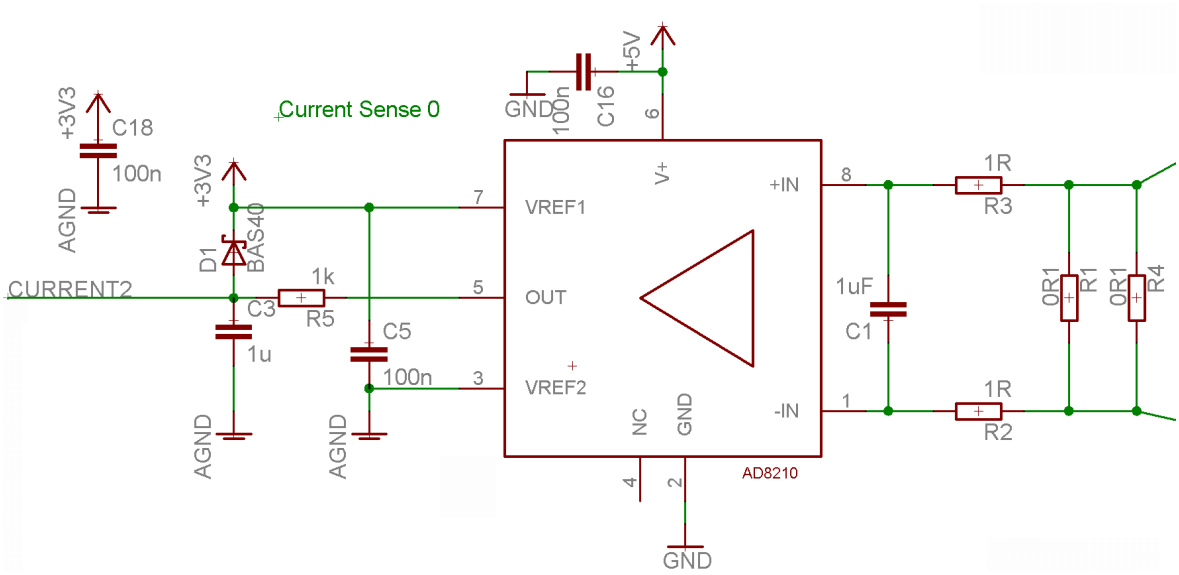


Abbildung 14: Schema Strommessung

3.5 Verbindungs-Board

Um den Leistungsteil, die Positionsmessung und die Kommunikation mit dem DSP-Evaluationsboard zu verbinden, musste ein Verbindungsboard, welches die verschiedenen Stecker umsetzt, erstellt werden. Diese Gelegenheit wurde auch genutzt, darauf einen „Notaus-Schalter“ anzubringen, mit dem die PWM-Ausgänge im Fehlerfall abgeschaltet werden können. Der Schaltplan ist im Bild 15 angegeben. Ein Layout wurde nicht erstellt, da diese einfache Schaltung, wie in Bild 16 zu sehen, auf einem Lochrasterprint aufgebaut wurde.

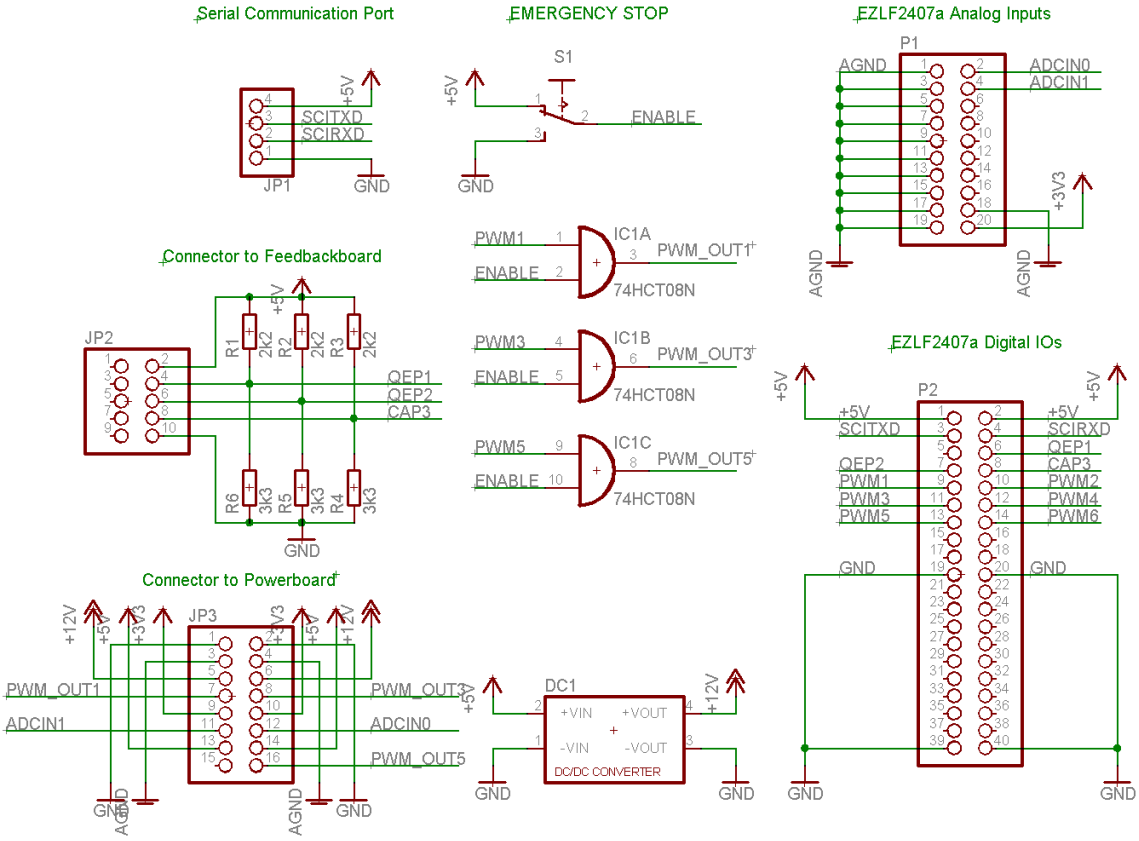


Abbildung 15: Schema Verbindungs-Board

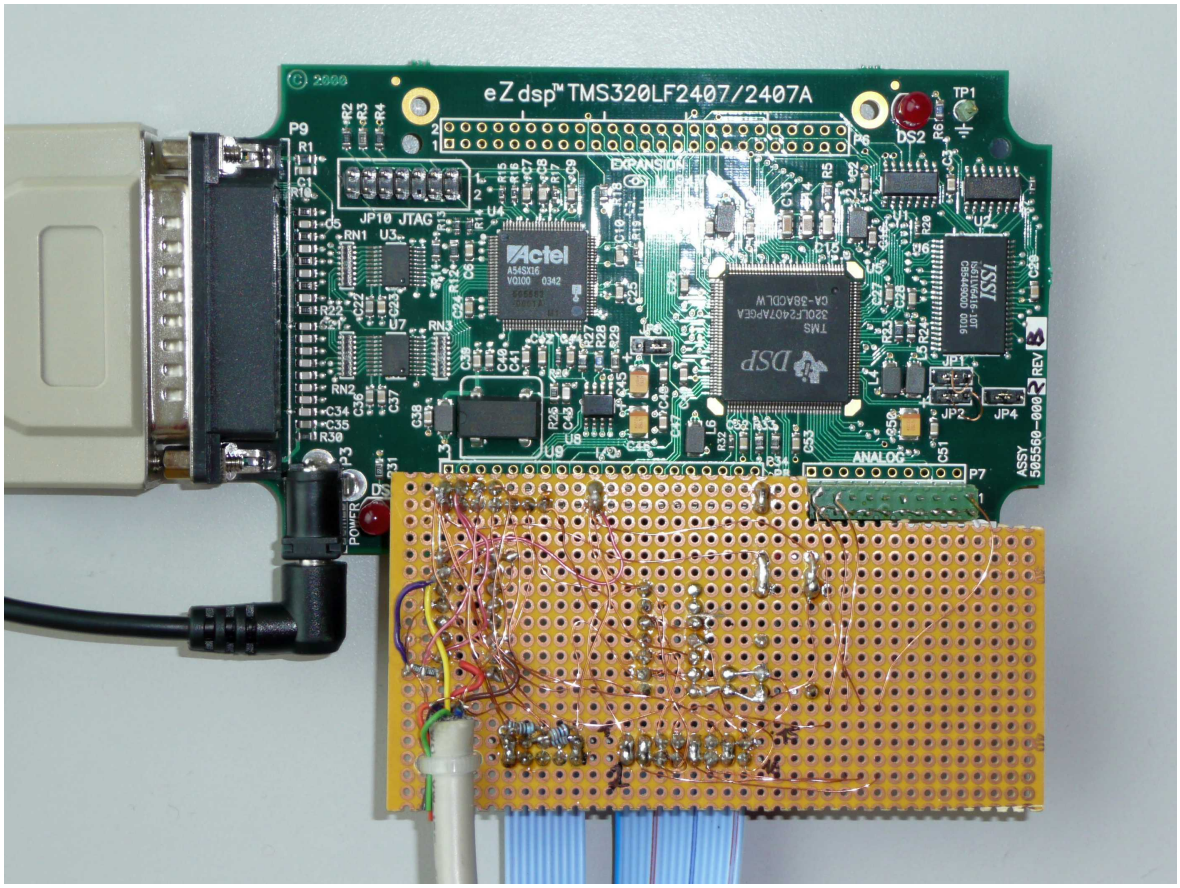


Abbildung 16: Foto des Verbindungs-Boards mit DSP-Evaluationsboard

3.6 Konstruktive Details und Zeichnungen

Die Platinenabmessungen des Leistungsprints ergeben sich aus den drei um 120 Grad versetzten Spulen, welche, wie schon beschrieben, auf eine halbe Europakarte passen. Auf dem freien Platz darüber wird die Leistungselektronik und die Strommessung mit SMD-Bauteilen aufgebaut. Der komplette Schaltplan und das Platinenlayout des bewegten Prints, sowie das Schema des Verbindungs-Boards ist im Anhang C zu finden.

3.7 Modultest

Nach einer optischen Kontrolle der Printplatte, wurde diese mit den Bauteilen bestückt. Es stellte sich dabei leider heraus, dass der Footprint der MosFET-Treiber nicht korrekt war. Diese mussten deshalb mit einigen Drähten fliegend verdrahtet werden, um den Print verwenden zu können.

Vor der Bestückung der Brücken zum Sternpunkt wurden die Spulenwerte nochmals ausgemessen: Es wurde eine U-I-Kennlinie (Bild 17) erstellt, indem mit einem Strombegrenzten Labornetzteil fixe Ströme eingeprägt, und mit zwei Multimetern Spannung und Strom direkt an der Spule gemessen wurde. Bei Strömen über 1.5A erhitzte sich die Wicklung recht schnell, daher wurde sehr kurz gemessen, um die Kennlinie nicht

zu sehr zu verfälschen. Es ergab sich ein sehr linearer Wert von 3Ω. Der leichte Ansteig des Widerstandes lässt sich auf die Erwärmung zurückführen.

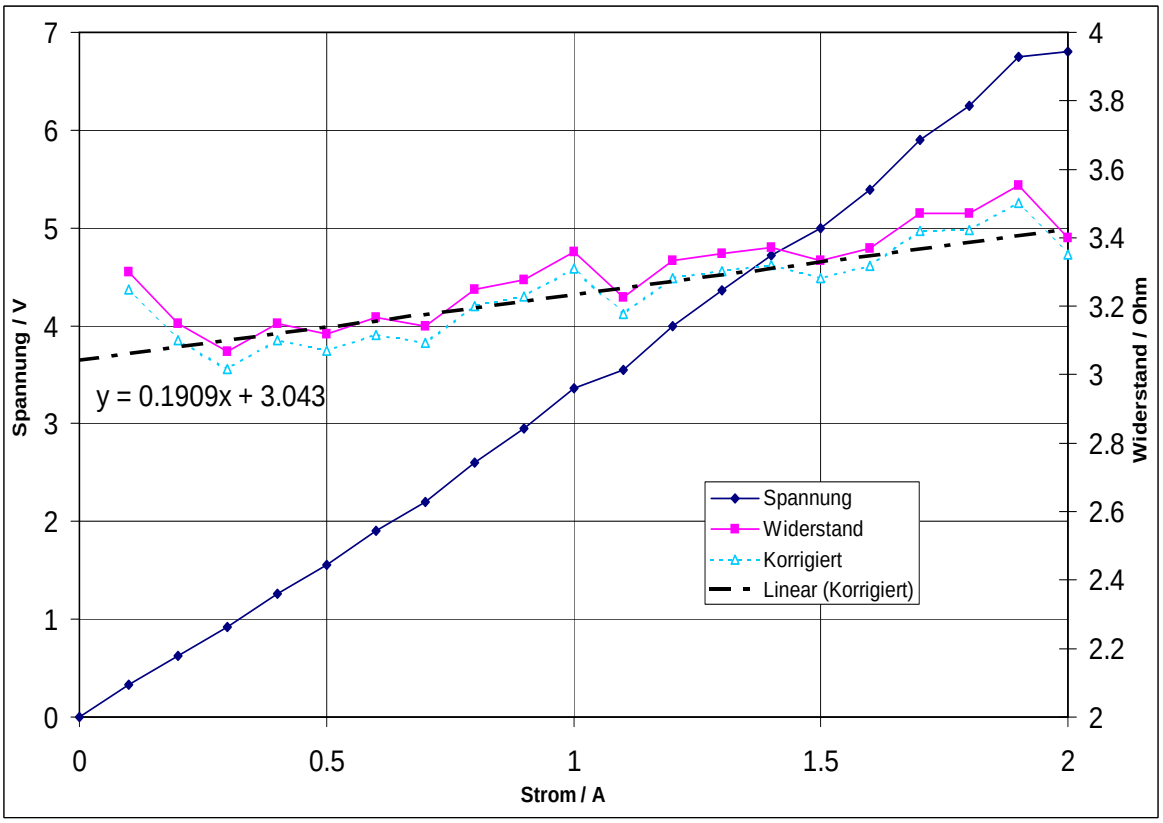


Abbildung 17: U-I Kennlinie der Printspule

Messungen mit den beiden am Institut verfügbaren RCL-Metern[1] ergaben ebenfalls einen Serienwiderstand von $3,0\Omega$ und eine Induktivität der Luftspule von $20\mu H$. Die Spule mit Eisenrückschluss konnte leider nicht gemessen werden, da hierfür längere Messleitungen nötig waren, die die Ergebnisse schon bei leichter Bewegung oder Berührung so weit verfälschten, dass die Ergenbiss unbrauchbar waren.

Die Messungen entsprechen nicht den oben berechneten Werten von 1.43Ω und $31.880\mu H$. Die Abweichungen des Widerstands um den Faktor zwei können teilweise auf nicht berücksichtigte Zuleitungen, Durchkontaktierungen und die PCB-Fertigungstoleranzen zurückgeführt werden. Ganz erklären lässt sich eine so große Abweichung dadurch jedoch nicht.
Da die Induktivität mit einer empirischen Formel, die nicht genau auf dieses Problem zugeschnitten ist, berechnet wurde, liegt die Abweichung dort im erwarteten Bereich.

Anschließend wurden die Versorgungsspannungen angelegt und die Strommessung getestet. Dies geschah durch Bestromen der Shuntwiderstands mit dem Labornetzteil und Messung der Ausgangsspannung des Sensors über ein Multimeter. Die Erebnisse

[1]HIOKI 3532-50 LCR HITEST und PHILIPS PM6303 RCL-METER

entsprachen genau den oben berechneten Sollwerten und werden hier nicht extra aufgeführt.

Die weitere Inbetriebnahme kann nur in Verbindung mit der DSP-Software erfolgen und wird deshalb im Kapitel 7 beschrieben.

4 Serielle Kommunikation

4.1 Grundüberlegungen

Der Linearantrieb soll über die serielle RS232 Schnittstelle eines PCs angesteuert werden. Um ihn einfacher und universell verwendbar zu machen, werden die Daten im lesbaren ASCII-Format ausgetauscht. Dies ist zwar nicht sehr effizient, ermöglicht es aber ohne eigene Steuersoftware auf dem PC auszukommen und stattdessen mit einem seriellen Terminal (z.B. das freie BRAY-Terminal [17], minicom oder Hyperterminal) das Auslangen zu finden.
Als Übertragungsparameter werden eine Baudrate von 9600bps, 8 Datenbits, 1 Stopbit, ohne Flusssteuerung verwendet.

4.2 Schaltungsentwurf

Der verwendete Prozessor verfügt über eine asynchrone serielle Schnittstelle, welche auf 3.3V-Pegel arbeitet. Um diese mit einem PC über RS232 verbinden zu können, müssen die Signalpegel angepasst werden. Für diese Aufgabe gibt es Bauteile wie z.B. der verwendete MAX3221 der Firma Maxim [12]. Die Schaltung kann nach dem Referenzdesign aus dem Datenblatt aufgebaut werden.
Da der Prozessor mit 3.3V arbeitet, das Eval-Board aber mit 5V versorgt wird, muss die Versorgung des Pegelwandlers durch einen zusätzlichen Spannungsregler erzeugt werden. Die Steckerbelegung des 9poligen Sub-D wurde als DCE gewählt, sodass ein 1:1 Kabel zur Verbindug ausreicht und kein Crossover-Kabel verwendet werden muss.

4.3 Konstruktive Details und Zeichnungen

Die Schaltung ist einseitig in SMD-Technik ausgeführt. Die Schaltung wurde so kompakt aufgebaut, dass sie direkt im Gehäuse einer 9-poligen Sub-D-Buchse Platz findet. In Bild 18 sind die Schaltung, die verwendeten Bauteile und auch das Platinenlayout dargestellt.

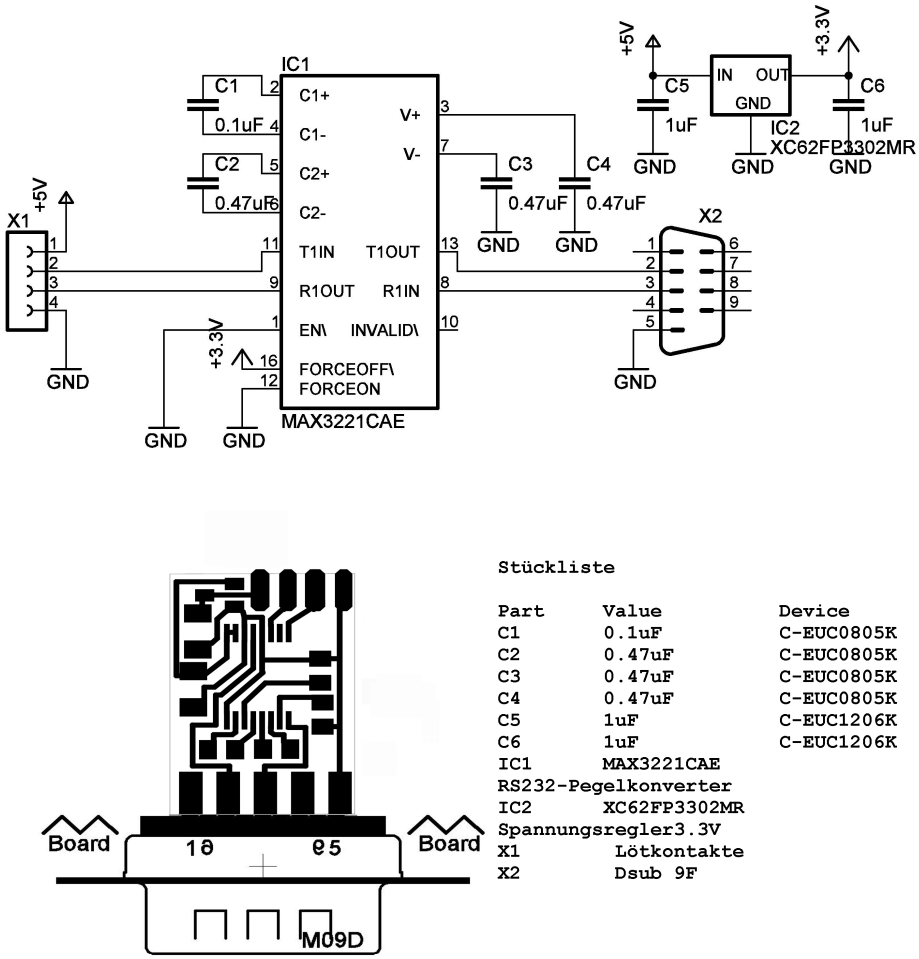


Abbildung 18: Schaltplan des RS232-Adapters mit Stückliste und Platinenlayout

4.4 Modultest

Die Schaltung wurde aufgebaut wie in Bild 19 zu sehen. In Betrieb genommen wurde sie mittels eines kleinen Testprogrammes, bei dem der DSP ein vom PC empfangenes Zeichen zurücksendet. Sie funktionierte sofort. Auch Tests mit höheren Baudraten bis 115.200 Baud waren erfolgreich. Daher wurde für die Endversion auch die höhere Baudrate von 115200bps verwendet.

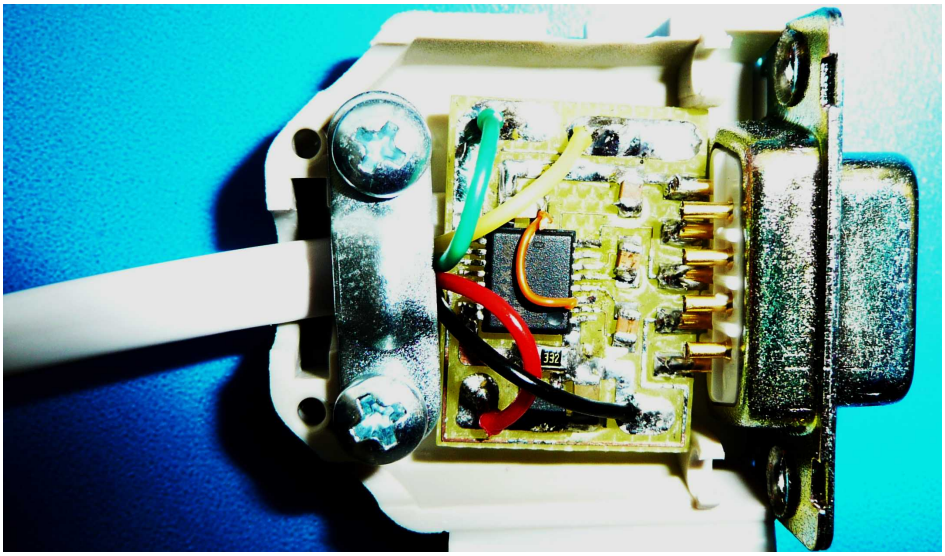


Abbildung 19: Foto des fertig bestücken Kommunikationsinterface im DSUB-Gehäuse

5 Positionssensor

5.1 Grundüberlegungen

Um die Positionierung zu realisieren und die Kommutierung der Antriebsspulen durchzuführen, muss die Position innerhalb des Systems bekannt sein. Diese kann absolut codiert werden, oder der Weg wird inkrementell mitgezählt und mit einer Referenzmarke auf den richtigen Wert gesetzt. Absolute Systeme erfordern viele Abtastspuren bei Digitalsystemen oder eine hohe Signalauflösung bei Analogsignalen, was einen höheren Realisierungsaufwand bringt. Inkrementalsysteme sind einfacher, haben aber den Nachteil, dass ein Zählfehler dauerhaft die Messung verfälscht.

In Verwendung sind oft auch Mischformen, bei denen in periodischen Abständen absolutkodierte Marken sind, dazwischen jedoch inkrementell gezählt wird. Dadurch gibt es mehrere Referenzen zur Kontrolle und die Absolutposition ist schon nach kurzem Verfahrweg bekannt.

Zur physikalischen Messung und Umwandlung in elektrische Signale bieten sich folgende grundsätzlichen Möglichkeiten an:

- magnetisch: Magnetfeldsensoren bestimmen die magnetische Feldstärke, die von der Position abhängt (z.B. mittels Hallsensoren, Feldplatten, magnetoresistiv) z.B.: SINCOS-Geber.
- induktiv: Veränderung der Induktivität einer Spule durch Einbringen eines ferromagnetischen Kerns oder als Transformator mit fester Erreger- und verschiebbaren Aufnehmerspulen. Zur Temperaturkompensation und Linearisierung werden diese als Brückenschaltung ausgeführt und sind unter den Namen Differentialdrossel, Differentialtransformator (LVDT) oder Resolver bekannt. Es gibt derzeit auch einige Ansätze dies sensorlos über die Antriebswicklung zu realisieren.
- Wirbelstrom: Eine Spule induziert Wirbelströme in einem leitfähigen Objekt. Durch die Rückwirkung erfolgt eine Impedanzänderung, welche gemessen wird.
- kapazitiv: Die Veränderung der Kapazität zwischen zwei Kondensatorplatten bei Änderung des Abstandes oder Einbringen eines anderen Dielektrikums wird gemessen.
- resistiv: Es wird die Abhängigkeit des elektrischen Widerstandes von der Leiterlänge ausgenutzt. (Potentiometerprinzip).
- optisch: Abbilden von hell-dunkel Raster oder Auswerten von Interferometriebilder (Moire-Muster) auf optische Sensoren.
- Wellenausbreitung: Laufzeitmessung nach dem Radarprinzip oder mittels Phasenverschiebung durch Doppler-Effekt z.B. mit Ultraschall oder RF-Signalen.
- Beschleunigungssensor: Verwendung eines MEMS-Beschleunigungssensors mit anschließender zweifachen Integration des Signals.

Die einfachste Möglichkeit wäre auch der Kauf eines fertigen Positionsmesssystems [2], bei dem lediglich ein Maßstab-Streifen aufgeklebt und der Sensor darüber am Schlitten angebracht werden muss. Die Aufgabenstellung beinhaltet jedoch den Bau einer eigenen Lösungsvariante.
Vom Aufbau her würde sich in dieser Arbeit die Messung des magnetischen Wechselfeldes an mehreren Stellen anbieten, woraus sich die Position ermitteln lässt. Vorteile wären eine hohe Robustheit gegenüber Verschmutzungen und die direkte Integration in den konstruktiven Aufbau durch Ausnutzung eines parasitären Effektes und der Kommutierung die immer synchron zum Magnetfeld ist. Allerdings wäre dies mit einem höheren Realisationsaufwand verbunden und es bräuchte auf jeden Fall einen zweiten Referenzpositionsgeber, um die korrekte Funktion zu evaluieren. AAuch könnte der Einfluss des antreibenden Magnetfeldes zu Feldverzerrungen führen und diese Lösung erschweren.

Da die Positionsmessung nicht die primäre Aufgabe dieser Arbeit darstellt, wurde die Inkrementalgeber-Methode gewählt.
Ein optischer Inkrementalgeber mit Gabel-Lichtschranke und Strichgitter nach dem Schatten-Prinzip ist mit der erforderlichen Genauigkeit im Eigenbau gut realisierbar.
Der Aufbau erfolgt als Durchlichtsensor mit zwei Strichgittern auf Transparentfolie. Das optische System mit einem Schattengitter und dazugehörigem Gegengitter mit einer Auflösung von 0,1mm kann gut im Eigenbau hergestellt werden. Die Strichgitterstreifen können direkt mit einem Photoplotter gedruckt werden. Photodioden und LEDs sind in vielen Variationen als Standardbauelemente erhältlich. Die digitalen Inkrementalsignale eignen sich auch gut zur Weiterverarbeitung in einem digitalen Controller. Eine detaillierte Beschreibung erfolgt in Kapitel 5.2.
Die Magnetfeld-Messung wäre jedoch eine Anregung zur Weiterführung dieser Arbeit. Da aber dieser und die weiteren Ansatzmöglichkeiten zu sehr vom eignetlichen Fokus dieser Diplomarbeit abweichen würden, werden sie nicht weiter verfolgt.

5.2 Schaltungsentwurf

Um die Position inkrementell zu bestimmen ist neben den Bewegungsimpulsen, die gezählt werden, auch die Richtung zu bestimmen. Es sind daher mindestens zwei Sensoren nötig. Aus diesen zwei um 90 ° versetzten Signalen [3] lassen sich mit Hilfe einer einfachen Logikschaltung Bewegungsimpulse und ein Richtungssignal generieren. Diese können dann auf einen Up-Down-Counter geführt werden, der die Position bestimmt. Ein Funktionsschema ist in Bild 20 dargestellt.

[2] z.B. die UDA-Serie der Firma Heidenhain oder der P9500-Encoder der Firma Phoenix, etc.
[3] daher auch oft der Name Quadratursignale

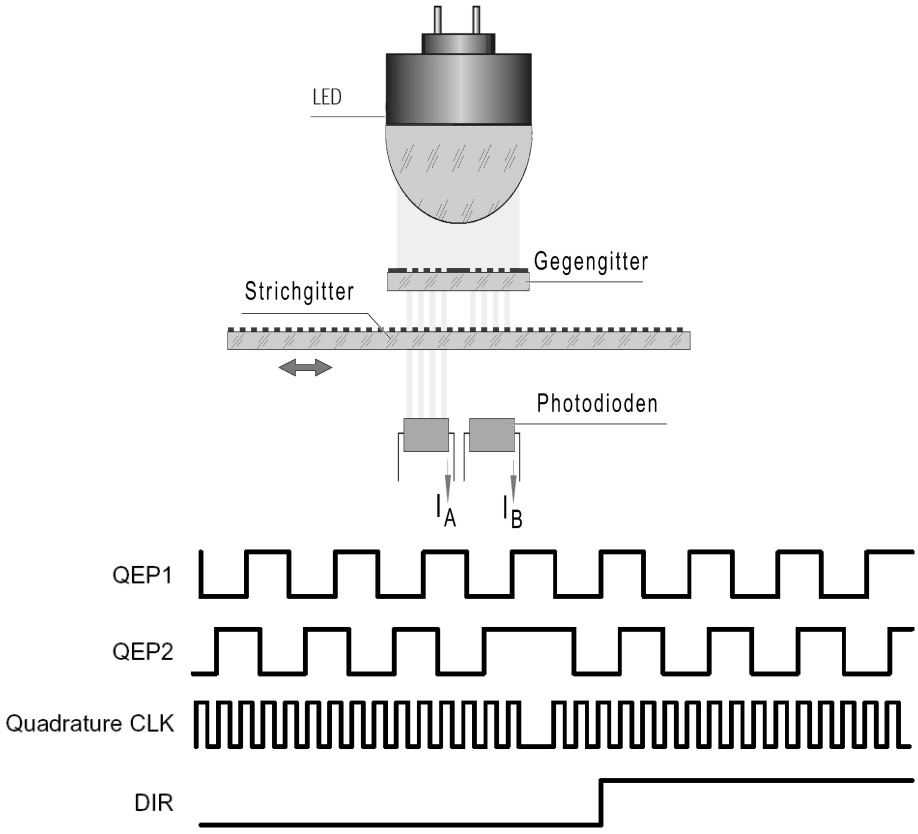


Abbildung 20: Funktionsweise und Signale der Quadraturcodierung

Der Aufbau dieser Logik kann in dieser Arbeit entfallen, da der verwendete μ-Controller bereits über einen Quadratureingang verfügt. Dieser besitzt eine interne Positionsdecoder-Schaltung mit Zähler. Die Positionsmessung beschränkt sich somit nur auf die Messung und Aufbereitung der Lichtsignale für die Prozessoreingänge und die Konfiguration des Quadratur-Decoders des Prozessors. Um Störungen durch das Umgebungslicht zu vermeiden, arbeitet das optische System im IR-Bereich und ist zusätzlich durch eine Abdeckung vor Umgebungslicht geschützt. Als optische Sender werden IR-LEDs verwendet, der Empfänger besteht aus Photodioden mit Pull-Up-Widerständen.

Durch die mit den beiden Sensoren erreichte Zweibit-Auflösung ist eine Verfierfachung des Gitterabstandes gegeben. Eine Strichstärke von 0.4mm ist somit ausreichend, um die geforderte Auflösung von 0.1mm zu erreichen.
Bei einer maximalen Fahrgeschwindigkeit von 1m/s und 0.1mm Auflösung ergibt sich eine Eingangsfrequenz von 10kHz. Dies lässt sich schaltungstechnisch problemlos realisieren. Der Prozessor ist für Zählfrequenzen bis 20Mhz ausgelegt.
Um absolute Werte zu erhalten ist ein Endschalter nötig, der als Referenz dient und den Positionszähler auf 0 setzt. Dieser wird am Ende des Fahrweges mit einem optischen Sensor gleichen Prinzips aufgebaut.

Als optische Sender werden drei LEDs mit Vorwiderständen zur Stromeinstellung verwendet. Bei der Versorgungsspannung von $5.0V$, einer Flussspannung der LEDs von ca. $1.5V$ und einem einzustellenden Strom von $35mA$ ergeben sich die Vorwiderstände zu 100Ω.
Durch die Schmitt-Trigger in den Eingängen des Prozessors könnte die Empfänger-Schaltung nur aus 3 Photodioden mit den zugehörigen Pull-Up-Widerständen von $22k\Omega$ aufgebaut werden. Da die Dioden nicht voll aussteuern und die Signalpegel durch Umgebungseinflüsse und Bauteiltoleranzen schwanken, wurde ein Komparator mit einstellbarer Schaltschwelle zwischengeschaltet. Somit erhält der Prozessor immer eindeutige Signale.

Die Schaltung wird mit 5V versorgt, der Prozessor arbeitet mit 3.3V. Da die Komparator-Ausgänge Open-Collector Signale sind, können diese durch Spannungsteiler auf 5V am Evalboard einfach angepasst werden. Die Leuchtdioden werden an den 5V betrieben.

5.3 Konstruktive Details und Zeichnungen

In Bild 22 sind die Schaltung, die verwendeten Bauteile und auch das Platinenlayout dargestellt. Die Schaltung ist einseitig in SMD-Technik ausgeführt. Es wurden Photodioden mit einer großen aktiven Fläche gewählt und das Gegengitter direkt darauf aufgeklebt. Die Sendedioden sind als bedrahtete Bauteile ausgeführt, bei welchen die Beine so gebogen werden, dass ihr Lichtaustritt über den Photodioden platziert ist und zwischen den Bauelementen der Strichgitterstreifen durchlaufen kann.

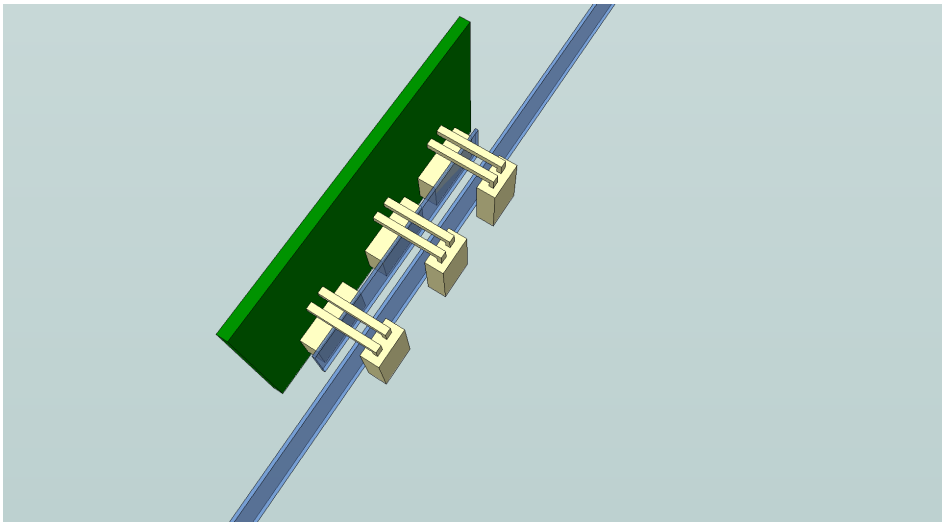


Abbildung 21: Aufbau des Positionsgebers

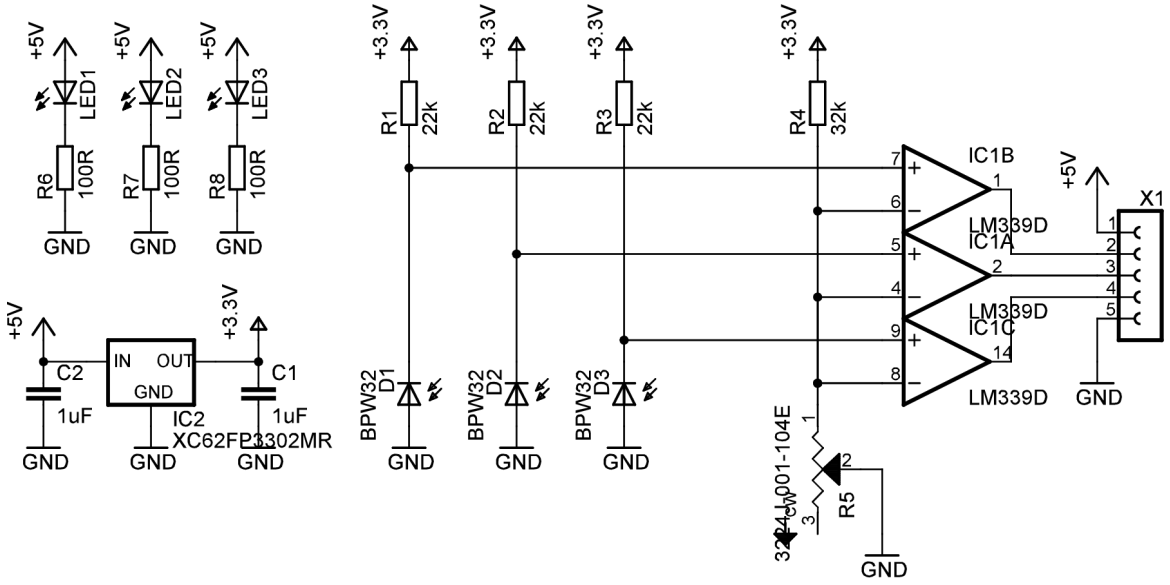


Abbildung 22: Schaltplan des Positionssensors

Die verwendeten Strichgitter sind in Bild 23 zu sehen. Sie wurden mittels eines Fotoplotters am Intitut für Messtechnik aus Gerber-Dateien erstellt. Dazu wurde ebenfalls das Softwarepaket EAGLE (mit dem auch das Layout der Elektronik-Prints erstellt wurde) verwendet.

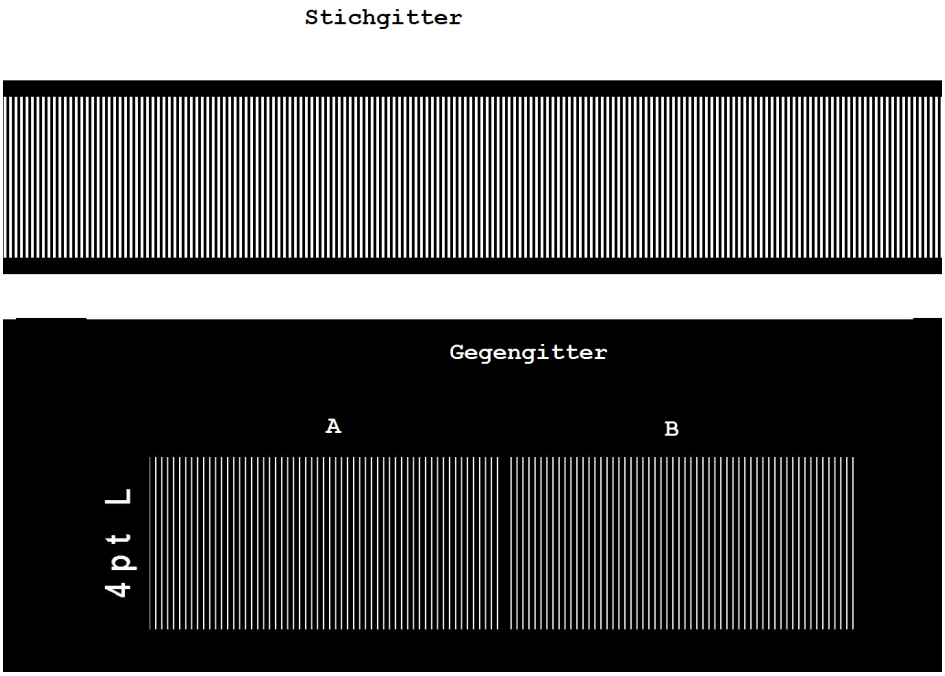


Abbildung 23: Strichgitterstreifen und Gegengitter für den Positionssensor

5.4 Modultest

Die korrekte Funktion der Schaltung und der Schattengitter wurden wie folgt getestet: Die Ausgangssignale der Schaltung wurden mit einem Speicheroszilloskop aufgenommen und sind in Bild 24 dargestellt. Die Schaltschwellen der Komperatoren wurden entsprechend nachjustiert um eindeutige Signale zu erhalten, die problemlos vom Microprozessor verarbeitet werden können.

Die Verarbeitung wurde getestet, indem die Position mittels des Referenzsensors auf Null gesetzt wurde und anschließend der Schlitten von Hand verfahren wurde. Bei erneutem Erreichen des Referenzsensors wurde die Positionsabweichung über RS232 ausgegeben. Die Abweichung waren relativ hoch. Dies stellte sich als ein Problem der Ausrichtung des Strichgitters und des Gegengitters zueinander heraus. Dies wurde korrigiert und danach entsprachen die Ergebnisse den Erwartungen. Die Abweichung lag im Bereich von ± 1 Counts, die auf mechanische Ungenauigkeiten zurückzuführen und tolerierbar ist.

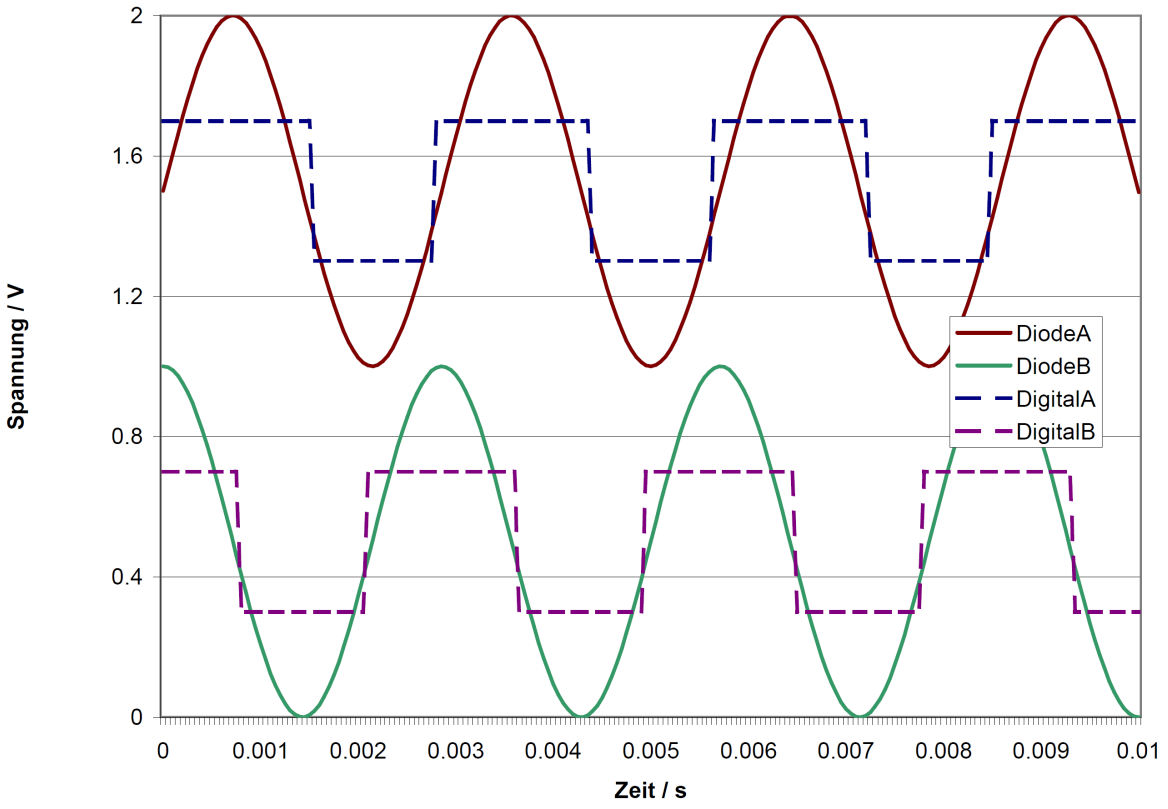


Abbildung 24: Oszilogramme der Positionssignale an den Photodioden und am Prozessoreingang

Weiters wurde ein definierter Verfahrweg mittels eines Messschiebers angefahren und mit dem Positionswert des Counters verglichen. Es stellte sich heraus, dass der Meßwert um $0.5mm$ vom Wert des Messchiebers abwich. Diese Abweichung war aber

bei mehrmaliger Wiederholung konstant. Dies lässt auf ein Problem der Fertigungsgenauigkeit des Strichgitters schließen. Einmal ausgemessen, ließe sich dies in der Software durch einen entsprechend angepassten Skalierungsfaktor von ausgleichen. Auf dies wurde jedoch verzichtet, da für die Komutierung eine so geringe Abweichung über den gesamten Verfahrweg kein Problem darstellt.

Iin Bild 25 ist ein Foto der fertig bestückte Printplatte zu sehen.

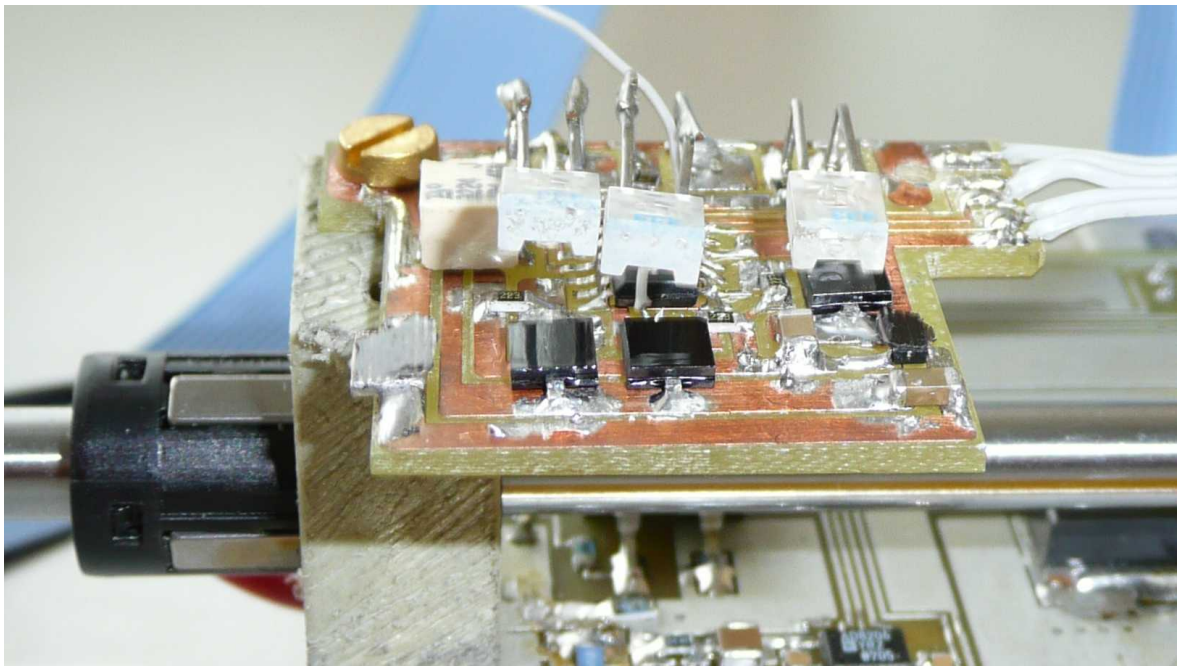


Abbildung 25: Foto des fertigen Positionsgebers ohne Strichgitter

6 Regelkreis

6.1 Grundüberlegungen

Als Reglerstruktur wurde eine Zweifachkaskade von Strom- und Positionsregler gewählt. Der Stromregler wird als PI-Regler ausgeführt. Der Positionsregler als PID-Regler. Der Regler wurde großteils mit den in der DSP-Entwicklungsumgebung vorhandenen Modulen realisiert. Die Reglerparameter sind über die RS232-Schnittstelle veränderbar.

Anfangs stand die Idee im Raum, den gesamten Linearmotor, verknüft mit den Daten der FME-Analyse aus MAXWELL, in Matlab nachzubilden. Dies wurde jedoch wegen des großen Aufwandes und der Tatsache, dass es sich hier nicht primär um eine regelungstechnische Arbeit handelt, nicht durchgeführt.

6.2 Reglerstruktur

Als Ausgangsbasis für den Regler wurde das in der DMC-LIB von TI vorhandene „pidreg3“ PID-Regler Modul mit Anti-Windup verwendet und entsprechend der Anwendung parametriert. Die Struktur ist in Bild 26 zu sehen.
Die Zykluszeit des Stromreglers ist duch die PWM-synchrone Stromabtastung gegeben und entspricht der PWM-Frequenz.

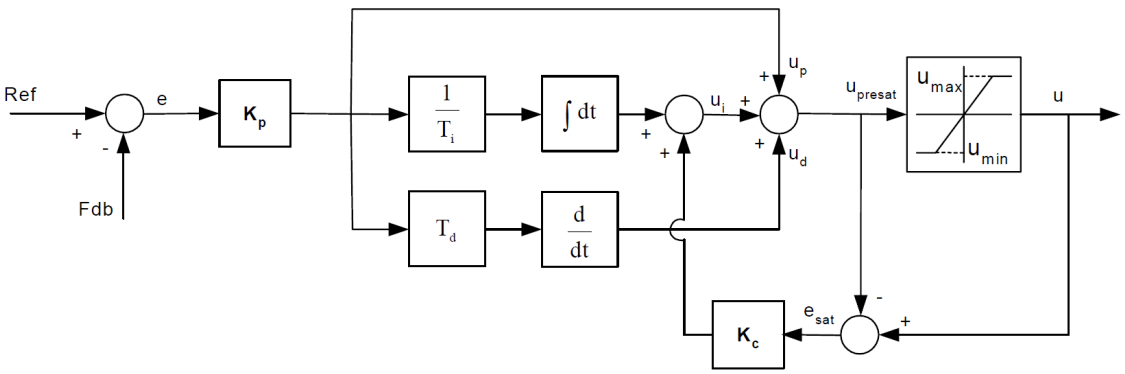


Abbildung 26: Blockdiagramm Regler [10]

Der gesammte Regler besteht aus einer Zweifachkaskade. Unterlagert läuft ein Stromregler und darüber ein Positionsregler, beide mit diesem Modul realisiert, lediglich anders parametriert. In der Standard DMC-Bibliothek wird im äußeren Kreis die Geschwindigkeit geregelt. Da dies in diesem Aufbau mit einem kurzen Fahrweg nicht sinnvoll möglich ist, wurden der Sollwert und das Feedback durch die Positionen statt der Geschwindigkeiten ersetzt und somit zu einem Positionsregler umgebaut.

6.3 Reglerentwurf und Implementierung

Der Regler wurde in der Standardform aus der Bibliothek belassen, lediglich die Regelparameter wurden angepasst.

Für den unterlagerten Stromregler wurde ein PI-Regler verwendet, in dem alle anderen Anteile auf Null gesetzt wurden. Hier würde es sich durch die nahezu ohmsche Streckencharakteristik der Printspulen anbieten, einen reinen I-Regler mit Feed-Forward zu verwenden. Bei diesem würde der Feed-Forward den ohmschen Teil abdecken und der I-Anteil allfällige Regelabweichungen ausgleichen. Es wird jedoch der Einfachheit halber erst versucht, ob die Strecke durch richtiges Parametrieren des Standardreglers unter Kontrolle zu bringen ist.

Der übergeordnete Positionsregler wird als PID-Regler ausgeführt. Hier wird, wie erwähnt, das Feedback von Geschwindigkeit zu Position geändert.

Details der realen Implementierung und Parametrierung werden in Kapitel 7.8 und 7.10 und beschrieben.

7 Software und Inbetriebnahme

7.1 Prozessorwahl

Zur Steuerung wurde der Signalprozessor TMS320F2407 der Firma Texas Instruments gewählt. Dieser Prozessor ist speziell auf die Bedürfnisse in der Antriebstechnik zugeschnitten. Er verfügt über einige Features, welche die Entwicklung von Motoransteuerungen sehr erleichtern:

- bis zu 8 PWM-Kanäle
- 2 Quadratur-Decoder
- softwarekonfigurierbare PWM-Verriegelungszeiten
- schnelle hardwaremäßige Sicherheitsschaltungen
- Kommunikationsschnittstellen wie CAN oder synchrone und asynchrone serielle Verbindungen
- synchrone Analogeingänge zur Istwerterfassung
- intgrierter RAM und FLASH Speicher
- Software-Bibliotheken zur Motoransteuerung (PID-Regler, Koordinatentransformation, etc.)

Dies erleichtert den Entwurf der Ansteuerungssoftware signifikant und verringert auch den Aufwand an zusätzlich notwendiger Hardware zur Ansteuerung.
Für den Aufbau wurde das Evaluationsboard der EZDSP-Serie von SpectrumDigital verwendet, welches an der Parallelschnittstelle eines PCs via JTAG programmiert und debugt werden kann. Inkludiert ist auch die Code-Composer-Entwicklungsumgebung. Weiters bietet die Firma Texas-Instuments einen großen Pool an Referenzbeispielen und fertigen Funktionsblöcken an, die ein schnelles Einarbeiten ermöglichen. Besonders zu erwähnen ist hier die DMC[4]-Bibliothek [10].

7.2 Inbetriebnahme des Evaluationsboardes

Bei der Inbetriebnahme der Entwicklungsumgebung gab es anfangs Schwierigkeiten mit der Installation und Konfiguration der JTAG-Treiber für das Evalboard. Dies konnte jedoch durch ein Software-Update behoben werden.

Als nächstes wurde, nach der Registrierung auf der TI-Homepage, die DMC-Lib heruntergeladen und installiert.
In der Beispielsoftware aus der Bibliothek muss der korrekte DSP-Typ mit *#define TARGET F2407* und die Clockfrequenz mit *SCSR1=0x20c* entsprechend der Hardware konfiguriert werden. Anschließend konnte die Software ein erstes Mal erfolgreich kompiliert werden. Nach Einfügen des Codeteils für die serielle Kommunikation gab es

[4]DMC = Digital Motor Control

jedoch Probleme beim Linken des Codes, da zu wenig Speicher in der Sektion .TEXT vorhanden war. Daher musste die Speicheraufteilung angepasst werden, was eine recht tiefe Einarbeitung in die Architektur des Prozessors erforderte. In diesem Zuge wurde auch der nicht benötigte RT-Monitor Code aus der Software entfernt um Platz zu schaffen.

Listing 1: Linker File mit angepasster Speicheraufteilung

```
-stack 090h

MEMORY
{
      PAGE 0:    /* Program Memory */
        VECS:       org=0000h,          len=00040h        /* external SRAM  */
        PROG:       org=0040h           len=0FDC0h        /* external SRAM  */

      PAGE 1:    /* Data Memory     */
        B0B1:       org=0200h,          len=01F0h         /* internal DARAM */
        B2:         org=0060h,          len=0020h         /* internal DARAM */
        DATA:       org=4000h,          len=4000h         /* EMIF*/
}

SECTIONS
{
        vectors >       VECS            PAGE 0

        .text   >       PROG            PAGE 0
        .cinit  >       PROG            PAGE 0
        .switch >       PROG            PAGE 0

        .const  >       B0B1            PAGE 1
        .bss    >       B0B1            PAGE 1
        .sysmem >       B0B1            PAGE 1

        .data   >       B0B1            PAGE 1
        .stack  >       B0B1            PAGE 1

        /* Additional sections for real time monitor */

        mon_pge0   : { } > B2     PAGE 1              /* Used by RT monitor    */
        mon_rgst   : { } > B2     PAGE 1              /* Used by RT monitor    */
        blk_b0     : { } > B0B1   PAGE 1              /* Stack space           */
        mon_main   : { } > PROG   PAGE 0              /* Main Monitor program  */
}
```

7.3 Softwarestruktur

Die Grundstruktur des Programms ergibt sich aus den folgenden Hauptaufgaben der Software, die auch im Bockdiagramm der DMC-Lib Beschreibung[11] in Bild 27 abgebildet sind. Diese muss entsprechend der Hardware parametriert werden. Die Beispielsoftware beinhaltet eine Geschwindigkeitsregelung, für den Linearmotor wird eine Positionsregelung benötigt. Diese Anpassung ist einfach zu realisieren. Es müssen lediglich statt der eingetragenen Soll- und Istwerte der Geschwindigkeit Positionen verwendet werden. Die fehlende Kommunikation zu einem Host-PC muss selbst geschrieben werden. Die Funtionsweise der Software-Blöcke wird in den foglenden Unterkapiteln genauer beschrieben.

Module der DSP-Software:

- Generierung der PWM-Signale
- Koordinatentransformation und Erzeugung des Drehfeldes
- Strommessung und Koordinatentransformation
- Implementierung eines Srom-Reglers

- Positionsmessung zur Kommutierung und Regelung
- Implementierung eines Positions-Reglers
- Fehlerüberwachung, Überstromabschaltung
- Kommunikation über die RS232-Schnittstelle

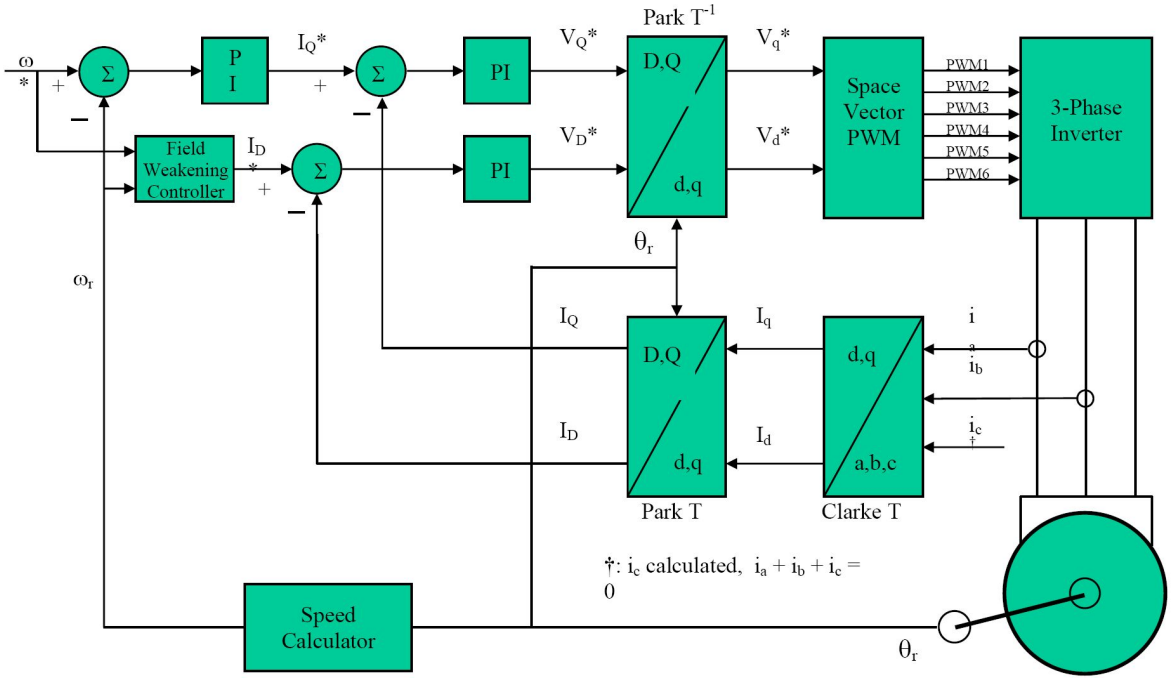


Abbildung 27: Blockdiagramm der Ansteuerungssoftware

7.4 Hauptprogramm

Das Hauptprogramm hat folgenden Aufgaben:

- Konfiguration der DSP-Hardware
- Initialisierung und Parametrierung der Softwarekomponenten
- Aufsetzen der Interrupts für die Regelung und Kommunikation
- Ausführen der Kommunikation im Hintergrundprogramm

Das Hauptprogramm läuft in einer Endlosschleife, die das Kommunikationsprotokoll abhandelt. Sie wird durch den PWM-Interrupt für die Strom- und Positionsregelung unterbrochen. Auch der Empfangsinterrupt der Kommunikationsschnittstelle, der niedriger als der PWM-Interrupt priorisiert ist, kann das Hauptprogramm unterbrechen.

Die detaillierte Beschreibung der einzelnen Software-Module folgt in den nachfolgenden Kapiteln.

7.5 PWM-Generierung

Die PWM-Ausgangssignale werden von einem weiteren Timer generiert, welcher durch Auf- und Abwärtszählen ein fiktives Dreiecksignal erzeugt. Dieser Zähler wird mit drei Compare-Registern verglichen, die je nach „größer“ oder „kleiner“ die PWM-Ausgänge schalten. Der DSP verfügt auch über je einen komplementären PWM-Ausgang mit programmierbarer Todzeit zwischen den Signalen. Dieser wird hier nicht verwendet, da die Todzeitgenerierung in den MosFET-Treibern selbst geschieht (siehe Kapitel 3.3).

Die Schaltfrequenz wird möglichst hoch gewählt, sodass sie außerhalb des Hörbereichs zu liegen kommt und der Stromrippel gering gehalten wird. Die oft limitierenden Schaltverluste in den MosFETs müssen in dieser Leistungsklasse nicht besonders beachtet werden. Allerdings wird systembedingt eine höhere Ausgangsfrequenz durch Verringerung des Wertebereichs des Zählers und somit der Auflösung der PWM erkauft. Um das System noch regelbar zu halten, sollte diese nicht zu gering gewählt werden.

Der Überlauf des PWM-Registers löst auch den Interrupt für die Strommessung und die Stromregelung aus, da dann der Strommesswert zu einem definierten Zeitpunkt in der halben Anstiegszeit gelesen wird und auch nicht durch Schaltvorgänge gestört ist. Eine schnellere PWM führt daher auch zu mehr Prozessorbelastung, wobei hier die Möglichkeit bestünde, den Stromregler nur jeden zweiten Interrupt ablaufen zu lassen.

Die hier gewählten 1000 Counts ergeben einen guten Kompromiss aus Ausgangsfrequenz, PWM-Auflösung und CPU-Belastung.
Zum Test wurden die entsprechenden Register des DSPs konfiguriert und die PWM-Signale vor und nach den Endstufen am Oszilloskop überprüft. Dies ist nach dem Starten und Initialisieren der Software bei angehaltenem Programm durch direktes Beschreiben der entsprechenden Compare-Register im Watchwindow der Entwicklungsumgebung möglich. Die Anpassungen in der Bibliothek sind im Listing 2 beschrieben. In Folge kann dann das Beispielprojekt im BUILDLEVEL1 gestartet werden, und an den drei Ausgängen das dreiphasige Drehfeld beobachtet werden. Um die PWM-Level am Oszilloskop zu beobachten, eignet sich ein am Ausgang angebrachter einfacher RC-Tiefpass mit einer Grenzfrequenz deutlich unter der PWM-Frequenz.

Listing 2: Code für die PWM-Ansteurung

```
/*------------------------------------------------------------------------------
Code-Part to Configure the DMC-LIB to the defaults needed for LinMot Thesis
------------------------------------------------------------------------------*/

/*------------------------------------------------------------------------------
Define the structure of the PWM Driver Object
------------------------------------------------------------------------------*/
  typedef struct {
    int period_max;         /* PWM Period in CPU clock cycles.  Q0-Input  */
    int mfunc_p;            /* Period scaler. Q15 - Input                 */
    int mfunc_c1;           /* PWM 1&2 Duty cycle ratio. Q15, Input       */
    int mfunc_c2;           /* PWM 3&4 Duty cycle ratio. Q15, Input       */
    int mfunc_c3;           /* PWM 5&6 Duty cycle ratio. Q15, Input       */
    int (*init)();          /* Pointer to the init function               */
    int (*update)();        /* Pointer to the update function             */
  } PWMGEN ;

/*------------------------------------------------------------------------------
Define a PWMGEN_handle
------------------------------------------------------------------------------*/
  typedef PWMGEN *PWMGEN_handle;

/*------------------------------------------------------------------------------
Default Initializers for the F2407 PWMGEN Object
```

```
-> 20kHz PWM on EZLF24007a
-----------------------------------------------------------------------------*/
  #define PWMGEN_SCRO {
    1000,    \
    0x7fff,  \
    0x4000,  \
    0x4000,  \
    0x4000,  \
    (int (*)(int))F2407_EV1_PWM_Init,   \
    (int (*)(int))F2407_EV1_PWM_Update  \
  }

/*-----------------------------------------------------------------------------
Define this PWMGEN as default at startup
-----------------------------------------------------------------------------*/
  #define PWMGEN_DEFAULTS PWMGEN_SCRO
```

7.6 Koordinatentransformation

Die Koordinatentransformation wird verwendet, um den Motorstrom in den drei Phasen in eine momentbildende-(Q)- und eine flussbildenden-(D)-Komponente umzurechnen. Diese werden dann von jeweils 2 Reglern auf dem Sollwert gehalten. Der Q-Anteil wird dabei auf das geforderte Drehmoment und der D-Anteil auf 0 geregelt.

Anschließend wird der damit entstandene Spannungssollwert in das Koordinatensystem U-V-W-Koordinatensystem des Motors umgerechnet und entsprechend der Rotorlage gedreht. Dies sind die Werte, die als PWM-Stellgrößen ausgegeben werden. Um auf Schwankungen der Versorgungsspannung reagieren zu können, wird zusätzlich noch die Versorgungsspannung gemessen. Spannungsschwankungen können damit durch Adaption des PWM-Sollwertes ausgeglichen werden. Auf eine genaue Beschreibung wird hier verzichtet, da die Details und mathematischen Grundlagen hierfür in der DMC-Beschreibung [11] oder einschlägigen Literatur [4] [9] oder [6] nachgelesen werden können.

Diese Struktur ist im Beispielprojekt bereits aufgebaut und kann mit BUILDLEVEL=2 getestet werden. Dabei wird ein Drehfeld erzeugt, dessen Geschwingigkeit über eine Rampe nach oben und unten gefahren werden kann. Um die Ansteuerung besser testen zu können wurde statt der Printspulen ein externer Dreiphasenmotor angeschlossen. Dieser lief wie gewünscht hoch, jedoch blieb er bei höherer Drehzahl stehen und begann zu vibrieren. Dies lässt sich auf die sehr niedrige Versorgungsspannung zurückführen, die, bedingt durch die Gegen-EMK, die Maximaldrehzahl begrenzt. Duch Erhöhen der Versorgungsspannung konnten dann auch höhere Drehzahlen erreicht werden.

Somit ist sichergestellt, dass dieser Codeteil funktioniert. Parameter mussten hierfür keine angepasst werden.

7.7 Strommessung

Die Messung des Phasenstromes wird durch den zur PWM synchronisierten Analog-Digital-Converter durchgeführt. Dieser tastet den Strom bei jedem Überlauf des PWM-Zählers ab.

Um die Messwerte zu kalibrieren wurde die Software mit BUILDLEVEL=2 gestartet, ohne dass die Spulen angeschlossen waren. Der zu messende DC-Strom wurde aus einem strombegrenzenden Labornetzteil eingespeist und mit einem Multimeter gemessen. Es wurde eine Strom - ADC-Wert Tabelle aufgenommen und daraus der Skalierungsfaktor des Softwaremoduls für die Strommessung ermittelt. Die Messwerte im DSP wurden auf $0.1mA$ skaliert. Dies ist zum einen ein gut lesbarer Wert und entspricht bei einem Wertebereich von $\pm 2A$ und einer Auflösung von 12Bit einem ADC-Count. Es wird somit der Analog-Digital-Wandler möglichst gut ausgenutzt.

Die Skalierungsfaktoren von ilg2.gain_a = -5734 und ilg2.gain_a = -6389 wurden in der Software eingetragen und erneut die skalierten Werte Ia und Ib mit denen des Multimeters verglichen. Die negativen Faktoren sind nötig um die Polarität der Ströme, die durch einen Fehler im Schema ein falsches Vorzeichen haben, zu korrigieren.

Nachdem diese den Erwartungen entsprachen, wurden die Stromsensoren und die Printspulen über eine Drahtschlaufe angeschlossen. An dieser wurde die Strommesszange des Oszilloskops angeschlossen. Zusätzlich wurde auch die Phasenspannung und der Ausgang des Stromsensors angezeigt. Durch Anhalten des Drehfeldes und Variation der PWM-Amplitude wurden nun der Mittelwert des Stroms am Oziloskop mit dem Messwert im DSP verglichen. Besonders von Interesse war hier das Verhalten des Strom-Messwerts im Vergleich zum Strom-Mittelwert der mit dem Oszilloskop gemessen wurde, da der Strom, bedingt durch die geringe Induktivität der Spulen, wie erwartet einen großen Ripple hatte.

Es zeigte sich, dass der Filter des Stromsensors gut funktioniert und tatsächlich recht genau der Strom-Mittelwert gemessen wurde. Siehe hierzu Oszillogramme im Bild 28.

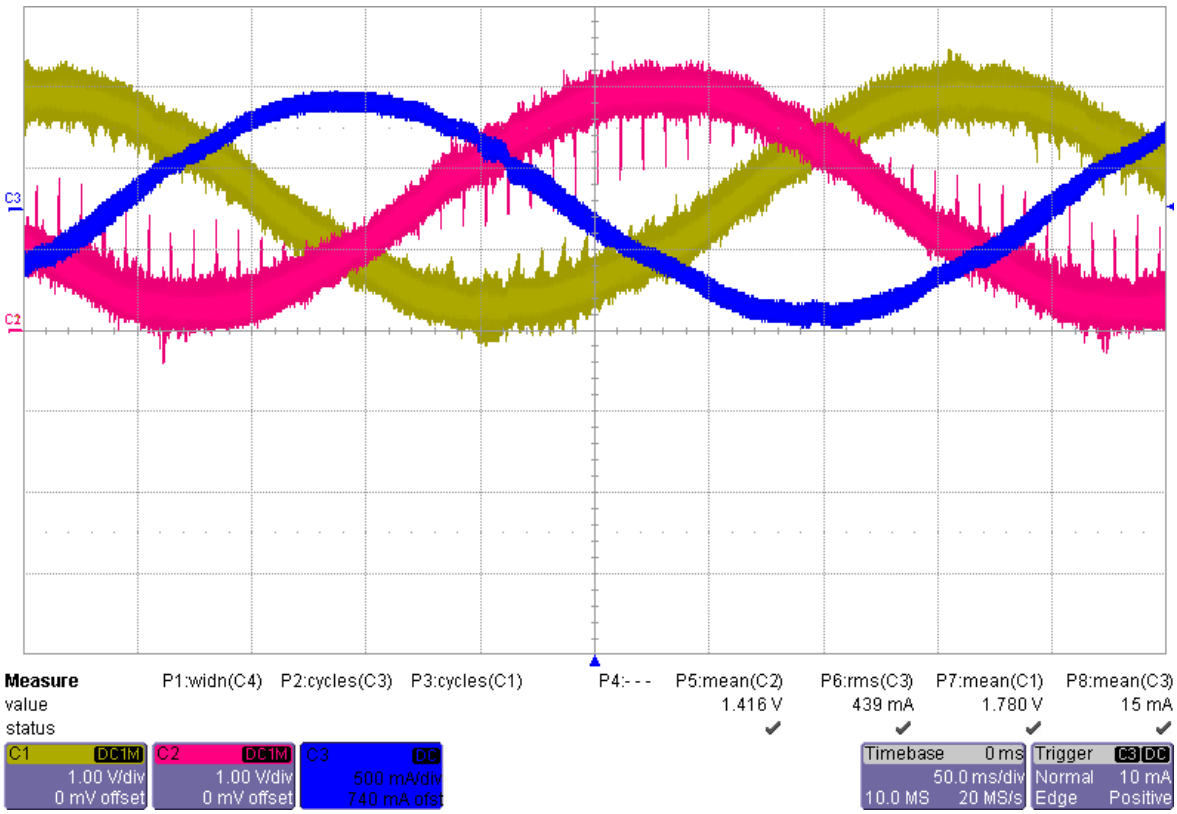


Abbildung 28: Oszillogramm der Stromwerte: gelb und rot - Ausgangsspannung der Stromsensoren, blau - Stromwert der 3.Phase gemessen mit Stromzange

Eine weitere Frage war die Auswirkung des Filters auf auf die Sprungantwort des Stroms. Dazu wurde die Phase über einen Schalter getrennt. Auf dem Oszilloskop wurde so die Sprungantwort des Messwertes der Strommesszange und des Stromsensors verglichen siehe Bild 29.
Der Sensorstrom war wie erwartet langsamer. Bei einer Regelung mit 20kHz beträgt die Periode $50\mu s$. In der Messung ist ersichtlich, dass der Stromsensor nach ca. $40\mu s$ den korrekten Wert liefert, somit also etwas schneller als der Regler ist. Dies hat den Vorteil einer guten Glättung des Messwertes, der, bedingt durch die geringe Induktivität, ansonsten sehr schwer auszuwerten wäre.

Bei der geforderten maximalen Kommutierungsfrequenz von 56kHz gibt es keine Gründe den Stromregler nicht in dieser Form betreiben zu können. Somit konnte die Inbetriebnahme der Stromregelung erfolgen.

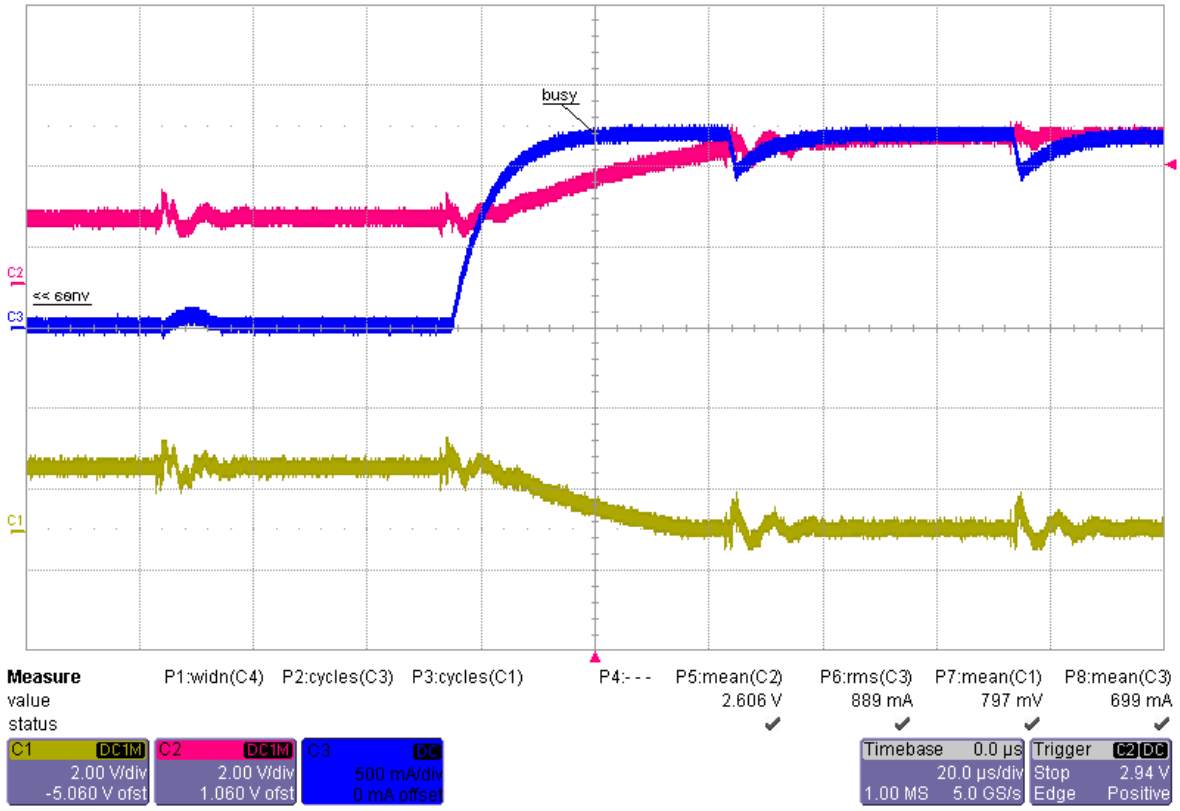


Abbildung 29: Sprungantwort des Stromsensors: gelb und rot - Ausgangsspannung der Stromsensoren, blau - Stromwert der 3.Phase gemessen mit Stromzange

Listing 3: Parametrierung der Strommessung

```
/*-------------------------------------------------------------------------------
Code-Part to Configure the DMC-LIB to the defaults needed for LinMot Thesis
-------------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------------
Define the structure of the ILEG2MEAS Object
-------------------------------------------------------------------------------*/
  typedef struct {
      int gain_a;         /* Channel A gain :       Q13 : Input */
      int offset_a;       /* Channel A offset       Q15 : Input */
      int out_a;          /* Channel A measurement Q15:  Output*/
      int gain_b;         /* Channel B gain :       Q13 : Input */
      int offset_b;       /* Channel B offset       Q15 : Input */
      int out_b;          /* Channel B measurement Q15:  Output*/
      int ch_a;           /* Channel A ADC channel Q0:see NOTE1*/
      int ch_b;           /* Channel B ADC channel Q0:see NOTE1*/
      int (*init)();      /* Pointer to the init function      */
      int (*read)();      /* Pointer to the read function      */
  } ILEG2MEAS ;


/*-------------------------------------------------------------------------------
    Initialize ILEG2MEAS to LinMot Values online
-------------------------------------------------------------------------------*/
  ilg2.init(&ilg2);

  //configure the Current scale to 0.1mA/count
  ilg2.gain_a = -5734;//0.7 Q13 -> Strom in +-1A
  ilg2.gain_b = -6389;//0.78 Q13 -> Strom in +-1A
```

7.8 Stromregler

Die Regelstrecke des Stromreglers kann im vereinfachten Ersatzschaltbild als RL-Serienschaltung dargestellt werden. Dies entspricht einer Strecke 1. Ordnung (TP1). Zur Regelung einer

solchen genügt ein PI-Regler. Der Regelalgorithmus hierfür ist in DMC-LIB vorhanden und muss lediglich noch parametriert werden.

Um fürs Erste Schwierigkeiten durch die geringe Induktivität der Printspulen zu umgehen, wurde der Regler mit BUILLEVEL=3 erst mit einem externen Motor getestet. Es wurden erst der P- dann der I-Anteil des Reglers schrittweise erhöht bis die Regelung den Stromwert gut ausregelte, und z.B. gegen Variationen der Versorgunsspannung unempfindlich war. Anschließend wurde die Drehung des Stromvektors angehalten und der Stromsollwert variiert.

Nachdem dies alles funktionierte, wurden wieder die Printspulen angeschlossen und das zuvor beschriebene Verfahren auch damit versucht.
Wie erwartet, stellte sich das Einstellen des Reglers hier als deutlich schwieriger heraus. Die geringe Induktivität erschwerte das Einstellen der Parameter enorm.
Da es sich um eine hauptsächlich ohmsche Regelstrecke handelt, wäre hier der Ansatz eines Feed-Forward-Pfades zur Abdeckung des ohmschen Anteiles kombiniert mit einem I-Regler zum langsamen Ausgleich des allfälligen Regelfehlers ein besserer Ansatz.

Da die Regelung letztlich doch mit den Standard-PI-Regler aus der Bibliothek in den Griff zu bekommen war, wurde darauf verzichtet. Insbsondere, da der Regler aufgrund der niedrigen Zykluszeit in Assembler und mit Fixkommazahl-Arithmetik geschrieben werden müsste, was erfahrungsgemäß recht tückisch sein kann

Listing 4: Parametrierung des Strommreglers

```
/*-------------------------------------------------------------------------
Code-Part to Configure the DMC-LIB to the defaults needed for LinMot Thesis
-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------
Define the structure of the PID3 Object
-------------------------------------------------------------------------*/
  typedef struct {
    int  pid_ref_reg3;          /* Input: Reference input (Q15) */
    int  pid_fdb_reg3;          /* Input: Feedback input (Q15) */
    int  e_reg3;                /* Variable: Error  (Q14)  */
    int  Kp_reg3;               /* Parameter: Proportional gain (Q15) */
    int  up_reg3;               /* Variable: Proportional output (Q14) */
    int  ui_hi_reg3;            /* Variable: Integral output  (Q30)  */
    int  ui_lo_reg3;            /* Variable: Integral output  (Q30)  */
    int  ud_lo_reg3;            /* Variable: Derivative output  (Q30)  */
    int  ud_hi_reg3;            /* Variable: Derivative output  (Q30)  */
    int  uprsat_reg3;           /* Variable: Pre-saturated output (Q14) */
    int  pid_out_max;           /* Parameter: Maximum output  (Q15)  */
    int  pid_out_min;           /* Parameter: Minimum output  (Q15)  */
    int  pid_out_reg3;          /* Output: PID output  (Q15)  */
    int  saterr_reg3;           /* Variable: Saturated difference (Q14) */
    int  Ki_reg3;               /* Parameter: Integral gain  (Q31-16bit) */
    int  Kc_reg3;               /* Parameter: Integral correction gain (Q15) */
    int  Kd_reg3;               /* Parameter: Derivative gain  (Q14)  */
    int  up1_reg3;              /* History: Previous proportional output (Q14) */
    int  (*calc)();             /* Pointer to calculation function */
  } PIDREG3;

/*-------------------------------------------------------------------------
    Set the current controllers for LinMot online
-------------------------------------------------------------------------*/
  foc.pid_id.Kp_reg3 = 16312;
  foc.pid_iq.Kp_reg3 = 16312;
  foc.pid_id.Ki_reg3 = 32399;
  foc.pid_iq.Ki_reg3 = 32399;
  foc.pid_id.Kc_reg3 = 32399;
  foc.pid_iq.Kc_reg3 = 32399;
  foc.pid_id.Kd_reg3 = 0;
  foc.pid_iq.Kd_reg3 = 0;
  foc.Id_ref = 0;
  foc.Iq_ref = 16384;
  foc.speed_ref = 742;//10000 = max speed bei 18mm polabstand = 18ms pro periode
```

7.9 Positionsmessung

Die Positionsmessung ist Dank der guten Unterstützung des Prozessors sehr einfach zu realisieren.
Ein Zähler wird so konfiguriert, dass er durch die Quadratur-Eingänge (=QEP) mit der integrierten Logikschaltung angesteuert wird. Weiters wird der Interrupt des Referenzsignaleingangs zum Nullsetzen des Zählers verwendet. Die 16 Bit Zählerbreite ergeben 65536 Stufen und sind somit groß genug die ganze Strecke von $350mm$ bei einer Auflösung von $0,1mm$ ohne Überlauf abzudecken. Der Zähler wird beim Start auch auf einen genügend hohen Wert gesetzt, damit es bei der Initialisierung zu keinen Überlaufproblemen kommt. Somit kann auf eine Überlaufbehandlung verzichtet werden.

Auf eine spezielle Skalierung der Positions-Messwerte kann verzichtet werden, da die hardwaremäßige 0.1mm-Teilung zur Regelung gut verwendbar ist. Die Funktion kann mit BUILDLEVEL=4 getestet werden.

Die Funktion des Quadratureingangs wurde zu Testzwecken dazu genützt, einen kontrolliert drehenden Stromvektor zu erstellen. Dazu wurde ein externer Drehgeber angeschlossen, und dessen Winkelposition als Sollwinkel für den Stromvektor vorgegeben. Somit konnte der Motor durch Drehen des Drehgebers bereits das erste Mal fahren und die Kommutierung getestet werden. Bis auf Probleme mit dem sich verklemmenden Schleppkabel zum DSP-Board konnte bei genügend hohem Sollstrom recht gut verfahren werden.

Um die Positionsregelung aktivieren zu können, musste noch der Kommutierungswinkel eingestellt werden. Aufgrund der mechanischen Konstruktion lag dieser bei 18mm, also bei 180Counts des Inkrementalgebers. Die richtige Phasenlage wurde durch das Fahren an den Endanschlag und manuellem Variieren des Phasenwinkels experimentell ermittelt. Der Phasenwinkel mit der größten Kraft wurde notiert. Anschliessend wurde der Abstand des Anschlages zum Referenzsensor-Signal anhand der Positionsdifferenz des Counters gemessen. Dieser Wert wird nun bei jedem Referenzsignal-Interrupt als Istposition festgelegt.

Um die Referenzposition zu erreichen, wird der Motor mit einem drehenden Feld in Richtung Sensor bewegt. Wenn dieser erreicht wurde, und smoit die Absolutposition bekannt ist, wird der Motor auf Positionsregelung umkonfiguriert. Die Funktion dieses Verfahrens wurde durch mehrmalige Versuche verifiziert. dieses Verfahrens wurde durch mehrmalige Versuche erfolgreich verifiziert.

7.10 Positionsregler

In der Beispielsoftware ist ein Geschwindigkeitsregler vorgesehen. In dieser Aufgabe ist jedoch ein Positionsregler, der den Sollwert für den Stromregler vorgibt, nötig. Für die hier geforderte einfache Positionierungsaufgabe eignet sich der vorhandene PID-Regler

aber ebenfalls. (Regelalgorithmus siehe Kapitel 6.3) Es mussten lediglich die Soll- und Istwerte der Geschwindigkeit durch die entsprechenden Positionswerte ersetzt werden.

Der Algorithmus kann deutlich langsamer als der Stromregler ausgeführt werden, da die dominate Zeitkonstante hier wesentlich größer ist. In dieser Implementierung werden die beiden Reglerzyklen gleich schnell betrieben. Ein langsameres Ausführen würde es ermöglichen mehr CPU-Zeit für den Background-Taskmit der Kommunikation oder höhere Stromreglerfrequenzen zu erzielen. Da es diesbezüglich aber keine Probleme gab, wurde darauf verzichtet.

Die Reglerparameter wurden durch empirisches Ausprobieren ermittelt. Das Programmieren einer digitalen Scoping-Funktion zur Aufnahme der Sprungantwort und der nötigen Auswertungssoftware wären zu aufwändig. In diesem Bereich wären aber sicher noch optimierungen möglich. Auch zur Parametrierung der Stromregelung wäre dies von Vorteil.

Listing 5: Parametrierung und Konfiguration des Positionsreglers

```
/*-------------------------------------------------------------------------
Code-Part to Configure the DMC-LIB to the defaults needed for LinMot Thesis
-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------
Define the structure of the PID3 Object
-------------------------------------------------------------------------*/
  typedef struct {
    int  pid_ref_reg3;          /* Input: Reference input (Q15) */
    int  pid_fdb_reg3;          /* Input: Feedback input (Q15) */
    int  e_reg3;                /* Variable: Error  (Q14)  */
    int  Kp_reg3;               /* Parameter: Proportional gain (Q15) */
    int  up_reg3;               /* Variable: Proportional output (Q14) */
    int  ui_hi_reg3;            /* Variable: Integral output  (Q30)  */
    int  ui_lo_reg3;            /* Variable: Integral output  (Q30)  */
    int  ud_lo_reg3;            /* Variable: Derivative output  (Q30)  */
    int  ud_hi_reg3;            /* Variable: Derivative output  (Q30)  */
    int  uprsat_reg3;           /* Variable: Pre-saturated output (Q14) */
    int  pid_out_max;           /* Parameter: Maximum output  (Q15)  */
    int  pid_out_min;           /* Parameter: Minimum output  (Q15)  */
    int  pid_out_reg3;          /* Output: PID output  (Q15)  */
    int  saterr_reg3;           /* Variable: Saturated difference (Q14) */
    int  Ki_reg3;               /* Parameter: Integral gain  (Q31-16bit) */
    int  Kc_reg3;               /* Parameter: Integral correction gain (Q15) */
    int  Kd_reg3;               /* Parameter: Derivative gain  (Q14)  */
    int  up1_reg3;              /* History: Previous proportional output (Q14) */
    int  (*calc)();             /* Pointer to calculation function */
  } PIDREG3;


/*-------------------------------------------------------------------------
    Set the position controller for LinMot online
-------------------------------------------------------------------------*/
  foc.pid_spd.Kp_reg3 = 34392;
  foc.pid_spd.Ki_reg3 = 12379;
  foc.pid_spd.Kc_reg3 = 32399;
  foc.pid_spd.Kd_reg3 = 0;


/*-------------------------------------------------------------------------
Connect PIDREG3 inputs for position control instead of speed control
-------------------------------------------------------------------------*/
    v->pid_spd.pid_ref_reg3 = v->speed_ref;         //speed is position here!
    v->pid_spd.pid_fdb_reg3 = v->shaft_theta_elec;  //use Positon !!!
```

7.11 Kommunikationsprotokoll

Da in der DMC-Bibliothek keine Kommunikation außer mit dem Debgger der Entwicklungsumgebung vorgesehen ist, musste dieser Codeteil komplett selbst geschrieben werden.

Die serielle Schnittstelle wird bei der Initialisierung auf eine Baudrate von 115200 mit 8 Datenbits und einem Stoppbit ohne Handshake-Protokoll konfiguriert. Der Empfang eines Zeichens über die serielle Schnittstelle löst einen Interrupt mit niedrigster Priorität aus, welcher die empfangenen Zeichen zur weiteren Verarbeitung puffert. Die empfangenen Zeichen werden nach einem Carrige-Return im Hintergrundtask ausgewertet. Von dort wird auch die entsprechende Antwort in einen Puffer geschrieben und in jedem Zyklus wird ein Zeichen daraus verschickt.

Es werden folgende Steuerbefehle implementiert:

Funktion	**Befehl**	**PC $\Longleftrightarrow$ Motor**
Version Anfrag	r	$\Longleftarrow$
Version Rückgabe	r[SW-Version]	$\Longleftarrow$
Sollposition setzen	l=*[Position in mm]*	$\Longrightarrow$
Istposition anfragen	l	$\Longleftarrow$
Momentanposition	l*[Position in mm]*	$\Longleftarrow$
Status abfragen	s	$\Longrightarrow$
Ziel erreicht	s0	$\Longleftarrow$
Fahrt aktiv	s1	$\Longleftarrow$
Fehlercode	s[2..7]	$\Longleftarrow$
D-Stromregler P-Faktor setzen	a=*[Wert]*	$\Longrightarrow$
D-Stromregler P-Faktor abfragen	a	$\Longleftarrow$
D-Stromregler I-Faktor setzen	b=*[Wert]*	$\Longrightarrow$
D-Stromregler I-Faktor abfragen	b	$\Longleftarrow$
Q-Stromregler P-Faktor setzen	c=*[Wert]*	$\Longrightarrow$
Q-Stromregler P-Faktor abfragen	c	$\Longleftarrow$
Q-Stromregler I-Faktor setzen	d=*[Wert]*	$\Longrightarrow$
Q-Stromregler I-Faktor abfragen	d	$\Longleftarrow$
Positionsregler P-Faktor setzen	p=*[Wert]*	$\Longrightarrow$
Positionsregler P-Faktor abfragen	p	$\Longleftarrow$
Positionsregler I-Faktor setzen	i=*[Wert]*t	$\Longrightarrow$
Positionsregler I-Faktor abfragen	i	$\Longleftarrow$
Positionsregler D-Faktor setzen	d=*[Wert]*	$\Longrightarrow$
Positionsregler D-Faktor abfragen	d	$\Longleftarrow$

Tabelle 3: Steuerbefehle der seriellen Kommunikation

Listing 6: Code für das serielle Interface

```
/*-------------------------------------------------------------------------
Code-Part to Configure the DMC-LIB to the defaults needed for LinMot Thesis
-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------
String parsing functions needed for Serial Communication
-------------------------------------------------------------------------*/
  #include <stdlib.h>
  #include <ctype.h>

  /*-------------------------------------------------------------------------
  Convert ASCII-Text to Integer
  -------------------------------------------------------------------------*/
  int t2i(const char *st)
  {
```

```
    register const char *fst    = st;
    register       int   result = 0;
    register       char  fc;

    while (_isspace(*fst++));                       // SKIP WHITE SPACE

    if ((fc = *--fst) == '-' || *fst == '+') ++fst;

    while (_isdigit(*fst))
    {
        result *= 10;
        result += *fst++ - '0';
    }
    return (fc == '-') ? -result : result;
}


/*-------------------------------------------------------------------------------
Convert Intege to ASCII-Text
-------------------------------------------------------------------------------*/
#define BUFLEN 20
int itoa(int val, char *buffer)
{
    char           tempc[BUFLEN];
    register char *bufptr;
    register int   neg = val < 0;
    unsigned int   uval = neg ? -val : val;

    *(bufptr = &tempc[BUFLEN - 1]) = 0;

    do {*--bufptr = (uval % 10) + '0';}  while(uval /= 10);
    if (neg) *--bufptr = '-';

    memmove(buffer, bufptr, neg = (tempc + BUFLEN) - bufptr);
    return neg - 1;     // DON'T COUNT NULL TERMINATION
}


/*-------------------------------------------------------------------------------
Send a welcome String at startup
-------------------------------------------------------------------------------*/
void comm_welcome(void){
              ret[0] = 'P';
              ret[1] = 'r';
              ret[2] = 'i';
              ret[3] = 'M';
              ret[4] = 'o';
              ret[5] = 't';
              ret[6] = 0;
              strcpy(serial.tx_string,ret);
              serial.tx_busy = 1;
}

/*-------------------------------------------------------------------------------
Defines of the known serial Commands
-------------------------------------------------------------------------------*/
              #define POSITION 'l'
              #define STATUS   's'
              #define HOME     'h'
              #define SCOPTIME 't'
              #define ERROR    'e'
              #define ENABLE   'o'
              #define SCOPGET  'g'
              #define CURD_P   'a'
              #define CURD_I   'b'
              #define CURQ_P   'c'
              #define CURQ_I   'd'
              #define POS_P    'p'
              #define POS_I    'i'
              #define POS_D    'd'
              #define REVISION 'r'
              #define UNKONWN  'u'
              #define CNTUP    '+'
              #define CNTDWN   '-'
              #define CR       0x0d
              #define END      0x00

/*-------------------------------------------------------------------------------
Handle the Communication in background loop
-------------------------------------------------------------------------------*/
 void communicate(void)
 {
       //handle the TX of Data
       serial.txHandle(&serial);

       //handle the RX of Data
       if (serial.rx_finshed == 1){
          while(serial.rx_busy == 1);          // warten bis Semaphore frei
          serial.rx_busy = 1;                  // Semaphore öffnen
          strcpy(rcv, serial.rx_string);       // string kopieren
          serial.rx_finshed = 0;
          serial.rx_busy = 0;                  // Semaphore frei geben
```

```
            //command finished so parse the Serial communication
            switch (rcv[0]){
              //read only ones
              case REVISION: ret[0] = REVISION; itoa(VERSION, &ret[1]); ret[3] = END; break;
              case CNTUP: ret[0] = '='; itoa(cnt++, &ret[1]);  break;
              case CNTDWN: ret[0] = '='; itoa(cnt--, &ret[1]); break;
              case STATUS: ret[0] = STATUS; itoa(0, &ret[1]); break; //error 0 at first
              //write only ones
              case ENABLE:
                if(rcv[1] == '1')
                  drive.enable_flg=1;
                else
                  drive.enable_flg=0;
              //read&write
              case POSITION:
                if(rcv[1] == 0){
                  ret[0] = POSITION; itoa(speed.theta_elec, &ret[1]);}
                else{
                  speed.speed_rpm = t2i(&rcv[1]);
                  ret[0] = POSITION; itoa(speed.speed_rpm, &ret[1]);}
                break;
              case VELOCITY:
                if(rcv[1] == 0){
                  ret[0] = VELOCITY; itoa(foc.Id_ref, &ret[1]);}
                else{
                  foc.Id_ref = t2i(&rcv[1]);
                  ret[0] = VELOCITY; itoa(foc.Id_ref, &ret[1]);}
                break;
              case CUR_P:
                if(rcv[1] == 0){
                  ret[0] = CUR_P; itoa(foc.pid_iq.Kp_reg3, &ret[1]);}
                else{
                  foc.pid_iq.Kp_reg3 = t2i(&rcv[1]);
                  ret[0] = CUR_P; itoa(foc.pid_iq.Kp_reg3, &ret[1]);}
                break;
              case CUR_I:
                if(rcv[1] == 0){
                  ret[0] = CUR_I; itoa(foc.pid_spd.Ki_reg3, &ret[1]);}
                else{
                  foc.pid_spd.Ki_reg3 = t2i(&rcv[1]);
                  ret[0] = CUR_I; itoa(foc.pid_spd.Ki_reg3, &ret[1]);}
                break;
              default: ret[0] = UNKONWN; ret[1]=0;
            }
            //send out the answer
            //TODO: telegrams might get lost, if comunication is too fast for tx ...
            if(serial.tx_busy != 1){
              strcpy(serial.tx_string,ret);
              serial.tx_busy = 1;
            }
        }
    }


/*-----------------------------------------------------------------------------
The serial Communication Initialisation
-----------------------------------------------------------------------------*/
void inline F2407_SER_Init(SERVALS *p){
    long BR;

    //baudrate calculation for a system clocked with 40Mhz
    BR = 40000000/((p->baudrate*8)-1); //BR = (clk/(baud*8) )-1
    //z.B.: 0x81=38400, 0x103=19200, 0x208=9600

    SCSR1=SCSR1|0x0040; //ENABLE clock to SCI

    // Setup only the GP I/O only for SCI functionality
    MCRA=MCRA|0x0003; /* Set SCI pins */

    //Set registesr for the Serial Communication
    SCICCR =0x0007;   //1 stop bit, No loopback, No parity,8 bits, async mode,
                      //idle-line protocol + reset
    SCICTL1 =0x0003; //enable TX, RX, int SCICLK, Disable RX ERR, SLEEP, TXWAKE
    SCICTL2 =0x02;    //rx interrupts enabeln
    SCIHBAUD = BR>>8;//baudrate;
    SCILBAUD = BR;    //baudrate;
    SCIPRI = 0x78;    //low interrupt priroty, free run, ignore suspend

    SCICTL1 =0x0023; // release reset
}


/*-----------------------------------------------------------------------------
The serial Communication Interrupt
-----------------------------------------------------------------------------*/
void inline F2407_SER_ISR(SERVALS *p){
        while (p->rx_busy == 1);                // warten bis Semaphore frei

        p->rx_busy = 1;                         // Semaphore öffnen
        p->rx_string[p->rx_cnt] = SCIRXBUF;     // Daten kopieren

        if (p->rx_string[p->rx_cnt] == 13)      // Abfrage auf Enter
                {
```

```
                p->rx_finshed = 1;                  // Finish Flag setzen
                p->rx_string[p->rx_cnt] = '\0'; // String abschliessen
                p->rx_cnt = 0;
                }
        else
                p->rx_cnt++;
        if( p->rx_cnt >= MAX_STRING_LENGTH)      // zu langer Befehl
                p->rx_cnt = 0;
        p->rx_busy = 0;                          // Semaphore frei geben

        PIACKR1 = 0x0100; // Clear Interrupt flag
}


/*-----------------------------------------------------------------------------
The serial Communication TX handling (to be called cyclical)
-----------------------------------------------------------------------------*/
void inline F2407_SER_TX_HANDLE (SERVALS *p){
  if (SCICTL2 & 0x0080){                         //if TX-Ready
    if (p->tx_busy == 1){                        //sending active
      if(p->tx_string[p->tx_cnt] == '\0' ){      //end of string?
        p->tx_cnt = 0;
        p->tx_busy = 0;
        SCITXBUF = '\r';                         //Append a Carrige return
      }
      else{
        SCITXBUF = p->tx_string[p->tx_cnt];
        p->tx_cnt++;
      }
    }
  }
}
```

8 Ergebnisse und Ausblick

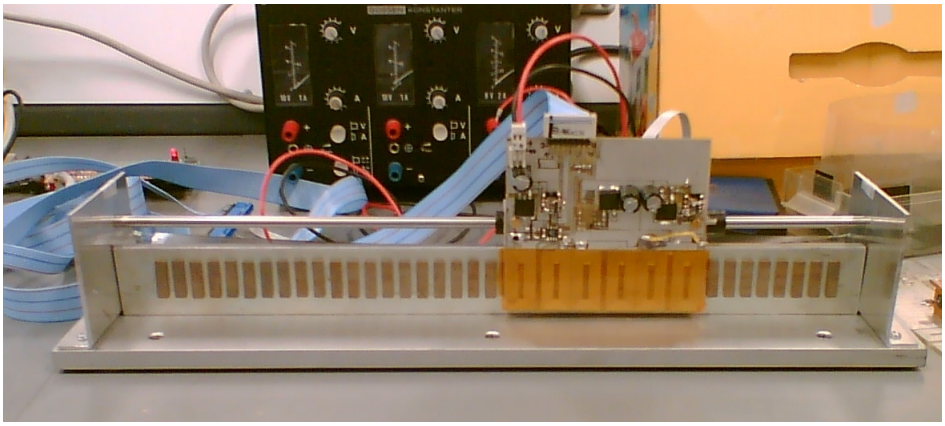


Abbildung 30: Bild des fertigen Vesuchsaufbaues

Abschließend lässt sich folgendes Fazit aus der Arbeit ziehen:

- Ein Linearmotor mit Wicklungen auf der Printplatte ist gut realisierbar.
- Es sollten Multilayer Prints verwendet werden, um die Windungszahl erhöhen zu können.
- Nachteilig ist sicher der hohe Verbrauch an Magnetmaterial.
- Die derzeitige Positionsmessung ist recht fehleranfällig. Es könnte daher z.B. eine integrierte Lösung mittels Messung des Magnetfeldes angestrebt werden.
- Eine höhere Auflösung des Positionssensors würde wahrscheinlich eine deutlich bessere Regelung ermöglichen.
- Eine gute, reibungsarme Führung des Schlittens ist sehr wichtig.
- Die Schleppkabel sollten vermieden werden.
- Der DSP sollte auf das Stromrichter-Board integriert werden, um die breiten und daher starren Schleppkabel zu reduzieren.
- Es sind sicher auch noch Verbesserungen der Regelkreise möglich. Besonders sollte eine Digital-Scoping-Möglichkeit geschaffen werden, um die Regler besser parametrieren zu können. Besonders der Ansatz mit einem Feedforward und I-Regler erscheint interessant und sollte zumindest mittels Simulation verifiziert werden.

Literatur

[1] Allegro. *ACS712 Datasheet*.

[2] ANSOFT. Maxwell. http://www.ansoft.com/maxwell/. Finite-Elemente-Simulation von elektrischen und magnetischen Feldern.

[3] Analog Devices. *AD8120 Datasheet*.

[4] George Ellis. *Control System Design Guide*, volume 3. Elsevier, 2003.

[5] Fairchild. *FDB8441 Datasheet*.

[6] Rolf Fischer. *Elektrische Maschinen*, volume 12. HANSER Verlag, 2004.

[7] Herbert Föller. Leiterbahnströme nach ipc d 275. *IPC*, 1900.

[8] K. Föllner. Pcb therm. http://www.selfmadehifi.de/. V0.4.

[9] Steven Campbell Hamid a.Toliyat. *DSP-Based Electromechanical Motion Control*, volume 1. CRC Press, 2004.

[10] Digital Control Systems (DCS) Group Texas Instruments. Dmc-lib. http://www.ti.com. reference Designs for various Digital Motor Control Projects.

[11] Digital Control Systems (DCS) Group Texas Instruments. Field oriented control of three-phase permanent-magnet synchronous motor. *DMC-LIB*, 2005. Included in DMC-LIB.

[12] Dallas Semiconductor Maxim Integrated Products. *MAX3221 Datasheet*.

[13] Richard F. Post. Toward more efficient transport: The inductrackmaglev system. *Stanford Global Climate and Energy Project*, pages 1–28, 2005.

[14] Michael Witt Rainer Schach. *Proceedings of the 19th International Conference on Magnetically Levitated Systems and Linear Drives*, volume 1. Technische Uni Dresden, 2006.

[15] International Rectifier. *IR2183 Datasheet*.

[16] H. Wakiwaka S.Senoh. Smoother thrust on multi-polar type linear dc motor. *IEEE Transactions Vol.33*, 33:3877 – 3879, 1997.

[17] Bray Velenje. Bray terminal. http://bray.velenje.cx/avr/terminal. Serielles Terminal für Windows.

[18] Harold A. Wheeler. Simple inductance formulas for radio coils. *Proceedings of the I.R.E*, 1928. pp. 1398-1400.

[19] Harold A. Wheeler. Formulas for the skin effect. *Proceedings of the I.R.E*, 1942. pp. 412-424.

[20] Bo Quihnhua Xiong Guangyu. Analytical solutions to determine the field and thrust in a linear servomotor. *Proceedings of the Fifth International Conference on Electrical Machines and Systems ICEMS 2001.*, 2:800 – 802, 2001.

A Abkürzungen, Formelzeichen

Alle Formelzeichen und Abkürzungen beschreiben:

Abküzung	Bedeutung
PWM	Puls Weiten Modulation
PCB	Printed Circuit Board, Gedruckte Leiterplatte
QEP	Quadratue Encoded Position, Incrementalgeber
DSP	Digital Signal Processor
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
DMC	Digital Motion Control
JTAG	Joint Test Action Group, IEEE-Standard 1149.1
LPT	parallele Schnittstelle nach IEEE-1284-Standard
N	Windungszahl einer Spule
TI	Texas Instruments Corp, Halbleiterhersteller
COM1	Asyncrone Serielle Schnittstelle nach RS232
EAGLE	Einfach Anzuwendender Grafischer Layout-Editor
GERBER	Datenaustauschprogramm für CAD-Daten
SMD	Surface Mounted Device
LED	Light Emitting Diode
MAGLEV	magnetic Levitation, Magnetschwebetechnik

B Elektroschemas und Layouts

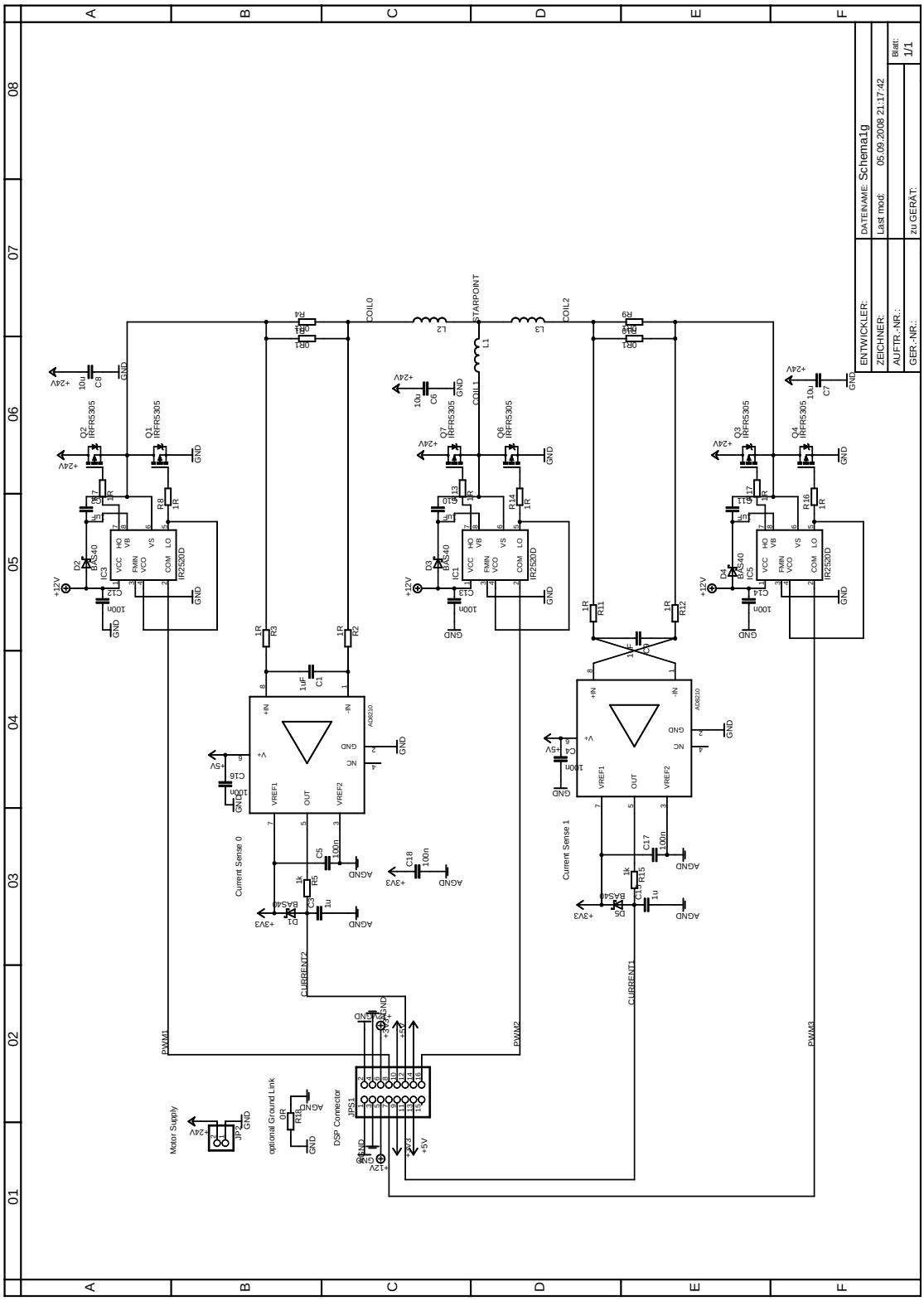


Abbildung 31: Schaltung der Leistungsansteuerung

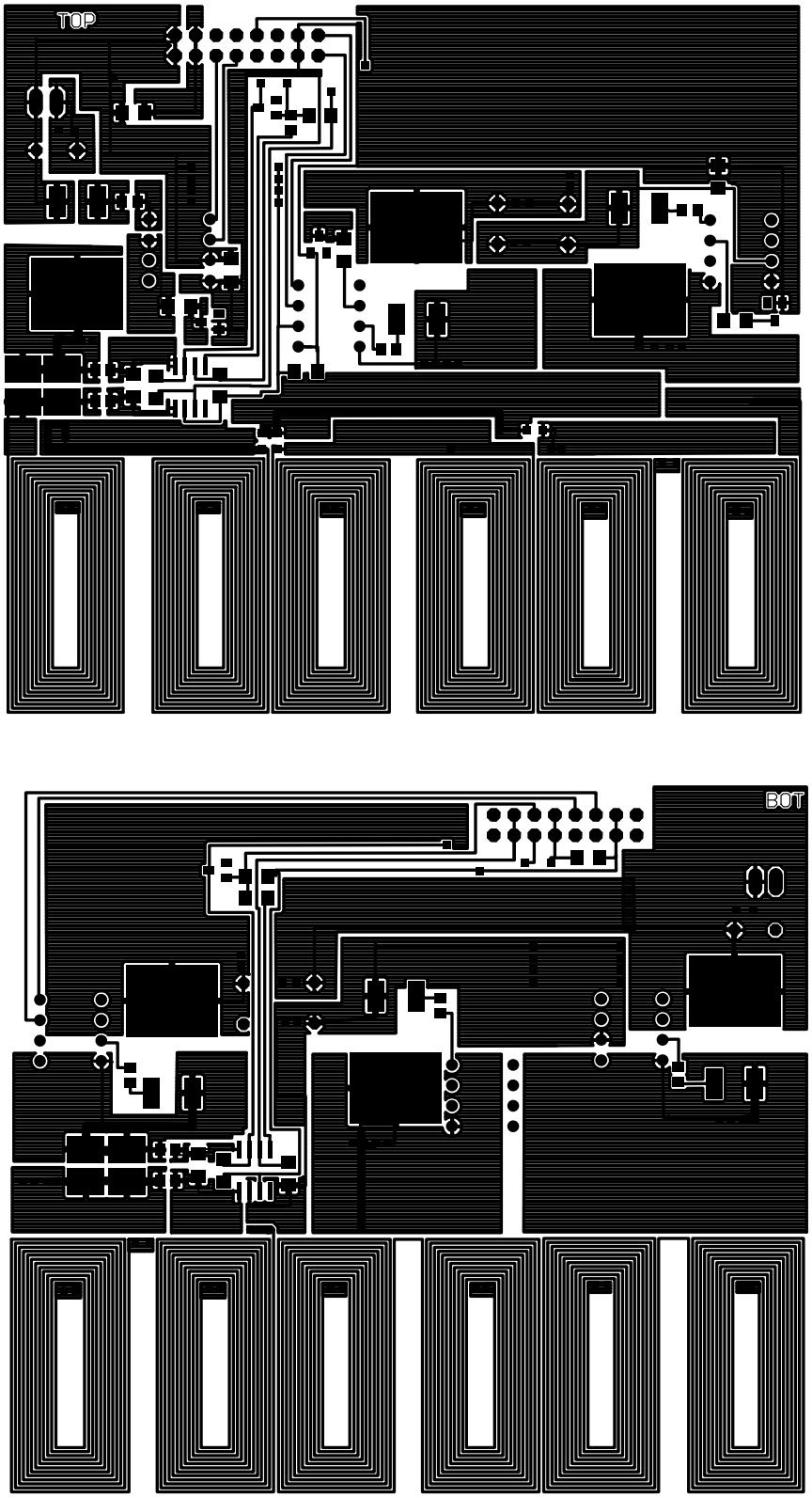


Abbildung 32: PCB-Layouts der Leistungsansteuerung, Top und Bottom

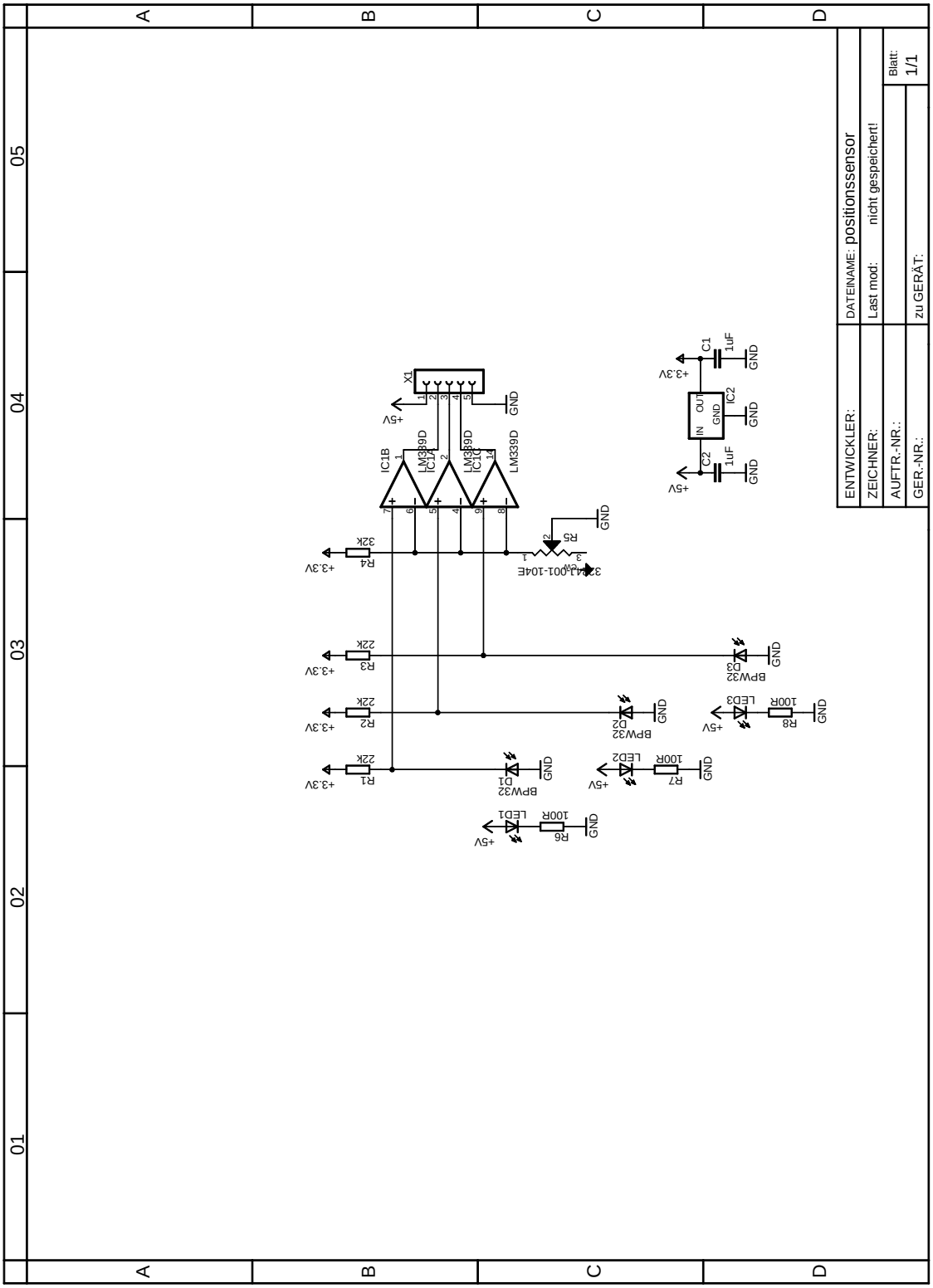


Abbildung 33: Schaltung der Positionsmessung

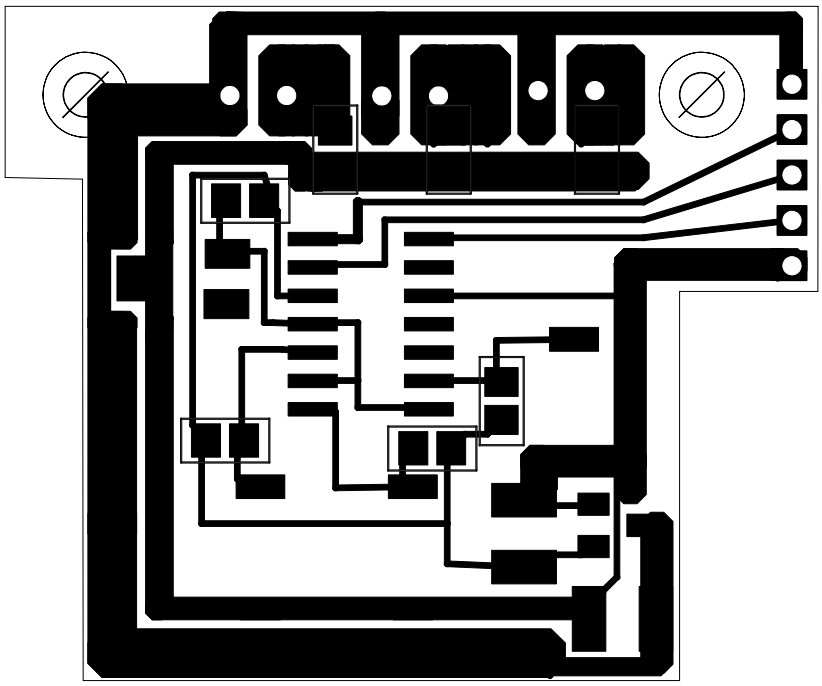


Abbildung 34: PCB-Layouts der Positionsmessung

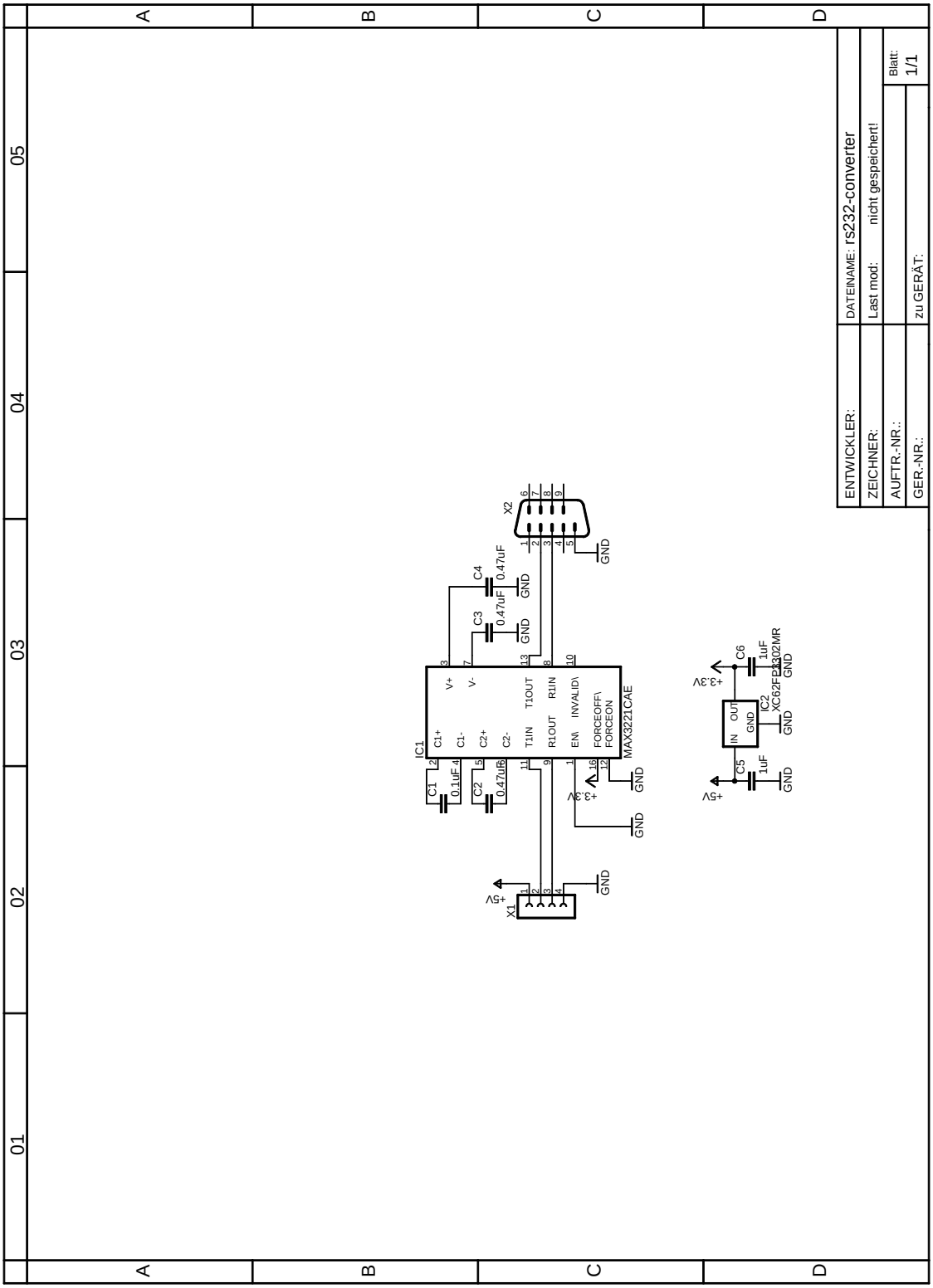


Abbildung 35: Schaltung des Kommunikationsinterface

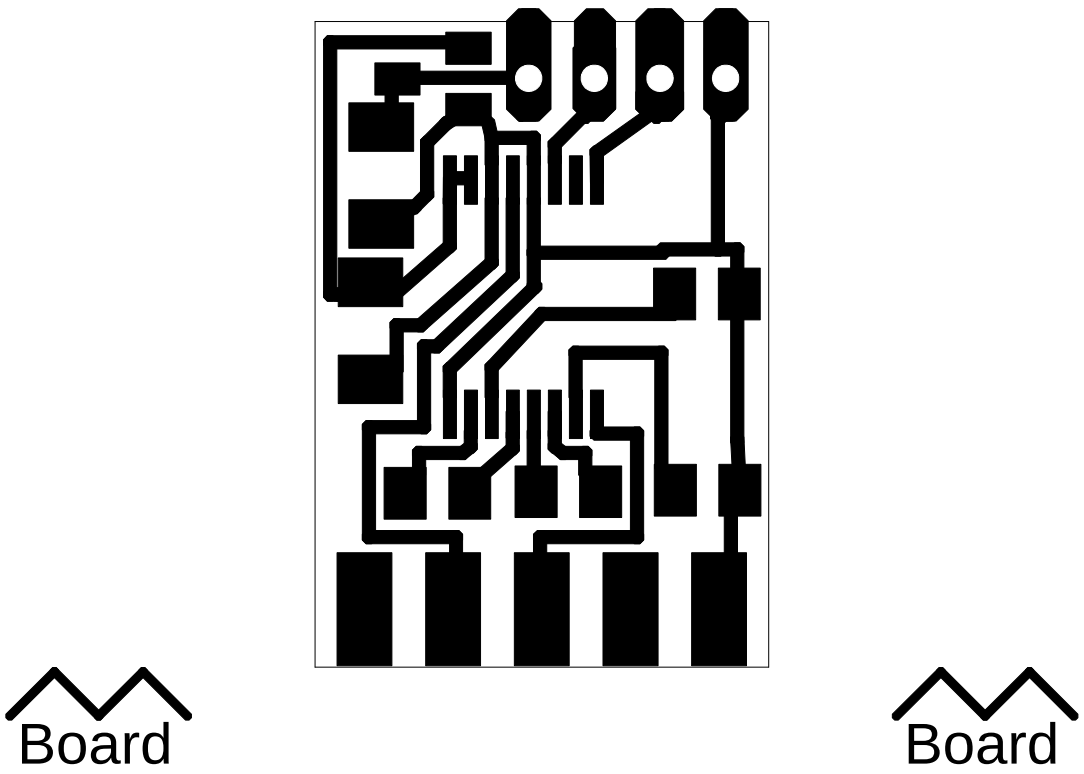


Abbildung 36: PCB-Layouts des Kommunikationsinterface

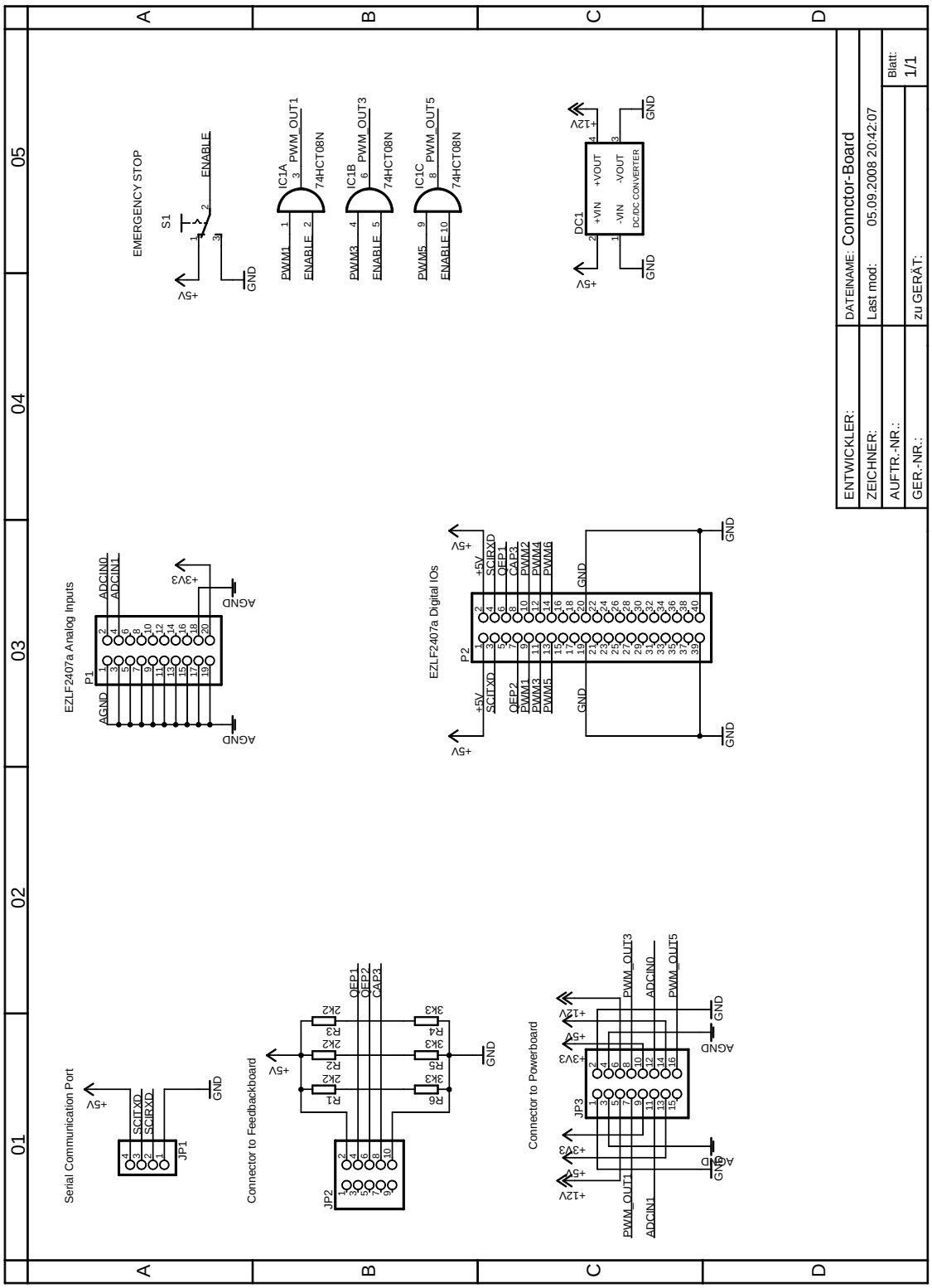


Abbildung 37: Schaltung des Verbindungs-Board

C Mechanische Zeichnungen

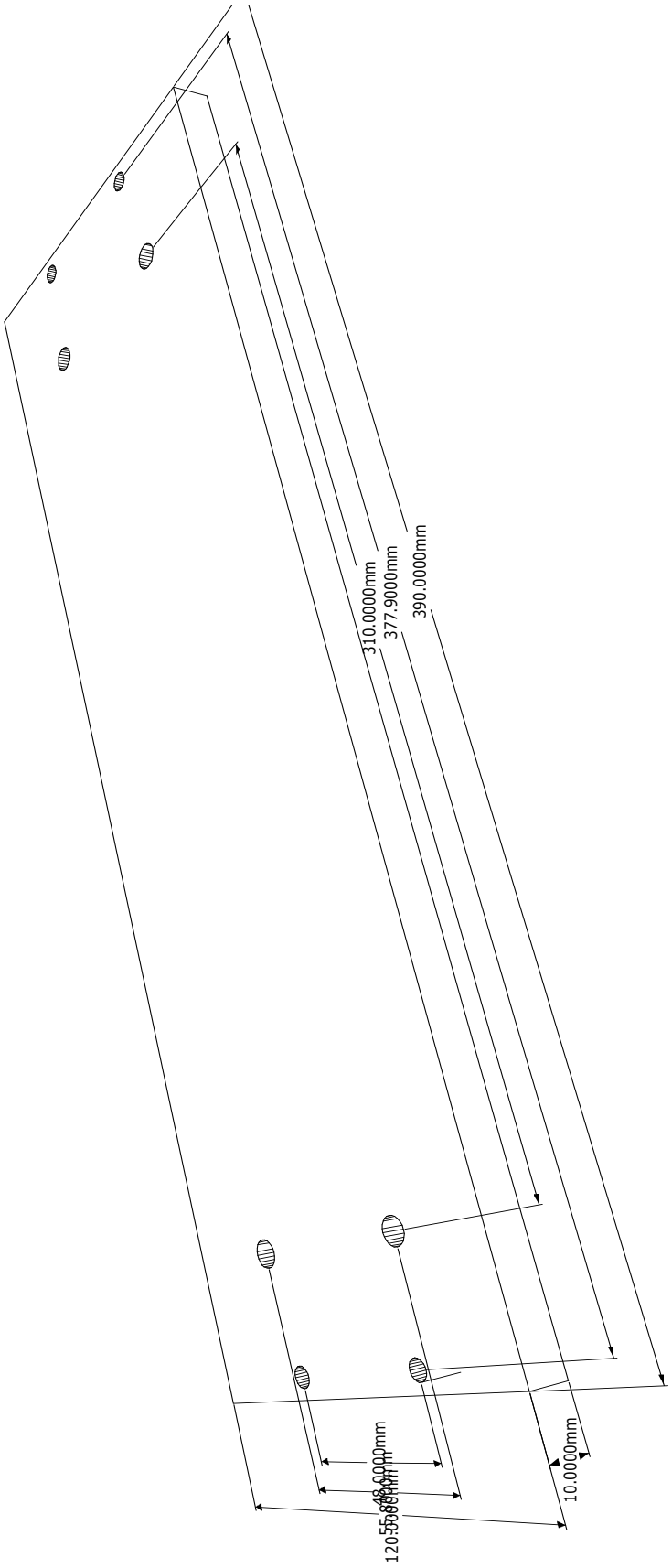


Abbildung 38: Konstruktionszeichnung der Grundplatte

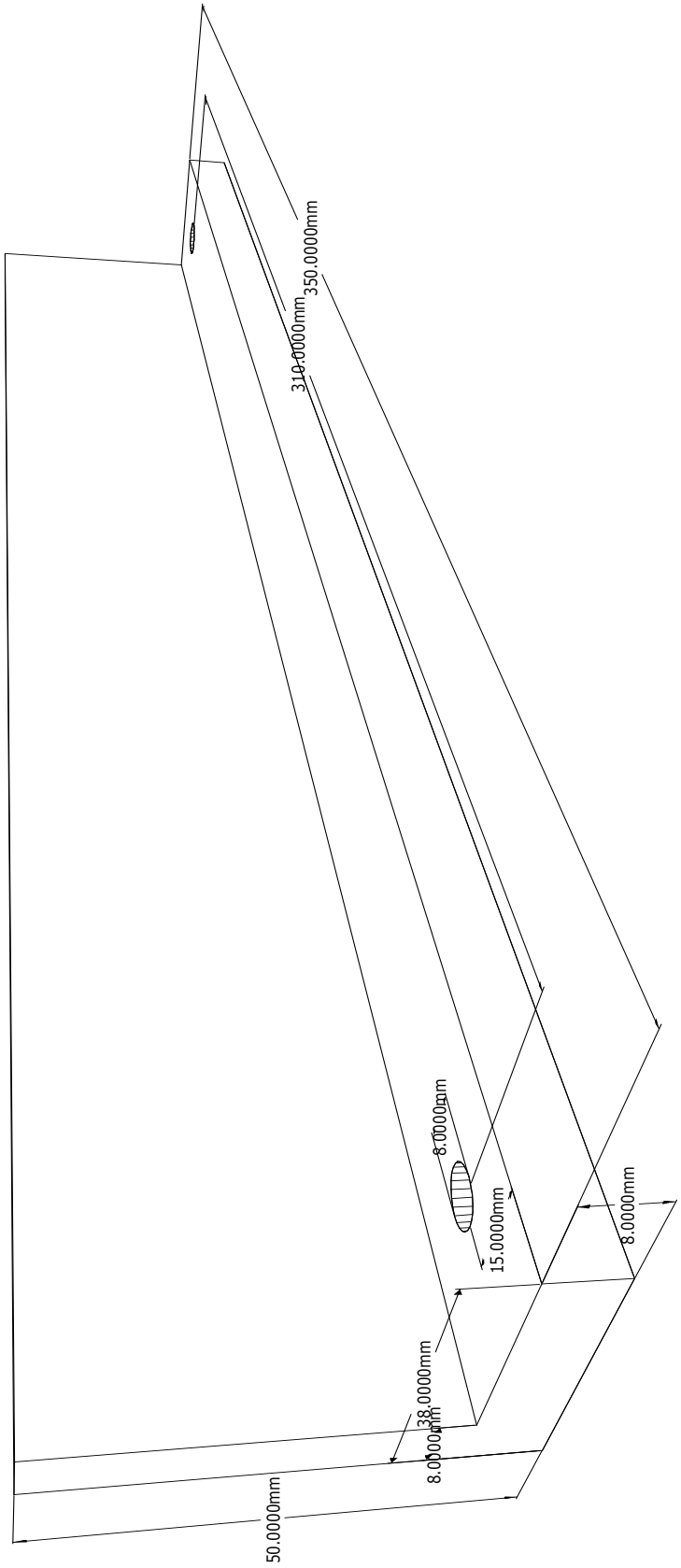


Abbildung 39: Konstruktionszeichnung der Seitenwinkel

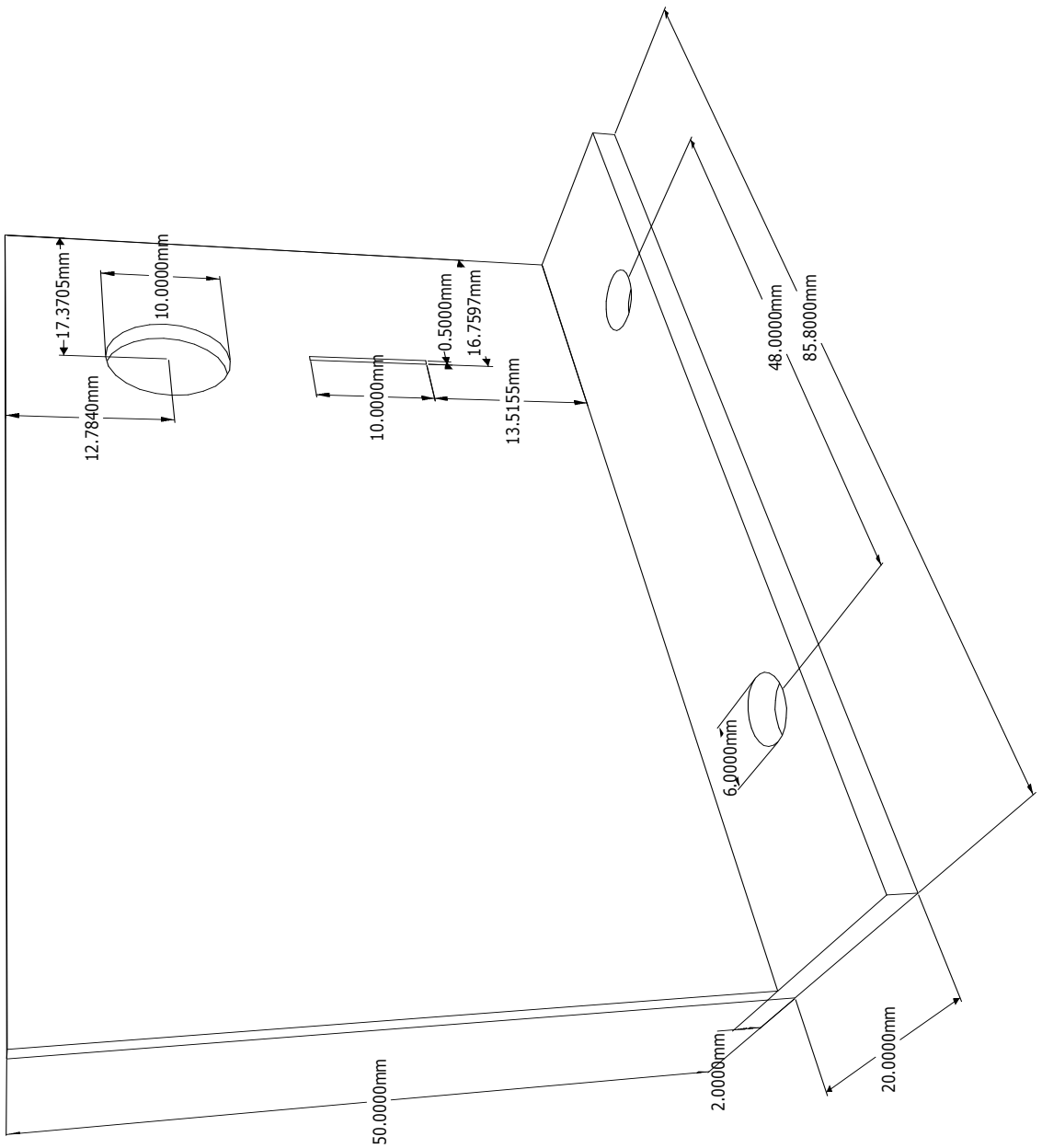


Abbildung 40: Konstruktionszeichnung der Endhalter

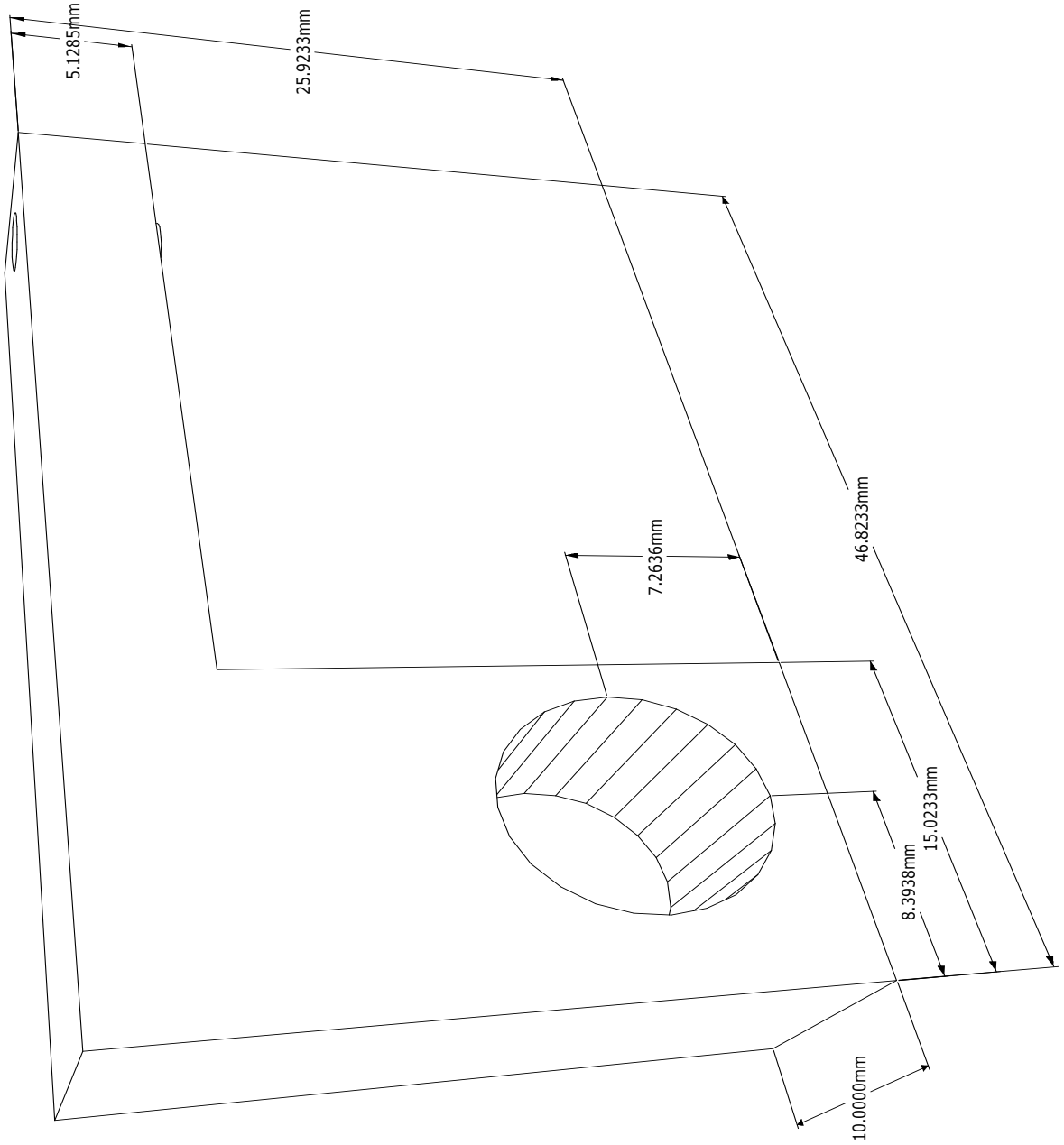


Abbildung 41: Konstruktionszeichnung der Printhalter

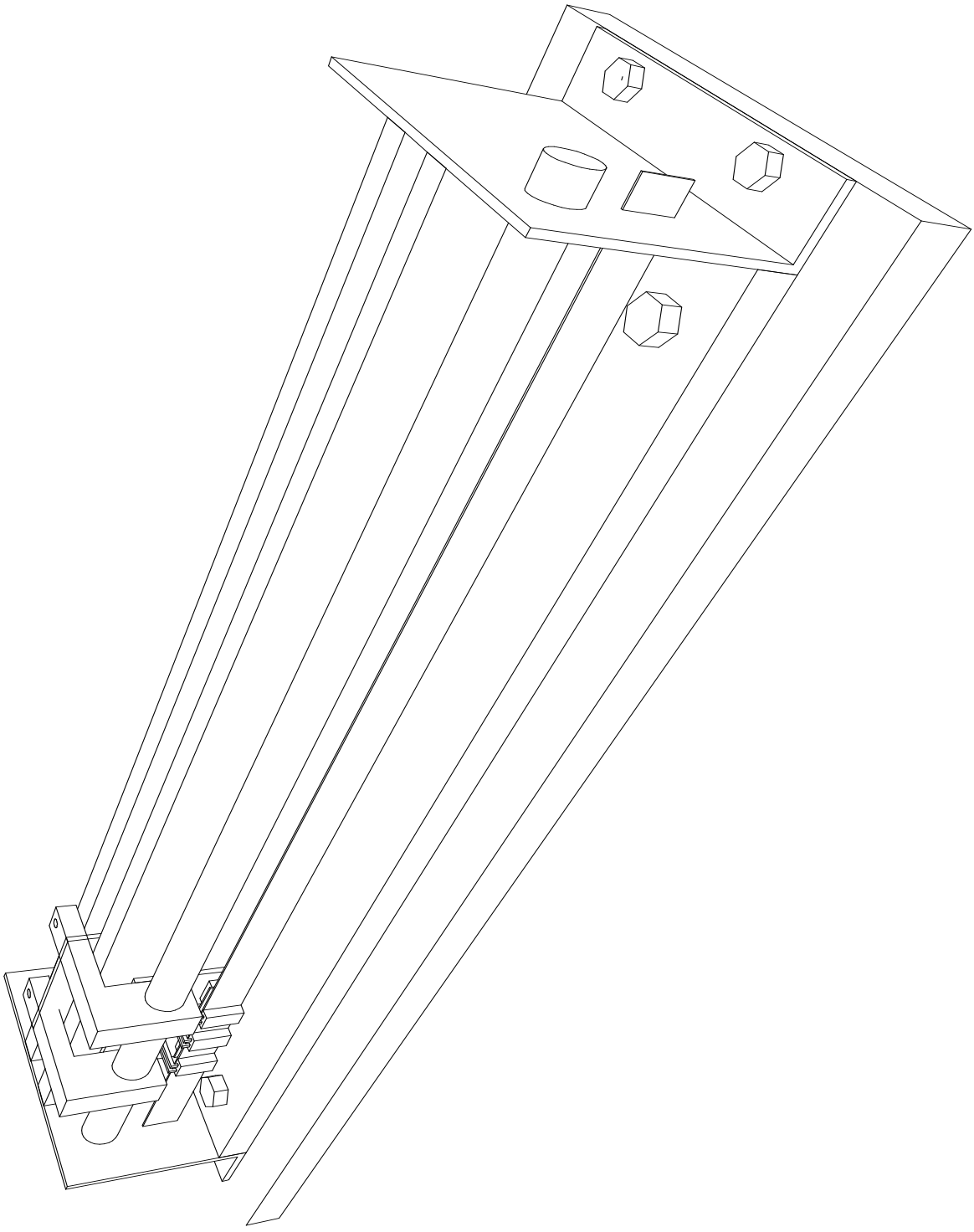


Abbildung 42: Konstruktionszeichnung des Zusammenbaus

D Software Code

Die gesamte modifizierte Software

Listing 7: Langer Codeteil

```
/* ============================================================================
System Name:  PMSM3_1   (QEP version)

File Name:       PMSM31.C

Description:  Peripheral independent object for the implementation
              of Sensored Field Orientation Control for a Three Phase
              Permanent-Magnet Synchronous Motor using measured rotor angle.


Originator:              Digital control systems Group - Texas Instruments


Target dependency:       x240/1/2/3/07
To Select the target device see target.h file.

Note that the PWM/sampling frequency in C system is running at 15 kHz, which
is different from the PWM/samping frequency in ASM system (i.e., 20 kHz) as
indicated in the PMSM3-1 system documentation. Also, this PWM/samping frequency
in both ASM and C systems are based on x2407 with 30 MHz clock. If the x243 with
20 MHz clock is used, then the PWM/samping frequency may be decreased accordingly.

============================================================================
 History:
----------------------------------------------------------------------------
 03-01-2001      Release Rev 1.0
============================================================================ */

/*-----------------------------------------------------------------------------
     Get the compilation target setting.
     This target is defined by TARGET.H. To change the target, or to find out
     the present target, see that file.
-----------------------------------------------------------------------------*/
#include <TARGET.H>
/*-----------------------------------------------------------------------------
     Include header information for this file.
-----------------------------------------------------------------------------*/
#include <pmsm31.h>
/*-----------------------------------------------------------------------------
     System settings
-----------------------------------------------------------------------------*/
#define WAIT_STATES      0x40
/*-----------------------------------------------------------------------------
     Global Declarations

     Instance the PWM Generator (Driver) Interface.
     Also initalize the PWMGEN object.
     This pre-initializer takes on the nature depending on the TARGET device.


     An IMPORTANT NOTE :
          This pre-initalization initializes the PWMGEN data structure in
          memory. This WILL NOT initialize the PWM Generator timers and
          so on. This is accomplished by calling the init method in the
          PWMGEN object. This applies to most drivers that supply an init
          method.
-----------------------------------------------------------------------------*/
          //PWMGEN pwm = PWMGEN_DEFAULTS;
          PWMGEN pwm = PWMGEN_SCRO;
/*-----------------------------------------------------------------------------
     Instance a speed measurement object. The defaults are set in
     SPEED_FR.H. Also  the init defaults to measure speed by frequency.
-----------------------------------------------------------------------------*/
          //SPEED_MEAS speed = SPEED_FR_MEAS_DEFAULTS;
          SPEED_MEAS speed = SPEED_FR_MEAS_SCRO;
/*-----------------------------------------------------------------------------
     Declaration for the QEP Interface Driver. The defaults are set in
     F2407QEP.H or F243_QEP.H
-----------------------------------------------------------------------------*/
          //QEP qep = QEP_DEFAULTS;
          QEP qep = QEP_SCRO;
/*-----------------------------------------------------------------------------
     Instance a single FOC_TI object.
-----------------------------------------------------------------------------*/
          FOC_TI foc = FOC_TI_INITVALS;

/*-----------------------------------------------------------------------------
     Instance the EVMDAC Interface.
-----------------------------------------------------------------------------*/
               EVMDAC dac = EVMDAC_DEFAULTS;

/*-----------------------------------------------------------------------------
     Instance the WATCHDOG Interface.
-----------------------------------------------------------------------------*/
          WATCHDOG wdog = WATCHDOG_DEFAULTS;
```

```
/*-------------------------------------------------------------------------
    Instance ILEG2MEAS object. The defaults are set in F2407ILG.H or F243ILG.H
-------------------------------------------------------------------------*/
        ILEG2MEAS ilg2 = ILEG2MEAS_DEFAULTS;

/*-------------------------------------------------------------------------
    Instance DRIVE object. The defaults are set in F07DRIVE.H or F243DRIVE.H
-------------------------------------------------------------------------*/
        DRIVE drive = DRIVE_DEFAULTS;

/*-------------------------------------------------------------------------
    Instance the Serial Communication
-------------------------------------------------------------------------*/
        SERVALS serial =        SER_DEFAULTS;
/*-------------------------------------------------------------------------
    Instance the ISR checking variable
-------------------------------------------------------------------------*/
        int     isr_ticker;

//user functions

//user defined functions
 #define VERSION 64
//the needed string-convert functions exported from lib
#include <stdlib.h>
#include <ctype.h>

int t2i(const char *st)
{
    register const char *fst    = st;
    register       int   result = 0;
    register       char  fc;

    while (_isspace(*fst++));                       // SKIP WHITE SPACE

    if ((fc = *--fst) == '-' || *fst == '+') ++fst;

    while (_isdigit(*fst))
    {
        result *= 10;
        result += *fst++ - '0';
    }

    return (fc == '-') ? -result : result;
}

#define BUFLEN 20
int itoa(int val, char *buffer)
{
    char           tempc[BUFLEN];
    register char *bufptr;
    register int   neg = val < 0;
    unsigned int   uval = neg ? -val : val;

    *(bufptr = &tempc[BUFLEN - 1]) = 0;

    do {*--bufptr = (uval % 10) + '0';}  while(uval /= 10);
    if (neg) *--bufptr = '-';

    memmove(buffer, bufptr, neg = (tempc + BUFLEN) - bufptr);
    return neg - 1;     // DON'T COUNT NULL TERMINATION
}


//serial communication
                //the serial commands
                #define POSITION 'l'
                #define VELOCITY 'v'
                #define STATUS   's'
                #define HOME     'h'
                #define SCOPTIME 't'
                #define ERROR    'e'
                #define ENABLE   'o'
                #define SCOPGET  'g'
                #define CUR_P    'a'
                #define CUR_I    'b'
                #define POS_P    'p'
                #define POS_I    'i'
                #define POS_D    'd'
                #define REVISION 'r'
                #define UNKONWN  'u'
                #define CNTUP    '+'
                #define CNTDWN   '-'
                #define CR              0x0d
                #define END             0x00

                char ret[7] = {CR,CR,CR,CR,CR};
                char rcv[7] = {CR,CR,CR,CR,CR};
                char cnt = 0;


void comm_welcome(void){
```

```
                //send welcome string
                ret[0] = 'P';
                ret[1] = 'r';
                ret[2] = 'i';
                ret[3] = 'M';
                ret[4] = 'o';
                ret[5] = 't';
                ret[6] = 0;
              strcpy(serial.tx_string,ret);
                serial.tx_busy = 1;
}

 void communicate(void)// Handle the Communication in background
 {
        //handle the TX of Data
                serial.txHandle(&serial);

                //handle the RX of Data
                if (serial.rx_finshed == 1)
                        {
                        while(serial.rx_busy == 1);             // warten bis Semaphore frei
                        serial.rx_busy = 1;                             // Semaphore öffnen
                        strcpy(rcv, serial.rx_string);  // string kopieren
                        serial.rx_finshed = 0;
                        serial.rx_busy = 0;                             // Semaphore frei geben
                        //parse the Serial communication
                                //command finished so parse
                                switch (rcv[0]){
                                        //read only ones
                                        case REVISION: ret[0] = REVISION; itoa(VERSION, &ret[1]); ret[3] = END; b
                                        case CNTUP: ret[0] = '='; itoa(cnt++, &ret[1]);  break;
                                        case CNTDWN: ret[0] = '='; itoa(cnt--, &ret[1]); break;
                                        case STATUS: ret[0] = STATUS; itoa(0, &ret[1]); break; //error 0 at first
                                        //write only ones
                                        case ENABLE:
                                                if(rcv[1] == '1')
                                                        drive.enable_flg=1;
                                                else
                                                        drive.enable_flg=0;
                                        //read&write
                                        case POSITION:
                                                if(rcv[1] == 0){
                                                        ret[0] = POSITION; itoa(speed.theta_elec, &ret[1]);}
                                                else{
                                                        speed.speed_rpm = t2i(&rcv[1]);
                                                        ret[0] = POSITION; itoa(speed.speed_rpm, &ret[1]);}      /
                                                break;
                                        case VELOCITY:
                                                if(rcv[1] == 0){
                                                        ret[0] = VELOCITY; itoa(foc.Id_ref, &ret[1]);}
                                                else{
                                                        foc.Id_ref = t2i(&rcv[1]);
                                                        ret[0] = VELOCITY; itoa(foc.Id_ref, &ret[1]);}
                                                break;
                                        case CUR_P:
                                                if(rcv[1] == 0){
                                                        ret[0] = CUR_P; itoa(foc.pid_iq.Kp_reg3, &ret[1]);}
                                                else{
                                                        foc.pid_iq.Kp_reg3 = t2i(&rcv[1]);
                                                        ret[0] = CUR_P; itoa(foc.pid_iq.Kp_reg3, &ret[1]);}
                                                break;
                                        case CUR_I:
                                                if(rcv[1] == 0){
                                                        ret[0] = CUR_I; itoa(foc.pid_spd.Ki_reg3, &ret[1]);}
                                                else{
                                                        foc.pid_spd.Ki_reg3 = t2i(&rcv[1]);
                                                        ret[0] = CUR_I; itoa(foc.pid_spd.Ki_reg3, &ret[1]);}
                                                break;
                                        default: ret[0] = UNKONWN; ret[1]=0;
                                }
                                //send out the answer
//TODO: telegrams might get lost, if comunication is too fast for tx ...
                                if(serial.tx_busy != 1)
                                {
                                        strcpy(serial.tx_string,ret);
                                        serial.tx_busy = 1;
                                }
                         }
}
//END serial communication

//end user functions


void main()
{
/*-------------------------------------------------------------------------
    Return system to a sane condition
-------------------------------------------------------------------------*/
        RstSystem();
/*-------------------------------------------------------------------------
    Initialize the Serial Communication
-------------------------------------------------------------------------*/
```

```
    serial.baudrate = 115200; //change baudrate
        serial.init(&serial);


/*-------------------------------------
Test serial communication
-------------------------------------* /
        IMR=0x1A;                   // Set up interrupt mask to enable INT2 until an explicit enable.
                IFR=0x10;
                PIACKR1 = 0x0100; //acc rx-int
                PIRQR1 = 0x0000;
                enable_ints();//INMT=0;
/*


/*-------------------------------------------------------------------------------*/


#if (BUILDLEVEL==LEVEL1)
/*-------------------------------------------------------------------------------
    Set the pwm period to 500 cycles,(F243) [ or 750 cycles, LF240x ].
    This assumes a CPU CLKIN of 5MHz for the F243 and CLKIN of 7.5 MHz
    for the LF240x.
    Defaults for pwm are set in F243_PWM.H
    Note that this will ONLY change runtime configurable parameters.
    For changing settings like timer modes or PWM Polarity, once the
    constants like PWM_INIT_STATE, or ACTR_INIT_STATE are changed, the
    driver file must be re-compiled.

    To do this change the constant and then run the batch file
    ..\..\drv011\build\f243drv.bat or f2407drv.bat

    This will re-build the driver, with the new timer mode or PWM polarity,
    and then run the (re)build for this project. This will update the
    setting.
-------------------------------------------------------------------------------*/
#if     TARGET==F243
        pwm.period_max=667;             /* This is based on 15kHz PWM frequency (20MHz) */

#elif   TARGET==F2407
        //pwm.period_max=1000;          /* This is based on 15kHz PWM frequency (30MHz) */
        //pwm.period_max=1333;          /* This is based on 15kHz PWM frequency (40MHz)*/
                pwm.period_max=1000;            /* This is based on 20kHz PWM frequency (40MHz)*/
#endif  /* TARGET */

/*-------------------------------------------------------------------------------
    Initialize the FOC_TI object. This is a call to the init method within
    the FOC_TI object foc.
-------------------------------------------------------------------------------*/
        FOC_TI_Init(&foc);

/*-------------------------------------------------------------------------------
    Initialize the real time monitor
-------------------------------------------------------------------------------*/
//We do not use the Monitor for now..       rtmon_init();           /* Call the monitor init function
*/
        enable_ints();          /* Set off the system running.              */

/*-------------------------------------------------------------------------------
    Initialize DRIVE --  Waiting for PWM enable flag setting
-------------------------------------------------------------------------------*/
        comm_welcome();
        while(drive.enable_flg==0)
        {
        drive.init(&drive);
                communicate();
        }

/*-------------------------------------------------------------------------------
    Initialize PWM Generator
-------------------------------------------------------------------------------*/
        pwm.init(&pwm);

/*-------------------------------------------------------------------------------
    Clear any EV(A)IFRA flags.
-------------------------------------------------------------------------------*/
#if     TARGET==F243
        EVIFRA=0xffff;
#elif   TARGET==F2407
            EVAIFRA=0xffff;
#endif  /* TARGET */

/*-------------------------------------------------------------------------------
    Set the dac pointers
-------------------------------------------------------------------------------*/
        dac.qptr0=&foc.svgen.va;
        dac.qptr1=&foc.svgen.vb;
        dac.qptr2=&foc.svgen.vc;
        dac.qptr3=&foc.rg.rmp_out;

#endif /* (BUILDLEVEL==LEVEL1) */
```

```
#if (BUILDLEVEL==LEVEL2)

/*-------------------------------------------------------------------------------
    Set the pwm period to 500 cycles,(F243) [ or 750 cycles, LF240x ].
    This assumes a CPU CLKIN of 5MHz for the F243 and CLKIN of 7.5 MHz
    for the LF240x.
    Defaults for pwm are set in F243_PWM.H
    Note that this will ONLY change runtime configurable parameters.
    For changing settings like timer modes or PWM Polarity, once the
    constants like PWM_INIT_STATE, or ACTR_INIT_STATE are changed, the
    driver file must be re-compiled.

    To do this change the constant and then run the batch file
    ..\..\drv011\build\f243drv.bat or f2407drv.bat

    This will re-build the driver, with the new timer mode or PWM polarity,
    and then run the (re)build for this project. This will update the
    setting.
-------------------------------------------------------------------------------*/
#if     TARGET==F243
        pwm.period_max=667;          /* This is based on 15kHz PWM frequency (20MHz) */

#elif   TARGET==F2407
/*        pwm.period_max=1000;     */    /* This is based on 15kHz PWM frequency (30MHz) */
        pwm.period_max=1000;        /* This is based on 15kHz PWM frequency (40MHz)*/
#endif  /* TARGET */

/*-------------------------------------------------------------------------------
    Initialize the FOC_TI object. This is a call to the init method within
    the FOC_TI object foc.
-------------------------------------------------------------------------------*/
        FOC_TI_Init(&foc);

/*-------------------------------------------------------------------------------
    Initialize the real time monitor
-------------------------------------------------------------------------------*/
        //DONT USE RTMON rtmon_init();            /* Call the monitor init function          */
        enable_ints();           /* Set off the system running.             */

/*-------------------------------------------------------------------------------
    Initialize DRIVE --  Waiting for PWM enable flag setting
-------------------------------------------------------------------------------*/
        comm_welcome();
        while(drive.enable_flg==0)
        {
        drive.init(&drive);
                communicate();
        }

/*-------------------------------------------------------------------------------
    Initialize PWM Generator
-------------------------------------------------------------------------------*/
        pwm.init(&pwm);

/*-------------------------------------------------------------------------------
    Initialize ILEG2MEAS
-------------------------------------------------------------------------------*/
        ilg2.init(&ilg2);

/*-------------------------------------------------------------------------------
    Set the dac pointers
-------------------------------------------------------------------------------*/
        dac.qptr0=&ilg2.out_a;
        dac.qptr1=&ilg2.out_b;
        dac.qptr2=&foc.rg.rmp_out;
        dac.qptr3=&foc.voltage_dq.q;

#endif /* (BUILDLEVEL==LEVEL2) */


#if (BUILDLEVEL==LEVEL3)

/*-------------------------------------------------------------------------------
    Set the pwm period to 500 cycles,(F243) [ or 750 cycles, LF240x ].
    This assumes a CPU CLKIN of 5MHz for the F243 and CLKIN of 7.5 MHz
    for the LF240x.
    Defaults for pwm are set in F243_PWM.H
    Note that this will ONLY change runtime configurable parameters.
    For changing settings like timer modes or PWM Polarity, once the
    constants like PWM_INIT_STATE, or ACTR_INIT_STATE are changed, the
    driver file must be re-compiled.

    To do this change the constant and then run the batch file
    ..\..\drv011\build\f243drv.bat or f2407drv.bat

    This will re-build the driver, with the new timer mode or PWM polarity,
    and then run the (re)build for this project. This will update the
    setting.
-------------------------------------------------------------------------------*/
#if     TARGET==F243
        pwm.period_max=667;          /* This is based on 15kHz PWM frequency (20MHz) */

#elif   TARGET==F2407
```

```
/*        pwm.period_max=1000;     */    /* This is based on 15kHz PWM frequency (30MHz) */
        pwm.period_max=1000;       /* This is based on 15kHz PWM frequency (40MHz)*/
#endif  /* TARGET */

/*-------------------------------------------------------------------------------
    Initialize the FOC_TI object. This is a call to the init method within
    the FOC_TI object foc.
-------------------------------------------------------------------------------*/
        FOC_TI_Init(&foc);

/*-------------------------------------------------------------------------------
    Initialize the real time monitor
-------------------------------------------------------------------------------*/
        //NO USE OF RT-MON rtmon_init();            /* Call the monitor init function          */
        enable_ints();           /* Set off the system running.             */

/*-------------------------------------------------------------------------------
    Initialize DRIVE  --  Waiting for PWM enable flag setting
-------------------------------------------------------------------------------*/
        comm_welcome();
        while(drive.enable_flg==0)
        {
        drive.init(&drive);
                communicate();
        }

/*-------------------------------------------------------------------------------
    Initialize PWM Generator
-------------------------------------------------------------------------------*/
        pwm.init(&pwm);

/*-------------------------------------------------------------------------------
    Initialize ILEG2MEAS
-------------------------------------------------------------------------------*/
        ilg2.init(&ilg2);

                //configure the Current scale to 0.1mA/count
                ilg2.gain_a = -5734;//0.7 Q13 -> Strom in +-1A
                ilg2.gain_b = -6389;//0.78 Q13 -> Strom in +-1A ... anders wegen anderem Filter?

/*-------------------------------------------------------------------------------
    Set the dac pointers
-------------------------------------------------------------------------------*/
        dac.qptr0=&ilg2.out_a;
        dac.qptr1=&foc.svgen.vb;
        dac.qptr2=&foc.svgen.vc;
        dac.qptr3=&foc.rg.rmp_out;

/*-------------------------------------------------------------------------------
    Set the current controllers -> set this to the defaults
-------------------------------------------------------------------------------*/
foc.pid_id.Kp_reg3 = 16312;
foc.pid_iq.Kp_reg3 = 16312;
foc.pid_id.Ki_reg3 = 32399;
foc.pid_iq.Ki_reg3 = 32399;
foc.pid_id.Kc_reg3 = 32399;
foc.pid_iq.Kc_reg3 = 32399;
foc.pid_id.Kd_reg3 = 0;
foc.pid_iq.Kd_reg3 = 0;
foc.Id_ref = 0;
foc.Iq_ref = 16384;
foc.speed_ref = 990;//10000 = max speed bei 24mm polabstand = 12ms pro periode

#endif /* (BUILDLEVEL==LEVEL3) */


#if (BUILDLEVEL==LEVEL4)

/*-------------------------------------------------------------------------------
    Set the pwm period to 500 cycles,(F243) [ or 750 cycles, LF240x ].
    This assumes a CPU CLKIN of 5MHz for the F243 and CLKIN of 7.5 MHz
    for the LF240x.
    Defaults for pwm are set in F243_PWM.H
    Note that this will ONLY change runtime configurable parameters.
    For changing settings like timer modes or PWM Polarity, once the
    constants like PWM_INIT_STATE, or ACTR_INIT_STATE are changed, the
    driver file must be re-compiled.

    To do this change the constant and then run the batch file
    ..\..\drv011\build\f243drv.bat or f2407drv.bat

    This will re-build the driver, with the new timer mode or PWM polarity,
    and then run the (re)build for this project. This will update the
    setting.
-------------------------------------------------------------------------------*/
#if     TARGET==F243
        pwm.period_max=667;           /* This is based on 15kHz PWM frequency (20MHz) */

#elif   TARGET==F2407
/*        pwm.period_max=1000;     */    /* This is based on 15kHz PWM frequency (30MHz) */
        pwm.period_max=1000;       /* This is based on 15kHz PWM frequency (40MHz)*/
#endif  /* TARGET */

/*-------------------------------------------------------------------------------
```

```
    Initialize the FOC_TI object. This is a call to the init method within
    the FOC_TI object foc.
------------------------------------------------------------------------------*/
        FOC_TI_Init(&foc);

/*------------------------------------------------------------------------------
    Initialize the real time monitor
------------------------------------------------------------------------------*/
        //no rt mon used rtmon_init();            /* Call the monitor init function           */
        enable_ints();          /* Set off the system running.              */

/*------------------------------------------------------------------------------
    Initialize DRIVE --  Waiting for PWM enable flag setting
------------------------------------------------------------------------------*/
        comm_welcome();
        while(drive.enable_flg==0)
        {
        drive.init(&drive);
                communicate();
        }

/*------------------------------------------------------------------------------
    Initialize PWM Generator
------------------------------------------------------------------------------*/
        pwm.init(&pwm);

/*------------------------------------------------------------------------------
    Initialize the Quadrature Encoder Interface Driver.
------------------------------------------------------------------------------*/
        qep.init(&qep);

/*------------------------------------------------------------------------------
    Initialize parameters of the speed calculation based on angle
------------------------------------------------------------------------------*/
        speed.K1_fr = 4800;
        speed.K2_fr = 32361;
        speed.K3_fr = 407;
        speed.rpm_max = 6000;

/*------------------------------------------------------------------------------
    Initialize ILEG2MEAS
------------------------------------------------------------------------------*/
        ilg2.init(&ilg2);

                //configure the Current scale to 0.1mA/count
                ilg2.gain_a = -2000;
                ilg2.gain_b = -2000;

/*------------------------------------------------------------------------------
    Set the dac pointers
------------------------------------------------------------------------------*/
        dac.qptr0=&foc.rg.rmp_out;
        dac.qptr1=&foc.svgen.va;
        dac.qptr2=&qep.theta_elec;
        dac.qptr3=&foc.svgen.vb;

#endif /* (BUILDLEVEL==LEVEL4) */


#if (BUILDLEVEL==LEVEL5)

/*------------------------------------------------------------------------------
    Set the pwm period to 500 cycles,(F243) [ or 750 cycles, LF240x ].
    This assumes a CPU CLKIN of 5MHz for the F243 and CLKIN of 7.5 MHz
    for the LF240x.
    Defaults for pwm are set in F243_PWM.H
    Note that this will ONLY change runtime configurable parameters.
    For changing settings like timer modes or PWM Polarity, once the
    constants like PWM_INIT_STATE, or ACTR_INIT_STATE are changed, the
    driver file must be re-compiled.

    To do this change the constant and then run the batch file
    ..\..\drv011\build\f243drv.bat or f2407drv.bat

    This will re-build the driver, with the new timer mode or PWM polarity,
    and then run the (re)build for this project. This will update the
    setting.
------------------------------------------------------------------------------*/
#if     TARGET==F243
        pwm.period_max=667;           /* This is based on 15kHz PWM frequency (20MHz) */

#elif   TARGET==F2407
/*        pwm.period_max=1000;     */    /* This is based on 15kHz PWM frequency (30MHz) */
        pwm.period_max=1000;        /* This is based on 15kHz PWM frequency (40MHz)*/
#endif  /* TARGET */

/*------------------------------------------------------------------------------
    Initialize the FOC_TI object. This is a call to the init method within
    the FOC_TI object foc.
------------------------------------------------------------------------------*/
        FOC_TI_Init(&foc);

/*------------------------------------------------------------------------------
    Initialize the real time monitor
```

```
------------------------------------------------------------------------------*/
        //rtmon_init();             /* Call the monitor init function          */
        enable_ints();            /* Set off the system running.             */

/*-----------------------------------------------------------------------------
    Initialize DRIVE --  Waiting for PWM enable flag setting
-----------------------------------------------------------------------------*/
        comm_welcome();
        while(drive.enable_flg==0)
        {
        drive.init(&drive);
                communicate();
        }

/*-----------------------------------------------------------------------------
    Initialize PWM Generator
-----------------------------------------------------------------------------*/
        pwm.init(&pwm);

/*-----------------------------------------------------------------------------
    Initialize the Quadrature Encoder Interface Driver.
-----------------------------------------------------------------------------*/
        qep.init(&qep);

/*-----------------------------------------------------------------------------
    Initialize parameters of the speed calculation based on angle
-----------------------------------------------------------------------------*/
        speed.K1_fr = 4800;
        speed.K2_fr = 32361;
        speed.K3_fr = 407;
        speed.rpm_max = 6000;

/*-----------------------------------------------------------------------------
    Initialize ILEG2MEAS
-----------------------------------------------------------------------------*/
        ilg2.init(&ilg2);

                //configure the Current scale to 0.1mA/count
                ilg2.gain_a = -2000;
                ilg2.gain_b = -2000;

/*-----------------------------------------------------------------------------
    Set the dac pointers
-----------------------------------------------------------------------------*/
        dac.qptr0=&ilg2.out_a;
        dac.qptr1=&foc.svgen.va;
        dac.qptr2=&qep.theta_elec;
        dac.qptr3=&foc.svgen.vb;

#endif /* (BUILDLEVEL==LEVEL5) */


#if     TARGET==F243
        EVIFRA=0x0ffff;           /* Clear all Group A EV interrupt flags    */

#elif   TARGET==F2407
        EVAIFRA=0x0ffff;          /* Clear all EV1 Group A EV interrupt flags*/
#endif


/*-----------------------------------------------------------------------------*/
        while(1)                /* Nothing running in the background at present */
                {

                communicate(); //over serial

        }
/*-----------------------------------------------------------------------------*/

} /* End: main() */


void interrupt c_int02()
{
     asm("      CLRC     XF ");

        isr_ticker++;

#if (BUILDLEVEL==LEVEL1)
/*-----------------------------------------------------------------------------*/
#if     TARGET==F243
        EVIFRA=0x0ffff;           /* Clear all Group A EV interrupt flags    */

#elif   TARGET==F2407
        EVAIFRA=0x0ffff;          /* Clear all EV1 Group A EV interrupt flags*/
#endif

/*-----------------------------------------------------------------------------
    Call the enable/disable PWM signal drive function.
-----------------------------------------------------------------------------*/
        drive.calc(&drive);

/*-----------------------------------------------------------------------------
    Call the FOC_TI_Run function to perform the runtime tasks of the
```

```
    FOC_TI algorithm.
-------------------------------------------------------------------------------*/
        FOC_TI_Run(&foc);

/*-------------------------------------------------------------------------------
    Connect output of the FOC.SVGEN to the PWM Driver and call the PWM driver
    update function.
-------------------------------------------------------------------------------*/
        pwm.mfunc_c1 = foc.svgen.va;
        pwm.mfunc_c2 = foc.svgen.vb;
        pwm.mfunc_c3 = foc.svgen.vc;
        pwm.update(&pwm);

/*-------------------------------------------------------------------------------
    Call the diagnostic DAC utility update function.
-------------------------------------------------------------------------------*/
        dac.update(&dac);

/*-------------------------------------------------------------------------------*/
#endif /* BUILDLEVEL==LEVEL1 */


#if (BUILDLEVEL==LEVEL2)
/*-------------------------------------------------------------------------------*/
#if     TARGET==F243
        EVIFRA=0x0ffff;             /* Clear all Group A EV interrupt flags     */

#elif   TARGET==F2407
        EVAIFRA=0x0ffff;            /* Clear all EV1 Group A EV interrupt flags*/
#endif

/*-------------------------------------------------------------------------------
    Call the enable/disable PWM drive function.
-------------------------------------------------------------------------------*/
        drive.calc(&drive);

/*-------------------------------------------------------------------------------
    Call the FOC_TI_Run function to perform the runtime tasks of the
    FOC_TI algorithm.
-------------------------------------------------------------------------------*/
        FOC_TI_Run(&foc);

/*-------------------------------------------------------------------------------
    Connect output of the FOC.SVGEN to the PWM Driver and call the PWM driver
    update function.
-------------------------------------------------------------------------------*/
        pwm.mfunc_c1 = foc.svgen.va;
        pwm.mfunc_c2 = foc.svgen.vb;
        pwm.mfunc_c3 = foc.svgen.vc;
        pwm.update(&pwm);

/*-------------------------------------------------------------------------------
    Call the ileg2_drv function to perform the ADC tasks for 2 currents measurement.
-------------------------------------------------------------------------------*/
        ilg2.read(&ilg2);

/*-------------------------------------------------------------------------------
    Connect outputs of the ILEG2MEAS to FOC
-------------------------------------------------------------------------------*/
                foc.current_abc.a = ilg2.out_a;
                foc.current_abc.b = ilg2.out_b;
                foc.current_abc.c = -(ilg2.out_a+ilg2.out_b);

/*-------------------------------------------------------------------------------
    Call the diagnostic DAC utility update function.
-------------------------------------------------------------------------------*/
        dac.update(&dac);

/*-------------------------------------------------------------------------------*/
#endif /* BUILDLEVEL==LEVEL2 */


#if (BUILDLEVEL==LEVEL3)
/*-------------------------------------------------------------------------------*/
#if     TARGET==F243
        EVIFRA=0x0ffff;             /* Clear all Group A EV interrupt flags     */

#elif   TARGET==F2407
        EVAIFRA=0x0ffff;            /* Clear all EV1 Group A EV interrupt flags*/
#endif

/*-------------------------------------------------------------------------------
    Call the enable/disable PWM signal drive function.
-------------------------------------------------------------------------------*/
        drive.calc(&drive);

/*-------------------------------------------------------------------------------
    Call the FOC_TI_Run function to perform the runtime tasks of the
    FOC_TI algorithm.
-------------------------------------------------------------------------------*/
        FOC_TI_Run(&foc);

/*-------------------------------------------------------------------------------
    Connect output of the FOC.SVGEN to the PWM Driver and call the PWM driver
```

```
    update function.
-----------------------------------------------------------------------------*/
        pwm.mfunc_c1 = foc.svgen.va;
        pwm.mfunc_c2 = foc.svgen.vb;
        pwm.mfunc_c3 = foc.svgen.vc;
        pwm.update(&pwm);

/*-----------------------------------------------------------------------------
    Call the ileg2_drv function to perform the ADC tasks for 2 currents measurement.
-----------------------------------------------------------------------------*/
        ilg2.read(&ilg2);

/*-----------------------------------------------------------------------------
    Connect outputs of the ILEG2MEAS to FOC
-----------------------------------------------------------------------------*/
                foc.current_abc.a = ilg2.out_a;
                foc.current_abc.b = ilg2.out_b;
                foc.current_abc.c = -(ilg2.out_a+ilg2.out_b);

/*-----------------------------------------------------------------------------
    Call the diagnostic DAC utility update function.
-----------------------------------------------------------------------------*/
        dac.update(&dac);


/*-----------------------------------------------------------------------------*/
#endif /* BUILDLEVEL==LEVEL3 */


#if (BUILDLEVEL==LEVEL4)
/*-----------------------------------------------------------------------------*/
#if     TARGET==F243
        EVIFRA=0x0ffff;             /* Clear all Group A EV interrupt flags    */

#elif   TARGET==F2407
        EVAIFRA=0x0ffff;            /* Clear all EV1 Group A EV interrupt flags*/
#endif

/*-----------------------------------------------------------------------------
    Call the enable/disable PWM signal drive function.
-----------------------------------------------------------------------------*/
        drive.calc(&drive);

/*-----------------------------------------------------------------------------
    Call the FOC_TI_Run function to perform the runtime tasks of the
    FOC_TI algorithm.
-----------------------------------------------------------------------------*/
        FOC_TI_Run(&foc);

/*-----------------------------------------------------------------------------
    Connect output of the FOC.SVGEN to the PWM Driver and call the PWM driver
    update function.
-----------------------------------------------------------------------------*/
        pwm.mfunc_c1 = foc.svgen.va;
        pwm.mfunc_c2 = foc.svgen.vb;
        pwm.mfunc_c3 = foc.svgen.vc;
        pwm.update(&pwm);

/*-----------------------------------------------------------------------------
    Call the ileg2_drv function to perform the ADC tasks for 2 currents measurement.
-----------------------------------------------------------------------------*/
        ilg2.read(&ilg2);


/*-----------------------------------------------------------------------------
    Connect outputs of the ILEG2MEAS to SFOC
-----------------------------------------------------------------------------*/
                foc.current_abc.a = ilg2.out_a;
                foc.current_abc.b = ilg2.out_b;
                foc.current_abc.c = -(ilg2.out_a+ilg2.out_b);

/*-----------------------------------------------------------------------------
        First collect the position information. The driver function updates the
        qep data structure with the information.
-----------------------------------------------------------------------------*/
        qep.calc(&qep);

/*-----------------------------------------------------------------------------
        Connect outputs of the QEP to SPEED
-----------------------------------------------------------------------------*/
        speed.theta_elec = qep.theta_elec;
        speed.calc(&speed);
        foc.Mea_spd = speed.speed_frq;

/*-----------------------------------------------------------------------------
    Call the diagnostic DAC utility update function.
-----------------------------------------------------------------------------*/
        dac.update(&dac);

/*-----------------------------------------------------------------------------*/
#endif /* BUILDLEVEL==LEVEL4 */
```

```
#if (BUILDLEVEL==LEVEL5)
/*------------------------------------------------------------------------------*/
#if     TARGET==F243
        EVIFRA=0x0ffff;             /* Clear all Group A EV interrupt flags     */

#elif   TARGET==F2407
        EVAIFRA=0x0ffff;            /* Clear all EV1 Group A EV interrupt flags*/
#endif

/*------------------------------------------------------------------------------
    Call the enable/disable PWM signal drive function.
------------------------------------------------------------------------------*/
        drive.calc(&drive);

/*------------------------------------------------------------------------------
    Call the FOC_TI_Run function to perform the runtime tasks of the
    FOC_TI algorithm.
------------------------------------------------------------------------------*/
        FOC_TI_Run(&foc);

/*------------------------------------------------------------------------------
    Connect output of the FOC.SVGEN to the PWM Driver and call the PWM driver
    update function.
------------------------------------------------------------------------------*/
        pwm.mfunc_c1 = foc.svgen.va;
        pwm.mfunc_c2 = foc.svgen.vb;
        pwm.mfunc_c3 = foc.svgen.vc;
        pwm.update(&pwm);

/*------------------------------------------------------------------------------
    Call the ileg2_drv function to perform the ADC tasks for 2 currents measurement.
------------------------------------------------------------------------------*/
        ilg2.read(&ilg2);

/*------------------------------------------------------------------------------
    Connect outputs of the ILEG2MEAS to FOC
------------------------------------------------------------------------------*/
                foc.current_abc.a = ilg2.out_a;
                foc.current_abc.b = ilg2.out_b;
                foc.current_abc.c = -(ilg2.out_a+ilg2.out_b);

/*------------------------------------------------------------------------------
        First collect the position information. The driver function updates the
        qep data structure with the information.
------------------------------------------------------------------------------*/
        qep.calc(&qep);
        foc.shaft_theta_elec = qep.theta_elec;
        foc.shaft_direction = qep.dir_QEP;

/*------------------------------------------------------------------------------
        Connect outputs of the QEP to SPEED
------------------------------------------------------------------------------*/
        speed.theta_elec = qep.theta_elec;
        speed.calc(&speed);
        foc.Mea_spd = speed.speed_frq;

/*------------------------------------------------------------------------------
    Call the diagnostic DAC utility update function.
------------------------------------------------------------------------------*/
        dac.update(&dac);

/*------------------------------------------------------------------------------*/
#endif /* BUILDLEVEL==LEVEL5 */


/* Change reference speed from pu value to rpm value (Q15 -> Q0)    */
/* Here the base speed is 6000 rpm  */
        foc.speed_ref_rpm = mul_q(foc.speed_ref,15,6000,0,0);


        asm("    SETC    XF ");

}    /* c_int02() */


/*------------------------------------------------------------------------------
        The CAP3 Interrupt invokes the Int 4 core Interrupt. This is vectored
        to the c_int04 function below.
        Since in our system the only source of interrupts at level 4 is the
        CAP3 interrupt, the PIVR value is not read or interpreted. In case there
        are multiple events happening on this level, that would be needed.
------------------------------------------------------------------------------*/
void interrupt c_int04()
{

/*------------------------------------------------------------------------------
        Since the CAP3 pin is connected to the Encoder Index Signal, execution
        of this code implies that a Index event has occured. So the index
        event handler is invoked.
------------------------------------------------------------------------------*/
        qep.indexevent(&qep);

}    /* c_int04() */
```

```
interrupt void ser_isr(void)
{
        //run the serial Interrupt
        serial.isr(&serial);
}


void RstSystem(void)
{
/*-------------------------------------------------------------------------
    First execute the initialization for the Wait Stage Genrator,
    Global interrupt disable, Shut off the Watchdog,
    and set up the Interupt Mask Register
-------------------------------------------------------------------------*/
        WSGR=WAIT_STATES;          /* Initialize Wait State Generator          */
        disable_ints();            /* Make sure the interrupts are disabled    */
        wdog.disable();            /* Vccp/Wddis pin/bit must be high           */
        IMR=0x1A;                  /* Set up interrupt mask to enable INT2 and serial
                                      until an explicit enable.                */

/*-------------------------------------------------------------------------
    Next we do the code for setting up the SCSR register, which is dependent
    on the exact device the code is being compiled for (F243 / F2407).
-------------------------------------------------------------------------*/
#if (TARGET==F243)
        SCSR=0x40c0;               /* Init SCSR */
        EVIMRA=0x0201;             /* Enable the timer 1 underflow/ CAP3 (index QEP) interrupts  */
#endif


#if (TARGET==F2407)
        //SCSR1=0xc;                 /* Init SCSR */
                SCSR1=0x004d;              /* Init SCSR */

        EVAIMRA=0x0201;            /* Enable the timer 1 underflow/ CAP3 (index QEP) interrupts  */
#endif

}   /* RstSystem(void) */

void interrupt phantom(void)
{

        static int phantom_count;
        phantom_count ++;

/*  Empty function: Used to handle any unwanted interrupts or events.
    All unused interrupt vectors are pointed to this function so that
    if any un-handled interrupts do get enabled, they are handled in a
    benign manner, preventing un-intended branches, calls or execution
    into garbage.
    Note that this function is an ISR, not a ordinary function.
*/
} /* phantom() */


/*-------------------------------------------------------------------------
    This function just provides a c-interface to the asm RTMON init function.
-------------------------------------------------------------------------*/
void rtmon_init(void)
{
    asm("       CALL    MON_RT_CNFG ");
}  /* rtmon_init() */
```